Bootcamp 4 Lecture Part 2

pandas dataframes (II)

import pandas

Because pandas is one of the most commonly used Python packages, it often gets imported as a shortened version of it's actual name. This makes it quicker to type.

```
import pandas as pd

import numpy as np
```

Using real data

About the practice data

We will be working with a dataset from forest fires in NE Portugal. I have included the dataset as a csv file in today's materials, but the data is available publically at this site: https://archive.ics.uci.edu/ml/datasets/Forest+Fires

The notebook also uses different datasets at the end of the file to practice loading different file formats.

Please place all files in the SAME FOLDER as this Jupyter file.

loading a csv file

We will use the function `pd.read_csv()`. This will automatically create a DataFrame object, which we are saving as `df`. `df` is a common variable name for a DataFrame. You can open the file, define it as a Pandas DataFrame, assign it to a variable, and close the file in one line.

```
df = pd.read_csv("forestfires.csv")

?pd.read_csv

Signature:
pd.read_csv(
    filepath_or_buffer: 'FilePath | ReadCsvBuffer[bytes] |
ReadCsvBuffer[str]',
    *,
    sep: 'str | None | lib.NoDefault' = <no_default>,
    delimiter: 'str | None | lib.NoDefault' = None,
```

```
    header: "int | Sequence[int] | None | Literal['infer']" = 'infer',
    names: 'Sequence[Hashable] | None | lib.NoDefault' = <no_default>,
    index_col: 'IndexLabel | Literal[False] | None' = None,
    usecols: 'UsecolsArgType' = None,
    dtype: 'DtypeArg | None' = None,
    engine: 'CSVEngine | None' = None,
    converters: 'Mapping[Hashable, Callable] | None' = None,
    true_values: 'list | None' = None,
    false_values: 'list | None' = None,
    skipinitialspace: 'bool' = False,
    skiprows: 'list[int] | int | Callable[[Hashable], bool] | None' =
None,
    skipfooter: 'int' = 0,
    nrows: 'int | None' = None,
    na_values: 'Hashable | Iterable[Hashable] | Mapping[Hashable,
Iterable[Hashable]] | None' = None,
    keep_default_na: 'bool' = True,
    na_filter: 'bool' = True,
    verbose: 'bool | lib.NoDefault' = <no_default>,
    skip_blank_lines: 'bool' = True,
    parse_dates: 'bool | Sequence[Hashable] | None' = None,
    infer_datetime_format: 'bool | lib.NoDefault' = <no_default>,
    keep_date_col: 'bool | lib.NoDefault' = <no_default>,
    date_parser: 'Callable | lib.NoDefault' = <no_default>,
    date_format: 'str | dict[Hashable, str] | None' = None,
    dayfirst: 'bool' = False,
    cache_dates: 'bool' = True,
    iterator: 'bool' = False,
    chunksize: 'int | None' = None,
    compression: 'CompressionOptions' = 'infer',
    thousands: 'str | None' = None,
    decimal: 'str' = '.',
    lineterminator: 'str | None' = None,
    quotechar: 'str' = '"',
    quoting: 'int' = 0,
    doublequote: 'bool' = True,
    escapechar: 'str | None' = None,
    comment: 'str | None' = None,
    encoding: 'str | None' = None,
    encoding_errors: 'str | None' = 'strict',
    dialect: 'str | csv.Dialect | None' = None,
    on_bad_lines: 'str' = 'error',
    delim_whitespace: 'bool | lib.NoDefault' = <no_default>,
    low_memory: 'bool' = True,
    memory_map: 'bool' = False,
    float_precision: "Literal['high', 'legacy'] | None" = None,
    storage_options: 'StorageOptions | None' = None,
    dtype_backend: 'DtypeBackend | lib.NoDefault' = <no_default>,
) -> 'DataFrame | TextFileReader'
```

```
Docstring:
Read a comma-separated values (csv) file into DataFrame.

Also supports optionally iterating or breaking of the file
into chunks.

Additional help can be found in the online docs for
`IO Tools
<https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html>`_.

Parameters
----------
filepath_or_buffer : str, path object or file-like object
    Any valid string path is acceptable. The string could be a URL.
Valid
    URL schemes include http, ftp, s3, gs, and file. For file URLs, a
host is
    expected. A local file could be:
file://localhost/path/to/table.csv.

    If you want to pass in a path object, pandas accepts any
``os.PathLike``.

    By file-like object, we refer to objects with a ``read()`` method,
such as
    a file handle (e.g. via builtin ``open`` function) or
``StringIO``.
sep : str, default ','
    Character or regex pattern to treat as the delimiter. If
``sep=None``, the
    C engine cannot automatically detect
    the separator, but the Python parsing engine can, meaning the
latter will
    be used and automatically detect the separator from only the first
valid
    row of the file by Python's builtin sniffer tool, ``csv.Sniffer``.
    In addition, separators longer than 1 character and different from
    ``'\s+'`` will be interpreted as regular expressions and will also
force
    the use of the Python parsing engine. Note that regex delimiters
are prone
    to ignoring quoted data. Regex example: ``'\r\t'``.
delimiter : str, optional
    Alias for ``sep``.
header : int, Sequence of int, 'infer' or None, default 'infer'
    Row number(s) containing column labels and marking the start of
the
    data (zero-indexed). Default behavior is to infer the column
names: if no ``names``
    are passed the behavior is identical to ``header=0`` and column
```

```
    names are inferred from the first line of the file, if column
    names are passed explicitly to ``names`` then the behavior is
identical to
    ``header=None``. Explicitly pass ``header=0`` to be able to
    replace existing names. The header can be a list of integers that
    specify row locations for a :class:`~pandas.MultiIndex` on the
columns
    e.g. ``[0, 1, 3]``. Intervening rows that are not specified will
be
    skipped (e.g. 2 in this example is skipped). Note that this
    parameter ignores commented lines and empty lines if
    ``skip_blank_lines=True``, so ``header=0`` denotes the first line
of
    data rather than the first line of the file.
names : Sequence of Hashable, optional
    Sequence of column labels to apply. If the file contains a header
row,
    then you should explicitly pass ``header=0`` to override the
column names.
    Duplicates in this list are not allowed.
index_col : Hashable, Sequence of Hashable or False, optional
  Column(s) to use as row label(s), denoted either by column labels or
column
  indices.  If a sequence of labels or indices is
given, :class:`~pandas.MultiIndex`
  will be formed for the row labels.

  Note: ``index_col=False`` can be used to force pandas to *not* use
the first
  column as the index, e.g., when you have a malformed file with
delimiters at
  the end of each line.
usecols : Sequence of Hashable or Callable, optional
    Subset of columns to select, denoted either by column labels or
column indices.
    If list-like, all elements must either
    be positional (i.e. integer indices into the document columns) or
strings
    that correspond to column names provided either by the user in
``names`` or
    inferred from the document header row(s). If ``names`` are given,
the document
    header row(s) are not taken into account. For example, a valid
list-like
    ``usecols`` parameter would be ``[0, 1, 2]`` or ``['foo', 'bar',
'baz']``.
    Element order is ignored, so ``usecols=[0, 1]`` is the same as
``[1, 0]``.
    To instantiate a :class:`~pandas.DataFrame` from ``data`` with
```

```
element order
    preserved use ``pd.read_csv(data, usecols=['foo', 'bar'])[['foo',
'bar']]``
    for columns in ``['foo', 'bar']`` order or
    ``pd.read_csv(data, usecols=['foo', 'bar'])[['bar', 'foo']]``
    for ``['bar', 'foo']`` order.

    If callable, the callable function will be evaluated against the
column
    names, returning names where the callable function evaluates to
``True``. An
    example of a valid callable argument would be ``lambda x:
x.upper() in
    ['AAA', 'BBB', 'DDD']``. Using this parameter results in much
faster
    parsing time and lower memory usage.
dtype : dtype or dict of {Hashable : dtype}, optional
    Data type(s) to apply to either the whole dataset or individual
columns.
    E.g., ``{'a': np.float64, 'b': np.int32, 'c': 'Int64'}``
    Use ``str`` or ``object`` together with suitable ``na_values``
settings
    to preserve and not interpret ``dtype``.
    If ``converters`` are specified, they will be applied INSTEAD
    of ``dtype`` conversion.

    .. versionadded:: 1.5.0

        Support for ``defaultdict`` was added. Specify a
``defaultdict`` as input where
        the default determines the ``dtype`` of the columns which are
not explicitly
        listed.
engine : {'c', 'python', 'pyarrow'}, optional
    Parser engine to use. The C and pyarrow engines are faster, while
the python engine
    is currently more feature-complete. Multithreading is currently
only supported by
    the pyarrow engine.

    .. versionadded:: 1.4.0

        The 'pyarrow' engine was added as an *experimental* engine,
and some features
        are unsupported, or may not work correctly, with this engine.
converters : dict of {Hashable : Callable}, optional
    Functions for converting values in specified columns. Keys can
either
    be column labels or column indices.
true_values : list, optional
```

```
    Values to consider as ``True`` in addition to case-insensitive
variants of 'True'.
false_values : list, optional
    Values to consider as ``False`` in addition to case-insensitive
variants of 'False'.
skipinitialspace : bool, default False
    Skip spaces after delimiter.
skiprows : int, list of int or Callable, optional
    Line numbers to skip (0-indexed) or number of lines to skip
(``int``)
    at the start of the file.

    If callable, the callable function will be evaluated against the
row
    indices, returning ``True`` if the row should be skipped and
``False`` otherwise.
    An example of a valid callable argument would be ``lambda x: x in
[0, 2]``.
skipfooter : int, default 0
    Number of lines at bottom of file to skip (Unsupported with
``engine='c'``).
nrows : int, optional
    Number of rows of file to read. Useful for reading pieces of large
files.
na_values : Hashable, Iterable of Hashable or dict of {Hashable :
Iterable}, optional
    Additional strings to recognize as ``NA``/``NaN``. If ``dict``
passed, specific
    per-column ``NA`` values.  By default the following values are
interpreted as
    ``NaN``: " ", "#N/A", "#N/A N/A", "#NA", "-1.#IND", "-1.#QNAN", "-
NaN", "-nan",
    "1.#IND", "1.#QNAN", "<NA>", "N/A", "NA", "NULL", "NaN", "None",
    "n/a", "nan", "null ".

keep_default_na : bool, default True
    Whether or not to include the default ``NaN`` values when parsing
the data.
    Depending on whether ``na_values`` is passed in, the behavior is
as follows:

    * If ``keep_default_na`` is ``True``, and ``na_values`` are
specified, ``na_values``
      is appended to the default ``NaN`` values used for parsing.
    * If ``keep_default_na`` is ``True``, and ``na_values`` are not
specified, only
      the default ``NaN`` values are used for parsing.
    * If ``keep_default_na`` is ``False``, and ``na_values`` are
specified, only
      the ``NaN`` values specified ``na_values`` are used for parsing.
```

```
    * If ``keep_default_na`` is ``False``, and ``na_values`` are not
specified, no
      strings will be parsed as ``NaN``.

    Note that if ``na_filter`` is passed in as ``False``, the
``keep_default_na`` and
    ``na_values`` parameters will be ignored.
na_filter : bool, default True
    Detect missing value markers (empty strings and the value of
``na_values``). In
    data without any ``NA`` values, passing ``na_filter=False`` can
improve the
    performance of reading a large file.
verbose : bool, default False
    Indicate number of ``NA`` values placed in non-numeric columns.

    .. deprecated:: 2.2.0
skip_blank_lines : bool, default True
    If ``True``, skip over blank lines rather than interpreting as
``NaN`` values.
parse_dates : bool, list of Hashable, list of lists or dict of
{Hashable : list}, default False
    The behavior is as follows:

    * ``bool``. If ``True`` -> try parsing the index. Note:
Automatically set to
      ``True`` if ``date_format`` or ``date_parser`` arguments have
been passed.
    * ``list`` of ``int`` or names. e.g. If ``[1, 2, 3]`` -> try
parsing columns 1, 2, 3
      each as a separate date column.
    * ``list`` of ``list``. e.g.  If ``[[1, 3]]`` -> combine columns 1
and 3 and parse
      as a single date column. Values are joined with a space before
parsing.
    * ``dict``, e.g. ``{'foo' : [1, 3]}`` -> parse columns 1, 3 as
date and call
      result 'foo'. Values are joined with a space before parsing.

    If a column or index cannot be represented as an array of
``datetime``,
    say because of an unparsable value or a mixture of timezones, the
column
    or index will be returned unaltered as an ``object`` data type.
For
    non-standard ``datetime`` parsing, use :func:`~pandas.to_datetime`
after
    :func:`~pandas.read_csv`.

    Note: A fast-path exists for iso8601-formatted dates.
```

```
infer_datetime_format : bool, default False
    If ``True`` and ``parse_dates`` is enabled, pandas will attempt to
infer the
    format of the ``datetime`` strings in the columns, and if it can
be inferred,
    switch to a faster method of parsing them. In some cases this can
increase
    the parsing speed by 5-10x.

    .. deprecated:: 2.0.0
        A strict version of this argument is now the default, passing
it has no effect.

keep_date_col : bool, default False
    If ``True`` and ``parse_dates`` specifies combining multiple
columns then
    keep the original columns.
date_parser : Callable, optional
    Function to use for converting a sequence of string columns to an
array of
    ``datetime`` instances. The default uses
``dateutil.parser.parser`` to do the
    conversion. pandas will try to call ``date_parser`` in three
different ways,
    advancing to the next if an exception occurs: 1) Pass one or more
arrays
    (as defined by ``parse_dates``) as arguments; 2) concatenate (row-
wise) the
    string values from the columns defined by ``parse_dates`` into a
single array
    and pass that; and 3) call ``date_parser`` once for each row using
one or
    more strings (corresponding to the columns defined by
``parse_dates``) as
    arguments.

    .. deprecated:: 2.0.0
       Use ``date_format`` instead, or read in as ``object`` and then
apply
       :func:`~pandas.to_datetime` as-needed.
date_format : str or dict of column -> format, optional
    Format to use for parsing dates when used in conjunction with
``parse_dates``.
    The strftime to parse time, e.g. :const:`"%d/%m/%Y"`. See
    `strftime documentation
    <https://docs.python.org/3/library/datetime.html
    #strftime-and-strptime-behavior>`_ for more information on
choices, though
    note that :const:`"%f"` will parse all the way up to nanoseconds.
    You can also pass:
```

```
    - "ISO8601", to parse any `ISO8601
<https://en.wikipedia.org/wiki/ISO_8601>`_
        time string (not necessarily in exactly the same format);
    - "mixed", to infer the format for each element individually. This
is risky,
        and you should probably use it along with `dayfirst`.

    .. versionadded:: 2.0.0
dayfirst : bool, default False
    DD/MM format dates, international and European format.
cache_dates : bool, default True
    If ``True``, use a cache of unique, converted dates to apply the
``datetime``
    conversion. May produce significant speed-up when parsing
duplicate
    date strings, especially ones with timezone offsets.

iterator : bool, default False
    Return ``TextFileReader`` object for iteration or getting chunks
with
    ``get_chunk()``.
chunksize : int, optional
    Number of lines to read from the file per chunk. Passing a value
will cause the
    function to return a ``TextFileReader`` object for iteration.
    See the `IO Tools docs
    <https://pandas.pydata.org/pandas-docs/stable/io.html#io-
chunking>`_
    for more information on ``iterator`` and ``chunksize``.

compression : str or dict, default 'infer'
    For on-the-fly decompression of on-disk data. If 'infer' and
'filepath_or_buffer' is
    path-like, then detect compression from the following extensions:
'.gz',
    '.bz2', '.zip', '.xz', '.zst', '.tar', '.tar.gz', '.tar.xz' or
'.tar.bz2'
    (otherwise no compression).
    If using 'zip' or 'tar', the ZIP file must contain only one data
file to be read in.
    Set to ``None`` for no decompression.
    Can also be a dict with key ``'method'`` set
    to one of {``'zip'``, ``'gzip'``, ``'bz2'``, ``'zstd'``, ``'xz'``,
``'tar'``} and
    other key-value pairs are forwarded to
    ``zipfile.ZipFile``, ``gzip.GzipFile``,
    ``bz2.BZ2File``, ``zstandard.ZstdDecompressor``, ``lzma.LZMAFile``
or
    ``tarfile.TarFile``, respectively.
```

```
    As an example, the following could be passed for Zstandard
decompression using a
    custom compression dictionary:
    ``compression={'method': 'zstd', 'dict_data':
my_compression_dict}``.

    .. versionadded:: 1.5.0
        Added support for `.tar` files.

    .. versionchanged:: 1.4.0 Zstandard support.

thousands : str (length 1), optional
    Character acting as the thousands separator in numerical values.
decimal : str (length 1), default '.'
    Character to recognize as decimal point (e.g., use ',' for
European data).
lineterminator : str (length 1), optional
    Character used to denote a line break. Only valid with C parser.
quotechar : str (length 1), optional
    Character used to denote the start and end of a quoted item.
Quoted
    items can include the ``delimiter`` and it will be ignored.
quoting : {0 or csv.QUOTE_MINIMAL, 1 or csv.QUOTE_ALL, 2 or
csv.QUOTE_NONNUMERIC, 3 or csv.QUOTE_NONE}, default csv.QUOTE_MINIMAL
    Control field quoting behavior per ``csv.QUOTE_*`` constants.
Default is
    ``csv.QUOTE_MINIMAL`` (i.e., 0) which implies that only fields
containing special
    characters are quoted (e.g., characters defined in ``quotechar``,
``delimiter``,
    or ``lineterminator``.
doublequote : bool, default True
   When ``quotechar`` is specified and ``quoting`` is not
``QUOTE_NONE``, indicate
   whether or not to interpret two consecutive ``quotechar`` elements
INSIDE a
   field as a single ``quotechar`` element.
escapechar : str (length 1), optional
    Character used to escape other characters.
comment : str (length 1), optional
    Character indicating that the remainder of line should not be
parsed.
    If found at the beginning
    of a line, the line will be ignored altogether. This parameter
must be a
    single character. Like empty lines (as long as
``skip_blank_lines=True``),
    fully commented lines are ignored by the parameter ``header`` but
not by
    ``skiprows``. For example, if ``comment='#'``, parsing
```

```
    ``#empty\na,b,c\n1,2,3`` with ``header=0`` will result in
``'a,b,c'`` being
    treated as the header.
encoding : str, optional, default 'utf-8'
    Encoding to use for UTF when reading/writing (ex. ``'utf-8'``).
`List of Python
    standard encodings
    <https://docs.python.org/3/library/codecs.html#standard-
encodings>`_ .

encoding_errors : str, optional, default 'strict'
    How encoding errors are treated. `List of possible values
    <https://docs.python.org/3/library/codecs.html#error-handlers>`_ .

    .. versionadded:: 1.3.0

dialect : str or csv.Dialect, optional
    If provided, this parameter will override values (default or not)
for the
    following parameters: ``delimiter``, ``doublequote``,
``escapechar``,
    ``skipinitialspace``, ``quotechar``, and ``quoting``. If it is
necessary to
    override values, a ``ParserWarning`` will be issued. See
``csv.Dialect``
    documentation for more details.
on_bad_lines : {'error', 'warn', 'skip'} or Callable, default 'error'
    Specifies what to do upon encountering a bad line (a line with too
many fields).
    Allowed values are :

    - ``'error'``, raise an Exception when a bad line is encountered.
    - ``'warn'``, raise a warning when a bad line is encountered and
skip that line.
    - ``'skip'``, skip bad lines without raising or warning when they
are encountered.

    .. versionadded:: 1.3.0

    .. versionadded:: 1.4.0

        - Callable, function with signature
          ``(bad_line: list[str]) -> list[str] | None`` that will
process a single
          bad line. ``bad_line`` is a list of strings split by the
``sep``.
          If the function returns ``None``, the bad line will be
ignored.
          If the function returns a new ``list`` of strings with more
elements than
```

```
          expected, a ``ParserWarning`` will be emitted while dropping
extra elements.
          Only supported when ``engine='python'``

    .. versionchanged:: 2.2.0

        - Callable, function with signature
          as described in `pyarrow documentation

<https://arrow.apache.org/docs/python/generated/pyarrow.csv.ParseOptio
ns.html
          #pyarrow.csv.ParseOptions.invalid_row_handler>`_ when
``engine='pyarrow'``

delim_whitespace : bool, default False
    Specifies whether or not whitespace (e.g. ``' '`` or ``'\t'``)
will be
    used as the ``sep`` delimiter. Equivalent to setting ``sep='\
s+'``. If this option
    is set to ``True``, nothing should be passed in for the
``delimiter``
    parameter.

    .. deprecated:: 2.2.0
        Use ``sep="\s+"`` instead.
low_memory : bool, default True
    Internally process the file in chunks, resulting in lower memory
use
    while parsing, but possibly mixed type inference.  To ensure no
mixed
    types either set ``False``, or specify the type with the ``dtype``
parameter.
    Note that the entire file is read into a
single :class:`~pandas.DataFrame`
    regardless, use the ``chunksize`` or ``iterator`` parameter to
return the data in
    chunks. (Only valid with C parser).
memory_map : bool, default False
    If a filepath is provided for ``filepath_or_buffer``, map the file
object
    directly onto memory and access the data directly from there.
Using this
    option can improve performance because there is no longer any I/O
overhead.
float_precision : {'high', 'legacy', 'round_trip'}, optional
    Specifies which converter the C engine should use for floating-
point
    values. The options are ``None`` or ``'high'`` for the ordinary
converter,
    ``'legacy'`` for the original lower precision pandas converter,
```

```
and
    ``'round_trip'`` for the round-trip converter.

storage_options : dict, optional
    Extra options that make sense for a particular storage connection,
e.g.
    host, port, username, password, etc. For HTTP(S) URLs the key-
value pairs
    are forwarded to ``urllib.request.Request`` as header options. For
other
    URLs (e.g. starting with "s3://", and "gcs://") the key-value
pairs are
    forwarded to ``fsspec.open``. Please see ``fsspec`` and ``urllib``
for more
    details, and for more examples on storage options refer `here
    <https://pandas.pydata.org/docs/user_guide/io.html?
    highlight=storage_options#reading-writing-remote-files>`_.

dtype_backend : {'numpy_nullable', 'pyarrow'}, default
'numpy_nullable'
    Back-end data type applied to the resultant :class:`DataFrame`
    (still experimental). Behaviour is as follows:

    * ``"numpy_nullable"``: returns nullable-dtype-
backed :class:`DataFrame`
      (default).
    * ``"pyarrow"``: returns pyarrow-backed
nullable :class:`ArrowDtype`
      DataFrame.

    .. versionadded:: 2.0

Returns
-------
DataFrame or TextFileReader
    A comma-separated values (csv) file is returned as two-dimensional
    data structure with labeled axes.

See Also
--------
DataFrame.to_csv : Write DataFrame to a comma-separated values (csv)
file.
read_table : Read general delimited file into DataFrame.
read_fwf : Read a table of fixed-width formatted lines into DataFrame.

Examples
--------
>>> pd.read_csv('data.csv')  # doctest: +SKIP
File:
```

```
~/.local/lib/python3.12/site-packages/pandas/io/parsers/readers.py
Type:      function
```

viewing the DataFrame

```
df
```

```
     X  Y month  day  fuel_code  moisture_code  drought_code  \
0    7  5   mar  fri       86.2           26.2          94.3
1    7  4   oct  tue       90.6           35.4         669.1
2    7  4   oct  sat       90.6           43.7         686.9
3    8  6   mar  fri       91.7           33.3          77.5
4    8  6   mar  sun       89.3           51.3         102.2
..  .. ..   ...  ...        ...            ...           ...
512  4  3   aug  sun       81.6           56.7         665.6
513  2  4   aug  sun       81.6           56.7         665.6
514  7  4   aug  sun       81.6           56.7         665.6
515  1  4   aug  sat       94.4          146.0         614.7
516  6  3   nov  tue       79.5            3.0         106.7

     initial_spread_code  temp  humidity  wind  rain  area_burned
0                    5.1   8.2        51   6.7   0.0         0.00
1                    6.7  18.0        33   0.9   0.0         0.00
2                    6.7  14.6        33   1.3   0.0         0.00
3                    9.0   8.3        97   4.0   0.2         0.00
4                    9.6  11.4        99   1.8   0.0         0.00
..                   ...   ...       ...   ...   ...          ...
512                  1.9  27.8        32   2.7   0.0         6.44
513                  1.9  21.9        71   5.8   0.0        54.29
514                  1.9  21.2        70   6.7   0.0        11.16
515                 11.3  25.6        42   4.0   0.0         0.00
516                  1.1  11.8        31   4.5   0.0         0.00

[517 rows x 13 columns]
```

Take a minute to look at the data. The DataFrame will have a slightly different look on different versions of Jupyter. The number at the beginning of each row is called an **index**. The index was automatically assigned by pandas when the dataset was loaded. It was not in the original csv file. It is merely a series of consecutive numbers going down the rows. The rows were loaded in whatever order they were in the csv file.

There are ways to view pieces of the DataFrame. Try these to see what they do:

```
df.head()
```

```
   X  Y month  day  fuel_code  moisture_code  drought_code  \
0  7  5   mar  fri       86.2           26.2          94.3
1  7  4   oct  tue       90.6           35.4         669.1
2  7  4   oct  sat       90.6           43.7         686.9
3  8  6   mar  fri       91.7           33.3          77.5
```

```
4  8  6   mar  sun       89.3           51.3         102.2
```

```
   initial_spread_code  temp  humidity  wind  rain  area_burned
0                  5.1   8.2        51   6.7   0.0          0.0
1                  6.7  18.0        33   0.9   0.0          0.0
2                  6.7  14.6        33   1.3   0.0          0.0
3                  9.0   8.3        97   4.0   0.2          0.0
4                  9.6  11.4        99   1.8   0.0          0.0
```

```
df.tail()
```

```
     X  Y month  day  fuel_code  moisture_code  drought_code  \
512  4  3   aug  sun       81.6           56.7         665.6
513  2  4   aug  sun       81.6           56.7         665.6
514  7  4   aug  sun       81.6           56.7         665.6
515  1  4   aug  sat       94.4          146.0         614.7
516  6  3   nov  tue       79.5            3.0         106.7
```

```
     initial_spread_code  temp  humidity  wind  rain  area_burned
512                  1.9  27.8        32   2.7   0.0         6.44
513                  1.9  21.9        71   5.8   0.0        54.29
514                  1.9  21.2        70   6.7   0.0        11.16
515                 11.3  25.6        42   4.0   0.0         0.00
516                  1.1  11.8        31   4.5   0.0         0.00
```

```
df.tail(2)
```

```
     X  Y month  day  fuel_code  moisture_code  drought_code  \
515  1  4   aug  sat       94.4          146.0         614.7
516  6  3   nov  tue       79.5            3.0         106.7
```

```
     initial_spread_code  temp  humidity  wind  rain  area_burned
515                 11.3  25.6        42   4.0   0.0          0.0
516                  1.1  11.8        31   4.5   0.0          0.0
```

```
df.sample()
```

```
     X  Y month  day  fuel_code  moisture_code  drought_code  \
140  2  5   sep  mon       90.9          126.5         686.5
```

```
     initial_spread_code  temp  humidity  wind  rain  area_burned
140                  7.0  21.9        39   1.8   0.0         0.47
```

```
df.sample(6)
```

```
     X  Y month  day  fuel_code  moisture_code  drought_code  \
179  8  6   aug  tue       88.8          147.3         614.5
109  4  5   sep  mon       88.6           91.8         709.9
62   2  2   aug  thu       93.0           75.3         466.6
180  1  3   sep  sun       92.4          124.1         680.7
488  4  4   aug  tue       95.1          141.3         605.8
```

```
293  7  6   jul  tue       93.1          180.4         430.8

     initial_spread_code  temp  humidity  wind  rain  area_burned
179                  9.0  14.4        66   5.4   0.0         5.23
109                  7.1  17.4        56   5.4   0.0         0.00
62                   7.7  18.8        35   4.9   0.0         0.00
180                  8.5  23.9        32   6.7   0.0         5.33
488                 17.7  19.4        71   7.6   0.0        46.70
293                 11.0  26.9        28   5.4   0.0        86.45
```

getting basic info about the DataFrame

```
len(df)

517

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 517 entries, 0 to 516
Data columns (total 13 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   X                    517 non-null    int64
 1   Y                    517 non-null    int64
 2   month                517 non-null    object
 3   day                  517 non-null    object
 4   fuel_code            517 non-null    float64
 5   moisture_code        517 non-null    float64
 6   drought_code         517 non-null    float64
 7   initial_spread_code  517 non-null    float64
 8   temp                 517 non-null    float64
 9   humidity             517 non-null    int64
 10  wind                 517 non-null    float64
 11  rain                 517 non-null    float64
 12  area_burned          517 non-null    float64
dtypes: float64(8), int64(3), object(2)
memory usage: 52.6+ KB

df.describe() # Generate descriptive statistics

                X           Y   fuel_code  moisture_code  drought_code
\
count  517.000000  517.000000  517.000000     517.000000    517.000000

mean     4.669246    4.299807   90.644681     110.872340    547.940039

std      2.313778    1.229900    5.520111      64.046482    248.066192

min      1.000000    2.000000   18.700000       1.100000      7.900000
```

```
25%      3.000000    4.000000   90.200000      68.600000    437.700000

50%      4.000000    4.000000   91.600000     108.300000    664.200000

75%      7.000000    5.000000   92.900000     142.400000    713.900000

max      9.000000    9.000000   96.200000     291.300000    860.600000


       initial_spread_code        temp    humidity        wind
rain  \
count           517.000000  517.000000  517.000000  517.000000
517.000000
mean              9.021663   18.889168   44.288201    4.017602
0.021663
std               4.559477    5.806625   16.317469    1.791653
0.295959
min               0.000000    2.200000   15.000000    0.400000
0.000000
25%               6.500000   15.500000   33.000000    2.700000
0.000000
50%               8.400000   19.300000   42.000000    4.000000
0.000000
75%              10.800000   22.800000   53.000000    4.900000
0.000000
max              56.100000   33.300000  100.000000    9.400000
6.400000

       area_burned
count   517.000000
mean     12.847292
std      63.655818
min       0.000000
25%       0.000000
50%       0.520000
75%       6.570000
max    1090.840000
```

```
df.shape # Return a tuple representing the dimensionality of the
DataFrame.
```

```
(517, 13)
```

```
df.size # The number of elements in this object
```

```
6721
```

```
517 * 13
```

```
6721
```

You may have noticed that `df.size` and `df.shape` do not have parentheses. These are called **attributes**. They tell you something about the object and don't do anything to or with the object. Not all objects have attributes, but they will only work without the parentheses.

return the row indices or column names

```
df.index
```

```
RangeIndex(start=0, stop=517, step=1)
```

```
df.columns
```

```
Index(['X', 'Y', 'month', 'day', 'fuel_code', 'moisture_code',
'drought_code',
       'initial_spread_code', 'temp', 'humidity', 'wind', 'rain',
       'area_burned'],
      dtype='object')
```

Those look strange because they are pandas objects. You can make them into a list so that they are easier to work with:

```
list(df.columns)
```

```
['X',
 'Y',
 'month',
 'day',
 'fuel_code',
 'moisture_code',
 'drought_code',
 'initial_spread_code',
 'temp',
 'humidity',
 'wind',
 'rain',
 'area_burned']
```

```
len(list(df.index))
```

```
517
```

```
column_names = list(df.columns)
print(column_names)
```

```
['X', 'Y', 'month', 'day', 'fuel_code', 'moisture_code',
'drought_code', 'initial_spread_code', 'temp', 'humidity', 'wind',
'rain', 'area_burned']
```

transposing a dataframe

```
df.T # Transpose index and columns.
```

```
                        0      1      2     3      4      5      6
7    \
X                       7      7      7     8      8      8      8
8
Y                       5      4      4     6      6      6      6
6
month                 mar    oct    oct   mar    mar    aug    aug
aug
day                   fri    tue    sat   fri    sun    sun    mon
mon
fuel_code            86.2   90.6   90.6  91.7   89.3   92.3   92.3
91.5
moisture_code        26.2   35.4   43.7  33.3   51.3   85.3   88.9
145.4
drought_code         94.3  669.1  686.9  77.5  102.2  488.0  495.6
608.2
initial_spread_code   5.1    6.7    6.7   9.0    9.6   14.7    8.5
10.7
temp                  8.2   18.0   14.6   8.3   11.4   22.2   24.1
8.0
humidity               51     33     33    97     99     29     27
86
wind                  6.7    0.9    1.3   4.0    1.8    5.4    3.1
2.2
rain                  0.0    0.0    0.0   0.2    0.0    0.0    0.0
0.0
area_burned           0.0    0.0    0.0   0.0    0.0    0.0    0.0
0.0

                        8      9    ...    507    508    509    510
511  \
X                       8      7  ...      2      1      5      6
8
Y                       6      5  ...      4      2      4      5
6
month                 sep    sep  ...    aug    aug    aug    aug
aug
day                   tue    sat  ...    fri    fri    fri    fri
sun
fuel_code            91.0   92.5  ...   91.0   91.0   91.0   91.0
81.6
moisture_code       129.5   88.0  ...  166.9  166.9  166.9  166.9
56.7
drought_code        692.6  698.6  ...  752.6  752.6  752.6  752.6
665.6
initial_spread_code   7.0    7.1  ...    7.1    7.1    7.1    7.1
1.9
temp                 13.1   22.8  ...   25.9   25.9   21.1   18.2
27.8
humidity               63     40  ...     41     41     71     62
```

```
35
wind                   5.4    4.0  ...    3.6    3.6    7.6    5.4
2.7
rain                   0.0    0.0  ...    0.0    0.0    1.4    0.0
0.0
area_burned            0.0    0.0  ...    0.0    0.0   2.17   0.43
0.0

                       512    513    514    515    516
X                        4      2      7      1      6
Y                        3      4      4      4      3
month                  aug    aug    aug    aug    nov
day                    sun    sun    sun    sat    tue
fuel_code             81.6   81.6   81.6   94.4   79.5
moisture_code         56.7   56.7   56.7  146.0    3.0
drought_code         665.6  665.6  665.6  614.7  106.7
initial_spread_code    1.9    1.9    1.9   11.3    1.1
temp                  27.8   21.9   21.2   25.6   11.8
humidity                32     71     70     42     31
wind                   2.7    5.8    6.7    4.0    4.5
rain                   0.0    0.0    0.0    0.0    0.0
area_burned           6.44  54.29  11.16    0.0    0.0

[13 rows x 517 columns]
```

Let's see if that changed our DataFrame object:

```
df

     X  Y month  day  fuel_code  moisture_code  drought_code  \
0    7  5   mar  fri       86.2           26.2          94.3
1    7  4   oct  tue       90.6           35.4         669.1
2    7  4   oct  sat       90.6           43.7         686.9
3    8  6   mar  fri       91.7           33.3          77.5
4    8  6   mar  sun       89.3           51.3         102.2
..  .. ..   ...  ...        ...            ...           ...
512  4  3   aug  sun       81.6           56.7         665.6
513  2  4   aug  sun       81.6           56.7         665.6
514  7  4   aug  sun       81.6           56.7         665.6
515  1  4   aug  sat       94.4          146.0         614.7
516  6  3   nov  tue       79.5            3.0         106.7

     initial_spread_code  temp  humidity  wind  rain  area_burned
0                    5.1   8.2        51   6.7   0.0         0.00
1                    6.7  18.0        33   0.9   0.0         0.00
2                    6.7  14.6        33   1.3   0.0         0.00
3                    9.0   8.3        97   4.0   0.2         0.00
4                    9.6  11.4        99   1.8   0.0         0.00
..                   ...   ...       ...   ...   ...          ...
```

```
512                  1.9  27.8        32   2.7   0.0         6.44
513                  1.9  21.9        71   5.8   0.0        54.29
514                  1.9  21.2        70   6.7   0.0        11.16
515                 11.3  25.6        42   4.0   0.0         0.00
516                  1.1  11.8        31   4.5   0.0         0.00

[517 rows x 13 columns]
```

We could save a version of the transposed df:

```
df_t = df.T
df_t
```

```
                       0      1      2     3      4      5      6
7    \
X                      7      7      7     8      8      8      8
8
Y                      5      4      4     6      6      6      6
6
month                mar    oct    oct   mar    mar    aug    aug
aug
day                  fri    tue    sat   fri    sun    sun    mon
mon
fuel_code           86.2   90.6   90.6  91.7   89.3   92.3   92.3
91.5
moisture_code       26.2   35.4   43.7  33.3   51.3   85.3   88.9
145.4
drought_code        94.3  669.1  686.9  77.5  102.2  488.0  495.6
608.2
initial_spread_code  5.1    6.7    6.7   9.0    9.6   14.7    8.5
10.7
temp                 8.2   18.0   14.6   8.3   11.4   22.2   24.1
8.0
humidity              51     33     33    97     99     29     27
86
wind                 6.7    0.9    1.3   4.0    1.8    5.4    3.1
2.2
rain                 0.0    0.0    0.0   0.2    0.0    0.0    0.0
0.0
area_burned          0.0    0.0    0.0   0.0    0.0    0.0    0.0
0.0

                       8      9    ...    507    508    509    510
511  \
X                      8      7  ...      2      1      5      6
8
Y                      6      5  ...      4      2      4      5
6
month                sep    sep  ...    aug    aug    aug    aug
```

```
aug
day                    tue    sat  ...    fri    fri    fri    fri
sun
fuel_code             91.0   92.5  ...   91.0   91.0   91.0   91.0
81.6
moisture_code        129.5   88.0  ...  166.9  166.9  166.9  166.9
56.7
drought_code         692.6  698.6  ...  752.6  752.6  752.6  752.6
665.6
initial_spread_code    7.0    7.1  ...    7.1    7.1    7.1    7.1
1.9
temp                  13.1   22.8  ...   25.9   25.9   21.1   18.2
27.8
humidity                63     40  ...     41     41     71     62
35
wind                   5.4    4.0  ...    3.6    3.6    7.6    5.4
2.7
rain                   0.0    0.0  ...    0.0    0.0    1.4    0.0
0.0
area_burned            0.0    0.0  ...    0.0    0.0   2.17   0.43
0.0

                       512    513    514    515    516
X                        4      2      7      1      6
Y                        3      4      4      4      3
month                  aug    aug    aug    aug    nov
day                    sun    sun    sun    sat    tue
fuel_code             81.6   81.6   81.6   94.4   79.5
moisture_code         56.7   56.7   56.7  146.0    3.0
drought_code         665.6  665.6  665.6  614.7  106.7
initial_spread_code    1.9    1.9    1.9   11.3    1.1
temp                  27.8   21.9   21.2   25.6   11.8
humidity                32     71     70     42     31
wind                   2.7    5.8    6.7    4.0    4.5
rain                   0.0    0.0    0.0    0.0    0.0
area_burned           6.44  54.29  11.16    0.0    0.0

[13 rows x 517 columns]
```

select columns or rows

To create a DataFrame with only some columns, you use indexing, and you pass it a list of the columns that you want to include:

```
my_columns = ["month", "day", "area_burned"]
df[my_columns]
```

```
    month  day  area_burned
0     mar  fri         0.00
```

```
1     oct  tue          0.00
2     oct  sat          0.00
3     mar  fri          0.00
4     mar  sun          0.00
..    ...  ...           ...
512   aug  sun          6.44
513   aug  sun         54.29
514   aug  sun         11.16
515   aug  sat          0.00
516   nov  tue          0.00

[517 rows x 3 columns]
```

OR you could just include the list inside the indexing. This creates two sets of square brackets, which looks a little silly, but it works!

```
df[["month", "day", "area_burned"]]
```

```
    month  day  area_burned
0     mar  fri          0.00
1     oct  tue          0.00
2     oct  sat          0.00
3     mar  fri          0.00
4     mar  sun          0.00
..    ...  ...           ...
512   aug  sun          6.44
513   aug  sun         54.29
514   aug  sun         11.16
515   aug  sat          0.00
516   nov  tue          0.00

[517 rows x 3 columns]
```

The **interior** brackets are for **list**, and the **outside brackets** are **indexing** operator. Use double brackets if you select two or more columns. With one column name, single pair of brackets returns a **Series**, while double brackets return a **dataframe**.

If you want to return just one column as a DataFrame, you still use the list inside the index:

```
df[["temp"]]
```

```
     temp
0     8.2
1    18.0
2    14.6
3     8.3
4    11.4
..    ...
512  27.8
```

```
513  21.9
514  21.2
515  25.6
516  11.8

[517 rows x 1 columns]
```

If you only index the column name, without putting it in a list, you get a different type of object - the **Series** object.

```
df["temp"]
```

```
0       8.2
1      18.0
2      14.6
3       8.3
4      11.4
       ...
512    27.8
513    21.9
514    21.2
515    25.6
516    11.8
Name: temp, Length: 517, dtype: float64
```

A Series object only returns the values from one column. It can be turned into a list, which is very convenient:

```
temp_list = list(df["temp"])
print(temp_list)
```

```
[8.2, 18.0, 14.6, 8.3, 11.4, 22.2, 24.1, 8.0, 13.1, 22.8, 17.8, 19.3,
17.0, 21.3, 26.4, 22.9, 15.1, 16.7, 15.9, 9.3, 18.3, 19.1, 21.0, 19.5,
23.7, 16.3, 19.0, 19.4, 30.2, 22.8, 25.4, 11.2, 20.6, 17.7, 21.2,
18.2, 21.7, 11.3, 17.8, 14.1, 23.3, 18.4, 16.6, 19.6, 12.9, 25.9,
14.7, 23.0, 11.8, 11.0, 20.8, 21.5, 20.4, 20.4, 17.6, 27.7, 17.8,
13.8, 13.9, 12.3, 11.5, 5.5, 18.8, 20.8, 23.1, 18.6, 23.0, 19.6, 19.6,
17.2, 15.8, 17.7, 15.6, 17.3, 27.6, 6.7, 15.7, 8.3, 14.7, 21.6, 19.5,
17.9, 18.6, 16.6, 20.2, 21.5, 25.4, 22.4, 25.3, 17.4, 14.7, 17.4,
20.8, 18.2, 23.4, 17.8, 12.7, 17.4, 11.6, 19.8, 19.8, 14.4, 20.1,
24.1, 5.3, 12.7, 18.2, 21.4, 20.3, 17.4, 13.7, 18.8, 22.8, 18.9, 15.8,
15.5, 11.6, 15.2, 10.6, 19.6, 10.3, 17.1, 22.5, 17.9, 19.8, 20.6, 9.0,
17.2, 15.9, 15.4, 15.4, 14.0, 10.6, 17.6, 14.9, 17.6, 17.2, 15.6,
18.0, 21.7, 21.9, 23.3, 21.2, 16.6, 23.8, 27.4, 13.2, 24.2, 17.4,
23.7, 23.2, 24.8, 24.6, 20.1, 29.6, 16.4, 28.6, 18.4, 20.5, 19.0,
16.1, 20.3, 15.2, 17.8, 17.8, 5.3, 16.6, 23.4, 14.6, 20.7, 21.9, 17.4,
20.1, 17.7, 14.2, 20.3, 5.8, 19.2, 18.3, 14.4, 23.9, 19.1, 12.4, 16.8,
20.8, 17.6, 11.5, 21.0, 13.3, 11.5, 11.7, 24.2, 24.6, 24.3, 24.6,
23.5, 5.8, 21.5, 13.9, 22.6, 21.6, 12.4, 8.8, 20.2, 15.1, 22.1, 22.9,
```

```
20.7, 19.6, 23.2, 18.4, 5.1, 20.1, 11.0, 17.0, 17.0, 16.9, 12.4, 19.4,
15.2, 16.2, 18.6, 11.0, 13.4, 15.4, 22.9, 16.1, 20.1, 28.3, 16.4,
26.4, 27.8, 18.7, 24.3, 17.7, 19.6, 18.2, 18.8, 25.1, 13.4, 15.2,
16.7, 15.4, 21.9, 22.4, 26.8, 25.7, 20.7, 28.7, 21.7, 26.8, 24.0,
22.1, 21.4, 18.9, 22.3, 23.9, 21.4, 20.6, 23.7, 28.3, 11.2, 21.4,
19.3, 21.8, 22.1, 19.4, 23.7, 21.0, 19.1, 21.8, 20.1, 20.2, 4.8, 5.1,
5.1, 4.6, 4.6, 4.6, 4.6, 2.2, 5.1, 4.2, 8.8, 7.5, 23.4, 12.6, 22.1,
24.2, 24.3, 18.7, 25.3, 22.9, 26.9, 17.1, 22.2, 14.3, 15.4, 19.6,
10.6, 20.7, 19.1, 19.2, 19.2, 11.3, 19.0, 17.1, 23.8, 16.0, 24.9,
25.3, 24.8, 12.2, 24.3, 19.7, 18.5, 18.6, 19.2, 21.6, 21.6, 18.9,
16.8, 16.8, 12.9, 13.7, 24.2, 24.1, 21.2, 19.7, 23.5, 24.2, 21.5,
17.1, 18.1, 18.0, 9.8, 19.3, 23.0, 22.7, 20.4, 19.3, 15.7, 20.6, 15.9,
12.2, 16.8, 21.3, 10.1, 17.4, 12.8, 10.1, 15.4, 20.6, 19.8, 18.7,
20.8, 20.8, 15.9, 19.7, 21.1, 18.4, 17.3, 15.2, 15.9, 21.1, 19.6,
15.9, 16.4, 16.8, 13.8, 13.8, 14.2, 10.4, 20.3, 10.3, 15.4, 21.1,
21.9, 8.7, 5.2, 19.3, 16.2, 28.2, 20.5, 21.3, 20.9, 20.6, 11.6, 23.3,
23.3, 7.5, 20.7, 21.9, 15.2, 5.3, 10.1, 20.4, 24.3, 25.9, 28.0, 28.0,
22.8, 25.0, 21.3, 21.8, 27.9, 17.0, 14.2, 19.9, 23.4, 14.7, 8.2, 22.8,
26.4, 24.1, 27.5, 26.3, 13.8, 24.9, 24.8, 26.2, 30.8, 29.3, 22.3,
26.9, 20.4, 20.4, 27.9, 26.2, 24.6, 19.4, 23.3, 23.9, 20.9, 22.2,
23.8, 26.8, 14.2, 23.6, 19.1, 16.2, 25.5, 10.9, 14.8, 16.2, 17.3,
19.1, 8.9, 10.5, 19.3, 23.4, 11.8, 17.7, 17.4, 16.8, 17.9, 16.6, 19.9,
18.9, 15.5, 18.9, 18.9, 14.5, 4.6, 5.1, 4.6, 10.2, 11.2, 13.3, 13.7,
17.6, 18.0, 14.3, 24.5, 26.4, 22.7, 27.2, 26.1, 18.2, 22.6, 30.2,
30.2, 23.4, 31.0, 33.1, 30.6, 24.1, 26.4, 19.4, 20.6, 28.7, 32.4,
32.4, 27.5, 30.8, 23.9, 32.6, 32.3, 33.3, 27.3, 21.6, 21.6, 20.7,
29.2, 28.9, 26.7, 18.5, 25.9, 25.9, 21.1, 18.2, 27.8, 27.8, 21.9,
21.2, 25.6, 11.8]
```

If we want to return a DataFrame with only some rows, we can index a range:

```
df[0:10]
```

	X	Y	month	day	fuel_code	moisture_code	drought_code	initial_spread_code	temp	humidity	wind	rain	area_burned
0	7	5	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.0
1	7	4	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.0
2	7	4	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.0
3	8	6	mar	fri	91.7	33.3	77.5						
4	8	6	mar	sun	89.3	51.3	102.2						
5	8	6	aug	sun	92.3	85.3	488.0						
6	8	6	aug	mon	92.3	88.9	495.6						
7	8	6	aug	mon	91.5	145.4	608.2						
8	8	6	sep	tue	91.0	129.5	692.6						
9	7	5	sep	sat	92.5	88.0	698.6						

```
3                 9.0   8.3         97   4.0   0.2           0.0
4                 9.6  11.4         99   1.8   0.0           0.0
5                14.7  22.2         29   5.4   0.0           0.0
6                 8.5  24.1         27   3.1   0.0           0.0
7                10.7   8.0         86   2.2   0.0           0.0
8                 7.0  13.1         63   5.4   0.0           0.0
9                 7.1  22.8         40   4.0   0.0           0.0
```

```
df[495:-12]
```

```
     X  Y month  day  fuel_code  moisture_code  drought_code  \
495  6  6   aug  mon       96.2          175.5         661.8
496  4  5   aug  mon       96.2          175.5         661.8
497  3  4   aug  tue       96.1          181.1         671.2
498  6  5   aug  tue       96.1          181.1         671.2
499  7  5   aug  tue       96.1          181.1         671.2
500  8  6   aug  tue       96.1          181.1         671.2
501  7  5   aug  tue       96.1          181.1         671.2
502  4  4   aug  tue       96.1          181.1         671.2
503  2  4   aug  wed       94.5          139.4         689.1
504  4  3   aug  wed       94.5          139.4         689.1

     initial_spread_code  temp  humidity  wind  rain  area_burned
495                 16.8  23.9        42   2.2   0.0         0.00
496                 16.8  32.6        26   3.1   0.0         2.77
497                 14.3  32.3        27   2.2   0.0        14.68
498                 14.3  33.3        26   2.7   0.0        40.54
499                 14.3  27.3        63   4.9   6.4        10.82
500                 14.3  21.6        65   4.9   0.8         0.00
501                 14.3  21.6        65   4.9   0.8         0.00
502                 14.3  20.7        69   4.9   0.4         0.00
503                 20.0  29.2        30   4.9   0.0         1.95
504                 20.0  28.9        29   4.9   0.0        49.59
```

If you only want a single row, you still need to use indexing with a `:`:

```
df[4:5]
```

```
   X  Y month  day  fuel_code  moisture_code  drought_code  \
4  8  6   mar  sun       89.3           51.3         102.2

   initial_spread_code  temp  humidity  wind  rain  area_burned
4                  9.6  11.4        99   1.8   0.0          0.0
```

using a boolean to return parts of a DataFrame

To return a DataFrame that only has rows that meet a certain condition, we use this syntax. The outer `df[]` lets Python know that you want the answer to be returned as a DataFrame, meaning you want all the columns included in the output:

```
df[df["month"] == "aug"]
```

```
     X  Y month  day  fuel_code  moisture_code  drought_code  \
5    8  6   aug  sun       92.3           85.3         488.0
6    8  6   aug  mon       92.3           88.9         495.6
7    8  6   aug  mon       91.5          145.4         608.2
12   6  5   aug  fri       63.5           70.8         665.3
23   7  4   aug  sat       90.2          110.9         537.4
..  .. ..   ...  ...        ...            ...           ...
511  8  6   aug  sun       81.6           56.7         665.6
512  4  3   aug  sun       81.6           56.7         665.6
513  2  4   aug  sun       81.6           56.7         665.6
514  7  4   aug  sun       81.6           56.7         665.6
515  1  4   aug  sat       94.4          146.0         614.7

     initial_spread_code  temp  humidity  wind  rain  area_burned
5                   14.7  22.2        29   5.4   0.0         0.00
6                    8.5  24.1        27   3.1   0.0         0.00
7                   10.7   8.0        86   2.2   0.0         0.00
12                   0.8  17.0        72   6.7   0.0         0.00
23                   6.2  19.5        43   5.8   0.0         0.00
..                   ...   ...       ...   ...   ...          ...
511                  1.9  27.8        35   2.7   0.0         0.00
512                  1.9  27.8        32   2.7   0.0         6.44
513                  1.9  21.9        71   5.8   0.0        54.29
514                  1.9  21.2        70   6.7   0.0        11.16
515                 11.3  25.6        42   4.0   0.0         0.00

[184 rows x 13 columns]
```

```
df[df["temp"] > 20]
```

```
     X  Y month  day  fuel_code  moisture_code  drought_code  \
5    8  6   aug  sun       92.3           85.3         488.0
6    8  6   aug  mon       92.3           88.9         495.6
9    7  5   sep  sat       92.5           88.0         698.6
13   6  5   sep  mon       90.9          126.5         686.5
14   6  5   sep  wed       92.9          133.3         699.6
..  .. ..   ...  ...        ...            ...           ...
511  8  6   aug  sun       81.6           56.7         665.6
512  4  3   aug  sun       81.6           56.7         665.6
513  2  4   aug  sun       81.6           56.7         665.6
514  7  4   aug  sun       81.6           56.7         665.6
515  1  4   aug  sat       94.4          146.0         614.7

     initial_spread_code  temp  humidity  wind  rain  area_burned
5                   14.7  22.2        29   5.4   0.0         0.00
6                    8.5  24.1        27   3.1   0.0         0.00
9                    7.1  22.8        40   4.0   0.0         0.00
13                   7.0  21.3        42   2.2   0.0         0.00
```

```
14                   9.2  26.4        21   4.5   0.0         0.00
..                   ...   ...       ...   ...   ...          ...
511                  1.9  27.8        35   2.7   0.0         0.00
512                  1.9  27.8        32   2.7   0.0         6.44
513                  1.9  21.9        71   5.8   0.0        54.29
514                  1.9  21.2        70   6.7   0.0        11.16
515                 11.3  25.6        42   4.0   0.0         0.00

[231 rows x 13 columns]
```

If you don't use the outer `df[]` the return is a Series object that returns the boolean value for each row based on the condition you set:

```
df["month"] == "aug"
```

```
0      False
1      False
2      False
3      False
4      False
       ...
512     True
513     True
514     True
515     True
516    False
Name: month, Length: 517, dtype: bool
```

renaming columns

Here's what our column names look like:

```
df.head()
```

```
   X  Y month  day  fuel_code  moisture_code  drought_code  \
0  7  5   mar  fri       86.2           26.2          94.3
1  7  4   oct  tue       90.6           35.4         669.1
2  7  4   oct  sat       90.6           43.7         686.9
3  8  6   mar  fri       91.7           33.3          77.5
4  8  6   mar  sun       89.3           51.3         102.2

   initial_spread_code  temp  humidity  wind  rain  area_burned
0                  5.1   8.2        51   6.7   0.0          0.0
1                  6.7  18.0        33   0.9   0.0          0.0
2                  6.7  14.6        33   1.3   0.0          0.0
3                  9.0   8.3        97   4.0   0.2          0.0
4                  9.6  11.4        99   1.8   0.0          0.0
```

Four of the columns end in "_code". Let's remove that part from the column names. We can use the `rename()` method. We need to pass the function a dictionary of the old name to be replaced as the key and the new name as the value.

```
df.rename(columns = {"moisture_code": "moisture", "fuel_code":
"fuel"})
```

```
     X  Y month  day  fuel  moisture  drought_code
initial_spread_code  temp  \
0    7  5   mar  fri  86.2      26.2          94.3
5.1   8.2
1    7  4   oct  tue  90.6      35.4         669.1
6.7  18.0
2    7  4   oct  sat  90.6      43.7         686.9
6.7  14.6
3    8  6   mar  fri  91.7      33.3          77.5
9.0   8.3
4    8  6   mar  sun  89.3      51.3         102.2
9.6  11.4
..  .. ..   ...  ...   ...       ...           ...                    ..
.   ...
512  4  3   aug  sun  81.6      56.7         665.6
1.9  27.8
513  2  4   aug  sun  81.6      56.7         665.6
1.9  21.9
514  7  4   aug  sun  81.6      56.7         665.6
1.9  21.2
515  1  4   aug  sat  94.4     146.0         614.7
11.3  25.6
516  6  3   nov  tue  79.5       3.0         106.7
1.1  11.8

     humidity  wind  rain  area_burned
0          51   6.7   0.0         0.00
1          33   0.9   0.0         0.00
2          33   1.3   0.0         0.00
3          97   4.0   0.2         0.00
4          99   1.8   0.0         0.00
..        ...   ...   ...          ...
512        32   2.7   0.0         6.44
513        71   5.8   0.0        54.29
514        70   6.7   0.0        11.16
515        42   4.0   0.0         0.00
516        31   4.5   0.0         0.00

[517 rows x 13 columns]
```

```
df.head()
```

```
   X  Y month  day  fuel_code  moisture_code  drought_code  \
0  7  5   mar  fri       86.2           26.2          94.3
1  7  4   oct  tue       90.6           35.4         669.1
2  7  4   oct  sat       90.6           43.7         686.9
3  8  6   mar  fri       91.7           33.3          77.5
4  8  6   mar  sun       89.3           51.3         102.2

   initial_spread_code  temp  humidity  wind  rain  area_burned
0                  5.1   8.2        51   6.7   0.0          0.0
1                  6.7  18.0        33   0.9   0.0          0.0
2                  6.7  14.6        33   1.3   0.0          0.0
3                  9.0   8.3        97   4.0   0.2          0.0
4                  9.6  11.4        99   1.8   0.0          0.0
```

The change didn't stick. We've encountered this before with strings, so we know the answer - reassign it to a variable.

```
df = df.rename(columns = {"moisture_code": "moisture", "fuel_code":
"fuel"})

df.head()
```

```
   X  Y month  day  fuel  moisture  drought_code  initial_spread_code
temp  \
0  7  5   mar  fri  86.2      26.2          94.3                  5.1
8.2
1  7  4   oct  tue  90.6      35.4         669.1                  6.7
18.0
2  7  4   oct  sat  90.6      43.7         686.9                  6.7
14.6
3  8  6   mar  fri  91.7      33.3          77.5                  9.0
8.3
4  8  6   mar  sun  89.3      51.3         102.2                  9.6
11.4

   humidity  wind  rain  area_burned
0        51   6.7   0.0          0.0
1        33   0.9   0.0          0.0
2        33   1.3   0.0          0.0
3        97   4.0   0.2          0.0
4        99   1.8   0.0          0.0
```

dropping rows and columns

Let's drop a single row from the DataFrame. How about row 2? You still have to assign `df` to a variable to make the change permanent:

```
df = df.drop(2)
```

```
df.head()
```

```
   X  Y month  day  fuel  moisture  drought_code  initial_spread_code
temp  \
0  7  5   mar  fri  86.2      26.2          94.3                  5.1
8.2
1  7  4   oct  tue  90.6      35.4         669.1                  6.7
18.0
3  8  6   mar  fri  91.7      33.3          77.5                  9.0
8.3
4  8  6   mar  sun  89.3      51.3         102.2                  9.6
11.4
5  8  6   aug  sun  92.3      85.3         488.0                 14.7
22.2

   humidity  wind  rain  area_burned
0        51   6.7   0.0          0.0
1        33   0.9   0.0          0.0
3        97   4.0   0.2          0.0
4        99   1.8   0.0          0.0
5        29   5.4   0.0          0.0
```

The index numbers did **not** reset when we dropped a row. 2 is missing!

We can reset the index and pretend like 2 was never there. The `reset_index()` function takes one keyword argument. If we don't pass this argument, `drop=True`, an extra column will get added to our DataFrame containing the old index numbers.

```
df = df.reset_index(drop=True)

df.head()
```

```
   X  Y month  day  fuel  moisture  drought_code  initial_spread_code
temp  \
0  7  5   mar  fri  86.2      26.2          94.3                  5.1
8.2
1  7  4   oct  tue  90.6      35.4         669.1                  6.7
18.0
2  8  6   mar  fri  91.7      33.3          77.5                  9.0
8.3
3  8  6   mar  sun  89.3      51.3         102.2                  9.6
11.4
4  8  6   aug  sun  92.3      85.3         488.0                 14.7
22.2

   humidity  wind  rain  area_burned
0        51   6.7   0.0          0.0
1        33   0.9   0.0          0.0
2        97   4.0   0.2          0.0
```

```
3        99   1.8   0.0          0.0
4        29   5.4   0.0          0.0
```

The `drop()` function defaults to dropping rows. If we want to drop a column, we need to add one more argument. Let's drop the "X" column:

```
df = df.drop(["X"], axis=1)

df.head()
```

```
   Y month  day  fuel  moisture  drought_code  initial_spread_code
temp  \
0  5   mar  fri  86.2      26.2          94.3                  5.1
8.2
1  4   oct  tue  90.6      35.4         669.1                  6.7
18.0
2  6   mar  fri  91.7      33.3          77.5                  9.0
8.3
3  6   mar  sun  89.3      51.3         102.2                  9.6
11.4
4  6   aug  sun  92.3      85.3         488.0                 14.7
22.2

   humidity  wind  rain  area_burned
0        51   6.7   0.0          0.0
1        33   0.9   0.0          0.0
2        97   4.0   0.2          0.0
3        99   1.8   0.0          0.0
4        29   5.4   0.0          0.0
```

data aggregation

Data aggregation means taking many data points and reducing them to one number, whether it's a count, sum, mean, or other single statistic. Here are some DataFrame method functions:

- `.count()`
- `.sum()`
- `.mean()`
- `.median()`
- `.min()`
- `.max()`
- `.unique()`
- `.nunique()`
- `.std()`
- `.var()`
- And more!

If you use a method function on the entire dataset, it will try its best to execute the method for all columns.

```
df.count()
```

```
Y                       516
month                   516
day                     516
fuel                    516
moisture                516
drought_code            516
initial_spread_code     516
temp                    516
humidity                516
wind                    516
rain                    516
area_burned             516
dtype: int64
```

```
df.min()
```

```
Y                         2
month                   apr
day                     fri
fuel                   18.7
moisture                1.1
drought_code            7.9
initial_spread_code     0.0
temp                    2.2
humidity                 15
wind                    0.4
rain                    0.0
area_burned             0.0
dtype: object
```

```
df.sum()
```

```
Y
2219
month
maroctmarmaraugaugaugsepsepsepsepaugsepsepsepm...
day
frituefrisunsunmonmontuesatsatsatfrimonwedfris...
fuel
46772.7
moisture
57277.3
drought_code
282598.1
initial_spread_code
4657.5
```

```
temp
9751.1
humidity
22864
wind
2075.8
rain
11.2
area_burned
6642.05
dtype: object
```

```
df.unique()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/tmp/ipykernel_18361/1660334235.py in ?()
----> 1 df.unique()

~/.local/lib/python3.12/site-packages/pandas/core/generic.py in ?(self, name)
   6295             and name not in self._accessors
   6296             and self._info_axis._can_hold_identifiers_and_holds_name(name)
   6297         ):
   6298             return self[name]
-> 6299         return object.__getattribute__(self, name)

AttributeError: 'DataFrame' object has no attribute 'unique'
```

Not all functions will work on the entire DataFrame. Most of the time you are interested in only a subset of the data:

```
df["day"].unique()
```

```
array(['fri', 'tue', 'sun', 'mon', 'sat', 'wed', 'thu'], dtype=object)
```

```
list(df["day"].unique())
```

```
['fri', 'tue', 'sun', 'mon', 'sat', 'wed', 'thu']
```

```
df["month"].nunique()
```

```
12
```

```
df["temp"].var()
```

```
np.float64(33.746576164672234)
```

more subsampling

Earlier, we learned how to select rows based on one condition in a column. Here we will select with multiple conditions. The syntax requires us to 1. contain each boolean in parentheses, and 2. use `& | !` instead of `and or not`.

Fires on Fridays when the temperature was over 30 Celsius:

```
df[(df["day"] == "fri") & (df["temp"] > 30)]
```

```
     Y month  day  fuel  moisture  drought_code  initial_spread_code
temp  \
491  3   aug  fri  95.9     158.0         633.6                 11.3
32.4

     humidity  wind  rain  area_burned
491        27   2.2   0.0          0.0
```

Fires in either June or July:

```
df[(df["month"] == "jun") | (df["month"] == "jul")]
```

```
     Y month  day  fuel  moisture  drought_code  initial_spread_code
temp  \
21   4   jun  sun  94.3      96.3         200.0                 56.1
21.0
39   4   jul  tue  79.5      60.6         366.7                  1.5
23.3
46   6   jul  mon  94.2      62.3         442.9                 11.0
23.0
137  9   jul  tue  85.8      48.3         313.4                  3.9
18.0
142  2   jul  sat  90.0      51.3         296.3                  8.7
16.6
149  5   jun  fri  92.5      56.4         433.3                  7.1
23.2
150  9   jul  sun  90.1      68.6         355.2                  7.2
24.8
151  4   jul  sat  90.1      51.2         424.1                  6.2
24.6
222  2   jul  fri  88.3     150.3         309.9                  6.8
13.4
284  5   jul  sun  93.9     169.7         411.8                 12.3
23.4
285  6   jul  wed  91.2     183.1         437.7                 12.5
12.6
286  4   jul  sat  91.6     104.2         474.9                  9.0
22.1
287  4   jul  sat  91.6     104.2         474.9                  9.0
24.2
```

```
288  4   jul  sat  91.6     104.2          474.9                  9.0
24.3
289  5   jul  sat  91.6     104.2          474.9                  9.0
18.7
290  4   jul  sat  91.6     104.2          474.9                  9.0
25.3
291  5   jul  fri  91.6     100.2          466.3                  6.3
22.9
292  6   jul  tue  93.1     180.4          430.8                 11.0
26.9
293  6   jul  tue  92.3      88.8          440.9                  8.5
17.1
294  5   jun  sun  93.1     180.4          430.8                 11.0
22.2
295  4   jun  sun  90.4      89.5          290.8                  6.4
14.3
296  6   jun  sun  90.4      89.5          290.8                  6.4
15.4
297  6   jun  wed  91.2     147.8          377.2                 12.7
19.6
298  5   jun  sat  53.4      71.0          233.8                  0.4
10.6
299  5   jun  mon  90.4      93.3          298.1                  7.5
20.7
300  5   jun  mon  90.4      93.3          298.1                  7.5
19.1
301  6   jun  fri  91.1      94.1          232.1                  7.1
19.2
302  6   jun  fri  91.1      94.1          232.1                  7.1
19.2
370  4   jul  wed  91.9     133.6          520.5                  8.0
14.2
379  4   jul  wed  93.7     101.3          458.8                 11.9
19.3
398  5   jun  wed  93.3      49.5          297.7                 14.0
28.0
399  5   jun  wed  93.3      49.5          297.7                 14.0
28.0
408  4   jul  tue  92.3      96.2          450.2                 12.1
23.4
411  4   jul  mon  92.3      92.1          442.1                  9.8
22.8
415  3   jul  tue  92.7     164.1          575.8                  8.9
26.3
421  6   jul  sun  88.9     263.1          795.9                  5.2
29.3
434  5   jul  sat  90.8      84.7          376.6                  5.6
23.8
442  2   jul  fri  90.7      80.9          368.3                 16.8
```

```
14.8
454  4   jul  mon  94.6     160.0          567.2                  16.7
17.9
471  3   jun  mon  88.2      96.2          229.0                   4.7
14.3
472  4   jun  sat  90.5      61.1          252.6                   9.4
24.5
473  3   jun  thu  93.0     103.8          316.7                  10.8
26.4
474  5   jun  thu  93.7     121.7          350.2                  18.0
22.7
475  3   jul  thu  93.5      85.3          395.0                   9.9
27.2
476  3   jul  sun  93.7     101.3          423.4                  14.7
26.1
477  4   jul  sun  93.7     101.3          423.4                  14.7
18.2
478  4   jul  mon  89.2     103.9          431.6                   6.4
22.6
479  9   jul  thu  93.2     114.4          560.0                   9.5
30.2
480  3   jul  thu  93.2     114.4          560.0                   9.5
30.2

     humidity  wind  rain  area_burned
21         44   4.5   0.0         0.00
39         37   3.1   0.0         0.00
46         36   3.1   0.0         0.00
137        42   2.7   0.0         0.36
142        53   5.4   0.0         0.71
149        39   5.4   0.0         1.19
150        29   2.2   0.0         1.36
151        43   1.8   0.0         1.43
222        79   3.6   0.0        37.02
284        40   6.3   0.0         0.00
285        90   7.6   0.2         0.00
286        49   2.7   0.0         0.00
287        32   1.8   0.0         0.00
288        30   1.8   0.0         0.00
289        53   1.8   0.0         0.00
290        39   0.9   0.0         8.00
291        40   1.3   0.0         2.64
292        28   5.4   0.0        86.45
293        67   3.6   0.0         6.57
294        48   1.3   0.0         0.00
295        46   1.8   0.0         0.90
296        45   2.2   0.0         0.00
297        43   4.9   0.0         0.00
298        90   2.7   0.0         0.00
```

```
299         25   4.9   0.0          0.00
300         39   5.4   0.0          3.52
301         38   4.5   0.0          0.00
302         38   4.5   0.0          0.00
370         58   4.0   0.0          0.00
379         39   7.2   0.0          7.73
398         34   4.5   0.0          0.00
399         34   4.5   0.0          8.16
408         31   5.4   0.0          0.00
411         27   4.5   0.0          1.63
415         39   3.1   0.0          7.02
421         27   3.6   0.0          6.30
434         51   1.8   0.0          0.00
442         78   8.0   0.0          0.00
454         48   2.7   0.0          0.00
471         79   4.0   0.0          1.94
472         50   3.1   0.0         70.32
473         35   2.7   0.0         10.08
474         40   9.4   0.0          3.19
475         28   1.3   0.0          1.76
476         45   4.0   0.0          7.36
477         82   4.5   0.0          2.21
478         57   4.9   0.0        278.53
479         25   4.5   0.0          2.75
480         22   4.9   0.0          0.00
```

groupby

Often, you will want to calculate the statistics for a particular subgroup of a data column. The pandas `groupby` function allows us to group our data on the values in a column or column to look at summary measures for records sharing the same values.

For example, let's say we want to ask if more fires happen on certain days of the week. This code will tell you the count for every column in the DataFrame except the column that you are using to group your data (i.e. "day").

```
df.groupby("day").count()
```

```
       Y  month  fuel  moisture  drought_code  initial_spread_code
temp  \
day

fri   85     85    85        85            85                   85
85
mon   74     74    74        74            74                   74
74
sat   83     83    83        83            83                   83
83
sun   95     95    95        95            95                   95
```

```
95
thu  61     61    61        61            61                    61
61
tue  64     64    64        64            64                    64
64
wed  54     54    54        54            54                    54
54

     humidity  wind  rain  area_burned
day
fri        85    85    85           85
mon        74    74    74           74
sat        83    83    83           83
sun        95    95    95           95
thu        61    61    61           61
tue        64    64    64           64
wed        54    54    54           54
```

It looks like weekends are worse than weekdays. (The next section will show you how to sort the rows.)

If you only want to see the mean for one column in the DataFrame, you can add on the subsampling techniques we learned in part one of this lesson. With this code I will ask, What is the mean area burned on each day of the week?

```
df.groupby("day")[["area_burned"]].mean()
```

```
     area_burned
day
fri     5.261647
mon     9.547703
sat    25.841687
sun    10.104526
thu    16.345902
tue    12.621719
wed    10.714815
```

So Saturday fires are also the most destructive fires.

We can also add some other functions to the end of our code, like round:

```
df.groupby("day")[["area_burned"]].mean().round(2)
```

```
     area_burned
day
fri         5.26
mon         9.55
sat        25.84
sun        10.10
thu        16.35
```

```
tue         12.62
wed         10.71
```

Sort by Sunday-Saturday:

```
week_day = ["sun", "mon", "tue", "wed", "thu", "fri", "sat"]
area_burned_week = df.groupby("day")
[["area_burned"]].mean().round(2).reindex(week_day)
area_burned_week
```

```
     area_burned
day
sun         10.10
mon          9.55
tue         12.62
wed         10.71
thu         16.35
fri          5.26
sat         25.84
```

sorting a DataFrame

There are two functions for sorting your DataFrame.

If you want to sort by the index numbers, or if you want to sort by the column names (alphabetically), you use `sort_index`. It can take two arguments: the axis to sort by (row or column) and the order (reverse or not):

The default arguments are to sort by row index with 0 at the top, which is how we've already been viewing the data:

```
df.sort_index()
```

```
     Y month  day  fuel  moisture  drought_code  initial_spread_code
temp  \
0    5   mar  fri  86.2      26.2          94.3                  5.1
8.2
1    4   oct  tue  90.6      35.4         669.1                  6.7
18.0
2    6   mar  fri  91.7      33.3          77.5                  9.0
8.3
3    6   mar  sun  89.3      51.3         102.2                  9.6
11.4
4    6   aug  sun  92.3      85.3         488.0                 14.7
22.2
..  ..   ...  ...   ...       ...           ...                  ...
...
511  3   aug  sun  81.6      56.7         665.6                  1.9
27.8
```

```
512  4   aug  sun  81.6      56.7         665.6                    1.9
21.9
513  4   aug  sun  81.6      56.7         665.6                    1.9
21.2
514  4   aug  sat  94.4     146.0         614.7                   11.3
25.6
515  3   nov  tue  79.5       3.0         106.7                    1.1
11.8

     humidity  wind  rain  area_burned
0          51   6.7   0.0         0.00
1          33   0.9   0.0         0.00
2          97   4.0   0.2         0.00
3          99   1.8   0.0         0.00
4          29   5.4   0.0         0.00
..        ...   ...   ...          ...
511        32   2.7   0.0         6.44
512        71   5.8   0.0        54.29
513        70   6.7   0.0        11.16
514        42   4.0   0.0         0.00
515        31   4.5   0.0         0.00

[516 rows x 12 columns]
```

Let's try more arguments:

```
df.sort_index(ascending=False)

     Y month  day  fuel  moisture  drought_code  initial_spread_code
temp  \
515  3   nov  tue  79.5       3.0         106.7                    1.1
11.8
514  4   aug  sat  94.4     146.0         614.7                   11.3
25.6
513  4   aug  sun  81.6      56.7         665.6                    1.9
21.2
512  4   aug  sun  81.6      56.7         665.6                    1.9
21.9
511  3   aug  sun  81.6      56.7         665.6                    1.9
27.8
..  ..   ...  ...   ...       ...           ...                    ...
...
4    6   aug  sun  92.3      85.3         488.0                   14.7
22.2
3    6   mar  sun  89.3      51.3         102.2                    9.6
11.4
2    6   mar  fri  91.7      33.3          77.5                    9.0
8.3
1    4   oct  tue  90.6      35.4         669.1                    6.7
```

```
18.0
0    5   mar  fri  86.2      26.2          94.3                  5.1
8.2

     humidity  wind  rain  area_burned
515        31   4.5   0.0         0.00
514        42   4.0   0.0         0.00
513        70   6.7   0.0        11.16
512        71   5.8   0.0        54.29
511        32   2.7   0.0         6.44
..        ...   ...   ...          ...
4          29   5.4   0.0         0.00
3          99   1.8   0.0         0.00
2          97   4.0   0.2         0.00
1          33   0.9   0.0         0.00
0          51   6.7   0.0         0.00

[516 rows x 12 columns]
```

```
df.sort_index(axis=1) # The axis along which to sort. The value 0
identifies the rows, and 1 identifies the columns.
```

```
     Y  area_burned  day  drought_code  fuel  humidity
initial_spread_code  \
0    5         0.00  fri          94.3  86.2        51
5.1
1    4         0.00  tue         669.1  90.6        33
6.7
2    6         0.00  fri          77.5  91.7        97
9.0
3    6         0.00  sun         102.2  89.3        99
9.6
4    6         0.00  sun         488.0  92.3        29
14.7
..  ..          ...  ...           ...   ...       ...
...
511  3         6.44  sun         665.6  81.6        32
1.9
512  4        54.29  sun         665.6  81.6        71
1.9
513  4        11.16  sun         665.6  81.6        70
1.9
514  4         0.00  sat         614.7  94.4        42
11.3
515  3         0.00  tue         106.7  79.5        31
1.1

     moisture month  rain  temp  wind
0        26.2   mar   0.0   8.2   6.7
1        35.4   oct   0.0  18.0   0.9
```

```
2        33.3   mar   0.2   8.3   4.0
3        51.3   mar   0.0  11.4   1.8
4        85.3   aug   0.0  22.2   5.4
..        ...   ...   ...   ...   ...
511      56.7   aug   0.0  27.8   2.7
512      56.7   aug   0.0  21.9   5.8
513      56.7   aug   0.0  21.2   6.7
514     146.0   aug   0.0  25.6   4.0
515       3.0   nov   0.0  11.8   4.5

[516 rows x 12 columns]
```

```
df.sort_index(axis=1, ascending=False)
```

```
     wind  temp  rain month  moisture  initial_spread_code  humidity
fuel  \
0     6.7   8.2   0.0   mar      26.2                  5.1        51
86.2
1     0.9  18.0   0.0   oct      35.4                  6.7        33
90.6
2     4.0   8.3   0.2   mar      33.3                  9.0        97
91.7
3     1.8  11.4   0.0   mar      51.3                  9.6        99
89.3
4     5.4  22.2   0.0   aug      85.3                 14.7        29
92.3
..    ...   ...   ...   ...       ...                  ...       ...
...
511   2.7  27.8   0.0   aug      56.7                  1.9        32
81.6
512   5.8  21.9   0.0   aug      56.7                  1.9        71
81.6
513   6.7  21.2   0.0   aug      56.7                  1.9        70
81.6
514   4.0  25.6   0.0   aug     146.0                 11.3        42
94.4
515   4.5  11.8   0.0   nov       3.0                  1.1        31
79.5

     drought_code  day  area_burned  Y
0            94.3  fri         0.00  5
1           669.1  tue         0.00  4
2            77.5  fri         0.00  6
3           102.2  sun         0.00  6
4           488.0  sun         0.00  6
..            ...  ...          ... ..
511         665.6  sun         6.44  3
512         665.6  sun        54.29  4
513         665.6  sun        11.16  4
514         614.7  sat         0.00  4
```

```
515         106.7  tue         0.00  3

[516 rows x 12 columns]
```

The second sort function, `sort_values()`, will sort the frame by the data in a column:

```
df.sort_values(by="area_burned")

     Y month  day  fuel  moisture  drought_code  initial_spread_code
temp  \
371  5   aug  sun  92.0     203.2         664.5                  8.1
10.4
340  6   sep  mon  91.9     111.7         770.3                  6.5
15.7
451  4   aug  mon  91.5     238.2         730.6                  7.5
17.7
469  4   apr  sun  91.0      14.6          25.6                 12.3
17.6
499  6   aug  tue  96.1     181.1         671.2                 14.3
21.6
..  ..   ...  ...   ...       ...           ...                  ...
...
235  2   sep  sat  92.5     121.1         674.4                  8.6
18.2
236  2   sep  tue  91.0     129.5         692.6                  7.0
18.8
478  4   jul  mon  89.2     103.9         431.6                  6.4
22.6
414  6   aug  thu  94.8     222.4         698.6                 13.9
27.5
237  5   sep  sat  92.5     121.1         674.4                  8.6
25.1

     humidity  wind  rain  area_burned
371        75   0.9   0.0         0.00
340        51   2.2   0.0         0.00
451        65   4.0   0.0         0.00
469        27   5.8   0.0         0.00
499        65   4.9   0.8         0.00
..        ...   ...   ...          ...
235        46   1.8   0.0       200.94
236        40   2.2   0.0       212.88
478        57   4.9   0.0       278.53
414        27   4.9   0.0       746.28
237        27   4.0   0.0      1090.84

[516 rows x 12 columns]

df.sort_values(by="day")
```

```
     Y month  day  fuel  moisture  drought_code  initial_spread_code
temp  \
0    5   mar  fri  86.2      26.2          94.3                  5.1
8.2
352  4   sep  fri  92.1      99.0         745.3                  9.6
19.8
104  5   mar  fri  85.9      19.5          57.3                  2.8
12.7
354  4   sep  fri  92.1      99.0         745.3                  9.6
20.8
355  4   sep  fri  92.1      99.0         745.3                  9.6
20.8
..  ..   ...  ...   ...       ...           ...                  ...
...
51   3   aug  wed  92.1     111.2         654.1                  9.6
20.4
52   3   aug  wed  92.1     111.2         654.1                  9.6
20.4
379  4   jul  wed  93.7     101.3         458.8                 11.9
19.3
44   6   sep  wed  94.3      85.1         692.3                 15.9
25.9
420  4   aug  wed  95.2     217.7         690.0                 18.0
30.8

     humidity  wind  rain  area_burned
0          51   6.7   0.0         0.00
352        47   2.7   0.0         1.72
104        52   6.3   0.0         0.00
354        35   4.9   0.0        13.06
355        35   4.9   0.0         1.26
..        ...   ...   ...          ...
51         42   4.9   0.0         0.00
52         42   4.9   0.0         0.00
379        39   7.2   0.0         7.73
44         24   4.0   0.0         0.00
420        19   4.5   0.0         0.00

[516 rows x 12 columns]
```

FYI, you can sort days using the "correct" order by Lambdas. Lambdas are one line functions. They are also known as anonymous functions. lambda argument: manipulate(argument)`

```
add = lambda x, y: x + y

print(add(3, 5))
# Output: 8
```

```
8
```

```
week_day = {'sun': 0, 'mon': 1, 'tue': 2, "wed": 3, "thu": 4, "fri":
5, 'sat':6}
df.sort_values(by = "day", key = lambda x: x.map(week_day))
#
```

```
     Y month  day  fuel  moisture  drought_code  initial_spread_code
temp  \
511  3   aug  sun  81.6      56.7         665.6                  1.9
27.8
510  6   aug  sun  81.6      56.7         665.6                  1.9
27.8
4    6   aug  sun  92.3      85.3         488.0                 14.7
22.2
3    6   mar  sun  89.3      51.3         102.2                  9.6
11.4
512  4   aug  sun  81.6      56.7         665.6                  1.9
21.9
..  ..   ...  ...   ...       ...           ...                  ...
...
8    5   sep  sat  92.5      88.0         698.6                  7.1
22.8
10   5   sep  sat  92.8      73.2         713.0                 22.6
19.3
9    5   sep  sat  92.5      88.0         698.6                  7.1
17.8
18   4   apr  sat  86.3      27.4          97.1                  5.1
9.3
514  4   aug  sat  94.4     146.0         614.7                 11.3
25.6

     humidity  wind  rain  area_burned
511        32   2.7   0.0         6.44
510        35   2.7   0.0         0.00
4          29   5.4   0.0         0.00
3          99   1.8   0.0         0.00
512        71   5.8   0.0        54.29
..        ...   ...   ...          ...
8          40   4.0   0.0         0.00
10         38   4.0   0.0         0.00
9          51   7.2   0.0         0.00
18         44   4.5   0.0         0.00
514        42   4.0   0.0         0.00

[516 rows x 12 columns]
```

saving your changed DataFrame

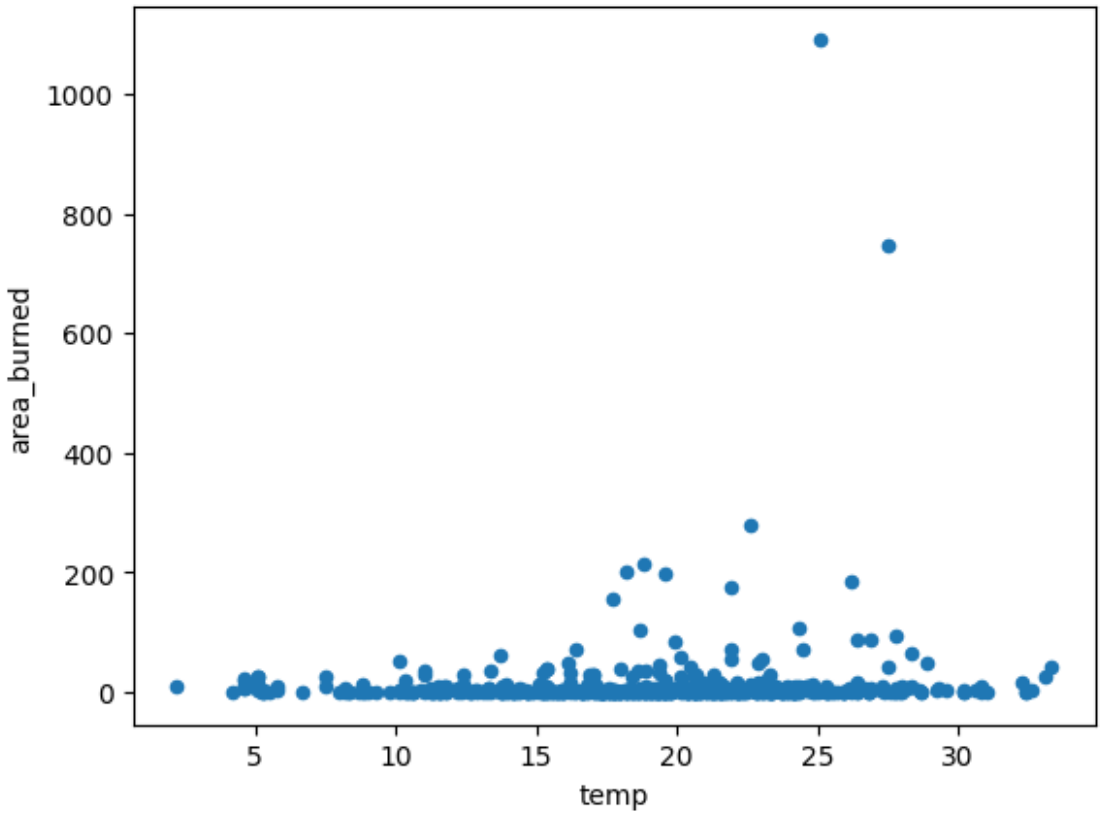
```
new_filename = "fire_changed.csv"

df.to_csv(new_filename)
```

basic plotting

Other python packages can help you make beautiful visualizations of your data. With Pandas, you can make several simple plots, including histograms, box and whisker plots, bar graphs, scatter plots, and pie charts.

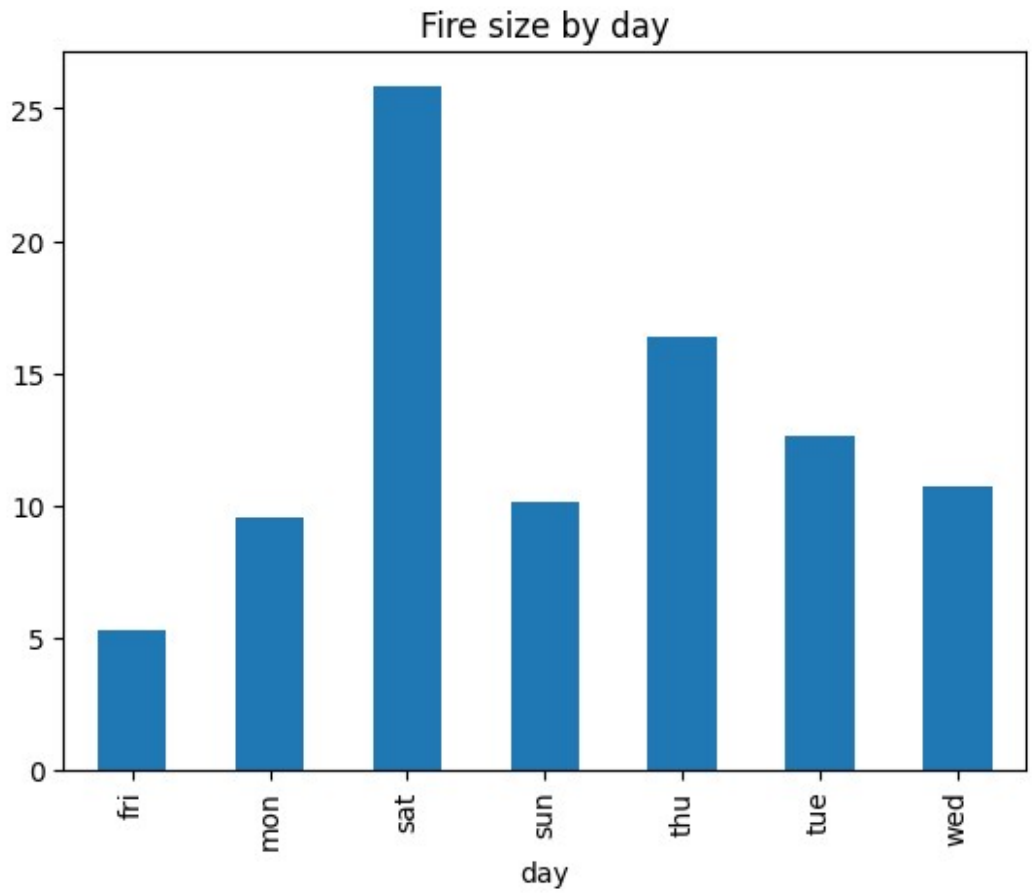
We will first make a simple scatter plot of the columns `temp` and `area_burned`. We use the `plot()` function. At the least, we need to include three arguments: the kind of plot to make, the data to use for the x axis, and the data to use for the y axis.

```
temp_scatter = df.plot(kind="scatter", x="temp", y="area_burned")
```



Let's make a bar graph of the mean area burned for the days of the week. First we group by day, then subsample only the area burned column, then calculate the means, and finally plot the means. I also added an argument for "title" to this plot.

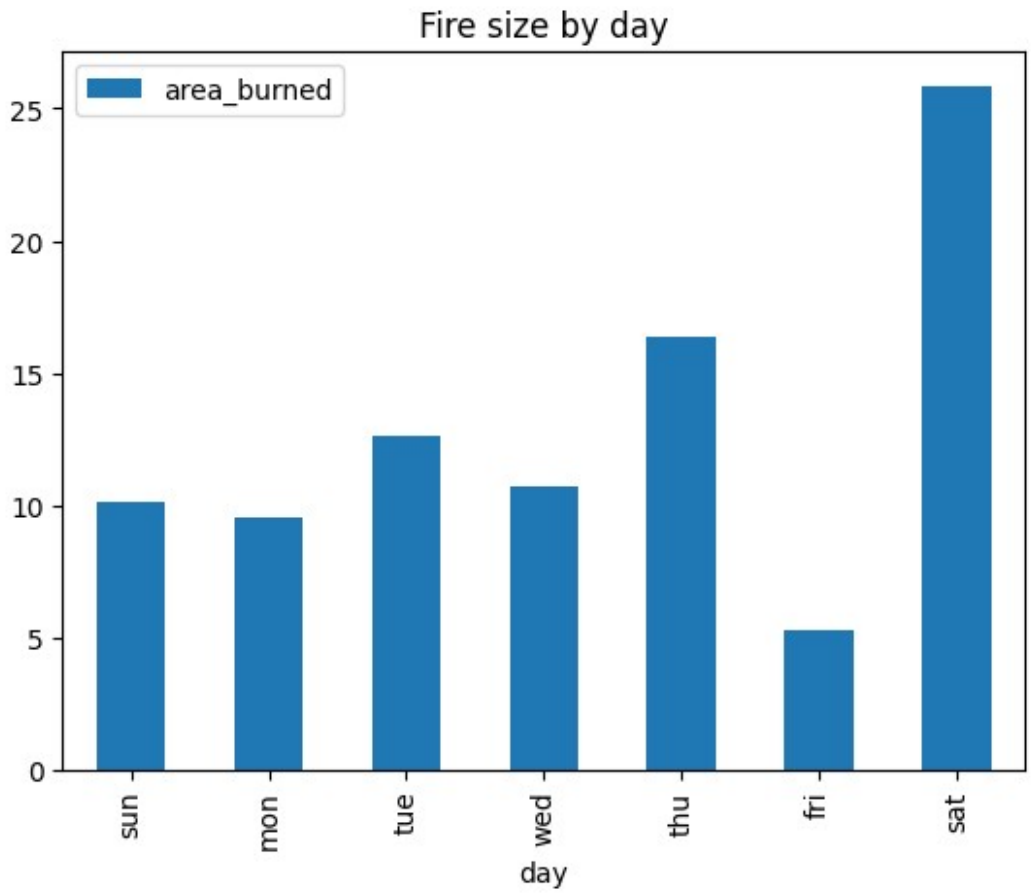
```
day_bar = df.groupby("day")["area_burned"].mean().plot(kind = 'bar',
title = "Fire size by day")
```




```
week_day = {'sun': 0, 'mon': 1, 'tue': 2, "wed": 3, "thu": 4, "fri":
5, 'sat':6}
area_mean_day = df.groupby("day")["area_burned"].mean()
area_mean_day
```

```
day
fri     5.261647
mon     9.547703
sat    25.841687
sun    10.104526
thu    16.345902
tue    12.621719
wed    10.714815
Name: area_burned, dtype: float64
```

```
area_mean_day = pd.DataFrame(area_mean_day).sort_values(by = 'day',
key = lambda x: x.map(week_day))
day_bar = area_mean_day.plot(kind = 'bar', title = "Fire size by day")
```



loading other types of files

We can open a tab-separated file using the same function we used to open a csv. We just have to pass another argument to tell it that the delimiter is a tab instead of the default (comma). This dataset contains rankings of profressional racing pigeons.

```
pigeon_df = pd.read_csv("pigeonRacing.txt", delimiter="\t") # tab
character

pigeon_df.head()
```

```
   Position Avg Unirate                Name      Racing Pigeon Color Sex
\
0         1       0.26%       Dean Schultz     751 AU 18 PURP    BB   H

1         2       1.08%        Dick Fassio     9027 AU 19 SLI   BBAR   H

2         3       1.42%        Gary Mosher     32826 AU 17 AA    BKC   H

3         4       2.21%  Todd Bartholomew   35624 AU 17 JEDD    BC   H

4         5       2.61%    Dustin Maxfield     3322 AU 17 OGN    BB   C
```

```
  Qualifying Race Miles  Average Birdage
0              469, 469              612
1              579, 500              139
2              494, 539              103
3              547, 468              226
4              462, 462              171
```

We will use a different function to open an Excel file. This file has information about animals and has two sheets within the excel file. We will first load sheet 1 and then sheet 2.

```
!pip install openpyxl
import openpyxl
zoo_df = pd.read_excel('zoo.xlsx', sheet_name=0, header=0)
```

```
Requirement already satisfied: openpyxl in
/usr/local/python/3.12.1/lib/python3.12/site-packages (3.1.5)
Requirement already satisfied: et-xmlfile in
/usr/local/python/3.12.1/lib/python3.12/site-packages (from openpyxl)
(2.0.0)

[notice] A new release of pip is available: 24.3.1 -> 25.0.1
[notice] To update, run: python3 -m pip install --upgrade pip
```

```
!pip install openpyxl
```

```
Requirement already satisfied: openpyxl in
/usr/local/python/3.12.1/lib/python3.12/site-packages (3.1.5)
Requirement already satisfied: et-xmlfile in
/usr/local/python/3.12.1/lib/python3.12/site-packages (from openpyxl)
(2.0.0)

[notice] A new release of pip is available: 24.3.1 -> 25.0.1
[notice] To update, run: python3 -m pip install --upgrade pip
```

```
zoo_df = pd.read_excel('zoo.xlsx', sheet_name=0, header=0)
zoo_df.head()
```

```
     animal  hair  feathers  eggs  milk  airbourne  aquatic  predator
\
0  aardvark     1         0     0     1          0        0         1

1  antelope     1         0     0     1          0        0         0

2      bass     0         0     1     0          0        1         1

3      bear     1         0     0     1          0        0         1

4      boar     1         0     0     1          0        0         1
```

```
   toothed  backbone  breathes  venomous  fins  legs  tail  domestic
catsize  \
0        1         1         1         0     0     4     0         0
1
1        1         1         1         0     0     4     1         0
1
2        1         1         0         0     1     0     1         0
0
3        1         1         1         0     0     4     0         0
1
4        1         1         1         0     0     4     1         0
1

   type
0     1
1     1
2     4
3     1
4     1
```

```
zoo_class_df = pd.read_excel("zoo.xlsx", sheet_name=1)
```

```
zoo_class_df.head()
```

```
   Unnamed: 0      class
0           1     mammal
1           2       bird
2           3    reptile
3           4       fish
4           5  amphibian
```

More on Grouping, Plotting, and Merging

Let's load the speed camera dataset and ask which camera locations or days of the week have produced the most violations.

```
df = pd.read_csv("Speed_Camera_Violations.csv")
df.head()
```

```
           ADDRESS CAMERA ID VIOLATION DATE  VIOLATIONS  X COORDINATE
\
0   7738 S WESTERN    CHI065       7/8/2014          65           NaN

1  1111 N HUMBOLDT    CHI010      7/16/2014          56           NaN

2   5520 S WESTERN    CHI069       7/8/2014          10           NaN

3  1111 N HUMBOLDT    CHI010      7/26/2014         101           NaN

4  1111 N HUMBOLDT    CHI010      7/27/2014          92           NaN
```

```
   Y COORDINATE  LATITUDE  LONGITUDE LOCATION
0           NaN       NaN        NaN      NaN
1           NaN       NaN        NaN      NaN
2           NaN       NaN        NaN      NaN
3           NaN       NaN        NaN      NaN
4           NaN       NaN        NaN      NaN
```

Now that the data is loaded, let's find the 10 locations with the most total violations recorded.

To do this, we need to group by the ADDRESS column, then examine the VIOLATIONS column of the resulting grouped dataframe.

```
# first let's group by address and look at descriptive statistics for the first 10 records
df.groupby(["ADDRESS"])["VIOLATIONS"].describe().head(10)
```

```
                        count        mean        std  min    25%   50%  \
75%  \
ADDRESS

10318 S INDIANAPOLIS  1084.0  99.974170  39.412342  4.0  72.00  96.0
120.25
1110 S PULASKI RD      499.0  22.412826  15.078174  1.0  11.00  18.0
31.00
1111 N HUMBOLDT       1096.0  58.250000  18.067744  6.0  46.00  57.0
70.00
11144 S VINCENNES      526.0  19.612167  10.058579  1.0  12.25  18.0
25.00
11153 S VINCENNES      517.0   9.698259   5.182134  1.0   6.00   9.0
13.00
1117 S PULASKI RD      492.0  19.483740  12.954320  1.0  10.00  17.0
27.00
1142 W IRVING PARK    1091.0  84.210816  29.672455  1.0  64.00  81.0
101.00
115 N OGDEN           1094.0  41.311700  35.887579  1.0  13.00  28.0
61.00
1226 N WESTERN AVE     531.0  18.617702  15.435526  1.0   6.00  14.0
28.00
1229 N WESTERN AVE     530.0  39.500000  21.222848  3.0  24.00  34.0
51.75

                         max
ADDRESS
10318 S INDIANAPOLIS  259.0
1110 S PULASKI RD      83.0
1111 N HUMBOLDT       117.0
11144 S VINCENNES      75.0
11153 S VINCENNES      31.0
```

```
1117 S PULASKI RD      61.0
1142 W IRVING PARK    248.0
115 N OGDEN           197.0
1226 N WESTERN AVE     88.0
1229 N WESTERN AVE    119.0
```

The above records aren't sorted in any meaningful way, but the first thing to note is that the Index is no longer just an integer, it is now the Address. This is because the `groupby` method returns a special object with a new index made up of the values of the column being grouped on.

We can still use the `loc` indexer with this new grouped object to, for example, find the count for a given address:

```
# `count` returns the number of rows for this address, **not** the
total violation count.
# i.e., this tells us the number of observation (in case of our
example data, Speed_Camera_Violations,
# this corresponds to the number of different days with at least one
violation).
df.groupby(["ADDRESS"])["VIOLATIONS"].count().loc["19 W CHICAGO AVE"]

np.int64(432)

# to get the total violation count, we want the `sum` method:
df.groupby(["ADDRESS"])["VIOLATIONS"].sum().loc["19 W CHICAGO AVE"]

np.int64(1618)

# Now let's get the top 10 camera locations by total violation count:
df.groupby(["ADDRESS"])
["VIOLATIONS"].sum().sort_values(ascending=False).head(10)

ADDRESS
4909 N CICERO AVE       220704
445 W 127TH             169337
2900 W OGDEN            139183
4124 W FOSTER AVE       127071
10318 S INDIANAPOLIS    108372
2705 W IRVING PARK      107599
1142 W IRVING PARK       91874
536 E MORGAN DR          82331
5816 W JACKSON           80174
4831 W LAWRENCE AVE      69538
Name: VIOLATIONS, dtype: int64
```

It's possible that some locations just have more observations (violation days) than others, so a more meaningful measure is probably the mean violation count per observation. To get this we just need to use the `mean` function rather than `sum`.

```
df.groupby(["ADDRESS"])
["VIOLATIONS"].mean().sort_values(ascending=False).head(10)
```

```
ADDRESS
4909 N CICERO AVE       226.595483
445 W 127TH             154.645662
2900 W OGDEN            126.876026
4124 W FOSTER AVE       120.332386
10318 S INDIANAPOLIS     99.974170
2705 W IRVING PARK       99.078269
215 E 63RD ST            84.689008
1142 W IRVING PARK       84.210816
536 E MORGAN DR          75.188128
2549 W ADDISON           73.668488
Name: VIOLATIONS, dtype: float64
```

How about days of the week? *When* are people most likely to be caught speeding?

The simplest way to do this is to create a new weekday column and group on that.

```
# datetime series have a special `dt` property that exposes the
date/time-specific functionality.
# In this case, dayofweek is a 0-based index where 0 = Monday, 6 =
Sunday.
df["VIOLATION DATE"] = pd.to_datetime(df["VIOLATION DATE"],
format="%m/%d/%Y")
df["VIOLATION DATE"].dt.dayofweek.head()
```

```
0    1
1    2
2    1
3    5
4    6
Name: VIOLATION DATE, dtype: int32
```

```
df["DAY OF WEEK"] = df["VIOLATION DATE"].dt.dayofweek
df.groupby(["DAY OF WEEK"])["VIOLATIONS"].mean()
```

```
DAY OF WEEK
0    27.446626
1    26.745172
2    27.110899
3    28.371709
4    30.219523
5    42.380211
6    41.233922
Name: VIOLATIONS, dtype: float64
```

Plotting

It's not easy to understand at a glance the distribution of speeding violations by day of the week above, so let's produce a simple plot to visualize and help understand it.

Pandas has some basic plotting functions, but I prefer how it interacts with a different visualization package called Seaborn

If you do not have seaborn, you can use pip to install it pip install seaborn

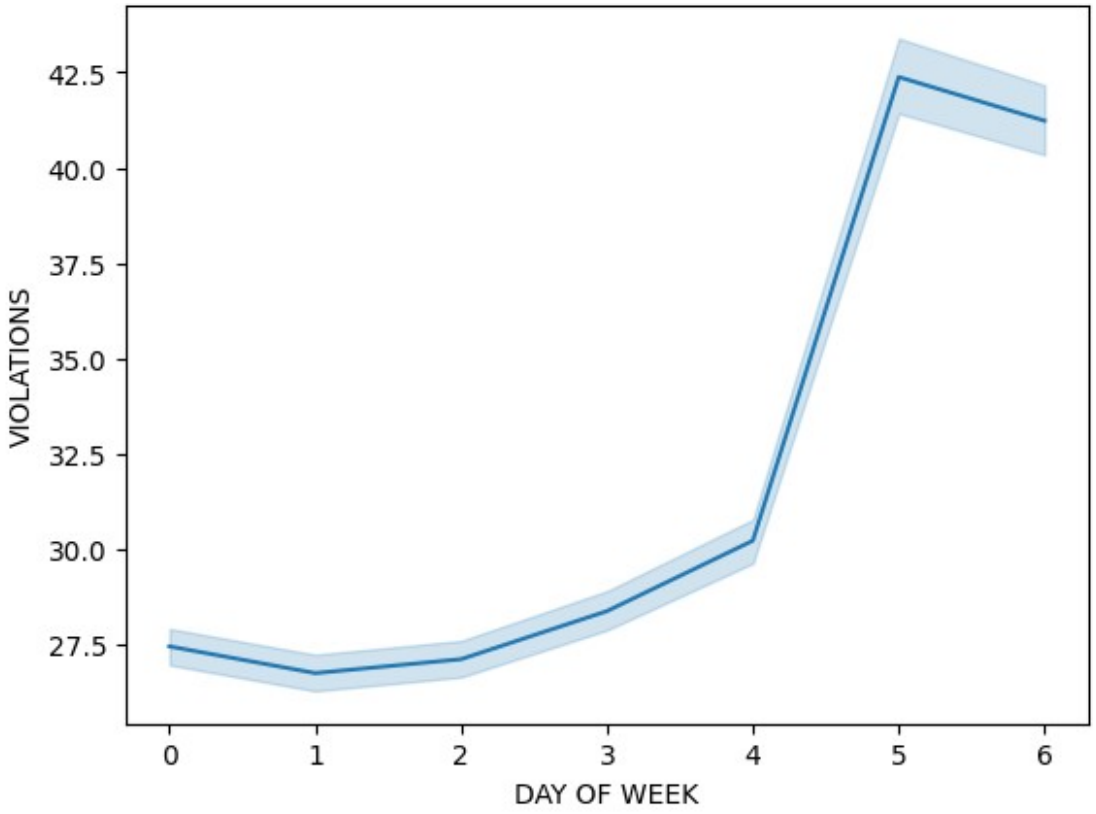
```
!pip install seaborn
```

```
Requirement already satisfied: seaborn in
/home/codespace/.local/lib/python3.12/site-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in
/home/codespace/.local/lib/python3.12/site-packages (from seaborn)
(2.2.0)
Requirement already satisfied: pandas>=1.2 in
/home/codespace/.local/lib/python3.12/site-packages (from seaborn)
(2.2.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in
/home/codespace/.local/lib/python3.12/site-packages (from seaborn)
(3.9.3)
Requirement already satisfied: contourpy>=1.0.1 in
/home/codespace/.local/lib/python3.12/site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn) (1.3.1)
Requirement already satisfied: cycler>=0.10 in
/home/codespace/.local/lib/python3.12/site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/home/codespace/.local/lib/python3.12/site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn) (4.55.3)
Requirement already satisfied: kiwisolver>=1.3.1 in
/home/codespace/.local/lib/python3.12/site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn) (1.4.7)
Requirement already satisfied: packaging>=20.0 in
/home/codespace/.local/lib/python3.12/site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn) (24.2)
Requirement already satisfied: pillow>=8 in
/home/codespace/.local/lib/python3.12/site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/home/codespace/.local/lib/python3.12/site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in
/home/codespace/.local/lib/python3.12/site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/home/codespace/.local/lib/python3.12/site-packages (from pandas>=1.2-
>seaborn) (2024.2)
```

```
Requirement already satisfied: tzdata>=2022.7 in
/home/codespace/.local/lib/python3.12/site-packages (from pandas>=1.2-
>seaborn) (2024.2)
Requirement already satisfied: six>=1.5 in
/home/codespace/.local/lib/python3.12/site-packages (from python-
dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.17.0)

[notice] A new release of pip is available: 24.3.1 -> 25.0.1
[notice] To update, run: python3 -m pip install --upgrade pip
```

```
import seaborn as sns #The cannonical way to import seaborn
# Why sns? --- see here:
https://stackoverflow.com/questions/41499857/seaborn-why-import-as-sns

# The beauty of pandas with seaborn is how cleanly they interact with
each other

sns.lineplot(x = 'DAY OF WEEK', y = 'VIOLATIONS', data = df)
```

```
<Axes: xlabel='DAY OF WEEK', ylabel='VIOLATIONS'>
```



What if time of year is a factor here? Seaborn has a wonderful feature called hue, which allows for a quick comparison of different types of data in one graph

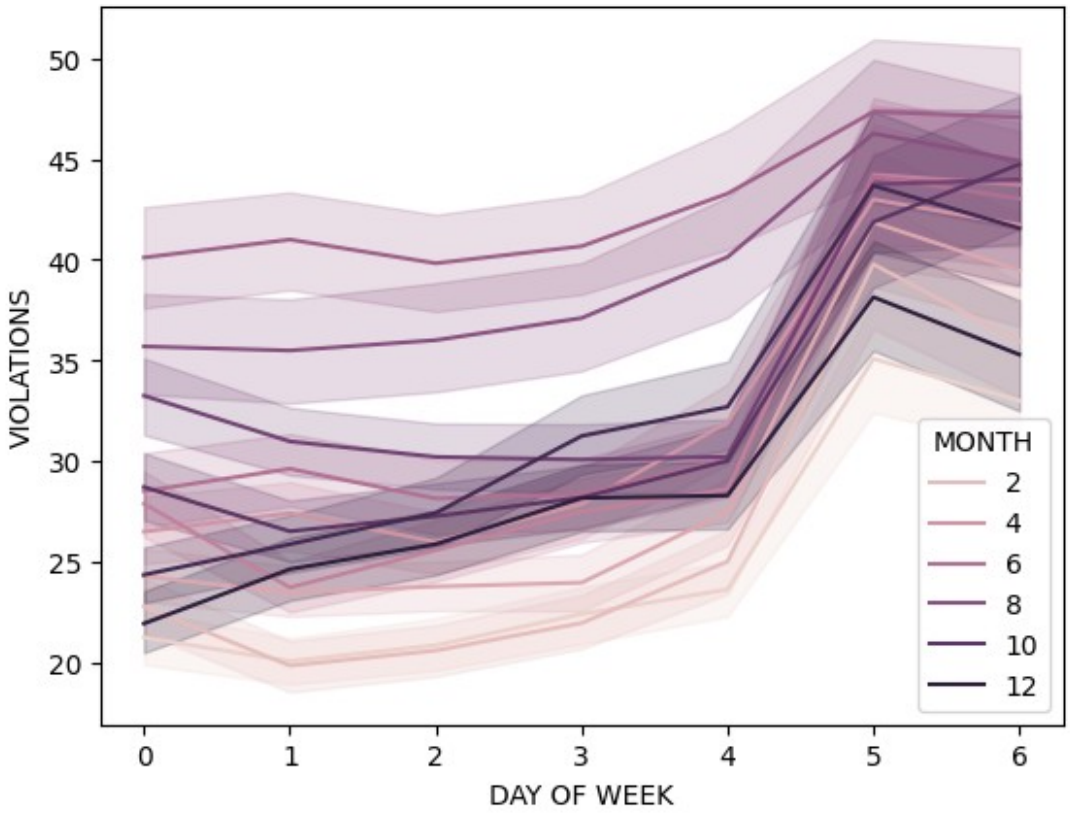
```
#first lets create a month column
df['MONTH'] = df['VIOLATION DATE'].dt.month
# The dt.month attribute returns a NumPy array containing the month of
the DateTime in the underlying data of the given Series object.
# The month as January=1, December=12.

df.head()
```

	ADDRESS	CAMERA ID	VIOLATION DATE	VIOLATIONS	X COORDINATE
0	7738 S WESTERN	CHI065	2014-07-08	65	NaN
1	1111 N HUMBOLDT	CHI010	2014-07-16	56	NaN
2	5520 S WESTERN	CHI069	2014-07-08	10	NaN
3	1111 N HUMBOLDT	CHI010	2014-07-26	101	NaN
4	1111 N HUMBOLDT	CHI010	2014-07-27	92	NaN

	Y COORDINATE	LATITUDE	LONGITUDE	LOCATION	DAY OF WEEK	MONTH
0	NaN	NaN	NaN	NaN	1	7
1	NaN	NaN	NaN	NaN	2	7
2	NaN	NaN	NaN	NaN	1	7
3	NaN	NaN	NaN	NaN	5	7
4	NaN	NaN	NaN	NaN	6	7

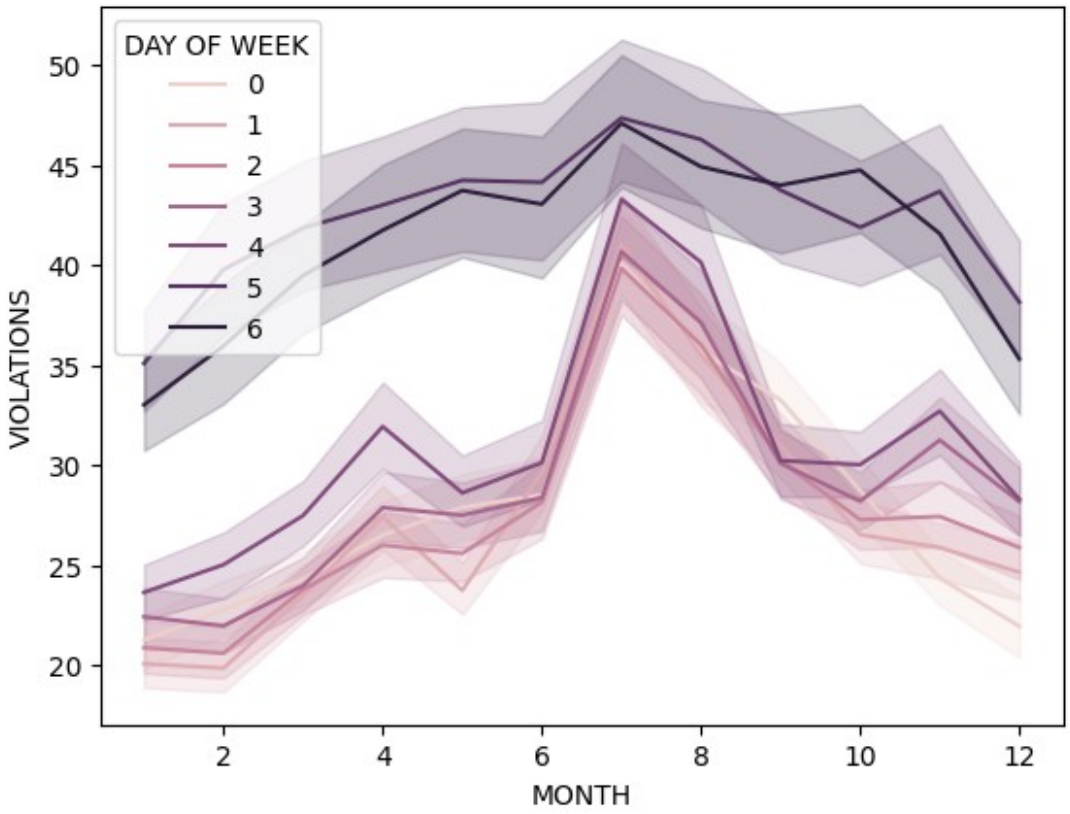
```
#then let's recreate that same plot but with the months separated out
sns.lineplot(x = 'DAY OF WEEK', y = 'VIOLATIONS', hue = 'MONTH', data
= df)
# Change color? see here:
https://seaborn.pydata.org/tutorial/color_palettes.html
```

```
<Axes: xlabel='DAY OF WEEK', ylabel='VIOLATIONS'>
```




```
#that's a little chaotic, lets try flipping the hue with the x axis

sns.lineplot(x = 'MONTH', y = 'VIOLATIONS', hue = 'DAY OF WEEK', data
= df)
```

```
<Axes: xlabel='MONTH', ylabel='VIOLATIONS'>
```



Two observations here. First is that violations are much more likely to occur on saturday and sunday, regardless of the time of year. The second is that that violations are much more likely to occur during the summer months.