EDA_p1

September 15, 2021

0.1 Mud card

- **When we want to find hyper parameters, we use embedded loops to go through it, but the problem I have is that how that related to cross-validation methods, since right now, we have a concrete training set and validation sets in order to get a good hyper parameters. From my personal knowledge, cross-validation method would change training set a few times, so I confused about what is the connections between them**
  - what you refer to is called k-fold cross validation
  - that's just one way to do CV
  - we will cover a handful of techniques to do CV/tune hyperparameters
- **In quiz 3, the option "Underfitting means that the model performs similarly on the training and validation sets", why is it incorrect? In the example you have shown in class, underfitting means that models perform poorly on both the training and validation sets.**
- **for the quiz 3, underfit means they both perform poorly, why they can't be said be similar?**
  - the best model can perform similarly on the training and validation sets, and that model performance is not poor, it's optimal
  - you need to see the whole training and validation curves as a function of the hyperparameter to decide when the model performance is poor
- **For a certain model, such as gradient descent model, I usually change the step size manually, how can I choose the step size in a "smarter" way?**
  - gradient descent is a numerical algorithm used to find a local or global minimum of a function
  - it is used in ML algorithms to find optimal model parameters
  - but it is not a ML model
- **Are there any resources we can use to practice the things we're learning in class that aren't graded? So like some practice exercises?**
  - kaggle.com
  - check out and participate in ML comeptitions
- **Just the last contour plot - I am interested in knowing more about how the background was colored**
- **I'm not familiar with some package/function we use in python**
  - I unfortunately don't have time to go through the code line by line during class so I highly recommend that you study the code outside of class
  - print out the variables
  - read the manuals of the functions

- change things in the code

- **Where and how "Cs = np.logspace(-1,3,13)" is developed is the muddiest for me.**
  - check out help(np.logspace)
  - it's a numpy function that generates uniformly spaced numbers in log space
  - I'll show you the pythonian tricks I came across and found useful
  - you'll find your own tricks and favorite functions
- **Going forward, will the mathematical definitions for some of these ideas be provided? I'm probably in the minority for preferrring this, but I think that it helps to see those definitions, even if they aren't talked about. For example "A dataset is structured if all elements can be minimally embedded in R^d for the same d" or "A dataset is unstructured if the minimal embedding for elements vary"**
  - not so much in this class
  - we will focus on practical issues rather than rigorous mathematical formulation
  - Sam is the mathematician :)

#

Exploratory data analysis in python, part 1

0.2 The steps

1. Exploratory Data Analysis (EDA): you need to understand your data and verify that it doesn't contain errors - do as much EDA as you can!

2. Split the data into different sets: most often the sets are train, validation, and test (or holdout) - practitioners often make errors in this step! - you can split the data randomly, based on groups, based on time, or any other non-standard way if necessary to answer your ML question

3. Preprocess the data: ML models only work if X and Y are numbers! Some ML models additionally require each feature to have 0 mean and 1 standard deviation (standardized features) - often the original features you get contain strings (for example a gender feature would contain 'male', 'female', 'non-binary', 'unknown') which needs to transformed into numbers - often the features are not standardized (e.g., age is between 0 and 100) but it needs to be standardized

4. Choose an evaluation metric: depends on the priorities of the stakeholders - often requires quite a bit of thinking and ethical considerations

5. Choose one or more ML techniques: it is highly recommended that you try multiple models - start with simple models like linear or logistic regression - try also more complex models like nearest neighbors, support vector machines, random forest, etc.

6. Tune the hyperparameters of your ML models (aka cross-validation) - ML techniques have hyperparameters that you need to optimize to achieve best performance - for each ML model, decide which parameters to tune and what values to try - loop through each parameter combination - train one model for each parameter combination - evaluate how well the model performs on the validation set - take the parameter combo that gives the best validation score - evaluate that model on the test set to report how well the model is expected to perform on previously unseen data

7. Interpret your model: black boxes are often not useful - check if your model uses features that make sense (excellent tool for debugging) - often model predictions are not enough, you need to be able to explain how the model arrived to a particular prediction (e.g., in health care)

##

Pandas

- data are often distributed over multiple files/databases (e.g., csv and excel files, sql databases)
- each file/database is read into a pandas dataframe
- you often need to filter dataframes (select specific rows/columns based on index or condition)
- pandas dataframes can be merged and appended

0.2.1 Some notes and advice

- **ALWAYS READ THE HELP OF THE METHODS/FUNCTIONS YOU USE!**

- stackoverflow is your friend, use it! https://stackoverflow.com/

#

Data transformations: pandas data frames

0.2.2 By the end of this lecture, you will be able to

- read in csv, excel, and sql data into a pandas data frame
- filter rows in various ways
- select columns
- merge and append data frames

#

Data transformations: pandas data frames

By the end of this lecture, you will be able to - **read in csv, excel, and sql data into a pandas data frame** - filter rows in various ways - select columns - merge and append data frames

[1]:
```
# how to read in a database into a dataframe and basic dataframe structure
import pandas as pd

# load data from a csv file
df = pd.read_csv('data/adult_data.csv') # there are also pd.read_excel(), and␣
 ↪pd.read_sql()

#print(df)
print(df.head()) # by default, shows the first five rows but check help(df.
 ↪head) to specify the number of rows to show
#print(df.shape) # the shape of your dataframe (number of rows, number of␣
 ↪columns)
#print(df.shape[0]) # number of rows
#print(df.shape[1]) # number of columns
```

```
   age          workclass  fnlwgt   education  education-num  \
0   39          State-gov   77516   Bachelors             13
1   50   Self-emp-not-inc   83311   Bachelors             13
2   38            Private  215646     HS-grad              9
```

```
3  53          Private  234721       11th              7
4  28          Private  338409  Bachelors             13

        marital-status          occupation   relationship   race      sex  \
0        Never-married        Adm-clerical  Not-in-family  White     Male
1   Married-civ-spouse     Exec-managerial        Husband  White     Male
2             Divorced   Handlers-cleaners  Not-in-family  White     Male
3   Married-civ-spouse   Handlers-cleaners        Husband  Black     Male
4   Married-civ-spouse      Prof-specialty           Wife  Black   Female

   capital-gain  capital-loss  hours-per-week  native-country gross-income
0          2174             0              40   United-States        <=50K
1             0             0              13   United-States        <=50K
2             0             0              40   United-States        <=50K
3             0             0              40   United-States        <=50K
4             0             0              40            Cuba        <=50K
```

0.2.3 Packages

A package is a collection of classes and functions. - a dataframe (pd.DataFrame()) is a pandas class - a class is the blueprint of how the data should be organized - classes have methods which can perform operations on the data (e.g., .head(), .shape) - df is an object, an instance of the class. - we put data into the class - methods are attached to objects - you cannot call pd.head(), you can only call df.head() - read_csv is a function - functions are called from the package - you cannot call df.read_csv, you can only call pd.read_csv()

0.2.4 DataFrame structure: both rows and columns are indexed!

- index column, no name
  - contains the row names
  - by default, index is a range object from 0 to number of rows - 1
  - any column can be turned into an index, so indices can be non-number, and also non-unique. more on this later.
- columns with column names on top

0.2.5 Always print your dataframe to check if it looks ok!

0.2.6 Most common reasons it might not look ok:

- the first row is not the column name
  - there are rows above the column names that need to be skipped
  - there is no column name but by default, pandas assumes the first row is the column name. as a result, the values of the first row end up as column names.
- character encoding is off
- separator is not comma but some other charachter

[2]:
```python
# check the help to find the solution
help(pd.read_csv)
```

```
Help on function read_csv in module pandas.io.parsers.readers:

read_csv(filepath_or_buffer: 'FilePathOrBuffer', sep=<no_default>,
delimiter=None, header='infer', names=<no_default>, index_col=None,
usecols=None, squeeze=False, prefix=<no_default>, mangle_dupe_cols=True, dtype:
'DtypeArg | None' = None, engine=None, converters=None, true_values=None,
false_values=None, skipinitialspace=False, skiprows=None, skipfooter=0,
nrows=None, na_values=None, keep_default_na=True, na_filter=True, verbose=False,
skip_blank_lines=True, parse_dates=False, infer_datetime_format=False,
keep_date_col=False, date_parser=None, dayfirst=False, cache_dates=True,
iterator=False, chunksize=None, compression='infer', thousands=None, decimal:
'str' = '.', lineterminator=None, quotechar='"', quoting=0, doublequote=True,
escapechar=None, comment=None, encoding=None, encoding_errors: 'str | None' =
'strict', dialect=None, error_bad_lines=None, warn_bad_lines=None,
on_bad_lines=None, delim_whitespace=False, low_memory=True, memory_map=False,
float_precision=None, storage_options: 'StorageOptions' = None)
    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 ','
        Delimiter to use. If sep is 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 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, default ``None``
        Alias for sep.
    header : int, list of int, default 'infer'
        Row number(s) to use as the column names, and the start of the
        data.  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 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 multi-index 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 : array-like, optional
        List of column names to use. 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 : int, str, sequence of int / str, or False, default ``None``
      Column(s) to use as the row labels of the ``DataFrame``, either given as
      string name or column index. If a sequence of int / str is given, a
      MultiIndex is used.

      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 : list-like or callable, optional
        Return a subset of the columns. 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). 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 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.
    squeeze : bool, default False
        If the parsed data only contains one column then return a Series.
    prefix : str, optional
        Prefix to add to column numbers when no header, e.g. 'X' for X0, X1, …
    mangle_dupe_cols : bool, default True
        Duplicate columns will be specified as 'X', 'X.1', …'X.N', rather than
        'X'…'X'. Passing in False will cause data to be overwritten if there
        are duplicate names in the columns.
    dtype : Type name or dict of column -> type, optional
        Data type for data or 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.
    engine : {'c', 'python'}, optional
        Parser engine to use. The C engine is faster while the python engine is
        currently more feature-complete.
    converters : dict, optional
        Dict of functions for converting values in certain columns. Keys can
either
        be integers or column labels.
    true_values : list, optional
        Values to consider as True.
    false_values : list, optional
        Values to consider as False.
    skipinitialspace : bool, default False
        Skip spaces after delimiter.
    skiprows : list-like, 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 : scalar, str, list-like, or dict, 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', '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 NAs, 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.
    skip_blank_lines : bool, default True
        If True, skip over blank lines rather than interpreting as NaN values.
    parse_dates : bool or list of int or names or list of lists or dict, default
False
        The behavior is as follows:

        * boolean. If True -> try parsing the index.
        * 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 lists. e.g.  If [[1, 3]] -> combine columns 1 and 3 and parse
as
```

```
          a single date column.
        * dict, e.g. {'foo' : [1, 3]} -> parse columns 1, 3 as date and call
          result 'foo'

        If a column or index cannot be represented as an array of datetimes,
        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 ``pd.to_datetime`` after
        ``pd.read_csv``. To parse an index or column with a mixture of
timezones,
        specify ``date_parser`` to be a partially-applied
        :func:`pandas.to_datetime` with ``utc=True``. See
        :ref:`io.csv.mixed_timezones` for more.

        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.
    keep_date_col : bool, default False
        If True and `parse_dates` specifies combining multiple columns then
        keep the original columns.
    date_parser : function, 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.
    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.

        .. versionadded:: 0.25.0
```

```
    iterator : bool, default False
        Return TextFileReader object for iteration or getting chunks with
        ``get_chunk()``.

        .. versionchanged:: 1.2

           ``TextFileReader`` is a context manager.
    chunksize : int, optional
        Return 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``.

        .. versionchanged:: 1.2

           ``TextFileReader`` is a context manager.
    compression : {'infer', 'gzip', 'bz2', 'zip', 'xz', None}, 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', or '.xz' (otherwise no
        decompression). If using 'zip', the ZIP file must contain only one data
        file to be read in. Set to None for no decompression.
    thousands : str, optional
        Thousands separator.
    decimal : str, default '.'
        Character to recognize as decimal point (e.g. use ',' for European
data).
    lineterminator : str (length 1), optional
        Character to break file into lines. Only valid with C parser.
    quotechar : str (length 1), optional
        The character used to denote the start and end of a quoted item. Quoted
        items can include the delimiter and it will be ignored.
    quoting : int or csv.QUOTE_* instance, default 0
        Control field quoting behavior per ``csv.QUOTE_*`` constants. Use one of
        QUOTE_MINIMAL (0), QUOTE_ALL (1), QUOTE_NONNUMERIC (2) or QUOTE_NONE
(3).
    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
        One-character string used to escape other characters.
    comment : str, optional
        Indicates 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
        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>`_ .

        .. versionchanged:: 1.2

           When ``encoding`` is ``None``, ``errors="replace"`` is passed to
           ``open()``. Otherwise, ``errors="strict"`` is passed to ``open()``.
           This behavior was previously only the case for ``engine="python"``.

        .. versionchanged:: 1.3.0

           ``encoding_errors`` is a new argument. ``encoding`` has no longer an
           influence on how encoding errors are handled.

    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.
    error_bad_lines : bool, default ``None``
        Lines with too many fields (e.g. a csv line with too many commas) will
by
        default cause an exception to be raised, and no DataFrame will be
returned.
        If False, then these "bad lines" will be dropped from the DataFrame that
is
        returned.

        .. deprecated:: 1.3.0
           The ``on_bad_lines`` parameter should be used instead to specify
behavior upon
           encountering a bad line instead.
    warn_bad_lines : bool, default ``None``
```

```
        If error_bad_lines is False, and warn_bad_lines is True, a warning for
each
        "bad line" will be output.

        .. deprecated:: 1.3.0
           The ``on_bad_lines`` parameter should be used instead to specify
behavior upon
           encountering a bad line instead.
    on_bad_lines : {'error', 'warn', 'skip'}, 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

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

        .. versionchanged:: 1.2

    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`` as header options. For other URLs (e.g.
        starting with "s3://", and "gcs://") the key-value pairs are forwarded
to
        ``fsspec``. Please see ``fsspec`` and ``urllib`` for more details.

        .. versionadded:: 1.2

    Returns
    -------
    DataFrame or TextParser
        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_csv : Read a comma-separated values (csv) file into DataFrame.
    read_fwf : Read a table of fixed-width formatted lines into DataFrame.

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

0.3 Exercise 1

How should we read in adult_test.csv properly? Identify and fix the problem.

[3]:
```python
# df = pd.read_csv('data/adult_test.csv')
# print(df.head())
```

#

Data transformations: pandas data frames

By the end of this lecture, you will be able to - read in csv, excel, and sql data into a pandas data frame - **filter rows in various ways** - select columns - merge and append data frames

0.3.1 How to select rows?

1) Integer-based indexing, numpy arrays are indexed the same way.

2) Select rows based on the value of the index column

3) select rows based on column condition

0.3.2 1) Integer-based indexing, numpy arrays are indexed the same way.

[4]:
```
# df.iloc[] - for more info, see https://pandas.pydata.org/pandas-docs/stable/
↪user_guide/indexing.html#indexing-integer
# iloc is how numpy arrays are indexed (non-standard python indexing)

# [start:stop:step] -  general indexing format

# start stop step are optional
#print(df.iloc[:])
#print(df.iloc[::])
#print(df.iloc[::1])

# select one row - 0-based indexing
#print(df.iloc[3])

# indexing from the end of the data frame
print(df.iloc[-1])
```

```
age                              52
workclass              Self-emp-inc
fnlwgt                       287927
education                   HS-grad
education-num                     9
marital-status   Married-civ-spouse
occupation          Exec-managerial
relationship                   Wife
race                          White
sex                          Female
capital-gain                  15024
capital-loss                      0
hours-per-week                   40
native-country        United-States
gross-income                   >50K
Name: 32560, dtype: object
```

[5]:
```
# select a slice - stop index not included
#print(df.iloc[3:7])

# select every second element of the slice - stop index not included
#print(df.iloc[3:7:2])

#print(df.iloc[3:7:-2]) # return empty dataframe
#print(df.iloc[7:3:-2])#  return rows with indices 7 and 5. 3 is the stop so it␣
↪is not included

# can be used to reverse rows
```

```
#print(df.iloc[::-1])

# here is where indexing gets non-standard python
# select the 2nd, 5th, and 10th rows
print(df.iloc[[1,4,9]]) # such indexing doesn't work with lists but it works␣
↪with numpy arrays
```

```
   age         workclass  fnlwgt   education  education-num  \
1   50  Self-emp-not-inc   83311   Bachelors             13
4   28           Private  338409   Bachelors             13
9   42           Private  159449   Bachelors             13

       marital-status        occupation relationship   race      sex  \
1  Married-civ-spouse   Exec-managerial      Husband  White     Male
4  Married-civ-spouse    Prof-specialty         Wife  Black   Female
9  Married-civ-spouse   Exec-managerial      Husband  White     Male

   capital-gain  capital-loss  hours-per-week  native-country gross-income
1             0             0              13   United-States        <=50K
4             0             0              40            Cuba        <=50K
9          5178             0              40   United-States         >50K
```

0.3.3 2) Select rows based on the value of the index column

[6]:
```
# df.loc[] - for more info, see https://pandas.pydata.org/pandas-docs/stable/
↪user_guide/indexing.html#indexing-label

print(df.index) # the default index when reading in a file is a range index. In␣
↪this case,
                 # .loc and .iloc works ALMOST the same.
# one difference:
#print(df.loc[3:9:2]) # this selects the 4th, 6th, 8th, 10th rows - the stop␣
↪element is included!

help(df.set_index)
```

```
RangeIndex(start=0, stop=32561, step=1)
Help on method set_index in module pandas.core.frame:

set_index(keys, drop: 'bool' = True, append: 'bool' = False, inplace: 'bool' =
False, verify_integrity: 'bool' = False) method of pandas.core.frame.DataFrame
instance
    Set the DataFrame index using existing columns.

    Set the DataFrame index (row labels) using one or more existing
    columns or arrays (of the correct length). The index can replace the
    existing index or expand on it.
```

```
Parameters
----------
keys : label or array-like or list of labels/arrays
    This parameter can be either a single column key, a single array of
    the same length as the calling DataFrame, or a list containing an
    arbitrary combination of column keys and arrays. Here, "array"
    encompasses :class:`Series`, :class:`Index`, ``np.ndarray``, and
    instances of :class:`~collections.abc.Iterator`.
drop : bool, default True
    Delete columns to be used as the new index.
append : bool, default False
    Whether to append columns to existing index.
inplace : bool, default False
    If True, modifies the DataFrame in place (do not create a new object).
verify_integrity : bool, default False
    Check the new index for duplicates. Otherwise defer the check until
    necessary. Setting to False will improve the performance of this
    method.

Returns
-------
DataFrame or None
    Changed row labels or None if ``inplace=True``.

See Also
--------
DataFrame.reset_index : Opposite of set_index.
DataFrame.reindex : Change to new indices or expand indices.
DataFrame.reindex_like : Change to same indices as other DataFrame.

Examples
--------
>>> df = pd.DataFrame({'month': [1, 4, 7, 10],
…                    'year': [2012, 2014, 2013, 2014],
…                    'sale': [55, 40, 84, 31]})
>>> df
   month  year  sale
0      1  2012    55
1      4  2014    40
2      7  2013    84
3     10  2014    31

Set the index to become the 'month' column:

>>> df.set_index('month')
       year  sale
month
```

```
1       2012    55
4       2014    40
7       2013    84
10      2014    31

Create a MultiIndex using columns 'year' and 'month':

>>> df.set_index(['year', 'month'])
            sale
year  month
2012  1     55
2014  4     40
2013  7     84
2014  10    31

Create a MultiIndex using an Index and a column:

>>> df.set_index([pd.Index([1, 2, 3, 4]), 'year'])
         month  sale
   year
1  2012  1      55
2  2014  4      40
3  2013  7      84
4  2014  10     31

Create a MultiIndex using two Series:

>>> s = pd.Series([1, 2, 3, 4])
>>> df.set_index([s, s**2])
      month  year  sale
1 1       1  2012    55
2 4       4  2014    40
3 9       7  2013    84
4 16     10  2014    31
```

[7]:

```
df_index_age = df.set_index('age',drop=False)

#print(df_index_age.index)
#print(df_index_age.head())

print(df_index_age.loc[30].head()) # collect everyone with age 30 - the index
 ↪is non-unique
```

```
     age     workclass  fnlwgt      education  education-num  \
age
30    30     State-gov  141297      Bachelors             13
30    30   Federal-gov   59951   Some-college             10
```

```
30    30        Private  188146        HS-grad                9
30    30        Private   59496      Bachelors               13
30    30        Private   54334            9th                5

          marital-status          occupation    relationship  \
age
30    Married-civ-spouse      Prof-specialty         Husband
30    Married-civ-spouse        Adm-clerical       Own-child
30    Married-civ-spouse   Machine-op-inspct         Husband
30    Married-civ-spouse               Sales         Husband
30         Never-married               Sales   Not-in-family

                    race    sex  capital-gain  capital-loss  hours-per-week  \
age
30    Asian-Pac-Islander   Male             0             0              40
30                 White   Male             0             0              40
30                 White   Male          5013             0              40
30                 White   Male          2407             0              40
30                 White   Male             0             0              40

     native-country gross-income
age
30            India         >50K
30    United-States        <=50K
30    United-States        <=50K
30    United-States        <=50K
30    United-States        <=50K
```

0.3.4 3) select rows based on column condition

[8]:
```
# one condition
#print(df[df['age']==30].head())
# here is the condition: it's a boolean series - series is basically a
 ↪dataframe with one column
#print(df['age']==30)

# multiple conditions can be combined with & (and) | (or)
#print(df[(df['age']>30)&(df['age']<35)].head())
print(df[(df['age']==90)|(df['native-country']==' Hungary')])
```

```
       age         workclass  fnlwgt      education  education-num  \
222     90           Private   51744        HS-grad              9
1040    90           Private  137018        HS-grad              9
1935    90           Private  221832      Bachelors             13
2303    90           Private   52386   Some-college             10
2891    90           Private  171956   Some-college             10
4070    90           Private  313986           11th              7
4109    90                 ?  256514      Bachelors             13
```

5104	90	Private	52386	Some-college	10
5272	90	Private	141758	9th	5
5370	90	Local-gov	227796	Masters	14
5406	90	Private	51744	Masters	14
6232	90	Self-emp-not-inc	155981	Bachelors	13
6624	90	Private	313986	11th	7
8562	49	Private	122066	HS-grad	9
8806	90	Private	87372	Prof-school	15
8963	90	?	77053	HS-grad	9
8973	90	Private	46786	Bachelors	13
10210	90	Self-emp-not-inc	282095	Some-college	10
10545	90	Private	175491	HS-grad	9
11512	90	Private	87285	HS-grad	9
11731	90	?	39824	HS-grad	9
11996	90	Private	40388	Bachelors	13
12451	90	?	225063	Some-college	10
12529	65	Private	172510	Some-college	10
12975	90	Private	250832	10th	6
13928	81	Self-emp-not-inc	123959	Bachelors	13
14159	90	Local-gov	187749	Assoc-acdm	12
15259	60	Private	114263	Bachelors	13
15356	90	Private	90523	HS-grad	9
15892	90	Private	88991	Bachelors	13
17144	28	Self-emp-not-inc	183523	HS-grad	9
17735	26	Private	358975	Some-college	10
18277	90	Private	311184	Bachelors	13
18413	90	Private	313749	Bachelors	13
18725	90	Local-gov	153602	HS-grad	9
18832	90	Private	115306	Masters	14
18839	66	Self-emp-not-inc	174995	Assoc-acdm	12
19212	90	Private	139660	Some-college	10
19489	90	Private	84553	HS-grad	9
19747	90	Private	226968	7th-8th	4
20610	90	Private	206667	Masters	14
21371	30	Private	207668	Bachelors	13
22220	90	Private	52386	Bachelors	13
22658	54	Private	188186	HS-grad	9
23023	24	Private	117779	Bachelors	13
24043	90	Self-emp-not-inc	82628	HS-grad	9
24238	90	?	166343	1st-4th	2
25303	90	?	175444	7th-8th	4
27041	57	Self-emp-inc	258883	HS-grad	9
27750	55	Private	143266	Assoc-voc	11
28463	90	Federal-gov	195433	HS-grad	9
30346	47	Private	180277	HS-grad	9
31030	90	Private	47929	HS-grad	9
31696	90	?	313986	HS-grad	9
32277	90	Private	313749	HS-grad	9

32367	90	Local-gov	214594	7th-8th	4

	marital-status	occupation	relationship \
222	Never-married	Other-service	Not-in-family
1040	Never-married	Other-service	Not-in-family
1935	Married-civ-spouse	Exec-managerial	Husband
2303	Never-married	Other-service	Not-in-family
2891	Separated	Adm-clerical	Own-child
4070	Never-married	Handlers-cleaners	Own-child
4109	Widowed	?	Other-relative
5104	Never-married	Other-service	Not-in-family
5272	Never-married	Adm-clerical	Not-in-family
5370	Married-civ-spouse	Exec-managerial	Husband
5406	Never-married	Exec-managerial	Not-in-family
6232	Married-civ-spouse	Prof-specialty	Husband
6624	Married-civ-spouse	Craft-repair	Husband
8562	Married-civ-spouse	Exec-managerial	Husband
8806	Married-civ-spouse	Prof-specialty	Husband
8963	Widowed	?	Not-in-family
8973	Married-civ-spouse	Sales	Husband
10210	Married-civ-spouse	Farming-fishing	Husband
10545	Married-civ-spouse	Craft-repair	Husband
11512	Never-married	Other-service	Own-child
11731	Widowed	?	Not-in-family
11996	Never-married	Exec-managerial	Not-in-family
12451	Never-married	?	Own-child
12529	Widowed	Prof-specialty	Not-in-family
12975	Married-civ-spouse	Exec-managerial	Husband
13928	Widowed	Prof-specialty	Not-in-family
14159	Married-civ-spouse	Adm-clerical	Husband
15259	Divorced	Exec-managerial	Not-in-family
15356	Widowed	Transport-moving	Unmarried
15892	Married-civ-spouse	Exec-managerial	Wife
17144	Married-civ-spouse	Craft-repair	Husband
17735	Never-married	Priv-house-serv	Not-in-family
18277	Married-civ-spouse	Sales	Husband
18413	Never-married	Prof-specialty	Own-child
18725	Married-civ-spouse	Other-service	Husband
18832	Never-married	Exec-managerial	Own-child
18839	Married-civ-spouse	Craft-repair	Husband
19212	Divorced	Sales	Unmarried
19489	Married-civ-spouse	Machine-op-inspct	Husband
19747	Married-civ-spouse	Machine-op-inspct	Husband
20610	Married-civ-spouse	Prof-specialty	Wife
21371	Never-married	Tech-support	Not-in-family
22220	Never-married	Prof-specialty	Not-in-family
22658	Never-married	Other-service	Other-relative
23023	Never-married	Prof-specialty	Not-in-family

24043	Never-married	Exec-managerial	Not-in-family
24238	Widowed	?	Not-in-family
25303	Separated	?	Not-in-family
27041	Married-civ-spouse	Transport-moving	Husband
27750	Married-civ-spouse	Craft-repair	Husband
28463	Married-civ-spouse	Craft-repair	Husband
30346	Married-civ-spouse	Adm-clerical	Wife
31030	Married-civ-spouse	Machine-op-inspct	Husband
31696	Married-civ-spouse	?	Husband
32277	Widowed	Adm-clerical	Unmarried
32367	Married-civ-spouse	Protective-serv	Husband

	race	sex	capital-gain	capital-loss \
222	Black	Male	0	2206
1040	White	Female	0	0
1935	White	Male	0	0
2303	Asian-Pac-Islander	Male	0	0
2891	White	Female	0	0
4070	White	Male	0	0
4109	White	Female	991	0
5104	Asian-Pac-Islander	Male	0	0
5272	White	Female	0	0
5370	White	Male	20051	0
5406	Black	Male	0	0
6232	White	Male	10566	0
6624	White	Male	0	0
8562	White	Male	0	0
8806	White	Male	20051	0
8963	White	Female	0	4356
8973	White	Male	9386	0
10210	White	Male	0	0
10545	White	Male	9386	0
11512	White	Female	0	0
11731	White	Male	401	0
11996	White	Male	0	0
12451	Asian-Pac-Islander	Male	0	0
12529	White	Female	1848	0
12975	White	Male	0	0
13928	White	Female	0	1668
14159	Asian-Pac-Islander	Male	0	0
15259	White	Female	0	0
15356	White	Male	0	0
15892	White	Female	0	0
17144	White	Male	0	0
17735	White	Female	0	0
18277	White	Male	0	0
18413	White	Female	0	0
18725	White	Male	6767	0

18832	White	Female	0	0
18839	White	Male	2290	0
19212	Black	Female	0	0
19489	White	Male	0	0
19747	White	Male	0	0
20610	White	Female	0	0
21371	White	Male	0	0
22220	Asian-Pac-Islander	Male	0	0
22658	White	Female	0	0
23023	White	Male	0	0
24043	White	Male	2964	0
24238	Black	Female	0	0
25303	White	Female	0	0
27041	White	Male	5178	0
27750	White	Male	0	0
28463	White	Male	0	0
30346	White	Female	0	0
31030	White	Male	0	0
31696	White	Male	0	0
32277	White	Female	0	0
32367	White	Male	2653	0

	hours-per-week	native-country	gross-income
222	40	United-States	<=50K
1040	40	United-States	<=50K
1935	45	United-States	<=50K
2303	35	United-States	<=50K
2891	40	Puerto-Rico	<=50K
4070	40	United-States	<=50K
4109	10	United-States	<=50K
5104	35	United-States	<=50K
5272	40	United-States	<=50K
5370	60	United-States	>50K
5406	50	United-States	>50K
6232	50	United-States	<=50K
6624	40	United-States	<=50K
8562	30	Hungary	<=50K
8806	72	United-States	>50K
8963	40	United-States	<=50K
8973	15	United-States	>50K
10210	40	United-States	<=50K
10545	50	Ecuador	>50K
11512	24	United-States	<=50K
11731	4	United-States	<=50K
11996	55	United-States	<=50K
12451	10	South	<=50K
12529	20	Hungary	<=50K
12975	40	United-States	<=50K

```
13928                    3         Hungary        <=50K
14159                   20     Philippines        <=50K
15259                   40         Hungary         >50K
15356                   99   United-States        <=50K
15892                   40         England         >50K
17144                   50         Hungary        <=50K
17735                   50         Hungary        <=50K
18277                   20               ?        <=50K
18413                   10   United-States        <=50K
18725                   40   United-States        <=50K
18832                   40   United-States        <=50K
18839                   30         Hungary        <=50K
19212                   37   United-States        <=50K
19489                   40   United-States        <=50K
19747                   40   United-States        <=50K
20610                   40   United-States         >50K
21371                   60         Hungary        <=50K
22220                   40   United-States        <=50K
22658                   20         Hungary        <=50K
23023                   10         Hungary        <=50K
24043                   12   United-States        <=50K
24238                   40   United-States        <=50K
25303                   15   United-States        <=50K
27041                   60         Hungary         >50K
27750                   50         Hungary         >50K
28463                   30   United-States        <=50K
30346                   40         Hungary        <=50K
31030                   40   United-States        <=50K
31696                   40   United-States         >50K
32277                   25   United-States        <=50K
32367                   40   United-States        <=50K
```

0.3.5 Exercise 2

How many people in adult_data.csv work at least 60 hours a week and have a doctorate?

#

Data transformations: pandas data frames

By the end of this lecture, you will be able to - read in csv, excel, and sql data into a pandas data frame - filter rows in various ways - **select columns** - merge and append data frames

[9]:

```
columns =  df.columns
#print(columns)

# select columns by column name
#print(df[['age','hours-per-week']])
#print(columns[[1,5,7]])
```

```python
#print(df[columns[[1,5,7]]])

# select columns by index using iloc
#print(df.iloc[:,3])

# select columns by index - not standard python indexing
#print(df.iloc[:,[3,5,6]])

# select columns by index -  standard python indexing
print(df.iloc[:,::2])
```

```
       age  fnlwgt  education-num         occupation   race  capital-gain  \
0       39   77516             13       Adm-clerical  White          2174
1       50   83311             13    Exec-managerial  White             0
2       38  215646              9  Handlers-cleaners  White             0
3       53  234721              7  Handlers-cleaners  Black             0
4       28  338409             13     Prof-specialty  Black             0
…     …     …              …                …      …           …
32556   27  257302             12       Tech-support  White             0
32557   40  154374              9  Machine-op-inspct  White             0
32558   58  151910              9       Adm-clerical  White             0
32559   22  201490              9       Adm-clerical  White             0
32560   52  287927              9    Exec-managerial  White         15024

       hours-per-week gross-income
0                  40        <=50K
1                  13        <=50K
2                  40        <=50K
3                  40        <=50K
4                  40        <=50K
…                 …          …
32556              38        <=50K
32557              40         >50K
32558              40        <=50K
32559              20        <=50K
32560              40         >50K

[32561 rows x 8 columns]
```

#

Data transformations: pandas data frames

By the end of this lecture, you will be able to - read in csv, excel, and sql data into a pandas data frame - filter rows in various ways - select columns - **merge and append data frames**

0.3.6 How to merge dataframes?

Merge - info on data points are distributed in multiple files

[10]:
```
# We have two datasets from two hospitals

hospital1 = {'ID':['ID1','ID2','ID3','ID4','ID5','ID6','ID7'],'col1':
↪[5,8,2,6,0,2,5],'col2':['y','j','w','b','a','b','t']}
df1 = pd.DataFrame(data=hospital1)
print(df1)

hospital2 = {'ID':['ID2','ID5','ID6','ID10','ID11'],'col3':
↪[12,76,34,98,65],'col2':['q','u','e','l','p']}
df2 = pd.DataFrame(data=hospital2)
print(df2)
```

```
    ID  col1 col2
0  ID1     5    y
1  ID2     8    j
2  ID3     2    w
3  ID4     6    b
4  ID5     0    a
5  ID6     2    b
6  ID7     5    t
     ID  col3 col2
0   ID2    12    q
1   ID5    76    u
2   ID6    34    e
3  ID10    98    l
4  ID11    65    p
```

[11]:
```
# we are interested in only patients from hospital1
#df_left = df1.merge(df2,how='left',on='ID') # IDs from the left dataframe␣
↪(df1) are kept
#print(df_left)

# we are interested in only patients from hospital2
#df_right = df1.merge(df2,how='right',on='ID') # IDs from the right dataframe␣
↪(df2) are kept
#print(df_right)

# we are interested in patiens who were in both hospitals
#df_inner = df1.merge(df2,how='inner',on='ID') # merging on IDs present in both␣
↪dataframes
#print(df_inner)

# we are interested in all patients who visited at least one of the hospitals
#df_outer = df1.merge(df2,how='outer',on='ID')  # merging on IDs present in any␣
↪dataframe
#print(df_outer)
```

0.3.7 How to append dataframes?

Append - new data comes in over a period of time. E.g., one file per month/quarter/fiscal year etc.

You want to combine these files into one data frame.

[12]:
```
#df_append = df1.append(df2) # note that rows with ID2, ID5, and ID6  are␣
↪duplicated! Indices are duplicated too.
#print(df_append)

#df_append = df1.append(df2,ignore_index=True) # note that rows with ID2, ID5,␣
↪and ID6  are duplicated!
#print(df_append)

#d3 = {'ID':['ID23','ID94','ID56','ID17'],'col1':['rt','h','st','ne'],'col2':
↪[23,86,23,78]}
#df3 = pd.DataFrame(data=d3)
#print(df3)

#df_append = df1.append([df2,df3],ignore_index=True) # multiple dataframes can␣
↪be appended to df1
#print(df_append)
```

0.3.8 Exercise 3

[13]:
```
raw_data_1 = {
        'subject_id': ['1', '2', '3', '4', '5'],
        'first_name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
        'last_name': ['Anderson', 'Ackerman', 'Ali', 'Aoni', 'Atiches']}

raw_data_2 = {
        'subject_id': ['6', '7', '8', '9', '10'],
        'first_name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
        'last_name': ['Bonder', 'Black', 'Balwner', 'Brice', 'Btisan']}

raw_data_3 = {
        'subject_id': ['1', '2', '3', '4', '5', '7', '8', '9', '10', '11'],
        'test_id': [51, 15, 15, 61, 16, 14, 15, 1, 61, 16]}

# Create three data frames from raw_data_1, 2, and 3.
# Append the first two data frames and assign it to df_append.
# Merge the third data frame with df_append such that only subject_ids from␣
↪df_append are present.
# Assign the new data frame to df_merge.
# How many rows and columns do we have in df_merge?
```

0.3.9 Always check that the resulting dataframe is what you wanted to end up with!

- small toy datasets are ideal to test your code.

0.3.10 If you need to do a more complicated dataframe operation, check out pd.concat()!

0.3.11 We will learn how to add/delete/modify columns later when we learn about feature engineering.

0.3.12 By now, you are able to

- read in csv, excel, and sql data into a pandas data frame
- filter rows in various ways
- select columns
- merge and append data frames

1 Mud card

[ ]:

```

```