Mud card

- **Could you specify on the baseline of accuracy in the spam email example?**
- **how unbalanced data affects good accuracy values**
    - the baseline accuracy is determined using the target variable of the training set
    - the baseline accuracy is the fraction of points in the most populous class
        - if 5% of emails are spam, and 95% of emails are not spam, 0.95 is the baseline accuracy because if you predict 'non-spam' for each point, you will be correct 95% of the times.
    - the baseline is calculated differently for other evaluation metrics
    - the general concept is to use the target variable only (no features) to figure out the baseline
    - we will cover this later
- **"What is the alternative to the hard-coded cross validation loop? Is there a sci-kit learn function for running cross-validation and tuning hyperparameters at this stage?**
    - yes there is and we will learn about it once you have solid foundations of the more basic techniques
    - the issue is that these one-line scikit-learn solutions hide a lot of stuff from you so you should only use them if you know exactly what you are doing.
- **Before this stage, during the train-test split, how important is training (10 models) using random_state_... is this just for small datasets?**
    - the smaller you dataset, the more important it is
    - generally speaking it is always a good idea to try at least 3 random states even if you dataset is large if you can manage it given your computational resources
- **I do not have much experience in coding, and am not a 100% sure what each line of code in the examples do or represent - what are some resources I can look into to understand what is happening line by line?**
- **I didn't understand the syntax and code of splitting**
- **Maybe more clear explaination on the code**
- **Would you be able to walk through the code for hyper tuning again? How did you come up with the parameters for np.logspace?**
    - you should read the help of each function used in a line and print out all variables used in a line to know what's happening and how variables change
- **If a new data point has a predicted probability of 0.2 does that mean it has an 80% probability that its target value will be 0?**
    - depends. you will see that classification models return two columns when you predict probabilities: the class 0 probability and the class 1 probability
    - the two probabilities sum to 1 for each point
    - if the class 0 probability is 0.2, it means the point is class 1 with 80% probability
    - if the class 1 probability is 0.2, it means the point is class 0 with 80% probability
- **How does SVM work? Is it like linear regression, but the line is curvy**

- **What exactly is C?**
- **what is c mean for SVC**
- **I am still unclear about what C is or what the hyperparameters are in the context of a ML algorithm.**
  - it is a non-linear model and we will cover it in a few weeks
- **I was a bit confused on the ML techniques portion. Will we eventually learn such techniques and when to apply them?**
  - yes :)
- **Could you please discuss bias-variance tradeoff again in the next class? Along with a few more real-time examples and how it can affect a data science project?**
- **I am still a little confused about finding the ideal fit of the model based on the graph showing the c parameter and accuracy.**
- **I thought the muddiest part of the lecture was the bias-variance tradeoff**
- **Why is it necessary to have test and validation sets?**
- **Could you explain in more detail what "validation" means and how its different from testing?**
- **I was still a little unsure what the difference between validation and testing is?**
- **I'm struggling with tuning the hyperparameters.**
- **Parts 5-7 and c-values/cross validation in general just went over my head.**
- **Still not quite clear on the rationale for validation vs. test set - what's the actual distinction?**
  - I'll cover this again now
- **What is the differencfe between X and Y and Curly XY?**
  - curly X and curly Y are the sets of all possible instances and target variables
  - regular X and Y are a (usually small) sample drawn from curly X and curly Y
- **How do you know when you have done 'enough' EDA for a given dataset?**
  - you don't know :)
  - but at the very least you should know what each feature in your dataset means, you should know the typical summary statistics of each feature (we will discuss summary stats next tuesday)
  - ML pipeline development is non-linear. we cover the steps in a linear fashion but sometimes you need to do more EDA after you do cross validation because something doesn't seem OK in your results for example
- **And will we go over what characteristics we see in EDA that point to different ML methods?**
  - such insights are very rare. generally you need to try as many models as you can.
  - it is rare that you can exclude a ML model based on EDA
- **How do we know the bounds and scale for hyperparameter tuning? For instance, the examples uses logscale.**
  - we will cover this for each ML technique separately in a couple of weeks
- **Once we get an optimal hyper parameter for one random state split of our data, do we then use that C retroactively for each random state we test after that?**
  - no, you would determine an optimal C for each random state

    - you might find that different Cs are optimal for different random states
- **What other evaluation metrics are common and how do we determine which is appropriate?**
    - we will spend a week on this question :)
- **It would have been better if there were an example table of the learner's input , especially the label set Y.**
- **Needs to know the basic concept of data set**
    - open the toy_data.csv in the data folder, that's a good example of a simple dataset

Exploratory data analysis in python, part 1

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

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

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

In [1]:

```python
# 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 p

#print(df)
print(df.head()) # by default, shows the first five rows but check help(df.head
#print(df.shape) # the shape of your dataframe (number of rows, number of colum
#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
```

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()

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

Always print your dataframe to check if it looks ok!

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

In [2]:

```
# 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: 'FilePath | ReadCsvBuffer[bytes] | ReadCsvBuffer
[str]', sep=<no_default>, delimiter=None, header='infer', names=<no_default>,
index_col=None, usecols=None, squeeze=None, prefix=<no_default>, mangle_dupe_c
ols=True, dtype: 'DtypeArg | None' = None, engine: 'CSVEngine | None' = 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=None, infer_
datetime_format=False, keep_date_col=False, date_parser=None, dayfirst=False,
cache_dates=True, iterator=False, chunksize=None, compression: 'CompressionOpt
ions' = 'infer', thousands=None, decimal: 'str' = '.', lineterminator=None, qu
otechar='"', quoting=0, doublequote=True, escapechar=None, comment=None, encod
ing=None, encoding_errors: 'str | None' = 'strict', dialect=None, error_bad_li
nes=None, warn_bad_lines=None, on_bad_lines=None, delim_whitespace=False, low_
memory=True, memory_map=False, float_precision=None, storage_options: 'Storage
Options' = 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, suc
h 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 de
tect
        the separator, but the Python parsing engine can, meaning the latter w
ill
        be used and automatically detect the separator by Python's builtin sni
ffer
        tool, ``csv.Sniffer``. In addition, separators longer than 1 character
and
        different from ``'\s+'`` will be interpreted as regular expressions an
d
        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, None, 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 na
mes.
        Duplicates in this list are not allowed.
    index_col : int, str, sequence of int / str, or False, optional, 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 f
irst
      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 stri
ngs
        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-li
ke
        `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 colu
mns
        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 colum
n
        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.

        .. deprecated:: 1.4.0
            Append ``.squeeze("columns")`` to the call to ``read_csv`` to sque
eze
            the data.
    prefix : str, optional
        Prefix to add to column numbers when no header, e.g. 'X' for X0, X1,
...

        .. deprecated:: 1.4.0
           Use a list comprehension on the DataFrame's columns after calling `
`read_csv``.
    mangle_dupe_cols : bool, default True
        Duplicate columns will be specified as 'X', 'X.1', ...'X.N', rather th
an
        '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', '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 s
upported by
        the pyarrow engine.

        .. versionadded:: 1.4.0

            The "pyarrow" engine was added as an *experimental* engine, and so
me features
            are unsupported, or may not work correctly, with this engine.
    converters : dict, optional
        Dict of functions for converting values in certain columns. Keys can e
ither
        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 otherwi
se.
        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 fil
es.
    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', '-n
an',
        '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 dat
a.
        Depending on whether `na_values` is passed in, the behavior is as foll
ows:

        * If `keep_default_na` is True, and `na_values` are specified, `na_val
ues`
          is appended to the default NaN values used for parsing.
        * If `keep_default_na` is True, and `na_values` are not specified, onl
y
          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_value
s). In
        data without any NAs, passing na_filter=False can improve the performa
nce
        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, defau
lt 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 pars
e 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 colu
mn
        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 timezon
es,
        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 inferr
ed,
        switch to a faster method of parsing them. In some cases this can incr
ease
        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 arra
y of
        datetime instances. The default uses ``dateutil.parser.parser`` to do
the
        conversion. Pandas will try to call `date_parser` in three different w
ays,
        advancing to the next if an exception occurs: 1) Pass one or more arra
ys
        (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 o
r
        more strings (corresponding to the columns defined by `parse_dates`) a
s
        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 : str or dict, default 'infer'
        For on-the-fly decompression of on-disk data. If 'infer' and '%s' is
        path-like, then detect compression from the following extensions: '.g
z',
        '.bz2', '.zip', '.xz', or '.zst' (otherwise no compression). If using
        'zip', 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'``} and other
        key-value pairs are forwarded to ``zipfile.ZipFile``, ``gzip.GzipFile`
`,
        ``bz2.BZ2File``, or ``zstandard.ZstdDecompressor``, respectively. As a
n
        example, the following could be passed for Zstandard decompression usi
ng a
        custom compression dictionary:
        ``compression={'method': 'zstd', 'dict_data': my_compression_dict}``.

        .. versionchanged:: 1.4.0 Zstandard support.

    thousands : str, optional
        Thousands separator.
    decimal : str, default '.'
        Character to recognize as decimal point (e.g. use ',' for European dat
a).
    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. Quote
d
        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 begi
nning
        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' bein
g
        treated as the header.
    encoding : str, optional
        Encoding to use for UTF when reading/writing (ex. 'utf-8'). `List of P
ython
        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 a
n
           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, optional, 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 retu
rned.
        If False, then these "bad lines" will be dropped from the DataFrame th
at is
        returned.

        .. deprecated:: 1.3.0
           The ``on_bad_lines`` parameter should be used instead to specify be
havior upon
           encountering a bad line instead.
    warn_bad_lines : bool, optional, 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'} 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

            - 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"``

        .. versionadded:: 1.4.0

    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 pa
irs
        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
```

Exercise 1

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

In [3]:
```
# 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

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

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

In [4]:

```
# df.iloc[] - for more info, see https://pandas.pydata.org/pandas-docs/stable/u
# 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          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
...    ...                ...     ...          ...            ...
32556   27            Private  257302   Assoc-acdm             12
32557   40            Private  154374      HS-grad              9
32558   58            Private  151910      HS-grad              9
32559   22            Private  201490      HS-grad              9
32560   52       Self-emp-inc  287927      HS-grad              9

            marital-status          occupation    relationship    race  \
0            Never-married        Adm-clerical   Not-in-family   White
1       Married-civ-spouse     Exec-managerial         Husband   White
2                 Divorced   Handlers-cleaners   Not-in-family   White
3       Married-civ-spouse   Handlers-cleaners         Husband   Black
4       Married-civ-spouse      Prof-specialty            Wife   Black
...                    ...                 ...             ...     ...
32556   Married-civ-spouse        Tech-support            Wife   White
32557   Married-civ-spouse   Machine-op-inspct         Husband   White
32558              Widowed        Adm-clerical       Unmarried   White
32559        Never-married        Adm-clerical       Own-child   White
32560   Married-civ-spouse     Exec-managerial            Wife   White

           sex  capital-gain  capital-loss  hours-per-week  native-country  \
0         Male          2174             0              40   United-States
1         Male             0             0              13   United-States
2         Male             0             0              40   United-States
3         Male             0             0              40   United-States
4       Female             0             0              40            Cuba
...        ...           ...           ...             ...             ...
32556   Female             0             0              38   United-States
32557     Male             0             0              40   United-States
32558   Female             0             0              40   United-States
32559     Male             0             0              20   United-States
32560   Female         15024             0              40   United-States

      gross-income
0            <=50K
1            <=50K
2            <=50K
3            <=50K
4            <=50K
...            ...
32556        <=50K
32557         >50K
32558        <=50K
32559        <=50K
32560         >50K

[32561 rows x 15 columns]
```

In [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 i

# 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 w
```

```
   age workclass  fnlwgt   education  education-num          marital-status  \
3   53   Private  234721        11th              7      Married-civ-spouse   
4   28   Private  338409   Bachelors             13      Married-civ-spouse   
5   37   Private  284582     Masters             14      Married-civ-spouse   
6   49   Private  160187         9th              5   Married-spouse-absent   

          occupation    relationship    race      sex  capital-gain  \
3  Handlers-cleaners         Husband   Black     Male             0   
4     Prof-specialty            Wife   Black   Female             0   
5    Exec-managerial            Wife   White   Female             0   
6      Other-service   Not-in-family   Black   Female             0   

   capital-loss  hours-per-week  native-country gross-income  
3             0              40   United-States        <=50K  
4             0              40            Cuba        <=50K  
5             0              40   United-States        <=50K  
6             0              16         Jamaica        <=50K  
```

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

In [6]:
```
# df.loc[] - for more info, see https://pandas.pydata.org/pandas-docs/stable/us

print(df.index) # the default index when reading in a file is a range index. In
                 # .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 el

#help(df.set_index)
```

```
RangeIndex(start=0, stop=32561, step=1)
```

In [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
```

3) select rows based on column condition

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

```
#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  \
11   30     State-gov  141297     Bachelors             13   
33   30   Federal-gov   59951  Some-college             10   
59   30       Private  188146       HS-grad              9   
60   30       Private   59496     Bachelors             13   
88   30       Private   54334           9th              5   

         marital-status          occupation    relationship  \
11   Married-civ-spouse      Prof-specialty         Husband   
33   Married-civ-spouse        Adm-clerical       Own-child   
59   Married-civ-spouse   Machine-op-inspct         Husband   
60   Married-civ-spouse               Sales         Husband   
88        Never-married               Sales   Not-in-family   

                   race    sex  capital-gain  capital-loss  hours-per-week  \
11   Asian-Pac-Islander   Male             0             0              40   
33                White   Male             0             0              40   
59                White   Male          5013             0              40   
60                White   Male          2407             0              40   
88                White   Male             0             0              40   

    native-country gross-income  
11           India         >50K  
33   United-States        <=50K  
59   United-States        <=50K  
60   United-States        <=50K  
88   United-States        <=50K  
```

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

In [9]:

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

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

```
#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**

How to merge dataframes?

Merge - info on data points are distributed in multiple files

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

```
hospital1 = {'ID':['ID1','ID2','ID3','ID4','ID5','ID6','ID7'],'col1':[5,8,2,6,
df1 = pd.DataFrame(data=hospital1)
print(df1)

hospital2 = {'ID':['ID2','ID5','ID6','ID10','ID11'],'col3':[12,76,34,98,65],'c
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
```

In [11]:

```
# we are interested in only patients from hospital1
#df_left = df1.merge(df2,how='left',on='ID') # IDs from the left dataframe (df
#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 (
#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
#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
#print(df_outer)
```

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.

In [12]:

```
#df_append = df1.append(df2) # note that rows with ID2, ID5, and ID6  are dupli
#print(df_append)

#df_append = df1.append(df2,ignore_index=True) # note that rows with ID2, ID5,
#print(df_append)

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

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

Exercise 3

In [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_
# Assign the new data frame to df_merge.
# How many rows and columns do we have in df_merge?
```

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

- small toy datasets are ideal to test your code.

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

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

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

Mud card

In [ ]:

```
```