Mudcard

- **What are the best cases for using violin?**
  - You usually want to use a violin plot if you have a categorical/ordinal feature vs. a continuous feature and the categorical/ordinal feature has maybe more than 3-4 categories so overlapping histograms won't work.
- **"There is no objective ground truth for several types of vars, I think that might be something more I would have to think about when trying to tell a story!**
  - While that's certianly true sometimes, it's usually not too difficult to decide the type of a variable.
  - If you are in doubt, try both data types in your ML pipeline and check if it has impact on the performance of your ML model
- **Too many parameters can be played with in the plot function!**
  - YES! And I only had time to show what I believe are the most imnportant ones!
  - Please read the manual to see all the parameters.
  - Your figure quality will greatly improve if you are aware and use the functionalities and you are intentional when you prepare figures
- **Both the box plot and the violin plot are used to visualize categorical v.s. continuous variables, so what should we consider while choosing one of these two types of plots (what is the tradeoff of picking one over the other)?**
  - Preparing visualizations can be subjective. In this case, I'd say that either a box and a violin plot works well, it's up to you to decide which one you like more.
- **I have not used violin plots in other courses, so I am curious about what purpose they hold and when is most appropriate to use.**
  - They are a good alternative for box plots if you want to see the histogram of the continuous feature.
- **I found continuous vs categorical and categorical vs continuous use different visualization, so how do we decided if it is cont vs cate or cate vs cont given two features?**
  - Nope, check the 2x2 matrix again.
  - The same figures should be used for either, the order of the features don't matter.
- **Are we supposed to memorize all specific implementations of the visualizing tools?**
  - The syntax, no. You can always look up the manual.
  - The visualization types, when to use them, when to use e.g., log axes, etc, that's what you need to know.
- **how to determine the appropriate number of bins.**
  - That's dataset specific so just experiment with a few values and see what's best.

- **Was a bit overwhelmed by content - would like more practice with guided creation of plts.**
  - You'll practice it in PS3.

Lecture 5: Data splitting, part 1

Split iid data

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

- describe what the iid assumption is
- apply basic split to iid datasets
- apply k-fold split to iid datasets

The supervised ML pipeline

0. Data collection/manipulation: you might have multiple data sources and/or you might have more data than you need

- you need to be able to read in datasets from various sources (like csv, excel, SQL, parquet, etc)
- you need to be able to filter the columns/rows you need for your ML model
- you need to be able to combine the datasets into one dataframe

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 be 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 or hyperparameter tuning)

- 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)

Split iid data

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

- **describe what the iid assumption is**
- apply basic split to iid datasets
- apply k-fold split to iid datasets

Let's revisit the papaya example from the first lecture!

- **the learner's input:**

    - Domain set $\mathcal{X}$ - a set of objects we wish to label (*all papayas on the island*).
    - The probability distribution over $\mathcal{X}$ is D.
        - D is pretty general
        - D might change as you sample from it
        - there might be multiple distributions you sample from
        - one sample might depend on one or multiple previous samples, etc.
    - Label set $\mathcal{Y}$ - a set of possible labels (*a papaya can be either tasty or not tasty*).
    - There is some correct labeling function $f : \mathcal{X} \rightarrow \mathcal{Y}$.
    - **a training example is then generated by sampling x_i from D, and the label y_i is generated using f.**
    - Training data $S = ((x_1, y_1),...,(x_m,y_m))$ - a finite sequence of pairs from $\mathcal{X}$, $\mathcal{Y}$. This is what the learner has access to (*the data I collected by sampling some papayas*).
        - $X = (x_1,...,x_m)$ is the feature matrix which is usually a 2D matrix, and $Y = (y_1,...,y_m)$ is the target variable which is a vector.

I.I.D. assumption

- **the i.i.d. assumption**: the examples in the training set are independently and identically distributed according to D
    - every x_i is freshly sampled from D and then labelled by f
    - that is, x_i and y_i are picked independently of the other instances
    - S is a window through which the learner gets partial info about D and the labeling function f
    - the larger the sample gets, the more likely it is that D and f are accurately reflected
- examples of not iid data:
    - data generated by time-dependent processes
    - data has group structure (samples collected from e.g., different subjects, experiments, measurement devices)
    - sampling from the distribution changes the properties of the distribution
- we will get back to this later in the term

Quiz 1

Which of these data generation processes or ML problems are not iid?

Split iid data

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

- describe what the iid assumption is
- **apply basic split to iid datasets**
- apply k-fold split to iid datasets

Why do we split the data?

- we want to find the best hyper-parameters of our ML algorithms
  - fit models to training data
  - evaluate each model on validation set
  - we find hyper-parameter values that optimize the validation score
- we want to know how the model will perform on previously unseen data
  - apply our final model on the test set

We need to split the data into three parts!

Splitting strategies for iid data: basic approach

- 60% train, 20% validation, 20% test for small datasets
- 98% train, 1% validation, 1% test for large datasets
  - if you have 1 million points, you still have 10000 points in validation and test which is plenty to assess model performance

Let's work with the adult data!

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

df = pd.read_csv('../data/adult_data.csv')

# let's separate the feature matrix X, and target variable y
y = df['gross-income'] # remember, we want to predict who earns more than 50
X = df.loc[:, df.columns != 'gross-income'] # all other columns are features
print(y)
print(X.head())
```

```
0        <=50K
1        <=50K
2        <=50K
3        <=50K
4        <=50K
          ...
32556    <=50K
32557     >50K
32558    <=50K
32559    <=50K
32560     >50K
Name: gross-income, Length: 32561, dtype: object
   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
0          2174             0              40   United-States
1             0             0              13   United-States
2             0             0              40   United-States
3             0             0              40   United-States
4             0             0              40            Cuba
```

In [2]:
```
# all sklearn transformers and models accept polars dataframes!
help(train_test_split)
```

```
Help on function train_test_split in module sklearn.model_selection._split:

train_test_split(*arrays, test_size=None, train_size=None, random_state=Non
e, shuffle=True, stratify=None)
    Split arrays or matrices into random train and test subsets.

    Quick utility that wraps input validation,
    ``next(ShuffleSplit().split(X, y))``, and application to input data
    into a single call for splitting (and optionally subsampling) data into
a
    one-liner.

    Read more in the :ref:`User Guide <cross_validation>`.

    Parameters
    ----------
    *arrays : sequence of indexables with same length / shape[0]
        Allowed inputs are lists, numpy arrays, scipy-sparse
        matrices or pandas dataframes.

    test_size : float or int, default=None
        If float, should be between 0.0 and 1.0 and represent the proportion
        of the dataset to include in the test split. If int, represents the
        absolute number of test samples. If None, the value is set to the
        complement of the train size. If ``train_size`` is also None, it wil
l
        be set to 0.25.

    train_size : float or int, default=None
        If float, should be between 0.0 and 1.0 and represent the
        proportion of the dataset to include in the train split. If
        int, represents the absolute number of train samples. If None,
        the value is automatically set to the complement of the test size.

    random_state : int, RandomState instance or None, default=None
        Controls the shuffling applied to the data before applying the spli
t.
        Pass an int for reproducible output across multiple function calls.
        See :term:`Glossary <random_state>`.

    shuffle : bool, default=True
        Whether or not to shuffle the data before splitting. If shuffle=Fals
e
        then stratify must be None.

    stratify : array-like, default=None
        If not None, data is split in a stratified fashion, using this as
        the class labels.
        Read more in the :ref:`User Guide <stratification>`.

    Returns
    -------
    splitting : list, length=2 * len(arrays)
        List containing train-test split of inputs.

        .. versionadded:: 0.16
```

```
            If the input is sparse, the output will be a
            ``scipy.sparse.csr_matrix``. Else, output type is the same as th
e
            input type.

    Examples
    --------
    >>> import numpy as np
    >>> from sklearn.model_selection import train_test_split
    >>> X, y = np.arange(10).reshape((5, 2)), range(5)
    >>> X
    array([[0, 1],
           [2, 3],
           [4, 5],
           [6, 7],
           [8, 9]])
    >>> list(y)
    [0, 1, 2, 3, 4]

    >>> X_train, X_test, y_train, y_test = train_test_split(
    ...     X, y, test_size=0.33, random_state=42)
    ...
    >>> X_train
    array([[4, 5],
           [0, 1],
           [6, 7]])
    >>> y_train
    [2, 0, 3]
    >>> X_test
    array([[2, 3],
           [8, 9]])
    >>> y_test
    [1, 4]

    >>> train_test_split(y, shuffle=False)
    [[0, 1, 2], [3, 4]]
```

In [3]:

```
random_state = 42

# first split to separate out the training set
X_train, X_other, y_train, y_other = train_test_split(X,y,\
                    train_size = 0.6,random_state = random_state)
print('training set:',X_train.shape, y_train.shape) # 60% of points are in t
print(X_other.shape, y_other.shape) # 40% of points are in other

# second split to separate out the validation and test sets
X_val, X_test, y_val, y_test = train_test_split(X_other,y_other,\
                    train_size = 0.5,random_state = random_state)
print('validation set:',X_val.shape, y_val.shape) # 20% of points are in val
print('test set:',X_test.shape, y_test.shape) # 20% of points are in test

print(X_train.head())
```

```
training set: (19536, 14) (19536,)
(13025, 14) (13025,)
validation set: (6512, 14) (6512,)
test set: (6513, 14) (6513,)
       age   workclass  fnlwgt      education  education-num  \
25823   31     Private   87418      Assoc-voc             11
10274   41     Private  121718   Some-college             10
27652   61     Private   79827        HS-grad              9
13941   33   State-gov  156015      Bachelors             13
31384   38     Private  167882   Some-college             10

            marital-status        occupation     relationship    race  \
25823   Married-civ-spouse   Exec-managerial          Husband   White
10274   Married-civ-spouse      Craft-repair          Husband   White
27652   Married-civ-spouse   Exec-managerial          Husband   White
13941   Married-civ-spouse   Exec-managerial          Husband   White
31384              Widowed     Other-service   Other-relative   Black

           sex  capital-gain  capital-loss  hours-per-week  native-country
25823     Male             0             0              40   United-States
10274     Male             0             0              40           Italy
27652     Male             0             0              50   United-States
13941     Male             0             0              40   United-States
31384   Female             0             0              45           Haiti
```

Randomness due to splitting

- the model performance, validation and test scores will change depending on which points are in train, val, test
  - inherent randomness or uncertainty of the ML pipeline
- change the random state a couple of times and repeat the whole ML pipeline to assess how much the random splitting affects your test score
  - you would expect a similar uncertainty when the model is deployed

Quiz 2

What's the second train_test_split line if you want to end up with 60-20-20 in train-val-test? Print out the sizes of X_train, X_val, X_test to verify!

In [4]:
```
X_other, X_test, y_other, y_test = train_test_split(X,y,\
                    train_size = 0.8,random_state=random_state)
# add your line below and choose the correct solution from canvas
```

Split iid data

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

- describe what the iid assumption is
- apply basic split to iid datasets

- **apply k-fold split to iid datasets**

Other splitting strategy for iid data: k-fold splitting

No description has been provided for this image

In [5]:
```
from sklearn.model_selection import KFold
help(KFold)
```

```
Help on class KFold in module sklearn.model_selection._split:

class KFold(_UnsupportedGroupCVMixin, _BaseKFold)
 |  KFold(n_splits=5, *, shuffle=False, random_state=None)
 |
 |  K-Fold cross-validator.
 |
 |  Provides train/test indices to split data in train/test sets. Split
 |  dataset into k consecutive folds (without shuffling by default).
 |
 |  Each fold is then used once as a validation while the k - 1 remaining
 |  folds form the training set.
 |
 |  Read more in the :ref:`User Guide <k_fold>`.
 |
 |  For visualisation of cross-validation behaviour and
 |  comparison between common scikit-learn split methods
 |  refer to :ref:`sphx_glr_auto_examples_model_selection_plot_cv_indices.py
`
 |
 |  Parameters
 |  ----------
 |  n_splits : int, default=5
 |      Number of folds. Must be at least 2.
 |
 |      .. versionchanged:: 0.22
 |          ``n_splits`` default value changed from 3 to 5.
 |
 |  shuffle : bool, default=False
 |      Whether to shuffle the data before splitting into batches.
 |      Note that the samples within each split will not be shuffled.
 |
 |  random_state : int, RandomState instance or None, default=None
 |      When `shuffle` is True, `random_state` affects the ordering of the
 |      indices, which controls the randomness of each fold. Otherwise, this
 |      parameter has no effect.
 |      Pass an int for reproducible output across multiple function calls.
 |      See :term:`Glossary <random_state>`.
 |
 |  Examples
 |  --------
 |  >>> import numpy as np
 |  >>> from sklearn.model_selection import KFold
 |  >>> X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
 |  >>> y = np.array([1, 2, 3, 4])
 |  >>> kf = KFold(n_splits=2)
 |  >>> kf.get_n_splits(X)
 |  2
 |  >>> print(kf)
 |  KFold(n_splits=2, random_state=None, shuffle=False)
 |  >>> for i, (train_index, test_index) in enumerate(kf.split(X)):
 |  ...     print(f"Fold {i}:")
 |  ...     print(f"  Train: index={train_index}")
 |  ...     print(f"  Test:  index={test_index}")
 |  Fold 0:
 |    Train: index=[2 3]
```

```
 |    Test:  index=[0 1]
 |  Fold 1:
 |    Train: index=[0 1]
 |    Test:  index=[2 3]
 |
 |  Notes
 |  -----
 |  The first ``n_samples % n_splits`` folds have size
 |  ``n_samples // n_splits + 1``, other folds have size
 |  ``n_samples // n_splits``, where ``n_samples`` is the number of samples.
 |
 |  Randomized CV splitters may return different results for each call of
 |  split. You can make the results identical by setting `random_state`
 |  to an integer.
 |
 |  See Also
 |  --------
 |  StratifiedKFold : Takes class information into account to avoid building
 |      folds with imbalanced class distributions (for binary or multiclass
 |      classification tasks).
 |
 |  GroupKFold : K-fold iterator variant with non-overlapping groups.
 |
 |  RepeatedKFold : Repeats K-Fold n times.
 |
 |  Method resolution order:
 |      KFold
 |      _UnsupportedGroupCVMixin
 |      _BaseKFold
 |      BaseCrossValidator
 |      sklearn.utils._metadata_requests._MetadataRequester
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, n_splits=5, *, shuffle=False, random_state=None)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __abstractmethods__ = frozenset()
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from _UnsupportedGroupCVMixin:
 |
 |  split(self, X, y=None, groups=None)
 |      Generate indices to split data into training and test set.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          Training data, where `n_samples` is the number of samples
 |          and `n_features` is the number of features.
 |
 |      y : array-like of shape (n_samples,)
```

```
 |          The target variable for supervised learning problems.
 |
 |      groups : object
 |          Always ignored, exists for compatibility.
 |
 |      Yields
 |      ------
 |      train : ndarray
 |          The training set indices for that split.
 |
 |      test : ndarray
 |          The testing set indices for that split.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from _UnsupportedGroupCVMixin:
 |
 |  __dict__
 |      dictionary for instance variables
 |
 |  __weakref__
 |      list of weak references to the object
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from _BaseKFold:
 |
 |  get_n_splits(self, X=None, y=None, groups=None)
 |      Returns the number of splitting iterations in the cross-validator.
 |
 |      Parameters
 |      ----------
 |      X : object
 |          Always ignored, exists for compatibility.
 |
 |      y : object
 |          Always ignored, exists for compatibility.
 |
 |      groups : object
 |          Always ignored, exists for compatibility.
 |
 |      Returns
 |      -------
 |      n_splits : int
 |          Returns the number of splitting iterations in the cross-validato
r.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from BaseCrossValidator:
 |
 |  __repr__(self)
 |      Return repr(self).
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.utils._metadata_requests._MetadataRequest
er:
 |
 |  get_metadata_routing(self)
```

```
 |      Get metadata routing of this object.
 |      
 |      Please check :ref:`User Guide <metadata_routing>` on how the routing
 |      mechanism works.
 |      
 |      Returns
 |      -------
 |      routing : MetadataRequest
 |          A :class:`~sklearn.utils.metadata_routing.MetadataRequest` encap
sulating
 |          routing information.
 |  
 |  ----------------------------------------------------------------------
 |  Class methods inherited from sklearn.utils._metadata_requests._MetadataR
equester:
 |  
 |  __init_subclass__(**kwargs)
 |      Set the ``set_{method}_request`` methods.
 |      
 |      This uses PEP-487 [1]_ to set the ``set_{method}_request`` methods.
It
 |      looks for the information available in the set default values which
are
 |      set using ``__metadata_request__*`` class attributes, or inferred
 |      from method signatures.
 |      
 |      The ``__metadata_request__*`` class attributes are used when a metho
d
 |      does not explicitly accept a metadata through its arguments or if th
e
 |      developer would like to specify a request value for those metadata
 |      which are different from the default ``None``.
 |      
 |      References
 |      ----------
 |      .. [1] https://www.python.org/dev/peps/pep-0487
```

In [6]:

```python
random_state =42

# first split to separate out the test set
X_other, X_test, y_other, y_test = train_test_split(X,y,test_size = 0.2,ranc
print(X_other.shape,y_other.shape)
print('test set:',X_test.shape,y_test.shape)

# do KFold split on other
kf = KFold(n_splits=5,shuffle=True,random_state=random_state)
for train_index, val_index in kf.split(X_other,y_other):
    X_train = X_other.iloc[train_index]
    y_train = y_other.iloc[train_index]
    X_val = X_other.iloc[val_index]
    y_val = y_other.iloc[val_index]
    print('   training set:',X_train.shape, y_train.shape)
    print('   validation set:',X_val.shape, y_val.shape)
    # the validation set contains different points in each iteration
```

```
    print(X_val[['age','workclass','education']].head())
```

```
(26048, 14) (26048,)
test set: (6513, 14) (6513,)
   training set: (20838, 14) (20838,)
   validation set: (5210, 14) (5210,)
       age   workclass      education
27240   38     Private      Bachelors
4       28     Private      Bachelors
14242   34     Private        HS-grad
16461   58     Private   Some-college
2209    49   Local-gov        HS-grad
   training set: (20838, 14) (20838,)
   validation set: (5210, 14) (5210,)
       age   workclass   education
5514    33   Local-gov   Bachelors
32240   21     Private   Assoc-voc
8615    33     Private        10th
7743    20     Private     HS-grad
20097   39     Private   Assoc-voc
   training set: (20838, 14) (20838,)
   validation set: (5210, 14) (5210,)
       age          workclass      education
9876    27            Private   Some-college
5455    44            Private      Bachelors
29805   62   Self-emp-not-inc      Bachelors
15081   20            Private        HS-grad
13770   40            Private     Assoc-acdm
   training set: (20839, 14) (20839,)
   validation set: (5209, 14) (5209,)
       age          workclass      education
19777   36            Private      Assoc-voc
10781   58   Self-emp-not-inc            9th
9747    24            Private      Bachelors
327     43            Private   Some-college
24431   25            Private        HS-grad
   training set: (20839, 14) (20839,)
   validation set: (5209, 14) (5209,)
       age workclass     education
17203   33   Private       HS-grad
12114   36   Private   Prof-school
231     41   Private       HS-grad
3272    30   Private          10th
26009   19   Private          11th
```

How many splits should I create?

- tough question, 3-5 is most common
- if you do n splits, n models will be trained, so the larger the n, the most computationally intensive it will be to train the models
- KFold is usually better suited to small datasets

- KFold is good to estimate uncertainty due to random splitting of train and val, but it is not perfect
  - the test set remains the same

Why shuffling iid data is important?

- by default, data is not shuffled by Kfold which can introduce errors!

No description has been provided for this image

Mudcard

In [ ]:

```

```