Mudcard

- **Could you provide more tools to test the correlation between two input? For example, I am interested in the correlation between age and gross-income. Should I do a simple Pearson correlation?**
  - Yes, we will cover quantitative ways to asses correlation between features during week4
- **I am still unsure about how to assess the number of bins for some of the different visualizations.**
  - It's by trial and error.
  - there is no approach that will always give you the correct number of bins under all circumstances
  - try a couple of values and be critical about your figure
- **Are 3 dimensional/video visualization tools easily available?**
- **What do you think of 3D visualizations? They can be effective (and abused) depending on the circumstances.**
- **Are there any 3D visualization ways we can use? And when should we use them?**
  - this is just my subjective opinion but I don't like 3D figures
  - it's too distractive in my opinion
- **what are other scales beyond logarithmic that are useful for axes?**
  - if you use log instead of linear axis when appropriate, you are way ahead of the curve :)
- **How to know what transform method should we choose when plotting a histogram, like 'log'?**
  - trial and error
  - experient until you are happy with the figure
- **What is the difference between bar plot and histogram?**
- **What is the difference between a histogram and a bar plot? Because from the plot, both of them are made up with bars.**
  - the bars can be shuffled in a bar plot, it doesn't matter how you order the counts of the categories
  - the bars in the histogram correspond to bins defined over the range of a continuous feature
    - the height of the bars tells you how many points fall into each bin
  - the bars of a histogram cannot be shuffled around
- **Is there 'a best choice' of plot types for visualization? If not, do I need to include all the appropriate plots for a set of data when writing reports?**
  - no, definitely not all plots
  - a report or a presentation is a distilled version of your work
  - you will work on a project for months if not years and the report is a couple of pages, the presentation is maybe 5 to 30 minutes
  - you will do a lot more work than what goes into the report and presentation
- **We spent a lot of time talking about visualization for EDA, which is just for our own understanding, but at what point do we stop the EDA and just get to the analysis. In**

other words, when do we know what we've done is good enough?
    - when will you be 100% sure that you absolutely and perfectly understand all aspects of your data?
    - never
    - but you have deadlines so at some point you need to move on and hope you have a sufficiently good understanding of your data
- **I was unsure about what exactly the log=True argument did**
    - run the code with and without it to figure this out
- **How much of an emphasis will there be on data visualization in the course and will we be touching libraries outside of the python programming language like D3?**
    - unfortunately we only have time for one lecture
    - we won't work with anything outside of python
    - and we won't even have time to learn all the visualization packages within python
    - as I said, data visualization could be a separate course
- **What if we have ordinary data in our dataset? Should we treat them as categorical or continuous variables when visualizing figures?**
    - that's exactly the point of Georgie charts I showed during the last lecture
    - it's categorical but you need to make sure that the categories are displayed in the correct order in your figure (like the months or dates, etc.).
- **When we get NA or "?" as a part of our data and its fraction is significant in the data set, what is the standard practice in addressing how it affects our data set?**
    - week 4 and week 9 :)
    - we will cover several techniques
    - there is no standard
    - you need to decide which approach is best given your problem, computational resources, etc.

Split iid and non-iid data

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

- apply a basic split and a k-fold split to iid datasets
- apply stratified splits to imbalanced data
- split non-iid data based on group ID or time

The supervised ML pipeline

The goal: Use the training data (X and y) to develop a model which can accurately predict the target variable (y_new') for previously unseen data (X_new).

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)

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!

How should we split the data into train/validation/test?

- data is **Independent and Identically Distributed** (iid)
    - all samples stem from the same generative process and the generative process is assumed to have no memory of past generated samples
    - identify cats and dogs on images
    - predict the house price
    - predict if someone's salary is above or below 50k
- 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)

Split iid and non-iid data

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

- **apply a basic split and a k-fold split to iid datasets**
- apply stratified splits to imbalanced data
- split non-iid data based on group ID or time

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
from sklearn.model_selection import train_test_split

df = pd.read_csv('data/adult_test.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 50k or
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.
          ...
16276     <=50K.
16277     <=50K.
16278     <=50K.
16279     <=50K.
16280      >50K.
Name: gross-income, Length: 16281, dtype: object
   age   workclass  fnlwgt      education  education-num       marital-status  \
0   25     Private  226802           11th              7        Never-married
1   38     Private   89814        HS-grad              9   Married-civ-spouse
2   28   Local-gov  336951     Assoc-acdm             12   Married-civ-spouse
3   44     Private  160323   Some-college             10   Married-civ-spouse
4   18           ?  103497   Some-college             10        Never-married

           occupation relationship    race      sex  capital-gain  \
0   Machine-op-inspct    Own-child   Black     Male             0
1     Farming-fishing      Husband   White     Male             0
2     Protective-serv      Husband   White     Male             0
3   Machine-op-inspct      Husband   Black     Male          7688
4                   ?    Own-child   White   Female             0

   capital-loss  hours-per-week  native-country
0             0              40   United-States
1             0              50   United-States
2             0              40   United-States
3             0              40   United-States
4             0              30   United-States
```

In [2]:

```
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=None, sh
uffle=True, stratify=None)
    Split arrays or matrices into random train and test subsets

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

    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 will
        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 split.
    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=False
    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 the
        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]:

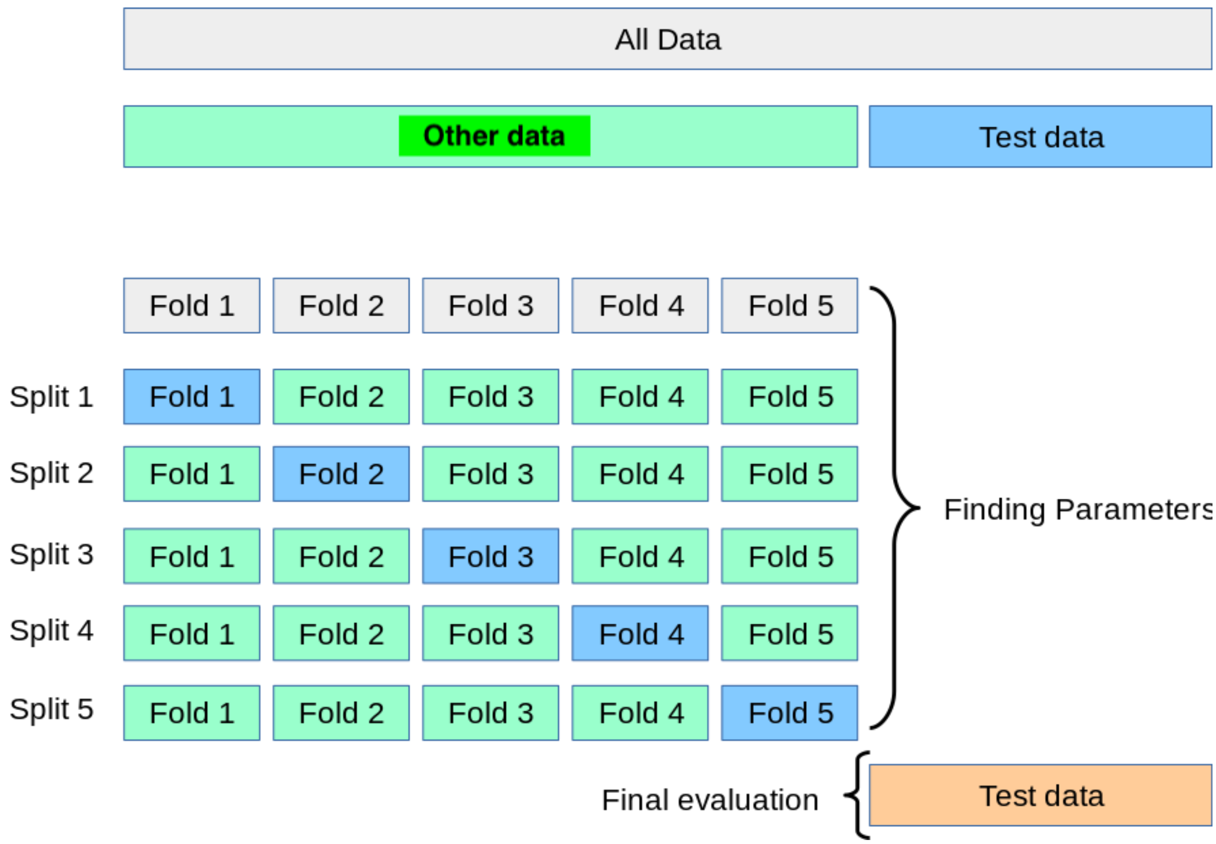
```
random_state = 42

# 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,rando
print('training set:',X_train.shape, y_train.shape) # 60% of points are in train
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
print('validation set:',X_val.shape, y_val.shape) # 20% of points are in validat
print('test set:',X_test.shape, y_test.shape) # 20% of points are in test
```

```
training set: (9768, 14) (9768,)
(6513, 14) (6513,)
validation set: (3256, 14) (3256,)
test set: (3257, 14) (3257,)
```

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

Splitting strategies for iid data: k-fold splitting



In [4]:

```
from sklearn.model_selection import KFold
help(KFold)
```

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

class KFold(_BaseKFold)
 |  KFold(n_splits=5, *, shuffle=False, random_state=None)
 |  
 |  K-Folds 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>`.
 |  
 |  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 train_index, test_index in kf.split(X):
 |  ...     print("TRAIN:", train_index, "TEST:", test_index)
 |  ...     X_train, X_test = X[train_index], X[test_index]
 |  ...     y_train, y_test = y[train_index], y[test_index]
 |  TRAIN: [2 3] TEST: [0 1]
 |  TRAIN: [0 1] TEST: [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 group 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
 |      _BaseKFold
 |      BaseCrossValidator
 |      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 _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-validator.
 |  
 |  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,), default=None
 |          The target variable for supervised learning problems.
 |      
 |      groups : array-like of shape (n_samples,), default=None
 |          Group labels for the samples used while splitting the dataset into
 |          train/test set.
 |      
 |      Yields
 |      ------
 |      train : ndarray
 |          The training set indices for that split.
 |      
 |      test : ndarray
 |          The testing set indices for that split.
 |  
 |  ----------------------------------------------------------------------
 |  Methods inherited from BaseCrossValidator:
 |  
 |  __repr__(self)
 |      Return repr(self).
 |  
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from BaseCrossValidator:
 |  
 |  __dict__
 |      dictionary for instance variables (if defined)
 |  
 |  __weakref__
 |      list of weak references to the object (if defined)
```

In [5]:

```
random_state = 42
```

```python
# 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,random_s
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())
```

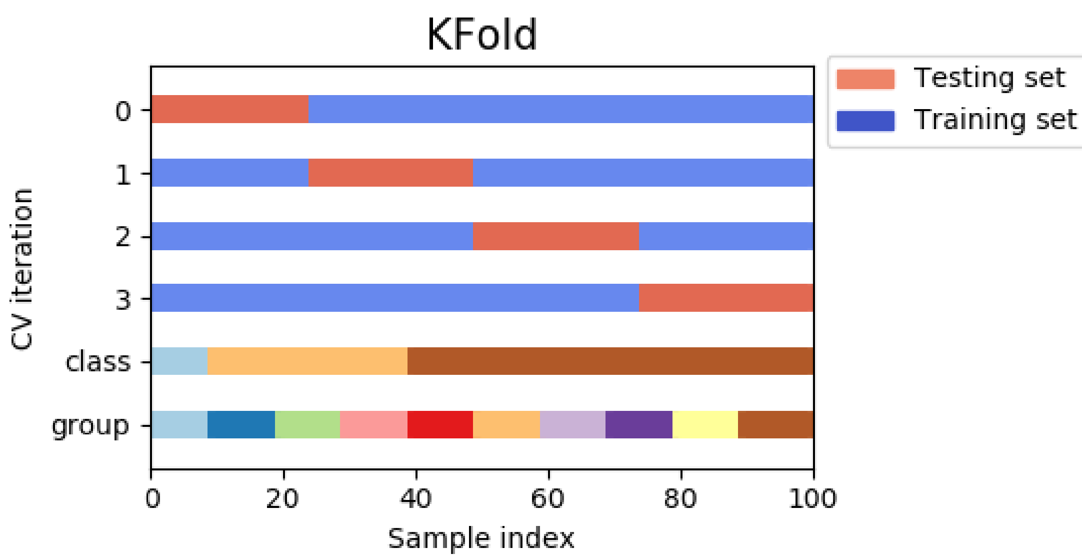
```
(13024, 14) (13024,)
test set: (3257, 14) (3257,)
   training set: (10419, 14) (10419,)
   validation set: (2605, 14) (2605,)
       age           workclass      education
9850    59             Private   Some-college
103     58    Self-emp-not-inc            9th
1383    45             Private        HS-grad
11034   49    Self-emp-not-inc      Bachelors
14876   59    Self-emp-not-inc      Bachelors
   training set: (10419, 14) (10419,)
   validation set: (2605, 14) (2605,)
       age     workclass      education
13384   60   Federal-gov      Bachelors
8471    20       Private        HS-grad
13406   21             ?   Some-college
13394   35       Private        HS-grad
15123   38       Private   Some-college
   training set: (10419, 14) (10419,)
   validation set: (2605, 14) (2605,)
       age     workclass      education
647     60             ?      Bachelors
9314    26       Private   Some-college
14499   52       Private        HS-grad
7332    53   Federal-gov     Assoc-acdm
12523   21       Private           10th
   training set: (10419, 14) (10419,)
   validation set: (2605, 14) (2605,)
       age workclass      education
5294    53   Private        HS-grad
3481    41   Private        HS-grad
7671    49   Private   Some-college
11055   39   Private      Bachelors
12751   18         ?           12th
   training set: (10420, 14) (10420,)
   validation set: (2604, 14) (2604,)
       age      workclass      education
4265    23              ?           10th
5290    23        Private        HS-grad
1157    56   Self-emp-inc    Prof-school
12344   18        Private           11th
13683   55        Private        HS-grad
```

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!



Quiz 1

Split the adult dataset into 70% train, 20% validation, and 10% test! How many points do we have in train, validation, test? Give your answer in the following format:

i, j, k

where i is the number of points in train, j is the number of points in validation, and k is the number of points in test.

Split iid and non-iid data

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

- apply a basic split and a k-fold split to iid datasets
- **apply stratified splits to imbalanced data**
- split non-iid data based on group ID or time

Imbalanced data

- imbalanced data: only a small fraction of the points are in one of the classes, usually ~5% or less but there is no hard limit here
- examples:
    - people visit a bank's website. do they sign up for a new credit card?
        - most customers just browse and leave the page
        - usually 1% or less of the customers get a credit card (class 1), the rest leaves the page without signing up (class 0).
    - fraud detection
        - only a tiny fraction of credit card payments are fraudulent
    - rare disease diagnosis
- the issue with imbalanced data:
    - if you apply train_test_split or KFold, you might not have class 1 points in one of your sets by chance
    - this is what we need to fix

Solution: stratified splits

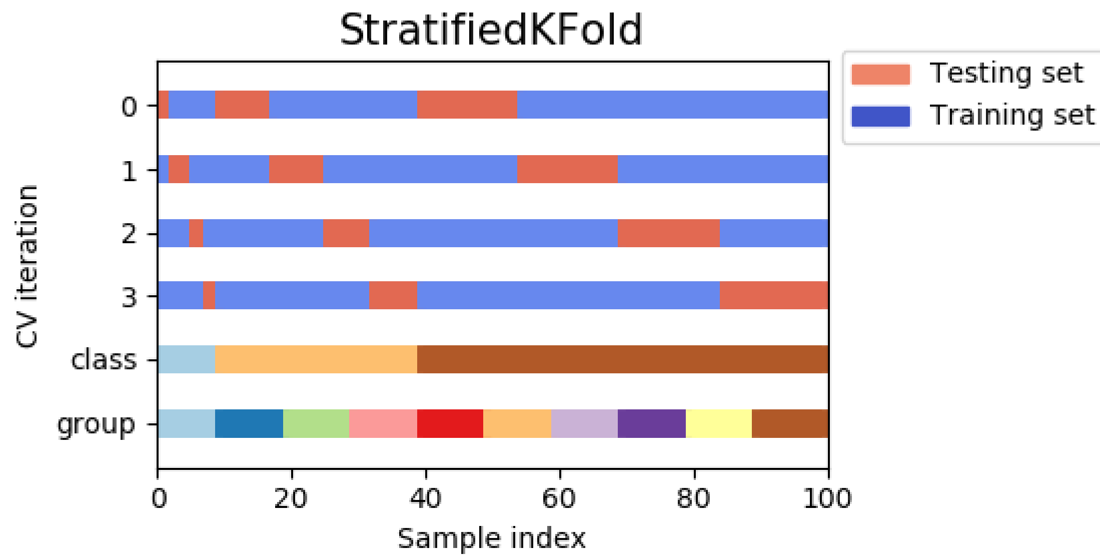
In [6]:

```
random_state = 137

X_train, X_other, y_train, y_other = train_test_split(X,y,train_size = 0.6,rando
X_val, X_test, y_val, y_test = train_test_split(X_other,y_other,train_size = 0.5

print('**balance without stratification:**')
# a variation on the order of 1% which would be too much for imbalanced data!
print(y_train.value_counts(normalize=True))
print(y_val.value_counts(normalize=True))
print(y_test.value_counts(normalize=True))

print()
X_train, X_other, y_train, y_other = train_test_split(X,y,train_size = 0.6,strat
X_val, X_test, y_val, y_test = train_test_split(X_other,y_other,train_size = 0.5
print('**balance with stratification:**')
# very little variation (in the 4th decimal point only) which is important if th
print(y_train.value_counts(normalize=True))
print(y_val.value_counts(normalize=True))
print(y_test.value_counts(normalize=True))
```

```
**balance without stratification:**
 <=50K.     0.76423
 >50K.      0.23577
Name: gross-income, dtype: float64
 <=50K.     0.770885
 >50K.      0.229115
Name: gross-income, dtype: float64
 <=50K.     0.755296
 >50K.      0.244704
Name: gross-income, dtype: float64

**balance with stratification:**
 <=50K.     0.763821
```

```
 >50K.      0.236179
Name: gross-income, dtype: float64
 <=50K.     0.763821
 >50K.      0.236179
Name: gross-income, dtype: float64
 <=50K.     0.763586
 >50K.      0.236414
Name: gross-income, dtype: float64
```

Stratified folds



In [7]:

```python
from sklearn.model_selection import StratifiedKFold
help(StratifiedKFold)
```

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

class StratifiedKFold(_BaseKFold)
 |  StratifiedKFold(n_splits=5, *, shuffle=False, random_state=None)
 |
 |  Stratified K-Folds cross-validator.
 |
 |  Provides train/test indices to split data in train/test sets.
 |
 |  This cross-validation object is a variation of KFold that returns
 |  stratified folds. The folds are made by preserving the percentage of
 |  samples for each class.
 |
 |  Read more in the :ref:`User Guide <stratified_k_fold>`.
 |
 |  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 each class's samples 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 for each class.
 |      Otherwise, leave `random_state` as `None`.
 |      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 StratifiedKFold
 |  >>> X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
 |  >>> y = np.array([0, 0, 1, 1])
 |  >>> skf = StratifiedKFold(n_splits=2)
 |  >>> skf.get_n_splits(X, y)
 |  2
 |  >>> print(skf)
 |  StratifiedKFold(n_splits=2, random_state=None, shuffle=False)
 |  >>> for train_index, test_index in skf.split(X, y):
 |  ...     print("TRAIN:", train_index, "TEST:", test_index)
 |  ...     X_train, X_test = X[train_index], X[test_index]
 |  ...     y_train, y_test = y[train_index], y[test_index]
 |  TRAIN: [1 3] TEST: [0 2]
 |  TRAIN: [0 2] TEST: [1 3]
 |  
 |  Notes
 |  -----
 |  The implementation is designed to:
 |  
 |  * Generate test sets such that all contain the same distribution of
 |    classes, or as close as possible.
 |  * Be invariant to class label: relabelling ``y = ["Happy", "Sad"]`` to
 |    ``y = [1, 0]`` should not change the indices generated.
 |  * Preserve order dependencies in the dataset ordering, when
 |    ``shuffle=False``: all samples from class k in some test set were
 |    contiguous in y, or separated in y by samples from classes other than k.
 |  * Generate test sets where the smallest and largest differ by at most one
 |    sample.
 |  
 |  .. versionchanged:: 0.22
 |      The previous implementation did not follow the last constraint.
 |  
 |  See Also
 |  --------
 |  RepeatedStratifiedKFold : Repeats Stratified K-Fold n times.
 |  
 |  Method resolution order:
 |      StratifiedKFold
 |      _BaseKFold
 |      BaseCrossValidator
 |      builtins.object
 |  
 |  Methods defined here:
 |  
 |  __init__(self, n_splits=5, *, shuffle=False, random_state=None)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |  
 |  split(self, X, y, 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.
 |
 |          Note that providing ``y`` is sufficient to generate the splits and
 |          hence ``np.zeros(n_samples)`` may be used as a placeholder for
 |          ``X`` instead of actual training data.
 |
 |      y : array-like of shape (n_samples,)
 |          The target variable for supervised learning problems.
 |          Stratification is done based on the y labels.
 |
 |      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.
 |
 |      Notes
 |      -----
 |      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.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __abstractmethods__ = frozenset()
 |
 |  ----------------------------------------------------------------------
 |  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-validator.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from BaseCrossValidator:
```

```
 |
 |  __repr__(self)
 |      Return repr(self).
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from BaseCrossValidator:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

In [8]:

```
# what we did before: variance in balance on the order of 1%
random_state = 42

X_other, X_test, y_other, y_test = train_test_split(X,y,test_size = 0.2,random_s
print('test balance:',y_test.value_counts(normalize=True))

# 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('train balance:')
    print(y_train.value_counts(normalize=True))
    print('val balance:')
    print(y_val.value_counts(normalize=True))
```

```
test balance:  <=50K.    0.770648
 >50K.     0.229352
Name: gross-income, dtype: float64
train balance:
 <=50K.    0.760726
 >50K.     0.239274
Name: gross-income, dtype: float64
val balance:
 <=50K.    0.76737
 >50K.     0.23263
Name: gross-income, dtype: float64
train balance:
 <=50K.    0.761781
 >50K.     0.238219
Name: gross-income, dtype: float64
val balance:
 <=50K.    0.763148
 >50K.     0.236852
Name: gross-income, dtype: float64
train balance:
 <=50K.    0.761685
 >50K.     0.238315
Name: gross-income, dtype: float64
val balance:
 <=50K.    0.763532
 >50K.     0.236468
Name: gross-income, dtype: float64
train balance:
```

```
 <=50K.    0.761014
 >50K.     0.238986
Name: gross-income, dtype: float64
val balance:
 <=50K.    0.766219
 >50K.     0.233781
Name: gross-income, dtype: float64
train balance:
 <=50K.    0.765067
 >50K.     0.234933
Name: gross-income, dtype: float64
val balance:
 <=50K.    0.75
 >50K.     0.25
Name: gross-income, dtype: float64
```

In [9]:

```python
# stratified K Fold: variation in balance is very small (4th decimal point)
random_state = 42

# stratified train-test split
X_other, X_test, y_other, y_test = train_test_split(X,y,test_size = 0.2,stratify
print('test balance:',y_test.value_counts(normalize=True))

# do StratifiedKFold split on other
kf = StratifiedKFold(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('train balance:')
    print(y_train.value_counts(normalize=True))
    print('val balance:')
    print(y_val.value_counts(normalize=True))
```

```
test balance:  <=50K.    0.763893
 >50K.     0.236107
Name: gross-income, dtype: float64
train balance:
 <=50K.    0.763701
 >50K.     0.236299
Name: gross-income, dtype: float64
val balance:
 <=50K.    0.763916
 >50K.     0.236084
Name: gross-income, dtype: float64
train balance:
 <=50K.    0.763701
 >50K.     0.236299
Name: gross-income, dtype: float64
val balance:
 <=50K.    0.763916
 >50K.     0.236084
Name: gross-income, dtype: float64
train balance:
 <=50K.    0.763797
 >50K.     0.236203
Name: gross-income, dtype: float64
val balance:
 <=50K.    0.763532
```

```
 >50K.     0.236468
Name: gross-income, dtype: float64
train balance:
 <=50K.    0.763797
 >50K.     0.236203
Name: gross-income, dtype: float64
val balance:
 <=50K.    0.763532
 >50K.     0.236468
Name: gross-income, dtype: float64
train balance:
 <=50K.    0.763724
 >50K.     0.236276
Name: gross-income, dtype: float64
val balance:
 <=50K.    0.763825
 >50K.     0.236175
Name: gross-income, dtype: float64
```

Quiz 2

Given the labels below, what are the balances of each class?

y = [0,0,0,2,2,0,1,2,0,1]

Split iid and non-iid data

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

- apply a basic split and a k-fold split to iid datasets
- apply stratified splits to imbalanced data
- **split non-iid data based on group ID or time**

Examples of non-iid data

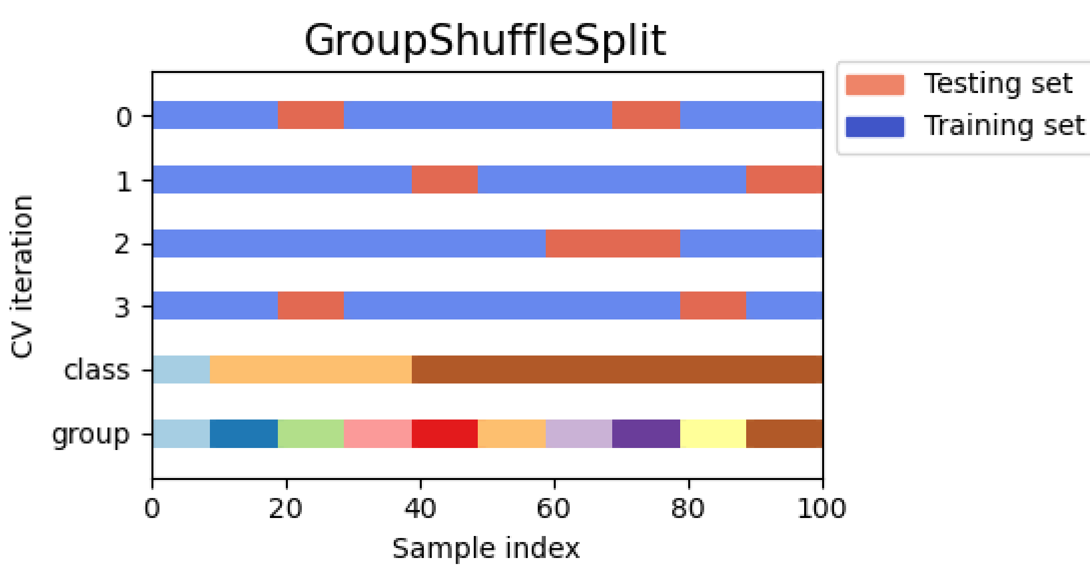
- if there is any sort of time or group structure in your data, it is likely non-iid
    - group structure:
        - each point is someone's visit to the ER and some people visited the ER multiple times
        - each point is stats of a youtube video and the stats are collected weekly, one of the stats is whether it is featured
        - each point is a customer's visit to CVS and customers tend to return regularly
    - time structure
        - each point is the stocks price at a given time
        - eahc point is a person's health or activity status

Ask yourself these questions!

- What is the intended use of the model? What is it supposed to do/predict?

- What data do you have available at the time of prediction?
- Your split must mimic the intended use of the model only then will you accurately estimate how well the model will perform on previously unseen points (generalization error).
- two examples:
  - if you want to predict the outcome of a new patient's visit to the ER:
    - your test score must be based on patients not included in training and validation
    - your validation score must be based on patients not included in training
    - points of one patient should not be distributed over multiple sets because your generalization error will be off
  - a youtube video was released 4 weeks ago and you want to predict if it will be featured a week from now, your training data should only contain info that will available upon predictions (stuff you know 4 weeks after release)
    - split data based on youtube vid ID
    - use info that's available 4 weeks after release
    - your classification label will be whether it was featured or not 5 weeks after release

Group-based split: GroupShuffleSplit



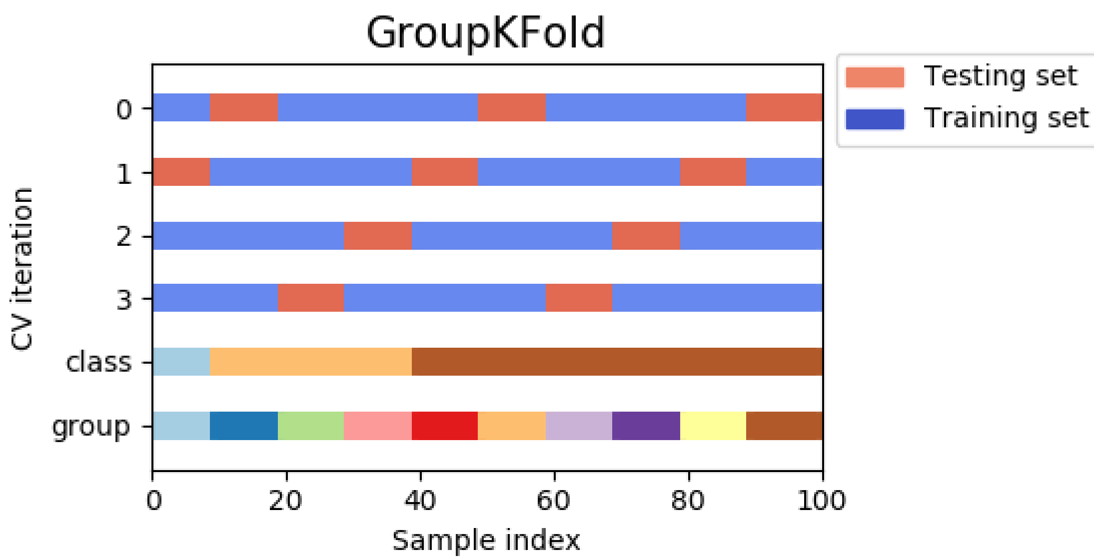
In [10]:

```
import numpy as np
from sklearn.model_selection import GroupShuffleSplit
X = np.ones(shape=(8, 2))
y = np.ones(shape=(8, 1))
groups = np.array([1, 1, 2, 2, 2, 3, 3, 3])

gss = GroupShuffleSplit(n_splits=10, train_size=.8, random_state=42)

for train_idx, test_idx in gss.split(X, y, groups):
    print("TRAIN:", train_idx, "TEST:", test_idx)
```

```
TRAIN: [2 3 4 5 6 7] TEST: [0 1]
TRAIN: [0 1 5 6 7] TEST: [2 3 4]
TRAIN: [2 3 4 5 6 7] TEST: [0 1]
TRAIN: [0 1 5 6 7] TEST: [2 3 4]
```

```
TRAIN: [2 3 4 5 6 7] TEST: [0 1]
TRAIN: [0 1 5 6 7] TEST: [2 3 4]
TRAIN: [0 1 5 6 7] TEST: [2 3 4]
TRAIN: [0 1 2 3 4] TEST: [5 6 7]
TRAIN: [2 3 4 5 6 7] TEST: [0 1]
TRAIN: [0 1 2 3 4] TEST: [5 6 7]
```

Group-based split: GroupKFold



In [11]:

```
from sklearn.model_selection import GroupKFold

group_kfold = GroupKFold(n_splits=3)

for train_index, test_index in group_kfold.split(X, y, groups):
    print("TRAIN:", train_index, "TEST:", test_index)
```

```
TRAIN: [0 1 2 3 4] TEST: [5 6 7]
TRAIN: [0 1 5 6 7] TEST: [2 3 4]
TRAIN: [2 3 4 5 6 7] TEST: [0 1]
```

In [12]:

```
help(GroupKFold)
```

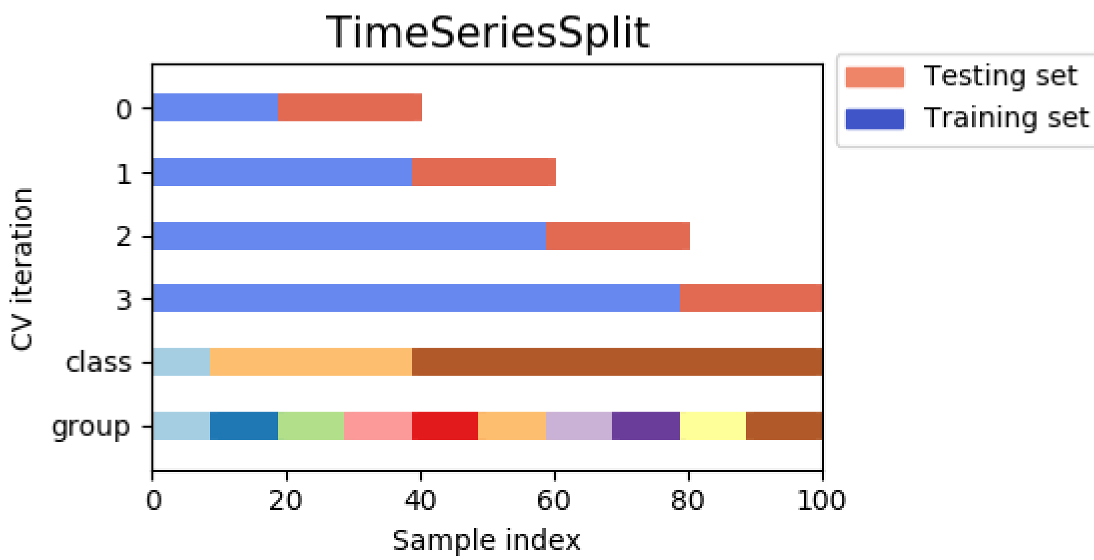
```
Help on class GroupKFold in module sklearn.model_selection._split:

class GroupKFold(_BaseKFold)
 |  GroupKFold(n_splits=5)
 |
 |  K-fold iterator variant with non-overlapping groups.
 |
 |  The same group will not appear in two different folds (the number of
 |  distinct groups has to be at least equal to the number of folds).
 |
 |  The folds are approximately balanced in the sense that the number of
 |  distinct groups is approximately the same in each fold.
 |
 |  Read more in the :ref:`User Guide <group_k_fold>`.
 |
 |  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.
 |
 |  Examples
 |  --------
 |  >>> import numpy as np
 |  >>> from sklearn.model_selection import GroupKFold
 |  >>> X = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])
 |  >>> y = np.array([1, 2, 3, 4])
 |  >>> groups = np.array([0, 0, 2, 2])
 |  >>> group_kfold = GroupKFold(n_splits=2)
 |  >>> group_kfold.get_n_splits(X, y, groups)
 |  2
 |  >>> print(group_kfold)
 |  GroupKFold(n_splits=2)
 |  >>> for train_index, test_index in group_kfold.split(X, y, groups):
 |  ...     print("TRAIN:", train_index, "TEST:", test_index)
 |  ...     X_train, X_test = X[train_index], X[test_index]
 |  ...     y_train, y_test = y[train_index], y[test_index]
 |  ...     print(X_train, X_test, y_train, y_test)
 |  ...
 |  TRAIN: [0 1] TEST: [2 3]
 |  [[1 2]
 |   [3 4]] [[5 6]
 |   [7 8]] [1 2] [3 4]
 |  TRAIN: [2 3] TEST: [0 1]
 |  [[5 6]
 |   [7 8]] [[1 2]
 |   [3 4]] [3 4] [1 2]
 |
 |  See Also
 |  --------
 |  LeaveOneGroupOut : For splitting the data according to explicit
 |      domain-specific stratification of the dataset.
 |
 |  Method resolution order:
 |      GroupKFold
 |      _BaseKFold
 |      BaseCrossValidator
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, n_splits=5)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  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,), default=None
 |          The target variable for supervised learning problems.
```

```
 |
 |      groups : array-like of shape (n_samples,)
 |          Group labels for the samples used while splitting the dataset into
 |          train/test set.
 |
 |      Yields
 |      ------
 |      train : ndarray
 |          The training set indices for that split.
 |
 |      test : ndarray
 |          The testing set indices for that split.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __abstractmethods__ = frozenset()
 |
 |  ----------------------------------------------------------------------
 |  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-validator.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from BaseCrossValidator:
 |
 |  __repr__(self)
 |      Return repr(self).
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from BaseCrossValidator:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

Data leakage in time series data is similar!

- do NOT use information in validation or test which will not be available once your model is deployed
  - don't use future information!



In [13]:

```
import numpy as np
from sklearn.model_selection import TimeSeriesSplit
X = np.array([[1, 2], [3, 4], [1, 2], [3, 4], [1, 2], [3, 4]])
y = np.array([1, 2, 3, 4, 5, 6])
tscv = TimeSeriesSplit()
for train_index, test_index in tscv.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

```
TRAIN: [0] TEST: [1]
TRAIN: [0 1] TEST: [2]
TRAIN: [0 1 2] TEST: [3]
TRAIN: [0 1 2 3] TEST: [4]
TRAIN: [0 1 2 3 4] TEST: [5]
```

Quiz 3

Go back to the GroupKFold example above. What happens when you change n_splits to 4? Why?

Why could we set the n_splits argument to 5 in GroupShuffleSplit?

Explain your answer in a couple of sentences!

Mudcard

In [ ]:

```

```