Mudcard

- **I still am having a hard time understanding why the curve for the autocorrelation plot is gradually flattening out as the lag increases?**
  - Autocorrelation is the correlation coefficient between the original and delayed time series data multiplied by a weight term.
  - That weight term is the fraction of the time series data used.
  - The weight term approaches 0 as the lag approaches the total duration of your time series data because you can only use a smaller and smaller fraction of your dataset.
  - modify my code and print out or plot the length of x or y as a function of the lag and you'll see.
- **For the splitting of data with group structures, can you clarify if the group kfold and shuffle splits also take into account making an equal proportion of the class?**
  - it does not
  - sklearn has a function called StratifiedGroupKFold which can do that:
  - "This cross-validation object is a variation of StratifiedKFold attempts to return stratified folds with non-overlapping groups. The folds are made by preserving the percentage of samples for each class. Each group will appear exactly once in the test set across all folds (the number of distinct groups has to be at least equal to the number of folds)."
  - read more in the sklearn manual
- **In what cases can you not use ML algorithmrs on time series data?**
  - I'm not sure there are any scenarios where you cannot apply ML to time series data
  - Certainly you need to do a lot of additional work like feature enginerring and careful splitting
  - And there might be other models that could do a better job.
- **Can it be problematic to have your testing set just compose of one group**
  - Yes, it is better to have multiple groups in the test set if possible
  - But that might not be in your control
  - Usually you just need to do the best you can with the dataset you receive
- **"I am still confused on when to use groupkfolds vs. groupshufflesplit. Wouldn't you want to use groupkfolds with a higher n because it will provide more non-repeated splits for you to train with? I understand if the n is too high, it will be computationally expensive, but then using groupshufflesplit won't use all of your data if nsplits is smaller than the groups?"**
  - Use GroupKFold if you want to guarantee that all groups are used once in test and you prefer deterministic splits.
  - Use GroupShuffleSplit when you want more flexibility in how you sample your data, especially if you are aiming to balance train-test size without strict group balancing. Using GroupShuffleSplit can be computationally less expensive so it's better suited for larger datasets.
- **I am confused what the lag is actually representing and why do I need to do this. If I know that my data fluctuates in it's correlation what does this mean I need to do with my data?**
- **Having some trouble to understand what exactly lag is**
  - autocorrelation gives you an idea of what kind of peridocities there are in your dataset
  - please go throguh the code line by line, come to the office hours to ask questions, post on the course forum, or read more about autocorrelation on the sklearn website.
- **Are there *any* situations in which it might be helpful to use 'future' time series data to predict 'past' time series data? If so, why is this? If not, why not?**
  - I don't think there are. It's information leakage to use future data to predict the past.
  - you can make this mistake while you develop a model
  - but once a model is deployed, you won't have access to the future data anymore

Data preprocessing

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

- apply one-hot encoding on categorical features
- apply ordinal encoding on ordinal features
- apply scaling and normalization to continuous variables

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)

Problem description, why preprocessing is necessary

Data format suitable for ML: 2D numerical values.

X	feature_1	feature_2	...	feature_j	...	feature_m	y
data_point_1	x_11	x_12	...	x_1j	...	x_1m	**y_1**
data_point_2	x_21	x_22	...	x_2j	...	x_2m	**y_2**
...	...	...	...	...	...	...	**...**
data_point_i	x_i1	x_i2	...	x_ij	...	x_im	**y_i**
...	...	...	...	...	...	...	**...**
data_point_n	x_n1	x_n2	...	x_nj	...	x_nm	**y_n**

Data almost never comes in a format that's directly usable in ML.

- let's check the adult data

In [1]:

```python
import pandas as pd
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 50k or less than 50k
X = df.loc[:, df.columns != 'gross-income'] # all other columns are features

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)

# 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('training set')
print(X_train.head()) # lots of strings!
print(y_train.head()) # even our labels are strings and not numbers!
```

```
training set
       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
25823     <=50K
10274     <=50K
27652     <=50K
13941      >50K
31384     <=50K
Name: gross-income, dtype: object
```

scikit-learn transformers to the rescue!

Preprocessing is done with various transformers. All transformes have three methods:

- **fit** method: estimates parameters necessary to do the transformation,
- **transform** method: transforms the data based on the estimated parameters,
- **fit_transform** method: both steps are performed at once, this can be faster than doing the steps separately.

Transformers we cover today

- **OneHotEncoder** - converts categorical features into dummy arrays
- **OrdinalEncoder** - converts categorical features into an integer array
- **MinMaxScaler** - scales continuous variables to be between 0 and 1
- **StandardScaler** - standardizes continuous features by removing the mean and scaling to unit variance

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

- **apply one-hot encoding on categorical features**
- apply ordinal encoding on ordinal features
- apply scaling and normalization to continuous variables

Unordered categorical data: one-hot encoder

- some categories cannot be ordered. e.g., workclass, relationship status

In [2]:
```python
from sklearn.preprocessing import OneHotEncoder
help(OneHotEncoder)
```

```
Help on class OneHotEncoder in module sklearn.preprocessing._encoders:

class OneHotEncoder(_BaseEncoder)
 |  OneHotEncoder(*, categories='auto', drop=None, sparse_output=True, dtype=<class 'numpy.float64'>, handle_unknown
='error', min_frequency=None, max_categories=None, feature_name_combiner='concat')
 |
 |  Encode categorical features as a one-hot numeric array.
 |
 |  The input to this transformer should be an array-like of integers or
 |  strings, denoting the values taken on by categorical (discrete) features.
 |  The features are encoded using a one-hot (aka 'one-of-K' or 'dummy')
 |  encoding scheme. This creates a binary column for each category and
 |  returns a sparse matrix or dense array (depending on the ``sparse_output``
 |  parameter).
 |
 |  By default, the encoder derives the categories based on the unique values
 |  in each feature. Alternatively, you can also specify the `categories`
 |  manually.
 |
 |  This encoding is needed for feeding categorical data to many scikit-learn
 |  estimators, notably linear models and SVMs with the standard kernels.
 |
 |  Note: a one-hot encoding of y labels should use a LabelBinarizer
 |  instead.
 |
 |  Read more in the :ref:`User Guide <preprocessing_categorical_features>`.
 |  For a comparison of different encoders, refer to:
 |  :ref:`sphx_glr_auto_examples_preprocessing_plot_target_encoder.py`.
 |
 |  Parameters
 |  ----------
 |  categories : 'auto' or a list of array-like, default='auto'
 |      Categories (unique values) per feature:
 |
 |      - 'auto' : Determine categories automatically from the training data.
 |      - list : ``categories[i]`` holds the categories expected in the ith
 |        column. The passed categories should not mix strings and numeric
 |        values within a single feature, and should be sorted in case of
 |        numeric values.
 |
 |      The used categories can be found in the ``categories_`` attribute.
 |
 |      .. versionadded:: 0.20
 |
 |  drop : {'first', 'if_binary'} or an array-like of shape (n_features,),             default=None
 |      Specifies a methodology to use to drop one of the categories per
 |      feature. This is useful in situations where perfectly collinear
 |      features cause problems, such as when feeding the resulting data
 |      into an unregularized linear regression model.
 |
 |      However, dropping one category breaks the symmetry of the original
 |      representation and can therefore induce a bias in downstream models,
 |      for instance for penalized linear classification or regression models.
 |
 |      - None : retain all features (the default).
 |      - 'first' : drop the first category in each feature. If only one
 |        category is present, the feature will be dropped entirely.
 |      - 'if_binary' : drop the first category in each feature with two
 |        categories. Features with 1 or more than 2 categories are
 |        left intact.
 |      - array : ``drop[i]`` is the category in feature ``X[:, i]`` that
 |        should be dropped.
 |
 |      When `max_categories` or `min_frequency` is configured to group
 |      infrequent categories, the dropping behavior is handled after the
 |      grouping.
 |
 |      .. versionadded:: 0.21
 |         The parameter `drop` was added in 0.21.
 |
 |      .. versionchanged:: 0.23
 |         The option `drop='if_binary'` was added in 0.23.
 |
 |      .. versionchanged:: 1.1
 |          Support for dropping infrequent categories.
 |
 |  sparse_output : bool, default=True
 |      When ``True``, it returns a :class:`scipy.sparse.csr_matrix`,
 |      i.e. a sparse matrix in "Compressed Sparse Row" (CSR) format.
 |
 |      .. versionadded:: 1.2
 |         `sparse` was renamed to `sparse_output`
 |
 |  dtype : number type, default=np.float64
 |      Desired dtype of output.
 |
 |  handle_unknown : {'error', 'ignore', 'infrequent_if_exist'},                      default='error'
```

```
 |      Specifies the way unknown categories are handled during :meth:`transform`.
 |
 |      - 'error' : Raise an error if an unknown category is present during transform.
 |      - 'ignore' : When an unknown category is encountered during
 |        transform, the resulting one-hot encoded columns for this feature
 |        will be all zeros. In the inverse transform, an unknown category
 |        will be denoted as None.
 |      - 'infrequent_if_exist' : When an unknown category is encountered
 |        during transform, the resulting one-hot encoded columns for this
 |        feature will map to the infrequent category if it exists. The
 |        infrequent category will be mapped to the last position in the
 |        encoding. During inverse transform, an unknown category will be
 |        mapped to the category denoted ``'infrequent'`` if it exists. If the
 |        ``'infrequent'`` category does not exist, then :meth:`transform` and
 |        :meth:`inverse_transform` will handle an unknown category as with
 |        `handle_unknown='ignore'`. Infrequent categories exist based on
 |        `min_frequency` and `max_categories`. Read more in the
 |        :ref:`User Guide <encoder_infrequent_categories>`.
 |
 |      .. versionchanged:: 1.1
 |          ``'infrequent_if_exist'`` was added to automatically handle unknown
 |          categories and infrequent categories.
 |
 |  min_frequency : int or float, default=None
 |      Specifies the minimum frequency below which a category will be
 |      considered infrequent.
 |
 |      - If `int`, categories with a smaller cardinality will be considered
 |        infrequent.
 |
 |      - If `float`, categories with a smaller cardinality than
 |        `min_frequency * n_samples`  will be considered infrequent.
 |
 |      .. versionadded:: 1.1
 |          Read more in the :ref:`User Guide <encoder_infrequent_categories>`.
 |
 |  max_categories : int, default=None
 |      Specifies an upper limit to the number of output features for each input
 |      feature when considering infrequent categories. If there are infrequent
 |      categories, `max_categories` includes the category representing the
 |      infrequent categories along with the frequent categories. If `None`,
 |      there is no limit to the number of output features.
 |
 |      .. versionadded:: 1.1
 |          Read more in the :ref:`User Guide <encoder_infrequent_categories>`.
 |
 |  feature_name_combiner : "concat" or callable, default="concat"
 |      Callable with signature `def callable(input_feature, category)` that returns a
 |      string. This is used to create feature names to be returned by
 |      :meth:`get_feature_names_out`.
 |
 |      `"concat"` concatenates encoded feature name and category with
 |      `feature + "_" + str(category)`.E.g. feature X with values 1, 6, 7 create
 |      feature names `X_1, X_6, X_7`.
 |
 |      .. versionadded:: 1.3
 |
 |  Attributes
 |  ----------
 |  categories_ : list of arrays
 |      The categories of each feature determined during fitting
 |      (in order of the features in X and corresponding with the output
 |      of ``transform``). This includes the category specified in ``drop``
 |      (if any).
 |
 |  drop_idx_ : array of shape (n_features,)
 |      - ``drop_idx_[i]`` is the index in ``categories_[i]`` of the category
 |        to be dropped for each feature.
 |      - ``drop_idx_[i] = None`` if no category is to be dropped from the
 |        feature with index ``i``, e.g. when `drop='if_binary'` and the
 |        feature isn't binary.
 |      - ``drop_idx_ = None`` if all the transformed features will be
 |        retained.
 |
 |      If infrequent categories are enabled by setting `min_frequency` or
 |      `max_categories` to a non-default value and `drop_idx[i]` corresponds
 |      to a infrequent category, then the entire infrequent category is
 |      dropped.
 |
 |      .. versionchanged:: 0.23
 |         Added the possibility to contain `None` values.
 |
 |  infrequent_categories_ : list of ndarray
 |      Defined only if infrequent categories are enabled by setting
 |      `min_frequency` or `max_categories` to a non-default value.
 |      `infrequent_categories_[i]` are the infrequent categories for feature
 |      `i`. If the feature `i` has no infrequent categories
```

```
 |          `infrequent_categories_[i]` is None.
 |
 |          .. versionadded:: 1.1
 |
 |      n_features_in_ : int
 |          Number of features seen during :term:`fit`.
 |
 |          .. versionadded:: 1.0
 |
 |      feature_names_in_ : ndarray of shape (`n_features_in_`,)
 |          Names of features seen during :term:`fit`. Defined only when `X`
 |          has feature names that are all strings.
 |
 |          .. versionadded:: 1.0
 |
 |      feature_name_combiner : callable or None
 |          Callable with signature `def callable(input_feature, category)` that returns a
 |          string. This is used to create feature names to be returned by
 |          :meth:`get_feature_names_out`.
 |
 |          .. versionadded:: 1.3
 |
 |      See Also
 |      --------
 |      OrdinalEncoder : Performs an ordinal (integer)
 |        encoding of the categorical features.
 |      TargetEncoder : Encodes categorical features using the target.
 |      sklearn.feature_extraction.DictVectorizer : Performs a one-hot encoding of
 |        dictionary items (also handles string-valued features).
 |      sklearn.feature_extraction.FeatureHasher : Performs an approximate one-hot
 |        encoding of dictionary items or strings.
 |      LabelBinarizer : Binarizes labels in a one-vs-all
 |        fashion.
 |      MultiLabelBinarizer : Transforms between iterable of
 |        iterables and a multilabel format, e.g. a (samples x classes) binary
 |        matrix indicating the presence of a class label.
 |
 |      Examples
 |      --------
 |      Given a dataset with two features, we let the encoder find the unique
 |      values per feature and transform the data to a binary one-hot encoding.
 |
 |      >>> from sklearn.preprocessing import OneHotEncoder
 |
 |      One can discard categories not seen during `fit`:
 |
 |      >>> enc = OneHotEncoder(handle_unknown='ignore')
 |      >>> X = [['Male', 1], ['Female', 3], ['Female', 2]]
 |      >>> enc.fit(X)
 |      OneHotEncoder(handle_unknown='ignore')
 |      >>> enc.categories_
 |      [array(['Female', 'Male'], dtype=object), array([1, 2, 3], dtype=object)]
 |      >>> enc.transform([['Female', 1], ['Male', 4]]).toarray()
 |      array([[1., 0., 1., 0., 0.],
 |             [0., 1., 0., 0., 0.]])
 |      >>> enc.inverse_transform([[0, 1, 1, 0, 0], [0, 0, 0, 1, 0]])
 |      array([['Male', 1],
 |             [None, 2]], dtype=object)
 |      >>> enc.get_feature_names_out(['gender', 'group'])
 |      array(['gender_Female', 'gender_Male', 'group_1', 'group_2', 'group_3'], ...)
 |
 |      One can always drop the first column for each feature:
 |
 |      >>> drop_enc = OneHotEncoder(drop='first').fit(X)
 |      >>> drop_enc.categories_
 |      [array(['Female', 'Male'], dtype=object), array([1, 2, 3], dtype=object)]
 |      >>> drop_enc.transform([['Female', 1], ['Male', 2]]).toarray()
 |      array([[0., 0., 0.],
 |             [1., 1., 0.]])
 |
 |      Or drop a column for feature only having 2 categories:
 |
 |      >>> drop_binary_enc = OneHotEncoder(drop='if_binary').fit(X)
 |      >>> drop_binary_enc.transform([['Female', 1], ['Male', 2]]).toarray()
 |      array([[0., 1., 0., 0.],
 |             [1., 0., 1., 0.]])
 |
 |      One can change the way feature names are created.
 |
 |      >>> def custom_combiner(feature, category):
 |      ...     return str(feature) + "_" + type(category).__name__ + "_" + str(category)
 |      >>> custom_fnames_enc = OneHotEncoder(feature_name_combiner=custom_combiner).fit(X)
 |      >>> custom_fnames_enc.get_feature_names_out()
 |      array(['x0_str_Female', 'x0_str_Male', 'x1_int_1', 'x1_int_2', 'x1_int_3'],
 |            dtype=object)
 |
 |      Infrequent categories are enabled by setting `max_categories` or `min_frequency`.
```

```
 |
 |  >>> import numpy as np
 |  >>> X = np.array([["a"] * 5 + ["b"] * 20 + ["c"] * 10 + ["d"] * 3], dtype=object).T
 |  >>> ohe = OneHotEncoder(max_categories=3, sparse_output=False).fit(X)
 |  >>> ohe.infrequent_categories_
 |  [array(['a', 'd'], dtype=object)]
 |  >>> ohe.transform([["a"], ["b"]])
 |  array([[0., 0., 1.],
 |         [1., 0., 0.]])
 |
 |  Method resolution order:
 |      OneHotEncoder
 |      _BaseEncoder
 |      sklearn.base.TransformerMixin
 |      sklearn.utils._set_output._SetOutputMixin
 |      sklearn.base.BaseEstimator
 |      sklearn.utils._estimator_html_repr._HTMLDocumentationLinkMixin
 |      sklearn.utils._metadata_requests._MetadataRequester
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, *, categories='auto', drop=None, sparse_output=True, dtype=<class 'numpy.float64'>, handle_unknow
n='error', min_frequency=None, max_categories=None, feature_name_combiner='concat')
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  fit(self, X, y=None)
 |      Fit OneHotEncoder to X.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          The data to determine the categories of each feature.
 |
 |      y : None
 |          Ignored. This parameter exists only for compatibility with
 |          :class:`~sklearn.pipeline.Pipeline`.
 |
 |      Returns
 |      -------
 |      self
 |          Fitted encoder.
 |
 |  get_feature_names_out(self, input_features=None)
 |      Get output feature names for transformation.
 |
 |      Parameters
 |      ----------
 |      input_features : array-like of str or None, default=None
 |          Input features.
 |
 |          - If `input_features` is `None`, then `feature_names_in_` is
 |            used as feature names in. If `feature_names_in_` is not defined,
 |            then the following input feature names are generated:
 |            `["x0", "x1", ..., "x(n_features_in_ - 1)"]`.
 |          - If `input_features` is an array-like, then `input_features` must
 |            match `feature_names_in_` if `feature_names_in_` is defined.
 |
 |      Returns
 |      -------
 |      feature_names_out : ndarray of str objects
 |          Transformed feature names.
 |
 |  inverse_transform(self, X)
 |      Convert the data back to the original representation.
 |
 |      When unknown categories are encountered (all zeros in the
 |      one-hot encoding), ``None`` is used to represent this category. If the
 |      feature with the unknown category has a dropped category, the dropped
 |      category will be its inverse.
 |
 |      For a given input feature, if there is an infrequent category,
 |      'infrequent_sklearn' will be used to represent the infrequent category.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape                 (n_samples, n_encoded_features)
 |          The transformed data.
 |
 |      Returns
 |      -------
 |      X_tr : ndarray of shape (n_samples, n_features)
 |          Inverse transformed array.
 |
 |  transform(self, X)
 |      Transform X using one-hot encoding.
 |
```

```
 |      If `sparse_output=True` (default), it returns an instance of
 |      :class:`scipy.sparse._csr.csr_matrix` (CSR format).
 |
 |      If there are infrequent categories for a feature, set by specifying
 |      `max_categories` or `min_frequency`, the infrequent categories are
 |      grouped into a single category.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          The data to encode.
 |
 |      Returns
 |      -------
 |      X_out : {ndarray, sparse matrix} of shape                 (n_samples, n_encoded_features)
 |          Transformed input. If `sparse_output=True`, a sparse matrix will be
 |          returned.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __annotations__ = {'_parameter_constraints': <class 'dict'>}
 |
 |  ----------------------------------------------------------------------
 |  Readonly properties inherited from _BaseEncoder:
 |
 |  infrequent_categories_
 |      Infrequent categories for each feature.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.TransformerMixin:
 |
 |  fit_transform(self, X, y=None, **fit_params)
 |      Fit to data, then transform it.
 |
 |      Fits transformer to `X` and `y` with optional parameters `fit_params`
 |      and returns a transformed version of `X`.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          Input samples.
 |
 |      y :  array-like of shape (n_samples,) or (n_samples, n_outputs),                 default=None
 |          Target values (None for unsupervised transformations).
 |
 |      **fit_params : dict
 |          Additional fit parameters.
 |
 |      Returns
 |      -------
 |      X_new : ndarray array of shape (n_samples, n_features_new)
 |          Transformed array.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.utils._set_output._SetOutputMixin:
 |
 |  set_output(self, *, transform=None)
 |      Set output container.
 |
 |      See :ref:`sphx_glr_auto_examples_miscellaneous_plot_set_output.py`
 |      for an example on how to use the API.
 |
 |      Parameters
 |      ----------
 |      transform : {"default", "pandas", "polars"}, default=None
 |          Configure output of `transform` and `fit_transform`.
 |
 |          - `"default"`: Default output format of a transformer
 |          - `"pandas"`: DataFrame output
 |          - `"polars"`: Polars output
 |          - `None`: Transform configuration is unchanged
 |
 |          .. versionadded:: 1.4
 |              `"polars"` option was added.
 |
 |      Returns
 |      -------
 |      self : estimator instance
 |          Estimator instance.
 |
 |  ----------------------------------------------------------------------
 |  Class methods inherited from sklearn.utils._set_output._SetOutputMixin:
 |
 |  __init_subclass__(auto_wrap_output_keys=('transform',), **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 method
 |      does not explicitly accept a metadata through its arguments or if the
 |      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
 |  
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from sklearn.utils._set_output._SetOutputMixin:
 |  
 |  __dict__
 |      dictionary for instance variables
 |  
 |  __weakref__
 |      list of weak references to the object
 |  
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.BaseEstimator:
 |  
 |  __getstate__(self)
 |      Helper for pickle.
 |  
 |  __repr__(self, N_CHAR_MAX=700)
 |      Return repr(self).
 |  
 |  __setstate__(self, state)
 |  
 |  __sklearn_clone__(self)
 |  
 |  get_params(self, deep=True)
 |      Get parameters for this estimator.
 |      
 |      Parameters
 |      ----------
 |      deep : bool, default=True
 |          If True, will return the parameters for this estimator and
 |          contained subobjects that are estimators.
 |      
 |      Returns
 |      -------
 |      params : dict
 |          Parameter names mapped to their values.
 |  
 |  set_params(self, **params)
 |      Set the parameters of this estimator.
 |      
 |      The method works on simple estimators as well as on nested objects
 |      (such as :class:`~sklearn.pipeline.Pipeline`). The latter have
 |      parameters of the form ``<component>__<parameter>`` so that it's
 |      possible to update each component of a nested object.
 |      
 |      Parameters
 |      ----------
 |      **params : dict
 |          Estimator parameters.
 |      
 |      Returns
 |      -------
 |      self : estimator instance
 |          Estimator instance.
 |  
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.utils._metadata_requests._MetadataRequester:
 |  
 |  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` encapsulating
 |          routing information.
```

In [3]:

```
# toy example
train = {'gender':['Male','Female','Unknown','Male','Female','Female'],\
         'browser':['Safari','Safari','Internet Explorer','Chrome','Chrome','Internet Explorer']}
```

```python
test = {'gender':['Female','Male','Unknown','Female'],'browser':['Chrome','Firefox','Internet Explorer','Safari']}

Xtoy_train = pd.DataFrame(train)
Xtoy_test = pd.DataFrame(test)

ftrs = ['gender','browser']

# initialize the encoder
enc = OneHotEncoder(sparse_output=False, handle_unknown = 'ignore') # by default, OneHotEncoder returns a sparse ma
# fit the training data
enc.fit(Xtoy_train)
print('categories:',enc.categories_)
print('feature names:',enc.get_feature_names_out(ftrs))
# transform X_train
X_train_ohe = enc.transform(Xtoy_train)
#print(X_train_ohe)
# do all of this in one step
X_train_ohe = enc.fit_transform(Xtoy_train)
print('X_train transformed')
print(X_train_ohe)

# transform X_test
X_test_ohe = enc.transform(Xtoy_test)
print('X_test transformed')
print(X_test_ohe)
```

```
categories: [array(['Female', 'Male', 'Unknown'], dtype=object), array(['Chrome', 'Internet Explorer', 'Safari'], dt
ype=object)]
feature names: ['gender_Female' 'gender_Male' 'gender_Unknown' 'browser_Chrome'
 'browser_Internet Explorer' 'browser_Safari']
X_train transformed
[[0. 1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 0.]
 [1. 0. 0. 1. 0. 0.]
 [1. 0. 0. 0. 1. 0.]]
X_test transformed
[[1. 0. 0. 1. 0. 0.]
 [0. 1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 1. 0.]
 [1. 0. 0. 0. 0. 1.]]
```

In [4]:

```python
# apply OHE to the adult dataset

# let's collect all categorical features first
onehot_ftrs = ['workclass','marital-status','occupation','relationship','race','sex','native-country']
# initialize the encoder
enc = OneHotEncoder(sparse_output=False,handle_unknown='ignore') # by default, OneHotEncoder returns a sparse matri
# fit the training data
enc.fit(X_train[onehot_ftrs])
print('feature names:',enc.get_feature_names_out(onehot_ftrs))
print(len(enc.get_feature_names_out(onehot_ftrs)))
```

```
feature names: ['workclass_ ?' 'workclass_ Federal-gov' 'workclass_ Local-gov'
 'workclass_ Never-worked' 'workclass_ Private' 'workclass_ Self-emp-inc'
 'workclass_ Self-emp-not-inc' 'workclass_ State-gov'
 'workclass_ Without-pay' 'marital-status_ Divorced'
 'marital-status_ Married-AF-spouse' 'marital-status_ Married-civ-spouse'
 'marital-status_ Married-spouse-absent' 'marital-status_ Never-married'
 'marital-status_ Separated' 'marital-status_ Widowed' 'occupation_ ?'
 'occupation_ Adm-clerical' 'occupation_ Armed-Forces'
 'occupation_ Craft-repair' 'occupation_ Exec-managerial'
 'occupation_ Farming-fishing' 'occupation_ Handlers-cleaners'
 'occupation_ Machine-op-inspct' 'occupation_ Other-service'
 'occupation_ Priv-house-serv' 'occupation_ Prof-specialty'
 'occupation_ Protective-serv' 'occupation_ Sales'
 'occupation_ Tech-support' 'occupation_ Transport-moving'
 'relationship_ Husband' 'relationship_ Not-in-family'
 'relationship_ Other-relative' 'relationship_ Own-child'
 'relationship_ Unmarried' 'relationship_ Wife' 'race_ Amer-Indian-Eskimo'
 'race_ Asian-Pac-Islander' 'race_ Black' 'race_ Other' 'race_ White'
 'sex_ Female' 'sex_ Male' 'native-country_ ?' 'native-country_ Cambodia'
 'native-country_ Canada' 'native-country_ China'
 'native-country_ Columbia' 'native-country_ Cuba'
 'native-country_ Dominican-Republic' 'native-country_ Ecuador'
 'native-country_ El-Salvador' 'native-country_ England'
 'native-country_ France' 'native-country_ Germany'
 'native-country_ Greece' 'native-country_ Guatemala'
 'native-country_ Haiti' 'native-country_ Holand-Netherlands'
 'native-country_ Honduras' 'native-country_ Hong'
 'native-country_ Hungary' 'native-country_ India' 'native-country_ Iran'
 'native-country_ Ireland' 'native-country_ Italy'
 'native-country_ Jamaica' 'native-country_ Japan' 'native-country_ Laos'
 'native-country_ Mexico' 'native-country_ Nicaragua'
 'native-country_ Outlying-US(Guam-USVI-etc)' 'native-country_ Peru'
 'native-country_ Philippines' 'native-country_ Poland'
 'native-country_ Portugal' 'native-country_ Puerto-Rico'
 'native-country_ Scotland' 'native-country_ South'
 'native-country_ Taiwan' 'native-country_ Thailand'
 'native-country_ Trinadad&Tobago' 'native-country_ United-States'
 'native-country_ Vietnam' 'native-country_ Yugoslavia']
86
```

In [5]:
```python
# transform X_train
onehot_train = enc.transform(X_train[onehot_ftrs])
print('transformed train features:')
print(onehot_train)
# transform X_val
onehot_val = enc.transform(X_val[onehot_ftrs])
print('transformed val features:')
print(onehot_val)
# transform X_test
onehot_test = enc.transform(X_test[onehot_ftrs])
print('transformed test features:')
print(onehot_test)
```

```
transformed train features:
[[0. 0. 0. ... 1. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 1. 0. 0.]
 ...
 [0. 0. 0. ... 1. 0. 0.]
 [0. 0. 0. ... 1. 0. 0.]
 [0. 0. 0. ... 1. 0. 0.]]
transformed val features:
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 1. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 1. 0. 0.]
 [0. 0. 0. ... 1. 0. 0.]
 [0. 0. 0. ... 1. 0. 0.]]
transformed test features:
[[0. 0. 0. ... 1. 0. 0.]
 [0. 0. 0. ... 1. 0. 0.]
 [0. 0. 0. ... 1. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 1. 0. 0.]
 [0. 0. 0. ... 1. 0. 0.]]
```

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

- apply one-hot encoding on categorical features
- **apply ordinal encoding on ordinal features**
- apply scaling and normalization to continuous variables

Ordered categorical data: OrdinalEncoder

- use it on categorical features if the categories can be ranked or ordered
    - educational level in the adult dataset
    - reaction to medication is described by words like 'severe', 'no response', 'excellent'
    - any time you know that the categories can be clearly ranked

In [6]:
```
from sklearn.preprocessing import OrdinalEncoder
help(OrdinalEncoder)
```

```
Help on class OrdinalEncoder in module sklearn.preprocessing._encoders:

class OrdinalEncoder(sklearn.base.OneToOneFeatureMixin, _BaseEncoder)
 |  OrdinalEncoder(*, categories='auto', dtype=<class 'numpy.float64'>, handle_unknown='error', unknown_value=None,
encoded_missing_value=nan, min_frequency=None, max_categories=None)
 |
 |  Encode categorical features as an integer array.
 |
 |  The input to this transformer should be an array-like of integers or
 |  strings, denoting the values taken on by categorical (discrete) features.
 |  The features are converted to ordinal integers. This results in
 |  a single column of integers (0 to n_categories - 1) per feature.
 |
 |  Read more in the :ref:`User Guide <preprocessing_categorical_features>`.
 |  For a comparison of different encoders, refer to:
 |  :ref:`sphx_glr_auto_examples_preprocessing_plot_target_encoder.py`.
 |
 |  .. versionadded:: 0.20
 |
 |  Parameters
 |  ----------
 |  categories : 'auto' or a list of array-like, default='auto'
 |      Categories (unique values) per feature:
 |
 |      - 'auto' : Determine categories automatically from the training data.
 |      - list : ``categories[i]`` holds the categories expected in the ith
 |        column. The passed categories should not mix strings and numeric
 |        values, and should be sorted in case of numeric values.
 |
 |      The used categories can be found in the ``categories_`` attribute.
 |
 |  dtype : number type, default=np.float64
 |      Desired dtype of output.
 |
 |  handle_unknown : {'error', 'use_encoded_value'}, default='error'
 |      When set to 'error' an error will be raised in case an unknown
 |      categorical feature is present during transform. When set to
 |      'use_encoded_value', the encoded value of unknown categories will be
 |      set to the value given for the parameter `unknown_value`. In
 |      :meth:`inverse_transform`, an unknown category will be denoted as None.
 |
 |      .. versionadded:: 0.24
 |
 |  unknown_value : int or np.nan, default=None
 |      When the parameter handle_unknown is set to 'use_encoded_value', this
 |      parameter is required and will set the encoded value of unknown
 |      categories. It has to be distinct from the values used to encode any of
 |      the categories in `fit`. If set to np.nan, the `dtype` parameter must
 |      be a float dtype.
 |
 |      .. versionadded:: 0.24
 |
 |  encoded_missing_value : int or np.nan, default=np.nan
 |      Encoded value of missing categories. If set to `np.nan`, then the `dtype`
 |      parameter must be a float dtype.
 |
 |      .. versionadded:: 1.1
 |
 |  min_frequency : int or float, default=None
 |      Specifies the minimum frequency below which a category will be
 |      considered infrequent.
 |
 |      - If `int`, categories with a smaller cardinality will be considered
 |        infrequent.
 |
 |      - If `float`, categories with a smaller cardinality than
 |        `min_frequency * n_samples`  will be considered infrequent.
 |
 |      .. versionadded:: 1.3
 |          Read more in the :ref:`User Guide <encoder_infrequent_categories>`.
 |
 |  max_categories : int, default=None
 |      Specifies an upper limit to the number of output categories for each input
 |      feature when considering infrequent categories. If there are infrequent
 |      categories, `max_categories` includes the category representing the
 |      infrequent categories along with the frequent categories. If `None`,
 |      there is no limit to the number of output features.
 |
 |      `max_categories` do **not** take into account missing or unknown
 |      categories. Setting `unknown_value` or `encoded_missing_value` to an
 |      integer will increase the number of unique integer codes by one each.
 |      This can result in up to `max_categories + 2` integer codes.
 |
 |      .. versionadded:: 1.3
 |          Read more in the :ref:`User Guide <encoder_infrequent_categories>`.
 |
 |  Attributes
```

```
 |  ----------
 |  categories_ : list of arrays
 |      The categories of each feature determined during ``fit`` (in order of
 |      the features in X and corresponding with the output of ``transform``).
 |      This does not include categories that weren't seen during ``fit``.
 |
 |  n_features_in_ : int
 |      Number of features seen during :term:`fit`.
 |
 |      .. versionadded:: 1.0
 |
 |  feature_names_in_ : ndarray of shape (`n_features_in_`,)
 |      Names of features seen during :term:`fit`. Defined only when `X`
 |      has feature names that are all strings.
 |
 |      .. versionadded:: 1.0
 |
 |  infrequent_categories_ : list of ndarray
 |      Defined only if infrequent categories are enabled by setting
 |      `min_frequency` or `max_categories` to a non-default value.
 |      `infrequent_categories_[i]` are the infrequent categories for feature
 |      `i`. If the feature `i` has no infrequent categories
 |      `infrequent_categories_[i]` is None.
 |
 |      .. versionadded:: 1.3
 |
 |  See Also
 |  --------
 |  OneHotEncoder : Performs a one-hot encoding of categorical features. This encoding
 |      is suitable for low to medium cardinality categorical variables, both in
 |      supervised and unsupervised settings.
 |  TargetEncoder : Encodes categorical features using supervised signal
 |      in a classification or regression pipeline. This encoding is typically
 |      suitable for high cardinality categorical variables.
 |  LabelEncoder : Encodes target labels with values between 0 and
 |      ``n_classes-1``.
 |
 |  Notes
 |  -----
 |  With a high proportion of `nan` values, inferring categories becomes slow with
 |  Python versions before 3.10. The handling of `nan` values was improved
 |  from Python 3.10 onwards, (c.f.
 |  `bpo-43475 <https://github.com/python/cpython/issues/87641>`_).
 |
 |  Examples
 |  --------
 |  Given a dataset with two features, we let the encoder find the unique
 |  values per feature and transform the data to an ordinal encoding.
 |
 |  >>> from sklearn.preprocessing import OrdinalEncoder
 |  >>> enc = OrdinalEncoder()
 |  >>> X = [['Male', 1], ['Female', 3], ['Female', 2]]
 |  >>> enc.fit(X)
 |  OrdinalEncoder()
 |  >>> enc.categories_
 |  [array(['Female', 'Male'], dtype=object), array([1, 2, 3], dtype=object)]
 |  >>> enc.transform([['Female', 3], ['Male', 1]])
 |  array([[0., 2.],
 |         [1., 0.]])
 |
 |  >>> enc.inverse_transform([[1, 0], [0, 1]])
 |  array([['Male', 1],
 |         ['Female', 2]], dtype=object)
 |
 |  By default, :class:`OrdinalEncoder` is lenient towards missing values by
 |  propagating them.
 |
 |  >>> import numpy as np
 |  >>> X = [['Male', 1], ['Female', 3], ['Female', np.nan]]
 |  >>> enc.fit_transform(X)
 |  array([[ 1.,  0.],
 |         [ 0.,  1.],
 |         [ 0., nan]])
 |
 |  You can use the parameter `encoded_missing_value` to encode missing values.
 |
 |  >>> enc.set_params(encoded_missing_value=-1).fit_transform(X)
 |  array([[ 1.,  0.],
 |         [ 0.,  1.],
 |         [ 0., -1.]])
 |
 |  Infrequent categories are enabled by setting `max_categories` or `min_frequency`.
 |  In the following example, "a" and "d" are considered infrequent and grouped
 |  together into a single category, "b" and "c" are their own categories, unknown
 |  values are encoded as 3 and missing values are encoded as 4.
 |
 |  >>> X_train = np.array(
```

```
 |  ...     [["a"] * 5 + ["b"] * 20 + ["c"] * 10 + ["d"] * 3 + [np.nan]],
 |  ...     dtype=object).T
 |  >>> enc = OrdinalEncoder(
 |  ...     handle_unknown="use_encoded_value", unknown_value=3,
 |  ...     max_categories=3, encoded_missing_value=4)
 |  >>> _ = enc.fit(X_train)
 |  >>> X_test = np.array([["a"], ["b"], ["c"], ["d"], ["e"], [np.nan]], dtype=object)
 |  >>> enc.transform(X_test)
 |  array([[2.],
 |         [0.],
 |         [1.],
 |         [2.],
 |         [3.],
 |         [4.]])
 |
 |  Method resolution order:
 |      OrdinalEncoder
 |      sklearn.base.OneToOneFeatureMixin
 |      _BaseEncoder
 |      sklearn.base.TransformerMixin
 |      sklearn.utils._set_output._SetOutputMixin
 |      sklearn.base.BaseEstimator
 |      sklearn.utils._estimator_html_repr._HTMLDocumentationLinkMixin
 |      sklearn.utils._metadata_requests._MetadataRequester
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, *, categories='auto', dtype=<class 'numpy.float64'>, handle_unknown='error', unknown_value=None,
encoded_missing_value=nan, min_frequency=None, max_categories=None)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  fit(self, X, y=None)
 |      Fit the OrdinalEncoder to X.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          The data to determine the categories of each feature.
 |
 |      y : None
 |          Ignored. This parameter exists only for compatibility with
 |          :class:`~sklearn.pipeline.Pipeline`.
 |
 |      Returns
 |      -------
 |      self : object
 |          Fitted encoder.
 |
 |  inverse_transform(self, X)
 |      Convert the data back to the original representation.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_encoded_features)
 |          The transformed data.
 |
 |      Returns
 |      -------
 |      X_tr : ndarray of shape (n_samples, n_features)
 |          Inverse transformed array.
 |
 |  transform(self, X)
 |      Transform X to ordinal codes.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          The data to encode.
 |
 |      Returns
 |      -------
 |      X_out : ndarray of shape (n_samples, n_features)
 |          Transformed input.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __annotations__ = {'_parameter_constraints': <class 'dict'>}
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.OneToOneFeatureMixin:
 |
 |  get_feature_names_out(self, input_features=None)
 |      Get output feature names for transformation.
 |
 |      Parameters
```

```
 |      ----------
 |      input_features : array-like of str or None, default=None
 |          Input features.
 |
 |          - If `input_features` is `None`, then `feature_names_in_` is
 |            used as feature names in. If `feature_names_in_` is not defined,
 |            then the following input feature names are generated:
 |            `["x0", "x1", ..., "x(n_features_in_ - 1)"]`.
 |          - If `input_features` is an array-like, then `input_features` must
 |            match `feature_names_in_` if `feature_names_in_` is defined.
 |
 |      Returns
 |      -------
 |      feature_names_out : ndarray of str objects
 |          Same as input features.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from sklearn.base.OneToOneFeatureMixin:
 |
 |  __dict__
 |      dictionary for instance variables
 |
 |  __weakref__
 |      list of weak references to the object
 |
 |  ----------------------------------------------------------------------
 |  Readonly properties inherited from _BaseEncoder:
 |
 |  infrequent_categories_
 |      Infrequent categories for each feature.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.TransformerMixin:
 |
 |  fit_transform(self, X, y=None, **fit_params)
 |      Fit to data, then transform it.
 |
 |      Fits transformer to `X` and `y` with optional parameters `fit_params`
 |      and returns a transformed version of `X`.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          Input samples.
 |
 |      y :  array-like of shape (n_samples,) or (n_samples, n_outputs),                 default=None
 |          Target values (None for unsupervised transformations).
 |
 |      **fit_params : dict
 |          Additional fit parameters.
 |
 |      Returns
 |      -------
 |      X_new : ndarray array of shape (n_samples, n_features_new)
 |          Transformed array.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.utils._set_output._SetOutputMixin:
 |
 |  set_output(self, *, transform=None)
 |      Set output container.
 |
 |      See :ref:`sphx_glr_auto_examples_miscellaneous_plot_set_output.py`
 |      for an example on how to use the API.
 |
 |      Parameters
 |      ----------
 |      transform : {"default", "pandas", "polars"}, default=None
 |          Configure output of `transform` and `fit_transform`.
 |
 |          - `"default"`: Default output format of a transformer
 |          - `"pandas"`: DataFrame output
 |          - `"polars"`: Polars output
 |          - `None`: Transform configuration is unchanged
 |
 |          .. versionadded:: 1.4
 |              `"polars"` option was added.
 |
 |      Returns
 |      -------
 |      self : estimator instance
 |          Estimator instance.
 |
 |  ----------------------------------------------------------------------
 |  Class methods inherited from sklearn.utils._set_output._SetOutputMixin:
 |
 |  __init_subclass__(auto_wrap_output_keys=('transform',), **kwargs)
```

```
 |      This method is called when a class is subclassed.
 |
 |      The default implementation does nothing. It may be
 |      overridden to extend subclasses.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.BaseEstimator:
 |
 |  __getstate__(self)
 |      Helper for pickle.
 |
 |  __repr__(self, N_CHAR_MAX=700)
 |      Return repr(self).
 |
 |  __setstate__(self, state)
 |
 |  __sklearn_clone__(self)
 |
 |  get_params(self, deep=True)
 |      Get parameters for this estimator.
 |
 |      Parameters
 |      ----------
 |      deep : bool, default=True
 |          If True, will return the parameters for this estimator and
 |          contained subobjects that are estimators.
 |
 |      Returns
 |      -------
 |      params : dict
 |          Parameter names mapped to their values.
 |
 |  set_params(self, **params)
 |      Set the parameters of this estimator.
 |
 |      The method works on simple estimators as well as on nested objects
 |      (such as :class:`~sklearn.pipeline.Pipeline`). The latter have
 |      parameters of the form ``<component>__<parameter>`` so that it's
 |      possible to update each component of a nested object.
 |
 |      Parameters
 |      ----------
 |      **params : dict
 |          Estimator parameters.
 |
 |      Returns
 |      -------
 |      self : estimator instance
 |          Estimator instance.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.utils._metadata_requests._MetadataRequester:
 |
 |  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` encapsulating
 |          routing information.
```

In [7]:

```python
# toy example
import pandas as pd

train_edu = {'educational level':['Bachelors','Masters','Bachelors','Doctorate','HS-grad','Masters']}
test_edu = {'educational level':['HS-grad','Masters','Masters','College','Bachelors']}

Xtoy_train = pd.DataFrame(train_edu)
Xtoy_test = pd.DataFrame(test_edu)

# initialize the encoder
cats = [['HS-grad','College','Bachelors','Masters','Doctorate']]

enc = OrdinalEncoder(categories = cats) # The ordered list of
# categories need to be provided. By default, the categories are alphabetically ordered!

# fit the training data
enc.fit(Xtoy_train)
# print the categories - not really important because we manually gave the ordered list of categories
print(enc.categories_)
# transform X_train. We could have used enc.fit_transform(X_train) to combine fit and transform
X_train_oe = enc.transform(Xtoy_train)
```

```
print(X_train_oe)
# transform X_test
X_test_oe = enc.transform(Xtoy_test) # OrdinalEncoder always throws an error message if
                                     # it encounters an unknown category in test
print(X_test_oe)
```

```
[array(['HS-grad', 'College', 'Bachelors', 'Masters', 'Doctorate'],
      dtype=object)]
[[2.]
 [3.]
 [2.]
 [4.]
 [0.]
 [3.]]
[[0.]
 [3.]
 [3.]
 [1.]
 [2.]]
```

In [8]:
```
# apply OE to the adult dataset
# initialize the encoder
ordinal_ftrs = ['education'] # if you have more than one ordinal feature, add the feature names here
ordinal_cats = [[' Preschool',' 1st-4th',' 5th-6th',' 7th-8th',' 9th',' 10th',' 11th',' 12th',' HS-grad',\
                ' Some-college',' Assoc-voc',' Assoc-acdm',' Bachelors',' Masters',' Prof-school',' Doctorate']]
# ordinal_cats must contain one list per ordinal feature! each list contains the ordered list of categories
# of the corresponding feature

enc = OrdinalEncoder(categories = ordinal_cats)   # By default, the categories are alphabetically ordered
                                                   # which is NOT what you want usually.

# fit the training data
enc.fit(X_train[ordinal_ftrs])  # the encoder expects a 2D array, that's why the column name is in a list

# transform X_train. We could use enc.fit_transform(X_train) to combine fit and transform
ordinal_train = enc.transform(X_train[ordinal_ftrs])
print('transformed train features:')
print(ordinal_train)
# transform X_val
ordinal_val = enc.transform(X_val[ordinal_ftrs])
print('transformed validation features:')
print(ordinal_val)
# transform X_test
ordinal_test = enc.transform(X_test[ordinal_ftrs])
print('transformed test features:')
print(ordinal_test)
```

```
transformed train features:
[[10.]
 [ 9.]
 [ 8.]
 ...
 [ 6.]
 [ 8.]
 [12.]]
transformed validation features:
[[14.]
 [13.]
 [ 9.]
 ...
 [12.]
 [ 8.]
 [ 8.]]
transformed test features:
[[12.]
 [ 9.]
 [12.]
 ...
 [ 9.]
 [ 9.]
 [11.]]
```

Quiz 1

Please explain how you would encode the race feature below and what would be the output of the encoder. Do not write code. The goal of this quiz is to test your conceptual understanding so write text and the output array.

race = [' Amer-Indian-Eskimo', 'White', 'Black', 'Asian-Pac-Islander', 'Black', 'White', 'White']

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

- apply one-hot encoding on categorical features
- apply ordinal encoding on ordinal features
- **apply scaling and normalization to continuous variables**

Continuous features: MinMaxScaler

- If the continuous feature values are reasonably bounded, MinMaxScaler is a good way to scale the features.
- Age is expected to be within the range of 0 and 100.
- Number of hours worked per week is in the range of 0 to 80.
- If unsure, plot the histogram of the feature to verify or just go with the standard scaler!

In [9]:
```
from sklearn.preprocessing import MinMaxScaler
help(MinMaxScaler)
```

```
Help on class MinMaxScaler in module sklearn.preprocessing._data:

class MinMaxScaler(sklearn.base.OneToOneFeatureMixin, sklearn.base.TransformerMixin, sklearn.base.BaseEstimator)
 |  MinMaxScaler(feature_range=(0, 1), *, copy=True, clip=False)
 |
 |  Transform features by scaling each feature to a given range.
 |
 |  This estimator scales and translates each feature individually such
 |  that it is in the given range on the training set, e.g. between
 |  zero and one.
 |
 |  The transformation is given by::
 |
 |      X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
 |      X_scaled = X_std * (max - min) + min
 |
 |  where min, max = feature_range.
 |
 |  This transformation is often used as an alternative to zero mean,
 |  unit variance scaling.
 |
 |  `MinMaxScaler` doesn't reduce the effect of outliers, but it linearly
 |  scales them down into a fixed range, where the largest occurring data point
 |  corresponds to the maximum value and the smallest one corresponds to the
 |  minimum value. For an example visualization, refer to :ref:`Compare
 |  MinMaxScaler with other scalers <plot_all_scaling_minmax_scaler_section>`.
 |
 |  Read more in the :ref:`User Guide <preprocessing_scaler>`.
 |
 |  Parameters
 |  ----------
 |  feature_range : tuple (min, max), default=(0, 1)
 |      Desired range of transformed data.
 |
 |  copy : bool, default=True
 |      Set to False to perform inplace row normalization and avoid a
 |      copy (if the input is already a numpy array).
 |
 |  clip : bool, default=False
 |      Set to True to clip transformed values of held-out data to
 |      provided `feature range`.
 |
 |      .. versionadded:: 0.24
 |
 |  Attributes
 |  ----------
 |  min_ : ndarray of shape (n_features,)
 |      Per feature adjustment for minimum. Equivalent to
 |      ``min - X.min(axis=0) * self.scale_``
 |
 |  scale_ : ndarray of shape (n_features,)
 |      Per feature relative scaling of the data. Equivalent to
 |      ``(max - min) / (X.max(axis=0) - X.min(axis=0))``
 |
 |      .. versionadded:: 0.17
 |         *scale_* attribute.
 |
 |  data_min_ : ndarray of shape (n_features,)
 |      Per feature minimum seen in the data
 |
 |      .. versionadded:: 0.17
 |         *data_min*
 |
 |  data_max_ : ndarray of shape (n_features,)
 |      Per feature maximum seen in the data
 |
 |      .. versionadded:: 0.17
 |         *data_max*
 |
 |  data_range_ : ndarray of shape (n_features,)
 |      Per feature range ``(data_max_ - data_min_)`` seen in the data
 |
 |      .. versionadded:: 0.17
 |         *data_range*
 |
 |  n_features_in_ : int
 |      Number of features seen during :term:`fit`.
 |
 |      .. versionadded:: 0.24
 |
 |  n_samples_seen_ : int
 |      The number of samples processed by the estimator.
 |      It will be reset on new calls to fit, but increments across
 |      ``partial_fit`` calls.
 |
 |  feature_names_in_ : ndarray of shape (`n_features_in_`,)
 |      Names of features seen during :term:`fit`. Defined only when `X`
```

```
 |      has feature names that are all strings.
 |
 |      .. versionadded:: 1.0
 |
 |  See Also
 |  --------
 |  minmax_scale : Equivalent function without the estimator API.
 |
 |  Notes
 |  -----
 |  NaNs are treated as missing values: disregarded in fit, and maintained in
 |  transform.
 |
 |  Examples
 |  --------
 |  >>> from sklearn.preprocessing import MinMaxScaler
 |  >>> data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]
 |  >>> scaler = MinMaxScaler()
 |  >>> print(scaler.fit(data))
 |  MinMaxScaler()
 |  >>> print(scaler.data_max_)
 |  [ 1. 18.]
 |  >>> print(scaler.transform(data))
 |  [[0.   0.  ]
 |   [0.25 0.25]
 |   [0.5  0.5 ]
 |   [1.   1.  ]]
 |  >>> print(scaler.transform([[2, 2]]))
 |  [[1.5 0. ]]
 |
 |  Method resolution order:
 |      MinMaxScaler
 |      sklearn.base.OneToOneFeatureMixin
 |      sklearn.base.TransformerMixin
 |      sklearn.utils._set_output._SetOutputMixin
 |      sklearn.base.BaseEstimator
 |      sklearn.utils._estimator_html_repr._HTMLDocumentationLinkMixin
 |      sklearn.utils._metadata_requests._MetadataRequester
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, feature_range=(0, 1), *, copy=True, clip=False)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  fit(self, X, y=None)
 |      Compute the minimum and maximum to be used for later scaling.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          The data used to compute the per-feature minimum and maximum
 |          used for later scaling along the features axis.
 |
 |      y : None
 |          Ignored.
 |
 |      Returns
 |      -------
 |      self : object
 |          Fitted scaler.
 |
 |  inverse_transform(self, X)
 |      Undo the scaling of X according to feature_range.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          Input data that will be transformed. It cannot be sparse.
 |
 |      Returns
 |      -------
 |      Xt : ndarray of shape (n_samples, n_features)
 |          Transformed data.
 |
 |  partial_fit(self, X, y=None)
 |      Online computation of min and max on X for later scaling.
 |
 |      All of X is processed as a single batch. This is intended for cases
 |      when :meth:`fit` is not feasible due to very large number of
 |      `n_samples` or because X is read from a continuous stream.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          The data used to compute the mean and standard deviation
 |          used for later scaling along the features axis.
```

```
|
|      y : None
|          Ignored.
|
|      Returns
|      -------
|      self : object
|          Fitted scaler.
|
|  transform(self, X)
|      Scale features of X according to feature_range.
|
|      Parameters
|      ----------
|      X : array-like of shape (n_samples, n_features)
|          Input data that will be transformed.
|
|      Returns
|      -------
|      Xt : ndarray of shape (n_samples, n_features)
|          Transformed data.
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __annotations__ = {'_parameter_constraints': <class 'dict'>}
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.OneToOneFeatureMixin:
|
|  get_feature_names_out(self, input_features=None)
|      Get output feature names for transformation.
|
|      Parameters
|      ----------
|      input_features : array-like of str or None, default=None
|          Input features.
|
|          - If `input_features` is `None`, then `feature_names_in_` is
|            used as feature names in. If `feature_names_in_` is not defined,
|            then the following input feature names are generated:
|            `["x0", "x1", ..., "x(n_features_in_ - 1)"]`.
|          - If `input_features` is an array-like, then `input_features` must
|            match `feature_names_in_` if `feature_names_in_` is defined.
|
|      Returns
|      -------
|      feature_names_out : ndarray of str objects
|          Same as input features.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from sklearn.base.OneToOneFeatureMixin:
|
|  __dict__
|      dictionary for instance variables
|
|  __weakref__
|      list of weak references to the object
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.TransformerMixin:
|
|  fit_transform(self, X, y=None, **fit_params)
|      Fit to data, then transform it.
|
|      Fits transformer to `X` and `y` with optional parameters `fit_params`
|      and returns a transformed version of `X`.
|
|      Parameters
|      ----------
|      X : array-like of shape (n_samples, n_features)
|          Input samples.
|
|      y :  array-like of shape (n_samples,) or (n_samples, n_outputs),                 default=None
|          Target values (None for unsupervised transformations).
|
|      **fit_params : dict
|          Additional fit parameters.
|
|      Returns
|      -------
|      X_new : ndarray array of shape (n_samples, n_features_new)
|          Transformed array.
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.utils._set_output._SetOutputMixin:
|
```

```
 |  set_output(self, *, transform=None)
 |      Set output container.
 |
 |      See :ref:`sphx_glr_auto_examples_miscellaneous_plot_set_output.py`
 |      for an example on how to use the API.
 |
 |      Parameters
 |      ----------
 |      transform : {"default", "pandas", "polars"}, default=None
 |          Configure output of `transform` and `fit_transform`.
 |
 |          - `"default"`: Default output format of a transformer
 |          - `"pandas"`: DataFrame output
 |          - `"polars"`: Polars output
 |          - `None`: Transform configuration is unchanged
 |
 |          .. versionadded:: 1.4
 |              `"polars"` option was added.
 |
 |      Returns
 |      -------
 |      self : estimator instance
 |          Estimator instance.
 |
 |  ----------------------------------------------------------------------
 |  Class methods inherited from sklearn.utils._set_output._SetOutputMixin:
 |
 |  __init_subclass__(auto_wrap_output_keys=('transform',), **kwargs)
 |      This method is called when a class is subclassed.
 |
 |      The default implementation does nothing. It may be
 |      overridden to extend subclasses.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.BaseEstimator:
 |
 |  __getstate__(self)
 |      Helper for pickle.
 |
 |  __repr__(self, N_CHAR_MAX=700)
 |      Return repr(self).
 |
 |  __setstate__(self, state)
 |
 |  __sklearn_clone__(self)
 |
 |  get_params(self, deep=True)
 |      Get parameters for this estimator.
 |
 |      Parameters
 |      ----------
 |      deep : bool, default=True
 |          If True, will return the parameters for this estimator and
 |          contained subobjects that are estimators.
 |
 |      Returns
 |      -------
 |      params : dict
 |          Parameter names mapped to their values.
 |
 |  set_params(self, **params)
 |      Set the parameters of this estimator.
 |
 |      The method works on simple estimators as well as on nested objects
 |      (such as :class:`~sklearn.pipeline.Pipeline`). The latter have
 |      parameters of the form ``<component>__<parameter>`` so that it's
 |      possible to update each component of a nested object.
 |
 |      Parameters
 |      ----------
 |      **params : dict
 |          Estimator parameters.
 |
 |      Returns
 |      -------
 |      self : estimator instance
 |          Estimator instance.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.utils._metadata_requests._MetadataRequester:
 |
 |  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` encapsulating
 |          routing information.
```

In [10]:
```
# toy data
# let's assume we have two continuous features:
train = {'age':[32,65,13,68,42,75,32],'number of hours worked':[0,40,10,60,40,20,40]}
test = {'age':[83,26,10,60],'number of hours worked':[0,40,0,60]}

# (value - min) / (max - min), if value is 32, min is 13 and max is 75, then we have 19 / 62 = 0.3064

Xtoy_train = pd.DataFrame(train)
Xtoy_test = pd.DataFrame(test)

scaler = MinMaxScaler()
scaler.fit(Xtoy_train)
print(scaler.transform(Xtoy_train))
print(scaler.transform(Xtoy_test)) # note how scaled X_test contains values larger than 1 and smaller than 0.
```

```
[[0.30645161 0.        ]
 [0.83870968 0.66666667]
 [0.         0.16666667]
 [0.88709677 1.        ]
 [0.46774194 0.66666667]
 [1.         0.33333333]
 [0.30645161 0.66666667]]
[[ 1.12903226  0.        ]
 [ 0.20967742  0.66666667]
 [-0.0483871   0.        ]
 [ 0.75806452  1.        ]]
```

In [11]:
```
# adult data

minmax_ftrs = ['age','hours-per-week']

scaler = MinMaxScaler()
scaler.fit(X_train[minmax_ftrs])
print(scaler.transform(X_train[minmax_ftrs]))
print(scaler.transform(X_val[minmax_ftrs]))
print(scaler.transform(X_test[minmax_ftrs]))
```

```
[[0.19178082 0.39795918]
 [0.32876712 0.39795918]
 [0.60273973 0.5       ]
 ...
 [0.01369863 0.19387755]
 [0.45205479 0.84693878]
 [0.23287671 0.60204082]]
[[0.35616438 0.5       ]
 [0.68493151 0.39795918]
 [0.09589041 0.39795918]
 ...
 [0.09589041 0.19387755]
 [0.02739726 0.44897959]
 [0.38356164 0.39795918]]
[[0.06849315 0.39795918]
 [0.23287671 0.39795918]
 [0.43835616 0.5       ]
 ...
 [0.20547945 0.39795918]
 [0.21917808 0.37755102]
 [0.08219178 0.35714286]]
```

Continuous features: StandardScaler

- If the continuous feature values follow a tailed distribution, StandardScaler is better to use!
- Salaries are a good example. Most people earn less than 100k but there are a small number of super-rich people.

In [12]:
```
from sklearn.preprocessing import StandardScaler
help(StandardScaler)
```

```
Help on class StandardScaler in module sklearn.preprocessing._data:

class StandardScaler(sklearn.base.OneToOneFeatureMixin, sklearn.base.TransformerMixin, sklearn.base.BaseEstimator)
 |  StandardScaler(*, copy=True, with_mean=True, with_std=True)
 |
 |  Standardize features by removing the mean and scaling to unit variance.
 |
 |  The standard score of a sample `x` is calculated as:
 |
 |      z = (x - u) / s
 |
 |  where `u` is the mean of the training samples or zero if `with_mean=False`,
 |  and `s` is the standard deviation of the training samples or one if
 |  `with_std=False`.
 |
 |  Centering and scaling happen independently on each feature by computing
 |  the relevant statistics on the samples in the training set. Mean and
 |  standard deviation are then stored to be used on later data using
 |  :meth:`transform`.
 |
 |  Standardization of a dataset is a common requirement for many
 |  machine learning estimators: they might behave badly if the
 |  individual features do not more or less look like standard normally
 |  distributed data (e.g. Gaussian with 0 mean and unit variance).
 |
 |  For instance many elements used in the objective function of
 |  a learning algorithm (such as the RBF kernel of Support Vector
 |  Machines or the L1 and L2 regularizers of linear models) assume that
 |  all features are centered around 0 and have variance in the same
 |  order. If a feature has a variance that is orders of magnitude larger
 |  than others, it might dominate the objective function and make the
 |  estimator unable to learn from other features correctly as expected.
 |
 |  `StandardScaler` is sensitive to outliers, and the features may scale
 |  differently from each other in the presence of outliers. For an example
 |  visualization, refer to :ref:`Compare StandardScaler with other scalers
 |  <plot_all_scaling_standard_scaler_section>`.
 |
 |  This scaler can also be applied to sparse CSR or CSC matrices by passing
 |  `with_mean=False` to avoid breaking the sparsity structure of the data.
 |
 |  Read more in the :ref:`User Guide <preprocessing_scaler>`.
 |
 |  Parameters
 |  ----------
 |  copy : bool, default=True
 |      If False, try to avoid a copy and do inplace scaling instead.
 |      This is not guaranteed to always work inplace; e.g. if the data is
 |      not a NumPy array or scipy.sparse CSR matrix, a copy may still be
 |      returned.
 |
 |  with_mean : bool, default=True
 |      If True, center the data before scaling.
 |      This does not work (and will raise an exception) when attempted on
 |      sparse matrices, because centering them entails building a dense
 |      matrix which in common use cases is likely to be too large to fit in
 |      memory.
 |
 |  with_std : bool, default=True
 |      If True, scale the data to unit variance (or equivalently,
 |      unit standard deviation).
 |
 |  Attributes
 |  ----------
 |  scale_ : ndarray of shape (n_features,) or None
 |      Per feature relative scaling of the data to achieve zero mean and unit
 |      variance. Generally this is calculated using `np.sqrt(var_)`. If a
 |      variance is zero, we can't achieve unit variance, and the data is left
 |      as-is, giving a scaling factor of 1. `scale_` is equal to `None`
 |      when `with_std=False`.
 |
 |      .. versionadded:: 0.17
 |         *scale_*
 |
 |  mean_ : ndarray of shape (n_features,) or None
 |      The mean value for each feature in the training set.
 |      Equal to ``None`` when ``with_mean=False`` and ``with_std=False``.
 |
 |  var_ : ndarray of shape (n_features,) or None
 |      The variance for each feature in the training set. Used to compute
 |      `scale_`. Equal to ``None`` when ``with_mean=False`` and
 |      ``with_std=False``.
 |
 |  n_features_in_ : int
 |      Number of features seen during :term:`fit`.
 |
 |      .. versionadded:: 0.24
```

```
 |
 |  feature_names_in_ : ndarray of shape (`n_features_in_`,)
 |      Names of features seen during :term:`fit`. Defined only when `X`
 |      has feature names that are all strings.
 |
 |      .. versionadded:: 1.0
 |
 |  n_samples_seen_ : int or ndarray of shape (n_features,)
 |      The number of samples processed by the estimator for each feature.
 |      If there are no missing samples, the ``n_samples_seen`` will be an
 |      integer, otherwise it will be an array of dtype int. If
 |      `sample_weights` are used it will be a float (if no missing data)
 |      or an array of dtype float that sums the weights seen so far.
 |      Will be reset on new calls to fit, but increments across
 |      ``partial_fit`` calls.
 |
 |  See Also
 |  --------
 |  scale : Equivalent function without the estimator API.
 |
 |  :class:`~sklearn.decomposition.PCA` : Further removes the linear
 |      correlation across features with 'whiten=True'.
 |
 |  Notes
 |  -----
 |  NaNs are treated as missing values: disregarded in fit, and maintained in
 |  transform.
 |
 |  We use a biased estimator for the standard deviation, equivalent to
 |  `numpy.std(x, ddof=0)`. Note that the choice of `ddof` is unlikely to
 |  affect model performance.
 |
 |  Examples
 |  --------
 |  >>> from sklearn.preprocessing import StandardScaler
 |  >>> data = [[0, 0], [0, 0], [1, 1], [1, 1]]
 |  >>> scaler = StandardScaler()
 |  >>> print(scaler.fit(data))
 |  StandardScaler()
 |  >>> print(scaler.mean_)
 |  [0.5 0.5]
 |  >>> print(scaler.transform(data))
 |  [[-1. -1.]
 |   [-1. -1.]
 |   [ 1.  1.]
 |   [ 1.  1.]]
 |  >>> print(scaler.transform([[2, 2]]))
 |  [[3. 3.]]
 |
 |  Method resolution order:
 |      StandardScaler
 |      sklearn.base.OneToOneFeatureMixin
 |      sklearn.base.TransformerMixin
 |      sklearn.utils._set_output._SetOutputMixin
 |      sklearn.base.BaseEstimator
 |      sklearn.utils._estimator_html_repr._HTMLDocumentationLinkMixin
 |      sklearn.utils._metadata_requests._MetadataRequester
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, *, copy=True, with_mean=True, with_std=True)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  fit(self, X, y=None, sample_weight=None)
 |      Compute the mean and std to be used for later scaling.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The data used to compute the mean and standard deviation
 |          used for later scaling along the features axis.
 |
 |      y : None
 |          Ignored.
 |
 |      sample_weight : array-like of shape (n_samples,), default=None
 |          Individual weights for each sample.
 |
 |          .. versionadded:: 0.24
 |             parameter *sample_weight* support to StandardScaler.
 |
 |      Returns
 |      -------
 |      self : object
 |          Fitted scaler.
 |
```

```
 |  inverse_transform(self, X, copy=None)
 |      Scale back the data to the original representation.
 |      
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The data used to scale along the features axis.
 |      copy : bool, default=None
 |          Copy the input X or not.
 |      
 |      Returns
 |      -------
 |      X_tr : {ndarray, sparse matrix} of shape (n_samples, n_features)
 |          Transformed array.
 |  
 |  partial_fit(self, X, y=None, sample_weight=None)
 |      Online computation of mean and std on X for later scaling.
 |      
 |      All of X is processed as a single batch. This is intended for cases
 |      when :meth:`fit` is not feasible due to very large number of
 |      `n_samples` or because X is read from a continuous stream.
 |      
 |      The algorithm for incremental mean and std is given in Equation 1.5a,b
 |      in Chan, Tony F., Gene H. Golub, and Randall J. LeVeque. "Algorithms
 |      for computing the sample variance: Analysis and recommendations."
 |      The American Statistician 37.3 (1983): 242-247:
 |      
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The data used to compute the mean and standard deviation
 |          used for later scaling along the features axis.
 |      
 |      y : None
 |          Ignored.
 |      
 |      sample_weight : array-like of shape (n_samples,), default=None
 |          Individual weights for each sample.
 |      
 |          .. versionadded:: 0.24
 |             parameter *sample_weight* support to StandardScaler.
 |      
 |      Returns
 |      -------
 |      self : object
 |          Fitted scaler.
 |  
 |  set_fit_request(self: sklearn.preprocessing._data.StandardScaler, *, sample_weight: Union[bool, NoneType, str] =
'$UNCHANGED$') -> sklearn.preprocessing._data.StandardScaler from sklearn.utils._metadata_requests.RequestMethod.__g
et__.<locals>
 |      Request metadata passed to the ``fit`` method.
 |      
 |      Note that this method is only relevant if
 |      ``enable_metadata_routing=True`` (see :func:`sklearn.set_config`).
 |      Please see :ref:`User Guide <metadata_routing>` on how the routing
 |      mechanism works.
 |      
 |      The options for each parameter are:
 |      
 |      - ``True``: metadata is requested, and passed to ``fit`` if provided. The request is ignored if metadata is
not provided.
 |      
 |      - ``False``: metadata is not requested and the meta-estimator will not pass it to ``fit``.
 |      
 |      - ``None``: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
 |      
 |      - ``str``: metadata should be passed to the meta-estimator with this given alias instead of the original nam
e.
 |      
 |      The default (``sklearn.utils.metadata_routing.UNCHANGED``) retains the
 |      existing request. This allows you to change the request for some
 |      parameters and not others.
 |      
 |      .. versionadded:: 1.3
 |      
 |      .. note::
 |          This method is only relevant if this estimator is used as a
 |          sub-estimator of a meta-estimator, e.g. used inside a
 |          :class:`~sklearn.pipeline.Pipeline`. Otherwise it has no effect.
 |      
 |      Parameters
 |      ----------
 |      sample_weight : str, True, False, or None,                     default=sklearn.utils.metadata_routing.UNCHAN
GED
 |          Metadata routing for ``sample_weight`` parameter in ``fit``.
 |      
 |      Returns
```

```
 |      -------
 |      self : object
 |          The updated object.
 |
 |  set_inverse_transform_request(self: sklearn.preprocessing._data.StandardScaler, *, copy: Union[bool, NoneType, s
tr] = '$UNCHANGED$') -> sklearn.preprocessing._data.StandardScaler from sklearn.utils._metadata_requests.RequestMeth
od.__get__.<locals>
 |      Request metadata passed to the ``inverse_transform`` method.
 |
 |      Note that this method is only relevant if
 |      ``enable_metadata_routing=True`` (see :func:`sklearn.set_config`).
 |      Please see :ref:`User Guide <metadata_routing>` on how the routing
 |      mechanism works.
 |
 |      The options for each parameter are:
 |
 |      - ``True``: metadata is requested, and passed to ``inverse_transform`` if provided. The request is ignored i
f metadata is not provided.
 |
 |      - ``False``: metadata is not requested and the meta-estimator will not pass it to ``inverse_transform``.
 |
 |      - ``None``: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
 |
 |      - ``str``: metadata should be passed to the meta-estimator with this given alias instead of the original nam
e.
 |
 |      The default (``sklearn.utils.metadata_routing.UNCHANGED``) retains the
 |      existing request. This allows you to change the request for some
 |      parameters and not others.
 |
 |      .. versionadded:: 1.3
 |
 |      .. note::
 |          This method is only relevant if this estimator is used as a
 |          sub-estimator of a meta-estimator, e.g. used inside a
 |          :class:`~sklearn.pipeline.Pipeline`. Otherwise it has no effect.
 |
 |      Parameters
 |      ----------
 |      copy : str, True, False, or None,                     default=sklearn.utils.metadata_routing.UNCHANGED
 |          Metadata routing for ``copy`` parameter in ``inverse_transform``.
 |
 |      Returns
 |      -------
 |      self : object
 |          The updated object.
 |
 |  set_partial_fit_request(self: sklearn.preprocessing._data.StandardScaler, *, sample_weight: Union[bool, NoneTyp
e, str] = '$UNCHANGED$') -> sklearn.preprocessing._data.StandardScaler from sklearn.utils._metadata_requests.Request
Method.__get__.<locals>
 |      Request metadata passed to the ``partial_fit`` method.
 |
 |      Note that this method is only relevant if
 |      ``enable_metadata_routing=True`` (see :func:`sklearn.set_config`).
 |      Please see :ref:`User Guide <metadata_routing>` on how the routing
 |      mechanism works.
 |
 |      The options for each parameter are:
 |
 |      - ``True``: metadata is requested, and passed to ``partial_fit`` if provided. The request is ignored if meta
data is not provided.
 |
 |      - ``False``: metadata is not requested and the meta-estimator will not pass it to ``partial_fit``.
 |
 |      - ``None``: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
 |
 |      - ``str``: metadata should be passed to the meta-estimator with this given alias instead of the original nam
e.
 |
 |      The default (``sklearn.utils.metadata_routing.UNCHANGED``) retains the
 |      existing request. This allows you to change the request for some
 |      parameters and not others.
 |
 |      .. versionadded:: 1.3
 |
 |      .. note::
 |          This method is only relevant if this estimator is used as a
 |          sub-estimator of a meta-estimator, e.g. used inside a
 |          :class:`~sklearn.pipeline.Pipeline`. Otherwise it has no effect.
 |
 |      Parameters
 |      ----------
 |      sample_weight : str, True, False, or None,                     default=sklearn.utils.metadata_routing.UNCHAN
GED
 |          Metadata routing for ``sample_weight`` parameter in ``partial_fit``.
 |
 |      Returns
```

```
 |      -------
 |      self : object
 |          The updated object.
 |
 |  set_transform_request(self: sklearn.preprocessing._data.StandardScaler, *, copy: Union[bool, NoneType, str] =
'$UNCHANGED$') -> sklearn.preprocessing._data.StandardScaler from sklearn.utils._metadata_requests.RequestMethod.__g
et__.<locals>
 |      Request metadata passed to the ``transform`` method.
 |
 |      Note that this method is only relevant if
 |      ``enable_metadata_routing=True`` (see :func:`sklearn.set_config`).
 |      Please see :ref:`User Guide <metadata_routing>` on how the routing
 |      mechanism works.
 |
 |      The options for each parameter are:
 |
 |      - ``True``: metadata is requested, and passed to ``transform`` if provided. The request is ignored if metada
ta is not provided.
 |
 |      - ``False``: metadata is not requested and the meta-estimator will not pass it to ``transform``.
 |
 |      - ``None``: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
 |
 |      - ``str``: metadata should be passed to the meta-estimator with this given alias instead of the original nam
e.
 |
 |      The default (``sklearn.utils.metadata_routing.UNCHANGED``) retains the
 |      existing request. This allows you to change the request for some
 |      parameters and not others.
 |
 |      .. versionadded:: 1.3
 |
 |      .. note::
 |          This method is only relevant if this estimator is used as a
 |          sub-estimator of a meta-estimator, e.g. used inside a
 |          :class:`~sklearn.pipeline.Pipeline`. Otherwise it has no effect.
 |
 |      Parameters
 |      ----------
 |      copy : str, True, False, or None,                     default=sklearn.utils.metadata_routing.UNCHANGED
 |          Metadata routing for ``copy`` parameter in ``transform``.
 |
 |      Returns
 |      -------
 |      self : object
 |          The updated object.
 |
 |  transform(self, X, copy=None)
 |      Perform standardization by centering and scaling.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix of shape (n_samples, n_features)
 |          The data used to scale along the features axis.
 |      copy : bool, default=None
 |          Copy the input X or not.
 |
 |      Returns
 |      -------
 |      X_tr : {ndarray, sparse matrix} of shape (n_samples, n_features)
 |          Transformed array.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __annotations__ = {'_parameter_constraints': <class 'dict'>}
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.OneToOneFeatureMixin:
 |
 |  get_feature_names_out(self, input_features=None)
 |      Get output feature names for transformation.
 |
 |      Parameters
 |      ----------
 |      input_features : array-like of str or None, default=None
 |          Input features.
 |
 |          - If `input_features` is `None`, then `feature_names_in_` is
 |            used as feature names in. If `feature_names_in_` is not defined,
 |            then the following input feature names are generated:
 |            `["x0", "x1", ..., "x(n_features_in_ - 1)"]`.
 |          - If `input_features` is an array-like, then `input_features` must
 |            match `feature_names_in_` if `feature_names_in_` is defined.
 |
 |      Returns
 |      -------
```

```
 |      feature_names_out : ndarray of str objects
 |          Same as input features.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from sklearn.base.OneToOneFeatureMixin:
 |
 |  __dict__
 |      dictionary for instance variables
 |
 |  __weakref__
 |      list of weak references to the object
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.TransformerMixin:
 |
 |  fit_transform(self, X, y=None, **fit_params)
 |      Fit to data, then transform it.
 |
 |      Fits transformer to `X` and `y` with optional parameters `fit_params`
 |      and returns a transformed version of `X`.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          Input samples.
 |
 |      y :  array-like of shape (n_samples,) or (n_samples, n_outputs),                 default=None
 |          Target values (None for unsupervised transformations).
 |
 |      **fit_params : dict
 |          Additional fit parameters.
 |
 |      Returns
 |      -------
 |      X_new : ndarray array of shape (n_samples, n_features_new)
 |          Transformed array.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.utils._set_output._SetOutputMixin:
 |
 |  set_output(self, *, transform=None)
 |      Set output container.
 |
 |      See :ref:`sphx_glr_auto_examples_miscellaneous_plot_set_output.py`
 |      for an example on how to use the API.
 |
 |      Parameters
 |      ----------
 |      transform : {"default", "pandas", "polars"}, default=None
 |          Configure output of `transform` and `fit_transform`.
 |
 |          - `"default"`: Default output format of a transformer
 |          - `"pandas"`: DataFrame output
 |          - `"polars"`: Polars output
 |          - `None`: Transform configuration is unchanged
 |
 |          .. versionadded:: 1.4
 |              `"polars"` option was added.
 |
 |      Returns
 |      -------
 |      self : estimator instance
 |          Estimator instance.
 |
 |  ----------------------------------------------------------------------
 |  Class methods inherited from sklearn.utils._set_output._SetOutputMixin:
 |
 |  __init_subclass__(auto_wrap_output_keys=('transform',), **kwargs)
 |      This method is called when a class is subclassed.
 |
 |      The default implementation does nothing. It may be
 |      overridden to extend subclasses.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.BaseEstimator:
 |
 |  __getstate__(self)
 |      Helper for pickle.
 |
 |  __repr__(self, N_CHAR_MAX=700)
 |      Return repr(self).
 |
 |  __setstate__(self, state)
 |
 |  __sklearn_clone__(self)
 |
 |  get_params(self, deep=True)
```

```
 |      Get parameters for this estimator.
 |      
 |      Parameters
 |      ----------
 |      deep : bool, default=True
 |          If True, will return the parameters for this estimator and
 |          contained subobjects that are estimators.
 |      
 |      Returns
 |      -------
 |      params : dict
 |          Parameter names mapped to their values.
 |  
 |  set_params(self, **params)
 |      Set the parameters of this estimator.
 |      
 |      The method works on simple estimators as well as on nested objects
 |      (such as :class:`~sklearn.pipeline.Pipeline`). The latter have
 |      parameters of the form ``<component>__<parameter>`` so that it's
 |      possible to update each component of a nested object.
 |      
 |      Parameters
 |      ----------
 |      **params : dict
 |          Estimator parameters.
 |      
 |      Returns
 |      -------
 |      self : estimator instance
 |          Estimator instance.
 |  
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.utils._metadata_requests._MetadataRequester:
 |  
 |  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` encapsulating
 |          routing information.
```

In [13]:

```python
# toy data
train = {'salary':[50_000,75_000,40_000,1_000_000,30_000,250_000,35_000,45_000]}
test = {'salary':[25_000,55_000,1_500_000,60_000]}

Xtoy_train = pd.DataFrame(train)
Xtoy_test = pd.DataFrame(test)

scaler = StandardScaler()
print(scaler.fit_transform(Xtoy_train))
print(scaler.transform(Xtoy_test))
```

```
[[-0.44873188]
 [-0.36895732]
 [-0.4806417 ]
 [ 2.58270127]
 [-0.51255153]
 [ 0.18946457]
 [-0.49659661]
 [-0.46468679]]
[[-0.52850644]
 [-0.43277697]
 [ 4.1781924 ]
 [-0.41682206]]
```

In [14]:

```python
# adult data

std_ftrs = ['capital-gain','capital-loss']
scaler = StandardScaler()
print(scaler.fit_transform(X_train[std_ftrs]))
print(scaler.transform(X_val[std_ftrs]))
print(scaler.transform(X_test[std_ftrs]))
```

```
[[-0.14633293 -0.22318878]
 [-0.14633293 -0.22318878]
 [-0.14633293 -0.22318878]
 ...
 [-0.14633293 -0.22318878]
 [-0.14633293 -0.22318878]
 [-0.14633293 -0.22318878]]
[[-0.14633293 -0.22318878]
 [-0.14633293 -0.22318878]
 [-0.14633293 -0.22318878]
 ...
 [-0.14633293 -0.22318878]
 [-0.14633293 -0.22318878]
 [-0.14633293 -0.22318878]]
[[-0.14633293 -0.22318878]
 [-0.14633293 -0.22318878]
 [-0.14633293 -0.22318878]
 ...
 [-0.14633293 -0.22318878]
 [-0.14633293 -0.22318878]
 [-0.14633293 -0.22318878]]
```

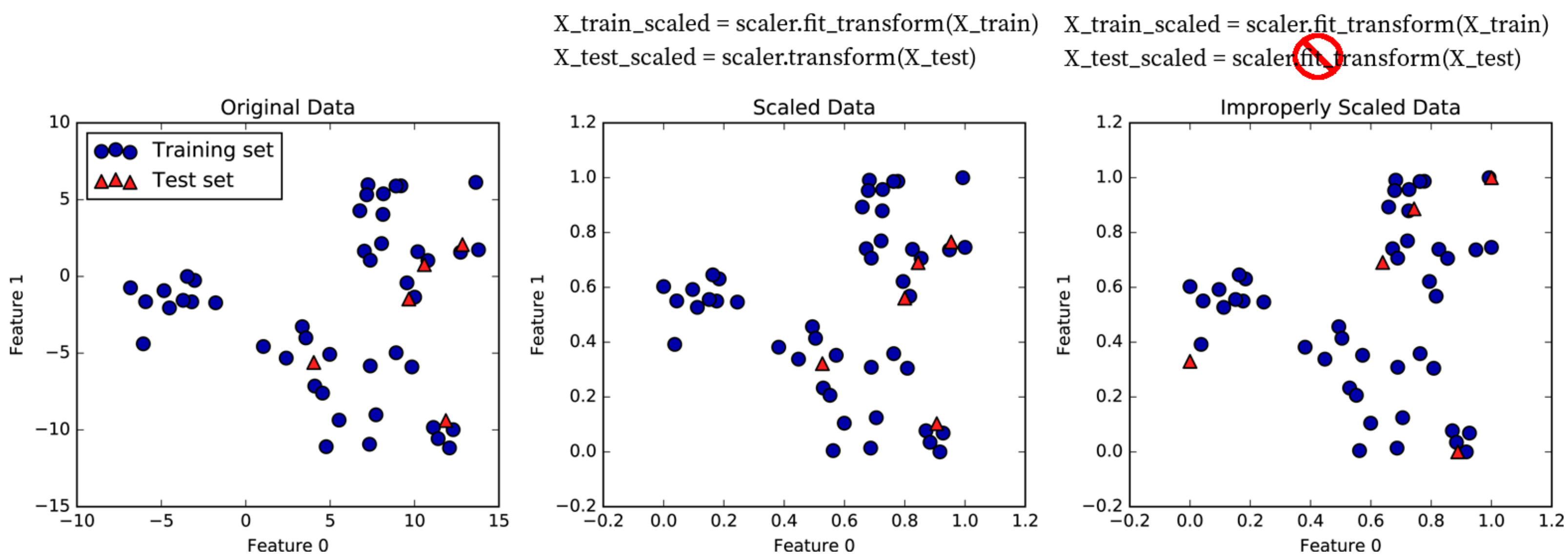
Quiz 2

Which of these features could be safely preprocessed by the minmax scaler?

- number of minutes spent on the website in a day
- number of days a year spent abroad in a year
- USD donated to charity

How and when to do preprocessing in the ML pipeline?

- **APPLY TRANSFORMER.FIT ONLY ON YOUR TRAINING DATA!** Then transform the validation and test sets.
- One of the most common mistake practitioners make is leaking statistics!
  - fit_transform is applied to the whole dataset, then the data is split into train/validation/test
    - this is wrong because the test set statistics impacts how the training and validation sets are transformed
    - but the test set must be separated by train and val, and val must be separated by train
  - or fit_transform is applied to the train, then fit_transform is applied to the validation set, and fit_transform is applied to the test set
    - this is wrong because the relative position of the points change



Scikit-learn's pipelines

- The steps in the ML pipleine can be chained together into a scikit-learn pipeline which consists of transformers and one final estimator which is usually your classifier or regression model.
- It neatly combines the preprocessing steps and it helps to avoid leaking statistics.

https://scikit-learn.org/stable/auto_examples/compose/plot_column_transformer_mixed_types.html

In [15]:

```
import pandas as pd
import numpy as np

from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder, OrdinalEncoder, MinMaxScaler
from sklearn.model_selection import train_test_split

#np.random.seed(0)

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

```python
# 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 less than 50k
X = df.loc[:, df.columns != 'gross-income'] # all other columns are features

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)

# 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)
```

In [16]:

```python
# collect which encoder to use on each feature
# needs to be done manually
ordinal_ftrs = ['education']
ordinal_cats = [[' Preschool',' 1st-4th',' 5th-6th',' 7th-8th',' 9th',' 10th',' 11th',' 12th',' HS-grad',\
                ' Some-college',' Assoc-voc',' Assoc-acdm',' Bachelors',' Masters',' Prof-school',' Doctorate']]
onehot_ftrs = ['workclass','marital-status','occupation','relationship','race','sex','native-country']
minmax_ftrs = ['age','hours-per-week']
std_ftrs = ['capital-gain','capital-loss']

# collect all the encoders
preprocessor = ColumnTransformer(
    transformers=[
        ('ord', OrdinalEncoder(categories = ordinal_cats), ordinal_ftrs),
        ('onehot', OneHotEncoder(sparse_output=False,handle_unknown='ignore'), onehot_ftrs),
        ('minmax', MinMaxScaler(), minmax_ftrs),
        ('std', StandardScaler(), std_ftrs)])

clf = Pipeline(steps=[('preprocessor', preprocessor)]) # for now we only preprocess
                                                       # later on we will add other steps here

X_train_prep = clf.fit_transform(X_train)
X_val_prep = clf.transform(X_val)
X_test_prep = clf.transform(X_test)

print(X_train.shape)
print(X_train_prep.shape)
print(X_train_prep)
```

```
(19536, 14)
(19536, 91)
[[10.          0.          0.         ...  0.39795918 -0.14633293
  -0.22318878]
 [ 9.          0.          0.         ...  0.39795918 -0.14633293
  -0.22318878]
 [ 8.          0.          0.         ...  0.5        -0.14633293
  -0.22318878]
 ...
 [ 6.          0.          0.         ...  0.19387755 -0.14633293
  -0.22318878]
 [ 8.          0.          0.         ...  0.84693878 -0.14633293
  -0.22318878]
 [12.          0.          0.         ...  0.60204082 -0.14633293
  -0.22318878]]
```

Mudcard

In [ ]: