Preprocessing

Index:

1. Importing libraries
2. Importing datasets
3. Subsetting on attributes
4. Dealing with Missing data

Importing libraries

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
print("Done")
```

Done

Importing dataset

In [11]:

```
# Data file in the same place as the code file - no need to give whole filepath
data = pd.read_csv("Data.csv")
data
```

Out[11]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

In [17]:

```
# Subsetting into independent and dependent variables
X = data.iloc[:,:-1].values # numpy array
Y = data.iloc[:,-1].values # to get numpy array
```

Note: In this, the '.values' is important, else it gives trouble in later stages with the shape of the array and stuff.

In [18]:

```
X
```

Out[18]:

```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, nan],
       ['France', 35.0, 58000.0],
       ['Spain', nan, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

In [19]:

```
Y
```

Out[19]:

```
array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'],
      dtype=object)
```

Handling missing data

Python does not have 'mice' corresponding to mice in R. Work on this part more.

Check for missing values

In [25]:

```
# Check for missing values
data_missing_count = data.isna().sum()
data_missing_count
```

Out[25]:

```
Country      0
Age          1
Salary       1
Purchased    0
dtype: int64
```

In [30]:

```
data.isna()
```

Out[30]:

	Country	Age	Salary	Purchased
0	False	False	False	False
1	False	False	False	False
2	False	False	False	False
3	False	False	False	False
4	False	False	True	False
5	False	False	False	False
6	False	True	False	False
7	False	False	False	False
8	False	False	False	False
9	False	False	False	False

Dropping missing values

Deleting the observation - if dataset is large and < 1% data is missing.

In [33]:

```
data_drop = data.dropna()
data_drop
```

Out[33]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
5	France	35.0	58000.0	Yes
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

Imputation

In [35]:

```
from sklearn.impute import SimpleImputer
```

In [36]:

```
help(SimpleImputer)
```

```
Help on class SimpleImputer in module sklearn.impute._base:

class SimpleImputer(_BaseImputer)
 |  SimpleImputer(missing_values=nan, strategy='mean', fill_value=None, ve
rbose=0, copy=True, add_indicator=False)
 |  
 |  Imputation transformer for completing missing values.
 |  
 |  Read more in the :ref:`User Guide <impute>`.
 |  
 |  Parameters
 |  ----------
 |  missing_values : number, string, np.nan (default) or None
 |      The placeholder for the missing values. All occurrences of
 |      `missing_values` will be imputed.
 |  
 |  strategy : string, default='mean'
 |      The imputation strategy.
 |  
 |      - If "mean", then replace missing values using the mean along
 |        each column. Can only be used with numeric data.
 |      - If "median", then replace missing values using the median along
 |        each column. Can only be used with numeric data.
 |      - If "most_frequent", then replace missing using the most frequent
 |        value along each column. Can be used with strings or numeric dat
a.
 |      - If "constant", then replace missing values with fill_value. Can
be
 |        used with strings or numeric data.
 |  
 |      .. versionadded:: 0.20
 |         strategy="constant" for fixed value imputation.
 |  
 |  fill_value : string or numerical value, default=None
 |      When strategy == "constant", fill_value is used to replace all
 |      occurrences of missing_values.
 |      If left to the default, fill_value will be 0 when imputing numeric
al
 |      data and "missing_value" for strings or object data types.
 |  
 |  verbose : integer, default=0
 |      Controls the verbosity of the imputer.
 |  
 |  copy : boolean, default=True
 |      If True, a copy of X will be created. If False, imputation will
 |      be done in-place whenever possible. Note that, in the following ca
ses,
 |      a new copy will always be made, even if `copy=False`:
 |  
 |      - If X is not an array of floating values;
 |      - If X is encoded as a CSR matrix;
 |      - If add_indicator=True.
 |  
 |  add_indicator : boolean, default=False
 |      If True, a :class:`MissingIndicator` transform will stack onto out
put
 |      of the imputer's transform. This allows a predictive estimator
 |      to account for missingness despite imputation. If a feature has no
 |      missing values at fit/train time, the feature won't appear on
 |      the missing indicator even if there are missing values at
 |      transform/test time.
```

```
 |  
 |  Attributes
 |  ----------
 |  statistics_ : array of shape (n_features,)
 |      The imputation fill value for each feature.
 |      Computing statistics can result in `np.nan` values.
 |      During :meth:`transform`, features corresponding to `np.nan`
 |      statistics will be discarded.
 |  
 |  indicator_ : :class:`sklearn.impute.MissingIndicator`
 |      Indicator used to add binary indicators for missing values.
 |      ``None`` if add_indicator is False.
 |  
 |  See also
 |  --------
 |  IterativeImputer : Multivariate imputation of missing values.
 |  
 |  Examples
 |  --------
 |  >>> import numpy as np
 |  >>> from sklearn.impute import SimpleImputer
 |  >>> imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean')
 |  >>> imp_mean.fit([[7, 2, 3], [4, np.nan, 6], [10, 5, 9]])
 |  SimpleImputer()
 |  >>> X = [[np.nan, 2, 3], [4, np.nan, 6], [10, np.nan, 9]]
 |  >>> print(imp_mean.transform(X))
 |  [[ 7.   2.   3. ]
 |   [ 4.   3.5  6. ]
 |   [10.   3.5  9. ]]
 |  
 |  Notes
 |  -----
 |  Columns which only contained missing values at :meth:`fit` are discard
ed
 |  upon :meth:`transform` if strategy is not "constant".
 |  
 |  Method resolution order:
 |      SimpleImputer
 |      _BaseImputer
 |      sklearn.base.TransformerMixin
 |      sklearn.base.BaseEstimator
 |      builtins.object
 |  
 |  Methods defined here:
 |  
 |  __init__(self, missing_values=nan, strategy='mean', fill_value=None, v
erbose=0, copy=True, add_indicator=False)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |  
 |  fit(self, X, y=None)
 |      Fit the imputer on X.
 |      
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix}, shape (n_samples, n_features)
 |          Input data, where ``n_samples`` is the number of samples and
 |          ``n_features`` is the number of features.
 |      
 |      Returns
 |      -------
 |      self : SimpleImputer
```

```
 |
 |  transform(self, X)
 |      Impute all missing values in X.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix}, shape (n_samples, n_features)
 |          The input data to complete.
 |
 |  ----------------------------------------------------------------------
 |  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 : numpy array of shape [n_samples, n_features]
 |          Training set.
 |
 |      y : numpy array of shape [n_samples]
 |          Target values.
 |
 |      **fit_params : dict
 |          Additional fit parameters.
 |
 |      Returns
 |      -------
 |      X_new : numpy array of shape [n_samples, n_features_new]
 |          Transformed array.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from sklearn.base.TransformerMixin:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.BaseEstimator:
 |
 |  __getstate__(self)
 |
 |  __repr__(self, N_CHAR_MAX=700)
 |      Return repr(self).
 |
 |  __setstate__(self, state)
 |
 |  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 : mapping of string to any
 |          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 pipelines). 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 : object
 |          Estimator instance.
```

In [37]:

```
imputer = SimpleImputer()
imputer.fit(X[:,1:3])
X[:,1:3] = imputer.transform(X[:,1:3])
```

In [38]:

```
X
```

Out[38]:

```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, 63777.77777777778],
       ['France', 35.0, 58000.0],
       ['Spain', 38.77777777777778, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

Encoding categorical variables

Case 1 - when you cannot directly encode the data into numerical labels

In [39]:

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
```

In [40]:

```
help(ColumnTransformer)
```

```
Help on class ColumnTransformer in module sklearn.compose._column_transfor
mer:

class ColumnTransformer(sklearn.base.TransformerMixin, sklearn.utils.metae
stimators._BaseComposition)
 |  ColumnTransformer(transformers, remainder='drop', sparse_threshold=0.
3, n_jobs=None, transformer_weights=None, verbose=False)
 |
 |  Applies transformers to columns of an array or pandas DataFrame.
 |
 |  This estimator allows different columns or column subsets of the input
 |  to be transformed separately and the features generated by each transf
ormer
 |  will be concatenated to form a single feature space.
 |  This is useful for heterogeneous or columnar data, to combine several
 |  feature extraction mechanisms or transformations into a single transfo
rmer.
 |
 |  Read more in the :ref:`User Guide <column_transformer>`.
 |
 |  .. versionadded:: 0.20
 |
 |  Parameters
 |  ----------
 |  transformers : list of tuples
 |      List of (name, transformer, column(s)) tuples specifying the
 |      transformer objects to be applied to subsets of the data.
 |
 |      name : string
 |          Like in Pipeline and FeatureUnion, this allows the transformer
and
 |          its parameters to be set using ``set_params`` and searched in
grid
 |          search.
 |      transformer : estimator or {'passthrough', 'drop'}
 |          Estimator must support :term:`fit` and :term:`transform`.
 |          Special-cased strings 'drop' and 'passthrough' are accepted as
 |          well, to indicate to drop the columns or to pass them through
 |          untransformed, respectively.
 |      column(s) : string or int, array-like of string or int, slice, boo
lean mask array or callable
 |          Indexes the data on its second axis. Integers are interpreted
as
 |          positional columns, while strings can reference DataFrame colu
mns
 |          by name.  A scalar string or int should be used where
 |          ``transformer`` expects X to be a 1d array-like (vector),
 |          otherwise a 2d array will be passed to the transformer.
 |          A callable is passed the input data `X` and can return any of
the
 |          above. To select multiple columns by name or dtype, you can us
e
 |          :obj:`make_column_transformer`.
 |
 |  remainder : {'drop', 'passthrough'} or estimator, default 'drop'
 |      By default, only the specified columns in `transformers` are
 |      transformed and combined in the output, and the non-specified
 |      columns are dropped. (default of ``'drop'``).
 |      By specifying ``remainder='passthrough'``, all remaining columns t
hat
 |      were not specified in `transformers` will be automatically passed
```

```
 |      through. This subset of columns is concatenated with the output of
 |      the transformers.
 |      By setting ``remainder`` to be an estimator, the remaining
 |      non-specified columns will use the ``remainder`` estimator. The
 |      estimator must support :term:`fit` and :term:`transform`.
 |      Note that using this feature requires that the DataFrame columns
 |      input at :term:`fit` and :term:`transform` have identical order.
 |  
 |  sparse_threshold : float, default = 0.3
 |      If the output of the different transformers contains sparse matric
es,
 |      these will be stacked as a sparse matrix if the overall density is
 |      lower than this value. Use ``sparse_threshold=0`` to always return
 |      dense.  When the transformed output consists of all dense data, th
e
 |      stacked result will be dense, and this keyword will be ignored.
 |  
 |  n_jobs : int or None, optional (default=None)
 |      Number of jobs to run in parallel.
 |      ``None`` means 1 unless in a :obj:`joblib.parallel_backend` contex
t.
 |      ``-1`` means using all processors. See :term:`Glossary <n_jobs>`
 |      for more details.
 |  
 |  transformer_weights : dict, optional
 |      Multiplicative weights for features per transformer. The output of
the
 |      transformer is multiplied by these weights. Keys are transformer n
ames,
 |      values the weights.
 |  
 |  verbose : boolean, optional(default=False)
 |      If True, the time elapsed while fitting each transformer will be
 |      printed as it is completed.
 |  
 |  Attributes
 |  ----------
 |  transformers_ : list
 |      The collection of fitted transformers as tuples of
 |      (name, fitted_transformer, column). `fitted_transformer` can be an
 |      estimator, 'drop', or 'passthrough'. In case there were no columns
 |      selected, this will be the unfitted transformer.
 |      If there are remaining columns, the final element is a tuple of th
e
 |      form:
 |      ('remainder', transformer, remaining_columns) corresponding to the
 |      ``remainder`` parameter. If there are remaining columns, then
 |      ``len(transformers_)==len(transformers)+1``, otherwise
 |      ``len(transformers_)==len(transformers)``.
 |  
 |  named_transformers_ : Bunch object, a dictionary with attribute access
 |      Read-only attribute to access any transformer by given name.
 |      Keys are transformer names and values are the fitted transformer
 |      objects.
 |  
 |  sparse_output_ : boolean
 |      Boolean flag indicating wether the output of ``transform`` is a
 |      sparse matrix or a dense numpy array, which depends on the output
 |      of the individual transformers and the `sparse_threshold` keyword.
 |  
 |  Notes
```

```
 |  -----
 |  The order of the columns in the transformed feature matrix follows the
 |  order of how the columns are specified in the `transformers` list.
 |  Columns of the original feature matrix that are not specified are
 |  dropped from the resulting transformed feature matrix, unless specifie
d
 |  in the `passthrough` keyword. Those columns specified with `passthroug
h`
 |  are added at the right to the output of the transformers.
 |
 |  See also
 |  --------
 |  sklearn.compose.make_column_transformer : convenience function for
 |      combining the outputs of multiple transformer objects applied to
 |      column subsets of the original feature space.
 |  sklearn.compose.make_column_selector : convenience function for select
ing
 |      columns based on datatype or the columns name with a regex patter
n.
 |
 |  Examples
 |  --------
 |  >>> import numpy as np
 |  >>> from sklearn.compose import ColumnTransformer
 |  >>> from sklearn.preprocessing import Normalizer
 |  >>> ct = ColumnTransformer(
 |  ...     [("norm1", Normalizer(norm='l1'), [0, 1]),
 |  ...      ("norm2", Normalizer(norm='l1'), slice(2, 4))])
 |  >>> X = np.array([[0., 1., 2., 2.],
 |  ...               [1., 1., 0., 1.]])
 |  >>> # Normalizer scales each row of X to unit norm. A separate scaling
 |  >>> # is applied for the two first and two last elements of each
 |  >>> # row independently.
 |  >>> ct.fit_transform(X)
 |  array([[0. , 1. , 0.5, 0.5],
 |         [0.5, 0.5, 0. , 1. ]])
 |
 |  Method resolution order:
 |      ColumnTransformer
 |      sklearn.base.TransformerMixin
 |      sklearn.utils.metaestimators._BaseComposition
 |      sklearn.base.BaseEstimator
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, transformers, remainder='drop', sparse_threshold=0.3, n
_jobs=None, transformer_weights=None, verbose=False)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  fit(self, X, y=None)
 |      Fit all transformers using X.
 |
 |      Parameters
 |      ----------
 |      X : array-like or DataFrame of shape [n_samples, n_features]
 |          Input data, of which specified subsets are used to fit the
 |          transformers.
 |
 |      y : array-like, shape (n_samples, ...), optional
 |          Targets for supervised learning.
```

```
 |  
 |      Returns
 |      -------
 |      self : ColumnTransformer
 |          This estimator
 |  
 |  fit_transform(self, X, y=None)
 |      Fit all transformers, transform the data and concatenate results.
 |  
 |      Parameters
 |      ----------
 |      X : array-like or DataFrame of shape [n_samples, n_features]
 |          Input data, of which specified subsets are used to fit the
 |          transformers.
 |  
 |      y : array-like, shape (n_samples, ...), optional
 |          Targets for supervised learning.
 |  
 |      Returns
 |      -------
 |      X_t : array-like or sparse matrix, shape (n_samples, sum_n_compone
nts)
 |          hstack of results of transformers. sum_n_components is the
 |          sum of n_components (output dimension) over transformers. If
 |          any result is a sparse matrix, everything will be converted to
 |          sparse matrices.
 |  
 |  get_feature_names(self)
 |      Get feature names from all transformers.
 |  
 |      Returns
 |      -------
 |      feature_names : list of strings
 |          Names of the features produced by transform.
 |  
 |  get_params(self, deep=True)
 |      Get parameters for this estimator.
 |  
 |      Parameters
 |      ----------
 |      deep : boolean, optional
 |          If True, will return the parameters for this estimator and
 |          contained subobjects that are estimators.
 |  
 |      Returns
 |      -------
 |      params : mapping of string to any
 |          Parameter names mapped to their values.
 |  
 |  set_params(self, **kwargs)
 |      Set the parameters of this estimator.
 |  
 |      Valid parameter keys can be listed with ``get_params()``.
 |  
 |      Returns
 |      -------
 |      self
 |  
 |  transform(self, X)
 |      Transform X separately by each transformer, concatenate results.
 |  
```

```
 |      Parameters
 |      ----------
 |      X : array-like or DataFrame of shape [n_samples, n_features]
 |          The data to be transformed by subset.
 |
 |      Returns
 |      -------
 |      X_t : array-like or sparse matrix, shape (n_samples, sum_n_compone
nts)
 |          hstack of results of transformers. sum_n_components is the
 |          sum of n_components (output dimension) over transformers. If
 |          any result is a sparse matrix, everything will be converted to
 |          sparse matrices.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  named_transformers_
 |      Access the fitted transformer by name.
 |
 |      Read-only attribute to access any transformer by given name.
 |      Keys are transformer names and values are the fitted transformer
 |      objects.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __abstractmethods__ = frozenset()
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from sklearn.base.TransformerMixin:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.BaseEstimator:
 |
 |  __getstate__(self)
 |
 |  __repr__(self, N_CHAR_MAX=700)
 |      Return repr(self).
 |
 |  __setstate__(self, state)
```

In [44]:

```
ct = ColumnTransformer(transformers =[('encoder',OneHotEncoder(),[0])],remainder='passt
hrough')
X = ct.fit_transform(X) # np.array(ct.fit_transform(X))
X
```

Out[44]:

```
array([[0.0, 1.0, 0.0, 0.0, 44.0, 72000.0],
       [1.0, 0.0, 0.0, 1.0, 27.0, 48000.0],
       [1.0, 0.0, 1.0, 0.0, 30.0, 54000.0],
       [1.0, 0.0, 0.0, 1.0, 38.0, 61000.0],
       [1.0, 0.0, 1.0, 0.0, 40.0, 63777.77777777778],
       [0.0, 1.0, 0.0, 0.0, 35.0, 58000.0],
       [1.0, 0.0, 0.0, 1.0, 38.77777777777778, 52000.0],
       [0.0, 1.0, 0.0, 0.0, 48.0, 79000.0],
       [1.0, 0.0, 1.0, 0.0, 50.0, 83000.0],
       [0.0, 1.0, 0.0, 0.0, 37.0, 67000.0]], dtype=object)
```

In [45]:

```
type(X)
```

Out[45]:

```
numpy.ndarray
```

Case -2 : Label Encoding

In [49]:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
Y = le.fit_transform(Y)
```

In [50]:

```
Y
```

Out[50]:

```
array([0, 1, 0, 0, 1, 1, 0, 1, 0, 1])
```

Train - test Split

Feature scaling after test-train split

Test set is brand new dataset, which you are not supposed to work with while training. If feature scaling is done before the split, the mean, sd, min, max, all these things are affected be the test data also, which should not happen, because we do not know anything about the future data in production.

Feature scaling before will cause information leakage from test split into the model.

In [52]:

```
from sklearn.model_selection import train_test_split
```

In [53]:

```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y, test_size = 0.2, random_state = 1
023)
```

In [55]:

```
X_train
```

Out[55]:

```
array([[0.0, 1.0, 0.0, 0.0, 44.0, 72000.0],
       [0.0, 1.0, 0.0, 0.0, 37.0, 67000.0],
       [1.0, 0.0, 1.0, 0.0, 30.0, 54000.0],
       [0.0, 1.0, 0.0, 0.0, 35.0, 58000.0],
       [1.0, 0.0, 1.0, 0.0, 40.0, 63777.77777777778],
       [1.0, 0.0, 1.0, 0.0, 50.0, 83000.0],
       [1.0, 0.0, 0.0, 1.0, 38.0, 61000.0],
       [0.0, 1.0, 0.0, 0.0, 48.0, 79000.0]], dtype=object)
```

In [56]:

```
X_test
```

Out[56]:

```
array([[1.0, 0.0, 0.0, 1.0, 27.0, 48000.0],
       [1.0, 0.0, 0.0, 1.0, 38.77777777777778, 52000.0]], dtype=object)
```

In [57]:

```
Y_train
```

Out[57]:

```
array([0, 1, 0, 1, 1, 0, 0, 1])
```

In [58]:

```
Y_test
```

Out[58]:

```
array([1, 0])
```

Feature Scaling

Not needed for all of the models, only some.

Needed in so that some of the features are not dominated by the others.

Scaling

1. Normal-transformation (Standardization)
2. Min-max - transformation (Normalization)

'1' is recommended when most features have a normal distribution

'2' works almost all the times

In [62]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train[:,-2:] = sc.fit_transform(X_train[:,-2:])
X_test[:,-2:] = sc.transform(X_test[:,-2:])
```

Scaling of dummy vairables? - Nope, role of scaling is to reduce the range of the feature, and dummy variables already are in the lower range. Also, Scalling dummy vairables, you loose interpretation.

In [63]:

```
X_train
```

Out[63]:

```
array([[0.0, 1.0, 0.0, 0.0, 0.5952568425994088, 0.5032769329433617],
       [0.0, 1.0, 0.0, 0.0, -0.5158892635861543, -0.023408229439225773],
       [1.0, 0.0, 1.0, 0.0, -1.6270353697717175, -1.3927896516339533],
       [0.0, 1.0, 0.0, 0.0, -0.8333595796391723, -0.9714415217278833],
       [1.0, 0.0, 1.0, 0.0, -0.03968378950662725, -0.36282755630800406],
       [1.0, 0.0, 1.0, 0.0, 1.5476677907584628, 1.6619842901850543],
       [1.0, 0.0, 0.0, 1.0, -0.3571541055596453, -0.6554304242983308],
       [0.0, 1.0, 0.0, 0.0, 1.230197474705445, 1.2406361602789844]],
      dtype=object)
```

In [64]:

```
X_test
```

Out[64]:

```
array([[1.0, 0.0, 0.0, 1.0, -2.1032408438512444, -2.0248118464930585],
       [1.0, 0.0, 0.0, 1.0, -0.2336934270945826, -1.6034637165869883]],
      dtype=object)
```