Data Preprocessing
Features engineering

Dr Ahmed Rebai
Esprit School of Engineering
2019/2020

Building good training sets: Data preprocessing

- Dealing with missing data
- Handling categorical data
- Partitioning a dataset into separate training and test sets
- Bringing features onto the same scale
- Selecting meaningful features
- Assessing feature importance with random forests

•Dealing with missing data

- Identifying missing values in tabular data
- Eliminating samples or features with missing values
- Imputing missing values

Identifying missing values in tabular data

```
>>> import pandas as pd
>>> from io import StringIO

>>> csv_data = \
... '''A,B,C,D
... 1.0,2.0,3.0,4.0
... 5.0,6.0,,8.0
... 10.0,11.0,12.0,'''
>>> # If you are using Python 2.7, you need
>>> # to convert the string to unicode:
>>> # csv_data = unicode(csv_data)
>>> df = pd.read_csv(StringIO(csv_data))
>>> df
      A     B     C    D
0  1.0   2.0   3.0  4.0
1  5.0   6.0   NaN  8.0
2 10.0  11.0  12.0  NaN
```

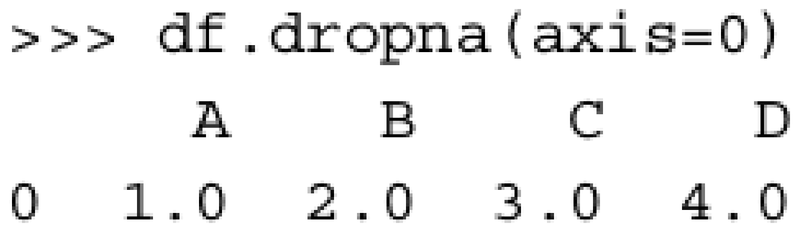
Identifying missing values in tabular data

```
>>> df.isnull().sum()
A    0
B    0
C    1
D    1
dtype: int64
```

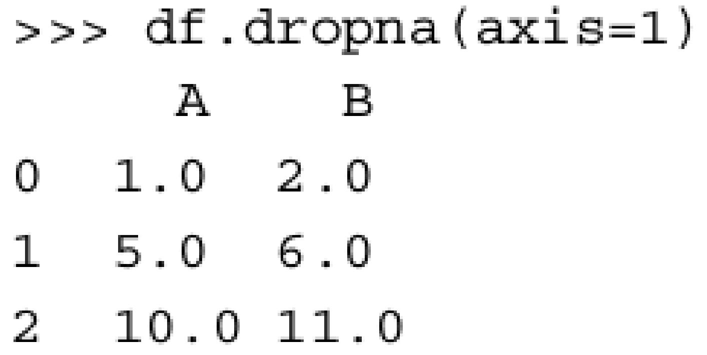
How a pandas dataframe works?

```
>>> df.values
array([[  1.,   2.,   3.,   4.],
       [  5.,   6.,  nan,   8.],
       [ 10.,  11.,  12.,  nan]])
```

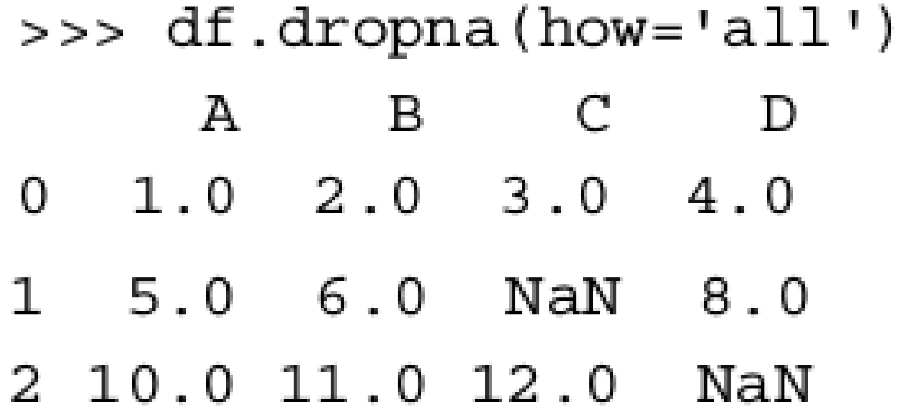
Eliminating samples or features with missing values



```
>>> df.dropna(axis=0)
     A    B    C    D
0  1.0  2.0  3.0  4.0
```



```
>>> df.dropna(axis=1)
      A     B
0   1.0   2.0
1   5.0   6.0
2  10.0  11.0
```



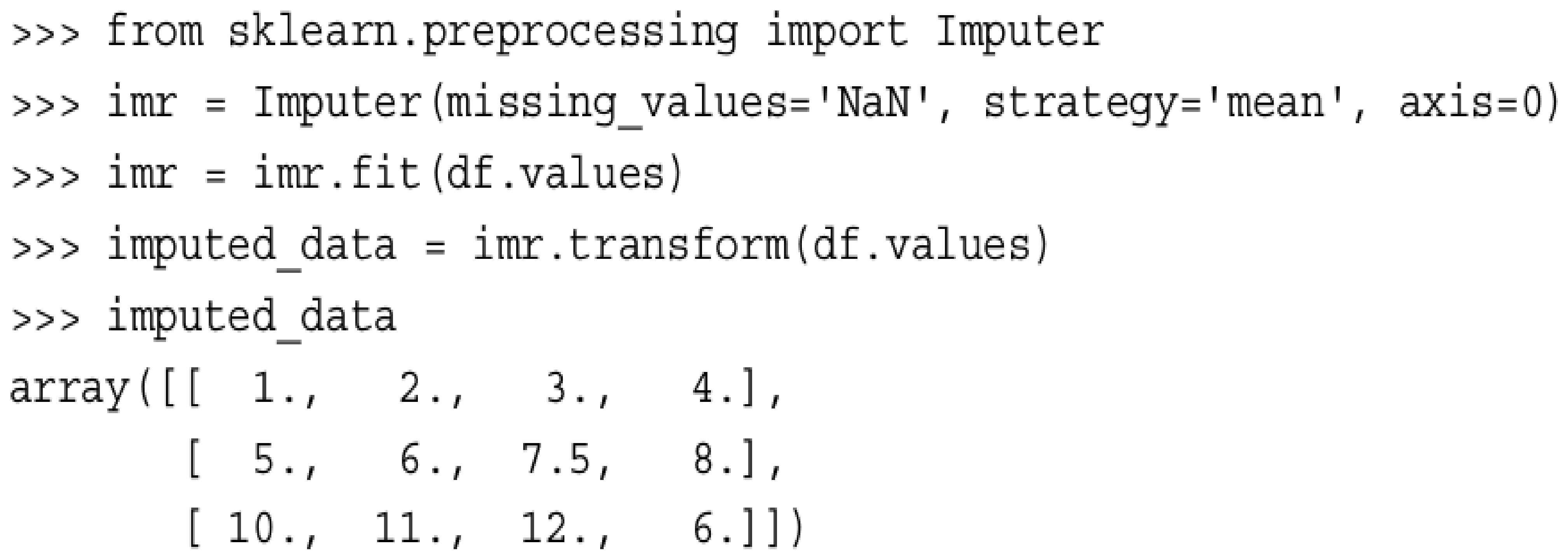
```
>>> df.dropna(how='all')
      A     B     C    D
0   1.0   2.0   3.0  4.0
1   5.0   6.0   NaN  8.0
2  10.0  11.0  12.0  NaN
```

Eliminating samples or features with missing values

```
# drop rows that have less than 4 real values
>>> df.dropna(thresh=4)
     A    B    C    D
0  1.0  2.0  3.0  4.0
```

```
# only drop rows where NaN appear in specific columns (here: 'C')
>>> df.dropna(subset=['C'])
     A    B    C   D
0  1.0  2.0  3.0 4.0
2 10.0 11.0 12.0 NaN
```

Imputing missing values



```
>>> from sklearn.preprocessing import Imputer
>>> imr = Imputer(missing_values='NaN', strategy='mean', axis=0)
>>> imr = imr.fit(df.values)
>>> imputed_data = imr.transform(df.values)
>>> imputed_data
array([[  1.,   2.,   3.,   4.],
       [  5.,   6.,  7.5,   8.],
       [ 10.,  11.,  12.,   6.]])
```

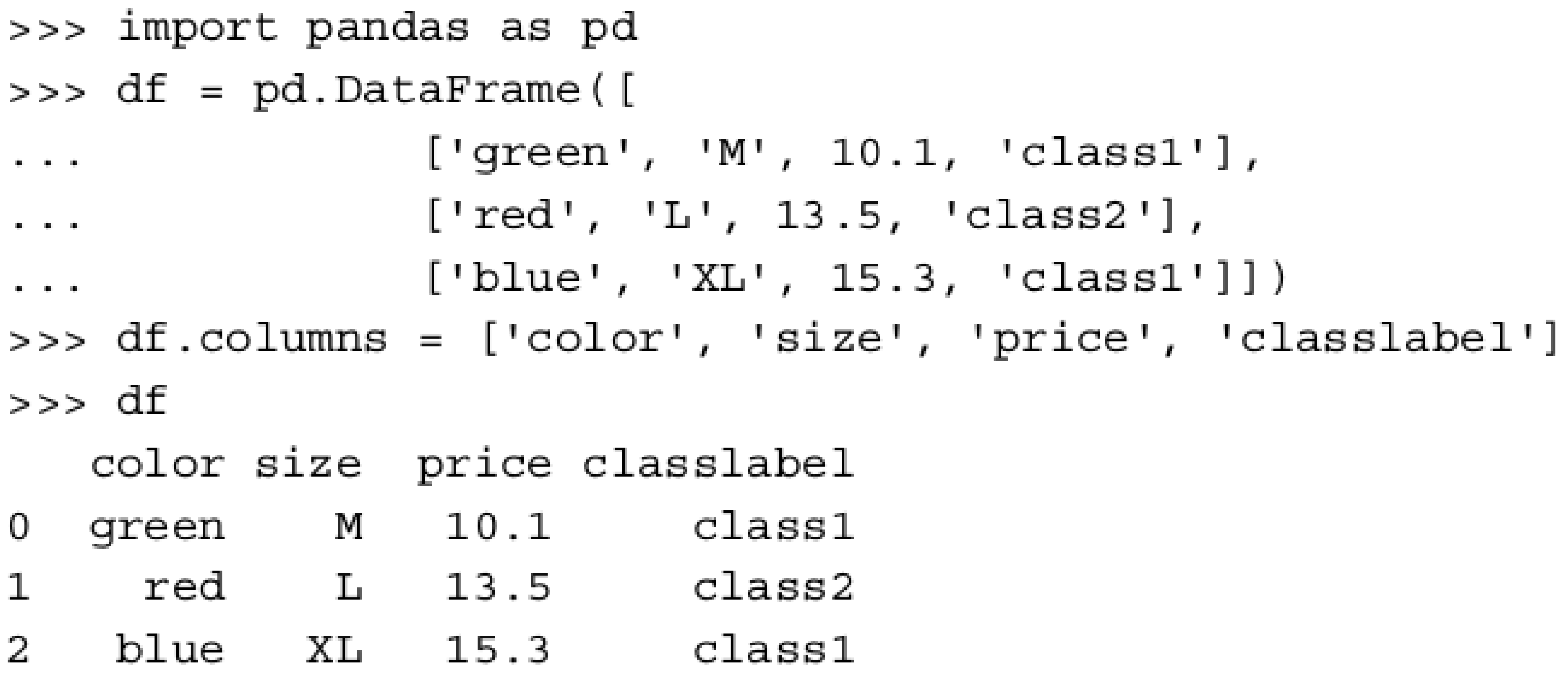
```
# Taking care of missing data
from sklearn.preprocessing import Imputer
imputer = Imputer(missing_values = 'NaN', strategy = 'mean', axis = 0)
imputer = imputer.fit(X[:, 1:3])
X[:, 1:3] = imputer.transform(X[:, 1:3])
```

Handling categorical data

- Nominal and ordinal features
- Mapping ordinal features
- Encoding class labels
- Performing one-hot encoding on nominal features

Nominal and ordinal features

Creating an example dataset



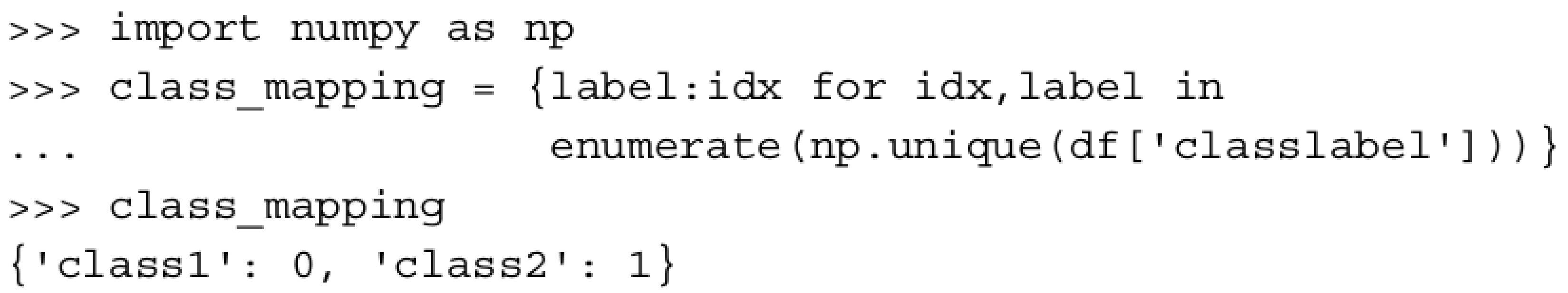
```
>>> import pandas as pd
>>> df = pd.DataFrame([
...             ['green', 'M', 10.1, 'class1'],
...             ['red', 'L', 13.5, 'class2'],
...             ['blue', 'XL', 15.3, 'class1']])
>>> df.columns = ['color', 'size', 'price', 'classlabel']
>>> df
   color size  price classlabel
0  green    M   10.1     class1
1    red    L   13.5     class2
2   blue   XL   15.3     class1
```

Mapping ordinal features

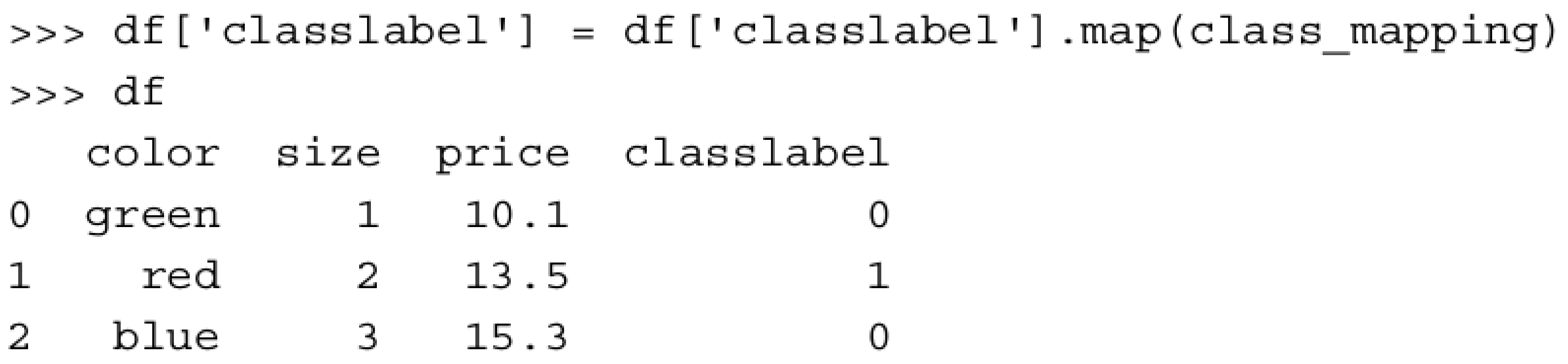
```
>>> size_mapping = {
...                 'XL': 3,
...                 'L': 2,
...                 'M': 1}
>>> df['size'] = df['size'].map(size_mapping)
>>> df
   color  size  price classlabel
0  green     1   10.1     class1
1    red     2   13.5     class2
2   blue     3   15.3     class1
```

```
>>> inv_size_mapping = {v: k for k, v in size_mapping.items()}
>>> df['size'].map(inv_size_mapping)
0      M
1      L
2     XL
Name: size, dtype: object
```

Encoding class labels

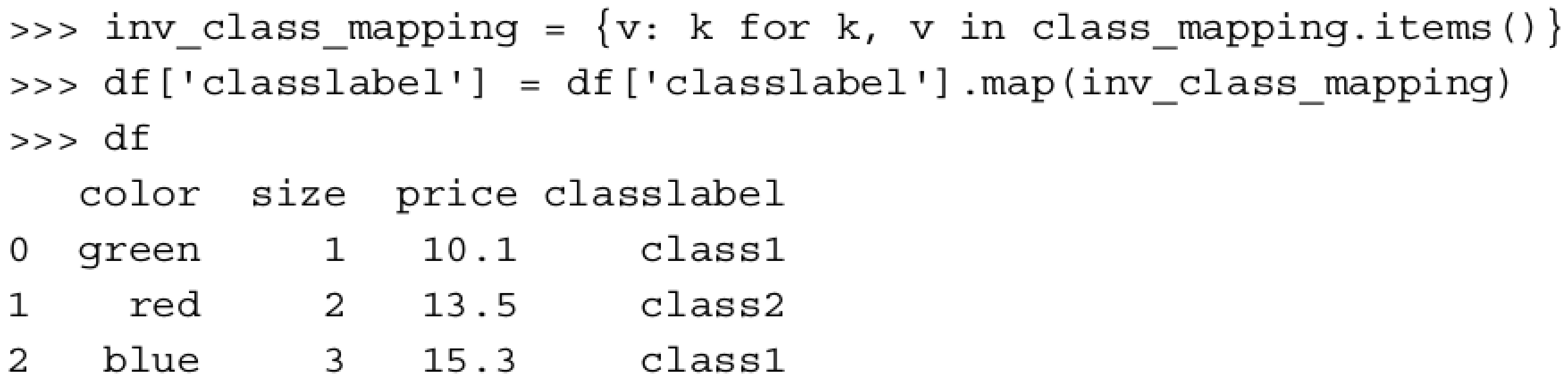


```
>>> import numpy as np
>>> class_mapping = {label:idx for idx,label in
...                   enumerate(np.unique(df['classlabel']))}
>>> class_mapping
{'class1': 0, 'class2': 1}
```



```
>>> df['classlabel'] = df['classlabel'].map(class_mapping)
>>> df
   color  size  price  classlabel
0  green     1   10.1           0
1    red     2   13.5           1
2   blue     3   15.3           0
```

Encoding class labels: inverting



```
>>> inv_class_mapping = {v: k for k, v in class_mapping.items()}
>>> df['classlabel'] = df['classlabel'].map(inv_class_mapping)
>>> df
   color  size  price classlabel
0  green     1   10.1     class1
1    red     2   13.5     class2
2   blue     3   15.3     class1
```

Encoding class labels

```
>>> from sklearn.preprocessing import LabelEncoder
>>> class_le = LabelEncoder()
>>> y = class_le.fit_transform(df['classlabel'].values)
>>> y
array([0, 1, 0])
```

```
>>> class_le.inverse_transform(y)
array(['class1', 'class2', 'class1'], dtype=object)
```

Encoding class labels

# Encoding categorical data

- # Encoding the Independent Variable
- from sklearn.preprocessing import LabelEncoder, OneHotEncoder
- labelencoder_X = LabelEncoder()
- X[:, 0] = labelencoder_X.fit_transform(X[:, 0])
- onehotencoder = OneHotEncoder(categorical_features = [0])
- X = onehotencoder.fit_transform(X).toarray()
- # Encoding the Dependent Variable
- labelencoder_y = LabelEncoder()
- y = labelencoder_y.fit_transform(y)

Performing one-hot encoding on nominal features

```
>>> X = df[['color', 'size', 'price']].values
>>> color_le = LabelEncoder()
>>> X[:, 0] = color_le.fit_transform(X[:, 0])
>>> X
array([[1, 1, 10.1],
       [2, 2, 13.5],
       [0, 3, 15.3]], dtype=object)
```

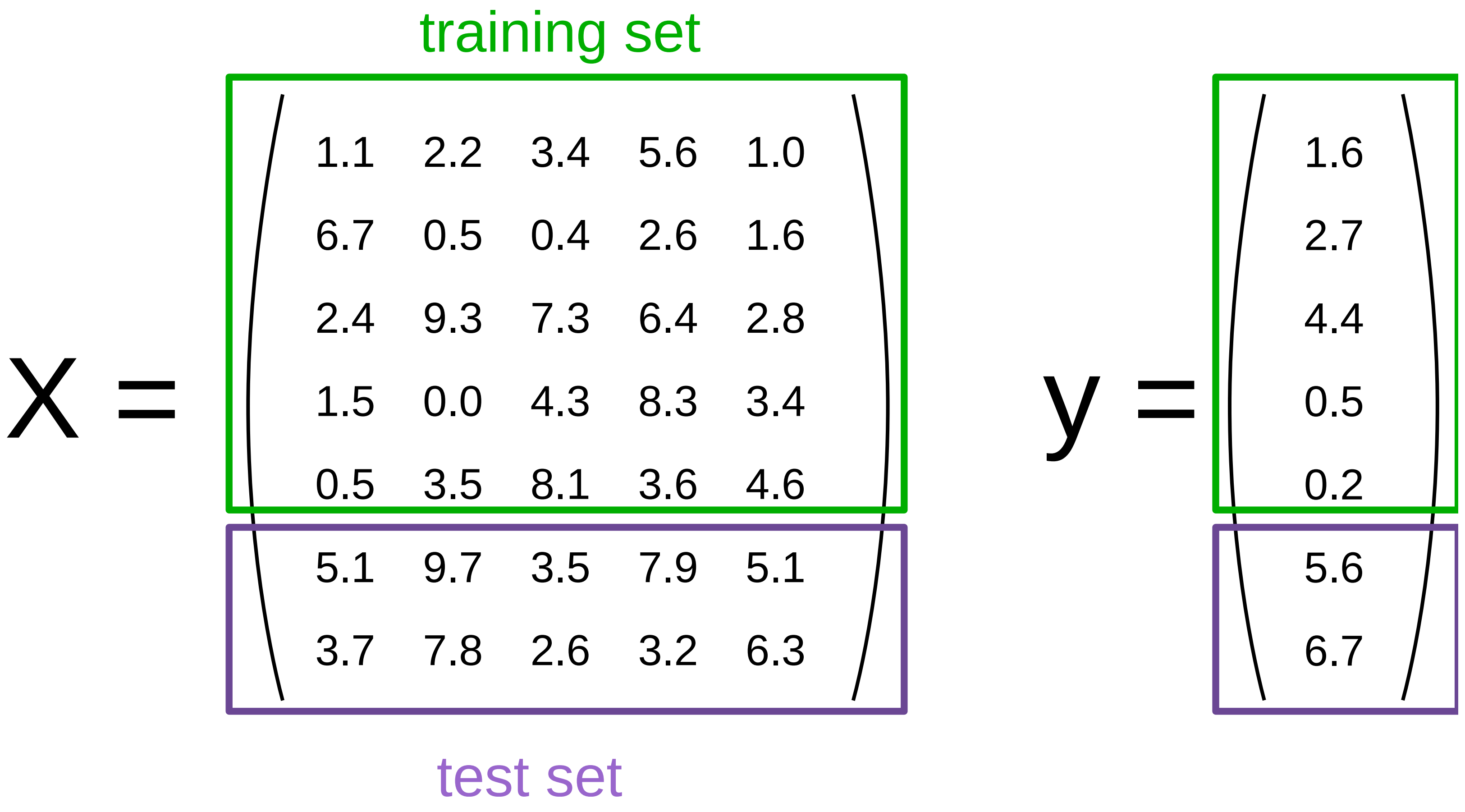
Encoding class labels: OneHotEncoder

```
>>> from sklearn.preprocessing import OneHotEncoder

>>> ohe = OneHotEncoder(categorical_features=[0])
>>> ohe.fit_transform(X).toarray()
```

```
  array([[  0. ,   1. ,   0. ,   1. ,  10.1],
         [  0. ,   0. ,   1. ,   2. ,  13.5],
         [  1. ,   0. ,   0. ,   3. ,  15.3]])
```

Partitioning a dataset into separate training and test sets

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```



$$X = \begin{pmatrix} 1.1 & 2.2 & 3.4 & 5.6 & 1.0 \\ 6.7 & 0.5 & 0.4 & 2.6 & 1.6 \\ 2.4 & 9.3 & 7.3 & 6.4 & 2.8 \\ 1.5 & 0.0 & 4.3 & 8.3 & 3.4 \\ 0.5 & 3.5 & 8.1 & 3.6 & 4.6 \\ 5.1 & 9.7 & 3.5 & 7.9 & 5.1 \\ 3.7 & 7.8 & 2.6 & 3.2 & 6.3 \end{pmatrix} \qquad y = \begin{pmatrix} 1.6 \\ 2.7 \\ 4.4 \\ 0.5 \\ 0.2 \\ 5.6 \\ 6.7 \end{pmatrix}$$

Partitioning an unbalanced dataset into separate training and test sets

```
>>> from sklearn.model_selection import train_test_split
>>> X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values
>>> X_train, X_test, y_train, y_test =\
...     train_test_split(X, y,
...                      test_size=0.3,
...                      random_state=0,
...                      stratify=y)
```

Bringing features onto the same scale: StanderScaler, RobustScaler, MinMaxScaler

```
>>> from sklearn.preprocessing import MinMaxScaler
>>> mms = MinMaxScaler()
>>> X_train_norm = mms.fit_transform(X_train)
>>> X_test_norm = mms.transform(X_test)
```

```
>>> from sklearn.preprocessing import StandardScaler
>>> stdsc = StandardScaler()
>>> X_train_std = stdsc.fit_transform(X_train)
>>> X_test_std = stdsc.transform(X_test)
```

Bringing features onto the same scale for regression: don’t forget the target variable!

# Feature Scaling

- """from sklearn.preprocessing import StandardScaler
- sc_X = StandardScaler()
- X_train = sc_X.fit_transform(X_train)
- X_test = sc_X.transform(X_test)
- sc_y = StandardScaler()
- y_train = sc_y.fit_transform(y_train)"""

Selecting meaningful features

- L1 and L2 regularization as penalties against model complexity
- Sequential feature selection algorithms
- Assessing feature importance with random forests

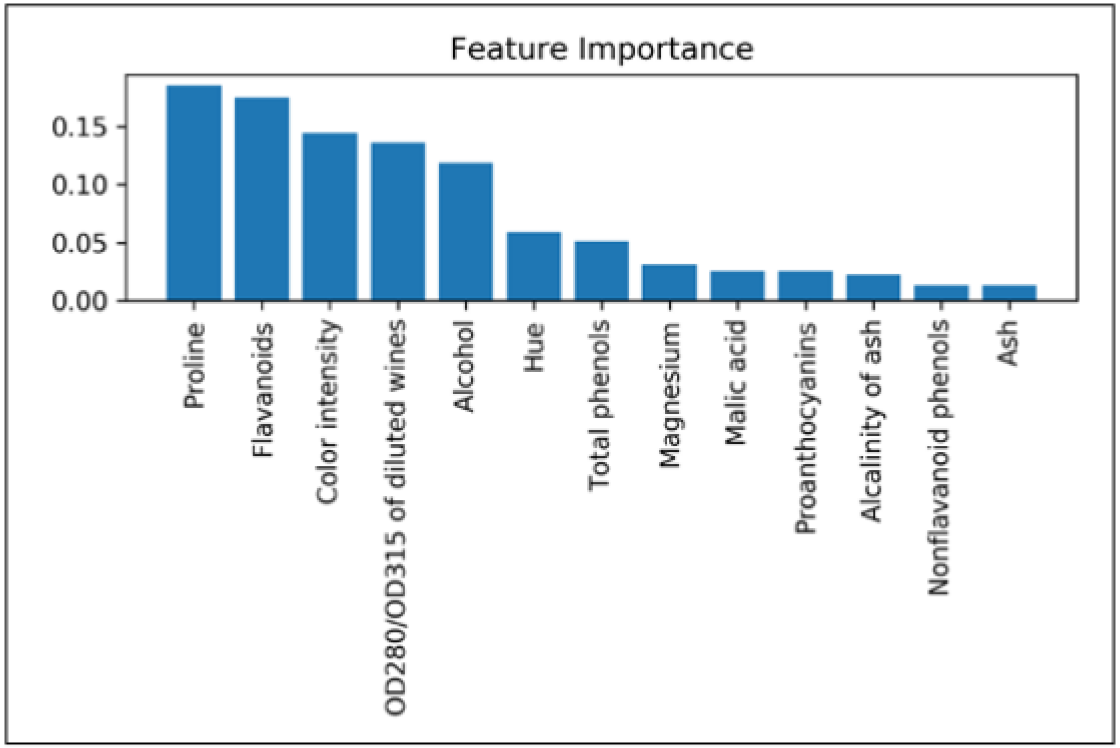
Assessing feature importance with random forests

```
>>> from sklearn.ensemble import RandomForestClassifier

>>> feat_labels = df_wine.columns[1:]

>>> forest = RandomForestClassifier(n_estimators=500,
...                                 random_state=1)
>>> forest.fit(X_train, y_train)
>>> importances = forest.feature_importances_

>>> indices = np.argsort(importances)[::-1]

>>> for f in range(X_train.shape[1]):
...     print("%2d) %-*s %f" % (f + 1, 30,
...                             feat_labels[indices[f]],
...                             importances[indices[f]]))
>>> plt.title('Feature Importance')
>>> plt.bar(range(X_train.shape[1]),
...         importances[indices],
...         align='center')

>>> plt.xticks(range(X_train.shape[1]),
...            feat_labels, rotation=90)
>>> plt.xlim([-1, X_train.shape[1]])
```

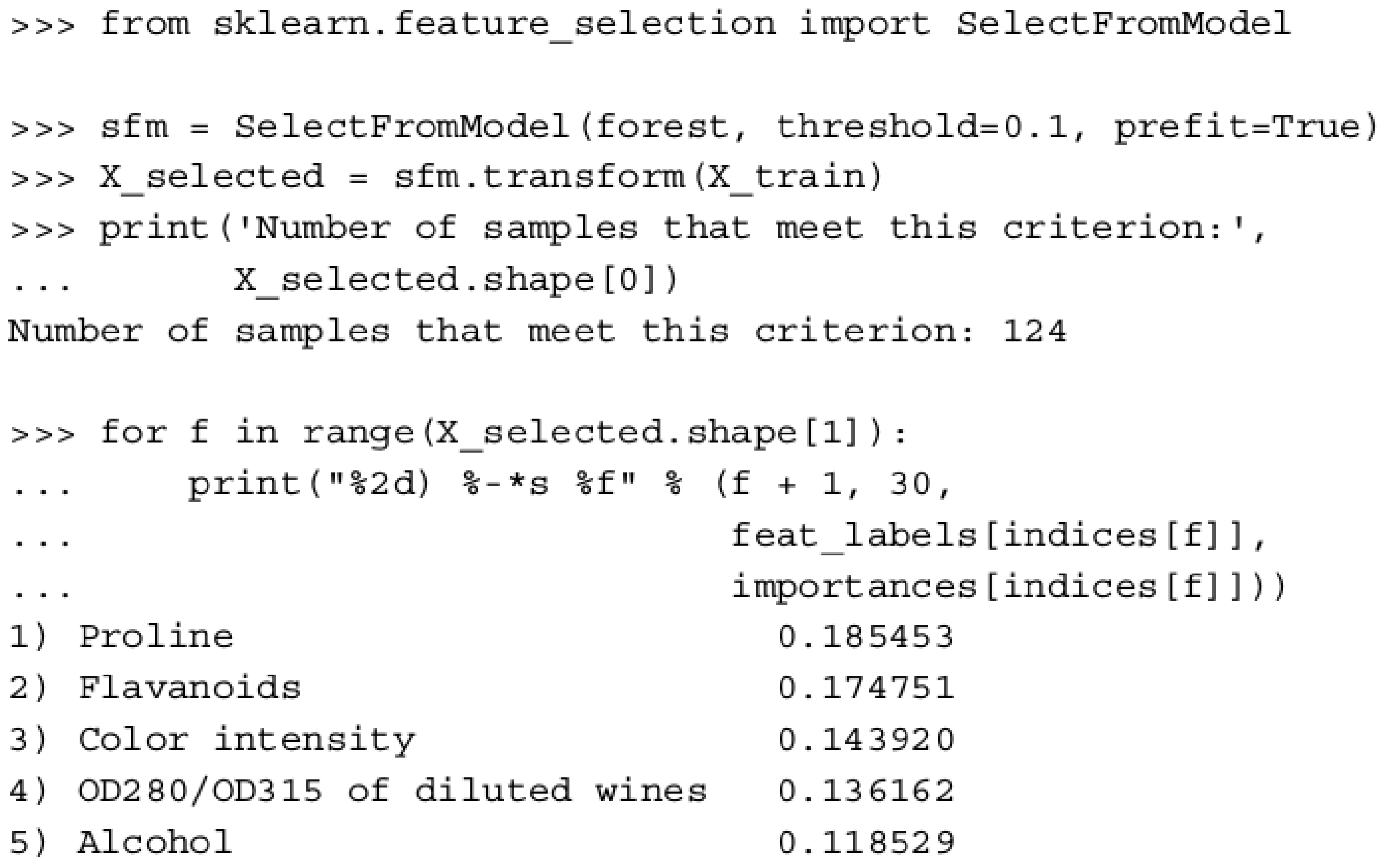
Assessing feature importance with random forests

```
>>> plt.tight_layout()
>>> plt.show()

 1) Proline                          0.185453
 2) Flavanoids                       0.174751
 3) Color intensity                  0.143920
 4) OD280/OD315 of diluted wines     0.136162
 5) Alcohol                          0.118529
 6) Hue                              0.058739
 7) Total phenols                    0.050872
 8) Magnesium                        0.031357
 9) Malic acid                       0.025648
10) Proanthocyanins                  0.025570
11) Alcalinity of ash                0.022366
12) Nonflavanoid phenols             0.013354
13) Ash                              0.013279
```

Assessing feature importance with random forests



Assessing feature importance with random forests



```
>>> from sklearn.feature_selection import SelectFromModel

>>> sfm = SelectFromModel(forest, threshold=0.1, prefit=True)
>>> X_selected = sfm.transform(X_train)
>>> print('Number of samples that meet this criterion:',
...       X_selected.shape[0])
Number of samples that meet this criterion: 124

>>> for f in range(X_selected.shape[1]):
...     print("%2d) %-*s %f" % (f + 1, 30,
...                             feat_labels[indices[f]],
...                             importances[indices[f]]))
1) Proline                        0.185453
2) Flavanoids                     0.174751
3) Color intensity                0.143920
4) OD280/OD315 of diluted wines   0.136162
5) Alcohol                        0.118529
```

Compressing Data via Dimensionality Reduction

Compressing Data via Dimensionality Reduction

- Principal Component Analysis (PCA) for unsupervised data compression
- Linear Discriminant Analysis (LDA) as a supervised dimensionality reduction technique for maximizing class separability
- Nonlinear dimensionality reduction via Kernel Principal Component Analysis (KPCA)