Lecture 05

Data Preprocessing and Machine Learning with Scikit-Learn

(Computational Foundations Part 3/3)

STAT 479: Machine Learning, Fall 2019

Sebastian Raschka

http://stat.wisc.edu/~sraschka/teaching/stat479-fs2019/

Where We Currently Are ...

Part I: Introduction

- Lecture 1: What is Machine Learning? An Overview.
- Lecture 2: Intro to Supervised Learning: KNN

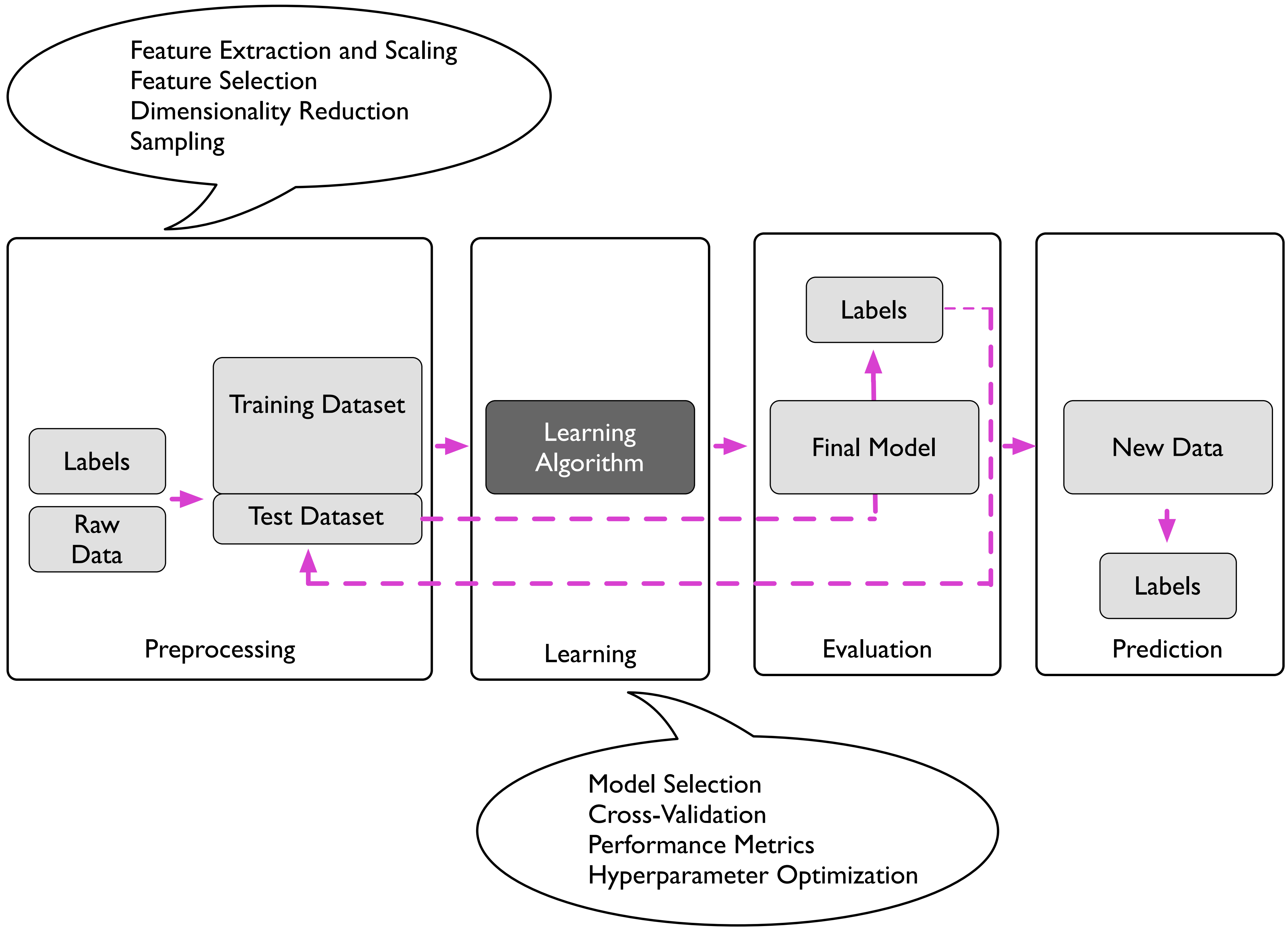
Part II: Computational Foundations

- Lecture 3: Using Python, Anaconda, IPython, Jupyter Notebooks
- Lecture 4: Scientific Computing with NumPy, SciPy, and Matplotlib
- Lecture 5: Data Preprocessing and Machine Learning with Scikit-Learn

Part III: Tree-Based Methods

- Lecture 6: Decision Trees
- Lecture 7: Ensemble Methods

Machine Learning Workflow



Reading a Dataset from a Tabular Text File

The Iris Dataset

Iris-Setosa

Iris-Versicolor

Iris-Virginica

Fisher, R.A. "The use of multiple measurements in taxonomic problems" Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to Mathematical Statistics" (John Wiley, NY, 1950).

Sometimes Useful: Executing "Bash" Terminal Commands Via "!"

```
!head iris.csv
```

```
Id,SepalLength[cm],SepalWidth[cm],PetalLength[cm],PetalWidth[cm],Species
1,5.1,3.5,1.4,0.2,Iris-setosa
2,4.9,3.0,1.4,0.2,Iris-setosa
3,4.7,3.2,1.3,0.2,Iris-setosa
4,4.6,3.1,1.5,0.2,Iris-setosa
5,5.0,3.6,1.4,0.2,Iris-setosa
6,5.4,3.9,1.7,0.4,Iris-setosa
7,4.6,3.4,1.4,0.3,Iris-setosa
8,5.0,3.4,1.5,0.2,Iris-setosa
9,4.4,2.9,1.4,0.2,Iris-setosa
```

A DataFrame Library for Data Wrangling

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

https://pandas.pydata.org

(PANel DAta S)

McKinney, Wes. "Data structures for statistical computing in python." *Proceedings of the 9th Python in Science Conference*. Vol. 445. 2010.

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

https://pandas.pydata.org

```python
import pandas as pd


df = pd.read_csv('iris.csv')
df.head()
```

	Id	SepalLength[cm]	SepalWidth[cm]	PetalLength[cm]	PetalWidth[cm]	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```python
df.shape
```

```
(150, 6)
```

Basic Data Handling

Python Function

```python
def some_func(x):
    return 'Hello World ' + str(x)

some_func(123)
```

```
'Hello World 123'
```

Regular Function vs Lambda Function

```
def some_func(x):
    return 'Hello World ' + str(x)

some_func(123)
```

```
'Hello World 123'
```

```
f = lambda x: 'Hello World ' + str(x)
f(123)
```

```
'Hello World 123'
```

Column-based Data Processing via Lambda Functions and ".apply"

```
df['Species'] = df['Species'].apply(lambda x: 0 if x=='Iris-setosa' else x)
df.head()
```

	Id	SepalLength[cm]	SepalWidth[cm]	PetalLength[cm]	PetalWidth[cm]	Species
0	1	5.1	3.5	1.4	0.2	0
1	2	4.9	3.0	1.4	0.2	0
2	3	4.7	3.2	1.3	0.2	0
3	4	4.6	3.1	1.5	0.2	0
4	5	5.0	3.6	1.4	0.2	0

Column-based Data Processing via Dictionaries and ".map"

```
d = {'Iris-setosa': 0,
     'Iris-versicolor': 1,
     'Iris-virginica': 2}

df = pd.read_csv('iris.csv')
df['Species'] = df['Species'].map(d)
df.head()
```

	Id	SepalLength[cm]	SepalWidth[cm]	PetalLength[cm]	PetalWidth[cm]	Species
0	1	5.1	3.5	1.4	0.2	0
1	2	4.9	3.0	1.4	0.2	0
2	3	4.7	3.2	1.3	0.2	0
3	4	4.6	3.1	1.5	0.2	0
4	5	5.0	3.6	1.4	0.2	0

Quick Inspections via "head" and "tail"

```
df.tail()
```

	Id	SepalLength[cm]	SepalWidth[cm]	PetalLength[cm]	PetalWidth[cm]	Species
145	146	6.7	3.0	5.2	2.3	2
146	147	6.3	2.5	5.0	1.9	2
147	148	6.5	3.0	5.2	2.0	2
148	149	6.2	3.4	5.4	2.3	2
149	150	5.9	3.0	5.1	1.8	2

Accessing the Uderlying NumPy Array(s) via the ".values" Attribute

```python
y = df['Species'].values
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

"Creating*" the Label Vector "y" and Design Matrix "X"

```
y = df['Species'].values
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
X = df.iloc[:, 1:5].values
X[:5]
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
```

*** why did I put "Creating" in quotation marks?**

A Library with Additional Data Science- & Machine Learning-related Functions

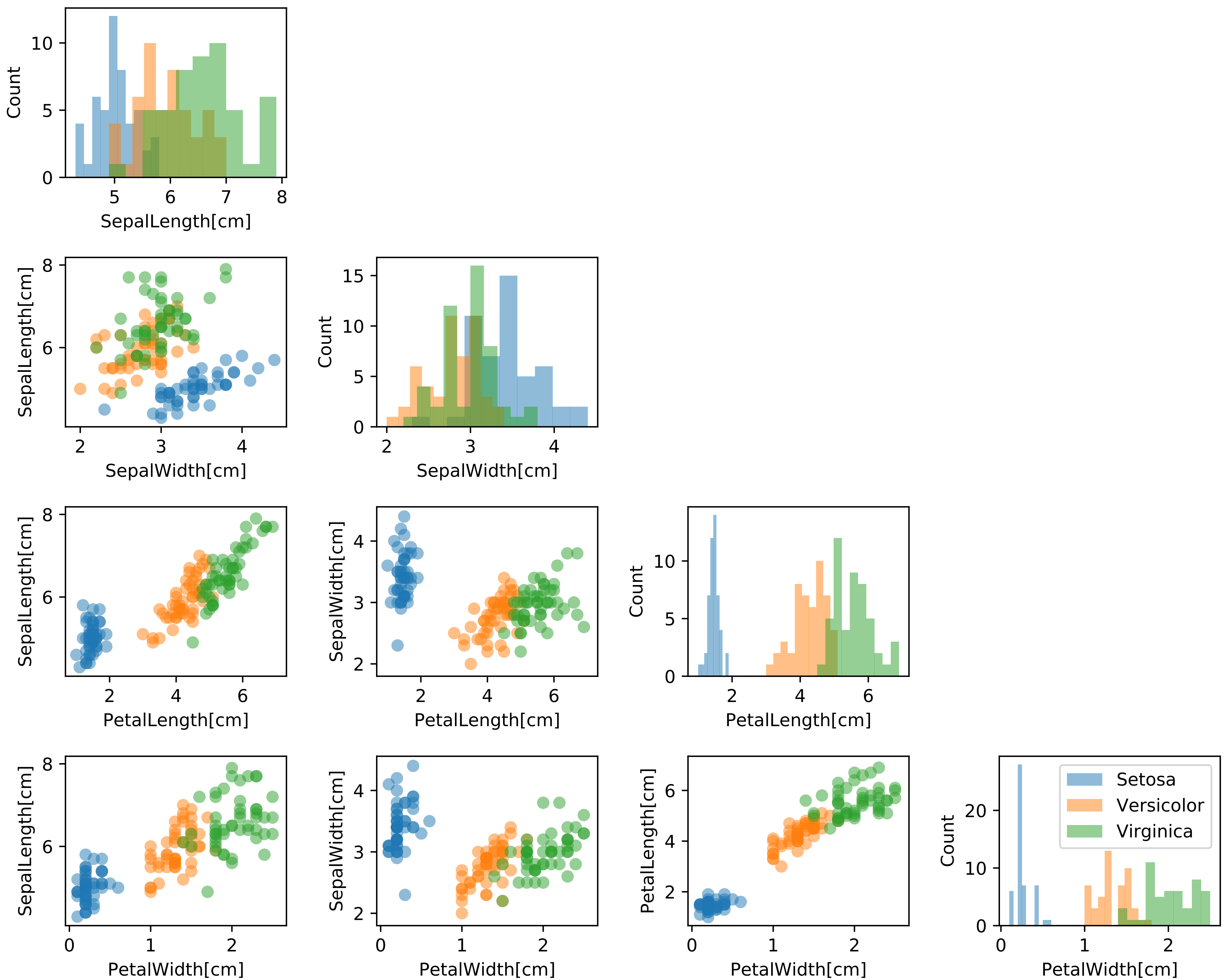
http://rasbt.github.io/mlxtend/

Raschka, Sebastian. "MLxtend: Providing machine learning and data science utilities and extensions to Python's scientific computing stack." *The Journal of Open Source Software* 3.24 (2018).

Exploratory Data Analysis (EDA)

```
#!pip install git+git://github.com/rasbt/mlxtend.git
```

```
%matplotlib inline
import matplotlib.pyplot as plt
from mlxtend.data import iris_data
from mlxtend.plotting import scatterplotmatrix


names = df.columns[1:5]

fig, axes = scatterplotmatrix(X[y==0], figsize=(10, 8), alpha=0.5)
fig, axes = scatterplotmatrix(X[y==1], fig_axes=(fig, axes), alpha=0.5)
fig, axes = scatterplotmatrix(X[y==2], fig_axes=(fig, axes), alpha=0.5, names=names)

plt.tight_layout()
plt.legend(labels=['Setosa', 'Versicolor', 'Virginica'])
plt.show()
```



Count
SepalLength[cm]
SepalWidth[cm]
PetalLength[cm]
PetalWidth[cm]
Setosa
Versicolor
Virginica

Splitting a Dataset into Train, Validation, and Test Subsets

```
import numpy as np

indices = np.arange(X.shape[0])
rng = np.random.RandomState(123)
permuted_indices = rng.permutation(indices)
permuted_indices
```

Splitting a Dataset into Train, Validation, and Test Subsets

```
import numpy as np

indices = np.arange(X.shape[0])
rng = np.random.RandomState(123)
permuted_indices = rng.permutation(indices)
permuted_indices
```

```
array([ 72, 112, 132,  88,  37, 138,  87,  42,   8,  90, 141,  33,  59,
       116, 135, 104,  36,  13,  63,  45,  28, 133,  24, 127,  46,  20,
        31, 121, 117,   4, 130, 119,  29,   0,  62,  93, 131,   5,  16,
        82,  60,  35, 143, 145, 142, 114, 136,  53,  19,  38, 110,  23,
         9,  86,  91,  89,  79, 101,  65, 115,  41, 124,  95,  21,  11,
       103,  74, 122, 118,  44,  51,  81, 149,  12, 129,  56,  50,  25,
       128, 146,  43,   1,  71,  54, 100,  14,   6,  80,  26,  70, 139,
        30, 108,  15,  18,  77,  22,  10,  58, 107,  75,  64,  69,   3,
        40,  76, 134,  34,  27,  94,  85,  97, 102,  52,  92,  99, 105,
         7,  48,  61, 120, 137, 125, 147,  39,  84,   2,  67,  55,  49,
        68, 140,  78, 144, 111,  32,  73,  47, 148, 113,  96,  57, 123,
       106,  83,  17,  98,  66, 126, 109])
```

Splitting a Dataset into Train, Validation, and Test Subsets

```
import numpy as np

indices = np.arange(X.shape[0])
rng = np.random.RandomState(123)
permuted_indices = rng.permutation(indices)
permuted_indices
```

```
       106,  83,  17,  98,  66, 126, 109])
```

```
train_size, valid_size = int(0.65*X.shape[0]), int(0.15*X.shape[0])
test_size = X.shape[0] - (train_size + valid_size)
print(train_size, valid_size, test_size)
```

```
97 22 31
```

```
train_ind = permuted_indices[:train_size]
valid_ind = permuted_indices[train_size:(train_size + valid_size)]
test_ind = permuted_indices[(train_size + valid_size):]
```

```
X_train, y_train = X[train_ind], y[train_ind]
X_valid, y_valid = X[valid_ind], y[valid_ind]
X_test, y_test = X[test_ind], y[test_ind]
```

```
(97, 4)
```

(Later, we will see how to do this more conveniently)

Python Classes

To get a better understanding of the scikit-learn API, we need to understand the main concepts behind Object Oriented Programming (OOP) & classes in Python

Python Classes

```python
class VehicleClass():

    def __init__(self, horsepower):
        "This is the 'init' method"
        # this is a class attribute:
        self.horsepower = horsepower

    def horsepower_to_torque(self, rpm):
        "This is a regular method"
        numerator = self.horsepower * 33000
        denominator = 2* np.pi * 5000
        return numerator/denominator

    def tune_motor(self):
        self.horsepower *= 2

    def _private_method(self):
        print('this is private')

    def __very_private_method(self):
        print('this is very private')
```

Python Classes

```python
class VehicleClass():

    def __init__(self, horsepower):
        "This is the 'init' method"
        # this is a class attribute:
        self.horsepower = horsepower

    def horsepower_to_torque(self, rpm):
        "This is a regular method"
        numerator = self.horsepower * 33000
        denominator = 2* np.pi * 5000
        return numerator/denominator

    def tune_motor(self):
        self.horsepower *= 2

    def _private_method(self):
        print('this is private')

    def __very_private_method(self):
        print('this is very private')
```

```python
# instantiate an object:
car1 = VehicleClass(horsepower=123)
print(car1.horsepower)
```

```
123
```

Python Classes

```python
class VehicleClass():

    def __init__(self, horsepower):
        "This is the 'init' method"
        # this is a class attribute:
        self.horsepower = horsepower

    def horsepower_to_torque(self, rpm):
        "This is a regular method"
        numerator = self.horsepower * 33000
        denominator = 2* np.pi * 5000
        return numerator/denominator

    def tune_motor(self):
        self.horsepower *= 2

    def _private_method(self):
        print('this is private')

    def __very_private_method(self):
        print('this is very private')
```

```python
# instantiate an object:
car1 = VehicleClass(horsepower=123)
print(car1.horsepower)
```

```
123
```

```python
car1.horsepower_to_torque(rpm=5000)
```

```
129.20198280200063
```

```python
car1.tune_motor()
car1.horsepower_to_torque(rpm=5000)
```

```
258.40396560400126
```

```python
class VehicleClass():

    def __init__(self, horsepower):
        "This is the 'init' method"
        # this is a class attribute:
        self.horsepower = horsepower

    def horsepower_to_torque(self, rpm):
        "This is a regular method"
        numerator = self.horsepower * 33000
        denominator = 2* np.pi * 5000
        return numerator/denominator

    def tune_motor(self):
        self.horsepower *= 2

    def _private_method(self):
        print('this is private')

    def __very_private_method(self):
        print('this is very private')
```

```python
car1._private_method()
```

```
this is private
```

```python
car1.__very_private_method()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-23-818c47ec0aa2> in <module>()
----> 1 car1.__very_private_method()

AttributeError: 'VehicleClass' object has no attribute '__very_private_method'
```

Python Classes

```
class VehicleClass():

    def __init__(self, horsepower):
        "This is the 'init' method"
        # this is a class attribute:
        self.horsepower = horsepower

    def horsepower_to_torque(self, rpm):
        "This is a regular method"
        numerator = self.horsepower * 33000
        denominator = 2* np.pi * 5000
        return numerator/denominator

    def tune_motor(self):
        self.horsepower *= 2

    def _private_method(self):
        print('this is private')

    def __very_private_method(self):
        print('this is very private')
```

```
car1._private_method()
```

```
this is private
```

```
car1.__very_private_method()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-23-818c47ec0aa2> in <module>()
----> 1 car1.__very_private_method()

AttributeError: 'VehicleClass' object has no attribute '__very_private_method'
```

```
car1._VehicleClass__very_private_method()
```

```
this is very private
```

Python Classes

```
class CarClass(VehicleClass):

    def __init__(self, horsepower):
        super(CarClass, self).__init__(horsepower)
        self.num_wheels = 4

new_car = CarClass(horsepower=123)
print('Number of wheels:', new_car.num_wheels)
print('Horsepower:', new_car.horsepower)
new_car.tune_motor()
print('Horsepower:', new_car.horsepower)
```

```
Number of wheels: 4
Horsepower: 123
Horsepower: 246
```

K-Nearest Neighbors Implementation

```
class KNNClassifier(object):
    def __init__(self, k, dist_fn=None):
        self.k = k
        if dist_fn is None:
            self.dist_fn = self._euclidean_dist

    def _euclidean_dist(self, a, b):
        dist = 0.
        for ele_i, ele_j in zip(a, b):
            dist += ((ele_i - ele_j)**2)
        dist = dist**0.5
        return dist

    def _find_nearest(self, x):
        dist_idx_pairs = []
        for j in range(self.dataset_.shape[0]):
            d = self.dist_fn(x, self.dataset_[j])
            dist_idx_pairs.append((d, j))

        sorted_dist_idx_pairs = sorted(dist_idx_pairs)

        return sorted_dist_idx_pairs

    def fit(self, X, y):
        self.dataset_ = X.copy()
        self.labels_ = y.copy()
        self.possible_labels_ = np.unique(y)

    def predict(self, X):
        predictions = np.zeros(X.shape[0], dtype=int)
        for i in range(X.shape[0]):
            k_nearest = self._find_nearest(X[i])[:self.k]
            indices = [entry[1] for entry in k_nearest]
            k_labels = self.labels_[indices]
            counts = np.bincount(k_labels,
                                 minlength=self.possible_labels_.shape[0])
            pred_label = np.argmax(counts)
            predictions[i] = pred_label
        return predictions
```

K-Nearest Neighbors Implementation

```
class KNNClassifier(object):
    def __init__(self, k, dist_fn=None):
        self.k = k
        if dist_fn is None:
            self.dist_fn = self._euclidean_dist

    def _euclidean_dist(self, a, b):
        dist = 0.
        for ele_i, ele_j in zip(a, b):
            dist += ((ele_i - ele_j)**2)
        dist = dist**0.5
        return dist

    def _find_nearest(self, x):
        dist_idx_pairs = []
        for j in range(self.dataset_.shape[0]):
            d = self.dist_fn(x, self.dataset_[j])
            dist_idx_pairs.append((d, j))

        sorted

        return

    def fit(se
        self.d
        self.l
        self.p

    def predic
        predic
        for i
            k_nearest = self._find_nearest(X[i])[:self.k]
            indices = [entry[1] for entry in k_nearest]
            k_labels = self.labels_[indices]
            counts = np.bincount(k_labels,
                                 minlength=self.possible_labels_.shape[0])
            pred_label = np.argmax(counts)
            predictions[i] = pred_label
        return predictions
```

```
knn_model = KNNClassifier(k=3)
knn_model.fit(X_train, y_train)
```

```
print(knn_model.predict(X_valid))
```

```
[0 1 2 1 1 1 0 0 1 2 0 0 1 1 1 2 1 1 1 2 0 0]
```

The "Main" Machine Learning Library for Python

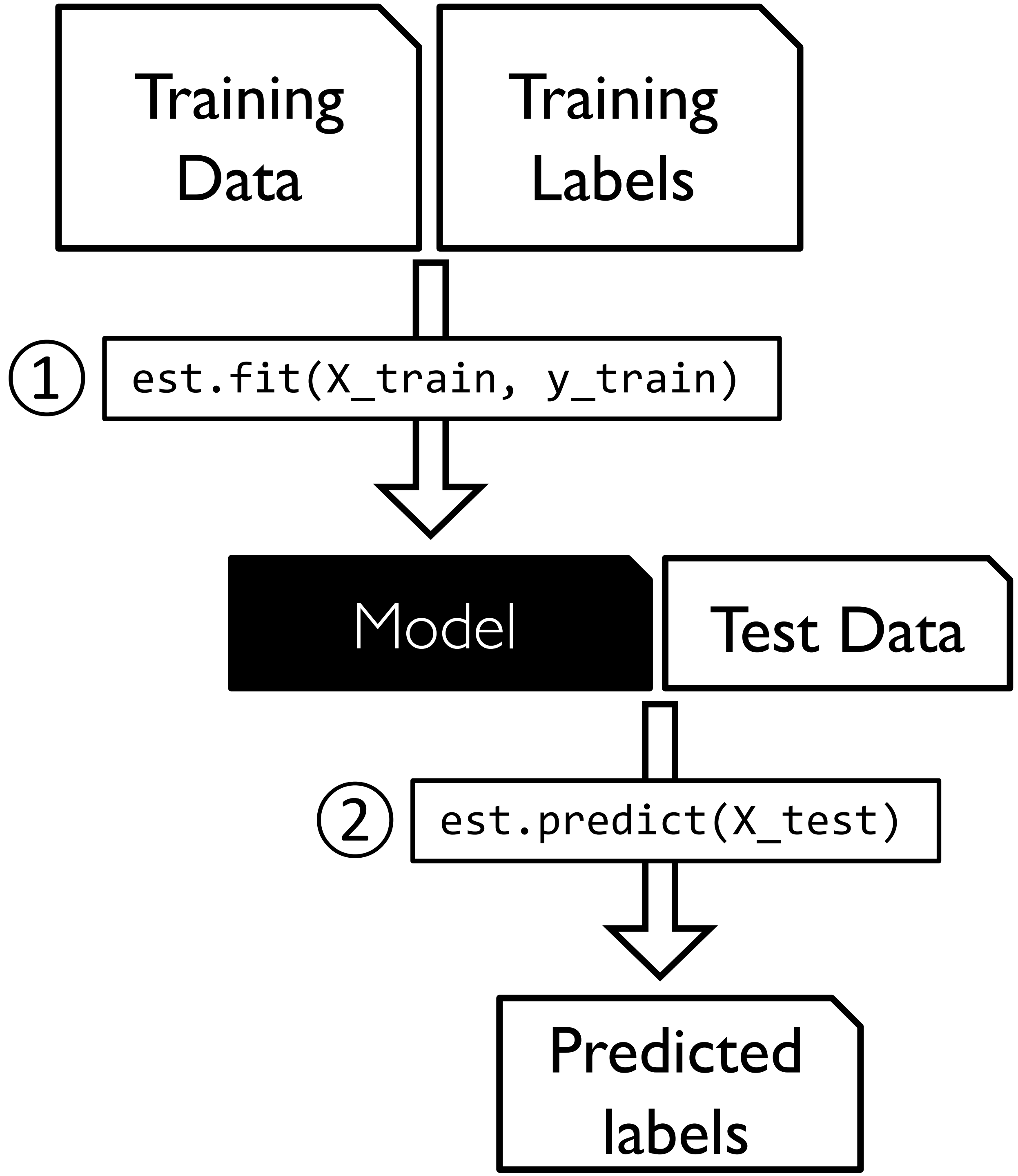
http://scikit-learn.org

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." *Journal of machine learning research* 12.Oct (2011): 2825-2830.

The Scikit-learn Estimator API (an OOP Paradigm)

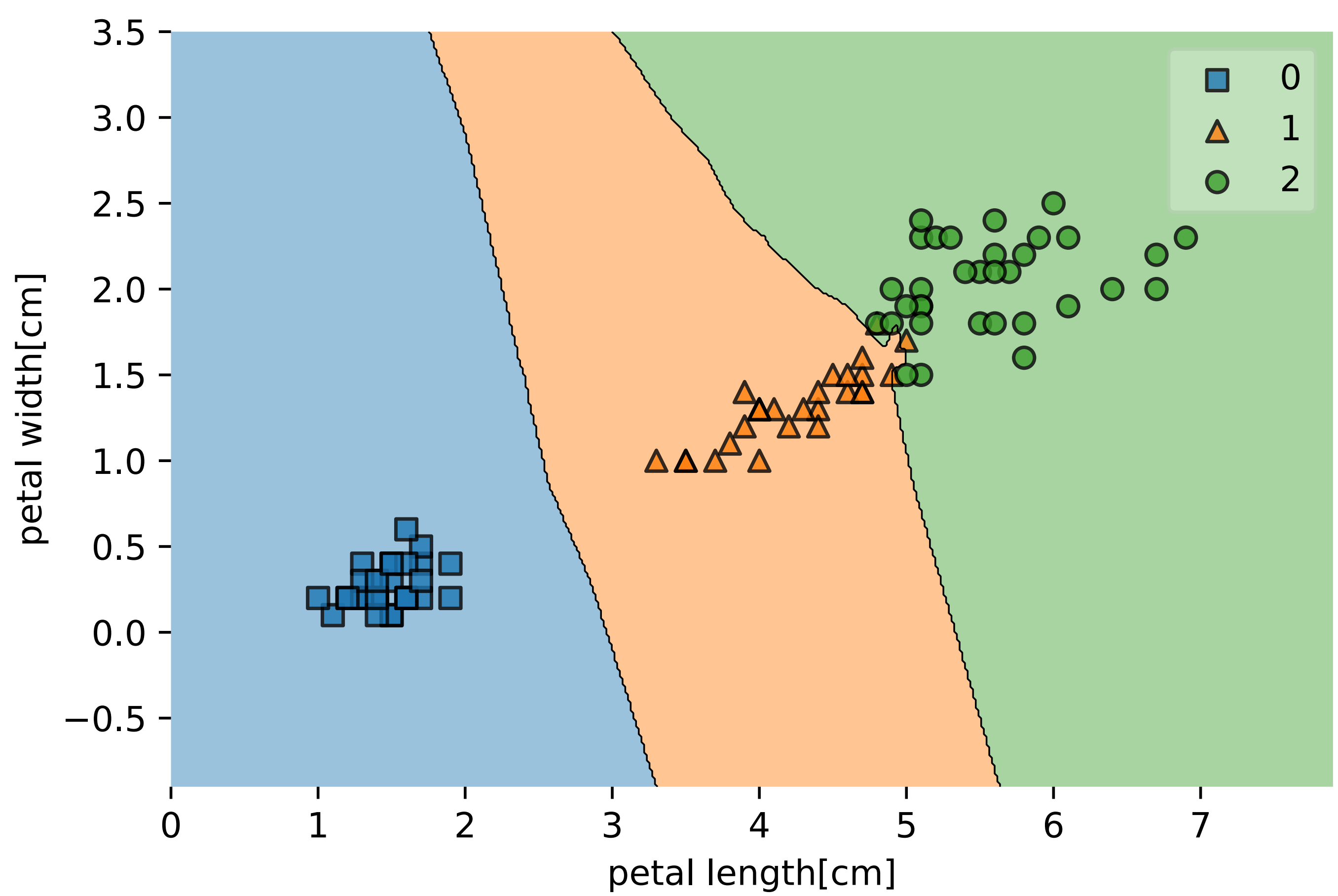
```python
class SupervisedEstimator(...):

    def __init__(self, hyperparam_1, ...):
        self.hyperparm_1
        ...

    def fit(self, X, y):
        ...
        self.fit_attribute_
        return self

    def predict(self, X):
        ...
        return y_pred

    def score(self, X, y):
        ...
        return score

    def _private_method(self):
        ...
    ...
```

The Scikit-learn Estimator API

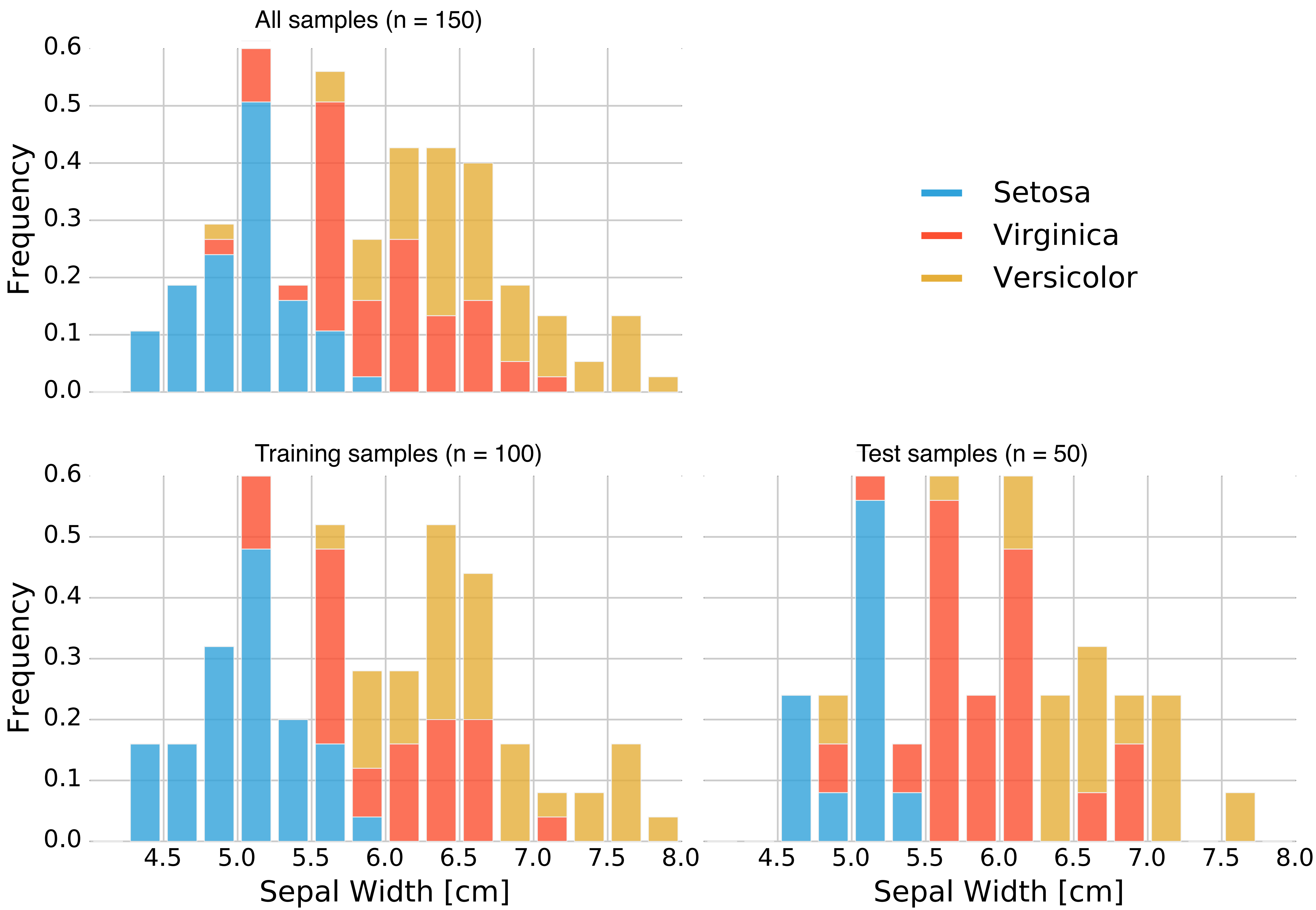


A 3-Nearest Neighbor Classifier & 2 Iris Features

```
from sklearn.neighbors import KNeighborsClassifier
from mlxtend.plotting import plot_decision_regions

knn_model = KNeighborsClassifier(n_neighbors=3)
knn_model.fit(X_train[:, 2:], y_train)
plot_decision_regions(X_train[:, 2:], y_train, knn_model)
plt.xlabel('petal length[cm]')
plt.ylabel('petal width[cm]')
plt.savefig('images/decisionreg.pdf')
plt.show()
```



Issues with Random Subsampling ...



Stratified Splits

```
from sklearn.model_selection import train_test_split

X_temp, X_test, y_temp, y_test = \
        train_test_split(X, y, test_size=0.2,
                         shuffle=True, random_state=123, stratify=y)
np.bincount(y_temp)
```

```
array([40, 40, 40])
```

```
X_train, X_valid, y_train, y_valid = \
        train_test_split(X_temp, y_temp, test_size=0.2,
                         shuffle=True, random_state=123, stratify=y_temp)
X_train.shape
```

```
(96, 4)
```

Normalization: Min-Max Scaling

$$x_{norm}^{[i]} = \frac{x^{[i]} - x_{min}}{x_{max} - x_{min}}$$

Normalization: Min-Max Scaling

$$x_{norm}^{[i]} = \frac{x^{[i]} - x_{min}}{x_{max} - x_{min}}$$

```
x = np.arange(6).astype(float)
x
```

```
array([0., 1., 2., 3., 4., 5.])
```

```
x_norm = (x - x.min()) / (x.max() - x.min())
x_norm
```

```
array([0. , 0.2, 0.4, 0.6, 0.8, 1. ])
```

Normalization: Standardization

$$x_{std}^{[i]} = \frac{x^{[i]} - \mu_x}{\sigma_x}$$

Normalization: Standardization

$$x_{std}^{[i]} = \frac{x^{[i]} - \mu_x}{\sigma_x}$$

```
x = np.arange(6).astype(float)
x
```

```
array([0., 1., 2., 3., 4., 5.])
```

```
x_std = (x - x.mean()) / x.std()
x_std
```

```
array([-1.46385011, -0.87831007, -0.29277002,  0.29277002,  0.87831007,
        1.46385011])
```

Normalization: Standardization

```
df = pd.DataFrame([1, 2, 1, 2, 3, 4])
df[0].std()
```

```
1.1690451944500122
```

```
df[0].values.std()
```

```
1.0671873729054748
```

Sample vs Population Standard Deviation

$$s_x = \sqrt{\frac{1}{n-1} \sum_{n}^{i=1} (x^{[i]} - \bar{x})^2}$$

$$\sigma_x = \sqrt{\frac{1}{n} \sum_{n}^{i=1} (x^{[i]} - \mu_x)^2}$$

Sample vs Population Standard Deviation

```
df = pd.DataFrame([1, 2, 1, 2, 3, 4])
df[0].std()
```

1.1690451944500122

```
df[0].values.std()
```

1.0671873729054748

```
df[0].values.std(ddof=1)
```

1.1690451944500122

$$s_x = \sqrt{\frac{1}{n-1} \sum_{n}^{i=1} (x^{[i]} - \bar{x})^2}$$

$$\sigma_x = \sqrt{\frac{1}{n} \sum_{n}^{i=1} (x^{[i]} - \mu_x)^2}$$

Scaling Validation and Test Sets

```
mu, sigma = X_train.mean(axis=0), X_train.std(axis=0)

X_train_std = (X_train - mu) / sigma
X_valid_std = (X_valid - mu) / sigma
X_test_std = (X_test - mu) / sigma
```

Scaling Validation and Test Sets

Given 3 training examples:

- example1: 10 cm -> class 2
- example2: 20 cm -> class 2
- example3: 30 cm -> class 1

Estimate:

mean: 20 cm
standard deviation: 8.2 cm

Scaling Validation and Test Sets

Given 3 training examples:

- example1: 10 cm -> class 2
- example2: 20 cm -> class 2
- example3: 30 cm -> class 1

Estimate:

mean: 20 cm
standard deviation: 8.2 cm

Standardize:

- example1: -1.21 -> class 2
- example2: 0.00 -> class 2
- example3: 1.21 -> class 1

Scaling Validation and Test Sets

Given 3 training examples:

- example1: 10 cm -> class 2
- example2: 20 cm -> class 2
- example3: 30 cm -> class 1

Estimate:

mean: 20 cm
standard deviation: 8.2 cm

Standardize (z scores):

- example1: -1.21 -> class 2
- example2: 0.00 -> class 2
- example3: 1.21 -> class 1

Assume you have the classification rule:

$$h(z) = \begin{cases} \text{class 2} & \text{if } z \leq 0.6 \\ \text{class 1} & \text{otherwise} \end{cases}$$

Scaling Validation and Test Sets

Given 3 training examples:

- example1: 10 cm -> class 2
- example2: 20 cm -> class 2
- example3: 30 cm -> class 1

Estimate:

mean: 20 cm
standard deviation: 8.2 cm

Standardize (z scores):

- example1: -1.21 -> class 2
- example2: 0.00 -> class 2
- example3: 1.21 -> class 1

$$h(z) = \begin{cases} \text{class } 2 & \text{if } z \leq 0.6 \\ \text{class } 1 & \text{otherwise} \end{cases}$$

Given 3 ***NEW*** *examples:*

- example4: 5 cm -> class ?
- example5: 6 cm -> class ?
- example6: 7 cm -> class ?

Estimate "new" mean and std.:

- *example5: -1.21 -> class 2*
- *example6: 0.00 -> class 2*
- *example7: 1.21 -> class 1*

Scaling Validation and Test Sets

Given 3 training examples:

- example1: 10 cm -> class 2
- example2: 20 cm -> class 2
- example3: 30 cm -> class 1

Estimate:

mean: 20 cm
standard deviation: 8.2 cm

Standardize (z scores):

- example1: -1.21 -> class 2
- example2: 0.00 -> class 2
- example3: 1.21 -> class 1

$$h(z) = \begin{cases} \text{class } 2 & \text{if } z \leq 0.6 \\ \text{class } 1 & \text{otherwise} \end{cases}$$

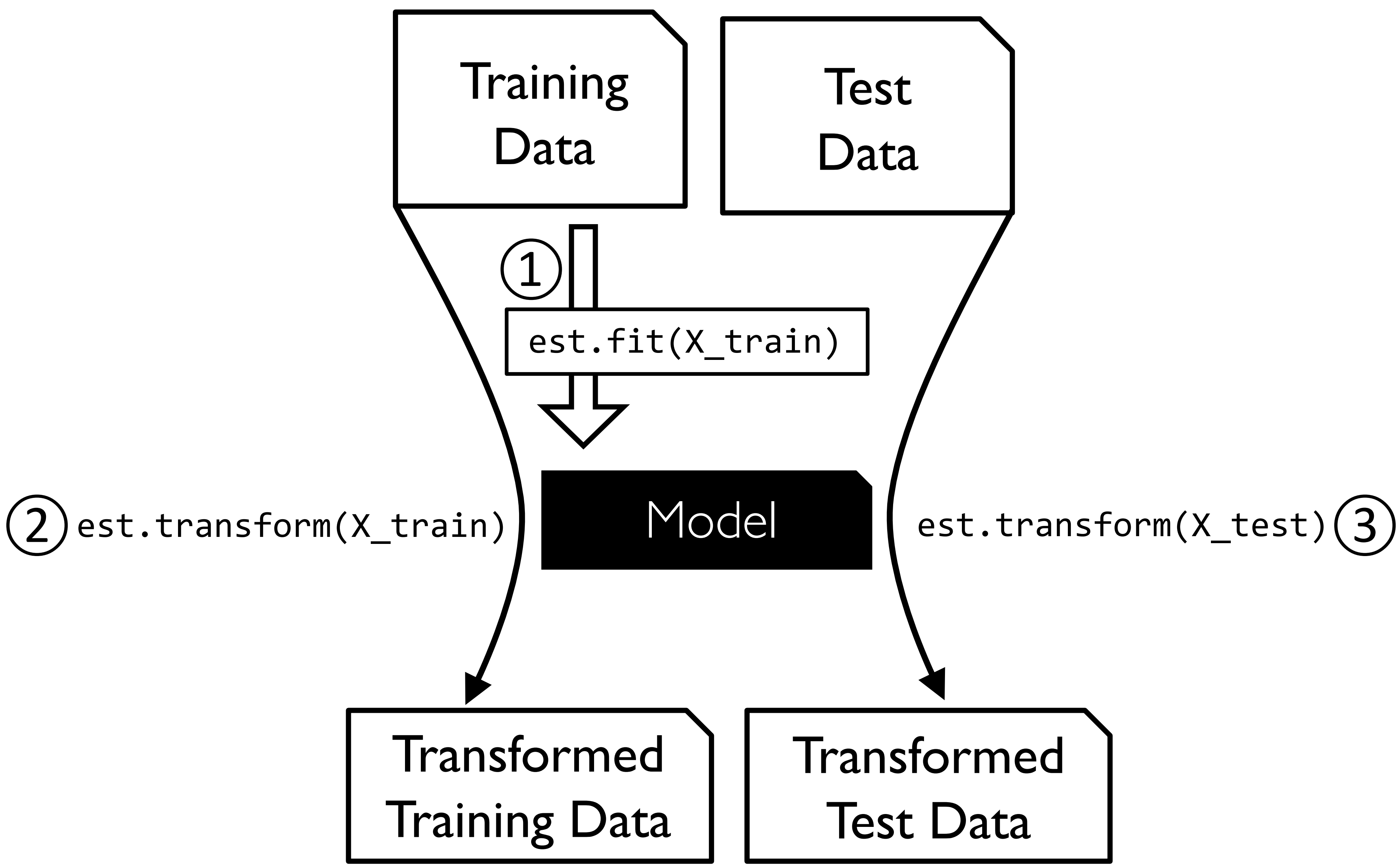
- example4: 5 cm -> class ?
- example5: 6 cm -> class ?
- example6: 7 cm -> class ?

~~*Estimate "new" mean and std.:*~~

- ~~*example5: -1.21 -> class 2*~~
- ~~*example6: 0.00 -> class 2*~~
- ~~*example7: 1.21 -> class 1*~~

- example5: -18.37
- example6: -17.15
- example7: -15.92

The Scikit-Learn Transformer API



The Scikit-Learn Transformer API

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler


scaler = StandardScaler()
scaler.fit(X_train)
X_train_std = scaler.transform(X_train)
X_valid_std = scaler.transform(X_valid)
X_test_std = scaler.transform(X_test)
```

Working with Categorical Data

```
df = pd.read_csv('categoricaldata.csv')
df
```

	color	size	price	classlabel
0	green	M	10.1	class1
1	red	L	13.5	class2
2	blue	XXL	15.3	class1

Categorical Data -> Ordinal Data

```
mapping_dict = {'M': 2,
                'L': 3,
                'XXL': 5}

df['size'] = df['size'].map(mapping_dict)
df
```

	color	size	price	classlabel
0	green	2	10.1	class1
1	red	3	13.5	class2
2	blue	5	15.3	class1

Categorical Data -> Nominal Data (Class Labels)

	color	size	price	classlabel
0	green	2	10.1	class1
1	red	3	13.5	class2
2	blue	5	15.3	class1

```
from sklearn.preprocessing import LabelEncoder


le = LabelEncoder()
df['classlabel'] = le.fit_transform(df['classlabel'])
df
```

	color	size	price	classlabel
0	green	2	10.1	0
1	red	3	13.5	1
2	blue	5	15.3	0

One-hot Encoding for Categorical (Nominal) Features

	color	size	price	classlabel
0	green	2	10.1	0
1	red	3	13.5	1
2	blue	5	15.3	0

```
pd.get_dummies(df)
```

	size	price	classlabel	color_blue	color_green	color_red
0	2	10.1	0	0	1	0
1	3	13.5	1	0	0	1
2	5	15.3	0	1	0	0

One-hot Encoding for Categorical (Nominal) Features

```
pd.get_dummies(df)
```

	size	price	classlabel	color_blue	color_green	color_red
0	2	10.1	0	0	1	0
1	3	13.5	1	0	0	1
2	5	15.3	0	1	0	0

```
pd.get_dummies(df, drop_first=True)
```

	size	price	classlabel	color_green	color_red
0	2	10.1	0	1	0
1	3	13.5	1	0	1
2	5	15.3	0	0	0

Dealing with Missing Data

```
df = pd.read_csv('missingdata.csv')
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

Dealing with Missing Data

```python
df = pd.read_csv('missingdata.csv')
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

```python
# drop rows with missing values:

df.dropna(axis=0)
```

	A	B	C	D
0	1.0	2.0	3.0	4.0

```python
# drop columns with missing values:

df.dropna(axis=1)
```

	A	B
0	1.0	2.0
1	5.0	6.0
2	10.0	11.0

Dealing with Missing Data

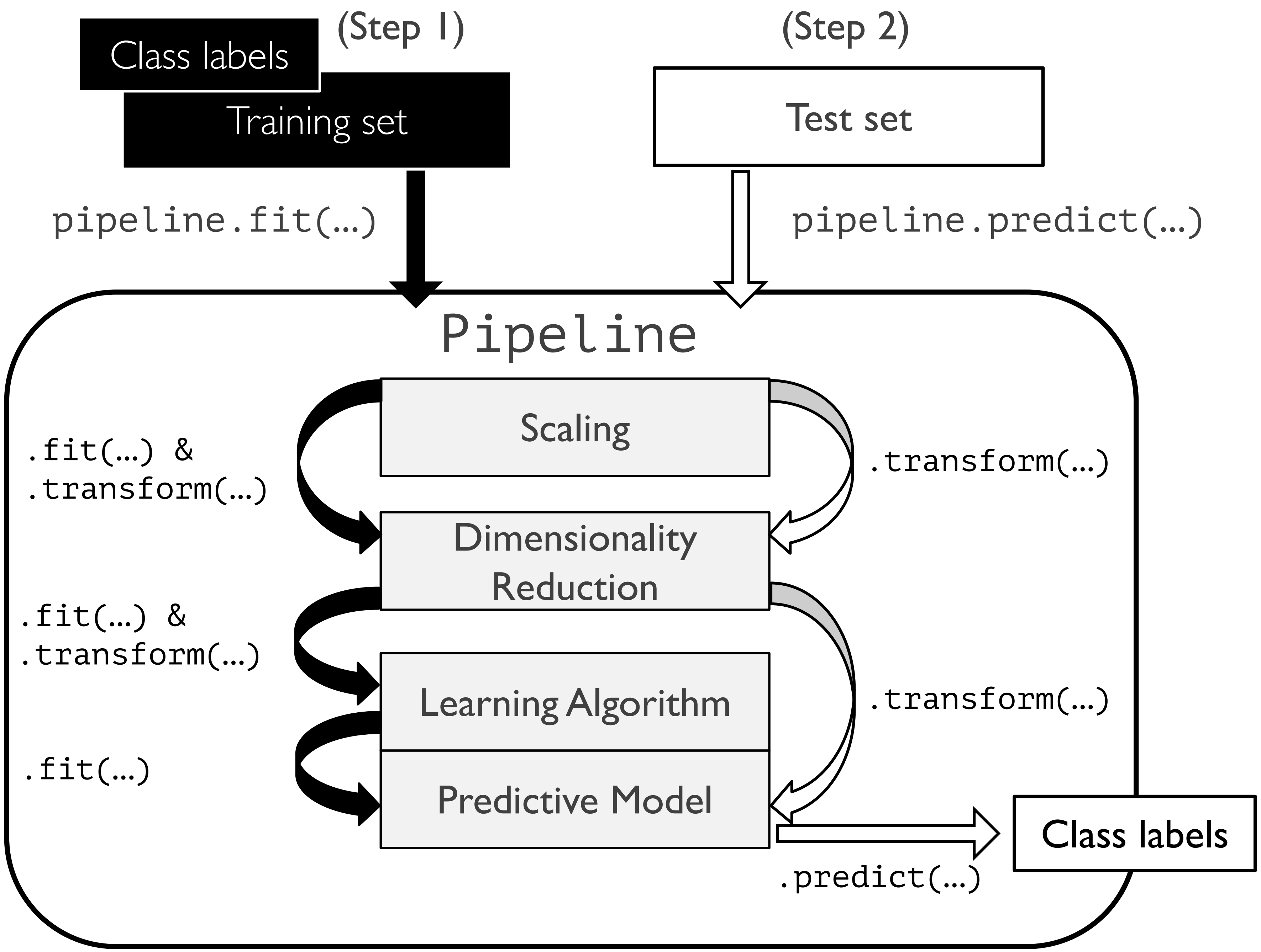
```
df = pd.read_csv('missingdata.csv')
df
```

	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

```
from sklearn.preprocessing import Imputer


imputer = Imputer(missing_values='NaN', strategy='mean', axis=0)
X = df.values
X = imputer.fit_transform(df.values)
X
```

```
array([[ 1. ,  2. ,  3. ,  4. ],
       [ 5. ,  6. ,  7.5,  8. ],
       [10. , 11. , 12. ,  6. ]])
```

Scikit-Learn Pipelines



Scikit-Learn Pipelines

```
from sklearn.pipeline import make_pipeline


pipe = make_pipeline(StandardScaler(),
                     KNeighborsClassifier(n_neighbors=3))
```

```
pipe
```

```
Pipeline(memory=None,
     steps=[('standardscaler', StandardScaler(copy=True, with_mean=Tr
ue, with_std=True)), ('kneighborsclassifier', KNeighborsClassifier(al
gorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=3, p=2,
           weights='uniform'))])
```

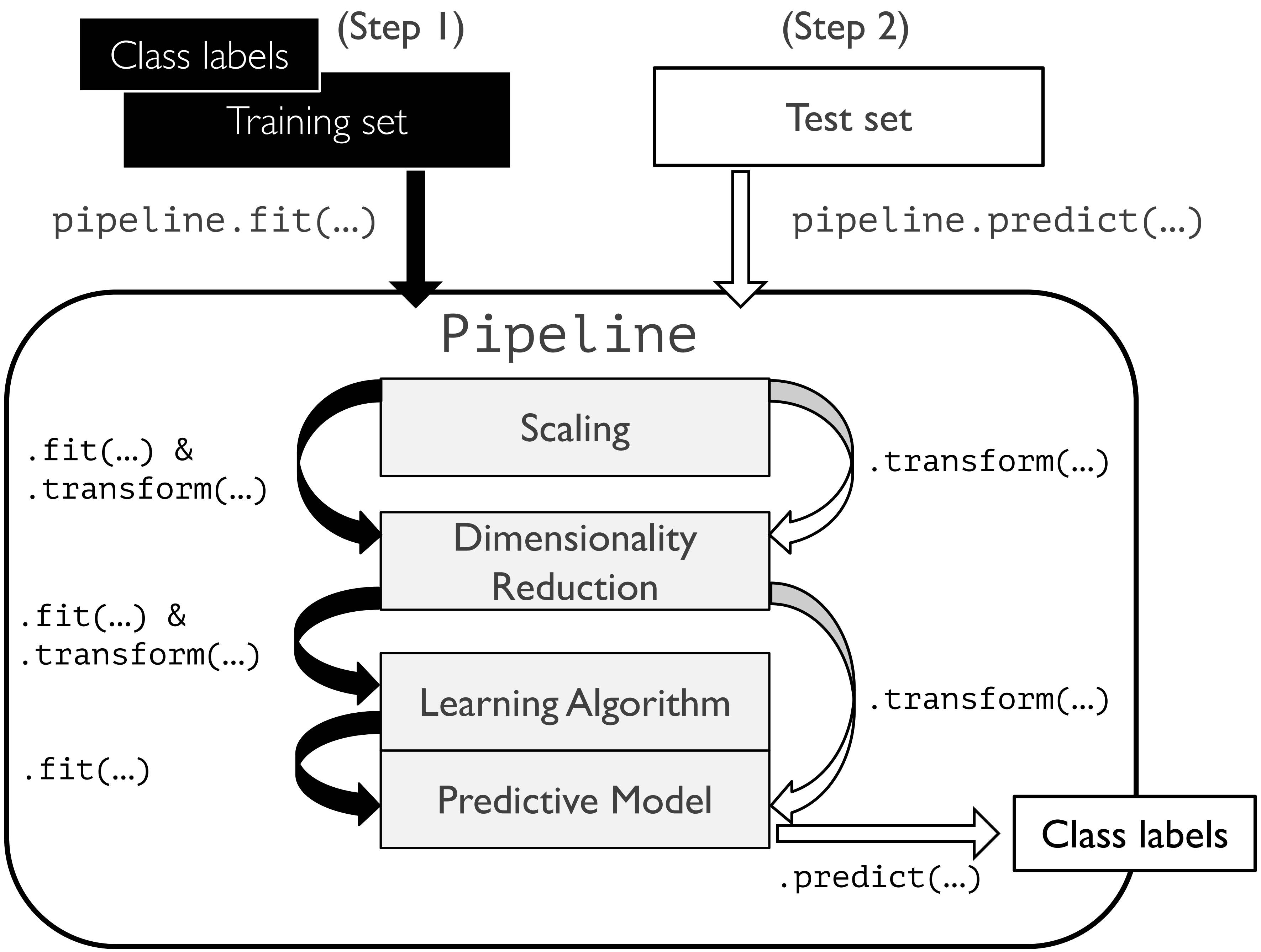
Scikit-Learn Pipelines

```
from sklearn.pipeline import make_pipeline


pipe = make_pipeline(StandardScaler(),
                     KNeighborsClassifier(n_neighbors=3))
```

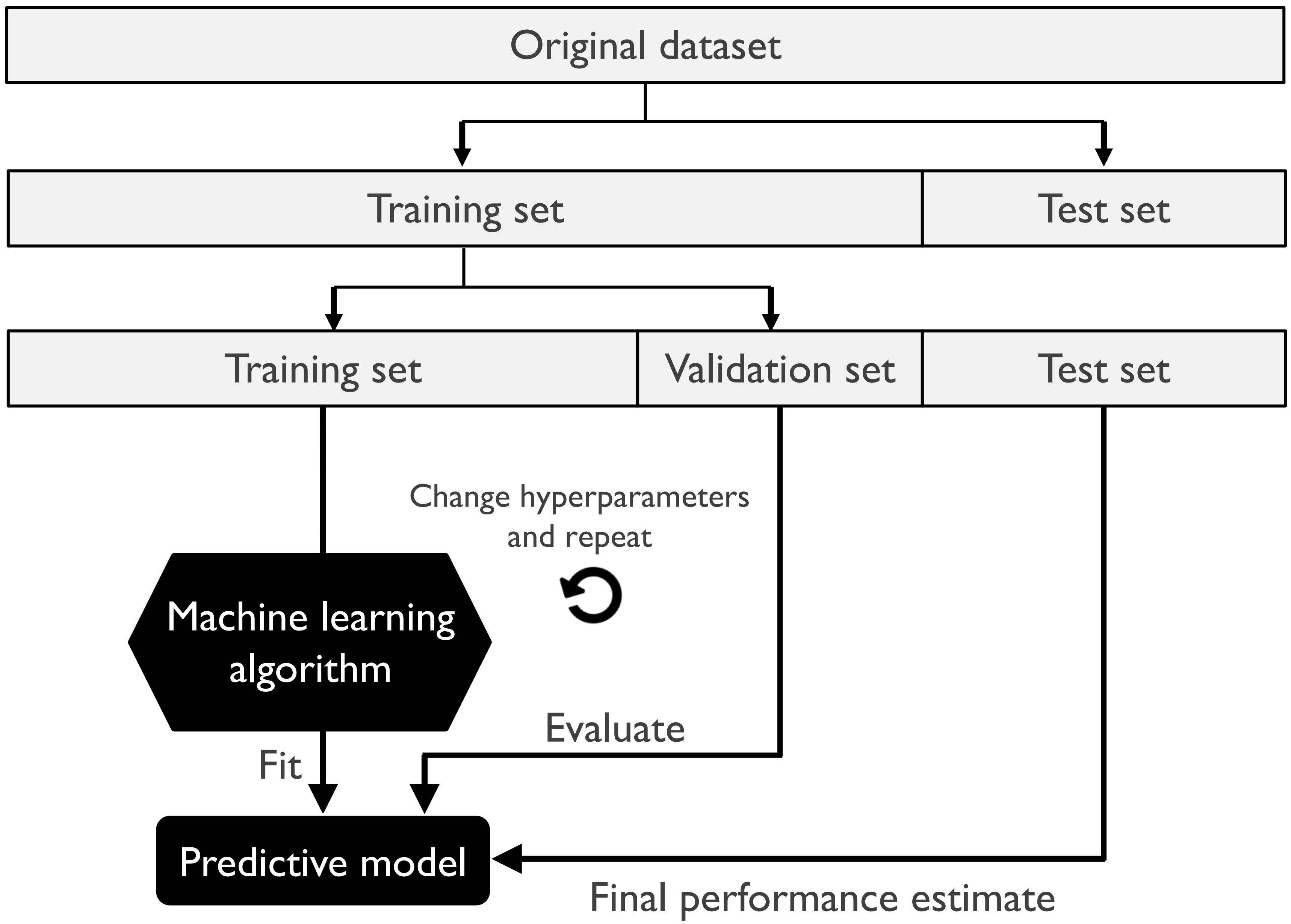
```
pipe.fit(X_train, y_train)
pipe.predict(X_test)
```

```
array([1, 0, 2, 2, 0, 0, 2, 1, 2, 0, 0, 2, 2, 1, 2, 1, 0, 0, 0, 0, 0,
2,
       2, 1, 2, 2, 1, 1, 1, 1])
```

Scikit-Learn Pipelines



Model Selection: Simple Holdout Method



Model Selection: Simple Holdout Method

```
from sklearn.model_selection import GridSearchCV
from mlxtend.evaluate import PredefinedHoldoutSplit
from sklearn.pipeline import make_pipeline
from sklearn.datasets import load_iris

iris = load_iris()
X, y = iris.data, iris.target

train_ind, valid_ind = train_test_split(np.arange(X.shape[0]),
                                        test_size=0.2, shuffle=True,
                                        random_state=123, stratify=y)
```

Model Selection: Simple Holdout Method

```python
from sklearn.model_selection import GridSearchCV
from mlxtend.evaluate import PredefinedHoldoutSplit
from sklearn.pipeline import make_pipeline
from sklearn.datasets import load_iris

iris = load_iris()
X, y = iris.data, iris.target

train_ind, valid_ind = train_test_split(np.arange(X.shape[0]),
                                        test_size=0.2, shuffle=True,
                                        random_state=123, stratify=y)
```

```python
pipe = make_pipeline(StandardScaler(),
                     KNeighborsClassifier())

params = {'kneighborsclassifier__n_neighbors': [1, 3, 5],
          'kneighborsclassifier__p': [1, 2]}

split = PredefinedHoldoutSplit(valid_indices=valid_ind)

grid = GridSearchCV(pipe,
                    param_grid=params,
                    cv=split)
```

Model Selection: Simple Holdout Method

```python
from sklearn.model_selection import GridSearchCV
from mlxtend.evaluate import PredefinedHoldoutSplit
from sklearn.pipeline import make_pipeline
from sklearn.datasets import load_iris

iris = load_iris()
X, y = iris.data, iris.target

train_ind, valid_ind = train_test_split(np.arange(X.shape[0]),
                                        test_size=0.2, shuffle=True,
                                        random_state=123, stratify=y)
```

```python
pipe = make_pipeline(StandardScaler(),
                     KNeighborsClassifier())

params = {'kneighborsclassifier__n_neighbors': [1, 3, 5],
          'kneighborsclassifier__p': [1, 2]}

split = PredefinedHoldoutSplit(valid_indices=valid_ind)

grid = GridSearchCV(pipe,
                    param_grid=params,
                    cv=split)
```

```python
grid.cv_results_
```

```
{'mean_fit_time': array([0.00151896, 0.00076985, 0.00071883, 0.00068808, 0.00069523,
        0.00067973]),
 'std_fit_time': array([0., 0., 0., 0., 0., 0.]),
 'mean_score_time': array([0.00145102, 0.00129414, 0.00130701, 0.00129294, 0.00127792,
        0.0012753 ]),
 'std_score_time': array([0., 0., 0., 0., 0., 0.]),
 'param_kneighborsclassifier__n_neighbors': masked_array(data=[1, 1, 3, 3, 5, 5],
              mask=[False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_kneighborsclassifier__p': masked_array(data=[1, 2, 1, 2, 1, 2],
              mask=[False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'kneighborsclassifier__n_neighbors': 1,
   'kneighborsclassifier__p': 1},
  {'kneighborsclassifier__n_neighbors': 1, 'kneighborsclassifier__p': 2},
  {'kneighborsclassifier__n_neighbors': 3, 'kneighborsclassifier__p': 1},
  {'kneighborsclassifier__n_neighbors': 3, 'kneighborsclassifier__p': 2},
  {'kneighborsclassifier__n_neighbors': 5, 'kneighborsclassifier__p': 1},
  {'kneighborsclassifier__n_neighbors': 5, 'kneighborsclassifier__p': 2}],
 'split0_test_score': array([0.9       , 0.96666667, 0.96666667, 0.93333333, 0.9       ,
        0.9       ]),
 'mean_test_score': array([0.9       , 0.96666667, 0.96666667, 0.93333333, 0.9       ,
        0.9       ]),
 'std_test_score': array([0., 0., 0., 0., 0., 0.]),
 'rank_test_score': array([4, 1, 1, 3, 4, 4], dtype=int32)}
```

Model Selection: Simple Holdout Method

```python
from sklearn.model_selection import GridSearchCV
from mlxtend.evaluate import PredefinedHoldoutSplit
from sklearn.pipeline import make_pipeline
from sklearn.datasets import load_iris

iris = load_iris()
X, y = iris.data, iris.target

train_ind, valid_ind = train_test_split(np.arange(X.shape[0]),
                                        test_size=0.2, shuffle=True,
                                        random_state=123, stratify=y)
```

```python
pipe = make_pipeline(StandardScaler(),
                     KNeighborsClassifier())

params = {'kneighborsclassifier__n_neighbors': [1, 3, 5],
          'kneighborsclassifier__p': [1, 2]}

split = PredefinedHoldoutSplit(valid_indices=valid_ind)

grid = GridSearchCV(pipe,
                    param_grid=params,
                    cv=split)
```

```python
print(grid.best_score_)
print(grid.best_params_)
```

```
0.9666666666666667
{'kneighborsclassifier__n_neighbors': 1, 'kneighborsclassifier__p': 2}
```

```python
clf = grid.best_estimator_
clf.fit(X_train, y_train)
print('Test accuracy: %.2f%%' % (clf.score(X_test, y_test)*100))
```

```
Test accuracy: 100.00%
```

Lecture Notes

This time in interactive Jupyter Notebook form:

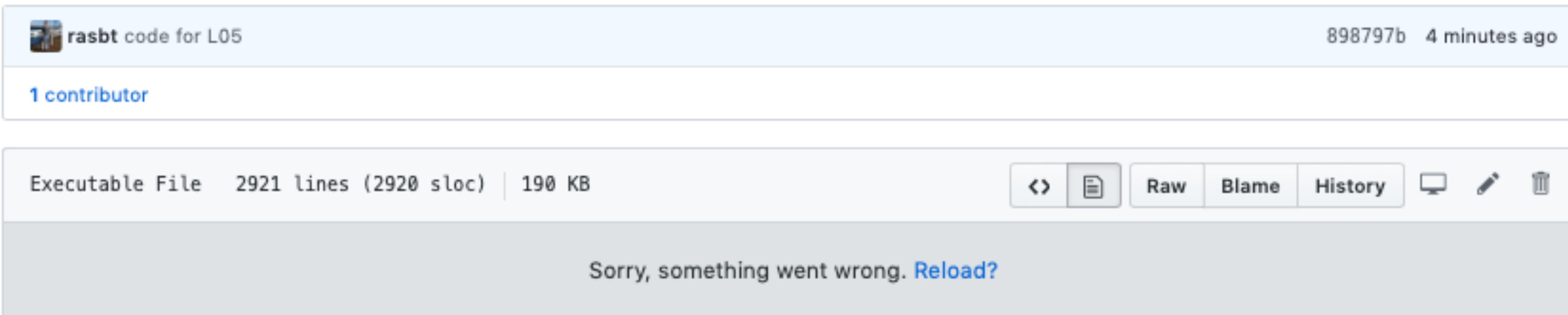
https://github.com/rasbt/stat479-machine-learning-fs19/blob/master/05_preprocessing-and-sklearn/code/05-preprocessing-and-sklearn__notes.ipynb

Bonus: Column Transformers for Heterogenous Data

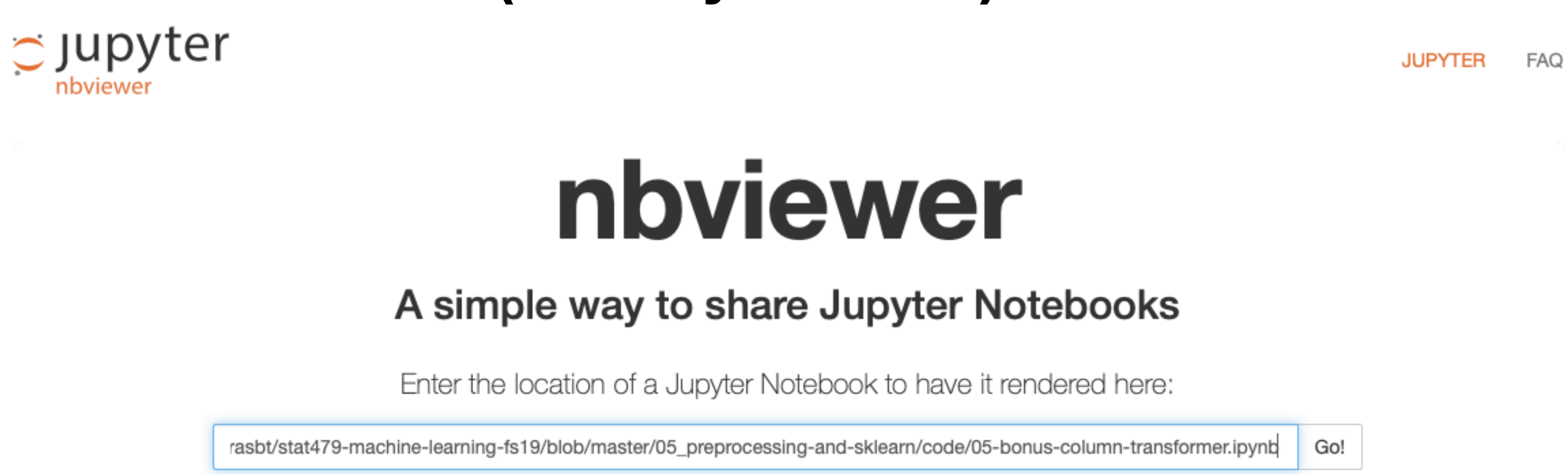
https://github.com/rasbt/stat479-machine-learning-fs19/blob/master/05_preprocessing-and-sklearn/code/05-bonus-column-transformer.ipynb

Tip

If you see this, the Notebook rendering on GitHub is having some hiccups again.



Simply copy and paste the notebook link into the NbViewer available at
https://nbviewer.jupyter.org
(it always works!)



Reading Assignments

- *Python Machine Learning, 2nd ed.*:
  Ch04 up to "Selecting Meaningful Features"
  (pg 107-123)

- *Python Machine Learning, 2nd ed.*:
  Ch06 up to "Debugging Algorithms with Learning and Validation Curves"
  (pg 185-194)