

# Transform Your Workflow with `scikit-learn`

---

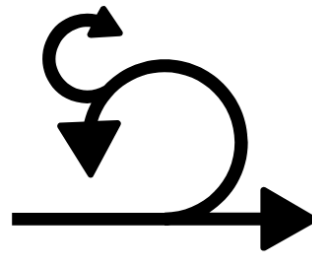
Sam Ballerini

KPMG D&A Super Day 2018

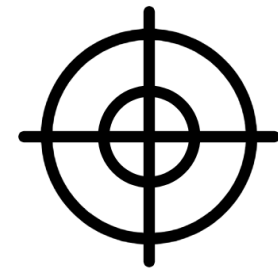




Speed



Agility



Accuracy

# Agenda

1. API Overview
2. Pipelines
3. Cross-validation
4. Customization



# Why `scikit-learn`?

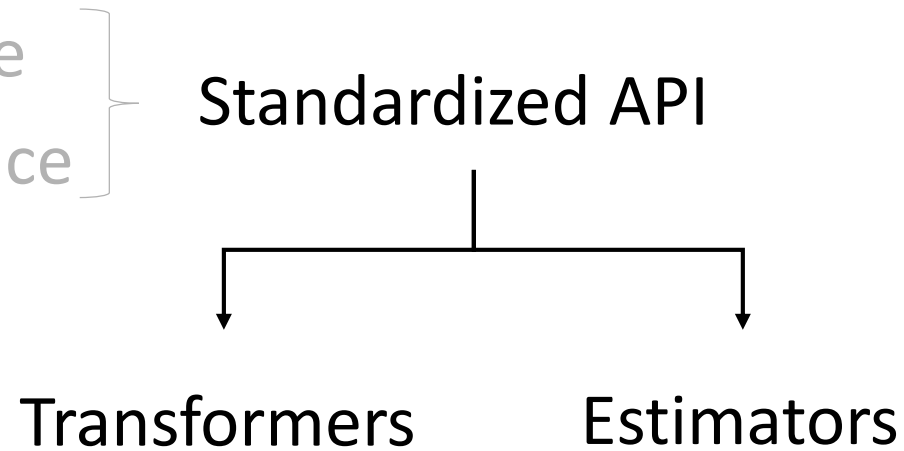
- Well known in the data science community
- Flexible and extensible
- A standardized interface

# Why `scikit-learn`?

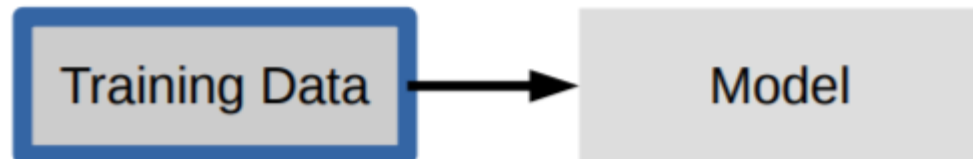
- Well known in the data science community
  - Flexible and extensible
  - A standardized interface
- } Standardized API

# Why `scikit-learn`?

- Well known in the data science community
- Flexible and extensible
- A standardized interface

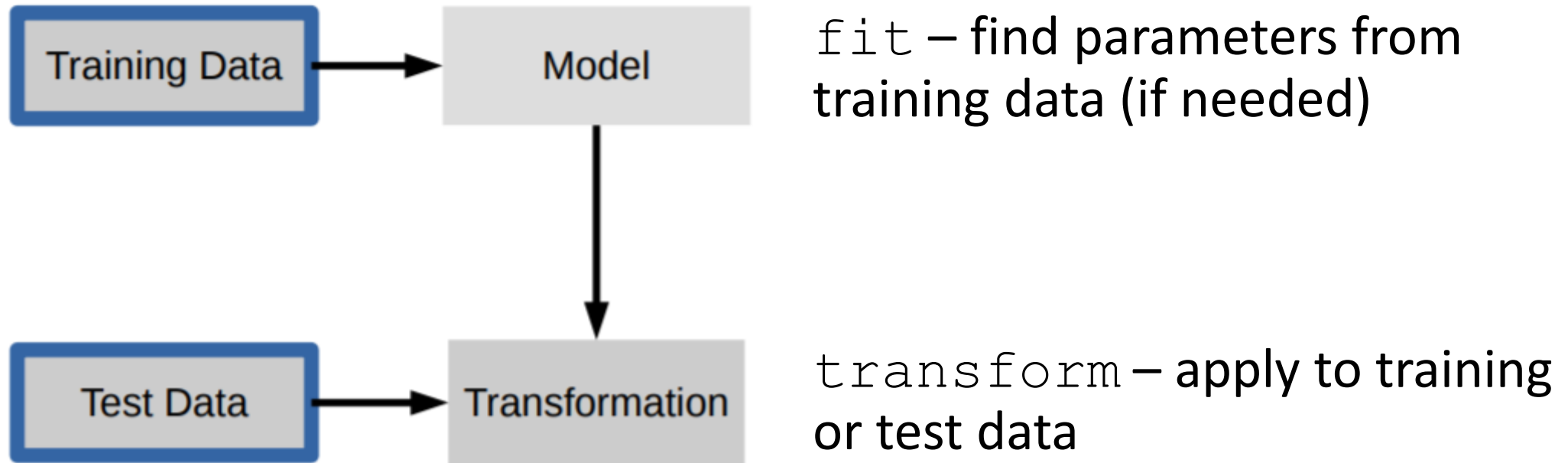


# scikit-learn Transformers



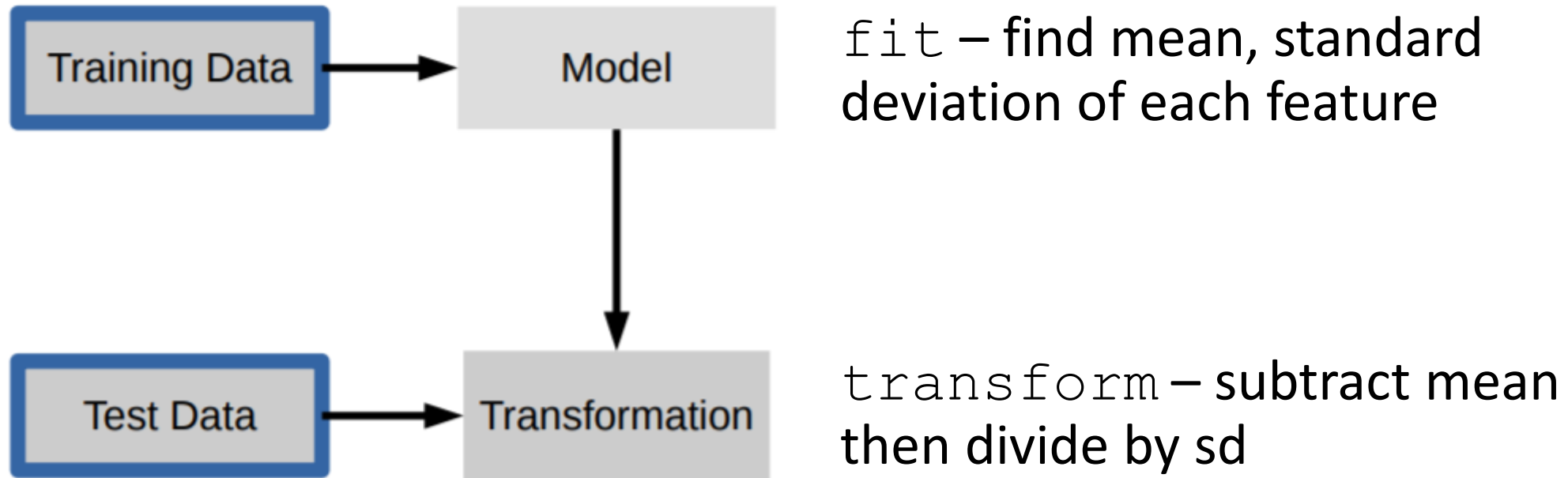
`fit` – find parameters from training data (if needed)

# scikit-learn Transformers

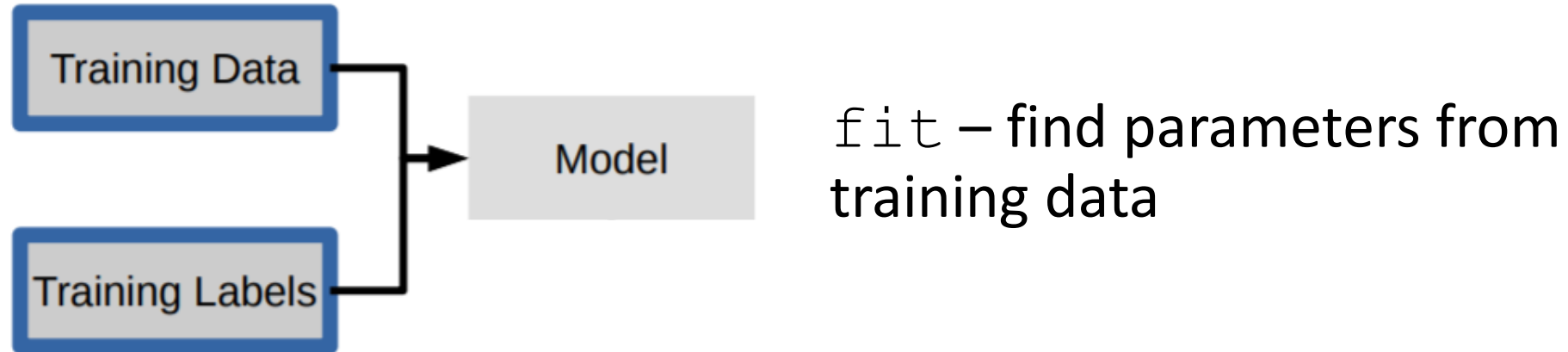




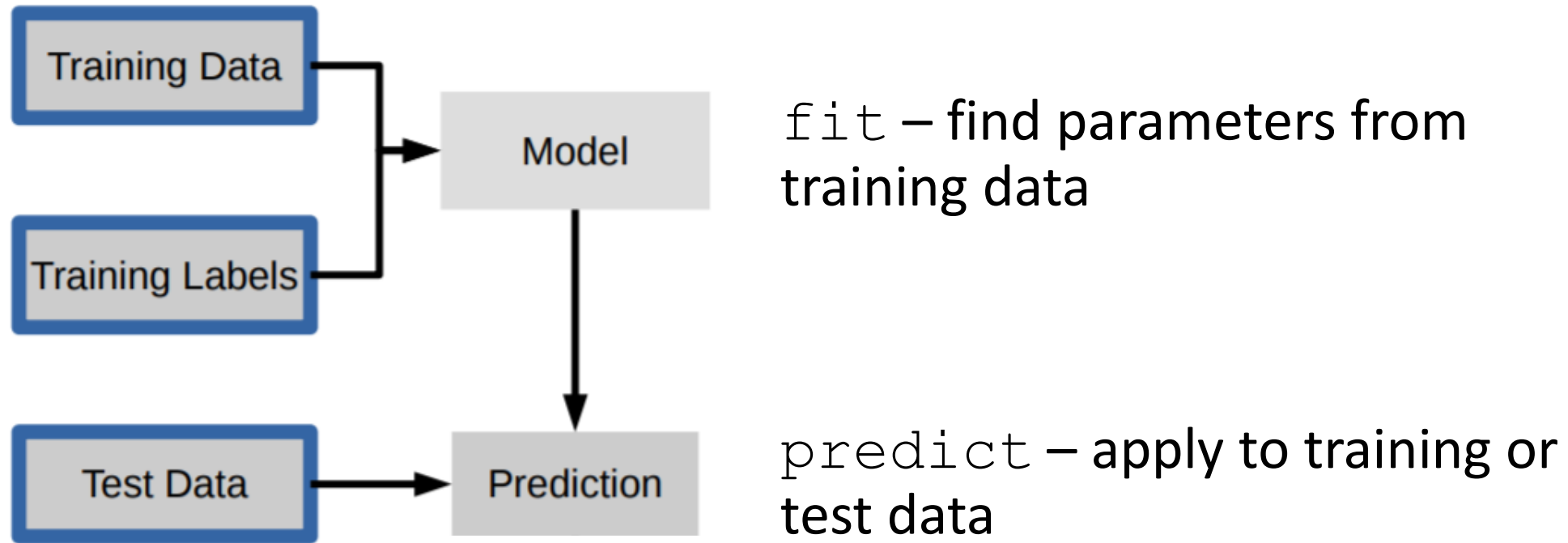
# scikit-learn StandardScaler



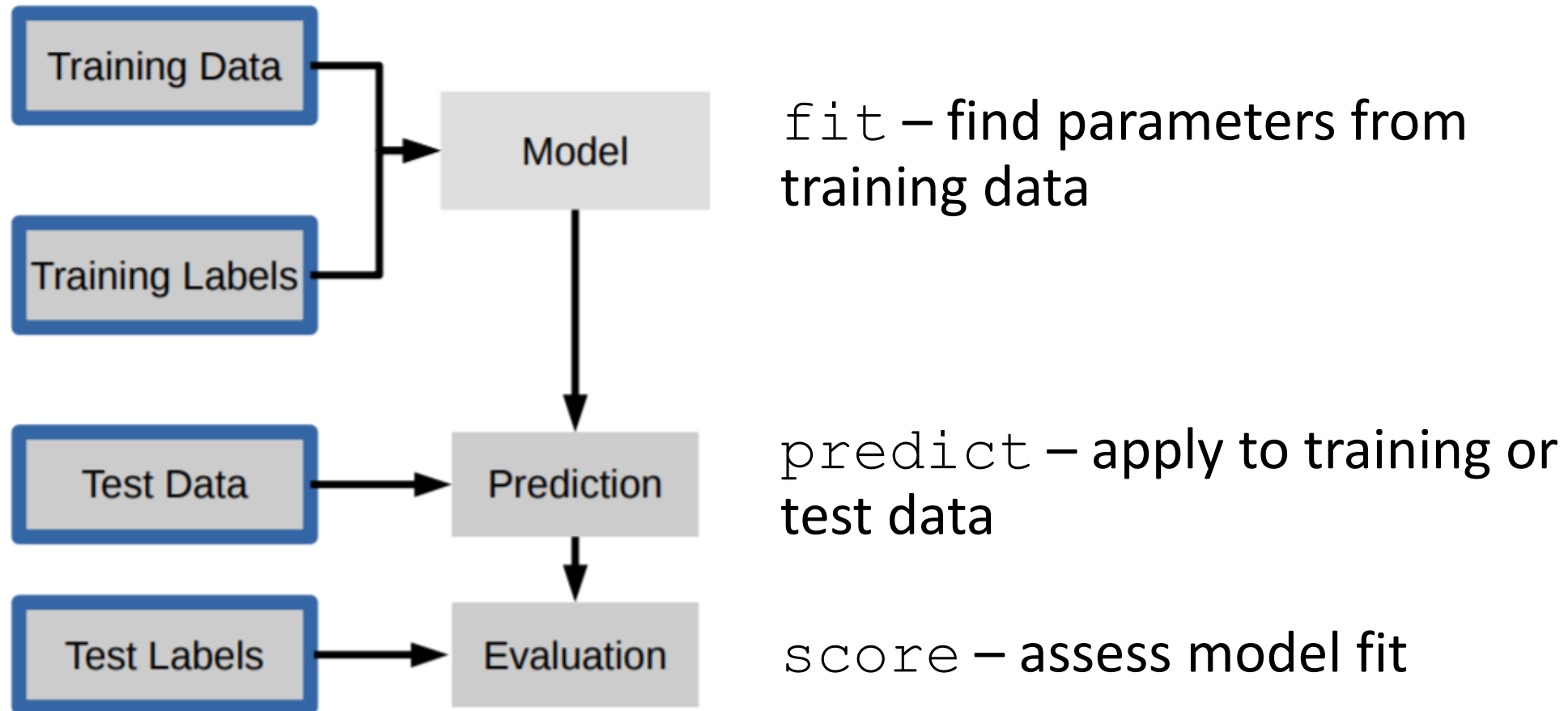
# scikit-learn Estimators



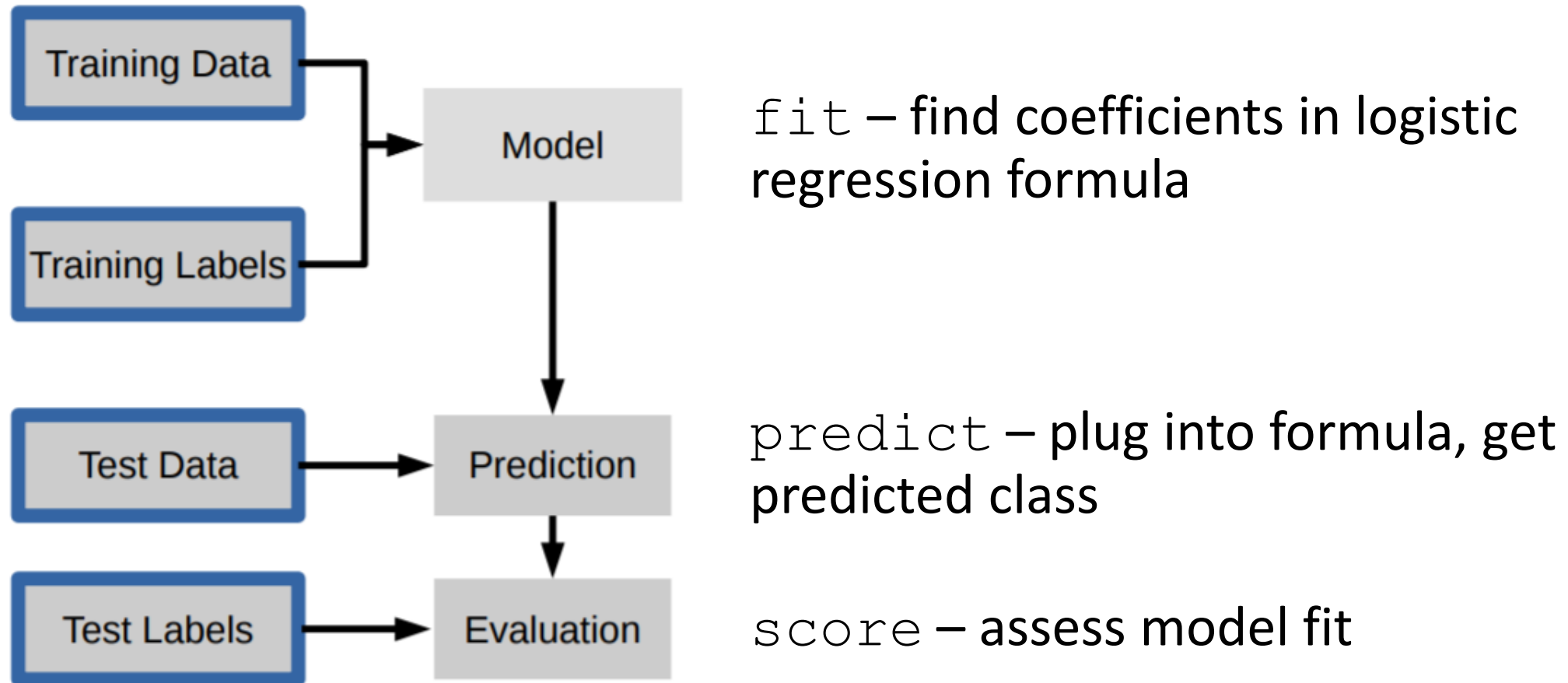
# scikit-learn Estimators



# scikit-learn Estimators



# scikit-learn LogisticRegression



# Alright then...let's see some code!

```
>>> imp = Imputer()
>>> quad = PolynomialFeatures()
>>> std = StandardScaler()

>>> X_train_imp = imp.fit_transform(X_train_raw)
>>> X_train_quad = quad.fit_transform(X_train_imp)
>>> X_train = std.fit_transform(X_train_quad)

>>> X_test_imp = imp.transform(X_test_raw)
>>> X_test_quad = quad.transform(X_test_imp)
>>> X_test = std.transform(X_test_quad)
```

# Alright then...let's see some code!

```
>>> imp = Imputer()
>>> quad = PolynomialFeatures()
>>> std = StandardScaler()

>>> X_train_imp = imp.fit_transform(X_train_raw)
>>> X_train_quad = quad.fit_transform(X_train_imp)
>>> X_train = std.fit_transform(X_train_quad)

>>> X_test_imp = imp.transform(X_test_raw)
>>> X_test_quad = quad.transform(X_test_imp)
>>> X_test = std.transform(X_test_quad)
```

# Notice anything?

- Repetitive, “boiler plate” code
- Crowded namespace
- Lots of objects to keep track of



# Notice anything?

- Repetitive, “boiler plate” code
- Crowded namespace
- Lots of objects to keep track of



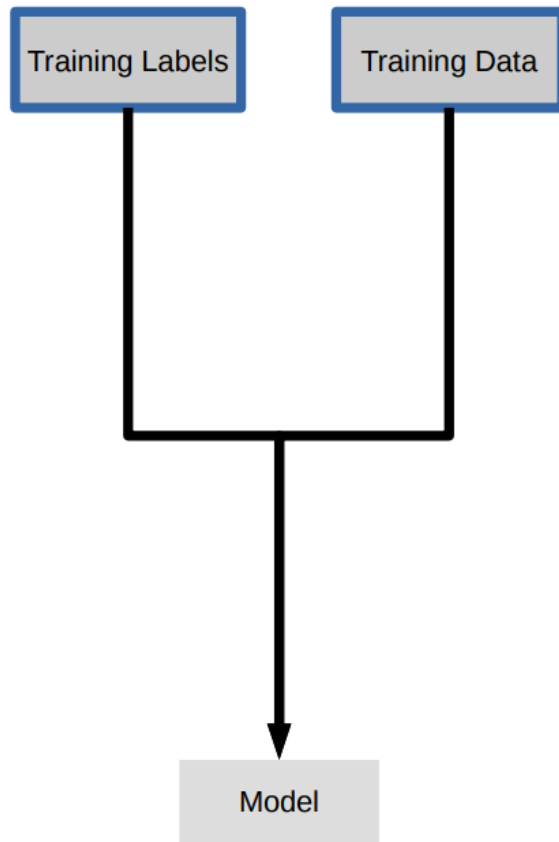
# Notice anything?

- Repetitive, “boiler plate” code
- Crowded namespace
- Lots of objects to keep track of



## Pipelines to the rescue!

# scikit-learn Pipelines



---

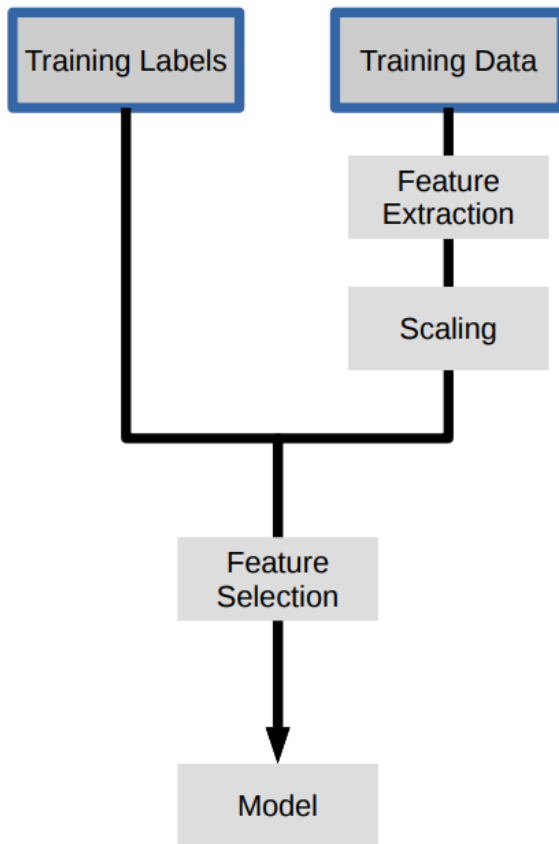
API Overview

**Pipelines**

Cross-validation

Customization

# scikit-learn Pipelines



---

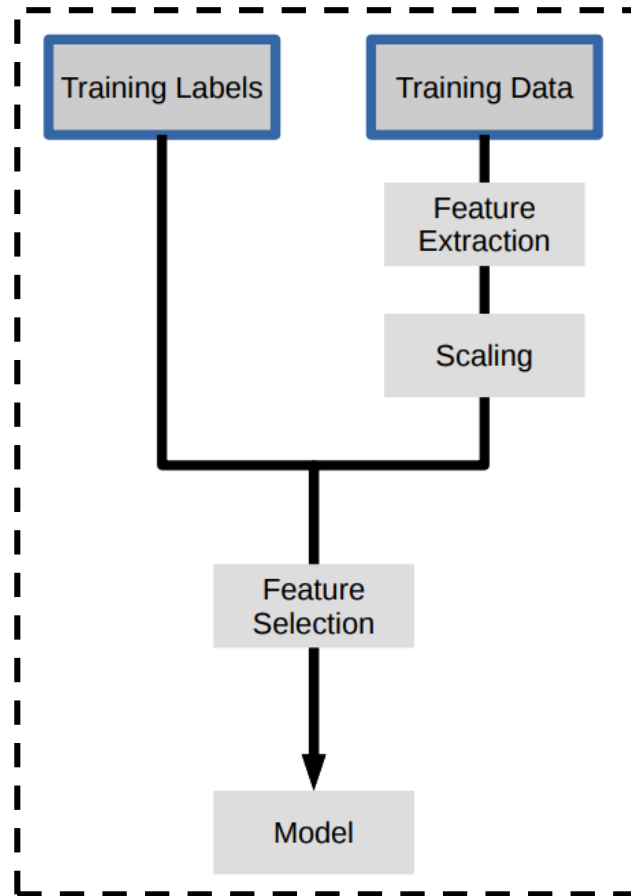
API Overview

**Pipelines**

Cross-validation

Customization

# scikit-learn Pipelines



- Encapsulate the modeling process
- Avoid repetitive code
- Hot-swap algorithms

# Back to the code

```
>>> imp = Imputer()
>>> quad = PolynomialFeatures()
>>> std = StandardScaler()

>>> X_train_imp = imp.fit_transform(X_train_raw)
>>> X_train_quad = quad.fit_transform(X_train_imp)
>>> X_train = std.fit_transform(X_train_quad)

>>> X_test_imp = imp.transform(X_test_raw)
>>> X_test_quad = quad.transform(X_test_imp)
>>> X_test = std.transform(X_test_quad)
```

# scikit-learn Pipelines

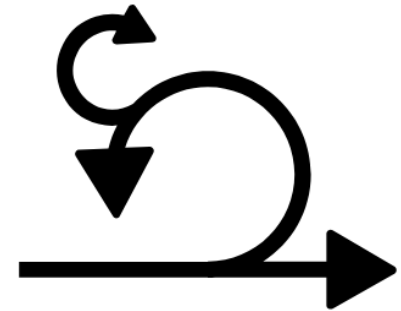
- Instead...use a pipeline!

```
>>> from sklearn.pipeline import Pipeline
>>> pipeline = Pipeline([
...     ('imp', Imputer()),
...     ('quad', PolynomialFeatures()),
...     ('std', StandardScaler())
... ])
>>> X_train = pipeline.fit_transform(X_train_raw)
>>> X_test = pipeline.transform(X_test_raw)
```

# scikit-learn Pipelines

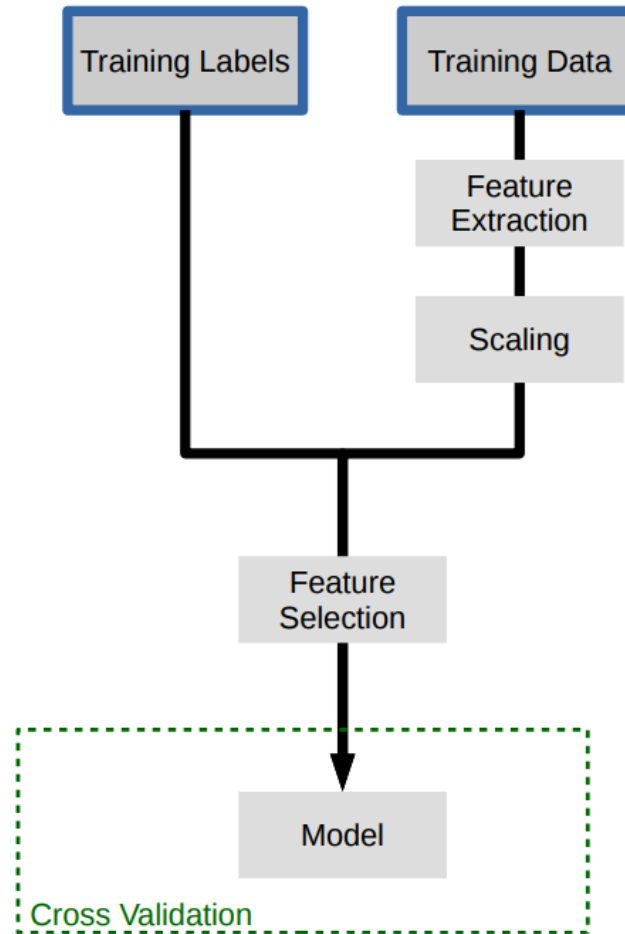
- Instead...use a pipeline!

```
>>> from sklearn.pipeline import Pipeline
>>> pipeline = Pipeline([
...     ('imp', Imputer()),
...     ('quad', PolynomialFeatures()),
...     ('std', StandardScaler())
... ])
>>> X_train = pipeline.fit_transform(X_train_raw)
>>> X_test = pipeline.transform(X_test_raw)
```





# Cross-validation the Wrong Way



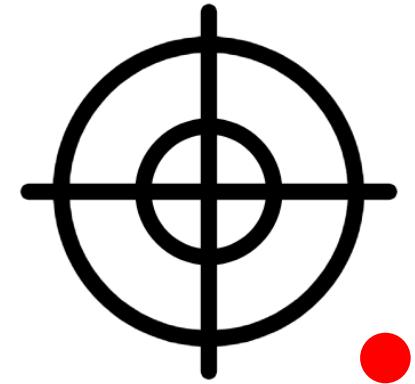
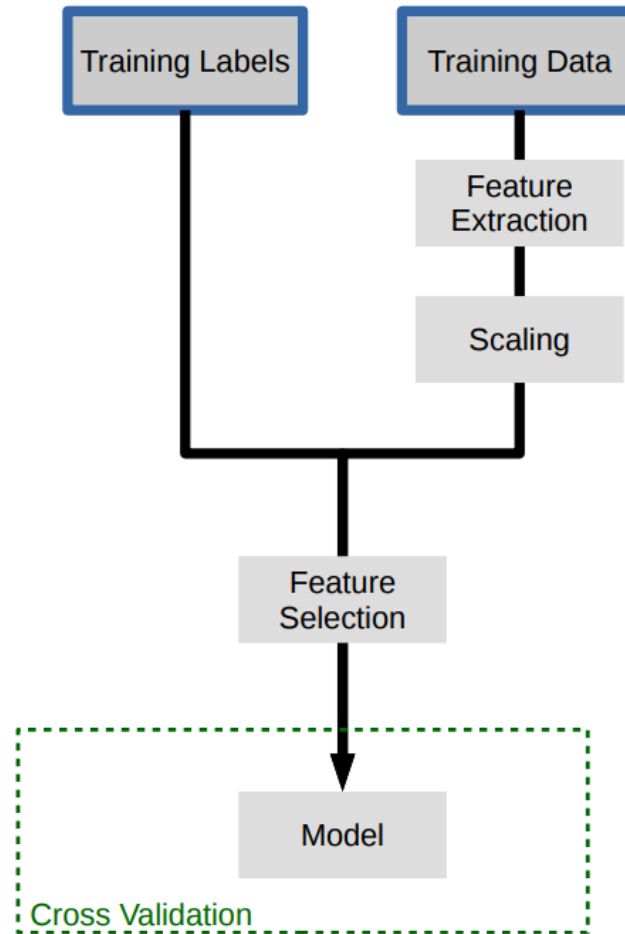
API Overview

Pipelines

**Cross-validation**

Customization

# Cross-validation the Wrong Way



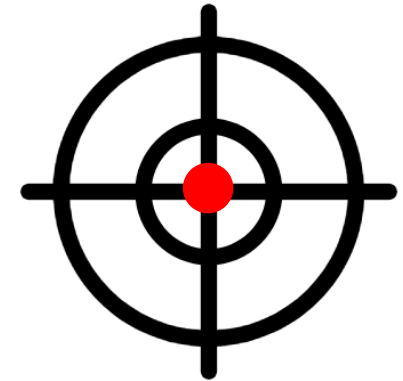
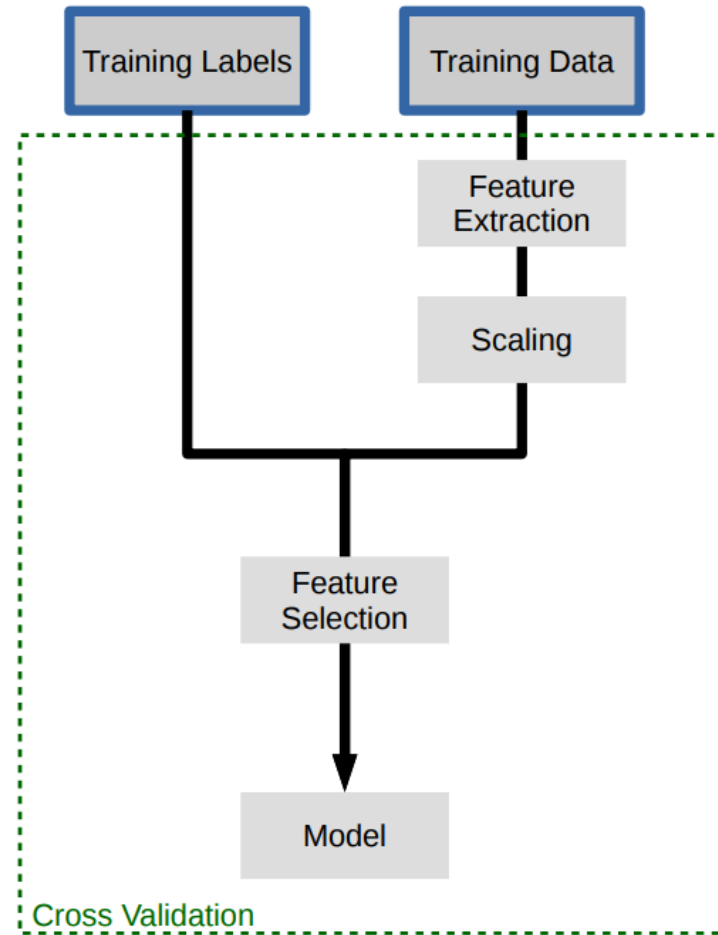
API Overview

Pipelines

**Cross-validation**

Customization

# Cross-validation the Right Way



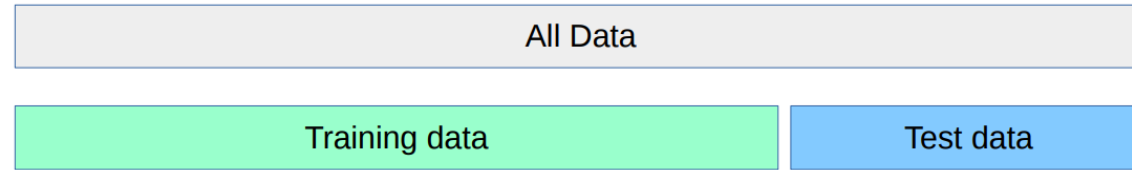
API Overview

Pipelines

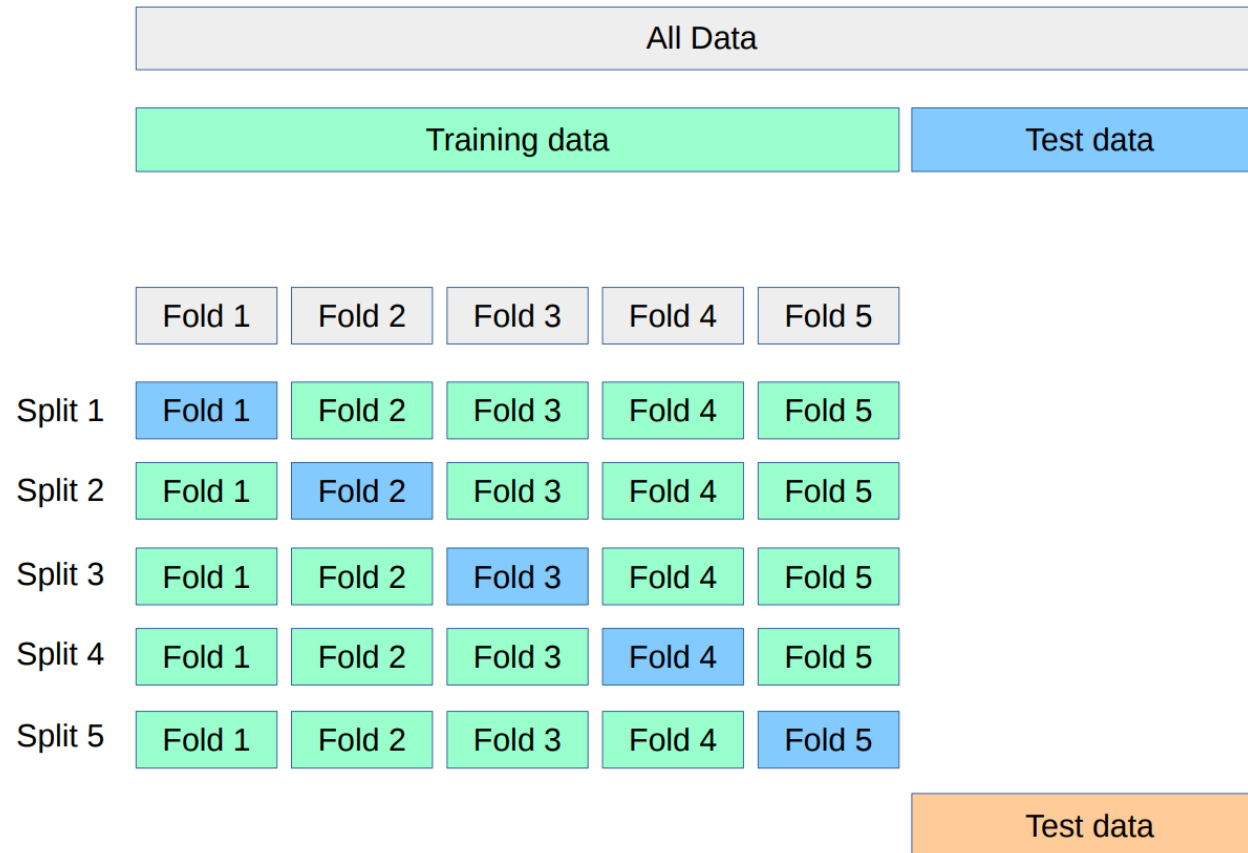
**Cross-validation**

Customization

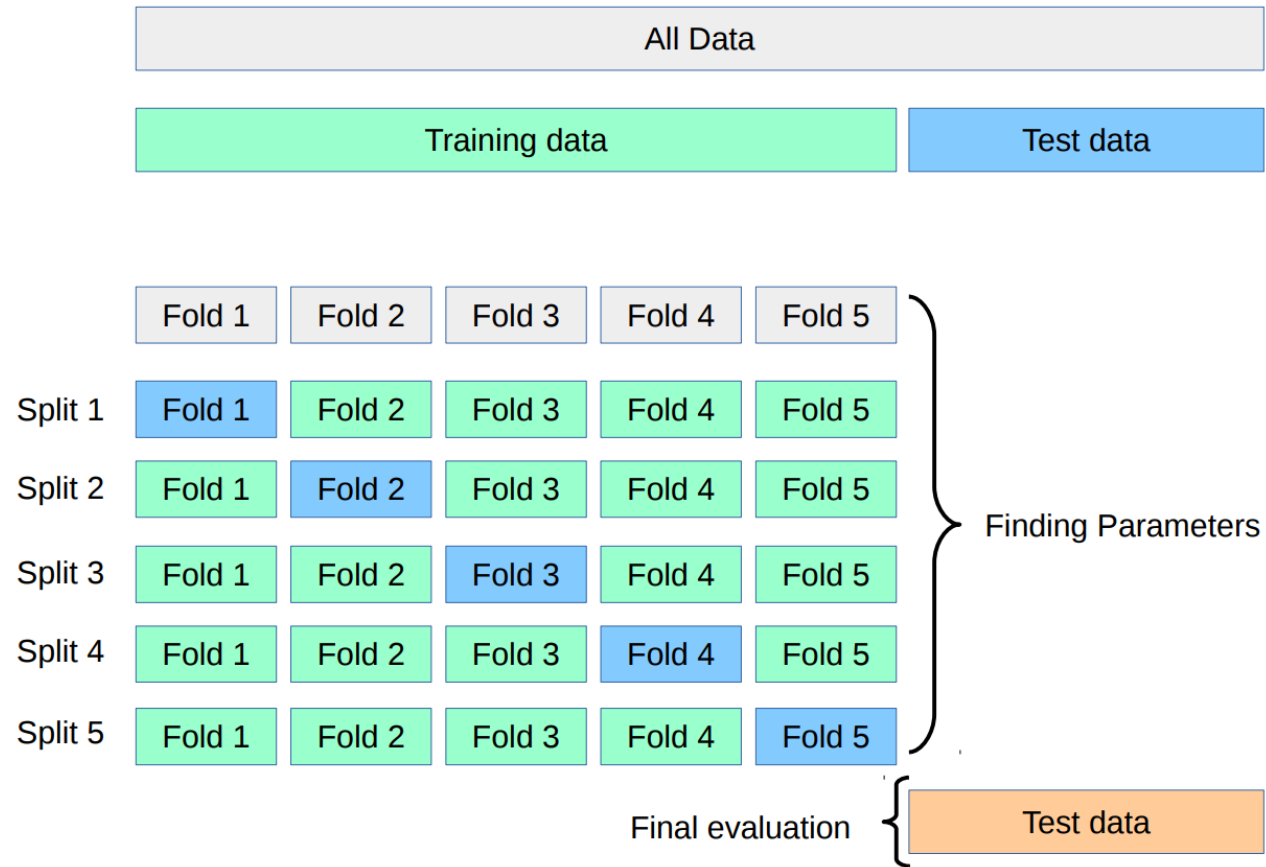
# Cross-validation



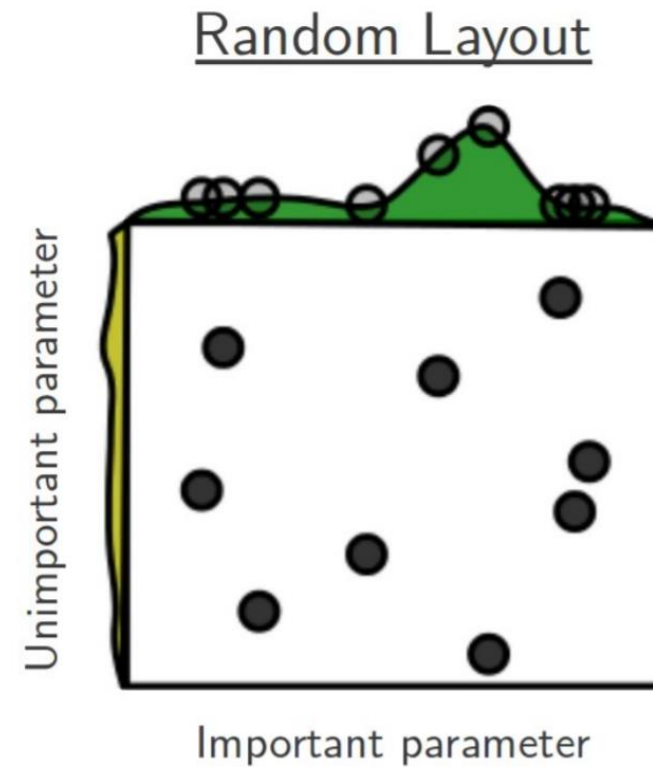
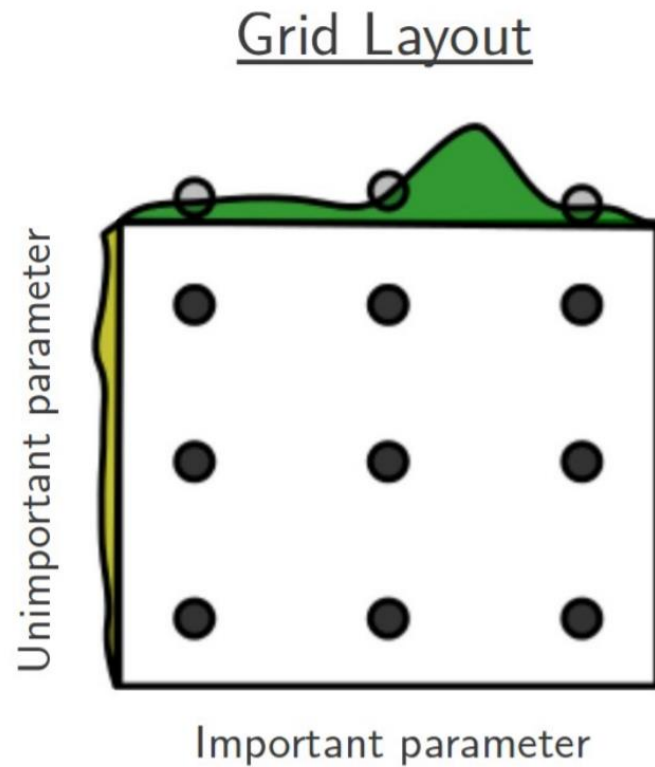
# Cross-validation



# Cross-validation



# Hyperparameter Tuning



What if `scikit-learn` doesn't meet our needs?

---

API Overview

Pipelines

Cross-validation

**Customization**



# What if `scikit-learn` doesn't meet our needs?

- Use the `FunctionTransformer`

```
>>> from sklearn.preprocessing import FunctionTransformer  
>>> logger = FunctionTransformer(np.log1p)  
>>> X_log = logger.fit_transform(X)
```

- Or...

## Extend the API!

# Custom Transformers

```
>>> from sklearn.base import TransformerMixin, BaseEstimator
>>> class SelectColumns(BaseEstimator, TransformerMixin):
>>>     def __init__(self, columns=[]):
>>>         self.columns = columns

>>>     def transform(self, X, **transform_params):
>>>         return X[self.columns].copy()

>>>     def fit(self, X, y=None, **fit_params):
>>>         return self
```

# Custom Transformers

```
>>> from sklearn.base import TransformerMixin, BaseEstimator
>>> class SelectColumns(BaseEstimator, TransformerMixin):
>>>     def __init__(self, columns=[]):
>>>         self.columns = columns

>>>     def transform(self, X, **transform_params):
>>>         return X[self.columns].copy()

>>>     def fit(self, X, y=None, **fit_params):
>>>         return self
```

# Custom Transformers

```
>>> from sklearn.base import TransformerMixin, BaseEstimator
>>> class SelectColumns(BaseEstimator, TransformerMixin):
>>>     def __init__(self, columns=[]):
>>>         self.columns = columns

>>>     def transform(self, X, **transform_params):
>>>         return X[self.columns].copy()

>>>     def fit(self, X, y=None, **fit_params):
>>>         return self
```

# Custom Transformers

```
>>> from sklearn.base import TransformerMixin, BaseEstimator
>>> class SelectColumns(BaseEstimator, TransformerMixin):
>>>     def __init__(self, columns=[]):
>>>         self.columns = columns

>>>     def transform(self, X, **transform_params):
>>>         return X[self.columns].copy()

>>>     def fit(self, X, y=None, **fit_params):
>>>         return self
```

# Custom Transformers

```
>>> from sklearn.base import TransformerMixin, BaseEstimator
>>> class SelectColumns(BaseEstimator, TransformerMixin):
>>>     def __init__(self, columns=[]):
>>>         self.columns = columns

>>>     def transform(self, X, **transform_params):
>>>         return X[self.columns].copy()

>>>     def fit(self, X, y=None, **fit_params):
>>>         return self
```

# Custom Transformer Use Cases

- Encoding multiple categorical variables
- Extracting columns by type
- Extracting units of time

# Custom Transformer Use Cases

- Encoding multiple categorical variables
- Extracting columns by type
- Extracting units of time

## **DictVectorizer**

- Not custom, just awesome!
- Handles new and unseen levels of categorical variables in test data
- Prevents mismatch of train and test data dimensions

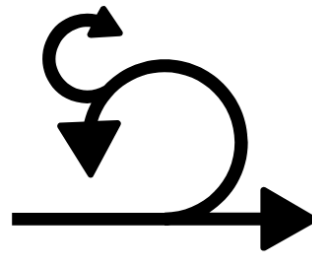


# Nested Pipeline with Custom Transformers

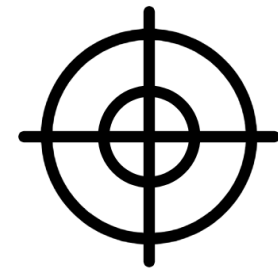
```
pipeline = Pipeline([
    ('drop_column', ColumnDropper(col=['service_cd', 'diagnosis_cd', 'county_calc'])),
    ('preproc', FeatureUnion([
        ('continuous', Pipeline([
            ('extract', ColumnExtractor(dtype='number')),
            ('impute', Imputer()),
            ('nearzero', VarianceThreshold())
        ])),
        ('factors', Pipeline([
            ('extract', ColumnExtractor(dtype='object')),
            ('labencode', MultiColumnLabelEncoder()),
            ('impute', Imputer(strategy='most_frequent')),
            ('onehot', OneHotEncoder(handle_unknown='ignore')),
        ])),
    ])),
    ('to_dense', DenseTransformer()),
    ('model', ExtraTreesRegressor(bootstrap=False))
])
```



Speed



Agility



Accuracy

# More ML Tools

- [Yellowbrick](#) – ML visualizations
- [Lime](#) – Explain ML predictions
- [imbalanced-learn](#) – Over- and under-sampling
- [sklearn-pandas](#) – Pandas integration with sklearn

# Special Thanks

- Andreas Mueller, Machine Learning Scientist at Columbia University
  - [Machine Learning with Scikit-learn](#), PyData NYC 2015
- Stephen Hoover, Lead Data Scientist at Civis Analytics
  - [Scaling Scikit-learn](#), PyData Seattle 2017
- Julie Michelman, Data Scientist at zulily
  - [Pandas, Pipelines, and Custom Transformers](#), PyData Seattle 2017
- Zac Stewart, Software Developer
  - [Using scikit-learn Pipelines and FeatureUnions](#)
- Zen Pursuits
  - [Pipelines, FeatureUnions, GridSearchCV, and Custom Transformers](#)