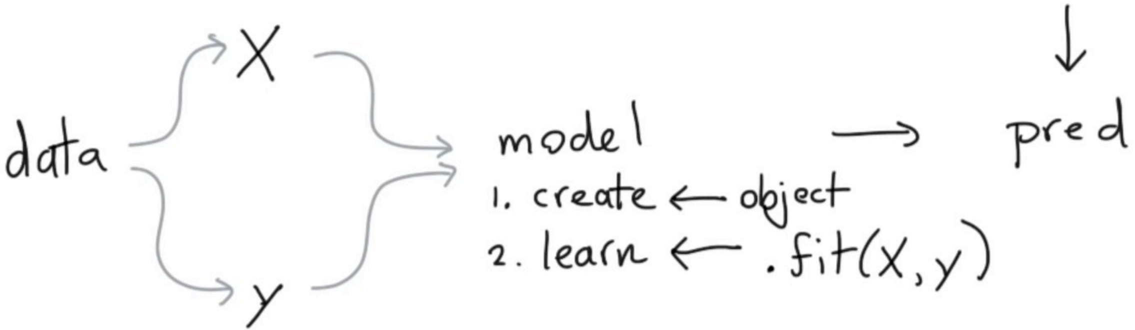

data
X
y
model
1. create ← object
2. learn ← .fit(X, y)
pred


X
y
info about the house
house prices

Scikit Learn

```
[ ]: %pip install --upgrade scikit-learn==0.23.0
```

```
[3]: from sklearn.datasets import load_boston
```

```
[6]: X, y = load_boston(return_X_y=True)
```

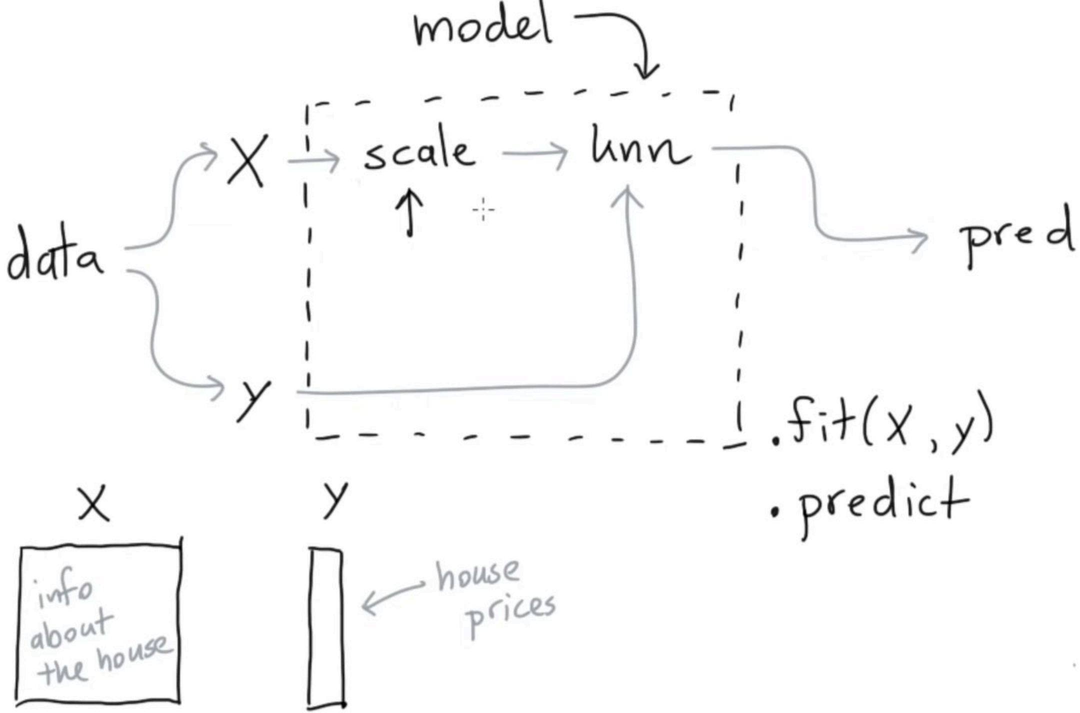
```
[7]: from sklearn.neighbors import KNeighborsRegressor
```

```
[8]: mod = KNeighborsRegressor()
```

```
[10]: mod.fit(X, y)
```

```
[10]: KNeighborsRegressor()
```

```
[ ]: mod.predict(X)
```


model
data
X
scale
knn
y
pred
.fit(X, y)
.predict
X
y
info about the house
house prices

```python
from sklearn.datasets import load_boston
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
import pandas as pd

X, y = load_boston(return_X_y=True)

# If n_neighbors = 1, we're totally cheating with the chart below.
pipe = Pipeline([
    ("scale", StandardScaler()),
    ("model", KNeighborsRegressor(n_neighbors=1))
])
pred = pipe.fit(X, y).predict(X)
plt.scatter(pred, y)

pipe.get_params()

###instead, lets do a gridSearch to look for diff. n_neighbors, and use cross validation = 3

mod = GridSearchCV(estimator=pipe,
                   param_grid={
                     'model__n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
                   },
                   cv=3)
mod.fit(X, y)
```

Python

```python
pd.DataFrame(mod.cv_results_)
```

Python

```
[71]: load_boston()
```

```
[71]: {'data': array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02,
              4.9800e+00],
             [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
              9.1400e+00],
             [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
              4.0300e+00],
             ...,
             [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
              5.6400e+00],
             [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
              6.4800e+00],
             [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
              7.8800e+00]]),
      'target': array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15. ,
             18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
             15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
             13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
             21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
             35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
```

```
[73]: print(load_boston()['DESCR'])
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually th
e target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
```

One Hot Encoding

```python
arr = np.array(["low", "low", "high", "medium"]).reshape(-1, 1)
arr
```

[12] ✓ 0.0s Python

```
array([['low'],
       ['low'],
       ['high'],
       ['medium']], dtype='<U6')
```

Alfabetical

```python
from sklearn.preprocessing import OneHotEncoder
```

[13] ✓ 0.0s Python

```python
enc = OneHotEncoder(sparse=False, handle_unknown='ignore')
enc.fit_transform(arr)
```

[73] Python

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

```python
enc.transform([["zero"]])
```

Edit ⌘K Chat ⌘L ···

[74] Python

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

Chat with Ca

Ask questions o
coding in gener

Ask anything (

Write Chat

0 available M

Past Conversat

Resolving RO

Using Natura

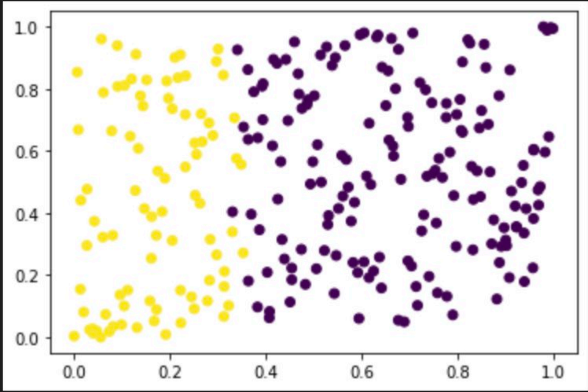
Fixing CartPo

Show 15 more...

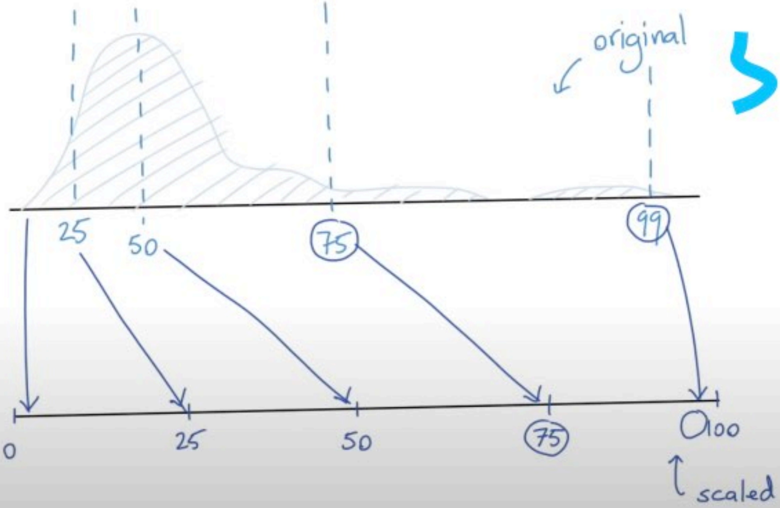
AI may make mist

```python
from sklearn.preprocessing import StandardScaler, QuantileTransformer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline

X_new = QuantileTransformer(n_quantiles=100).fit_transform(X)
plt.scatter(X_new[:, 0], X_new[:, 1], c=y);
```

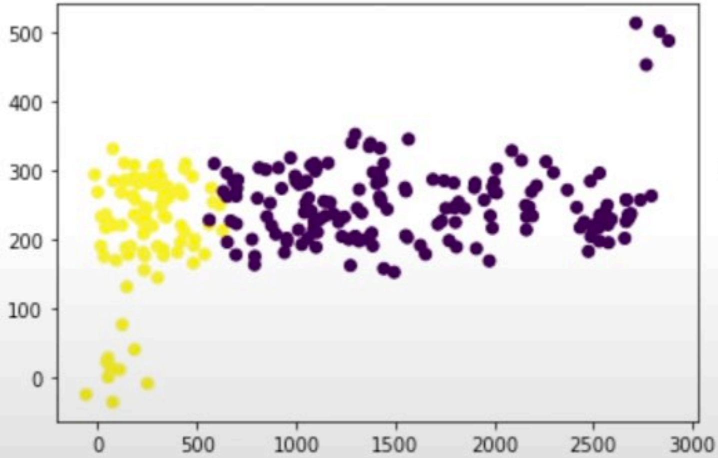
Python



Quantile Scaling



```
[5]: plt.scatter(X[:, 0], X[:, 1], c=y);
```



scaling

StanardScaler

each column

mean var

$$\frac{x - \text{mean}(x)}{\sqrt{\text{var}}}$$

```python
X = df.drop(columns=['Time', 'Amount', 'Class']).values
y = df['Class'].values
f"Shapes of X={X.shape} y={y.shape}, #Fraud Cases={y.sum()}"
```

```
'Shapes of X=(80000, 28) y=(80000,), #Fraud Cases=196'
```

+ Code + Markdown

```python
from sklearn.linear_model import LogisticRegression

mod = LogisticRegression(class_weight={0: 1, 1: 2}, max_iter=1000)
mod.fit(X, y).predict(X).sum()
```

```
171
```

```python
lr = LogisticRegression()
??lr.score
```

```
Signature: lr.score(X, y, sample_weight=None)
Source:
    def score(self, X, y, sample_weight=None):
        """
        Return the mean accuracy on the given test data and labels.

        In multi-label classification, this is the subset accuracy
        which is a harsh metric since you require for each sample that
        each label set be correctly predicted.

        Parameters
        ----------
        X : array-like of shape (n_samples, n_features)
            Test samples.

        y : array-like of shape (n_samples,) or (n_samples, n_outputs)
            True labels for X.

        sample_weight : array-like of shape (n_samples,), default=None
            Sample weights.
```

?? imp

```python
    param_grid={'class_weight': [{0: 1, 1: v} for v in range(1, 4)]},
    cv=4,
    n_jobs=-1
)
grid.fit(X, y);
```

given that i predict fraud
how accurate am i

```python
from sklearn.metrics import precision_score, recall_score

recall_score(y, grid.predict(X))
```

did i get all the fraud cases

```
0.5918367346938775
```

```python
pd.DataFrame(grid.cv_results_)
```

```python
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import precision_score, recall_score, make_scorer

def min_recall_precision(est, X, y_true, sample_weight=None):
    y_pred = est.predict(X)
    recall = recall_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    return min(recall, precision)

grid = GridSearchCV(
    estimator=LogisticRegression(max_iter=1000),
    param_grid={'class_weight': [{0: 1, 1: v} for v in np.linspace(1, 20, 30)]},
    scoring={'precision': make_scorer(precision_score),
             'recall': make_scorer(recall_score),
             'min_both': min_recall_precision},
    refit='min_both',
    return_train_score=True,
    cv=10,
    n_jobs=-1
)
grid.fit(X, y);
```

Pytho

```python
def min_recall_precision(y_true, y_pred):
    recall = recall_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    return min(recall, precision)

make_scorer(min_recall_precision, greater_is_better=False)
# ?make_scorer
```

Python

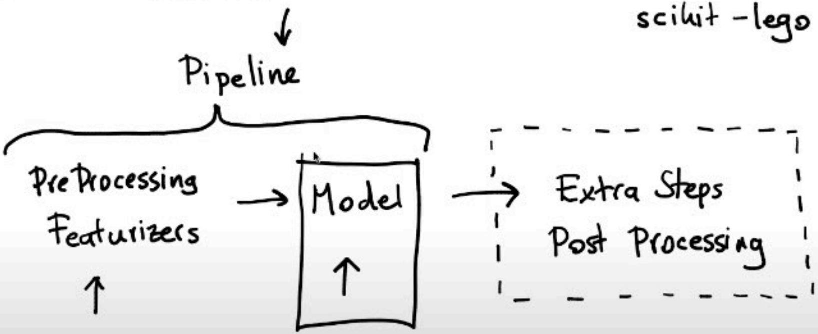
```
make_scorer(min_recall_precision, greater_is_better=False)
```

```python
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import precision_score, recall_score, make_scorer

def min_recall_precision(est, X, y_true, sample_weight=None):
    y_pred = est.predict(X)
    recall = recall_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    return min(recall, precision)

grid = GridSearchCV(
    estimator=LogisticRegression(max_iter=1000),
    param_grid={'class_weight': [{0: 1, 1: v} for v in np.linspace(1, 20, 30)]},
    scoring={'precision': make_scorer(precision_score),
             'recall': make_scorer(recall_score),
             'min_both': min_recall_precision},
    refit='min_both',
    return_train_score=True,
    cv=10,
    n_jobs=-1
)
grid.fit(X, y);
```

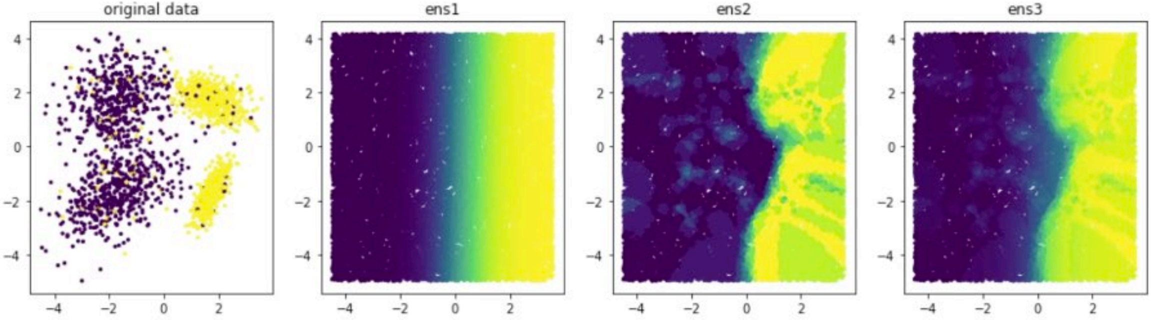
Python

Meta Estimators

scikit-learn
scikit-lego

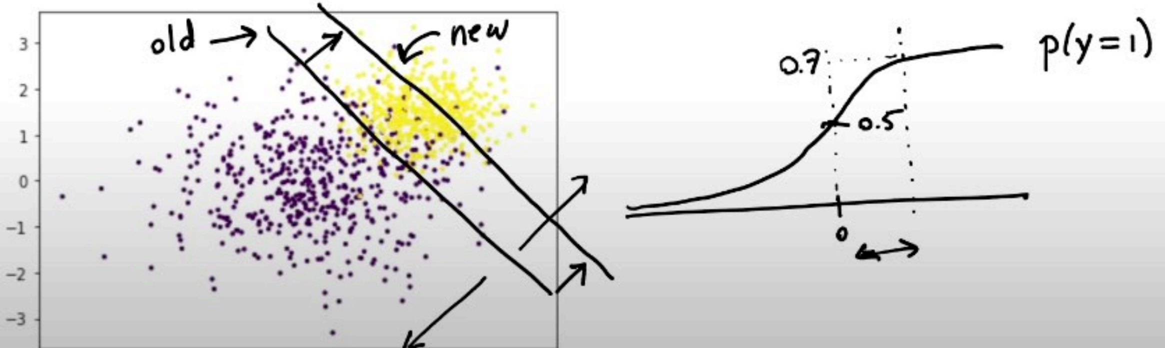


```
clf1 = LogisticRegression().fit(X, y)
clf2 = KNeighborsClassifier(n_neighbors=10).fit(X, y)
clf3 = VotingClassifier(estimators=[('clf1', clf1), ('clf2', clf2)],
                        voting='soft',
                        weights=[0.5, 0.5])
clf3.fit(X, y)

make_plots()
```

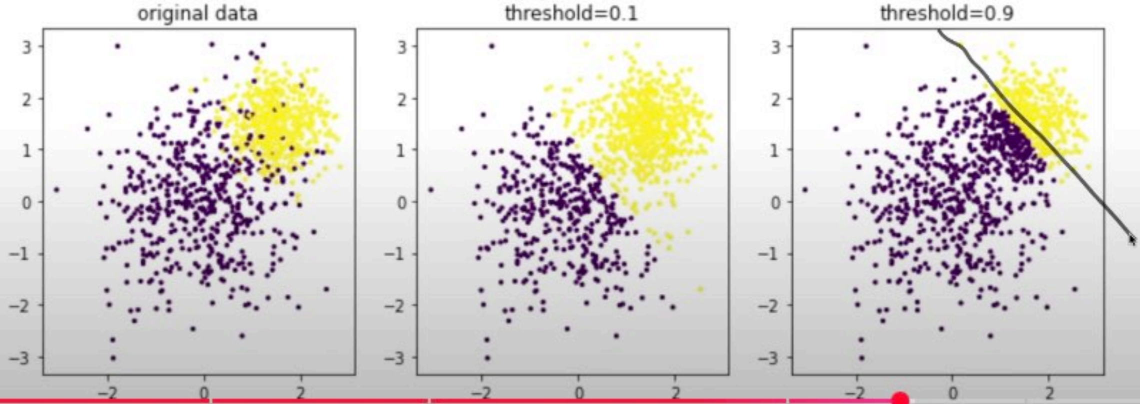


```
from sklego.meta import Thresholder
```

```
X, y = make_blobs(1000, centers=[(0, 0), (1.5, 1.5)], cluster_std=[1, 0.5])
plt.scatter(X[:, 0], X[:, 1], c=y, s=5);
```



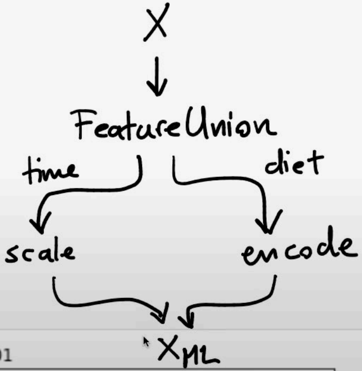
```python
m1 = Thresholder(LogisticRegression(solver='lbfgs'), threshold=0.1).fit(X, y)
m2 = Thresholder(LogisticRegression(solver='lbfgs'), threshold=0.9).fit(X, y)
```



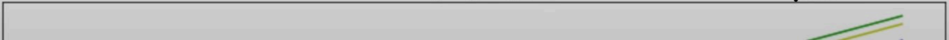
```
pipe = Pipeline([
    ("model", Thresholder(LogisticRegression(solver='lbfgs'), threshold=0.1))
])

mod = GridSearchCV(estimator=pipe,
                   param_grid = {"model__threshold": np.linspace(0.1, 0.9, 50)},
                   scoring={"precision": make_scorer(precision_score),
                            "recall": make_scorer(recall_score),
                            "accuracy": make_scorer(accuracy_score)},
                   refit="precision",
                   cv=5)

mod.fit(X, y);
```

[28]:

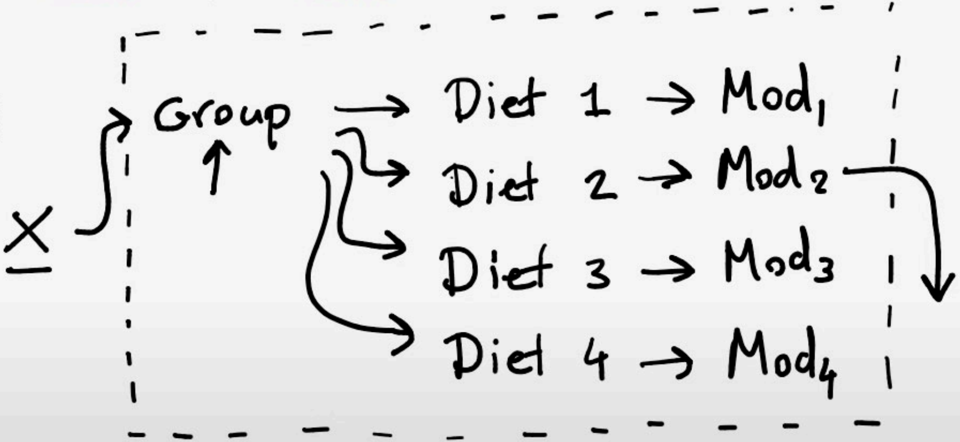
```
feature_pipeline = Pipeline([
    ("datagrab", FeatureUnion([
        ("discrete", Pipeline([
            ("grab", ColumnSelector("diet")),
            ("encode", OneHotEncoder(categories="auto", sparse=False))
        ])),
        ("continuous", Pipeline([
            ("grab", ColumnSelector("time")),
            ("standardize", StandardScaler())
        ]))
    ]))
])

pipe = Pipeline([
    ("transform", feature_pipeline),
    ("model", LinearRegression())
])

plot_model(pipe)
```



linear model per group, MAE: 25.01



```
ctor("diet")),
coder(categories="auto", sparse=False))

([
ctor("time")),
ndardScaler())




ne),
)
```



Grouped Models

```python
import numpy as np
import pandas as pd
import matplotlib.pylab as plt
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error

from sklego.datasets import load_chicken
from sklego.preprocessing import ColumnSelector

df = load_chicken(as_frame=True)

def plot_model(model):
    df = load_chicken(as_frame=True)
    model.fit(df[['diet', 'time']], df['weight'])
    metric_df = df[['diet', 'time', 'weight']].assign(pred=lambda d: model.predict(d[['diet', 'time
    metric = mean_absolute_error(metric_df['weight'], metric_df['pred'])
    plt.figure(figsize=(12, 4))
    # plt.scatter(df['time'], df['weight'])
    for i in [1, 2, 3, 4]:
        pltr = metric_df[['time', 'diet', 'pred']].drop_duplicates().loc[lambda d: d['diet'] == i]
        plt.plot(pltr['time'], pltr['pred'], color='.rbgy'[i])
    plt.title(f"linear model per group, MAE: {np.round(metric, 2)}");
```

[12]

Python

```python
feature_pipeline = Pipeline([
    ("datagrab", FeatureUnion([
        ("discrete", Pipeline([
            ("grab", ColumnSelector("diet")),
            ("encode", OneHotEncoder(categories="auto", sparse=False))
        ])),
        ("continuous", Pipeline([
            ("grab", ColumnSelector("time")),
            ("standardize", StandardScaler())
        ]))
    ]))
])

pipe = Pipeline([
    ("transform", feature_pipeline),
    ("model", LinearRegression())
])

plot_model(pipe)
```

Python

```python
from sklego.meta import GroupedPredictor
mod = GroupedPredictor(LinearRegression(), groups=["diet"])
plot_model(mod)
```

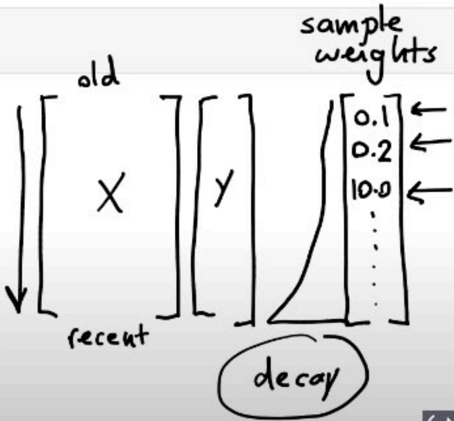
🧑‍💻 Example: Using `DummyRegressor`

python Copy Edit

```python
from sklearn.dummy import DummyRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import numpy as np

# Generate some dummy data
np.random.seed(42)
X = np.random.rand(100, 1)  # 100 samples, 1 feature
y = 3 * X.squeeze() + np.random.randn(100)  # Linear relation with noise

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Initialize and train DummyRegressor (mean strategy)
dummy = DummyRegressor(strategy="mean")
dummy.fit(X_train, y_train)

# Make predictions
y_pred_dummy = dummy.predict(X_test)

# Evaluate with MSE
mse_dummy = mean_squared_error(y_test, y_pred_dummy)
print(f"Dummy Regressor MSE: {mse_dummy:.4f}")
```

```
4]: _ = DummyRegressor().fit
    ?_
```

```
Signature: _(X, y, sample_weight=None)
Docstring:
Fit the random regressor.

Parameters
----------
X : {array-like, object with finite length or shape}
    Training data, requires length = n_samples

y : array-like of shape (n_samples,) or (n_samples, n_outputs)
    Target values.

sample_weight : array-like of shape (n_samples,), default=None
    Sample weights.

Returns
```



```python
mod1 = (GroupedPredictor(DummyRegressor(), groups=["m"])
        .fit(df[['m']], df['yt']))

mod2 = (GroupedPredictor(DecayEstimator(DummyRegressor(), decay=0.9), groups=["m"])
        .fit(df[['index', 'm']], df['yt']))
```