Today

- Recap dim. reduction
- Introduction this week
- Scikit-learn
- Keras

Recap dimensionality reduction

- High dimensional data prevents lots of difficulties: sparseness, proneness to overfitting, etc.
- We can reduce the number of dimensions while keeping most of the information we're interested in.
- Non-linear methods: UMAP and consorts who try to best capture high-dimensional distance properties in low dimensions
- Linear methods: PCA and consorts, where PCA defines linear combinations of old dimensions (i.e. rotations of the data) where the first new PC captures most of the variance, the second the second most, etc.

Introduction this week

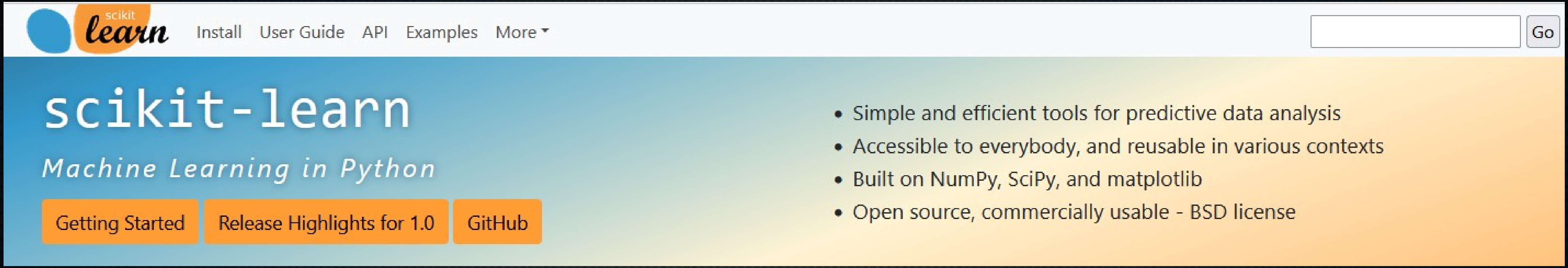
- So far:
  - Supervised learning implemented yourself (linear regression, logistic regression, simple/dense neural network)
  - Unsupervised learning implemented yourself (K-means, hierarchical clustering, PCA)
- Intimate knowledge of the fundamentals, of what happens under the hood and why.

Introduction this week

- Now:
  - Switch to current applied workflows
    - Scikit-learn for supervised and unsupervised ML
    - Keras for easy creation and training of (convolutional) neural networks
  - Do a project with these high-level libraries, using the knowledge you’ve gained

Scikit-learn

- Open-source library



scikit-learn

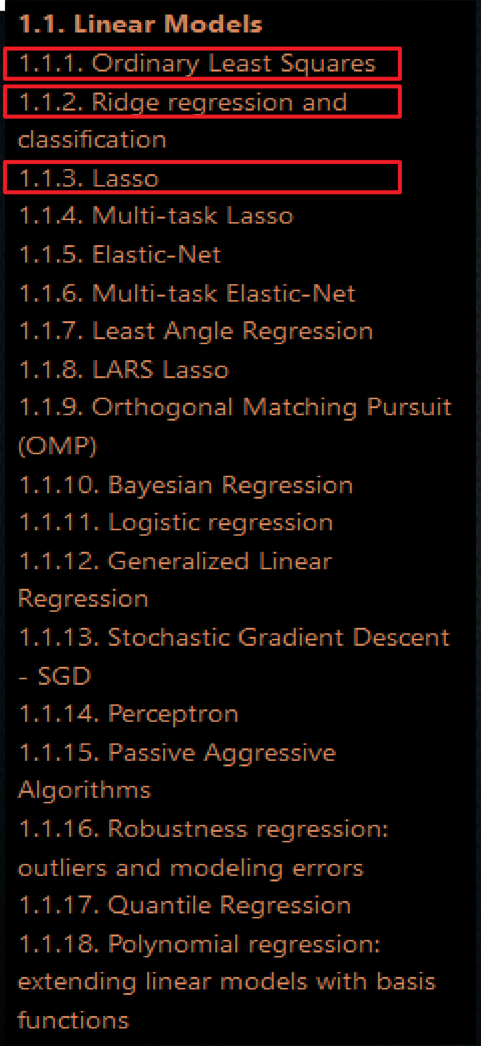
Machine Learning in Python

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

- Implements a wealth of ML algorithms

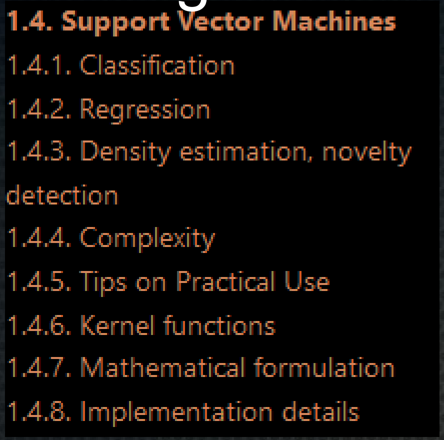
Scikit-learn

- Implements a wealth of ML algorithms and preprocessing/plotting functions:



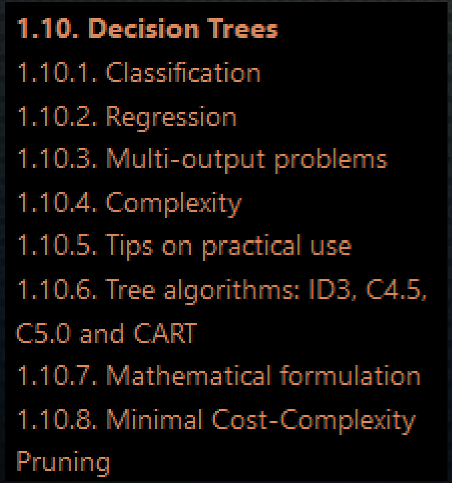
1.1. Linear Models

1.1.1. Ordinary Least Squares
1.1.2. Ridge regression and classification
1.1.3. Lasso
1.1.4. Multi-task Lasso
1.1.5. Elastic-Net
1.1.6. Multi-task Elastic-Net
1.1.7. Least Angle Regression
1.1.8. LARS Lasso
1.1.9. Orthogonal Matching Pursuit (OMP)
1.1.10. Bayesian Regression
1.1.11. Logistic regression
1.1.12. Generalized Linear Regression
1.1.13. Stochastic Gradient Descent - SGD
1.1.14. Perceptron
1.1.15. Passive Aggressive Algorithms
1.1.16. Robustness regression: outliers and modeling errors
1.1.17. Quantile Regression
1.1.18. Polynomial regression: extending linear models with basis functions



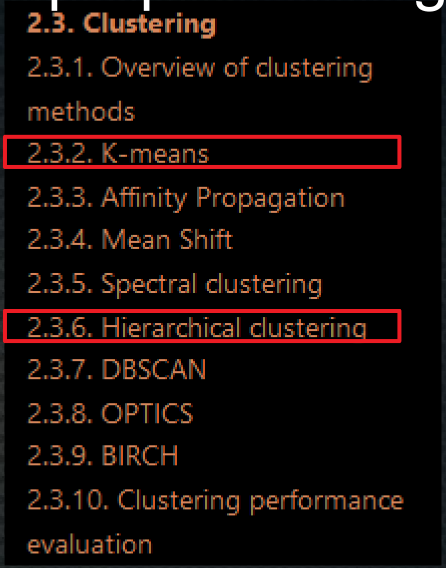
1.4. Support Vector Machines

1.4.1. Classification
1.4.2. Regression
1.4.3. Density estimation, novelty detection
1.4.4. Complexity
1.4.5. Tips on Practical Use
1.4.6. Kernel functions
1.4.7. Mathematical formulation
1.4.8. Implementation details



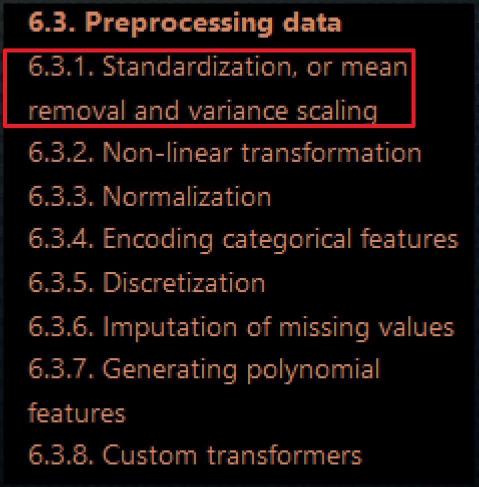
1.10. Decision Trees

1.10.1. Classification
1.10.2. Regression
1.10.3. Multi-output problems
1.10.4. Complexity
1.10.5. Tips on practical use
1.10.6. Tree algorithms: ID3, C4.5, C5.0 and CART
1.10.7. Mathematical formulation
1.10.8. Minimal Cost-Complexity Pruning



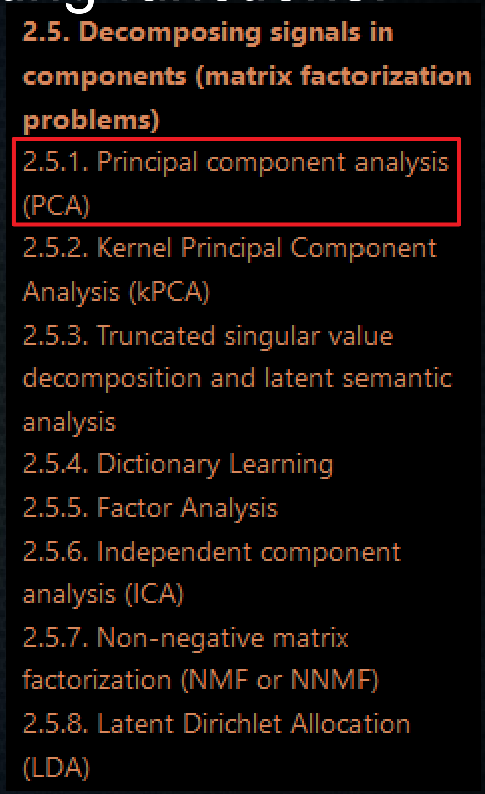
2.3. Clustering

2.3.1. Overview of clustering methods
2.3.2. K-means
2.3.3. Affinity Propagation
2.3.4. Mean Shift
2.3.5. Spectral clustering
2.3.6. Hierarchical clustering
2.3.7. DBSCAN
2.3.8. OPTICS
2.3.9. BIRCH
2.3.10. Clustering performance evaluation



6.3. Preprocessing data

6.3.1. Standardization, or mean removal and variance scaling
6.3.2. Non-linear transformation
6.3.3. Normalization
6.3.4. Encoding categorical features
6.3.5. Discretization
6.3.6. Imputation of missing values
6.3.7. Generating polynomial features
6.3.8. Custom transformers



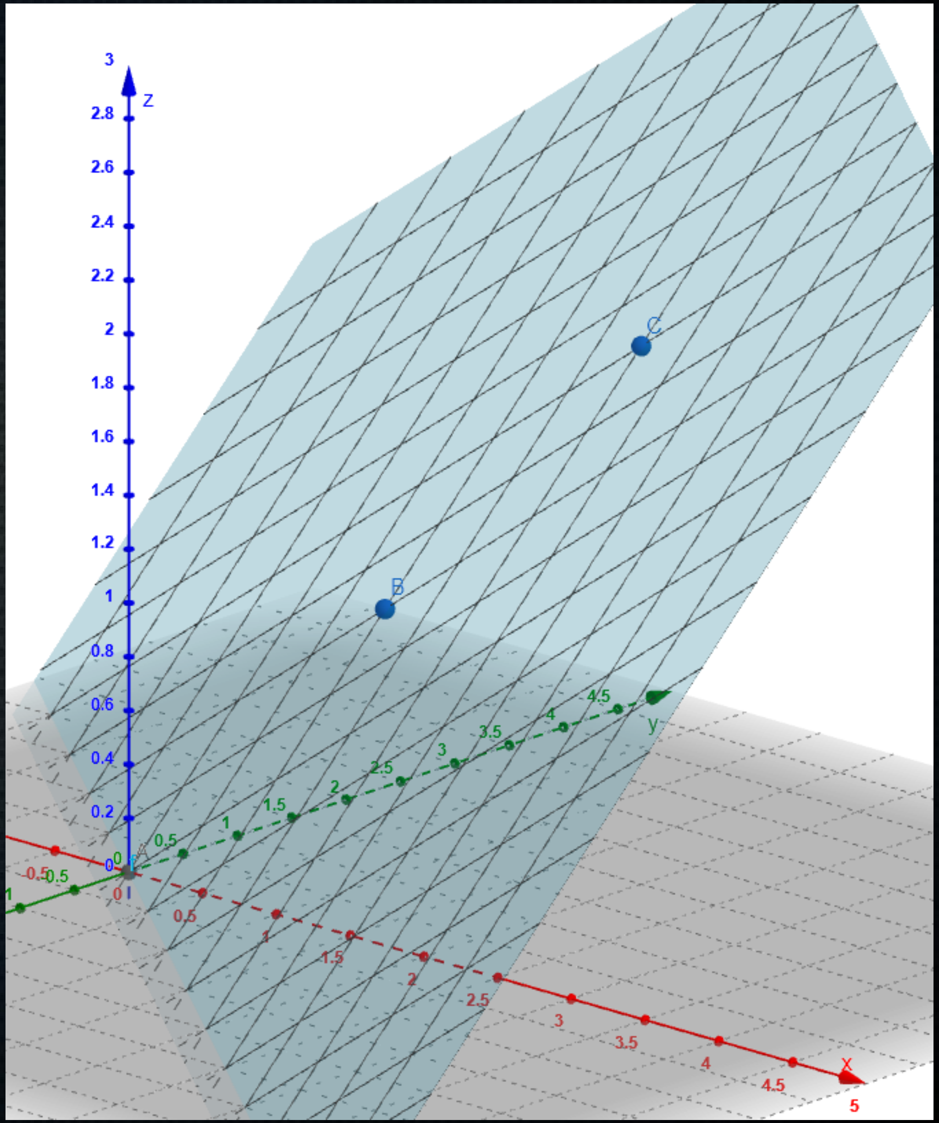
2.5. Decomposing signals in components (matrix factorization problems)

2.5.1. Principal component analysis (PCA)
2.5.2. Kernel Principal Component Analysis (kPCA)
2.5.3. Truncated singular value decomposition and latent semantic analysis
2.5.4. Dictionary Learning
2.5.5. Factor Analysis
2.5.6. Independent component analysis (ICA)
2.5.7. Non-negative matrix factorization (NMF or NNMF)
2.5.8. Latent Dirichlet Allocation (LDA)

Scikit-learn

- Similar API for all models:

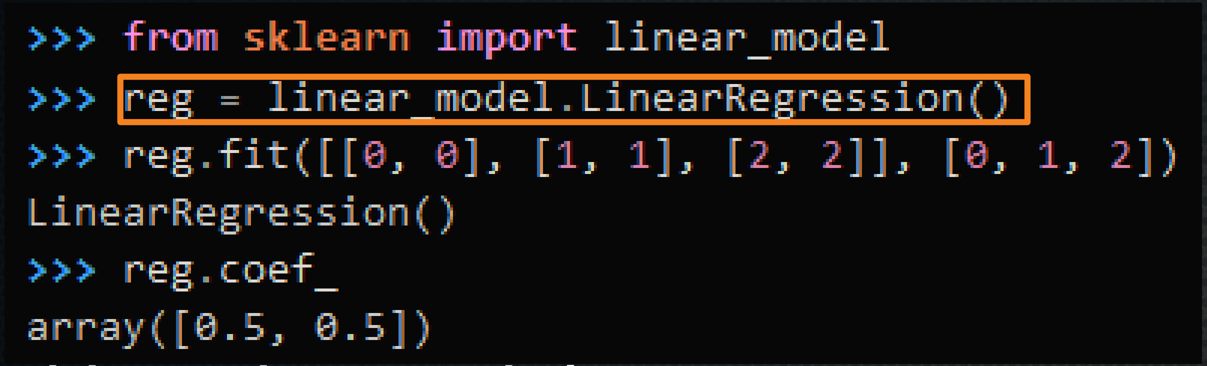
```
>>> from sklearn import linear_model
>>> reg = linear_model.LinearRegression()
>>> reg.fit([[0, 0], [1, 1], [2, 2]], [0, 1, 2])
LinearRegression()
>>> reg.coef_
array([0.5, 0.5])
```



Made using: https://www.geogebra.org/3d

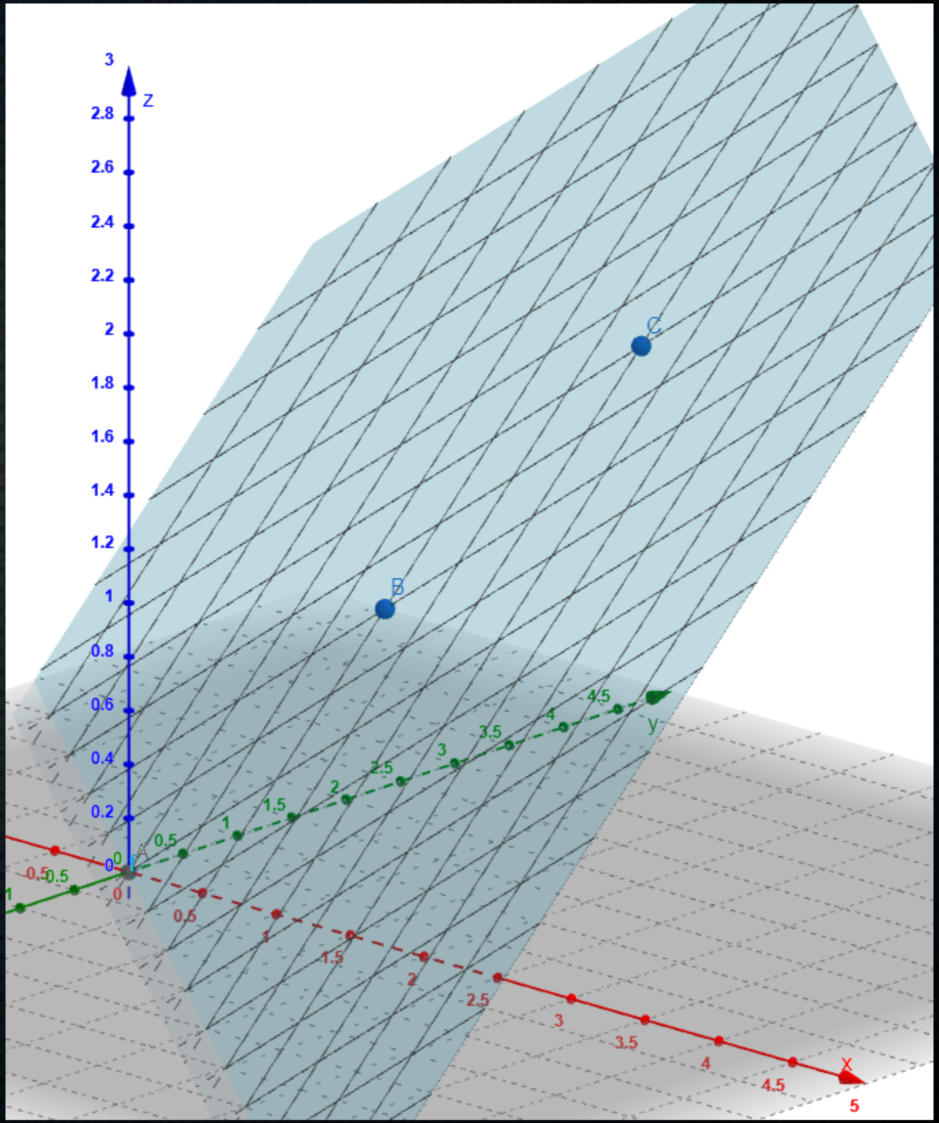
Scikit-learn

- Similar API for all models:



```
>>> from sklearn import linear_model
>>> reg = linear_model.LinearRegression()
>>> reg.fit([[0, 0], [1, 1], [2, 2]], [0, 1, 2])
LinearRegression()
>>> reg.coef_
array([0.5, 0.5])
```

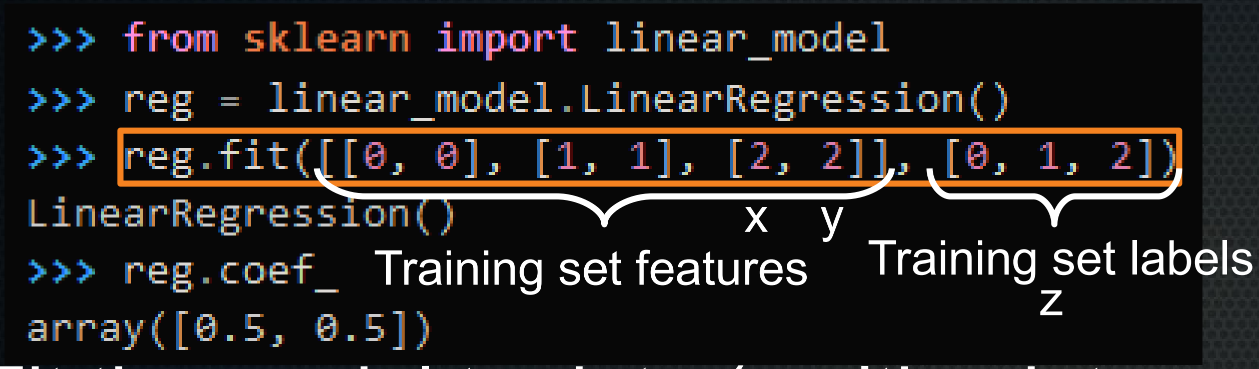
- Initiate the model



Made using: https://www.geogebra.org/3d

Scikit-learn

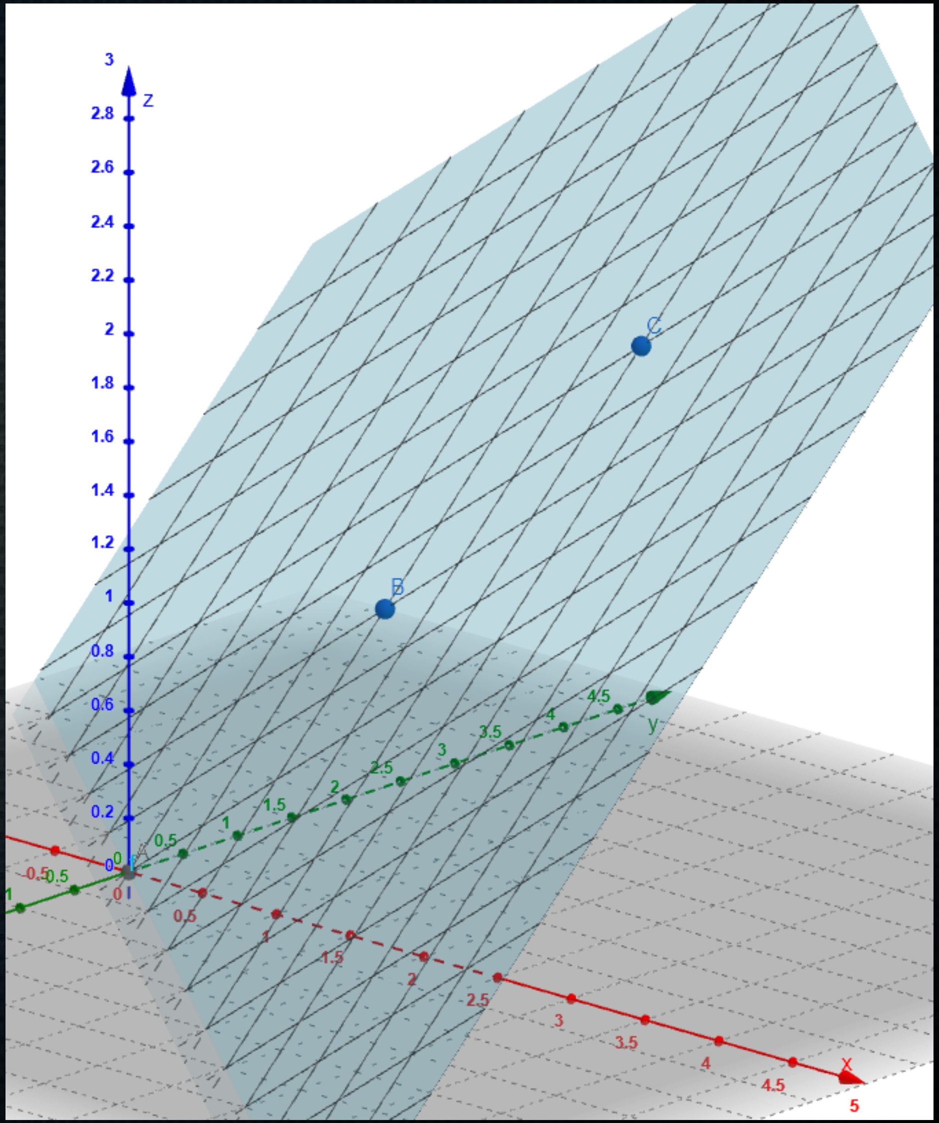
- Similar API for all models:



```
>>> from sklearn import linear_model
>>> reg = linear_model.LinearRegression()
>>> reg.fit([[0, 0], [1, 1], [2, 2]], [0, 1, 2])
LinearRegression()
>>> reg.coef_
array([0.5, 0.5])
```

- Fit the model to data (multivariate linear regression with z predicted from x and y)

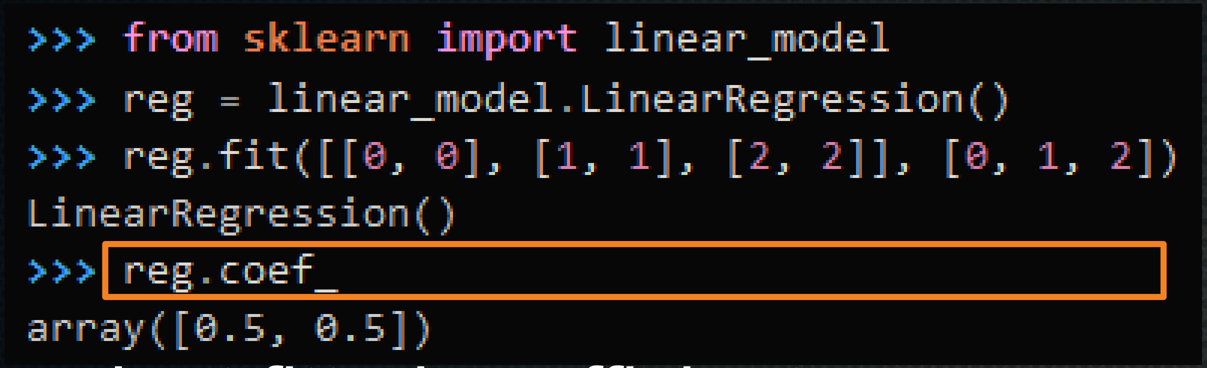
$$h_\theta(x, y) = [x \quad y] \cdot \theta^T$$



Made using: https://www.geogebra.org/3d

Scikit-learn

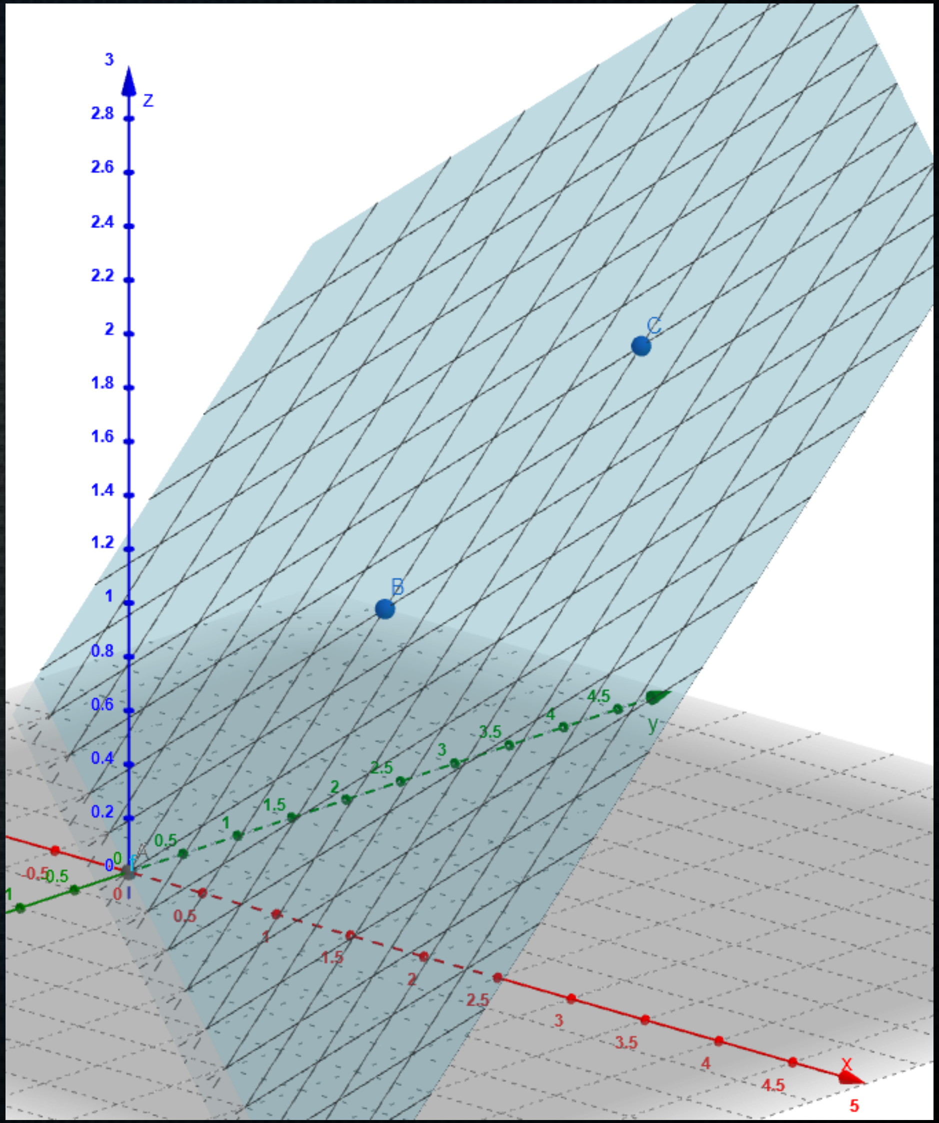
- Similar API for all models:



```
>>> from sklearn import linear_model
>>> reg = linear_model.LinearRegression()
>>> reg.fit([[0, 0], [1, 1], [2, 2]], [0, 1, 2])
LinearRegression()
>>> reg.coef_
array([0.5, 0.5])
```

- Look at fitted coefficients

$$h_\theta(x, y) = [x \quad y] \cdot \theta^T \quad \theta = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$



Made using: https://www.geogebra.org/3d

Scikit-learn

- Similar API for all models:

```
>>> from sklearn import linear_model
>>> reg = linear_model.LinearRegression()
>>> reg.fit([[0, 0], [1, 1], [2, 2]], [0, 1, 2])
LinearRegression()
>>> reg.coef_
array([0.5, 0.5])
```

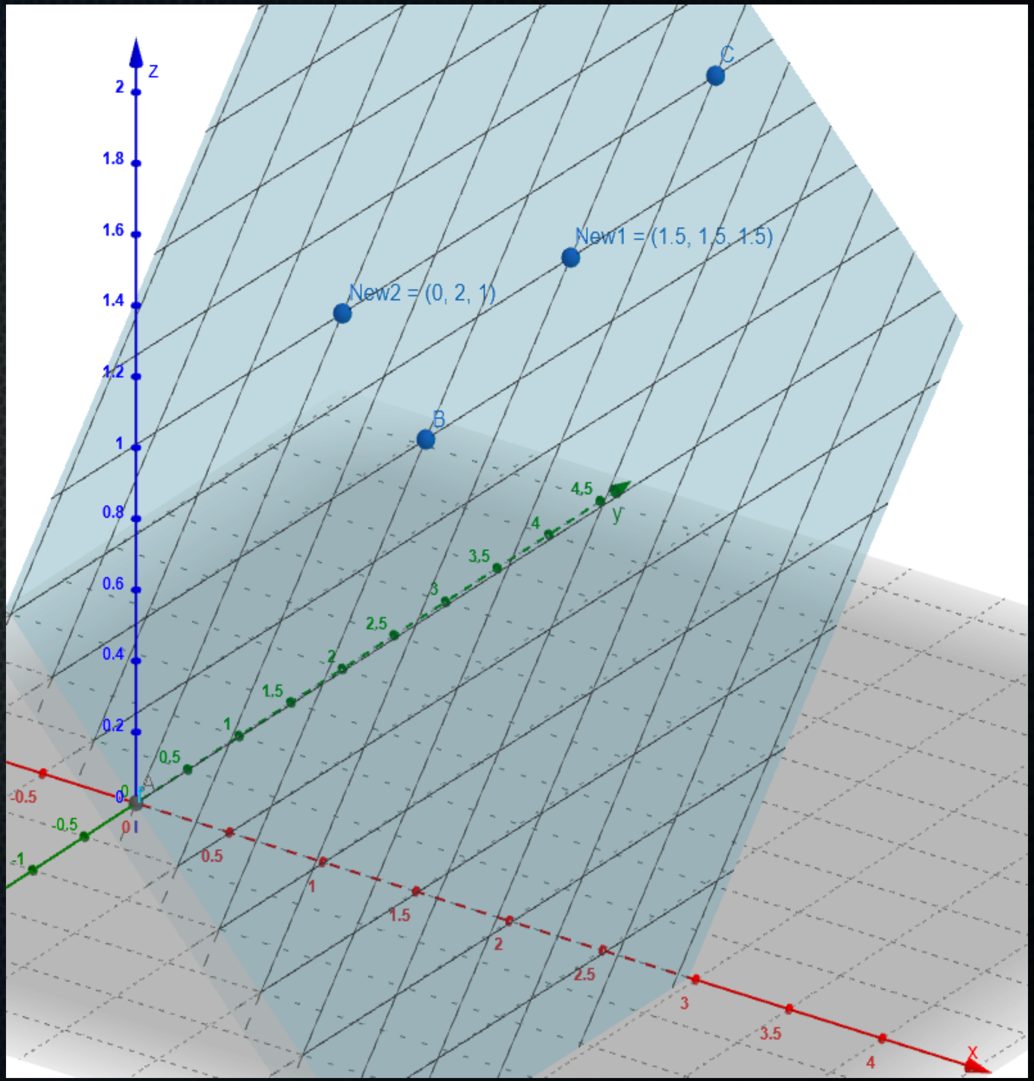
```
reg.predict([[1.5, 1.5], [0,2]])
```

```
array([1.5, 1. ])
```

$$\theta = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

- Predict new data

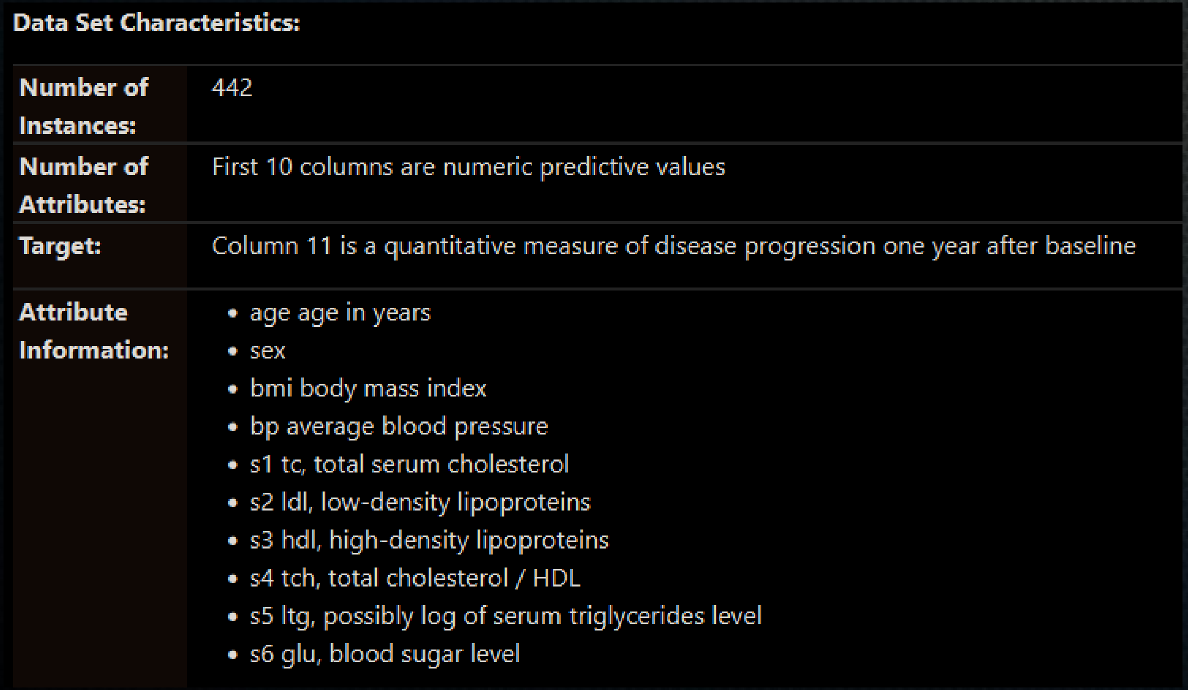
$$h_\theta(x, y) = \begin{bmatrix} 1.5 & 1.5 \\ 0 & 2 \end{bmatrix} \cdot \theta^T = \begin{bmatrix} 1.5 \\ 1 \end{bmatrix}$$



Made using: https://www.geogebra.org/3d

Scikit-learn: split data

```python
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
```



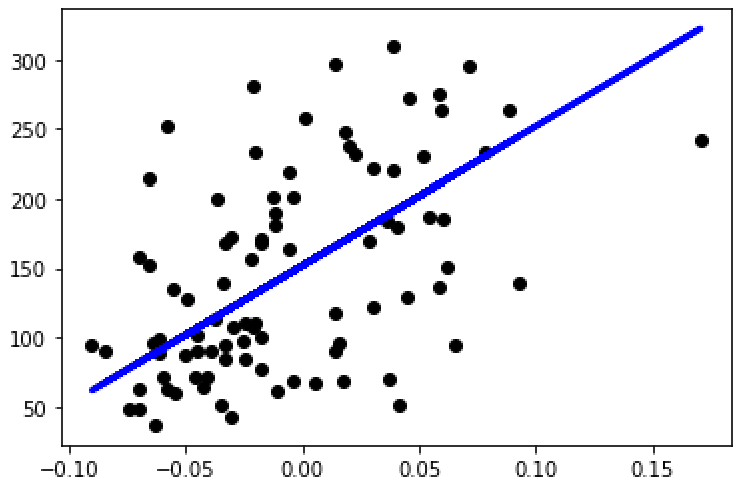
Data Set Characteristics:

Number of Instances:	442
Number of Attributes:	First 10 columns are numeric predictive values
Target:	Column 11 is a quantitative measure of disease progression one year after baseline
Attribute Information:	• age age in years • sex • bmi body mass index • bp average blood pressure • s1 tc, total serum cholesterol • s2 ldl, low-density lipoproteins • s3 hdl, high-density lipoproteins • s4 tch, total cholesterol / HDL • s5 ltg, possibly log of serum triglycerides level • s6 glu, blood sugar level

Scikit-learn: split data

```
# Use only one feature
diabetes_X = diabetes_X[:, np.newaxis, 2]

#split data:
diabetes_X_train, diabetes_X_test, diabetes_y_train, diabetes_y_test = train_test_split(
    diabetes_X, diabetes_y, test_size = 0.2, random_state = 42)

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

#MSE and R2 on test set
MSE = mean_squared_error(diabetes_y_test, diabetes_y_pred)
RSQ = r2_score(diabetes_y_test, diabetes_y_pred)
```

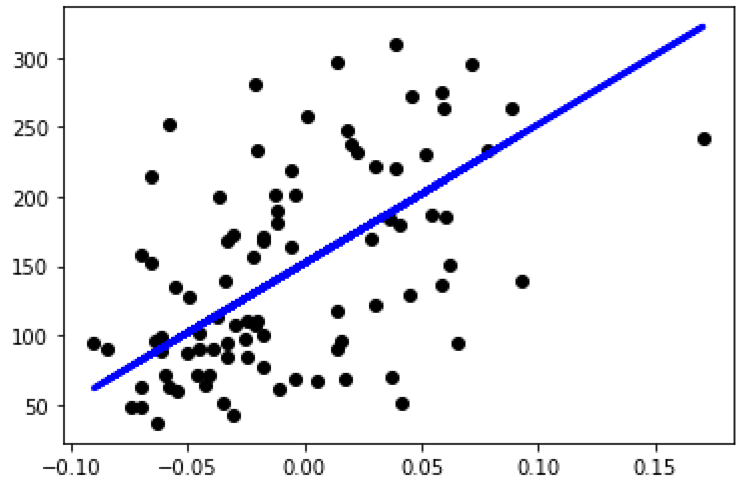
```
Coefficients:
 [998.57768914]
Mean squared error: 4061.83
Coefficient of determination: 0.23
```



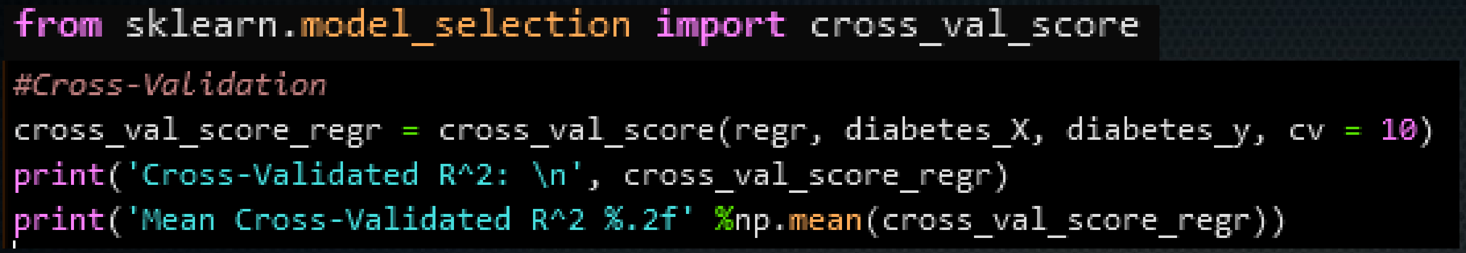
Scikit-learn: calculate metrics

```python
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

# Use only one feature
diabetes_X = diabetes_X[:, np.newaxis, 2]

#split data:
diabetes_X_train, diabetes_X_test, diabetes_y_train, diabetes_y_test = train_test_split(
    diabetes_X, diabetes_y, test_size = 0.2, random_state = 42)

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

#MSE and R2 on test set
MSE = mean_squared_error(diabetes_y_test, diabetes_y_pred)
RSQ = r2_score(diabetes_y_test, diabetes_y_pred)
```

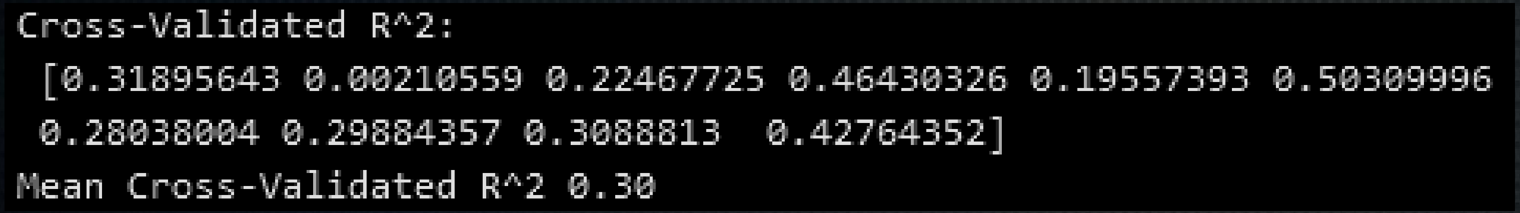
```
Coefficients:
 [998.57768914]
Mean squared error: 4061.83
Coefficient of determination: 0.23
```



Scikit-learn: cross-validation



```
from sklearn.model_selection import cross_val_score
#Cross-Validation
cross_val_score_regr = cross_val_score(regr, diabetes_X, diabetes_y, cv = 10)
print('Cross-Validated R^2: \n', cross_val_score_regr)
print('Mean Cross-Validated R^2 %.2f' %np.mean(cross_val_score_regr))
```



```
Cross-Validated R^2:
 [0.31895643 0.00210559 0.22467725 0.46430326 0.19557393 0.50309996
 0.28038004 0.29884357 0.3088813  0.42764352]
Mean Cross-Validated R^2 0.30
```

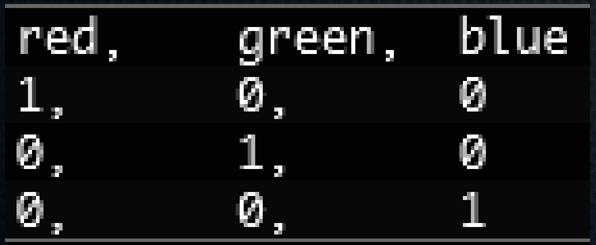
- Two lines of code → ten-fold cross-validated linear regression.
- If you want to plot the lines per cross-validation along with train and test data: can use sklearn.cross_validation.Kfold yourself and supply to cross_val score as argument cv.

Scikit-learn: preprocessing

- One-hot encoding:
  - Many ML algorithms work on continuous numbers and see 3 > 2.
  - If you have categorical variables (color = red | green | blue), you could encode them as 0, 1, 2. But that would make blue ‚More coloury‘ than green, which is ‚more coloury‘ than red. → wrong!

Scikit-learn: preprocessing

- One-hot encoding:
  - Many ML algorithms work on continuous numbers and see 3 > 2.
  - If you have categorical variables (color = red | green | blue), you could encode them as 0, 1, 2. But that would make blue ‚More coloury‘ than green, which is ‚more coloury‘ than red. Instead turn into 3 features:



```
red,      green,   blue
1,        0,       0
0,        1,       0
0,        0,       1
```

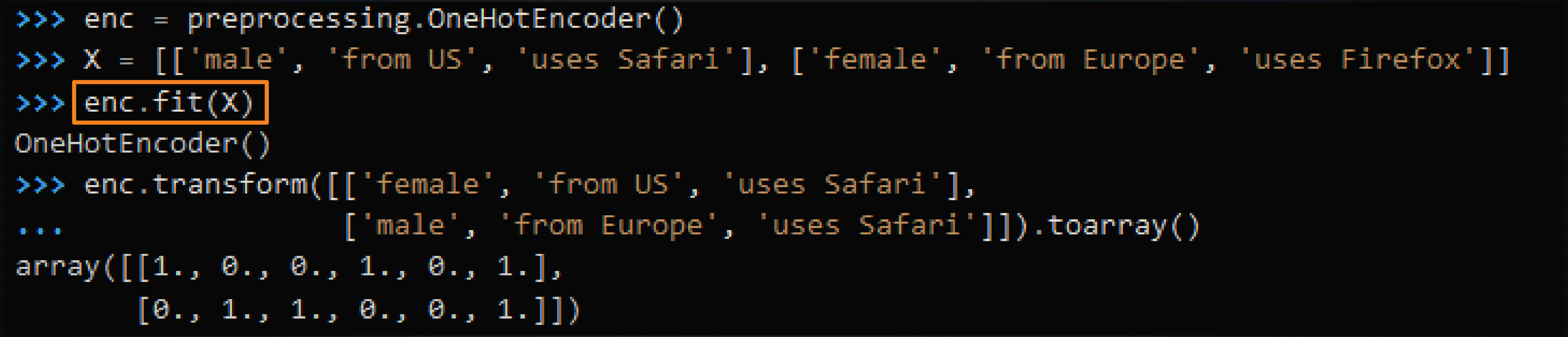
Scikit-learn: preprocessing

- One-hot encoding:

```
>>> enc = preprocessing.OneHotEncoder()
>>> X = [['male', 'from US', 'uses Safari'], ['female', 'from Europe', 'uses Firefox']]
>>> enc.fit(X)
OneHotEncoder()
>>> enc.transform([['female', 'from US', 'uses Safari'],
...                ['male', 'from Europe', 'uses Safari']]).toarray()
array([[1., 0., 0., 1., 0., 1.],
       [0., 1., 1., 0., 0., 1.]])
```

Scikit-learn: preprocessing

- One-hot encoding:

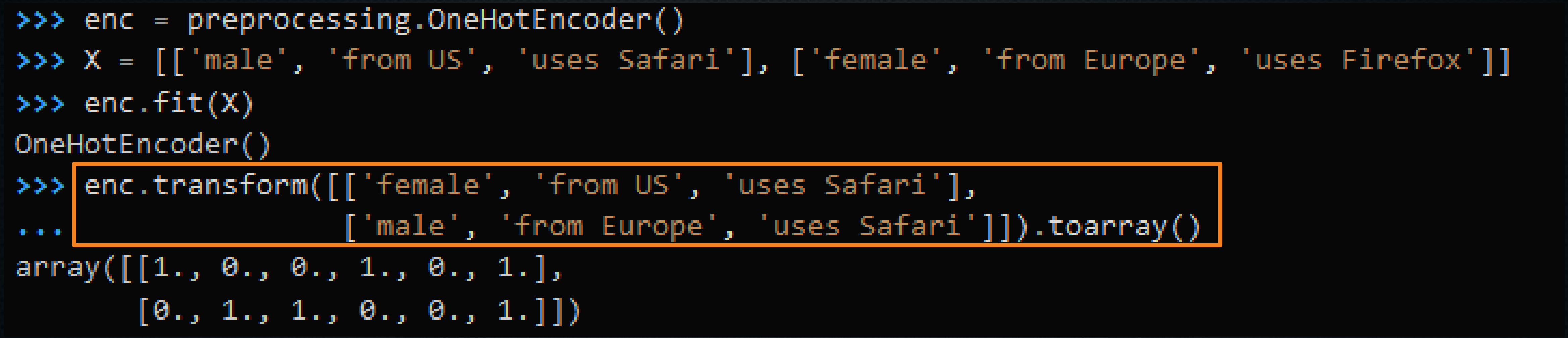


```
>>> enc = preprocessing.OneHotEncoder()
>>> X = [['male', 'from US', 'uses Safari'], ['female', 'from Europe', 'uses Firefox']]
>>> enc.fit(X)
OneHotEncoder()
>>> enc.transform([['female', 'from US', 'uses Safari'],
...                ['male', 'from Europe', 'uses Safari']]).toarray()
array([[1., 0., 0., 1., 0., 1.],
       [0., 1., 1., 0., 0., 1.]])
```

- Same .fit() method as estimators

Scikit-learn: preprocessing

- One-hot encoding:



```
>>> enc = preprocessing.OneHotEncoder()
>>> X = [['male', 'from US', 'uses Safari'], ['female', 'from Europe', 'uses Firefox']]
>>> enc.fit(X)
OneHotEncoder()
>>> enc.transform([['female', 'from US', 'uses Safari'],
...                ['male', 'from Europe', 'uses Safari']]).toarray()
array([[1., 0., 0., 1., 0., 1.],
       [0., 1., 1., 0., 0., 1.]])
```

- .transform() is similar to .predict() for estimators.

Scikit-learn: preprocessing

- One-hot encoding:

```
>>> enc = preprocessing.OneHotEncoder()
>>> X = [['male', 'from US', 'uses Safari'], ['female', 'from Europe', 'uses Firefox']]
>>> enc.fit(X)
OneHotEncoder()
>>> enc.transform([['female', 'from US', 'uses Safari'],
...                ['male', 'from Europe', 'uses Safari']]).toarray()
array([[1., 0., 0., 1., 0., 1.],
       [0., 1., 1., 0., 0., 1.]])
```

- Result is:

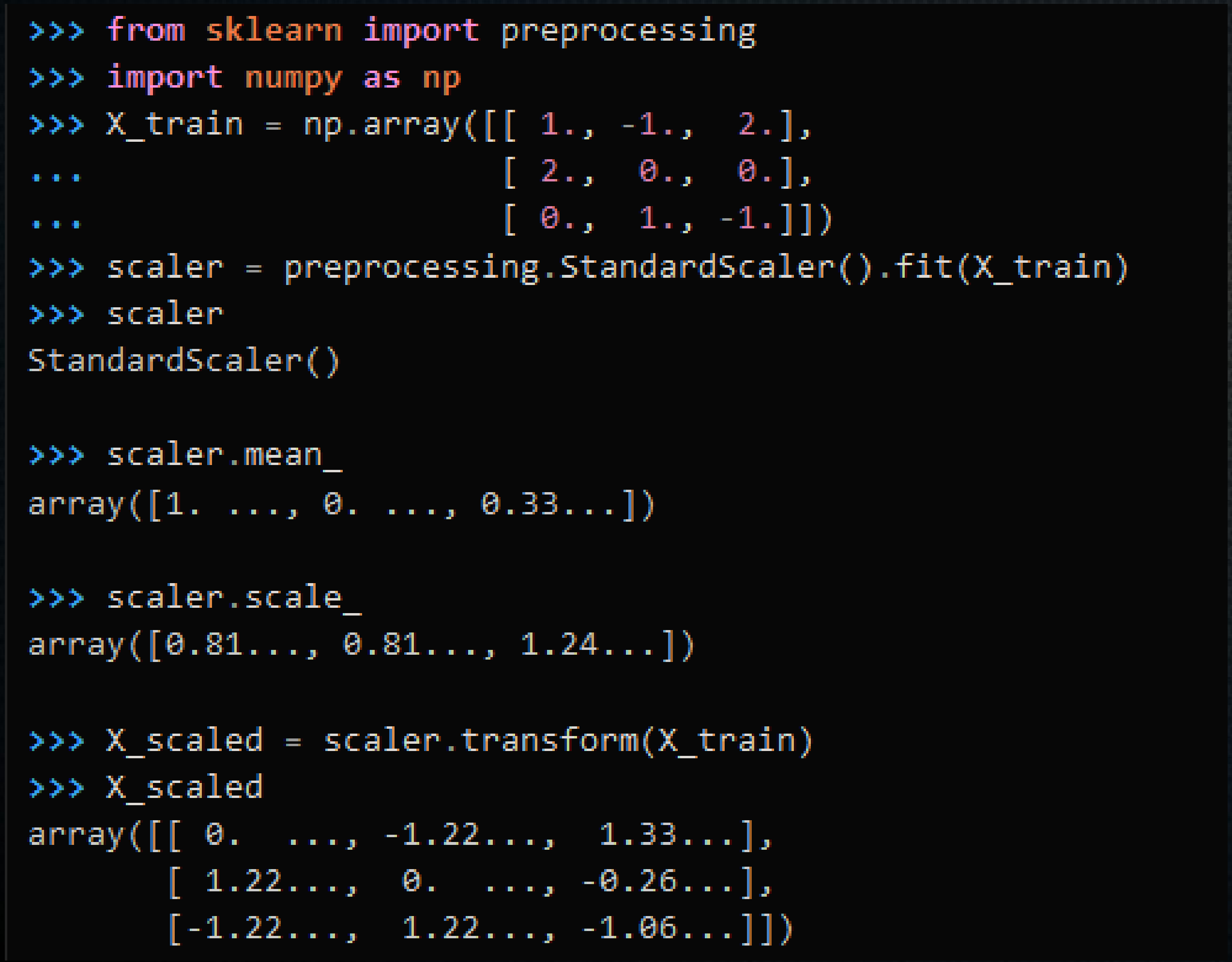
Sex	Geographic location	Browser
Female	US	Safari
Male	Europe	Safari

→

Female	Male	From Europe	From US	Uses Firefox	Uses Safari
1	0	0	1	0	1
0	1	1	0	0	1

Scikit-learn: preprocessing

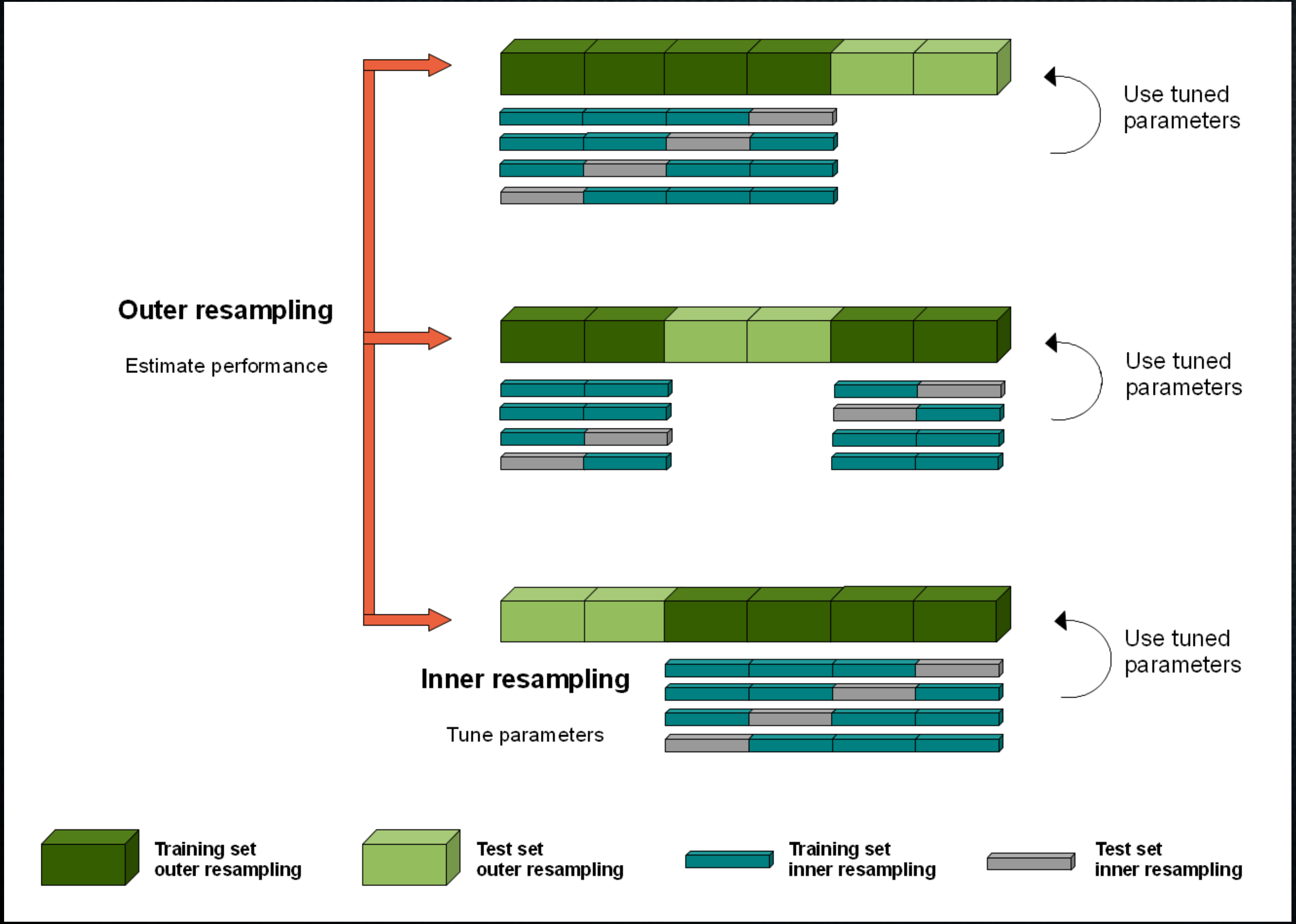
- Scaling:



```
>>> from sklearn import preprocessing
>>> import numpy as np
>>> X_train = np.array([[ 1., -1.,  2.],
...                     [ 2.,  0.,  0.],
...                     [ 0.,  1., -1.]])
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> scaler
StandardScaler()

>>> scaler.mean_
array([1. ..., 0. ..., 0.33...])

>>> scaler.scale_
array([0.81..., 0.81..., 1.24...])

>>> X_scaled = scaler.transform(X_train)
>>> X_scaled
array([[ 0.  ..., -1.22...,  1.33...],
       [ 1.22...,  0.  ..., -0.26...],
       [-1.22...,  1.22..., -1.06...]])
```

Scikit-learn: hyperparameter optimisation

- Gold standard is nested cross-validation



Source: https://mlr.mlr-org.com/articles/tutorial/nested_resampling.html

Scikit-learn: hyperparameter optimisation

```python
from sklearn.datasets import load_iris
from matplotlib import pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, cross_val_score, KFold
import numpy as np
```

```python
# Load the dataset
iris = load_iris()
X_iris = iris.data
y_iris = iris.target

# Set up possible values of parameters to optimize over
p_grid = {"C": [1, 10, 100],
          "gamma": [.01, .1]}

# We will use a Support Vector Classifier with "rbf" kernel
svm = SVC(kernel="rbf")

# Arrays to store scores
nested_scores = np.zeros(NUM_TRIALS)

# Loop 30 times
NUM_TRIALS = 30
for i in range(NUM_TRIALS):

    # Choose cross-validation techniques for the inner and outer loops,
    # independently of the dataset.
    # E.g "GroupKFold", "LeaveOneOut", "LeaveOneGroupOut", etc.
    inner_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    outer_cv = KFold(n_splits=4, shuffle=True, random_state=i)

    # Nested CV with parameter optimization
    clf = GridSearchCV(estimator=svm, param_grid=p_grid, cv=inner_cv)
    nested_score = cross_val_score(clf, X=X_iris, y=y_iris, cv=outer_cv)
    nested_scores[i] = nested_score.mean()

print(nested_scores)
```

Scikit-learn: hyperparameter optimisation

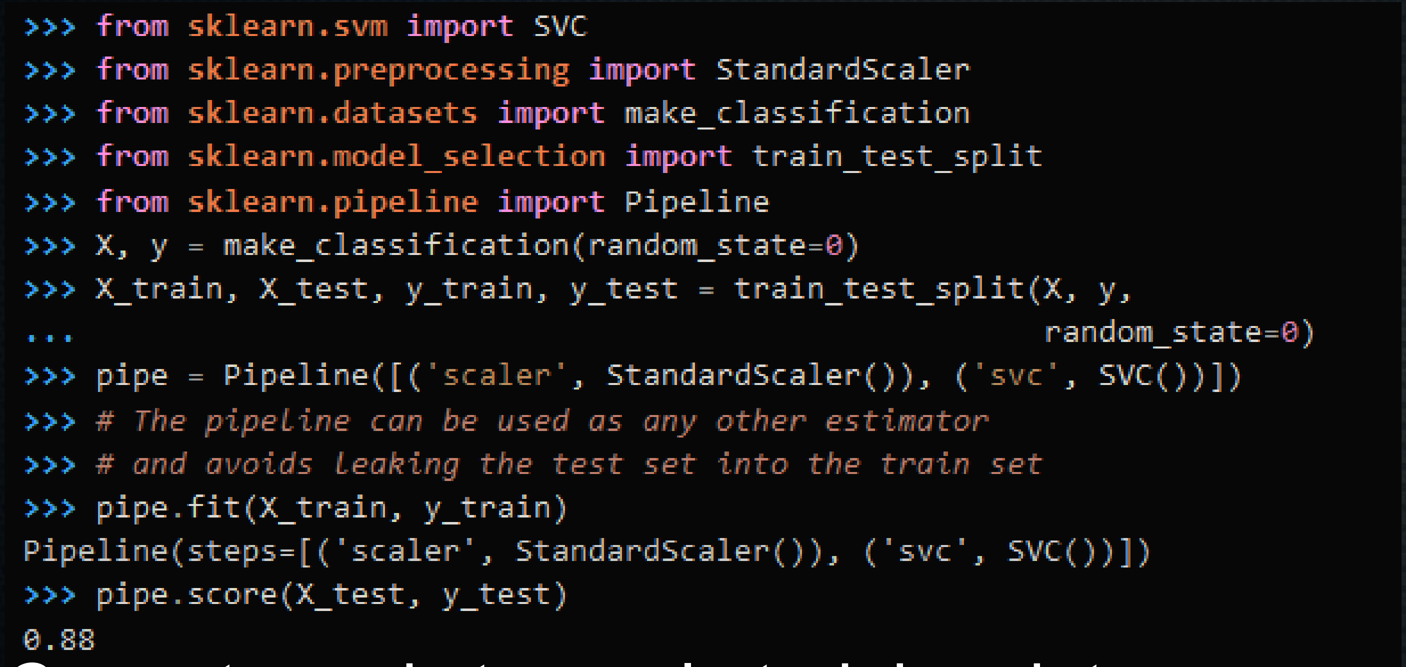
```python
from sklearn.datasets import load_iris
from matplotlib import pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, cross_val_score, KFold
import numpy as np
```

```python
# Load the dataset
iris = load_iris()
X_iris = iris.data
y_iris = iris.target

# Set up possible values of parameters to optimize over
p_grid = {"C": [1, 10, 100],
          "gamma": [.01, .1]}

# We will use a Support Vector Classifier with "rbf" kernel
svm = SVC(kernel="rbf")

# Arrays to store scores
nested_scores = np.zeros(NUM_TRIALS)

# Loop 30 times
NUM_TRIALS = 30
for i in range(NUM_TRIALS):

    # Choose cross-validation techniques for the inner and outer loops,
    # independently of the dataset.
    # E.g "GroupKFold", "LeaveOneOut", "LeaveOneGroupOut", etc.
    inner_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    outer_cv = KFold(n_splits=4, shuffle=True, random_state=i)

    # Nested CV with parameter optimization
    clf = GridSearchCV(estimator=svm, param_grid=p_grid, cv=inner_cv)
    nested_score = cross_val_score(clf, X=X_iris, y=y_iris, cv=outer_cv)
    nested_scores[i] = nested_score.mean()

print(nested_scores)
```

```
[0.94683499 0.94683499 0.97297297 0.9601707  0.95999289 0.96692745
 0.97332859 0.9601707  0.96034851 0.96639403 0.9601707  0.96674964
 0.95999289 0.97332859 0.96674964 0.96639403 0.9735064  0.95999289
 0.96639403 0.9735064  0.95963727 0.98684211 0.95359175 0.97332859
 0.97332859 0.95359175 0.95981508 0.98008535 0.98008535 0.98008535]
```

Scikit-learn: pipelines

- Combine preprocessing steps and classification/regression

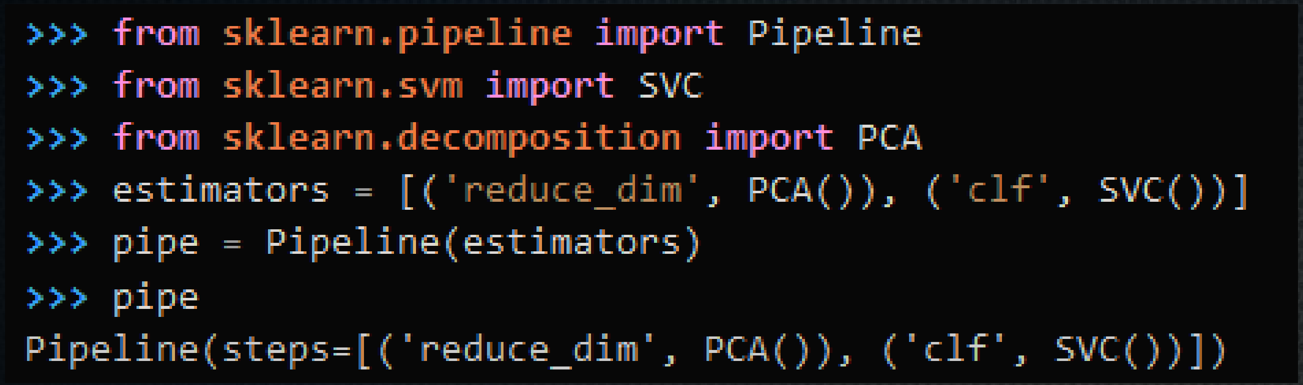


```
>>> from sklearn.svm import SVC
>>> from sklearn.preprocessing import StandardScaler
>>> from sklearn.datasets import make_classification
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.pipeline import Pipeline
>>> X, y = make_classification(random_state=0)
>>> X_train, X_test, y_train, y_test = train_test_split(X, y,
...                                                     random_state=0)
>>> pipe = Pipeline([('scaler', StandardScaler()), ('svc', SVC())])
>>> # The pipeline can be used as any other estimator
>>> # and avoids leaking the test set into the train set
>>> pipe.fit(X_train, y_train)
Pipeline(steps=[('scaler', StandardScaler()), ('svc', SVC())])
>>> pipe.score(X_test, y_test)
0.88
```

- Correct way is to: scale training data, save means and variances of training data, train classifier, subtract training data means and training data variances from test data, classify. → all done in simple step.

Scikit-learn: pipelines

- Combine preprocessing steps and classification/regression

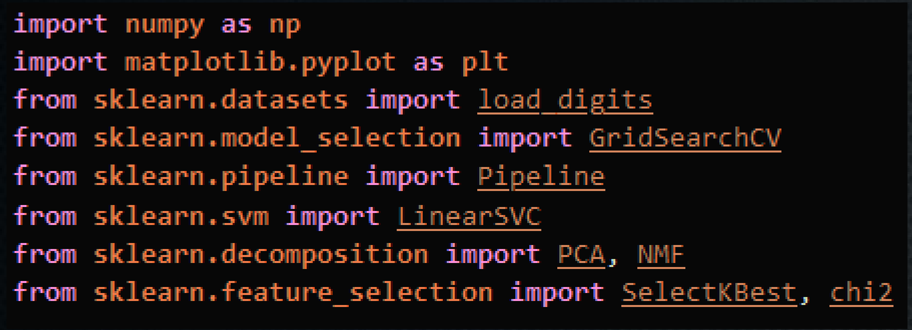


```
>>> from sklearn.pipeline import Pipeline
>>> from sklearn.svm import SVC
>>> from sklearn.decomposition import PCA
>>> estimators = [('reduce_dim', PCA()), ('clf', SVC())]
>>> pipe = Pipeline(estimators)
>>> pipe
Pipeline(steps=[('reduce_dim', PCA()), ('clf', SVC())])
```

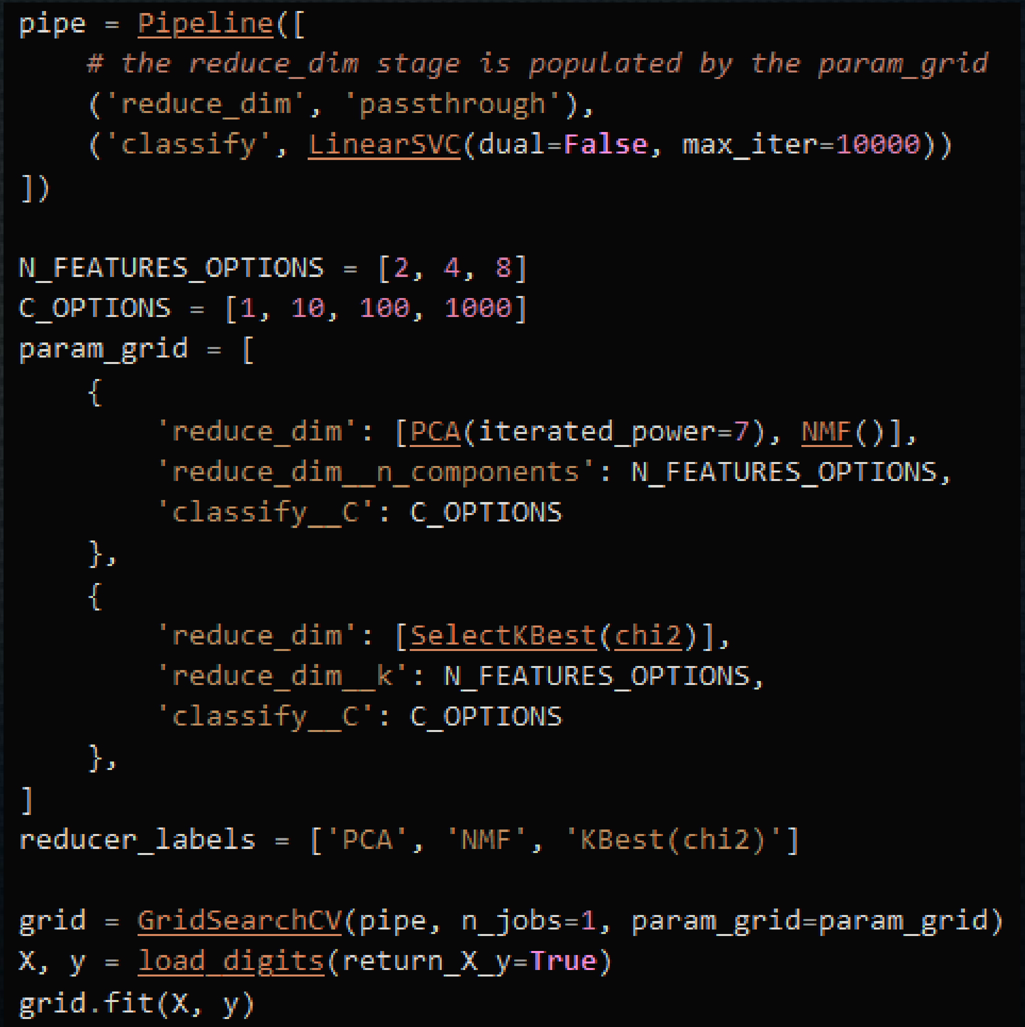
- Combine a PCA for dimensionality reduction with classification afterwards.

Scikit-learn: pipelines

- Easily cross-validate entire pipeline, compare different dim. reductions:



```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC
from sklearn.decomposition import PCA, NMF
from sklearn.feature_selection import SelectKBest, chi2
```



```python
pipe = Pipeline([
    # the reduce_dim stage is populated by the param_grid
    ('reduce_dim', 'passthrough'),
    ('classify', LinearSVC(dual=False, max_iter=10000))
])

N_FEATURES_OPTIONS = [2, 4, 8]
C_OPTIONS = [1, 10, 100, 1000]
param_grid = [
    {
        'reduce_dim': [PCA(iterated_power=7), NMF()],
        'reduce_dim__n_components': N_FEATURES_OPTIONS,
        'classify__C': C_OPTIONS
    },
    {
        'reduce_dim': [SelectKBest(chi2)],
        'reduce_dim__k': N_FEATURES_OPTIONS,
        'classify__C': C_OPTIONS
    },
]
reducer_labels = ['PCA', 'NMF', 'KBest(chi2)']

grid = GridSearchCV(pipe, n_jobs=1, param_grid=param_grid)
X, y = load_digits(return_X_y=True)
grid.fit(X, y)
```

Break for practical 1