Session 37

Evaluation metrics and Model selection

Table of Content

What will We Learn Today?

Model evaluation

- Metrics for Performance Evaluation
  - How to evaluate the performance of a model?
- Validation techniques
  - How to obtain reliable estimates?
- Model selection
  - How to compare the relative performance among competing models?

Performance metrics for Classification

Metrics for Performance Evaluation

- Focus on the predictive capability of a model
- Confusion Matrix:

	PREDICTED CLASS	
	Class=Yes (1)	Class=No (0)
ACTUAL CLASS Class=Yes (1)	a (TP)	b (FN)
ACTUAL CLASS Class=No (0)	c (FP)	d (TN)

a: TP (true positive)

b: FN (false negative)

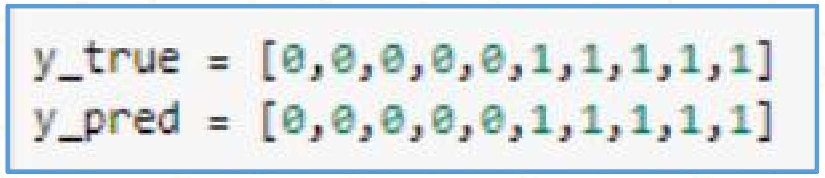
c: FP (false positive)

d: TN (true negative)

$$\text{Accuracy} = \frac{a+d}{a+b+c+d} = \frac{TP+TN}{TP+TN+FP+FN}$$

Example

- Example 1

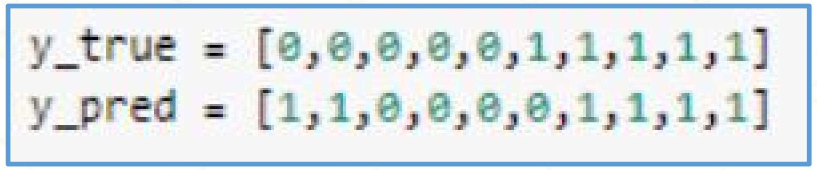


```
y_true = [0,0,0,0,0,1,1,1,1,1]
y_pred = [0,0,0,0,0,1,1,1,1,1]
```

		PREDICTED CLASS	
		Class=0	Class=1
ACTUAL CLASS	Class=0	5	0
	Class=1	0	5

  - Accuracy = (5+5) / (5+5+0+0) = 1

- Example 2



```
y_true = [0,0,0,0,0,1,1,1,1,1]
y_pred = [1,1,0,0,0,0,1,1,1,1]
```

		PREDICTED CLASS	
		Class=0	Class=1
ACTUAL CLASS	Class=0	3	2
	Class=1	1	4

  - Accuracy = (3 + 4) / (3 + 4 + 2 + 1) = 0.7

Limitation of accuracy

- Consider a 2-class problem
  - Number of Class 0 examples = 990
  - Number of Class 1 examples = 10
- If model predicts everything to be class 0, accuracy is 990/1000 = 99 %
  - Accuracy is misleading because model does not detect any class 1 example

		PREDICTED CLASS	
		Class=0	Class=1
ACTUAL CLASS	Class=0	990	0
	Class=1	10	0

TN : 990

FN : 10

TP : 0

FP : 0

Cost-sensitive measures

$$sensitivity\ (recall) = \frac{TP}{TP+FN}$$

$$precision = \frac{TP}{TP+FP}$$

$$specificity = \frac{TN}{TN+FP}$$

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

	PREDICTED CLASS		
ACTUAL CLASS		Class=Yes	Class=No
	Class=Yes	(TP)	(FN)
	Class=No	(FP)	(TN)

- True positive rate (TPR) = sensitivity or recall
- True negative rate (TNR) = specificity

- Recall = How good a model is at detecting the positives
- Precision = What proportion of positive identifications was actually correct?
- F1 score = conveys the balance between the precision and the recall

Example

- Example

```
y_true = [0,0,0,0,0,0,0,0,1,1]
y_pred = [1,1,0,0,0,0,0,0,0,1]
```

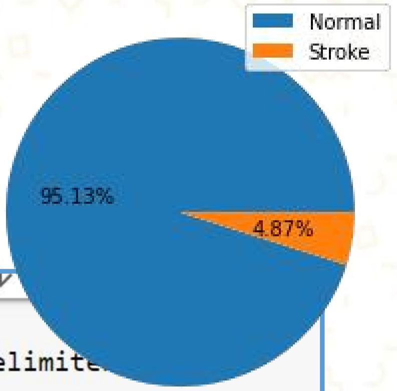
		PREDICTED CLASS	
ACTUAL CLASS		Class=0	Class=1
	Class=0	6	2
	Class=1	1	1

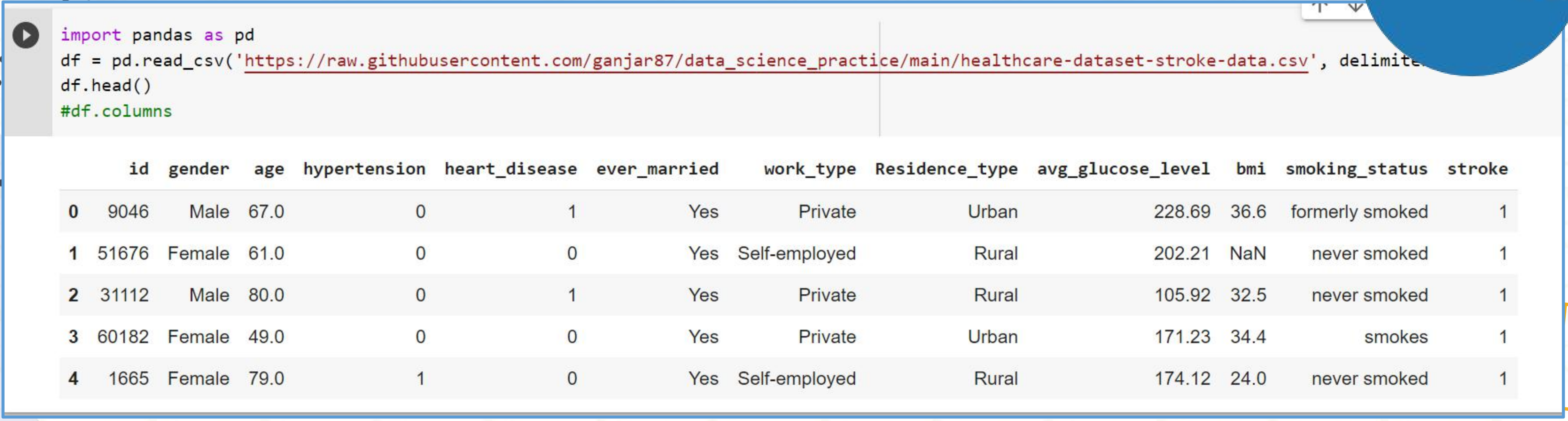
		PREDICTED CLASS	
ACTUAL CLASS		Class=0	Class=1
	Class=0	TN	FP
	Class=1	FN	TP

- Accuracy = (TP + TN)/(TP+TN+FP+FN) = (1+6)/(1+6+2+1) = 7/10 = 0.7
- Precision = TP / (TP+FP) = 1/(1+2) = 1/3 = 0.33
- Recall = TP / (TP + FN) = 1/(1+1) =1/2 = 0.5
- Specificity = TN/(TN+FP) = 6/(6+2) = 6/8 = 0.75
- F1 = 2 * (precision * recall) / (precision + recall) = 2 * (0.33*0.5) / (0.33 + 0.5) = 3.3 / 0.83 = 0.4

Read the dataset

https://www.kaggle.com/fedesoriano/stroke-prediction-dataset

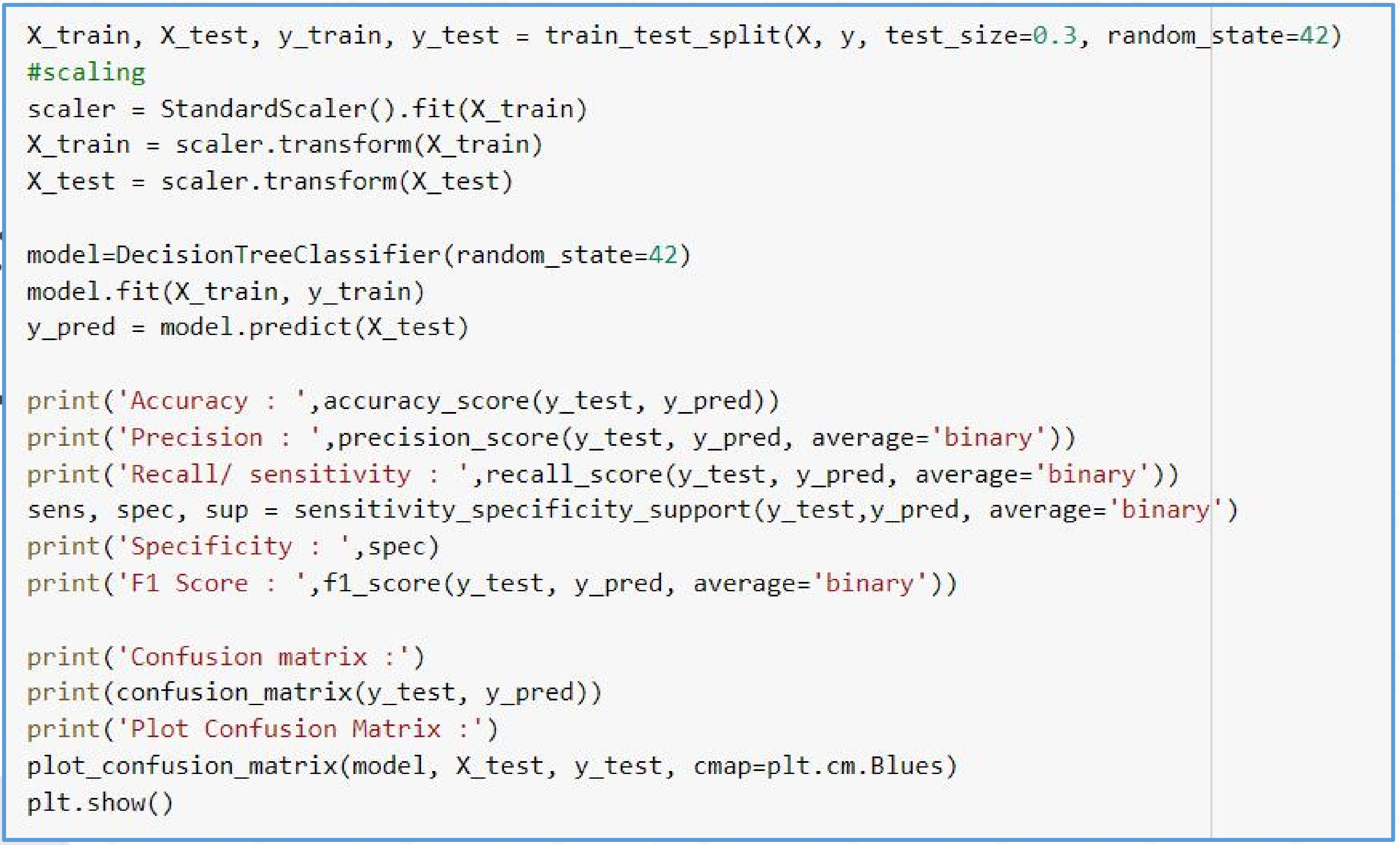




```
import pandas as pd
df = pd.read_csv('https://raw.githubusercontent.com/ganjar87/data_science_practice/main/healthcare-dataset-stroke-data.csv', delimite
df.head()
#df.columns
```

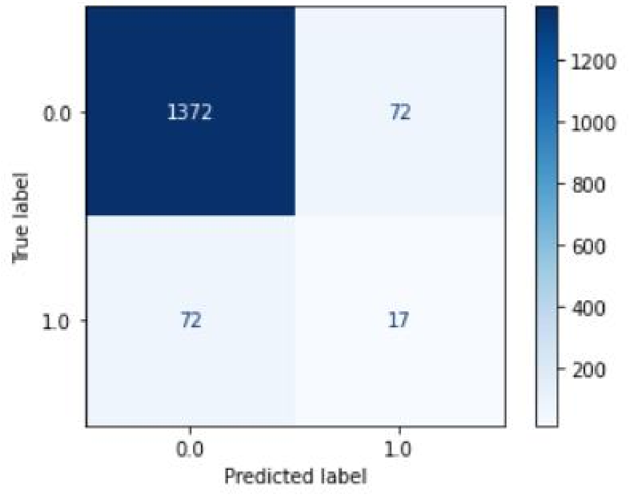
	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

Evaluation metrics



```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
#scaling
scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

model=DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print('Accuracy : ',accuracy_score(y_test, y_pred))
print('Precision : ',precision_score(y_test, y_pred, average='binary'))
print('Recall/ sensitivity : ',recall_score(y_test, y_pred, average='binary'))
sens, spec, sup = sensitivity_specificity_support(y_test,y_pred, average='binary')
print('Specificity : ',spec)
print('F1 Score : ',f1_score(y_test, y_pred, average='binary'))

print('Confusion matrix :')
print(confusion_matrix(y_test, y_pred))
print('Plot Confusion Matrix :')
plot_confusion_matrix(model, X_test, y_test, cmap=plt.cm.Blues)
plt.show()
```

```
Accuracy :  0.9060665362035225
Precision :  0.19101123595505617
Recall/ sensitivity :  0.19101123595505617
Specificity :  0.9501385041551247
F1 Score :  0.1910112359550562
Confusion matrix :
[[1372   72]
 [  72   17]]
Plot Confusion Matrix :
```

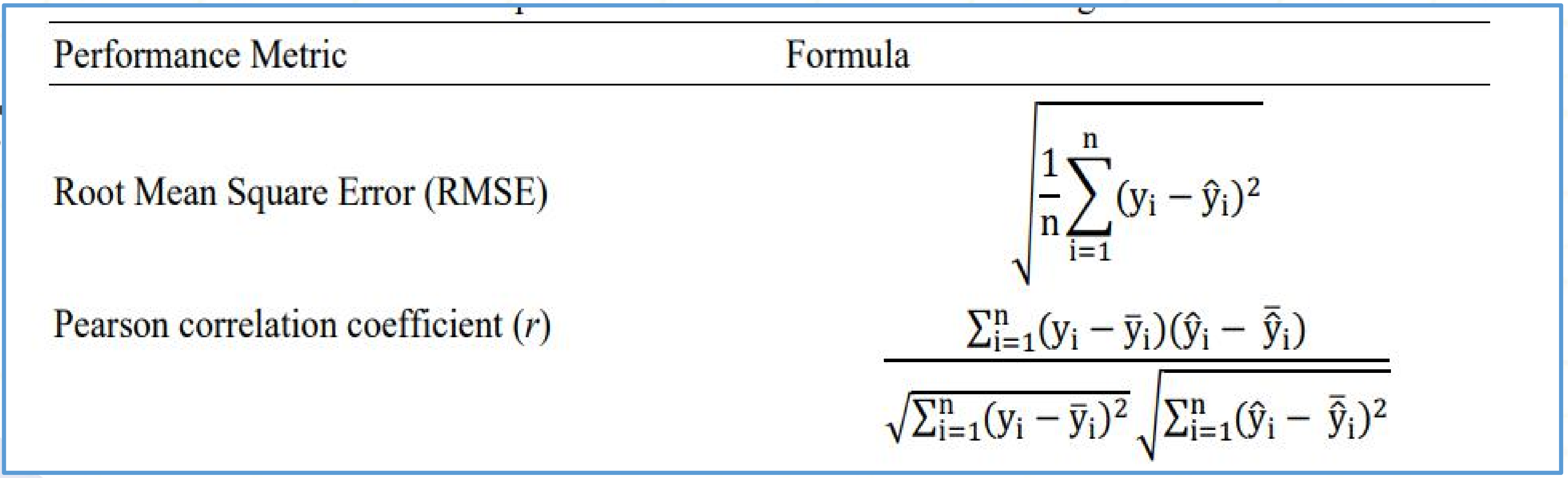


DigitalSkola
Uncover The World Of Digital Skills With Us

Performance metrics for Regression and Forecasting

Evaluation Metrics

- Pearson correlation coefficient (r) = measures the strength and the direction of a linear relationship between two variables. (-1 to 1)
- Coefficient determination (r2 or r square) = gives the proportion of the variance (fluctuation) of one variable that is predictable from the other variable. (0 to 1)
- Root mean square error (RMSE) = the standard deviation of the residuals (prediction errors)

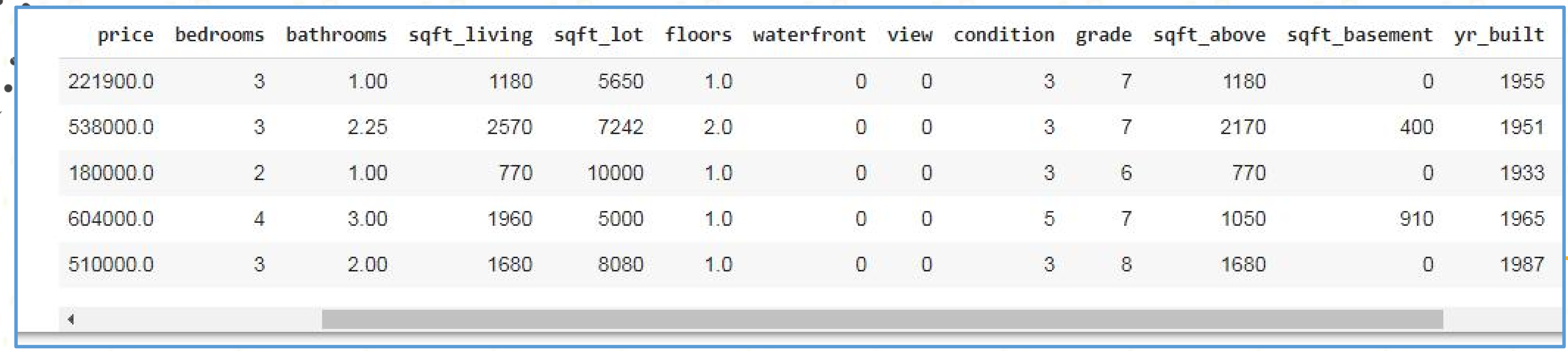


Performance Metric	Formula
Root Mean Square Error (RMSE)	$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$
Pearson correlation coefficient (*r*)	$\frac{\sum_{i=1}^{n}(y_i - \bar{y}_i)(\hat{y}_i - \bar{\hat{y}}_i)}{\sqrt{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}\sqrt{\sum_{i=1}^{n}(\hat{y}_i - \bar{\hat{y}}_i)^2}}$

MAE $\frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$

Example

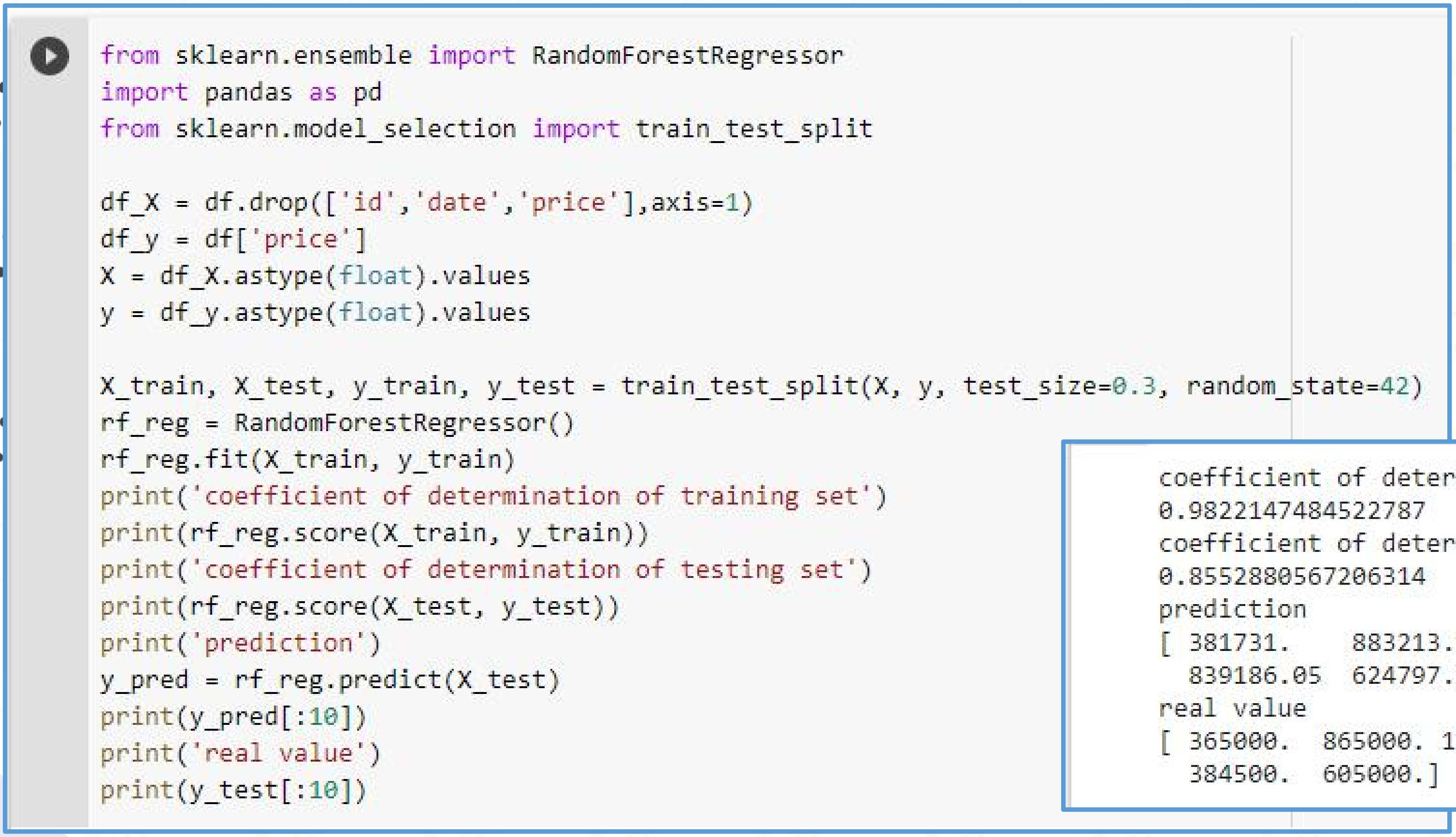
- House Sales in King County, USA.
- This dataset contains house sale prices for King County, which includes Seattle. It includes homes sold between May 2014 and May 2015.
- Source : https://www.kaggle.com/harlfoxem/housesalesprediction



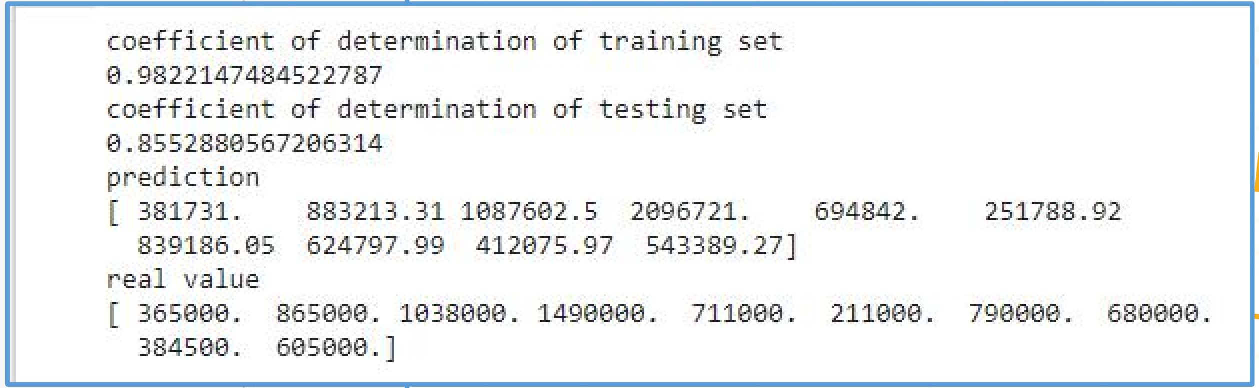
price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_built
221900.0	3	1.00	1180	5650	1.0	0	0	3	7	1180	0	1955
538000.0	3	2.25	2570	7242	2.0	0	0	3	7	2170	400	1951
180000.0	2	1.00	770	10000	1.0	0	0	3	6	770	0	1933
604000.0	4	3.00	1960	5000	1.0	0	0	5	7	1050	910	1965
510000.0	3	2.00	1680	8080	1.0	0	0	3	8	1680	0	1987

Random Forest Regression

- We use library from sklearn



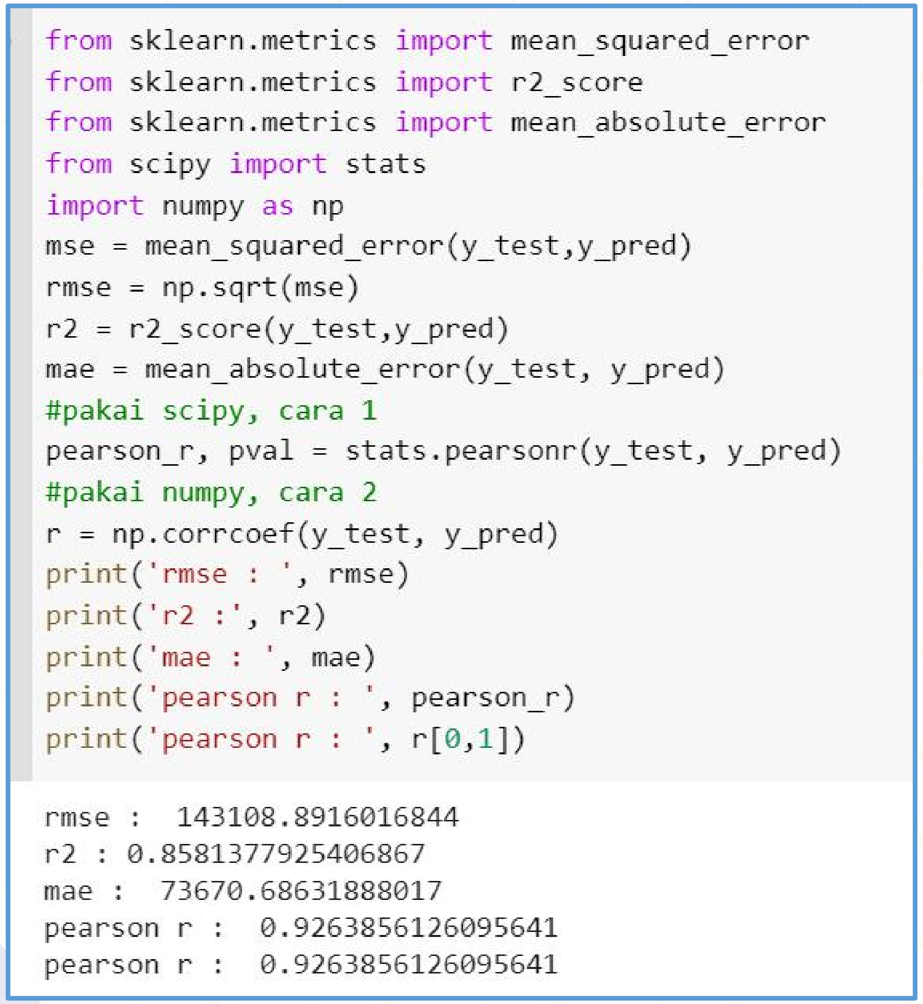
```
from sklearn.ensemble import RandomForestRegressor
import pandas as pd
from sklearn.model_selection import train_test_split

df_X = df.drop(['id','date','price'],axis=1)
df_y = df['price']
X = df_X.astype(float).values
y = df_y.astype(float).values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
rf_reg = RandomForestRegressor()
rf_reg.fit(X_train, y_train)
print('coefficient of determination of training set')
print(rf_reg.score(X_train, y_train))
print('coefficient of determination of testing set')
print(rf_reg.score(X_test, y_test))
print('prediction')
y_pred = rf_reg.predict(X_test)
print(y_pred[:10])
print('real value')
print(y_test[:10])
```



```
coefficient of determination of training set
0.9822147484522787
coefficient of determination of testing set
0.8552880567206314
prediction
[ 381731.    883213.31 1087602.5  2096721.     694842.     251788.92
  839186.05  624797.99  412075.97  543389.27]
real value
[ 365000.  865000. 1038000. 1490000.  711000.  211000.  790000.  680000.
  384500.  605000.]
```

Random Forest Regression

- Calculate model performance



```python
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from scipy import stats
import numpy as np
mse = mean_squared_error(y_test,y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test,y_pred)
mae = mean_absolute_error(y_test, y_pred)
#pakai scipy, cara 1
pearson_r, pval = stats.pearsonr(y_test, y_pred)
#pakai numpy, cara 2
r = np.corrcoef(y_test, y_pred)
print('rmse : ', rmse)
print('r2 :', r2)
print('mae : ', mae)
print('pearson r : ', pearson_r)
print('pearson r : ', r[0,1])
```

```
rmse :  143108.8916016844
r2 : 0.8581377925406867
mae :  73670.68631888017
pearson r :  0.9263856126095641
pearson r :  0.9263856126095641
```

DigitalSkola
Uncover The World Of Digital Skills With Us

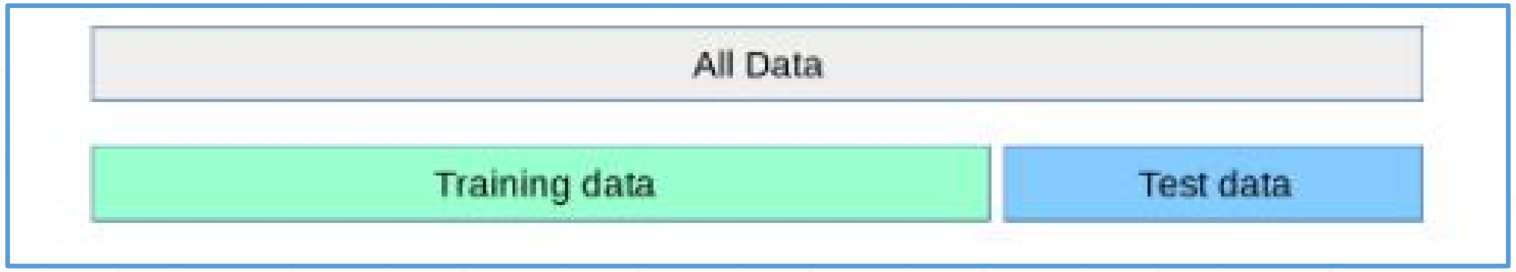
Validation techniques

Validation techniques

- How to obtain a reliable estimate of performance?
- Performance of a model may depend on other factors besides the learning algorithm:
  - Class distribution
  - Cost of misclassification
  - Size of training and test sets
- Techniques
  - Holdout
  - Cross validation

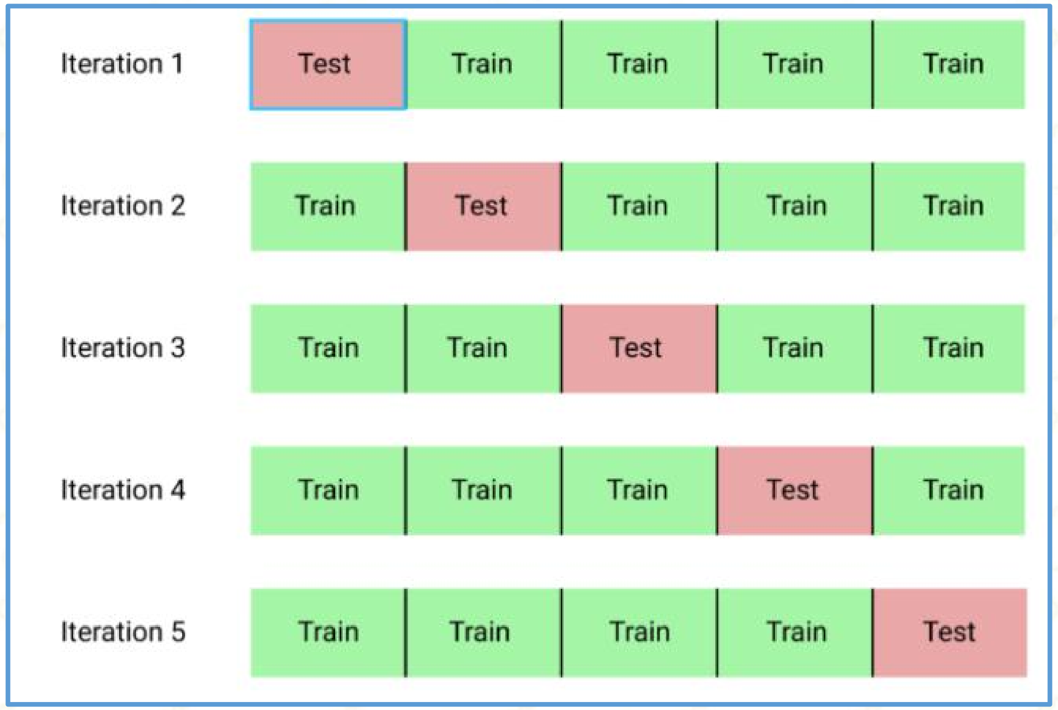
Holdout

- Holdout
  - The data is split into two different datasets as a training and a testing dataset.
  - This can be a 60/40 or 70/30 or 80/20 split.



Cross validation

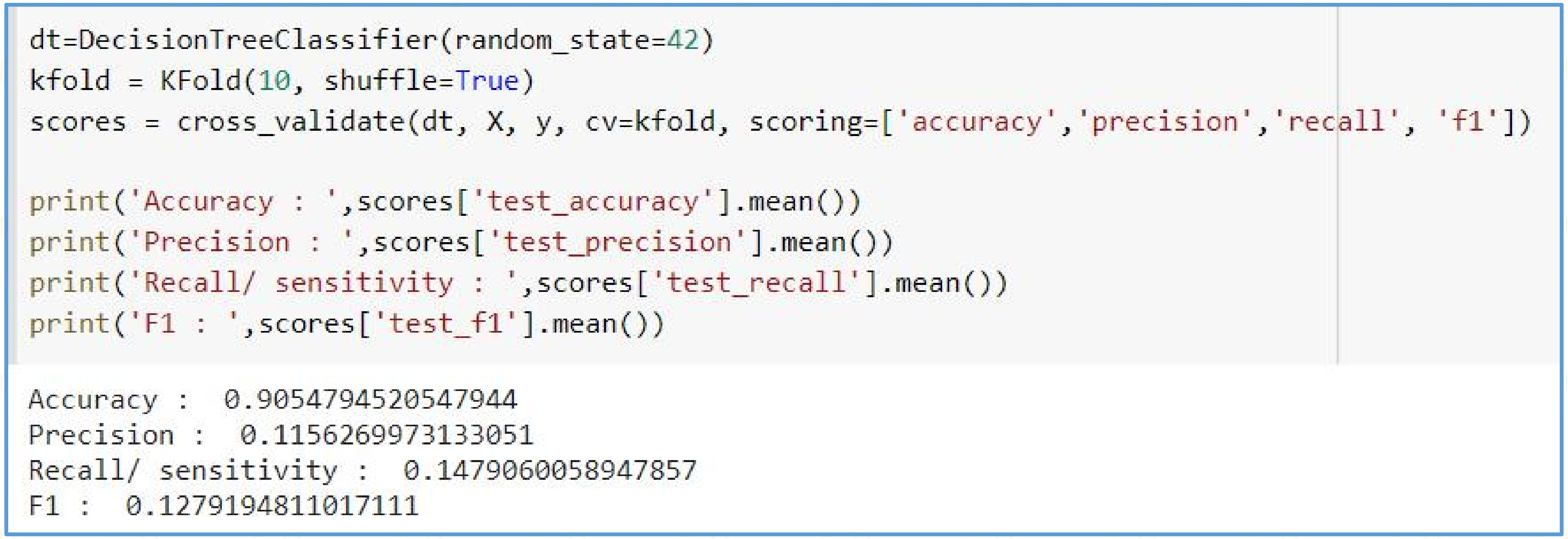
- Cross validation
  - Partition data into k disjoint subsets
  - k-fold: train on k-1 partitions, test on the remaining one



Iteration 1	Test	Train	Train	Train	Train
Iteration 2	Train	Test	Train	Train	Train
Iteration 3	Train	Train	Test	Train	Train
Iteration 4	Train	Train	Train	Test	Train
Iteration 5	Train	Train	Train	Train	Test

CV version 1

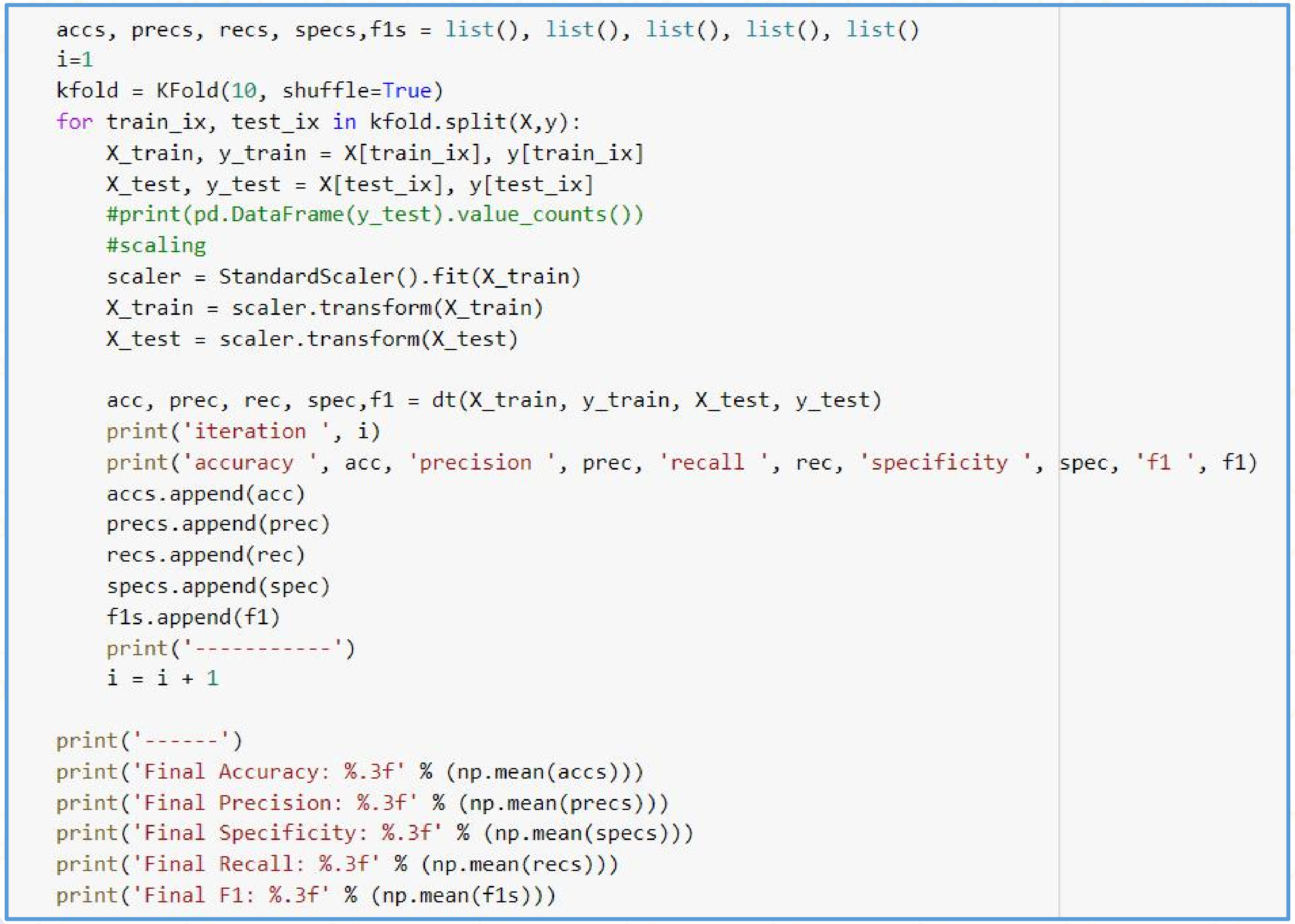
- Using *cross_validate*



```
dt=DecisionTreeClassifier(random_state=42)
kfold = KFold(10, shuffle=True)
scores = cross_validate(dt, X, y, cv=kfold, scoring=['accuracy','precision','recall', 'f1'])

print('Accuracy : ',scores['test_accuracy'].mean())
print('Precision : ',scores['test_precision'].mean())
print('Recall/ sensitivity : ',scores['test_recall'].mean())
print('F1 : ',scores['test_f1'].mean())
```

```
Accuracy :  0.9054794520547944
Precision :  0.1156269973133051
Recall/ sensitivity :  0.1479060058947857
F1 :  0.1279194811017111
```

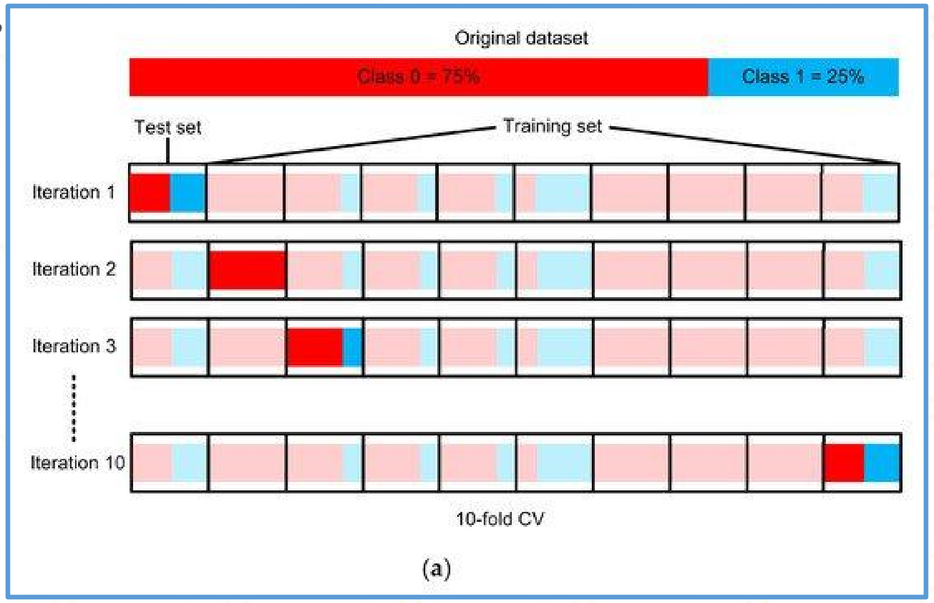
CV version 2



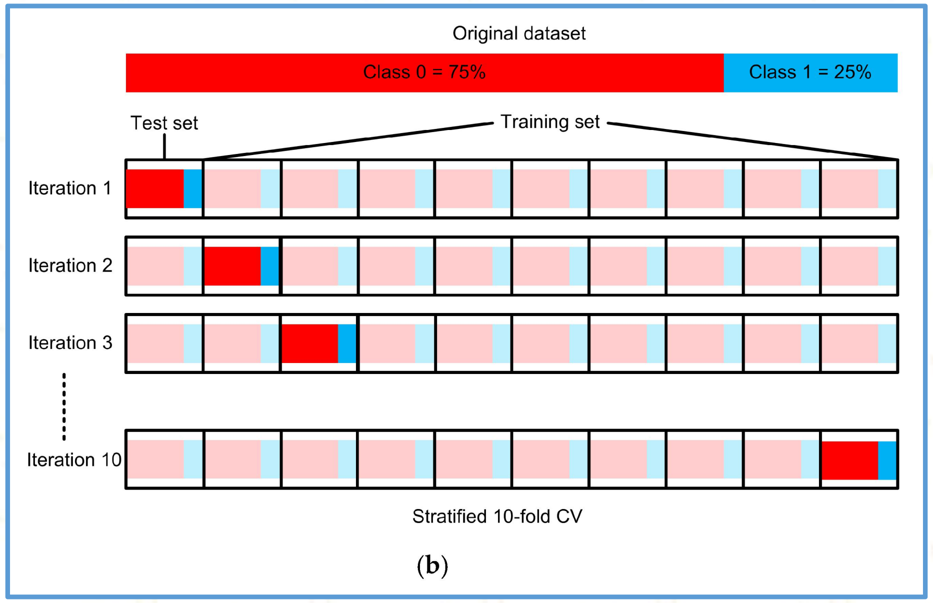
```
accs, precs, recs, specs,f1s = list(), list(), list(), list(), list()
i=1
kfold = KFold(10, shuffle=True)
for train_ix, test_ix in kfold.split(X,y):
    X_train, y_train = X[train_ix], y[train_ix]
    X_test, y_test = X[test_ix], y[test_ix]
    #print(pd.DataFrame(y_test).value_counts())
    #scaling
    scaler = StandardScaler().fit(X_train)
    X_train = scaler.transform(X_train)
    X_test = scaler.transform(X_test)

    acc, prec, rec, spec,f1 = dt(X_train, y_train, X_test, y_test)
    print('iteration ', i)
    print('accuracy ', acc, 'precision ', prec, 'recall ', rec, 'specificity ', spec, 'f1 ', f1)
    accs.append(acc)
    precs.append(prec)
    recs.append(rec)
    specs.append(spec)
    f1s.append(f1)
    print('-----------')
    i = i + 1

print('------')
print('Final Accuracy: %.3f' % (np.mean(accs)))
print('Final Precision: %.3f' % (np.mean(precs)))
print('Final Specificity: %.3f' % (np.mean(specs)))
print('Final Recall: %.3f' % (np.mean(recs)))
print('Final F1: %.3f' % (np.mean(f1s)))
```

Stratified cross validation

- Stratified cross validation
  - Partition data into k disjoint subsets
  - In stratified k-fold cross-validation, each subset is stratified so that they contain approximately the same proportion of class labels as the original dataset.
  - k-fold: train on k-1 partitions, test on the remaining one

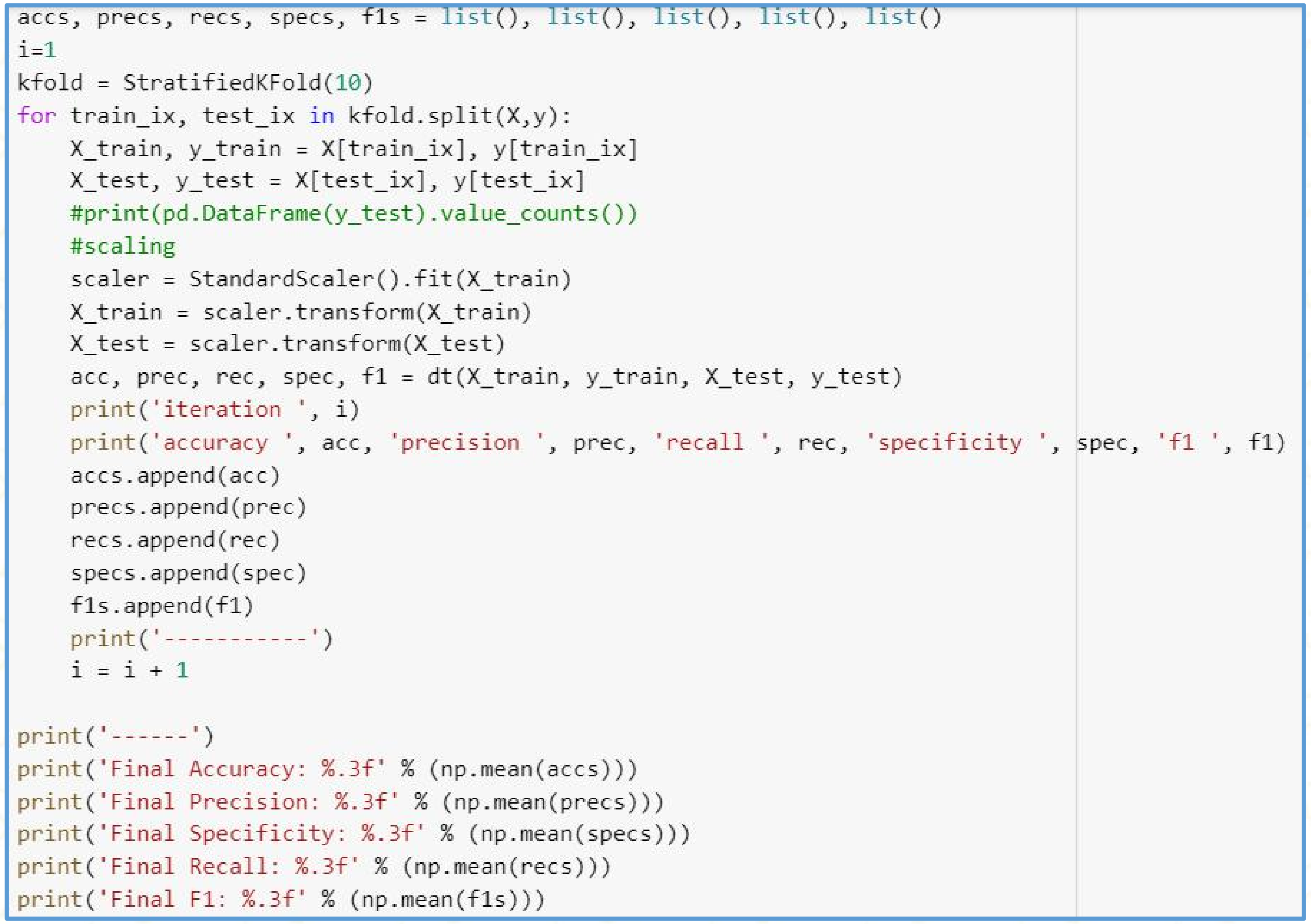


(a)



(b)

Stratified CV



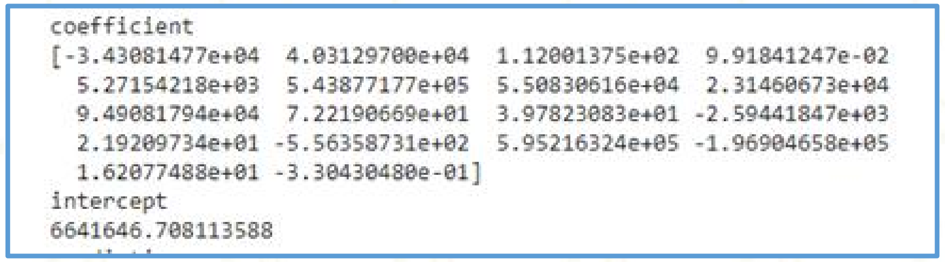
```
accs, precs, recs, specs, f1s = list(), list(), list(), list(), list()
i=1
kfold = StratifiedKFold(10)
for train_ix, test_ix in kfold.split(X,y):
    X_train, y_train = X[train_ix], y[train_ix]
    X_test, y_test = X[test_ix], y[test_ix]
    #print(pd.DataFrame(y_test).value_counts())
    #scaling
    scaler = StandardScaler().fit(X_train)
    X_train = scaler.transform(X_train)
    X_test = scaler.transform(X_test)
    acc, prec, rec, spec, f1 = dt(X_train, y_train, X_test, y_test)
    print('iteration ', i)
    print('accuracy ', acc, 'precision ', prec, 'recall ', rec, 'specificity ', spec, 'f1 ', f1)
    accs.append(acc)
    precs.append(prec)
    recs.append(rec)
    specs.append(spec)
    f1s.append(f1)
    print('-----------')
    i = i + 1

print('------')
print('Final Accuracy: %.3f' % (np.mean(accs)))
print('Final Precision: %.3f' % (np.mean(precs)))
print('Final Specificity: %.3f' % (np.mean(specs)))
print('Final Recall: %.3f' % (np.mean(recs)))
print('Final F1: %.3f' % (np.mean(f1s)))
```

Hyperparameter tuning

Hyperparameter tuning

- Hyperparameter tuning is choosing a set of optimal hyperparameters for a learning algorithm.

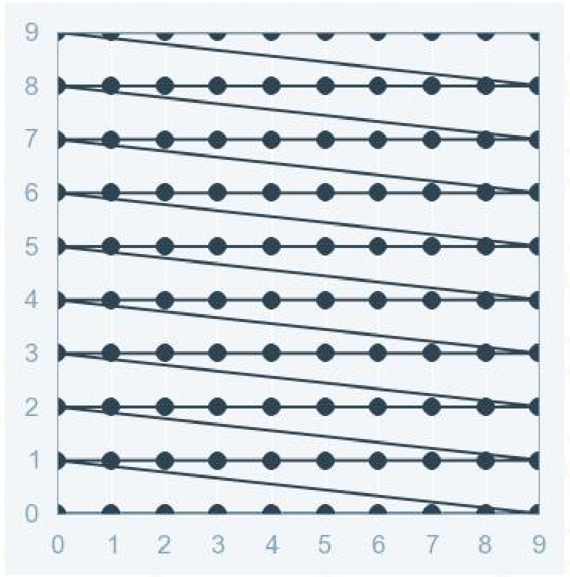
- Hyperparameters = part of the input that we supply to the ML algorithm
- Parameters = found out by training the model using trainset

Hyperparameter	Parameters
NearestNeighbors(n_neighbors=5)	Linear regression
KMeans(n_clusters=2)	
DecisionTreeClassifier(max_depth=5)	
RandomForestClassifier(max_features=4)	

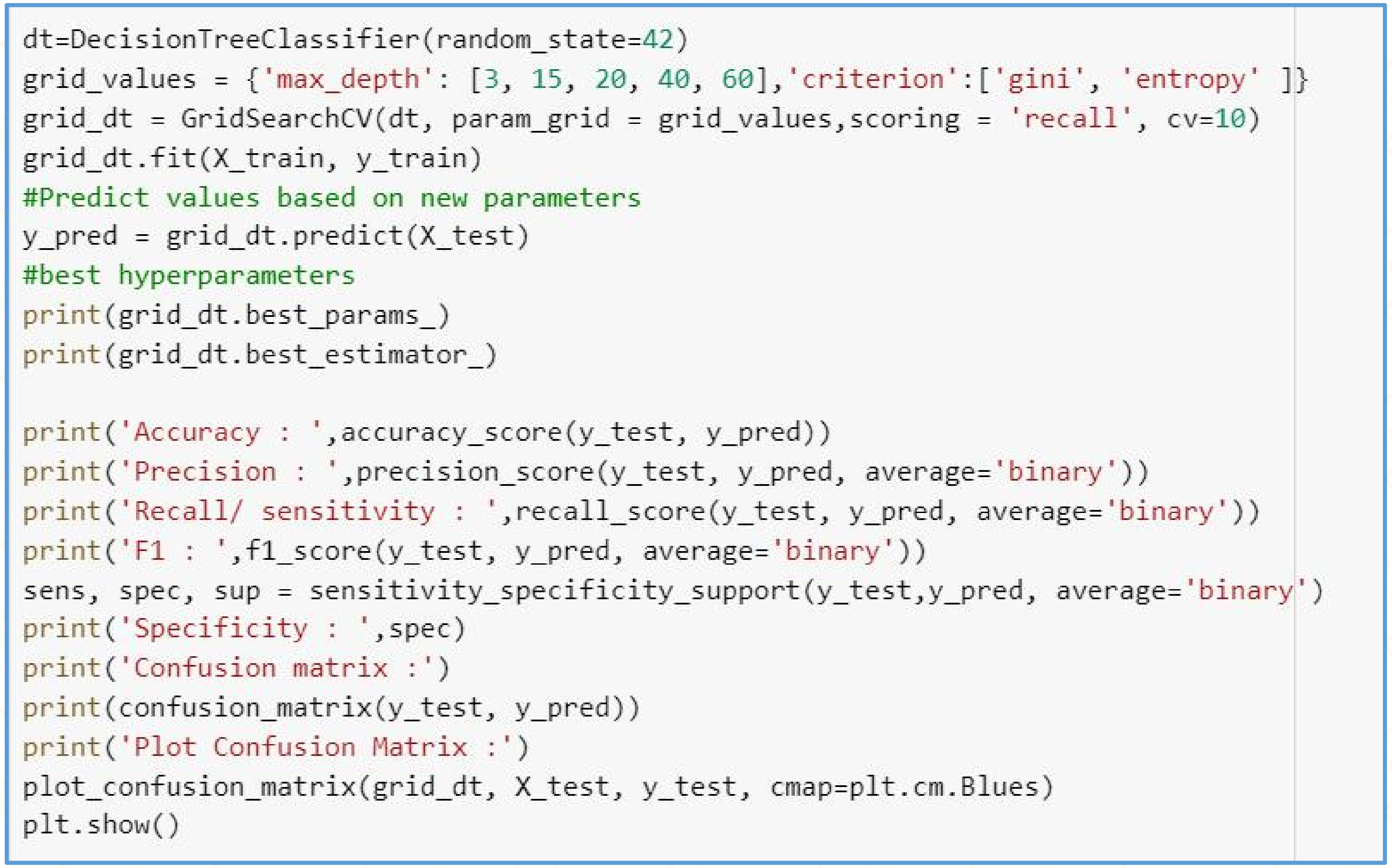
```
coefficient
[-3.43081477e+04  4.03129700e+04  1.12001375e+02  9.91841247e-02
  5.27154218e+03  5.43877177e+05  5.50830616e+04  2.31460673e+04
  9.49081794e+04  7.22190669e+01  3.97823083e+01 -2.59441847e+03
  2.19209734e+01 -5.56358731e+02  5.95216324e+05 -1.96904658e+05
  1.62077488e+01 -3.30430480e-01]
intercept
6641646.708113588
```

Grid Search

- Exhaustive search over specified hyperparameter values for an estimator.
- Steps
  - sets up a grid of hyperparameter values and for each combination,
  - trains a model and scores on the testing data.
- In this approach, every combination of hyperparameter values is tried which can be very inefficient.



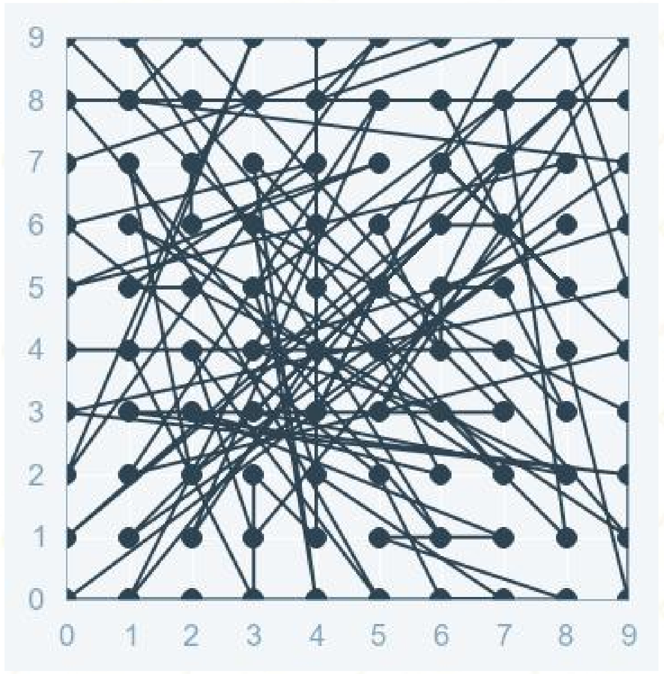
Grid Search



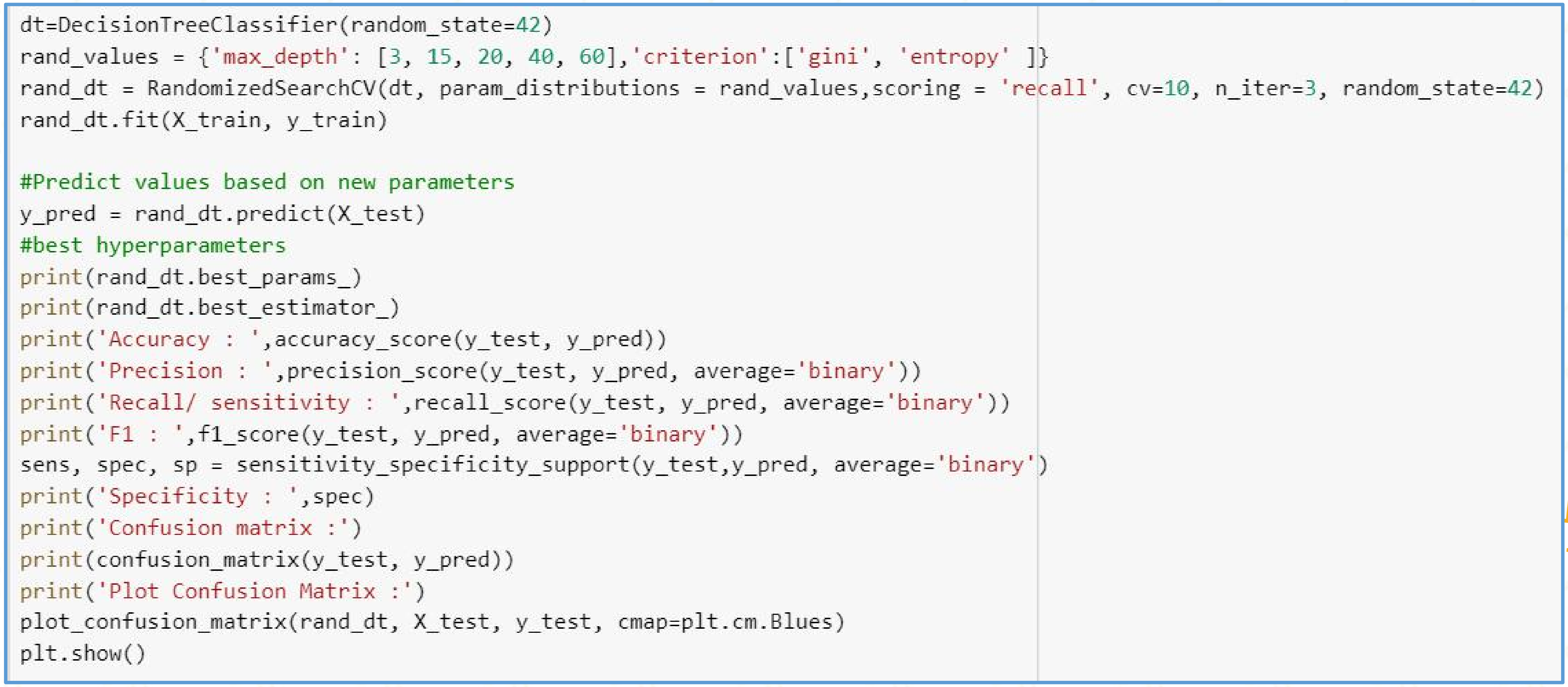
```
dt=DecisionTreeClassifier(random_state=42)
grid_values = {'max_depth': [3, 15, 20, 40, 60],'criterion':['gini', 'entropy' ]}
grid_dt = GridSearchCV(dt, param_grid = grid_values,scoring = 'recall', cv=10)
grid_dt.fit(X_train, y_train)
#Predict values based on new parameters
y_pred = grid_dt.predict(X_test)
#best hyperparameters
print(grid_dt.best_params_)
print(grid_dt.best_estimator_)

print('Accuracy : ',accuracy_score(y_test, y_pred))
print('Precision : ',precision_score(y_test, y_pred, average='binary'))
print('Recall/ sensitivity : ',recall_score(y_test, y_pred, average='binary'))
print('F1 : ',f1_score(y_test, y_pred, average='binary'))
sens, spec, sup = sensitivity_specificity_support(y_test,y_pred, average='binary')
print('Specificity : ',spec)
print('Confusion matrix :')
print(confusion_matrix(y_test, y_pred))
print('Plot Confusion Matrix :')
plot_confusion_matrix(grid_dt, X_test, y_test, cmap=plt.cm.Blues)
plt.show()
```

Randomized Search

- Random search is a method in which random combinations of hyperparameters are selected and used to train a model.
- The best random hyperparameter combinations are used.



Randomized Search



```
dt=DecisionTreeClassifier(random_state=42)
rand_values = {'max_depth': [3, 15, 20, 40, 60],'criterion':['gini', 'entropy' ]}
rand_dt = RandomizedSearchCV(dt, param_distributions = rand_values,scoring = 'recall', cv=10, n_iter=3, random_state=42)
rand_dt.fit(X_train, y_train)

#Predict values based on new parameters
y_pred = rand_dt.predict(X_test)
#best hyperparameters
print(rand_dt.best_params_)
print(rand_dt.best_estimator_)
print('Accuracy : ',accuracy_score(y_test, y_pred))
print('Precision : ',precision_score(y_test, y_pred, average='binary'))
print('Recall/ sensitivity : ',recall_score(y_test, y_pred, average='binary'))
print('F1 : ',f1_score(y_test, y_pred, average='binary'))
sens, spec, sp = sensitivity_specificity_support(y_test,y_pred, average='binary')
print('Specificity : ',spec)
print('Confusion matrix :')
print(confusion_matrix(y_test, y_pred))
print('Plot Confusion Matrix :')
plot_confusion_matrix(rand_dt, X_test, y_test, cmap=plt.cm.Blues)
plt.show()
```

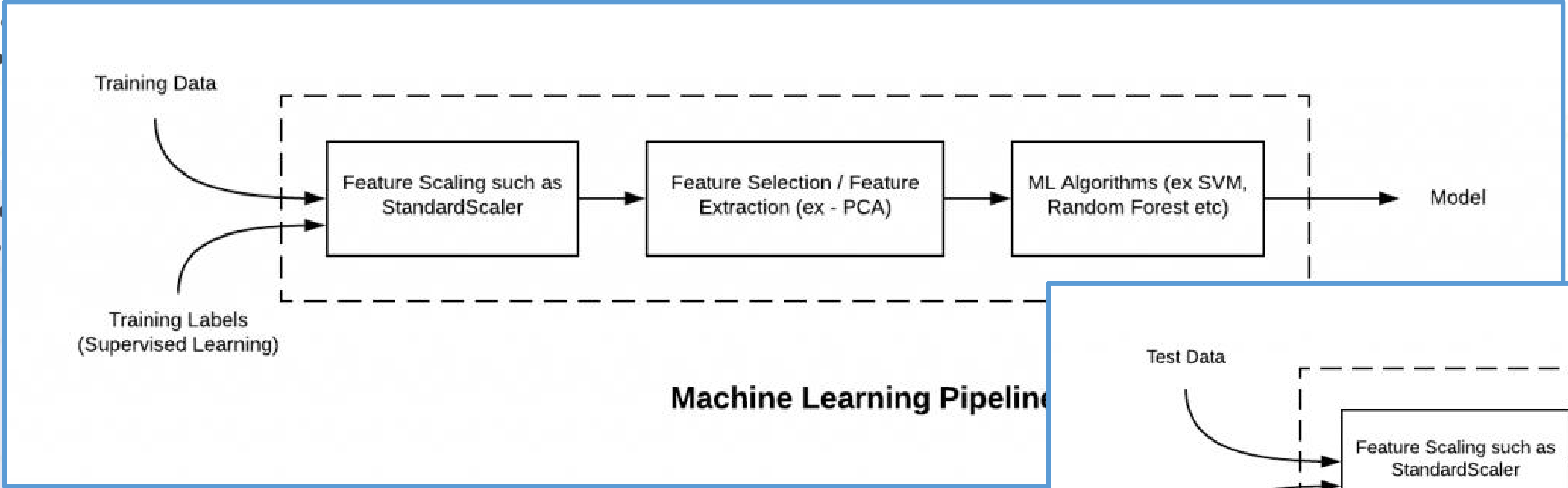
Model selection

Model selection

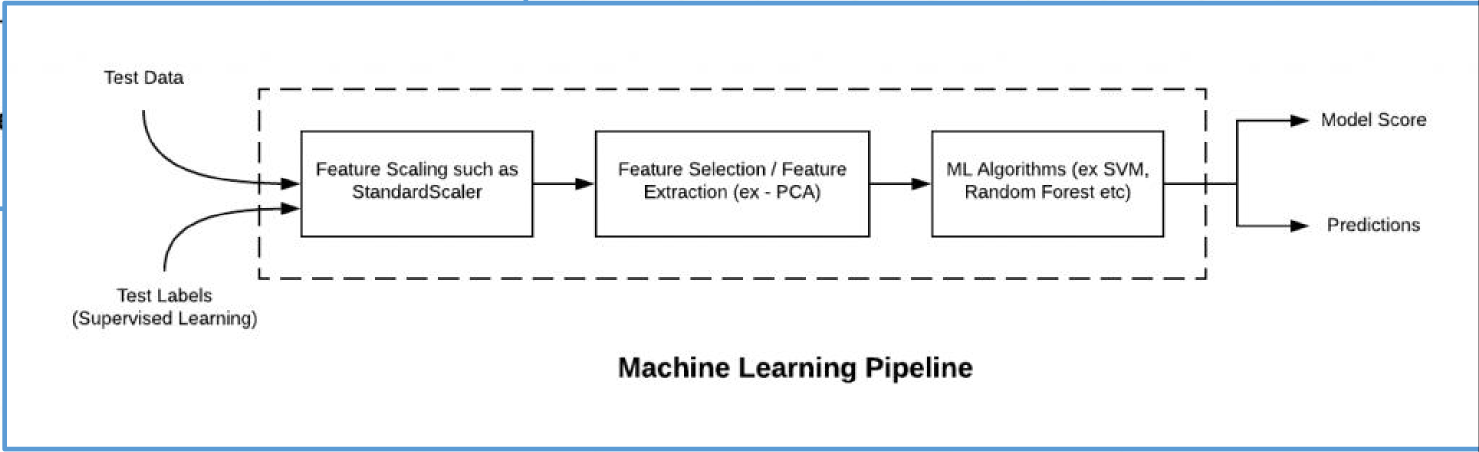
- Model selection refers to choose the best statistical machine learning model for a particular problem.
- For this task we need to compare the relative performance between models.

ML Pipeline

- *Machine Learning (ML) pipeline* = cara meng-otomatisasi alur kerja yang diperlukan untuk menghasilkan model *machine learning.*
- Sebelumnya kita men-transformasi data untuk training dan testing set secara terpisah.
- Dengan menggunakan *Sklearn.pipeline* kita dapat meng-otomatiskan langkah-langkah ini.



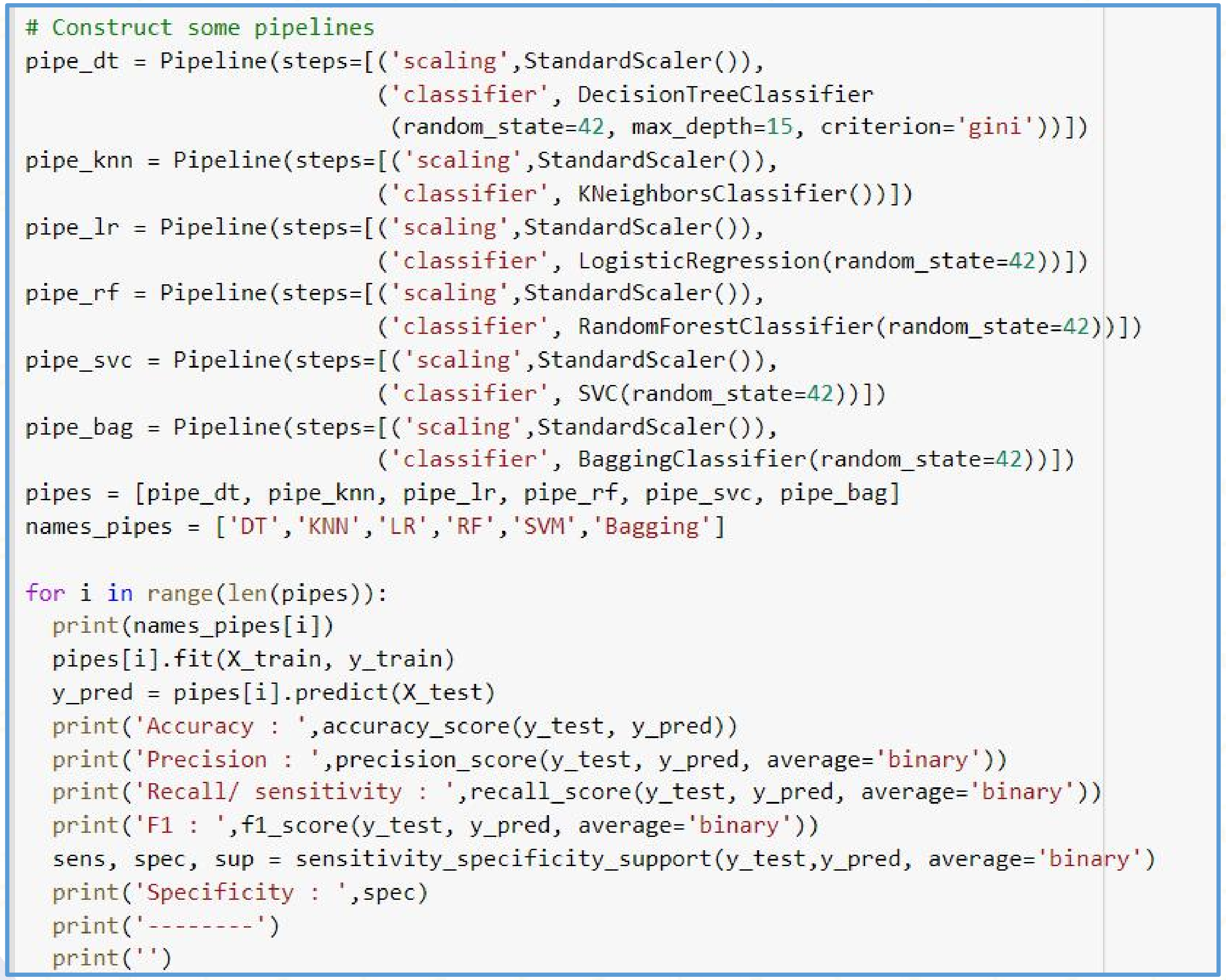
Training Data
Feature Scaling such as StandardScaler
Feature Selection / Feature Extraction (ex - PCA)
ML Algorithms (ex SVM, Random Forest etc)
Model
Training Labels (Supervised Learning)
Machine Learning Pipeline



Test Data
Feature Scaling such as StandardScaler
Feature Selection / Feature Extraction (ex - PCA)
ML Algorithms (ex SVM, Random Forest etc)
Model Score
Predictions
Test Labels (Supervised Learning)
Machine Learning Pipeline

https://vitalflux.com/sklearn-machine-learning-pipeline-python-example/

Model selection



```
# Construct some pipelines
pipe_dt = Pipeline(steps=[('scaling',StandardScaler()),
                          ('classifier', DecisionTreeClassifier
                           (random_state=42, max_depth=15, criterion='gini'))])
pipe_knn = Pipeline(steps=[('scaling',StandardScaler()),
                          ('classifier', KNeighborsClassifier())])
pipe_lr = Pipeline(steps=[('scaling',StandardScaler()),
                          ('classifier', LogisticRegression(random_state=42))])
pipe_rf = Pipeline(steps=[('scaling',StandardScaler()),
                          ('classifier', RandomForestClassifier(random_state=42))])
pipe_svc = Pipeline(steps=[('scaling',StandardScaler()),
                          ('classifier', SVC(random_state=42))])
pipe_bag = Pipeline(steps=[('scaling',StandardScaler()),
                          ('classifier', BaggingClassifier(random_state=42))])
pipes = [pipe_dt, pipe_knn, pipe_lr, pipe_rf, pipe_svc, pipe_bag]
names_pipes = ['DT','KNN','LR','RF','SVM','Bagging']

for i in range(len(pipes)):
  print(names_pipes[i])
  pipes[i].fit(X_train, y_train)
  y_pred = pipes[i].predict(X_test)
  print('Accuracy : ',accuracy_score(y_test, y_pred))
  print('Precision : ',precision_score(y_test, y_pred, average='binary'))
  print('Recall/ sensitivity : ',recall_score(y_test, y_pred, average='binary'))
  print('F1 : ',f1_score(y_test, y_pred, average='binary'))
  sens, spec, sup = sensitivity_specificity_support(y_test,y_pred, average='binary')
  print('Specificity : ',spec)
  print('--------')
  print('')
```

DigitalSkola
Uncover The World Of Digital Skills With Us

Thank YOU