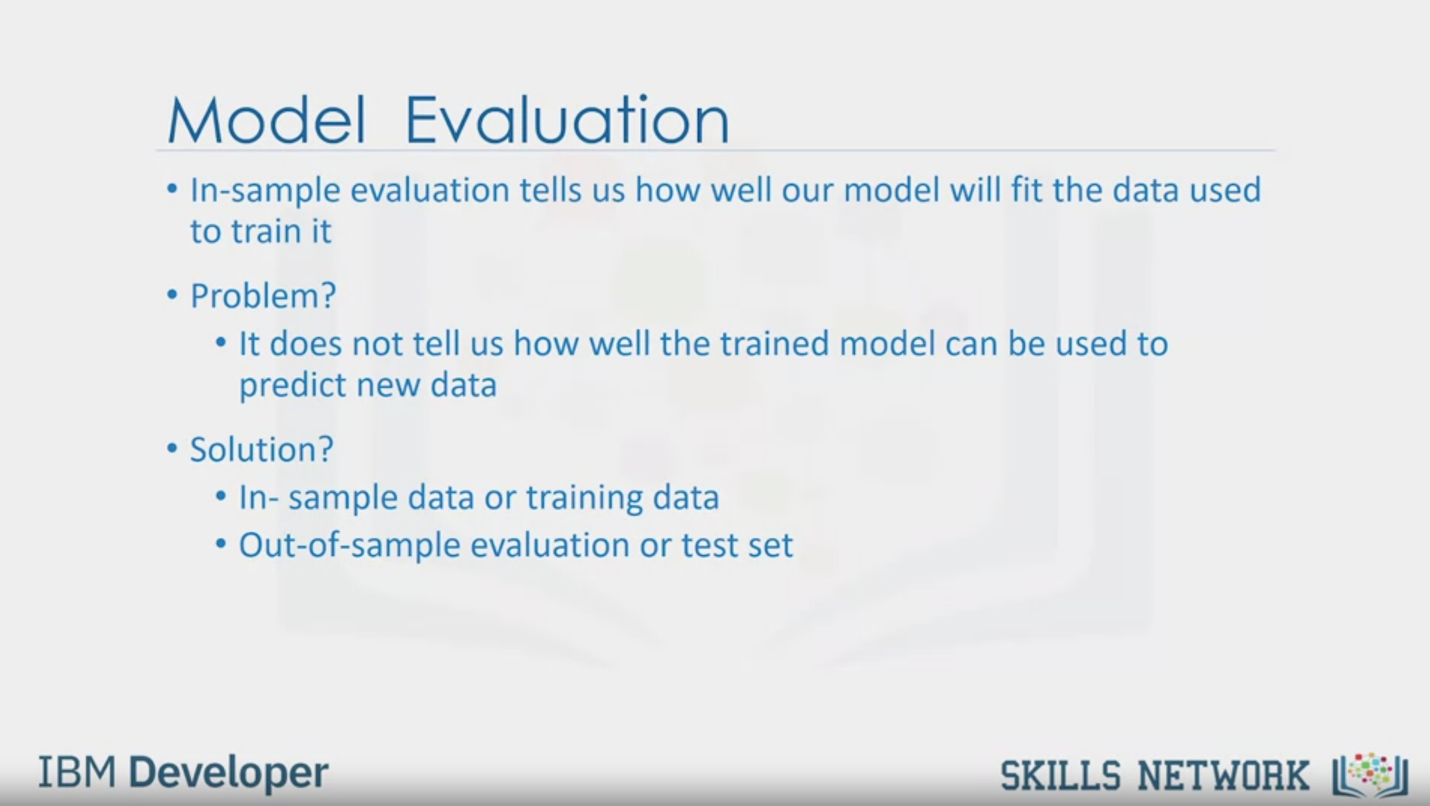


Model Evaluation

IBM Developer

SKILLS NETWORK



Model Evaluation

• In-sample evaluation tells us how well our model will fit the data used

to train it

• Problem?

• It does not tell us how well the trained model can be used to

predict new data

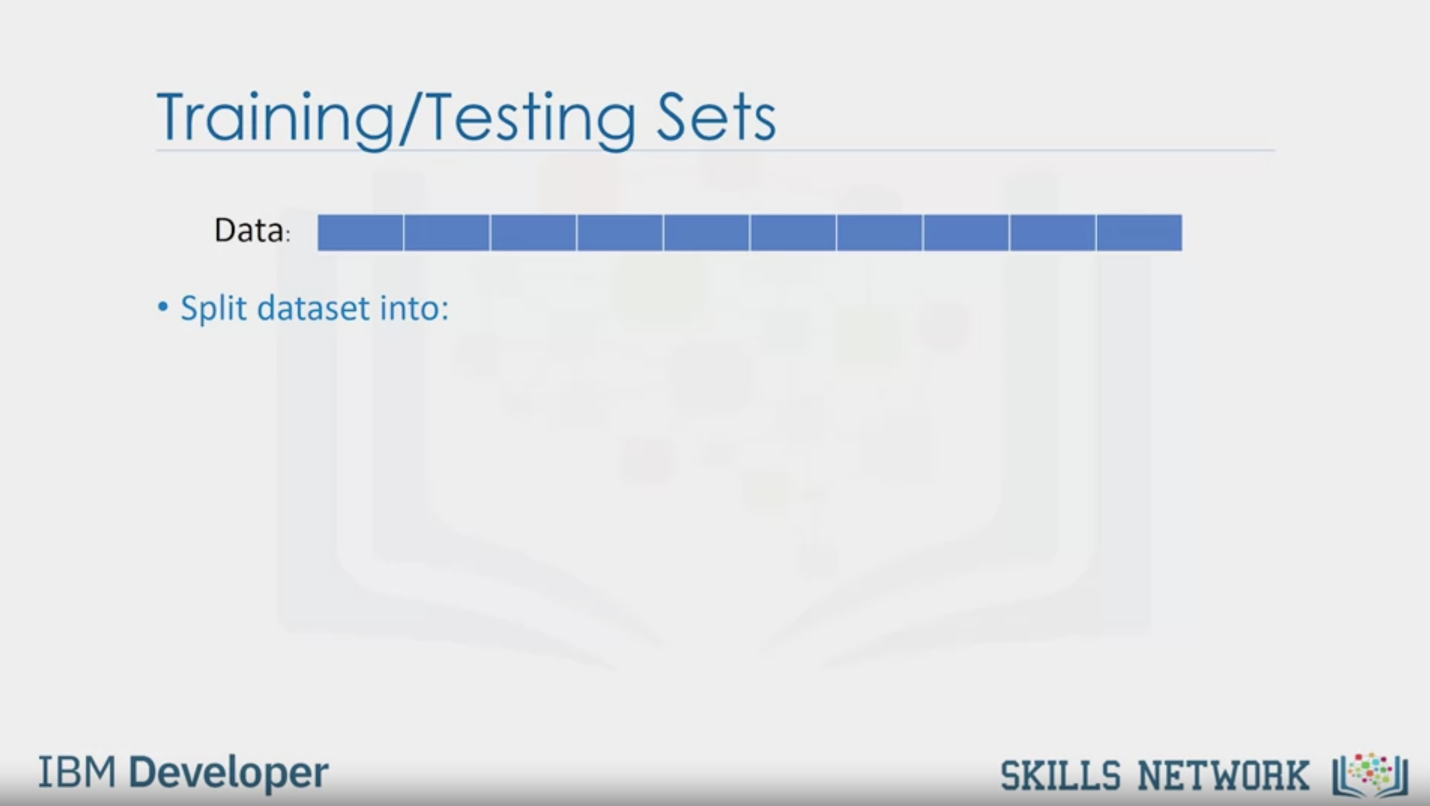
• Solution?

• In- sample data or training data

• Out-of-sample evaluation or test set

IBM Developer

SKILLS NETWORK



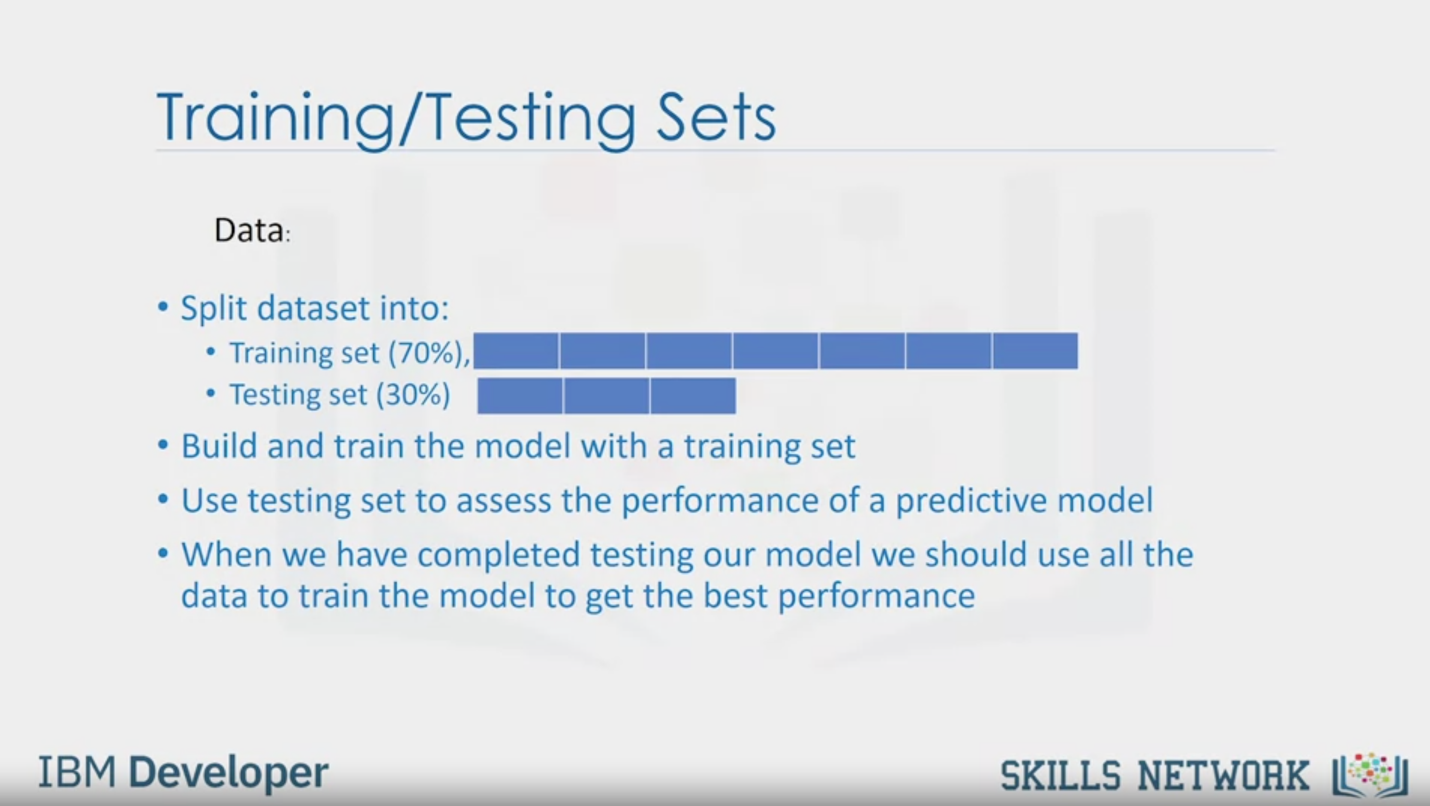
Training/Testing Sets

Data:

• Split dataset into:

IBM Developer

SKILLS NETWORK



Training/Testing Sets

Data:

• Split dataset into:

• Training set (70%),

• Testing set (30%)

• Build and train the model with a training set

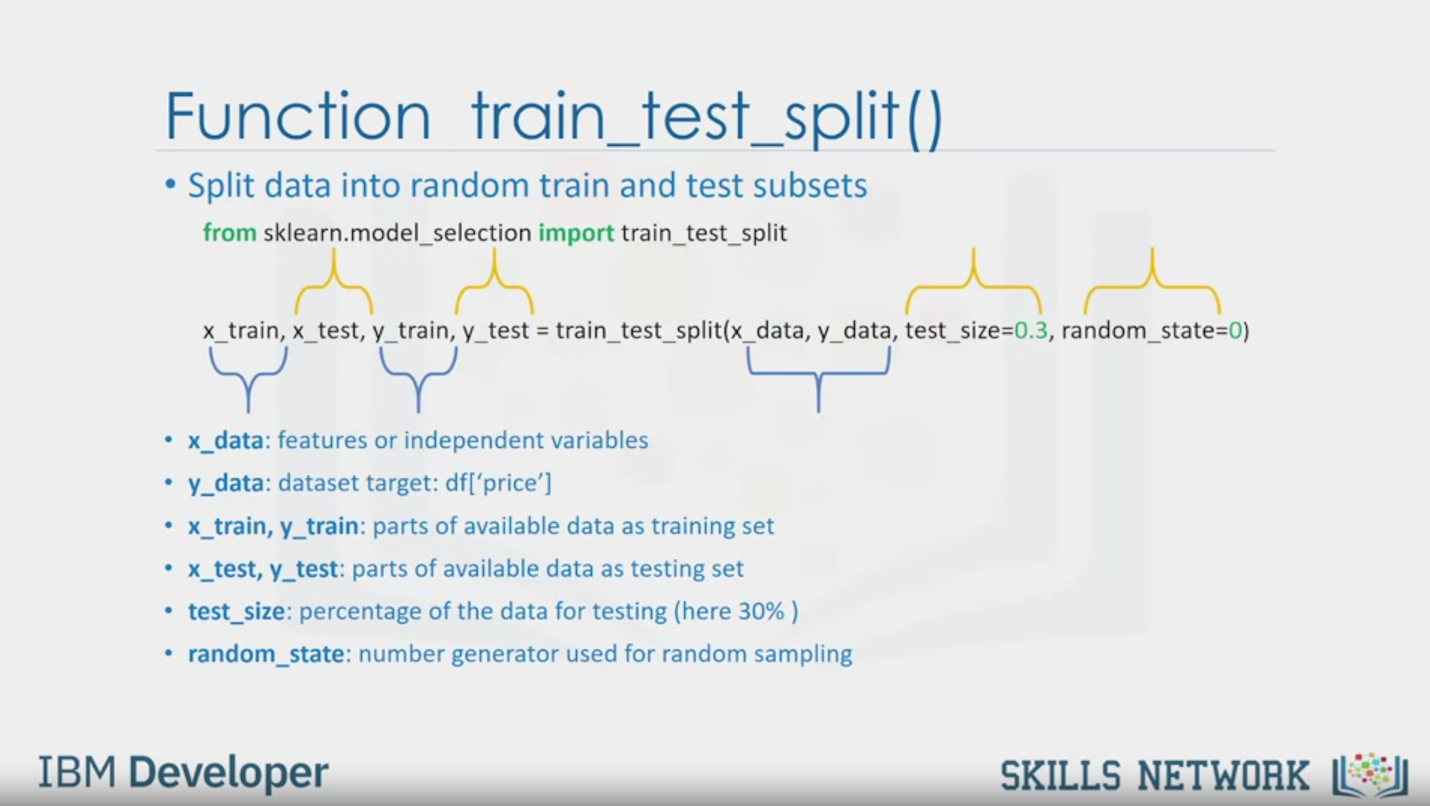
• Use testing set to assess the performance of a predictive model

• When we have completed testing our model we should use all the

data to train the model to get the best performance

IBM Developer

SKILLS NETWORK



Function train\_test\_split()

• Split data into random train and test subsets

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_data, y\_data, test\_size=0.3, random\_state=0)

• x\_data: features or independent variables

• y\_data: dataset target: df['price']

• x\_train, y\_train: parts of available data as training set

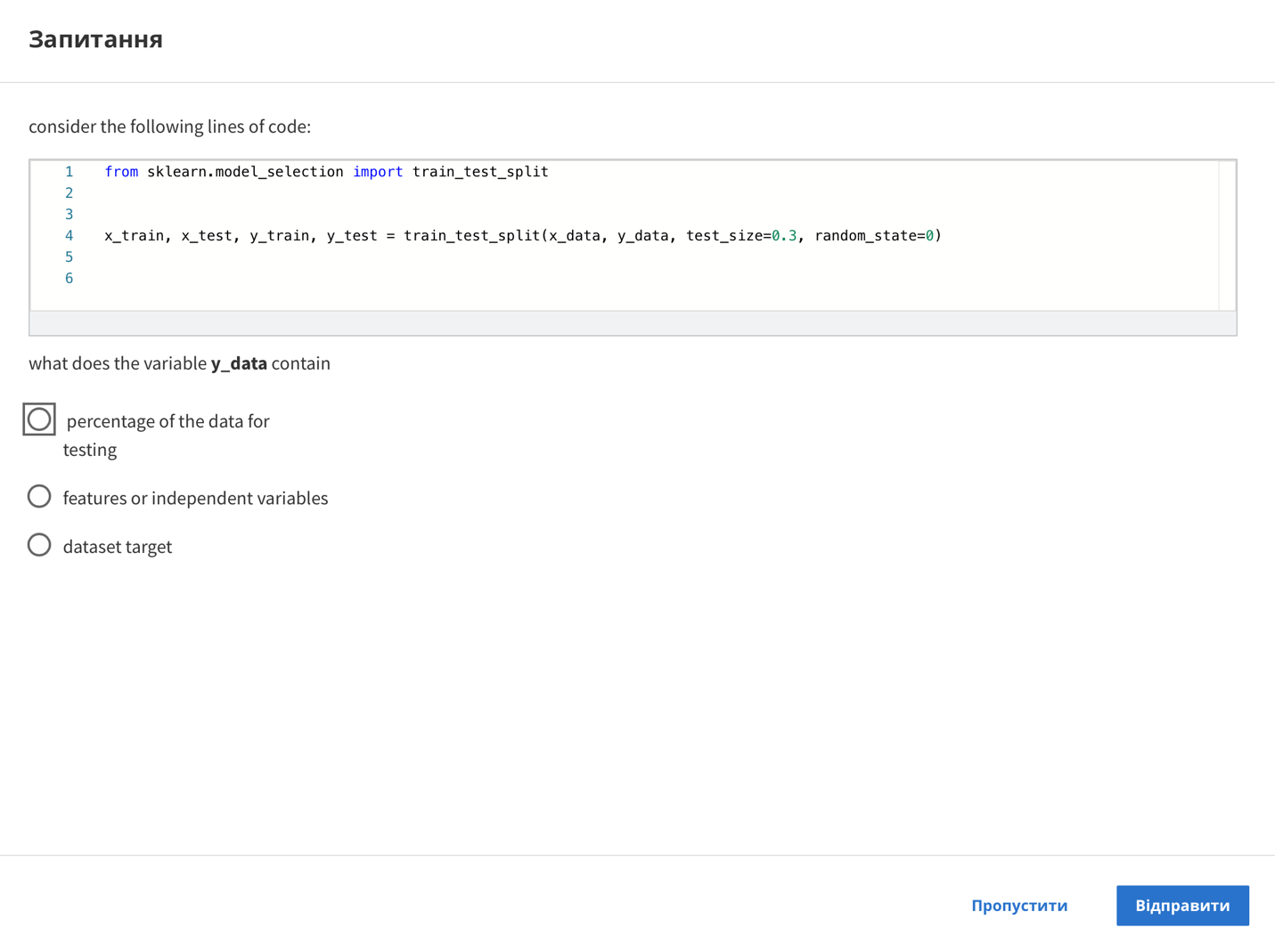
• x\_test, y\_test: parts of available data as testing set

• test\_size: percentage of the data for testing (here 30% )

• random\_state: number generator used for random sampling

IBM Developer

SKILLS NETWORK



Question

consider the following lines of code:

1 from sklearn.model\_selection import train\_test\_split

2

3

4 x\_train, x\_ test, y\_train, y\_test = train\_test\_split (x\_data, y\_data, test\_size=0.3, random\_state=0)

5

6

what does the variable y\_data contain

percentage of the data for

testing

features or independent variables

dataset target

Skip

Send



Generalization Performance

• Generalization error is measure of how well our data does at

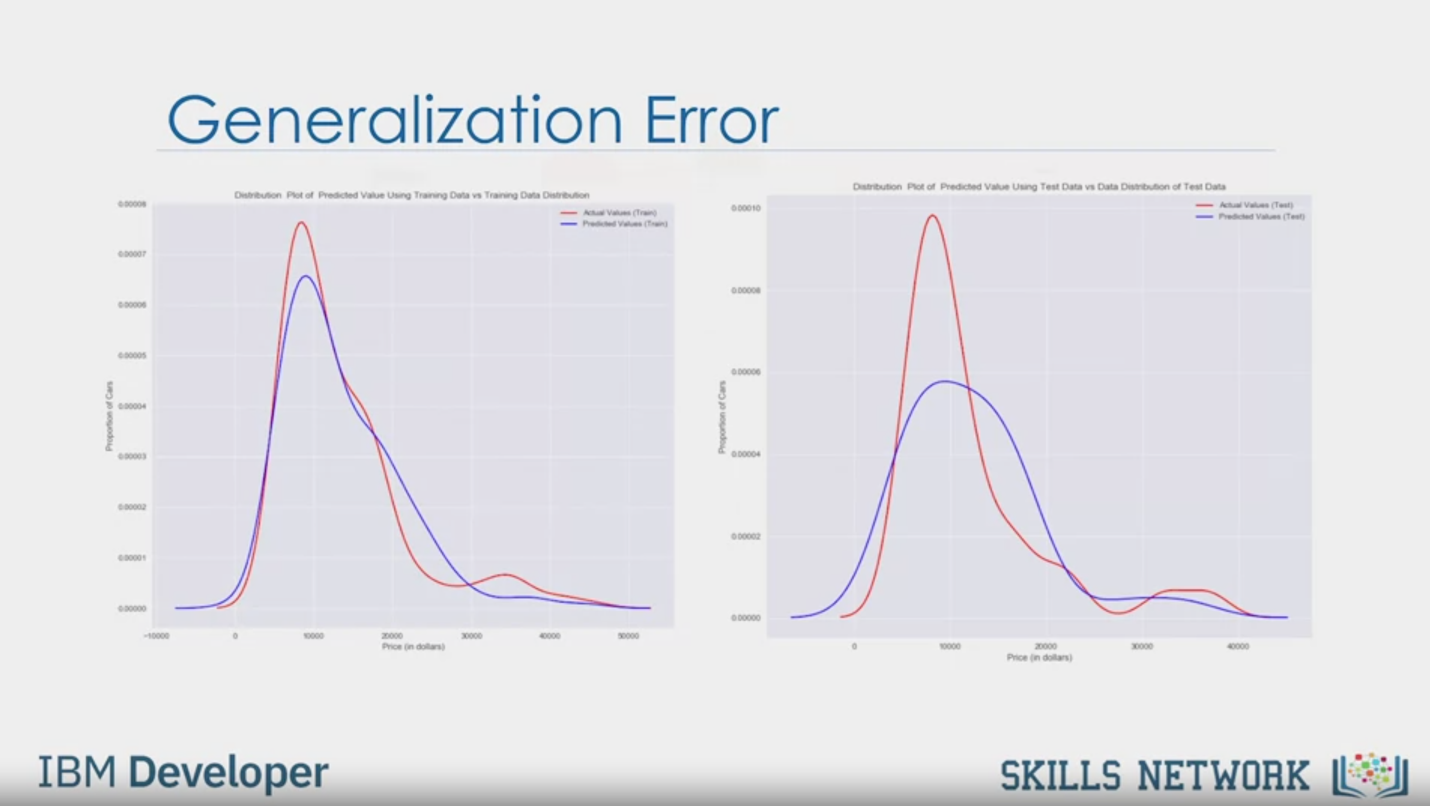
predicting previously unseen data

• The error we obtain using our testing data is an approximation of this

error

IBM Developer

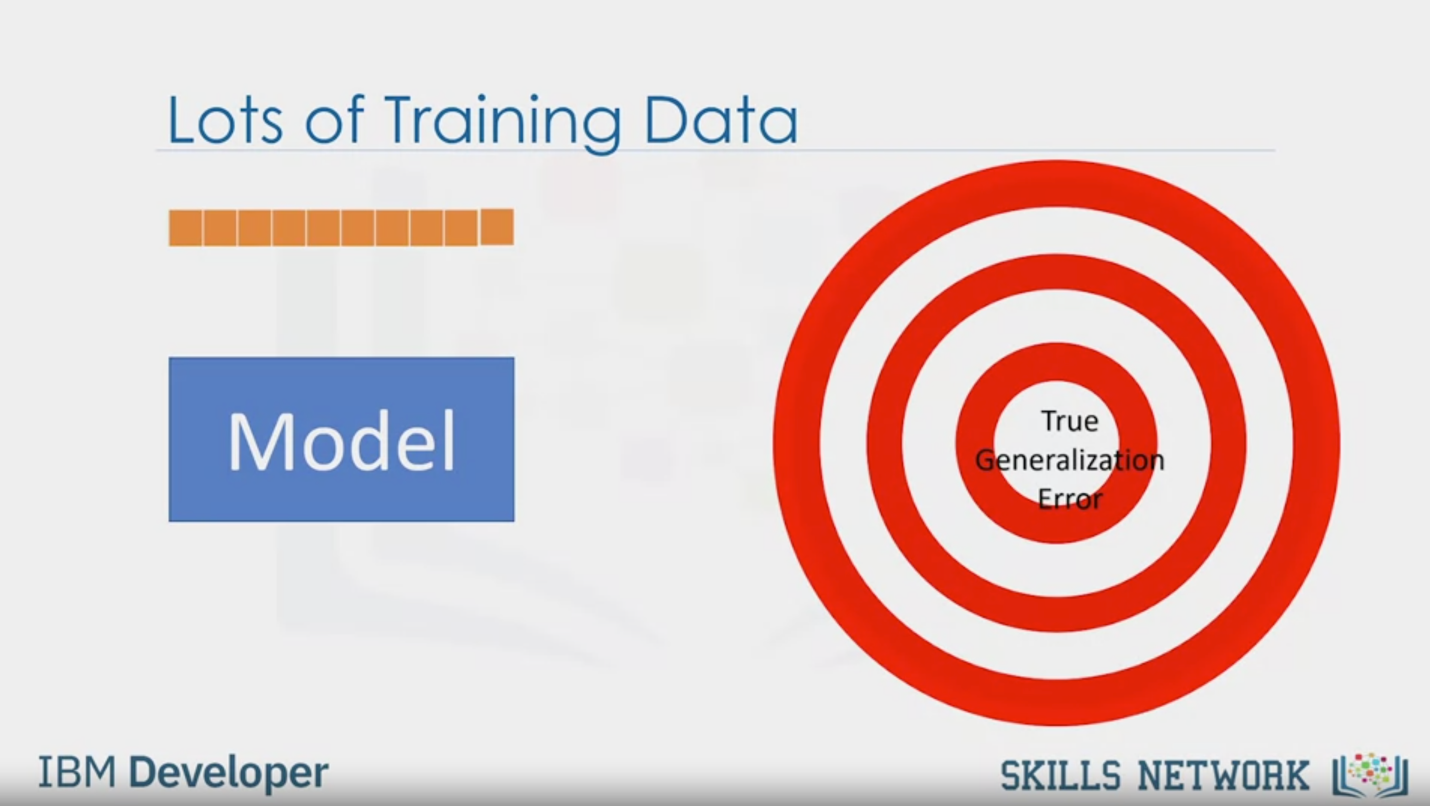
SKILLS NETWORK



Generalization Error

IBM Developer

SKILLS NETWORK



Lots of Training Data

Model

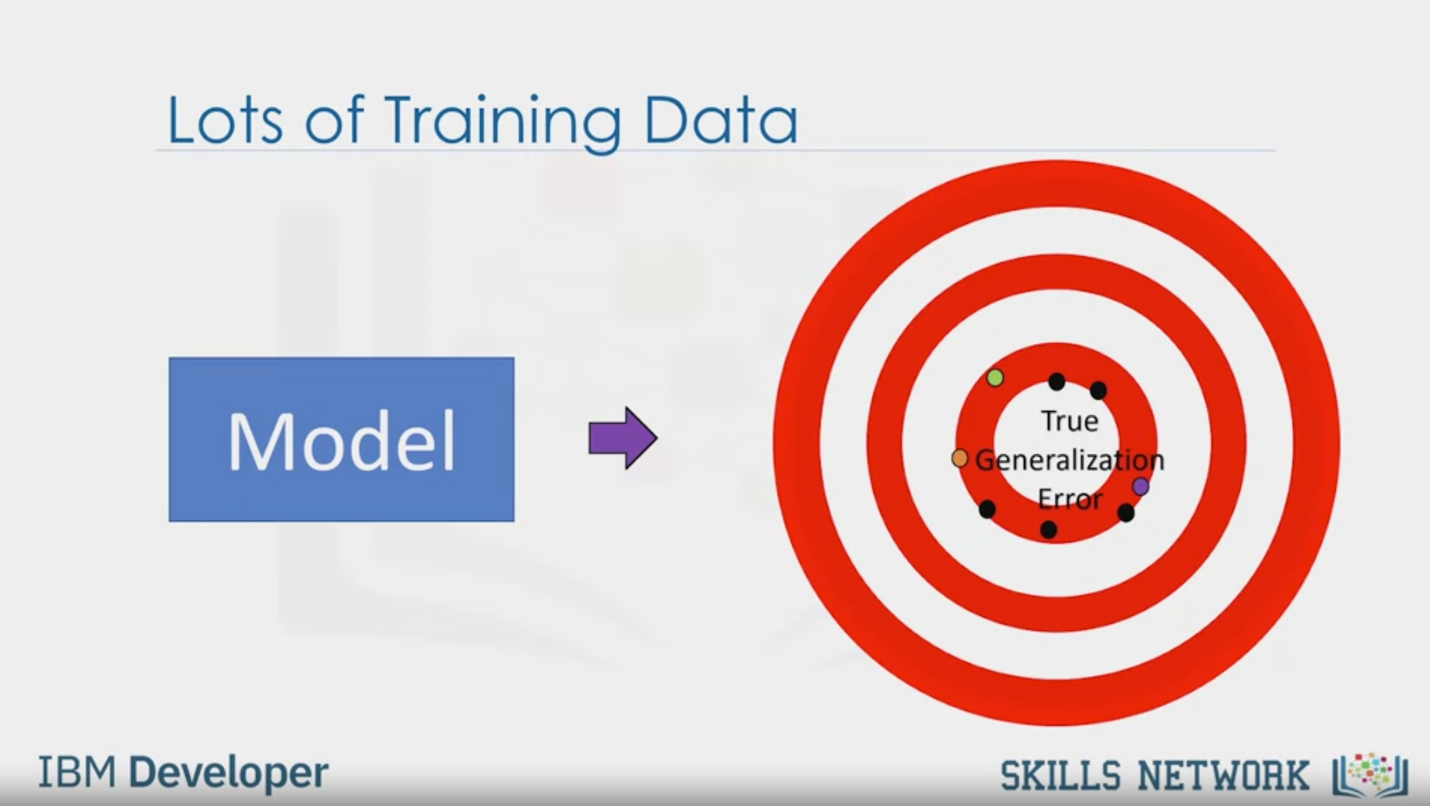
True

Generalization

Error

IBM Developer

SKILLS NETWORK



Lots of Training Data

Model

True

Generalization

Error

IBM Developer

SKILLS NETWORK



Lots of Training Data

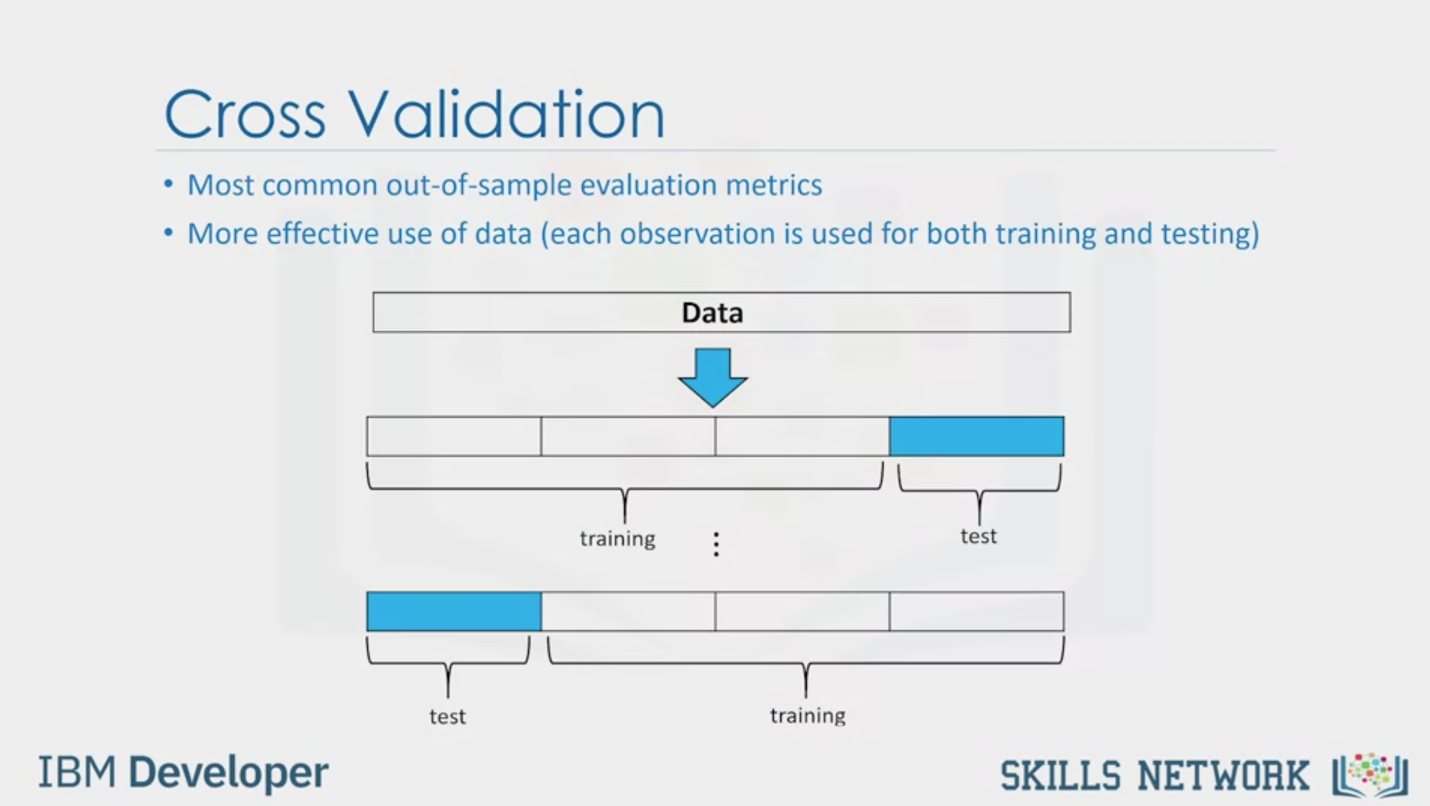
IBM Developer

True

Generalization

Error

SKILLS NETWORK



Cross Validation

• Most common out-of-sample evaluation metrics

• More effective use of data (each observation is used for both training and testing)

Data

training

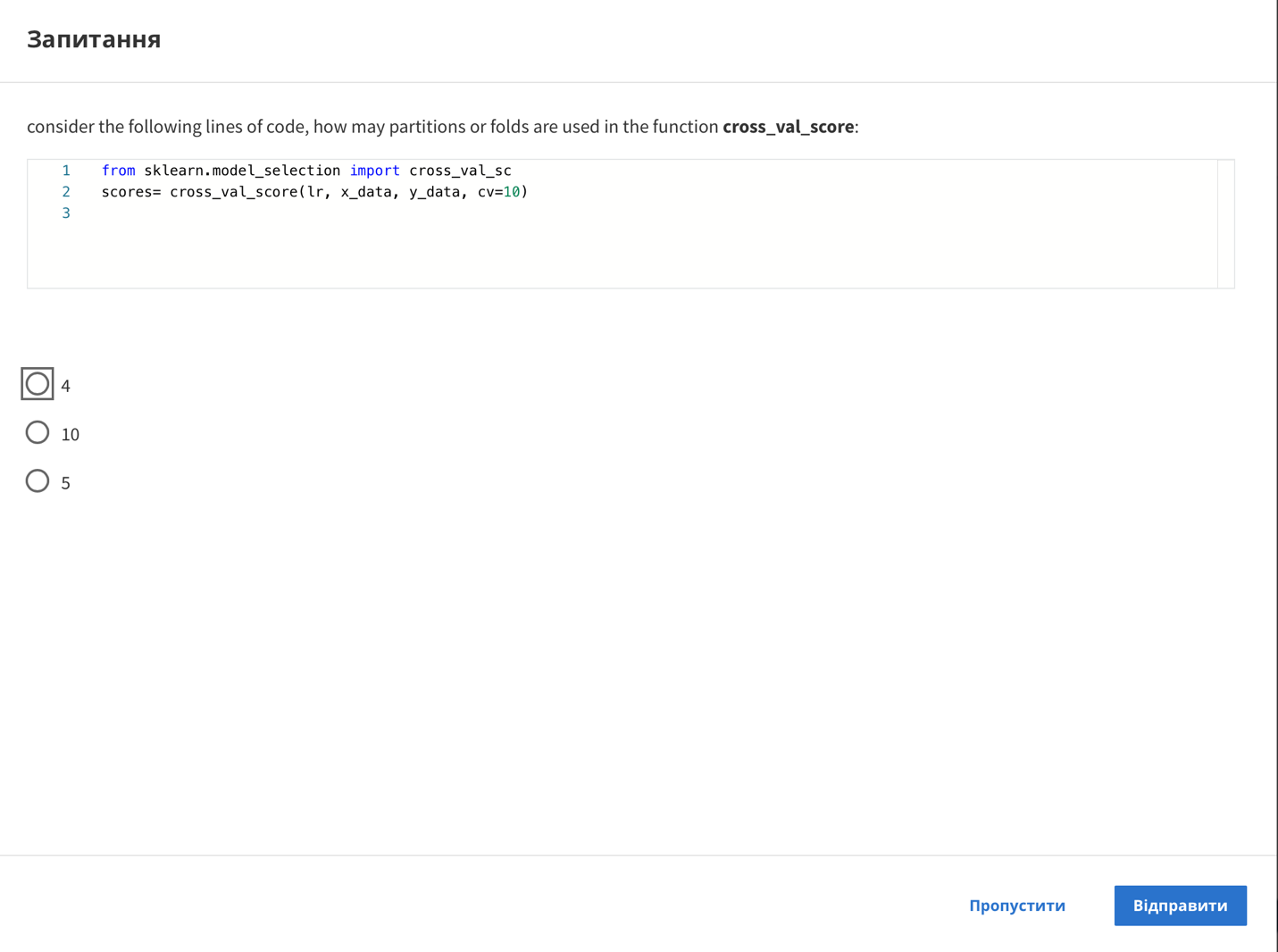
test

test

training

IBM Developer

SKILLS NETWORK



Question

consider the following lines of code, how may partitions or folds are used in the function cross\_val\_score:

1 from sklearn.model\_selection import cross\_val\_sc

2 scores= cross\_val\_score(lr, x\_data, y\_data, cv=10)

3

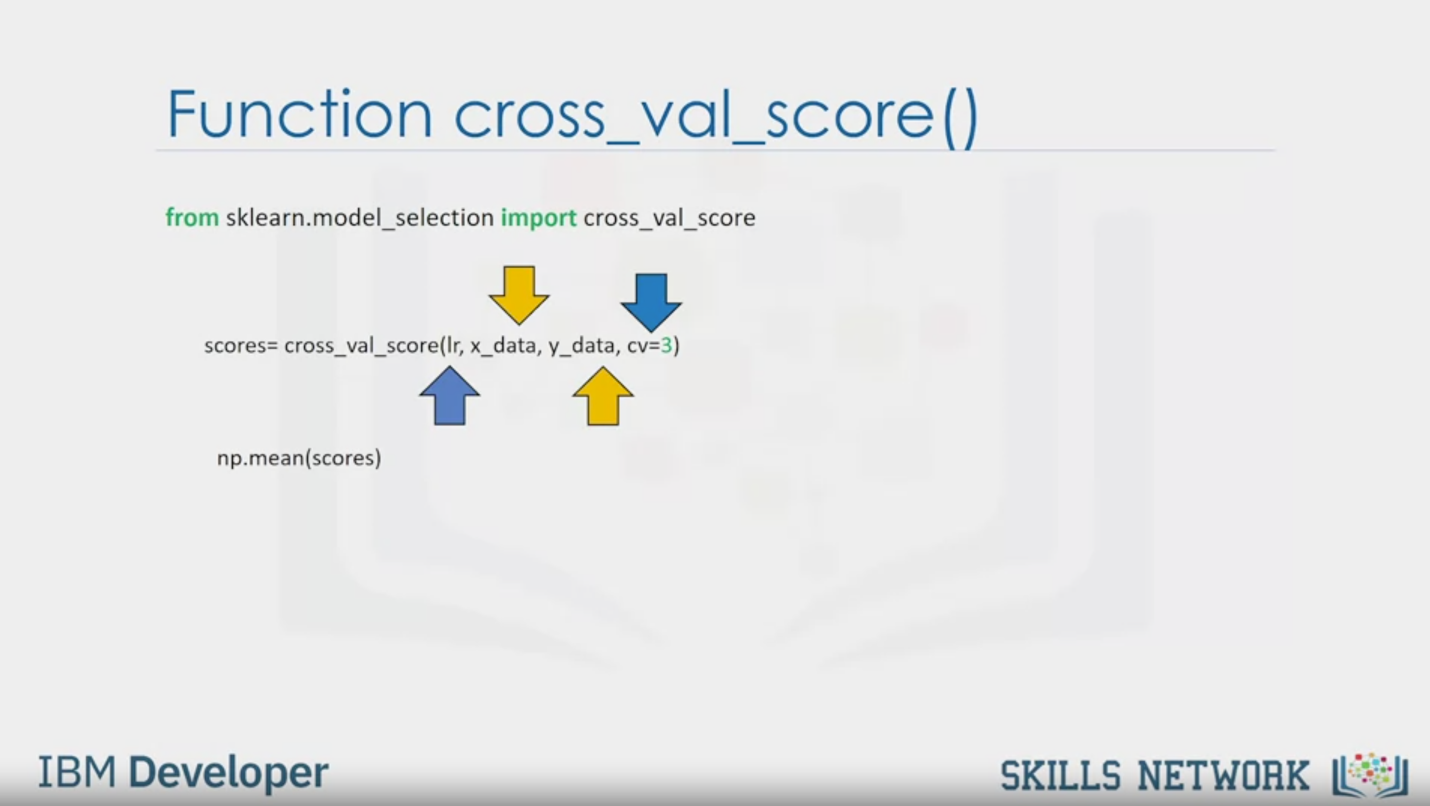
4

10

5

Skip

Send



Function cross\_val\_score()

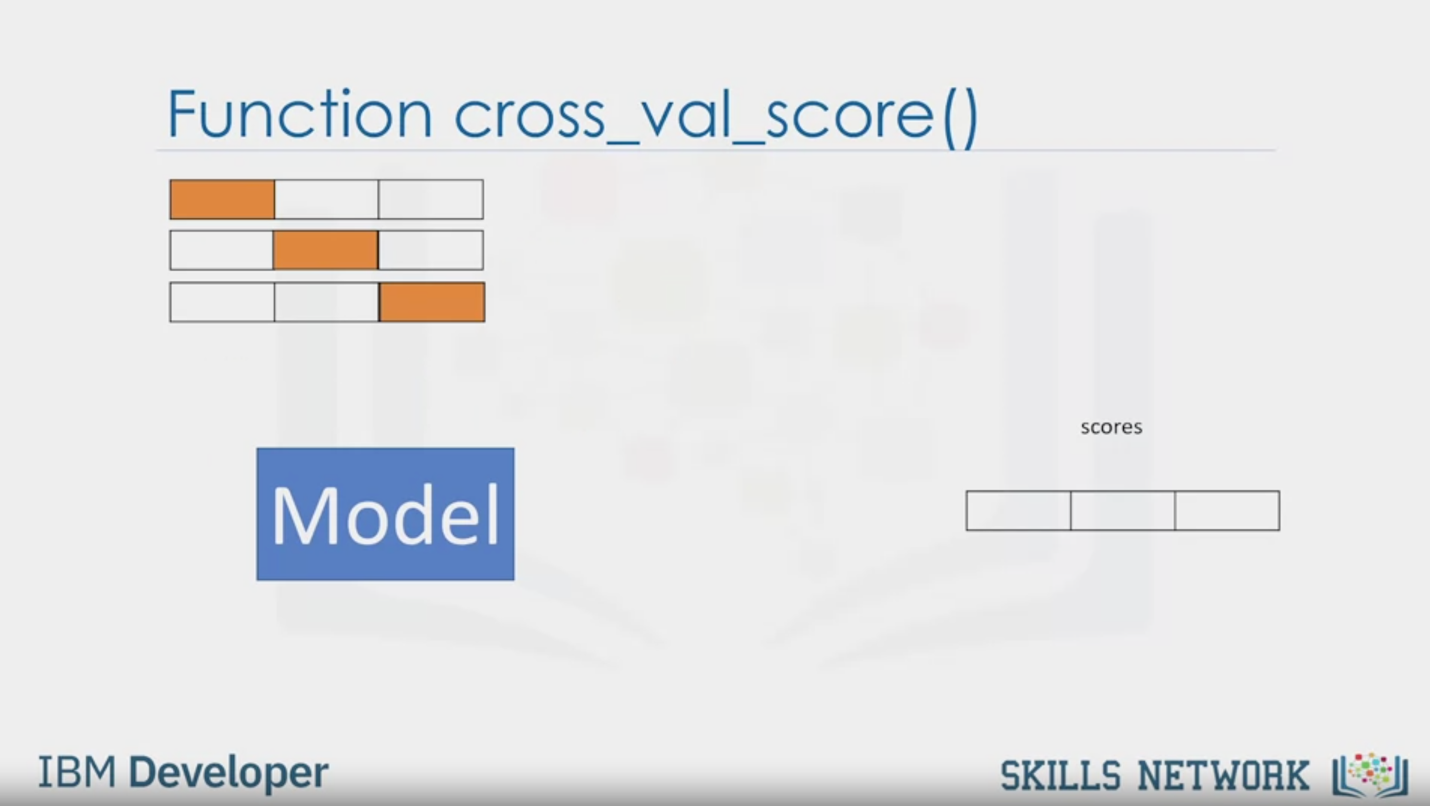
from sklearn.model\_selection import cross\_val\_score

scores= cross\_val\_score(Ir, x\_data, y\_data, cv=3)

np.mean(scores)

IBM Developer

SKILLS NETWORK



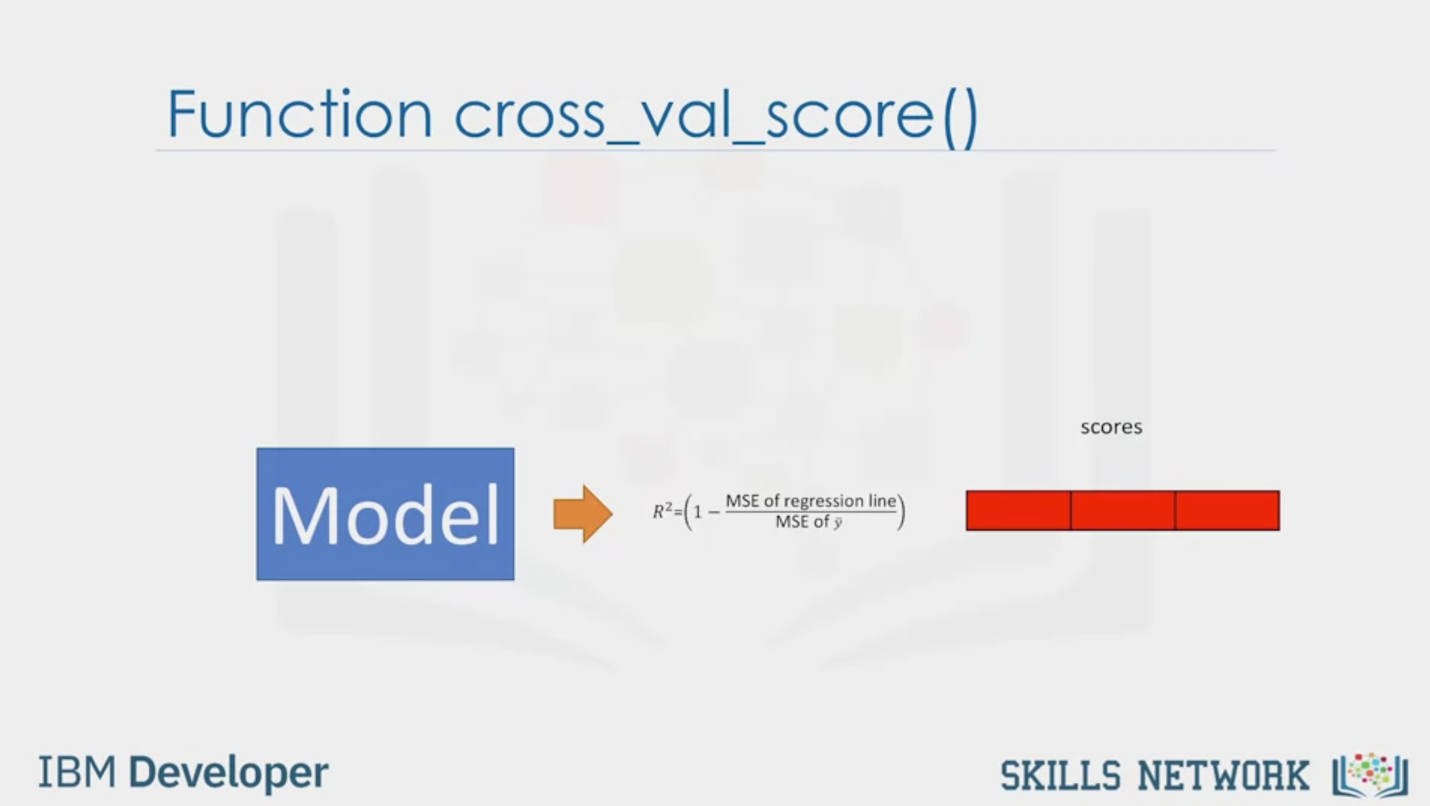
Function cross\_val\_score()

scores

Model

IBM Developer

SKILLS NETWORK



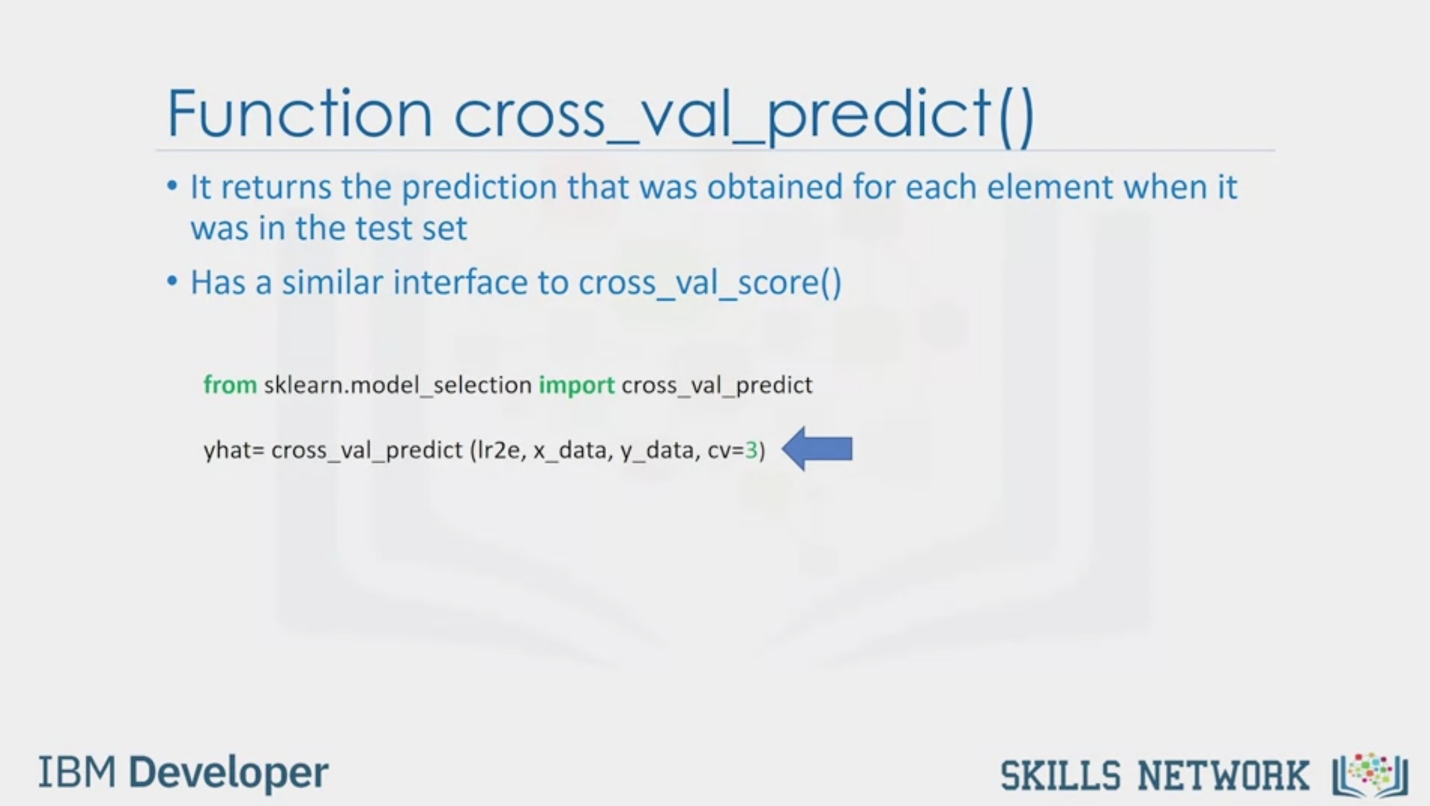
Function cross\_val\_score()

scores

Model

IBM Developer

SKILLS NETWORK



Function cross\_val\_predict()

• It returns the prediction that was obtained for each element when it

was in the test set

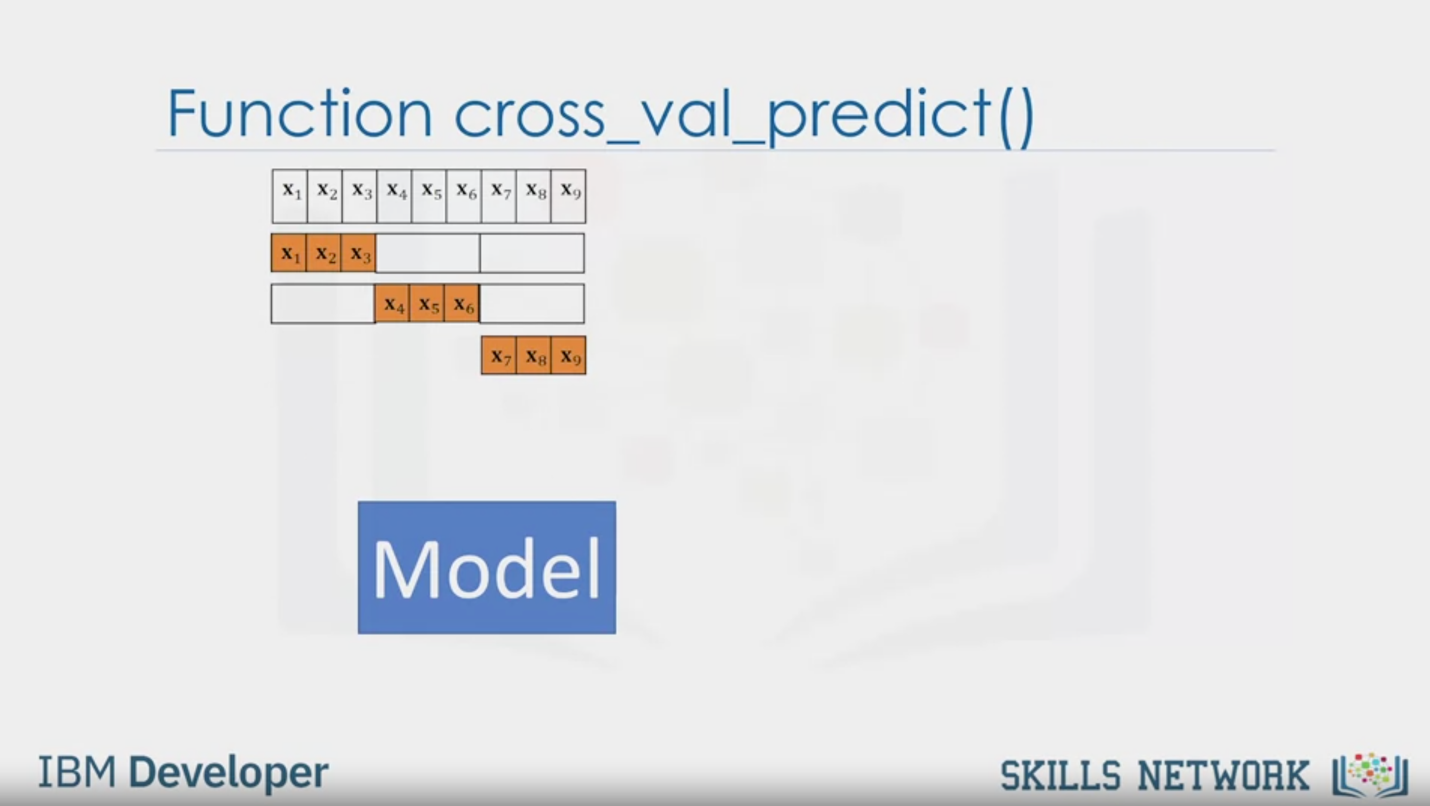
• Has a similar interface to cross\_val\_score()

from sklearn.model\_selection import cross\_val\_predict

yhat= cross\_val\_predict (Ir2e, x\_data, y\_data, cv=3)

IBM Developer

SKILLS NETWORK



Function cross\_val\_predict()

X1 X2 X3 X4 X5 X6 X7 X8 X9

X1 X2 X3

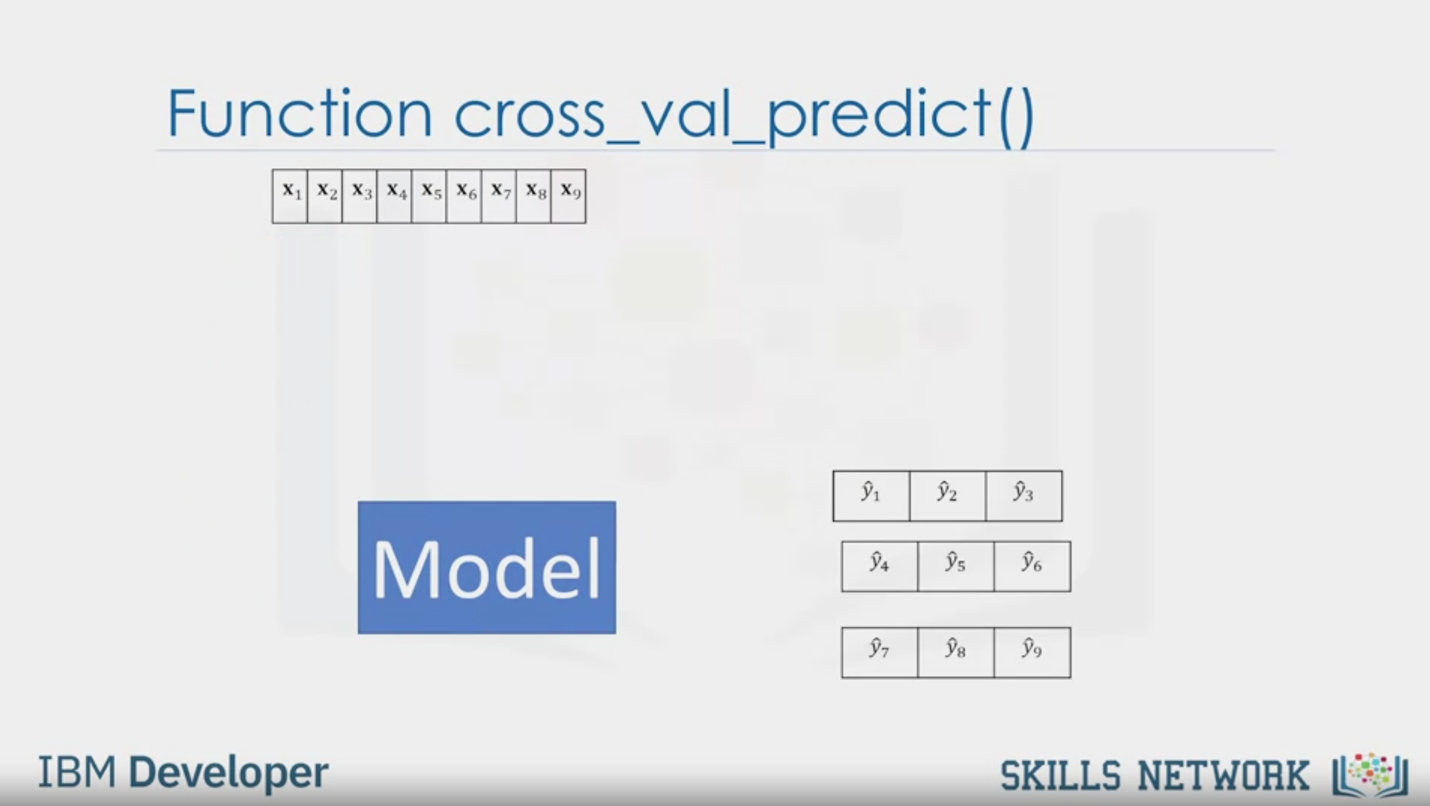
X4 X5 X6

X7 X8 X9

Model

IBM Developer

SKILLS NETWORK



Function cross\_val\_predict()

X1 X2 X3 X4 X5 X6 X7 X8 X9

1 2 3

4 5 6

7 8 9

Model

IBM Developer

SKILLS NETWORK



SKILLS NETWORK

IBM Developer

Model evaluation tells us how our model performs in the real world.

In the previous module,

we talked about in-sample evaluation.

In-sample evaluation tells us how well our model fits the data already given to train it.

It does not give us an estimate of how well the train model can predict new data.

The solution is to split our data up,

use the in-sample data or training data to train the model.

The rest of the data, called Test Data,

is used as out-of-sample data.

This data is then used to approximate, how the model performs in the real world.

Separating data into training and testing sets is an important part of model evaluation.

We use the test data to get an idea how our model will perform in the real world.

When we split a dataset,

usually the larger portion of data is used for

training and a smaller part is used for testing.

For example, we can use 70 percent of the data for training.

We then use 30 percent for testing.

We use training set to build a model and discover predictive relationships.

We then use a testing set to evaluate model performance.

When we have completed testing our model,

we should use all the data to train the model.

A popular function, in the scikit-learn package for

splitting datasets, is the train test split function.

This function randomly splits a dataset into training and testing subsets.

From the example code snippet,

this method is imported from sklearn.cross-validation.

The input parameters y\_data is the target variable.

In the car appraisal example,

it would be the price and x\_data,

the list of predictive variables.

In this case, it would be all the other variables in

the car dataset that we are using to try to predict the price.

The output is an array.

x\_train and y\_train the subsets for training.

x\_test and y\_test the subsets for testing.

In this case, the test size is a percentage of the data for the testing set.

Here, it is 30 percent.

The random state is a random seed for random data set splitting.

Generalization error is a measure of how well

our data does at predicting previously unseen data.

The error we obtain using our testing data is an approximation of this error.

This figure shows the distribution of the actual values in

red compared to the predicted values from a linear regression in blue.

We see the distributions are somewhat similar.

If we generate the same plot using the test data,

we see the distributions are relatively different.

The difference is due to

a generalization error and represents what we see in the real world.

Using a lot of data for training,gives us an accurate means

of determining how well our model will perform in the real world.

But the precision of the performance will be low.

Let's clarify this with an example.

The center of this bull's eye represents the correct generalization error.

Let's say we take a random sample of the data using

90 percent of the data for training and 10 percent for testing.

The first time we experiment,

we get a good estimate of the training data.

If we experiment again training the model with a different combination of samples,

we also get a good result.

But, the results will be different relative to the first time we run the experiment.

Repeating the experiment again with

a different combination of training and testing samples,

the results are relatively close to the generalization error,

but distinct from each other.

Repeating the process, we get good approximation of the generalization error,

but the precision is poor i.e.

all the results are extremely different from one another.

If we use fewer data points to train the model and more to test the model,

the accuracy of the generalization performance will be

less, but the model will have good precision.

The figure above demonstrates this.

All our error estimates are relatively close together,

but they are further away from the true generalization performance.

To overcome this problem, we use cross-validation.

One of the most common out of sample evaluation metrics is cross-validation.

In this method, the dataset is split into K equal groups.

Each group is referred to as a fold.

For example, four folds.

Some of the folds can be used as a training set which we use

to train the model and the remaining parts are used as a test set,

which we use to test the model.

For example, we can use three folds for training,

then use one fold for testing.

This is repeated until each partition is used for both training and testing.

At the end, we use the average results as the estimate of out-of-sample error.

The evaluation metric depends on the model,

for example, the r squared.

The simplest way to apply cross-validation is to call the cross\_val\_score function,

which performs multiple out-of-sample evaluations.

This method is imported from sklearn's model selection package.

We then use the function cross\_val\_score.

The first input parameters,

the type of model we are using to do the cross-validation.

In this example, we initialize a linear regression model or object

lr which we passed the cross\_val\_score function.

The other parameters are x\_data,

the predictive variable data,

and y\_data, the target variable data.

We can manage the number of partitions with the cv parameter.

Here, cv equals three,

which means the data set is split into three equal partitions.

The function returns an array of scores,

one for each partition that was chosen as the testing set.

We can average the result together to estimate out of

sample r squared using the mean function NnumPi.

Let's see an animation,

let's see the result of the score array in the last slide.

First, we split the data into three folds.

We use two folds for training,

the remaining fold for testing.

The model will produce an output.

We will use the output to calculate a score.

In the case of the r squared i.e.

coefficient of determination, we will store that value in an array.

We will repeat the process using two folds for training and one fold for testing.

Save the score, then use

a different combination for training and the remaining fold for testing.

We store the final result.

The cross\_val\_score function returns

a score value to tell us the cross-validation result.

What if we want a little more information?

What if we want to know the actual predicted values

supplied by our model before the r squared values are calculated?

To do this, we use the cross\_ val\_predict function.

The input parameters are exactly the same as the cross\_val\_score function,

but the output is a prediction.

Let's illustrate the process.

First, we split the data into three folds.

We use two folds for training,

the remaining fold for testing.

The model will produce an output, and we will store it in an array.

We will repeat the process using two folds for training, one for testing.

The model produces an output again.

Finally, we use the last two folds for training.

Then we use the testing data.

This final testing fold produces an output.

These predictions are stored in an array.