

Polynomial Regression

and Pipelines

IBM Developer

SKILLS NETWORK



Polynomial Regressions

• A special case of the general linear regression model

• Useful for describing curvilinear relationships

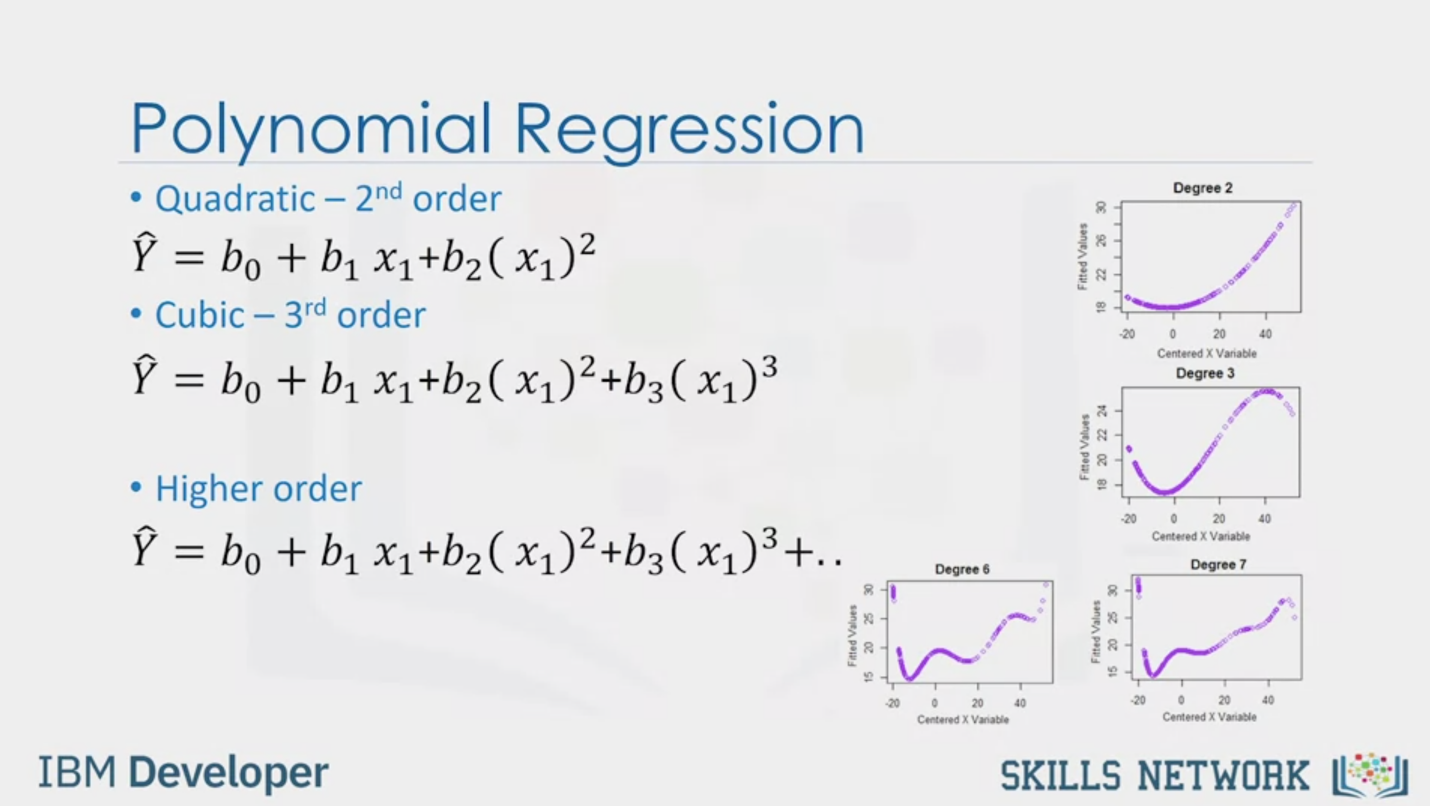
Curvilinear relationships:

By squaring or setting higher-order terms

of the predictor variables

IBM Developer

SKILLS NETWORK



Polynomial Regression

• Quadratic - 2nd order

= b0 + b1 x1+b2 (x1)2

• Cubic - 3rd order

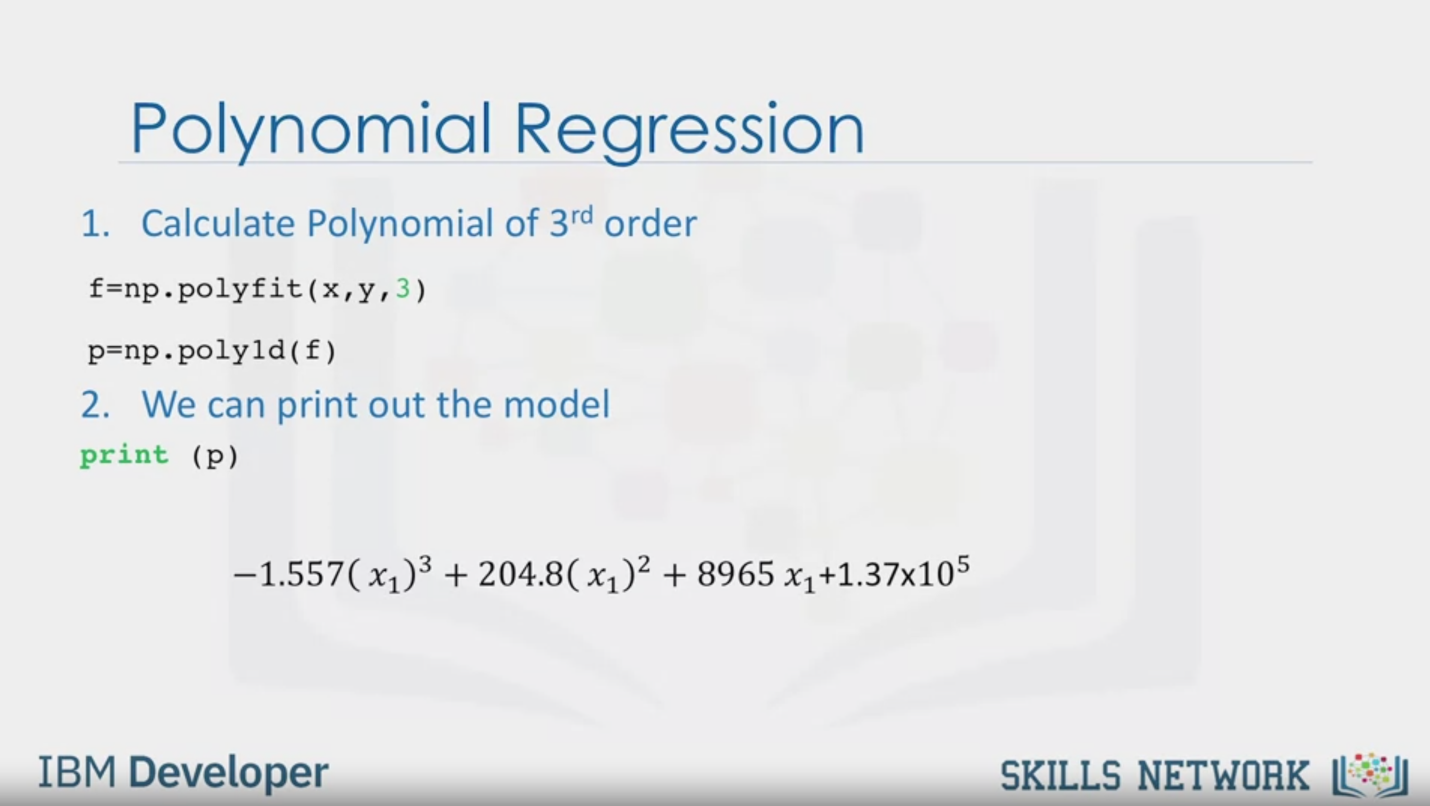
= b0 + b1 x1+b2 (x1)2+b3 (x1)3

• Higher order

= b0 + b1 x1+b2 (x1)2+b3 (x1)3+..

IBM Developer

SKILLS NETWORK



Polynomial Regression

1. Calculate Polynomial of 3rd order

f=np.polyfit(x,y,3)

p=np.polyld(f)

2. We can print out the model

print (p)

-1.557 (x1)3 + 204.8(x1)2 + 8965x1+1.37x105

IBM Developer

SKILLS NETWORK

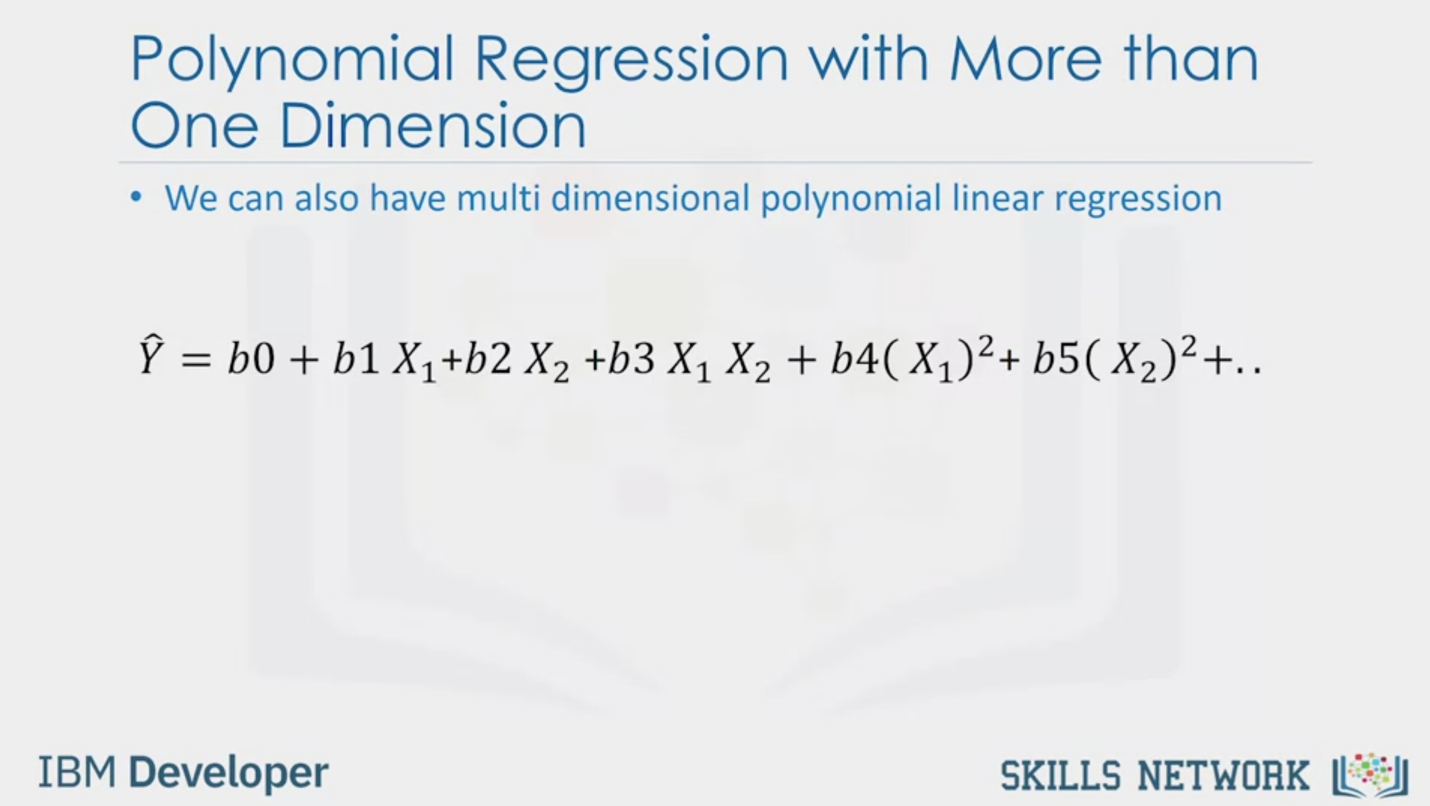


Polynomial Regression

• We can also have multi dimensional polynomial linear regression

IBM Developer

SKILLS NETWORK



Polynomial Regression with More than

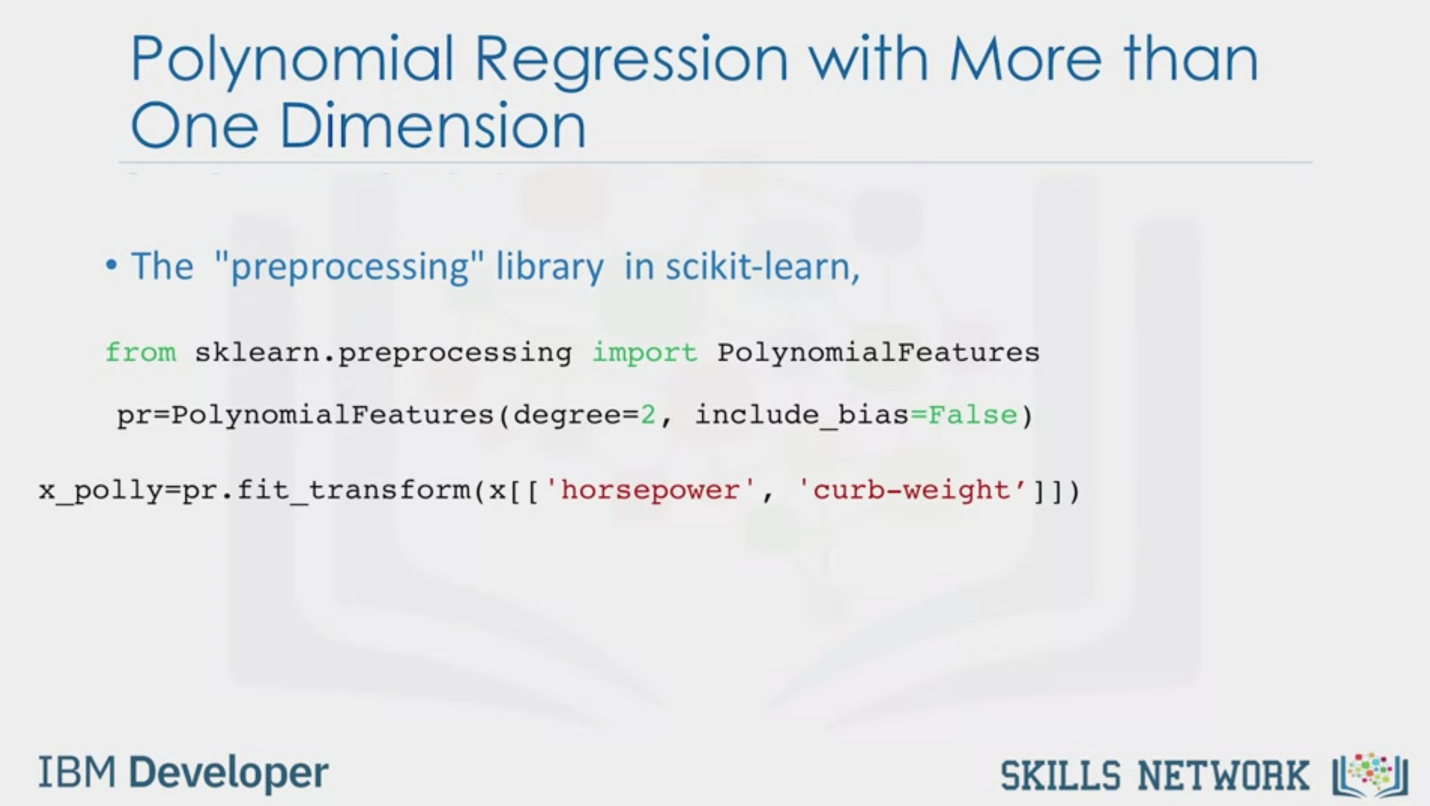
One Dimension

• We can also have multi dimensional polynomial linear regression

= b0 + b1 X1+b2X2+b3 X1 X2 + b4(X1)2+ b5(X2)2+..

IBM Developer

SKILLS NETWORK



Polynomial Regression with More than

One Dimension

• The "preprocessing" library in scikit-learn,

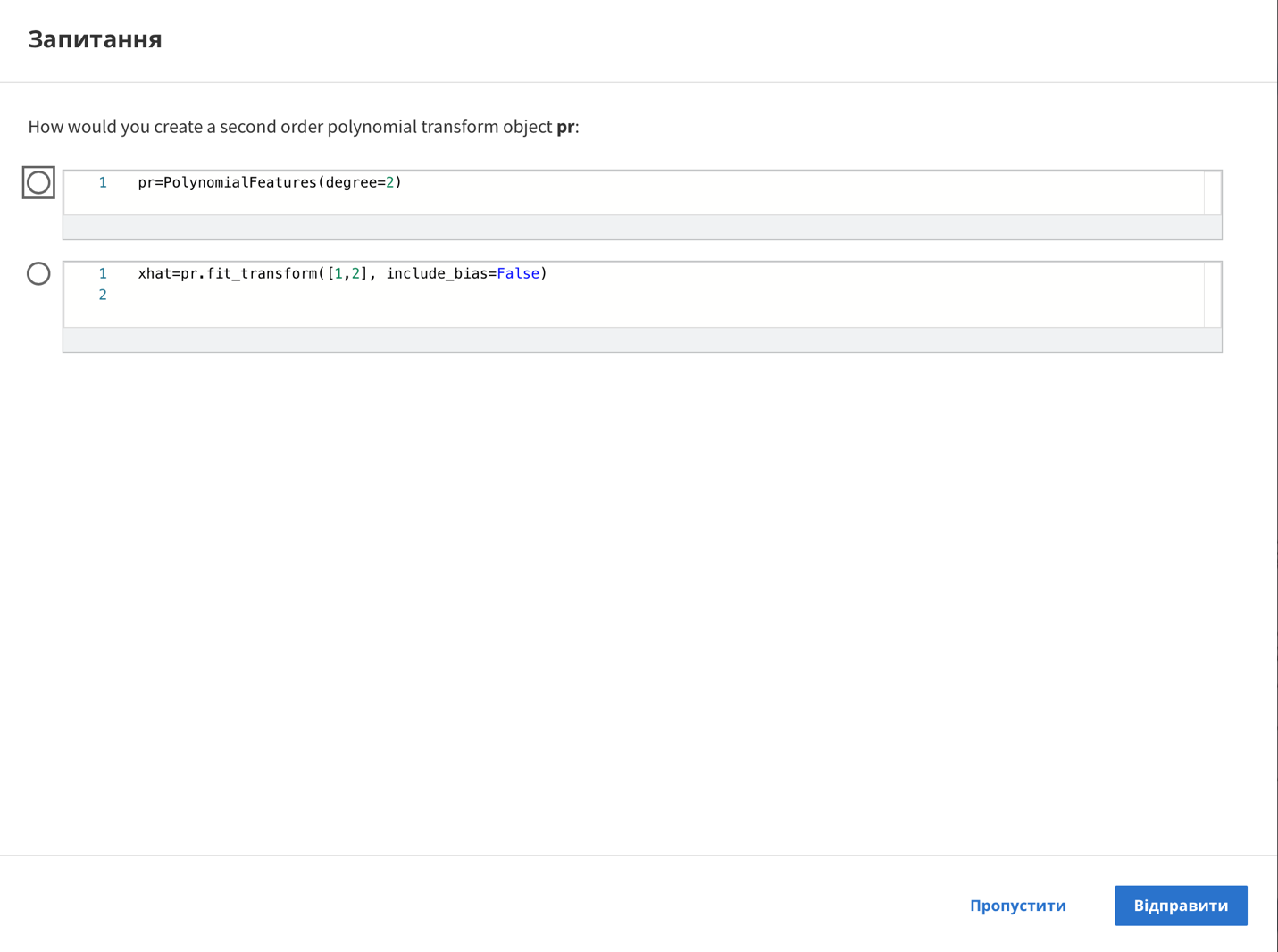
from sklearn.preprocessing import PolynomialFeatures

pr=PolynomialFeatures(degree=2, include\_bias=False)

x\_polly=pr.fit\_transform(x[['horsepower', 'curb-weight']])

IBM Developer

SKILLS NETWORK



Question

How would you create a second order polynomial transform object pr:

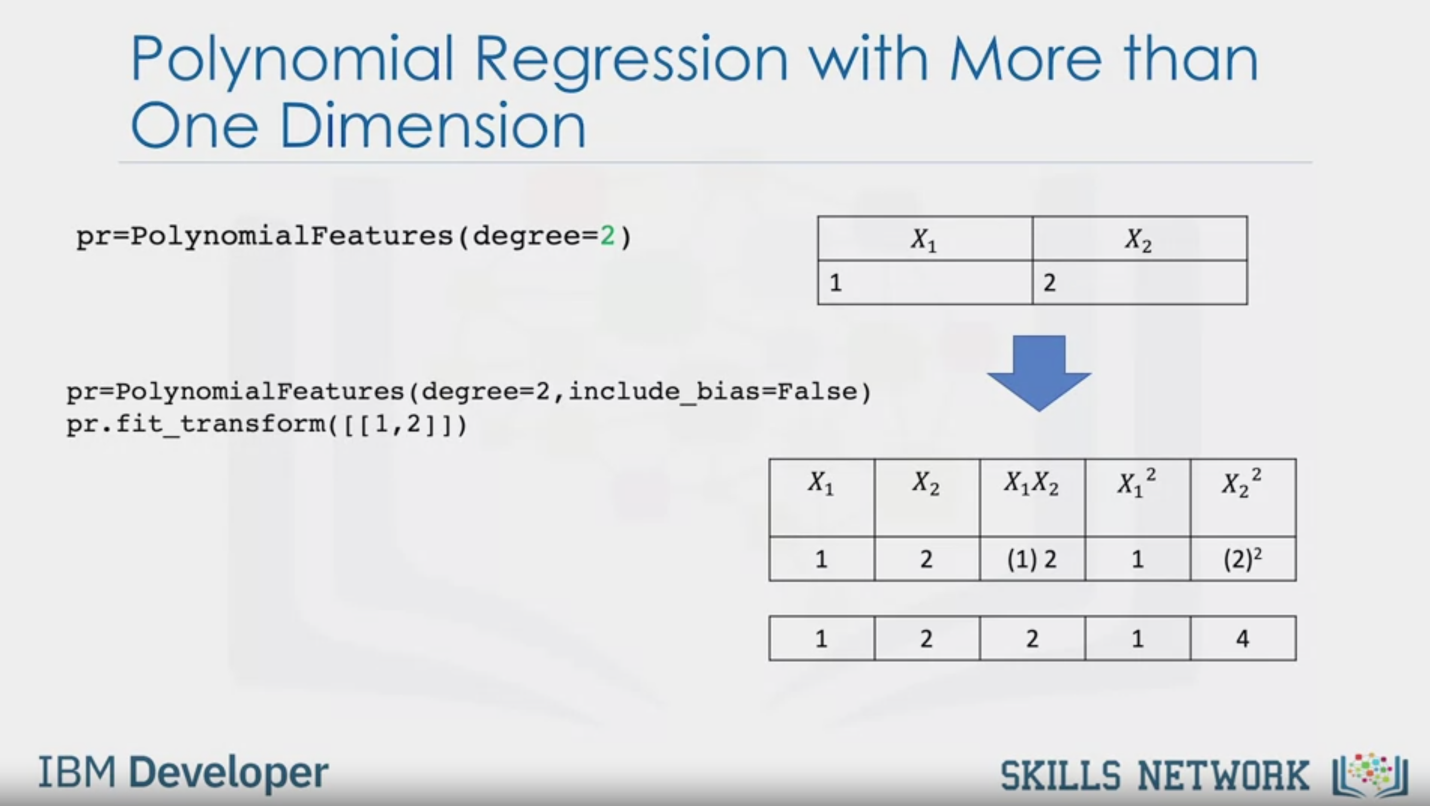
1 pr=PolynomialFeatures(degree=2)

1 xhat=pr.fit\_transform([1,2], include\_bias=False)

2

Skip

Send



Polynomial Regression with More than

One Dimension

pr=PolynomialFeatures (degree=2 )

X1

X2

1

2

pr=PolynomialFeatures(degree=2,include\_bias=False)

pr.fit\_transform([[1,2]])

X1

X2

X1 X2

X12

X22

1

2

(1) 2

(2)2

1

2

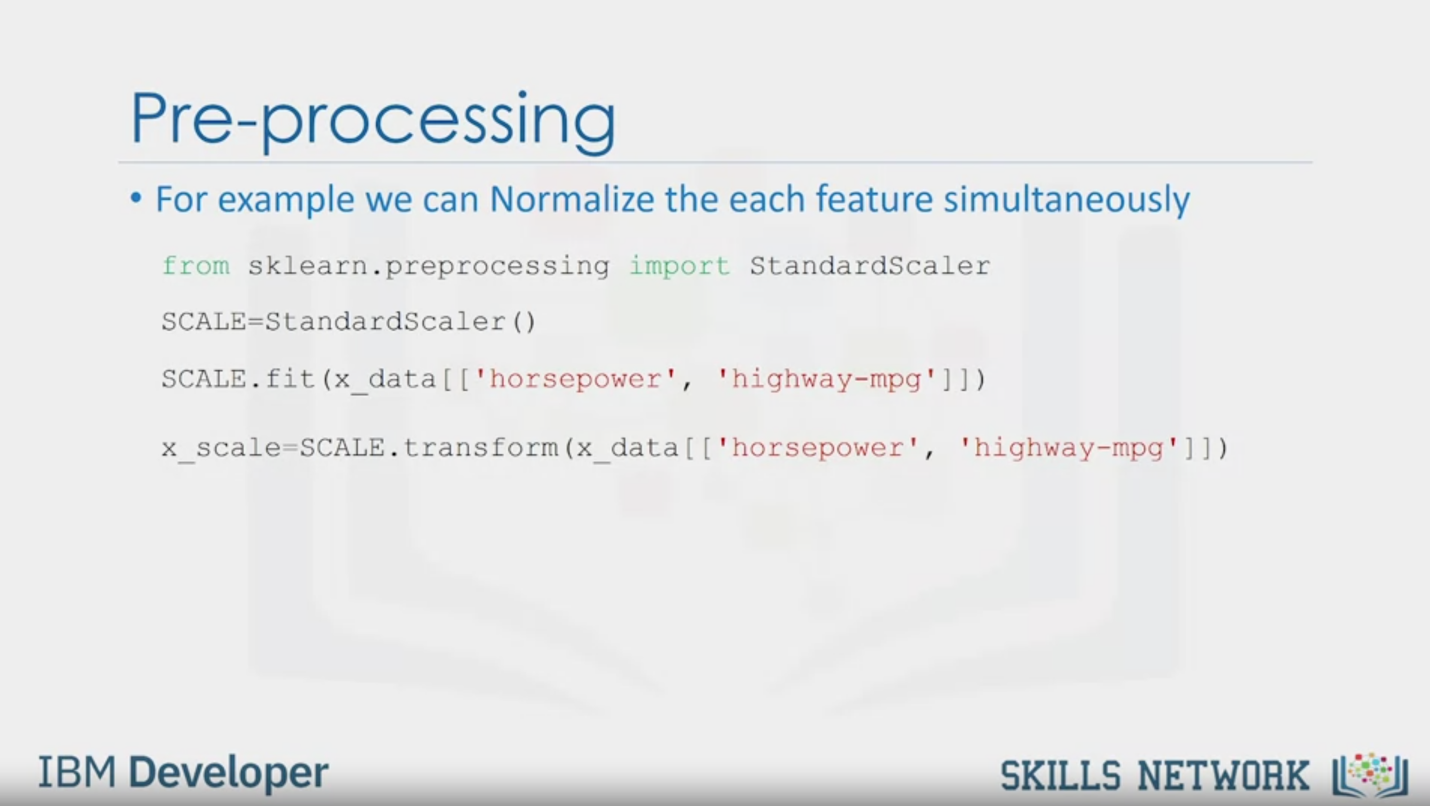
2

1

4

IBM Developer

SKILLS NETWORK



Pre-processing

• For example we can Normalize the each feature simultaneously

from sklearn.preprocessing import StandardScaler

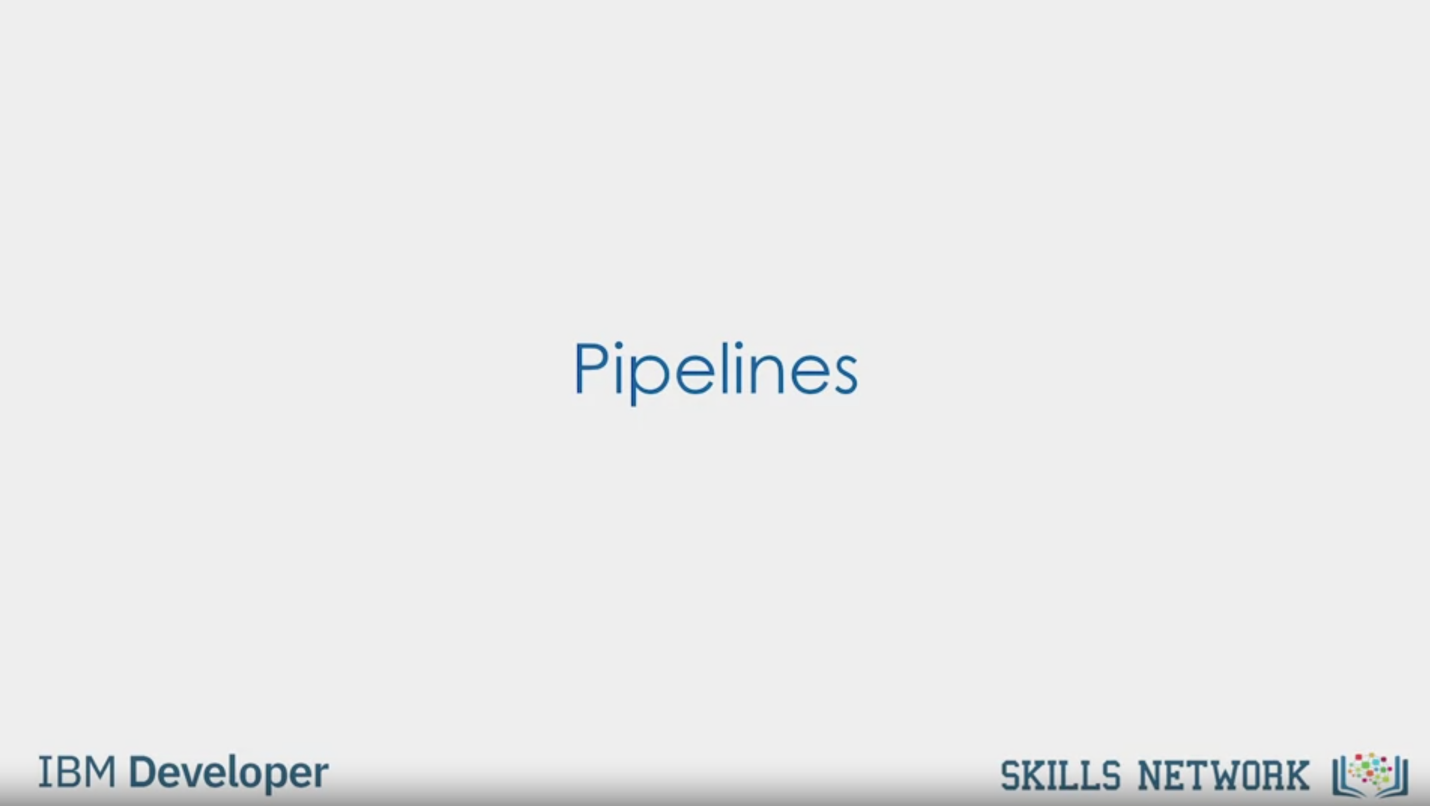
SCALE=StandardScaler()

SCALE.fit (x\_data [['horsepower', 'highway-mpg']])

x\_scale=SCALE.transform (x\_ data [['horsepower', 'highway-mog']])

IBM Developer

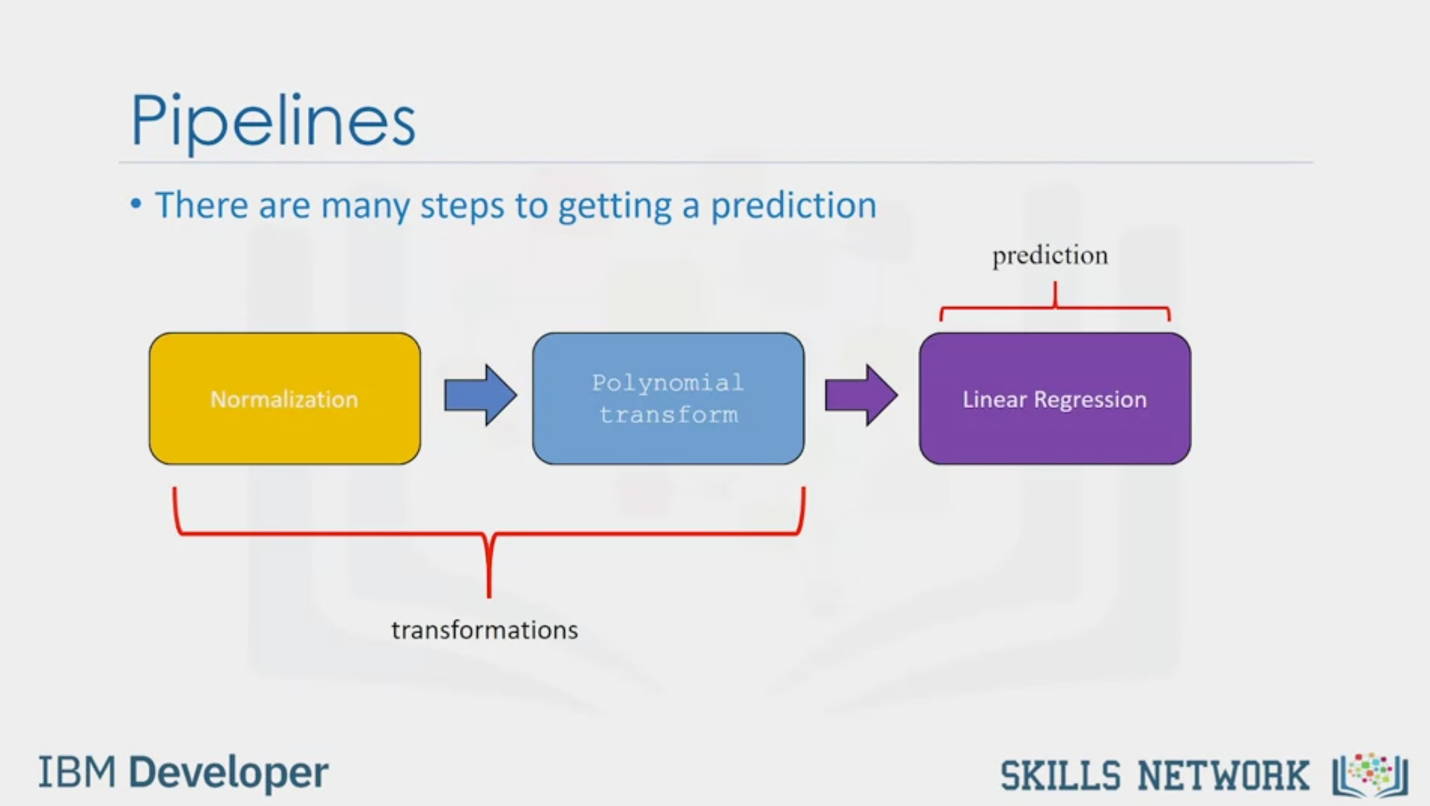
SKILLS NETWORK



Pipelines

IBM Developer

SKILLS NETWORK



Pipelines

• There are many steps to getting a prediction

Normalization

Polynomial

transform

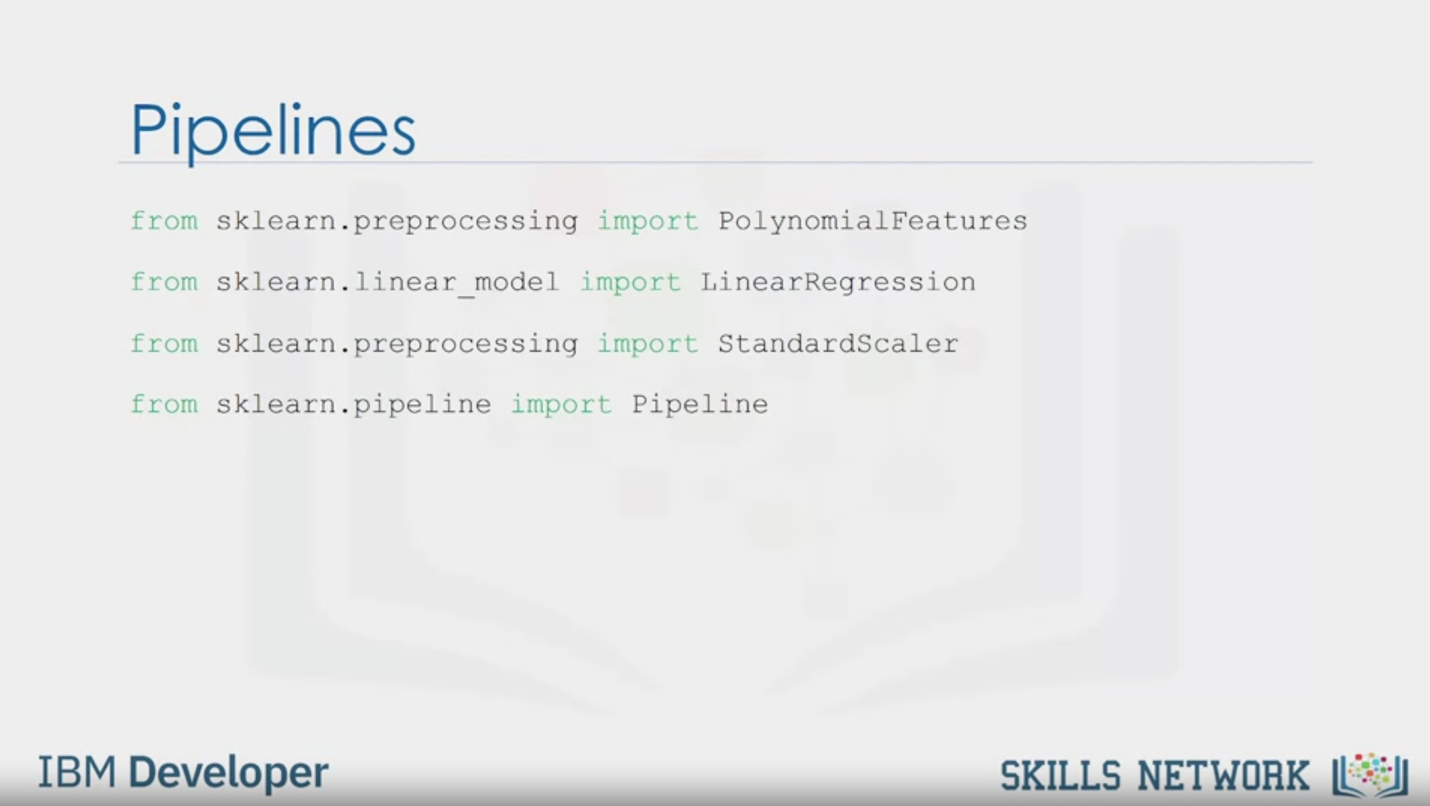
transformations

prediction

Linear Regression

IBM Developer

SKILLS NETWORK



Pipelines

from sklearn.preprocessing import PolynomialFeatures

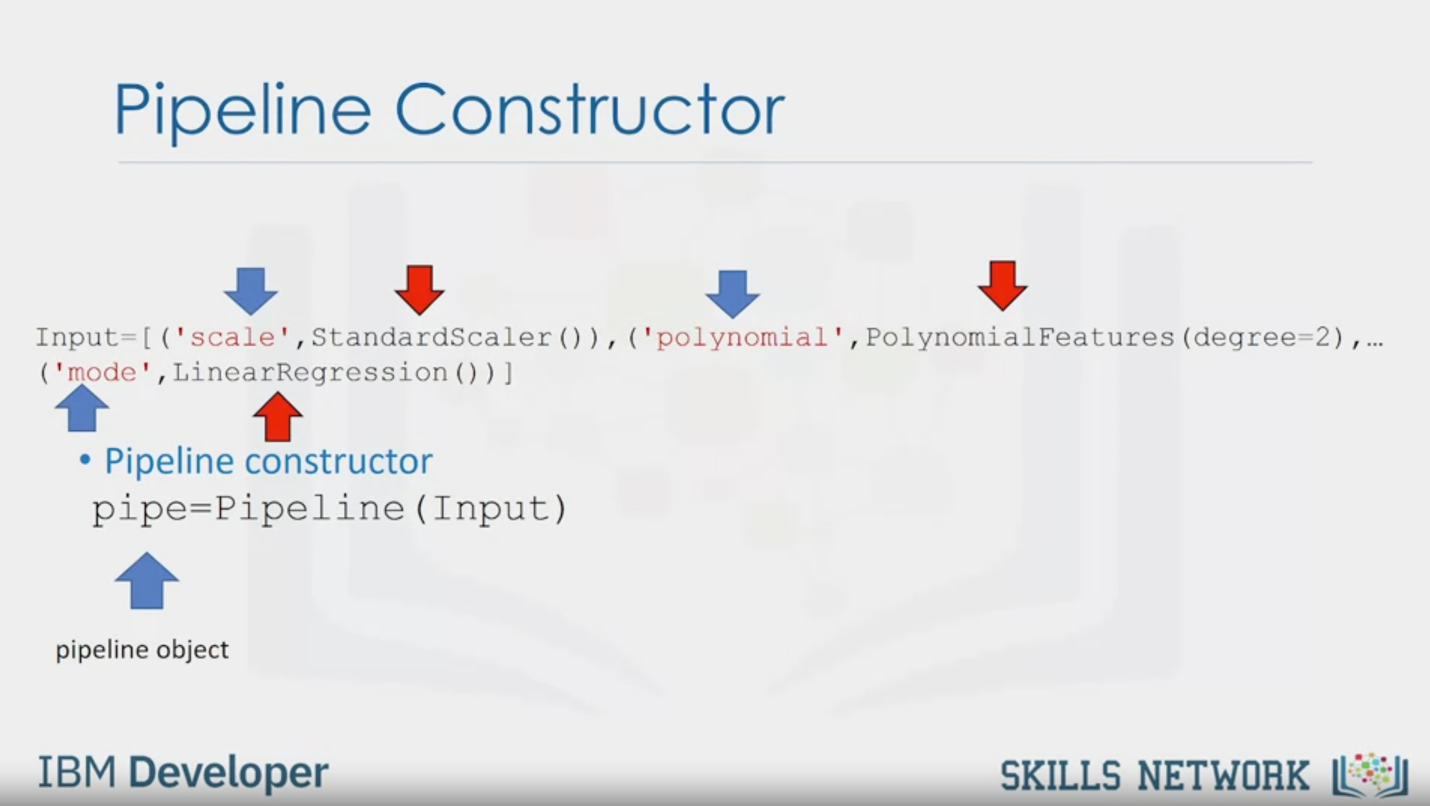
from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import Pipeline

IBM Developer

SKILLS NETWORK



Pipeline Constructor

Input=[ ('scale', StandardScaler () ), ('polynomial', PolynomialFeatures (degree=2) , ...

( 'mode', LinearRegression())]

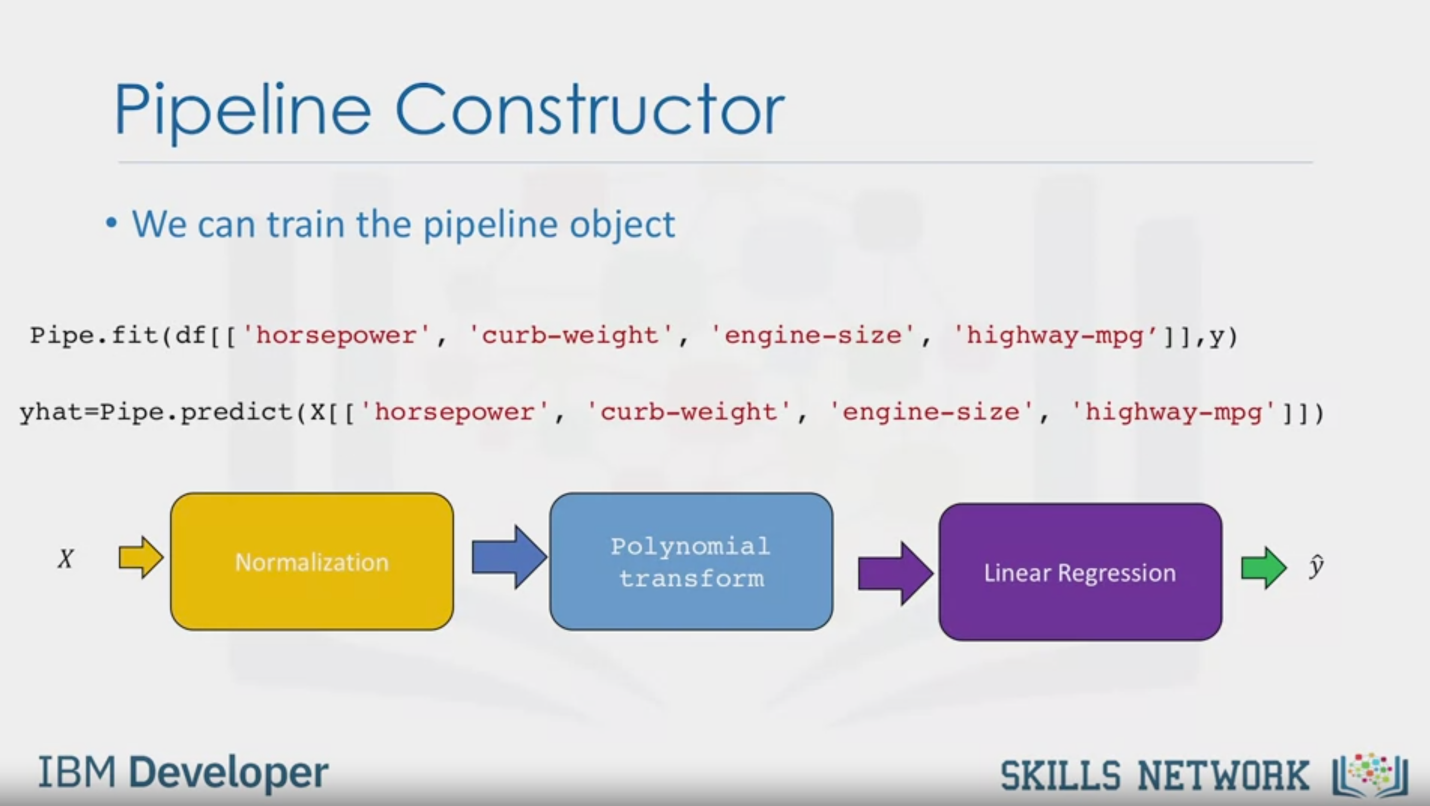
• Pipeline constructor

pipe=Pipeline (Input)

pipeline obiect

IBM Developer

SKILLS NETWORK



Pipeline Constructor

• We can train the pipeline object

Pipe.fit(df[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']],y)

yhat=Pipe.predict (X[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']])

X

Normalization

Polynomial

transform

Linear Regression

IBM Developer

SKILLS NETWORK



SKILLS NETWORK

IBM Developer

In this video, we will cover polynomial regression and pipelines.

What do we do when a linear model is not the best fit for our data?

Let's look into another type of regression model.

The polynomial regression.

We transform our data into a polynomial,

then use linear regression to fit the parameter.

Then we will discuss pipelines.

Pipelines are a way to simplify your code.

Polynomial regression is a special case of the general linear regression.

This method is beneficial for describing curvilinear relationships.

What is a curvilinear relationship?

It's what you get by squaring or setting higher order terms of

the predictor variables in the model transforming the data.

The model can be quadratic,

which means that the predictor variable in the model is squared.

We use a bracket to indicate it as an exponent.

This is a second order polynomial regression,

with a figure representing the function.

The model can be cubic,

which means that the predictor variable is cubed.

This is the third order polynomial regression.

We see by examining the figure that the function has more variation.

There also exists higher order polynomial regressions.

When a good fit hasn't been achieved by second or third order.

We can see in figures how much the graphs change,

when we change the order of the polynomial regression.

The degree of the regression makes

a big difference and can result in a better fit If you pick the right value.

In all cases, the relationship between the variable and the parameter is always linear.

Let's look at an example from our data where we generate a polynomial regression model.

In Python we do this by using the polyfit function.

In this example, we develop a third order polynomial regression model base.

We can print out the model.

Symbolic form for the model is given by the following expression:

negative 1.557 x1 cubed plus 204.8 x one

squared plus 8965 x1 plus 1.37 times 10 to the power of five.

We could also have multi-dimensional polynomial linear regression.

The expression can get complicated.

Here are just some of the terms for a two dimensional second order polynomial.

Numpy's polyfit function cannot perform this type of regression.

We use the preprocessing library in scikit-learn to create a polynomial feature object.

The constructor takes the degree of the polynomial as a parameter.

Then we transform the features into

a polynomial feature with the fit underscore transform method.

Let's do a more intuitive example.

Consider the features shown here.

Applying the method we transform the data,

we now have a new set of features that are

a transformed version of our original features.

As the dimension of the data gets larger,

we may want to normalize multiple features in scikit-learn.

Instead we can use the preprocessing module to simplify many tasks.

For example, we can standardize each feature simultaneously.

We import StandardScaler. We train the object,

fit the scale object,

then transform the data into a new data frame on array x\_scale.

There are more normalization methods available in

the preprocessing library as well as other transformations.

We can simplify our code by using a pipeline library.

There are many steps to getting a prediction.

For example, normalization, polynomial transform, and linear regression.

We simplify the process using a pipeline.

Pipeline sequentially perform a series of transformations.

The last step carries out a prediction.

First we import all the modules we need,

then we import the library pipeline.

We create a list of tuples,

the first element in the tuple contains the name of the estimator model.

The second element contains model constructor.

We input the list in the pipeline constructor.

We now have a pipeline object.

We can train the pipeline by applying the train method to the pipeline object.

We can also produce a prediction as well.

The method normalizes the data,

performs a polynomial transform,

then outputs a prediction.