FDC104: Programming for Data Analysis and Scientific Computing

Lecture 8&9: Linear Model Development

TRƯỜNG ĐẠI HỌC NGOẠI THƯƠNG

Lecture overview

Topics

- Simple and Multiple Linear Regression
- Polynomial Regression and Pipelines
- Model evaluation using visualization
- Measure for evaluation
- Logistic Regression - Classification

Question: How can you determine a fair value for a used car?

Activities

- Hand-on lab: Model Development

Lecture 8 & 9: Linear Model Development

Section 1: Simple Linear Model & Multiple Linear Model

Data Analytics Group

TRƯỜNG ĐẠI HỌC NGOẠI THƯƠNG
Foreign Trade University

Model Development

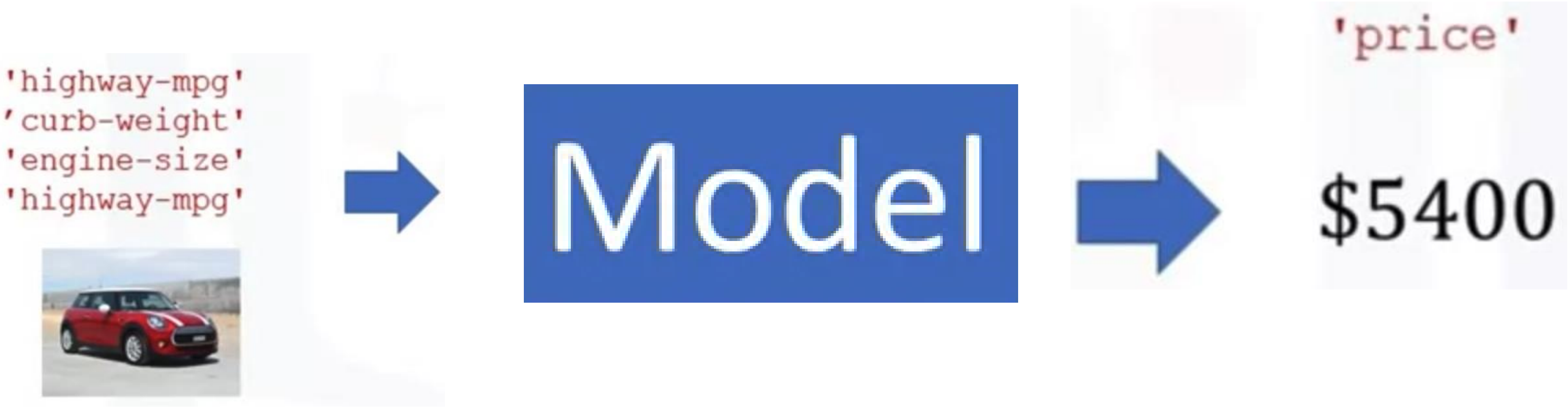
- A model can be simply defined as a mathematical equation used to predict a value given one or more other values

Independent variables or features

dependent variables

Model Development

- Usually the more relevant data we have the more accurate our model is



Simple & Multiple Linear Regression

- Linear regression refers to one independent variable to make a prediction

- Multiple linear regression refers to multiple independent variables to make a prediction

Simple Linear Regression

- The predictor (independent) variable – ***x***
- The target (dependent) variable - ***y***

chỉ sử dụng một biến

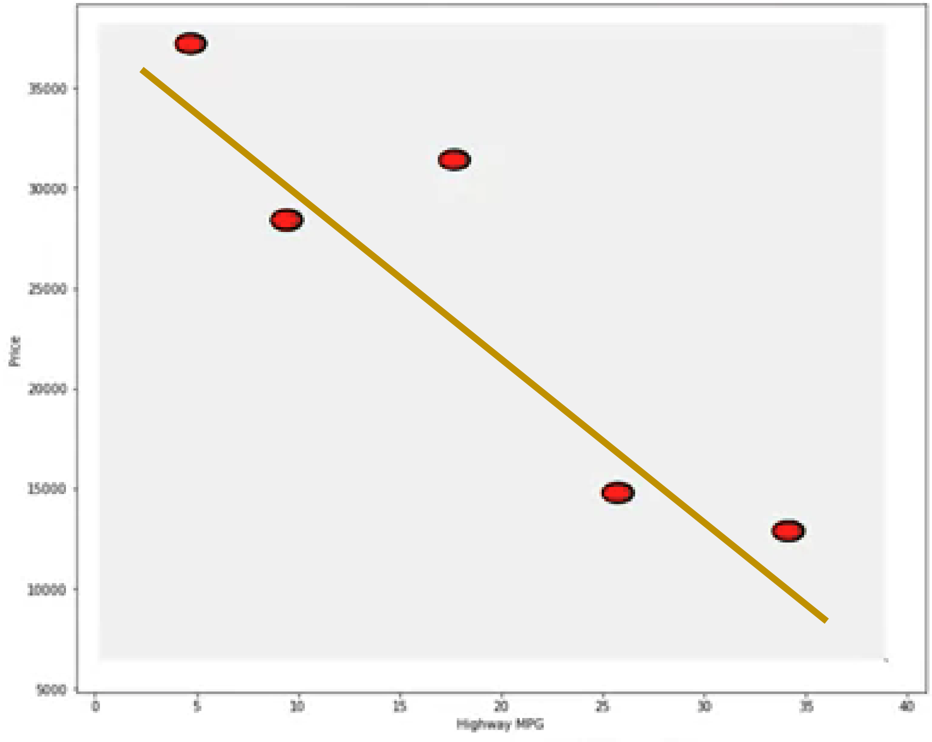
$$y = b_0 + b_1\,x$$

– b_0: **the intercept**

– b_1: **the slope**

Simple Linear Regression: Fit & Predict

thuật toán tốt --> đi qua càng nhiều điểm càng tốt ->> khớp vs dữ liệu trong quá khứ -> dự đoán tốt trong tương lai



$$(b_0\ , b_1)$$

học máy --> học để tìm ra hệ số này

$$\hat{y} = b_0 + b_1\, x$$

có quan sát mới --> x mới ->> muốn dự đoán giá ô tô

Building a Simple Linear Model

1. Import linear_model from 'scikit-learn' library:

```
from sklearn.linear_model import LinearRegression
```

2. Create a Linear Regression Object:

```
lm=LinearRegression()
```

3. We define the predictor variable and target variable:

```
X = df[['highway-mpg']]
Y = df['price']
```

4. Then we fit the model: `lm.fit(X, Y)`

5. We can obtain a prediction with: `Yhat=lm.predict(X)`

or we can view (b_0, b_1):

```
lm.intercept_
lm.coef_
```

Multiple Linear Regression (MLR)

MLR is used to explain the relationship between:

- One contiuous target (Y) variable
- Two or more predictor (X) variables

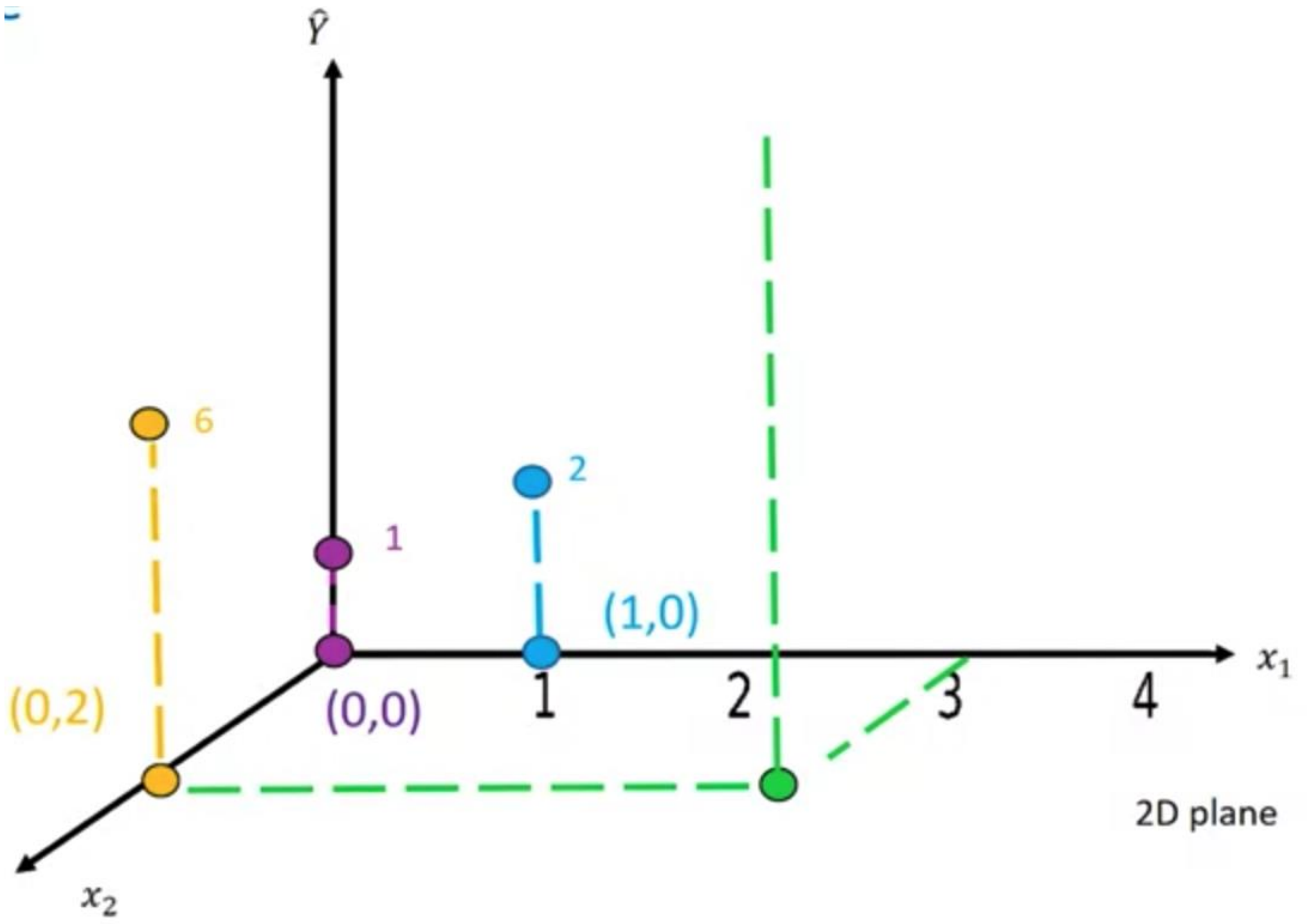
$$\hat{Y} = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + b_4x_4$$

- b_0: **The intercept**
- b_1: the **coefficient** or **parameter** of x_1
- b_2: the **coefficient** or **parameter** of x_2 and so on…

$$\hat{Y} = 1 + 2x_1 + 3x_2$$

n	x_1	x_2
1	0	0
2	0	2
3	1	0
4	3	2

$\hat{Y}$
1
6
2
13



Building a MLR Estimator

1. Create a Linear Regression Object

```
lm=LinearRegression()
```

2. Extract the predictor variables:

```
Z = df[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']]
```

3. Train model: `lm.fit(Z, df['price'])`

4. Obtain a prediction:

```
Yhat=lm.predict(X)
```

x_1	x_2	x_3	x_4
3	5	-4	3
:	:	:	:
2	4	2	-4

Yhat
2
:
3

Building a MLR Estimator

6. Find the intercept (b_0): `lm.intercept_`

   -15678.742628061467

   Find the coefficients: `lm.coef_`

   array([52.65851272 ,4.69878948,81.95906216 , 33.58258185])

Finally, the estimated linear model is:

Price = -15678.74 + (52.66) * **horsepower** + (4.70) * **curb-weight** + (81.96) * **engine-size** + (33.58) * **highway-mpg**

Activity – Hand-on Lab (~30 mins)

- Linear model development

Lecture 8 & 9: Linear Model Development

Section 2: Polynomial Regression

Polynomial Regression

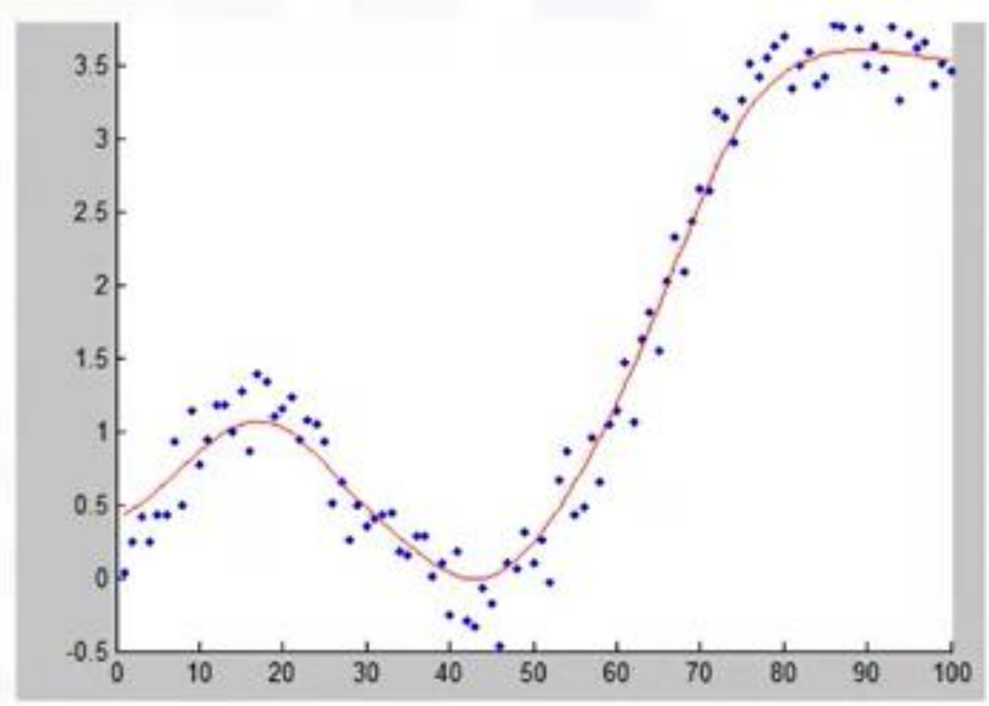
(dành cho mối quan hệ phi tuyến)

A special case of the general linear regression model:

- Useful for describing curvilinear relationship

Curvilinear relationships:

- Squaring or setting higher-order terms of the predictor variables

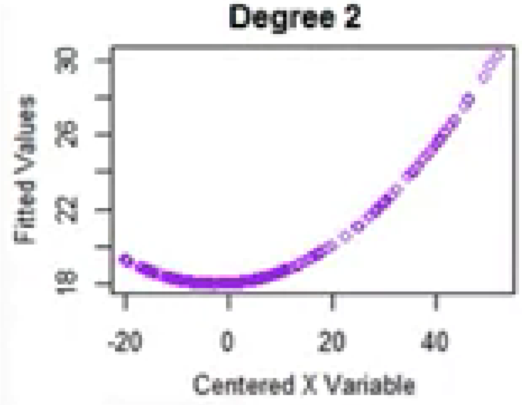


Polynomial Regression

- Quadratic – 2nd order

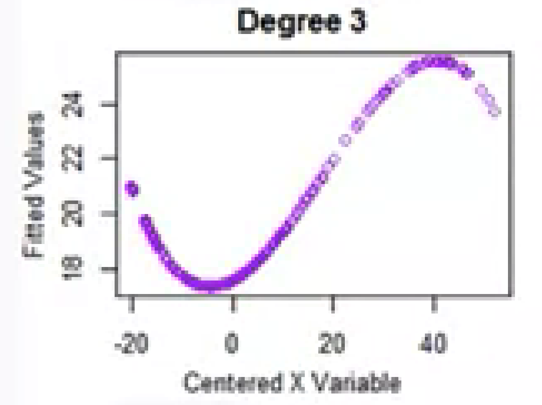
$$\hat{Y} = b_0 + b_1\, x_1 + b_2(\, x_1)^2$$

thay vì bậc 1 thì tạo ra bậc 2,3,4,5... -> ko có công thức đánh số bậc bao nhiêu
bậc càng cao thì mô hình càng phức tạp



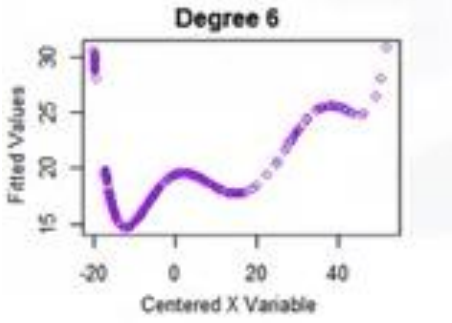
- Cubic – 3rd order

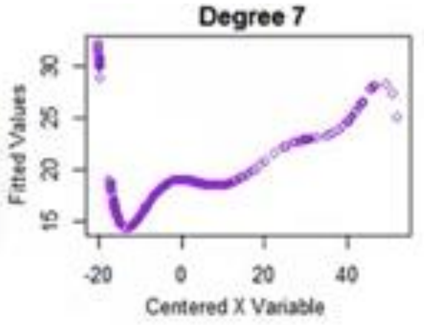
$$\hat{Y} = b_0 + b_1\, x_1 + b_2(\, x_1)^2 + b_3(\, x_1)^3$$



- Higher order

$$\hat{Y} = b_0 + b_1\, x_1 + b_2(\, x_1)^2 + b_3(\, x_1)^3 + ..$$





Polynomial Regression

Polynomial regression with more than one dimension

- We can also have multi dimensional polynomial linear regression

$$\hat{Y} = b0 + b1\,X_1 + b2\,X_2 + b3\,X_1\,X_2 + b4(\,X_1)^2 + b5(\,X_2)^2 + ..$$

Polynomial features

Polynomial regression: linear regression with polynomial features

Polynomial Regression

Creating polynomial features

- We can use “preprocessing” library in scikit-learn to transform variables to polynomial features

```
from sklearn.preprocessing import PolynomialFeatures

pr=PolynomialFeatures(degree=2, include_bias=False)

x_polly=pr.fit_transform(x[['horsepower', 'curb-weight']])
```

Polynomial Regression

Example:

X_1	X_2
1	2

```
pr=PolynomialFeatures(degree=2,include_bias=False)
pr.fit_transform([[1,2]])
```

X_1	X_2	X_1X_2	${X_1}^2$	${X_2}^2$
1	2	(1) 2	1	$(2)^2$

1	2	2	1	4

Polynomial Regression

Pre-processing

- When creating polynomial features, the dimension of the data gets larger → Need to consider normlize the features (before polynomial transform)
- We can easily use “preprocessing” library to normalize multiple features:

```
from sklearn.preprocessing import StandardScaler
SCALE=StandardScaler()
SCALE.fit(x_data[['horsepower', 'highway-mpg']])
x_scale=SCALE.transform(x_data[['horsepower', 'highway-mpg']])
```

Pipelines

There are many steps to build a model

We can simplify the process using a pipeline

- **Pipeline**: sequentially perform a series of transformations and training/predicting

Pipelines

Building a pipeline

1. Import libraries:

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
```

2. Create pipeline constructor:

```
Input=[('scale',StandardScaler()),('polynomial',PolynomialFeatures(degree=2),…
('mode',LinearRegression())]
```

3. Build pipeline:

```
pipe=Pipeline(Input)
```

Pipelines

- We can now train the pipeline with our data:

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

- We can also make predictions with the pipeline:

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

Activity – Hand-on Lab
(~30 mins)

- Polynomial Regression and Pipeline

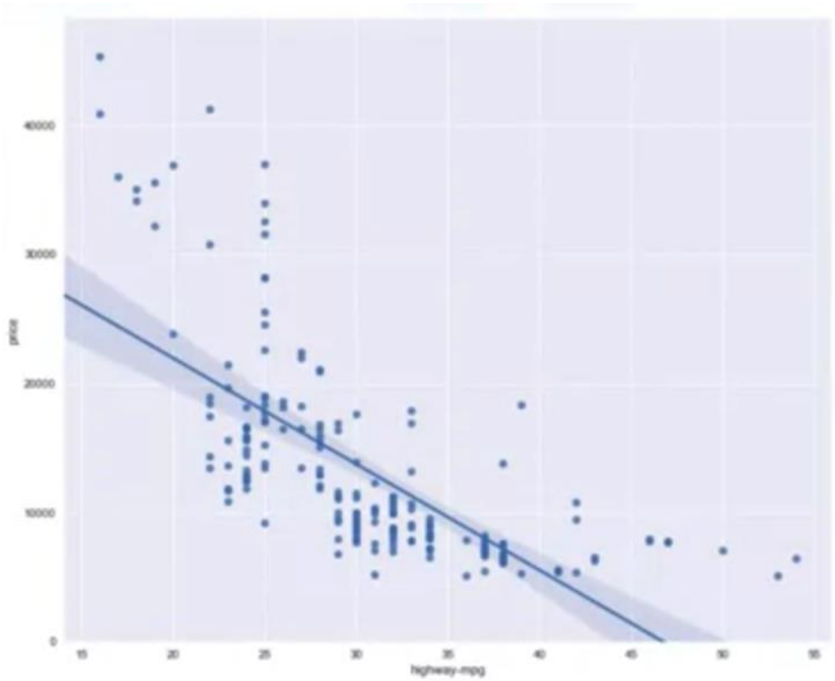
Lecture 8 & 9: Linear Model Development

Section 3: Model Evaluation using Visualization

Regression Plot

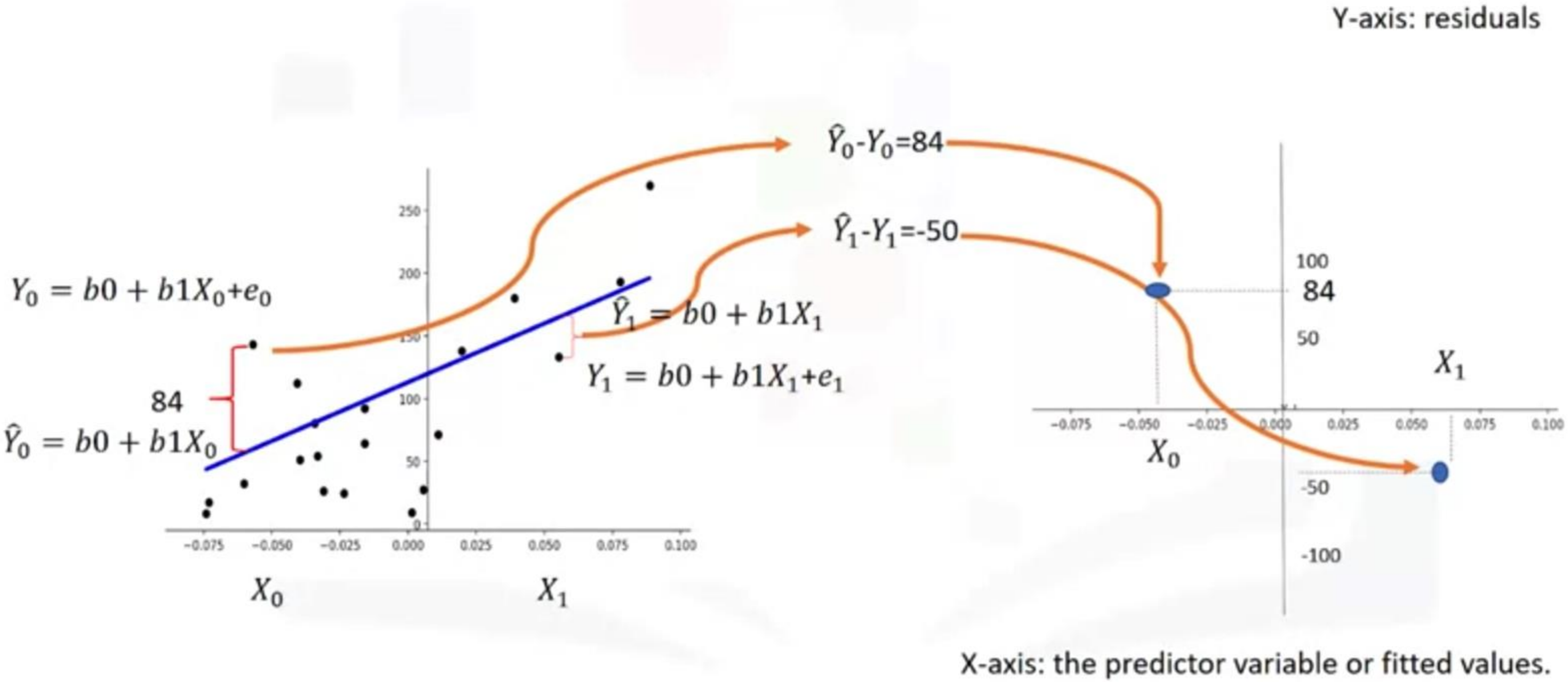
Regression Plot – Gives us a good estimate of:

- The relationship between two variables
- The strength of the correlation
- The direction of the relationship (positive or negative)

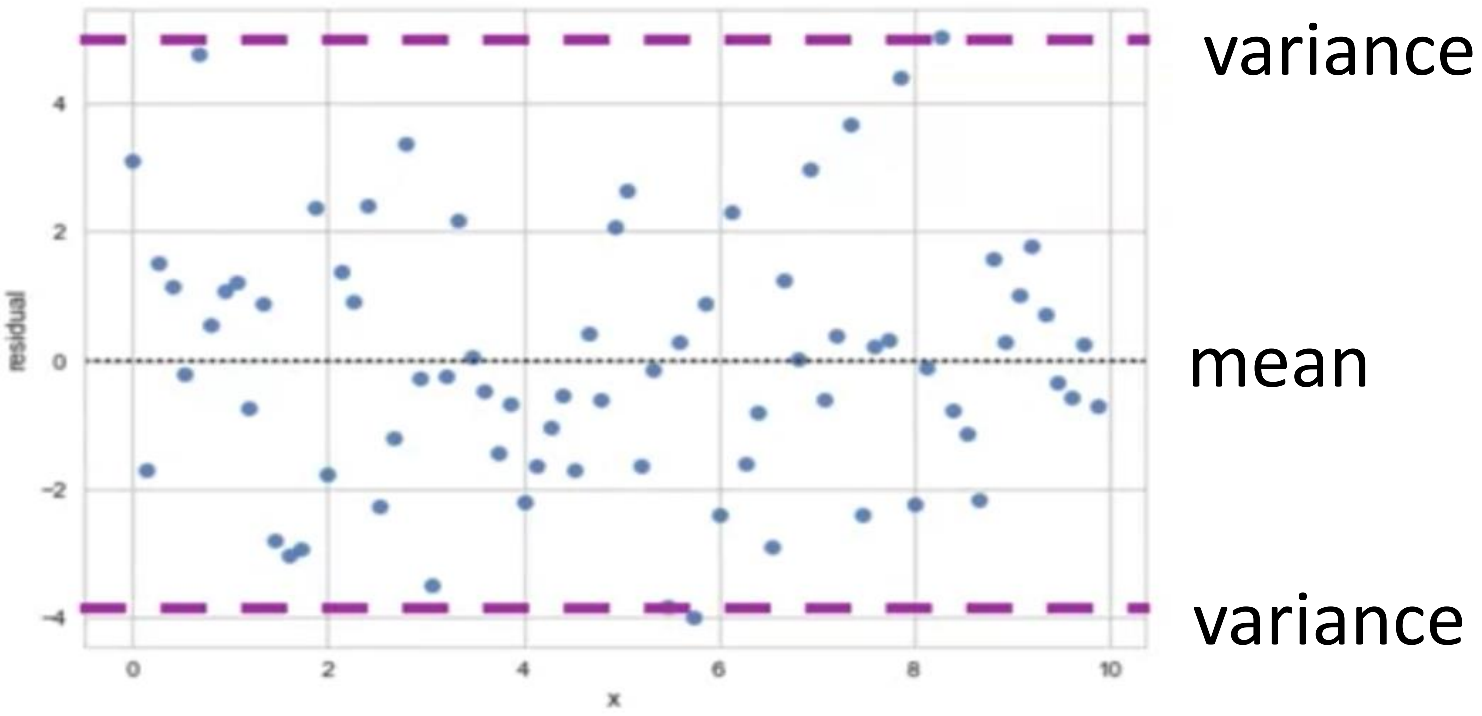


Residual Plot

Residual Plot – Represents the error between the actual value and the predicted value:

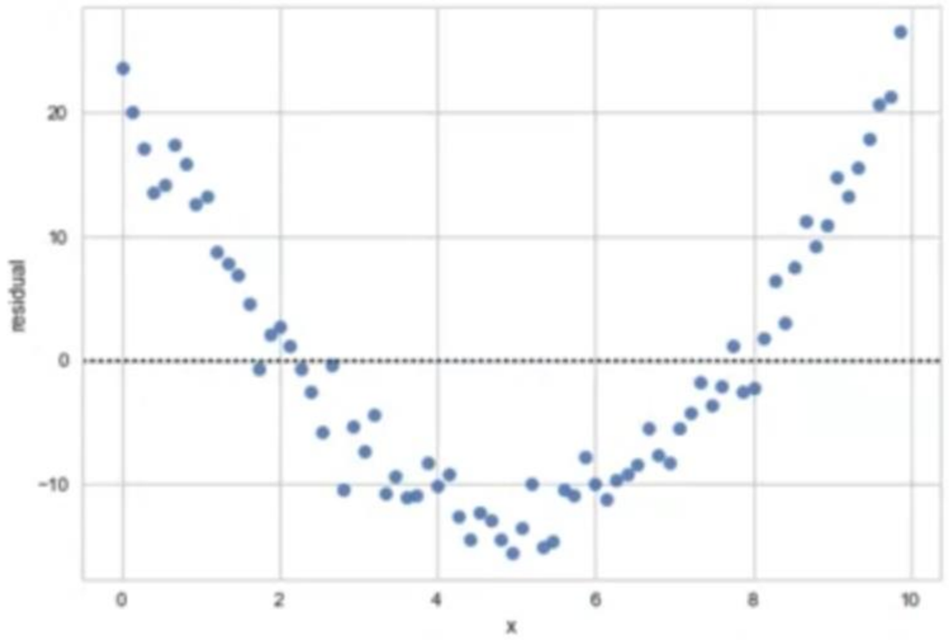


Residual Plot



The residuals randomly spread out around x-axis then a linear model is appropriate

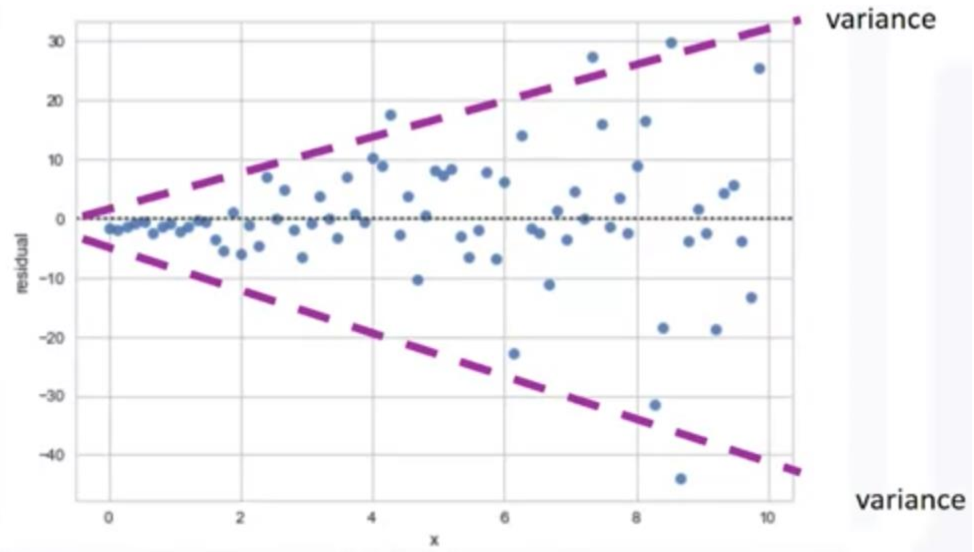
Residual Plot



The **residuals** are not randomly spread out around the x-axis:

- The linear model is not appropriate
- Nonlinear model may be more appropriate

Residual Plot



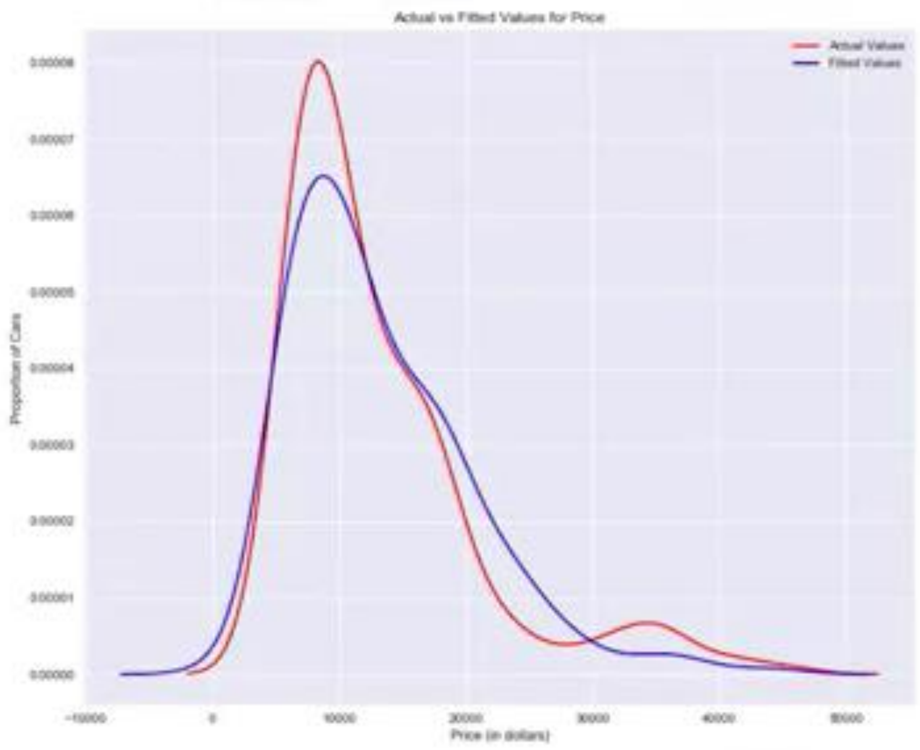
The **residuals** are not randomly spread out around the x-axis, variance appears to change with x axis:

- The linear model is not appropriate
- Nonlinear model may be more appropriate

Distribution Plots

Compare the distributions:

- The predited values that result from the model
- The actual values



Activity – Hand-on Lab (~30 mins)

- Model evaluation with visualization

Lecture 8 & 9: Linear Model Development

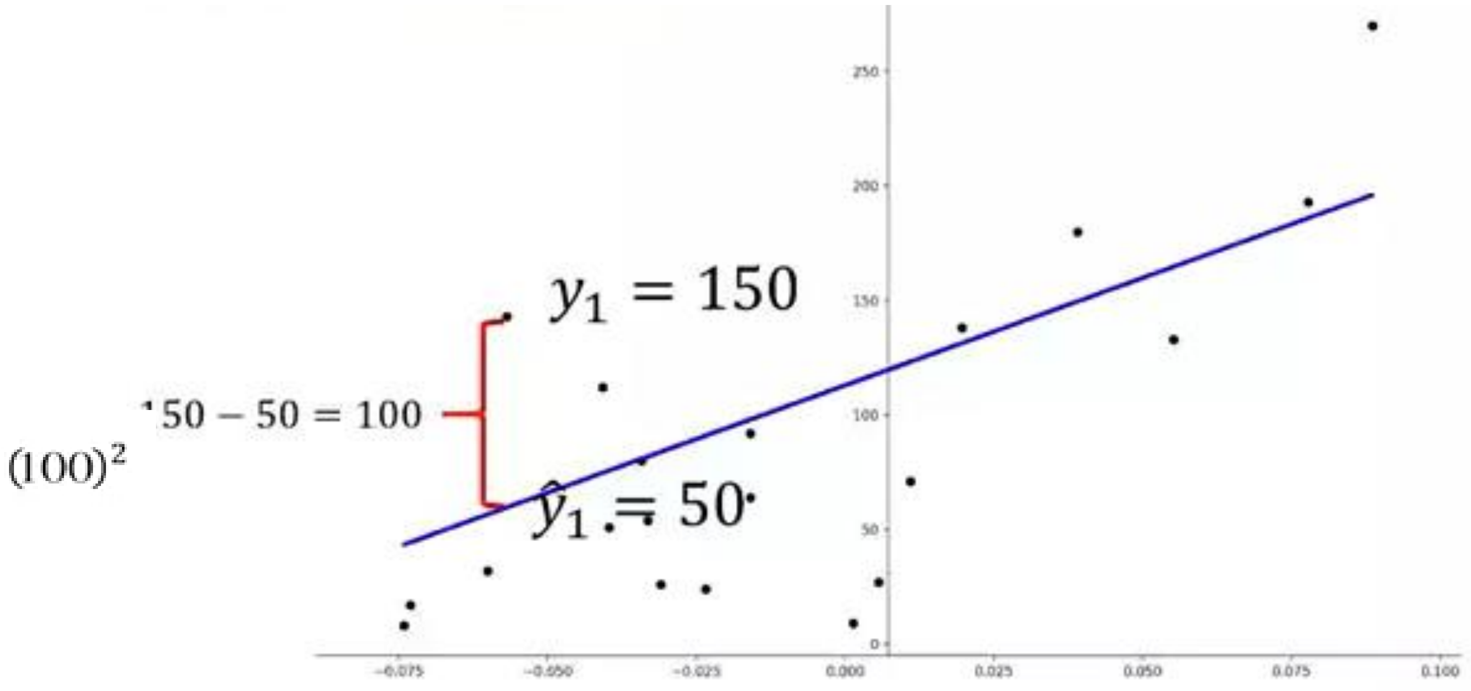
Section 4: Measures for In-sample evaluation

Measures for In-sample Evaluation

- A way to numerically determine how good the model fits on dataset
- Two important measures to determine the fit of a model:
  - **Mean Squared Error (MSE)**
  - **R-squared (R^2)**

Mean Square Error (MSE)

- Example



$$MSE = \frac{sum\ all\ squared\ errors}{number\ of\ samples}$$

Mean Square Error (MSE)

- In python we can easily measue the MSE as follows

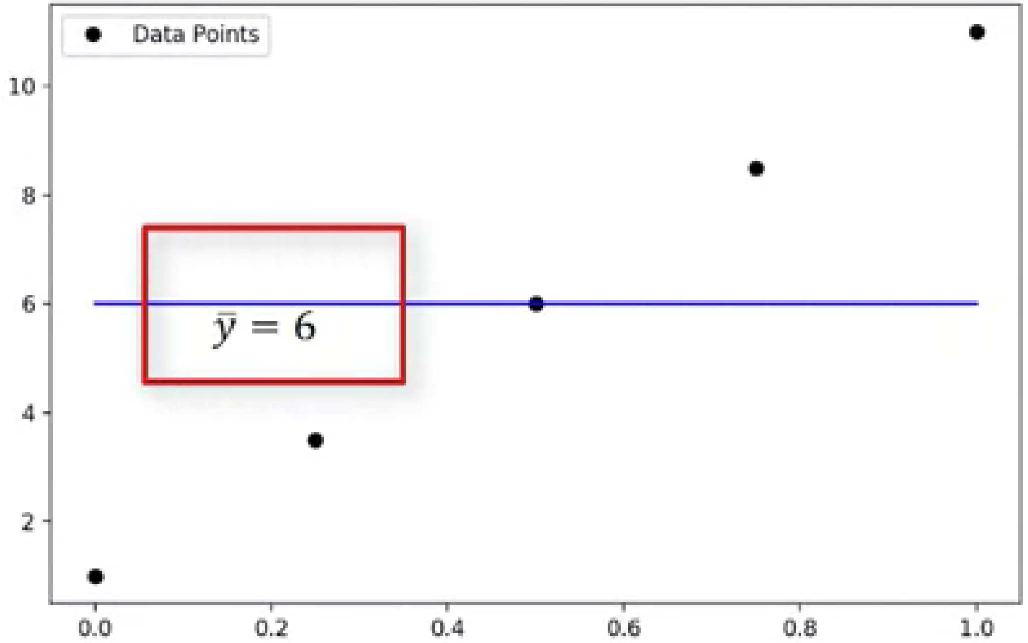
```
from  sklearn.metrics import mean_squared_error

mean_squared_error(df['price'],Y_predict_simple_fit)


3163502.944639888
```

R-squared/ R^2

- A measure to determine how close the data is to the fitted regression line
- Think about as comparing a regression model to the mean of the data points



R-squared/ R^2

$$R^2 = \left(1 - \frac{\text{MSE of regression line}}{\text{MSE of the average of the data}}\right)$$

- For the most part, R^2 take values between 0 and 1
- Good model: R^2 closes to 1
- If R^2 < 0, it may be caused by over-fitting

R-squared/ R^2

- Generally the values of the R-squared are between 0 and 1
- We can calculate R-squared as follows

```
X = df[['highway-mpg']]
Y = df['price']

lm.fit(X, Y)


lm.score(X,y)
0.496591188
```

Activity – Hand-on Lab (~30 mins)

- Measures for In-sample evaluation

Lecture 8 & 9: Linear Model Development

Section 5: Logistic Regression for Classification

DATAPOT
Data Analytics Group

TRƯỜNG ĐẠI HỌC NGOẠI THƯƠNG
Foreign Trade University

Classification

- Up to this point, the methods we have seen have centered around modeling and the prediction of a **quantitative** response variable (ex, the used cars' price, etc). Linear **regression** perform well under these situations

- When the response variable is **categorical**, then the problem is no longer called a regression problem but is instead labeled as a **classification problem**.

- The goal is to attempt to classify each observation into a category (aka, class or cluster) defined by Y, based on a set of predictor variables X.

Example: Heart Data

response variable ***Y*** is Yes/No

Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	Ca	Thal	AHD
63	1	typical	145	233	1	2	150	0	2.3	3	0.0	fixed	No
67	1	asymptomatic	160	286	0	2	108	1	1.5	2	3.0	normal	Yes
67	1	asymptomatic	120	229	0	2	129	1	2.6	2	2.0	reversable	Yes
37	1	nonanginal	130	250	0	0	187	0	3.5	3	0.0	normal	No
41	0	nontypical	130	204	0	2	172	0	1.4	1	0.0	normal	No

Binary Classification

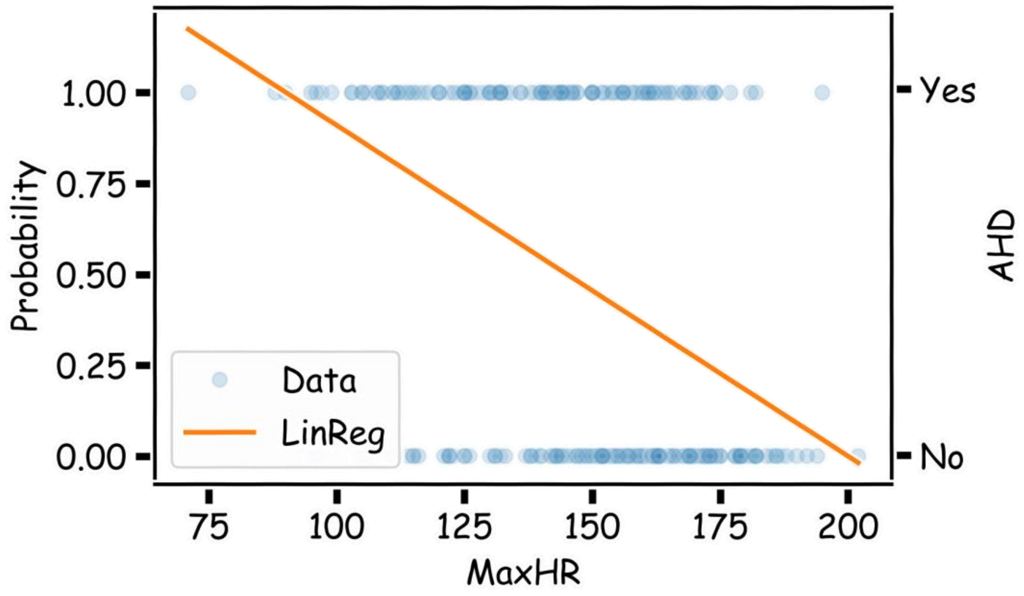
- The simplest form of classification is when the response variable y has only two categories, and then an ordering of the categories is natural. For our example, a patient in the ICU could be categorized as having [atherosclerotic] heart disease (AHD) or not (note, the $y = 0$ category is a "catch-all" so it would involve those patients with lots of other diseases or diagnoses):

$$y = \begin{cases} 1 \text{ if patient has heart disease} \\ 0 \text{ otherwise.} \end{cases}$$

- Linear regression could be used to predict y directly from a set of covariates (like sex, age, resting HR, etc.), and if $\hat{y} \geq 0.5$, we could predict the patient to have AHD and predict not to have heart disease if $\hat{y} < 0.5$.

Binary Classification

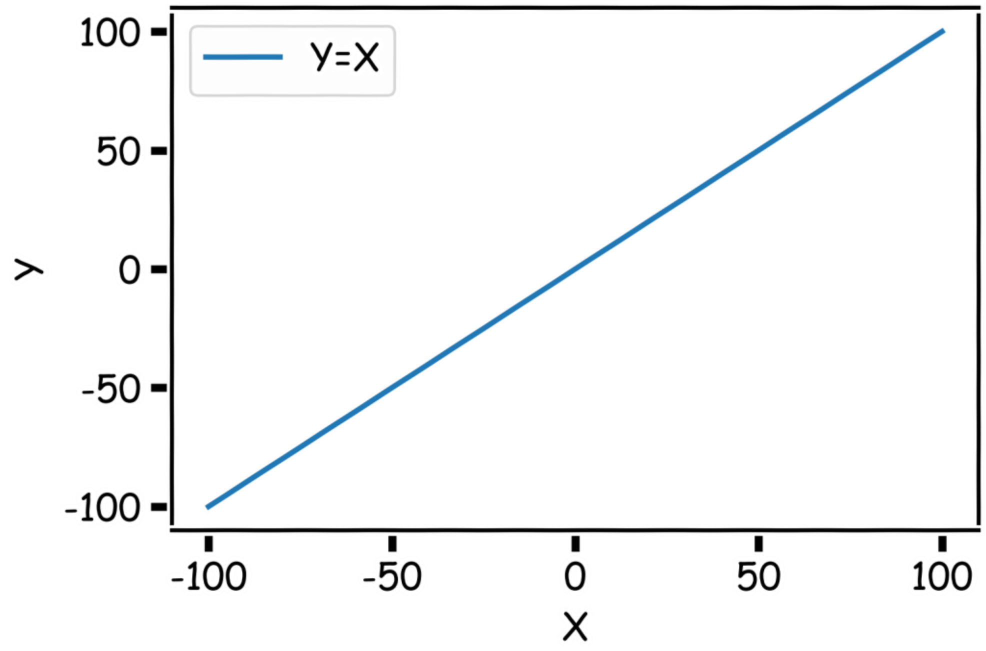
What could go wrong with this linear regression model ?



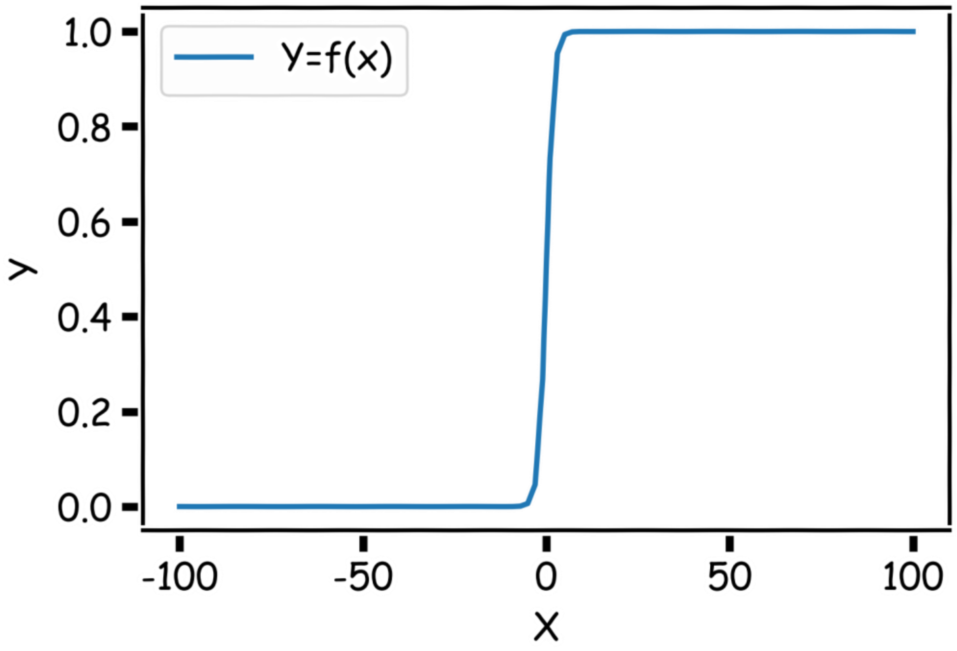
The main issue is you could get non-sensical values for y. Since this is modeling $P(y = 1)$, values for $\hat{y}$ below 0 and above 1 would be at odds with the natural measure for y. Linear regression can lead to this issue.

Logistic regression curve

Think of a function that would do this for us

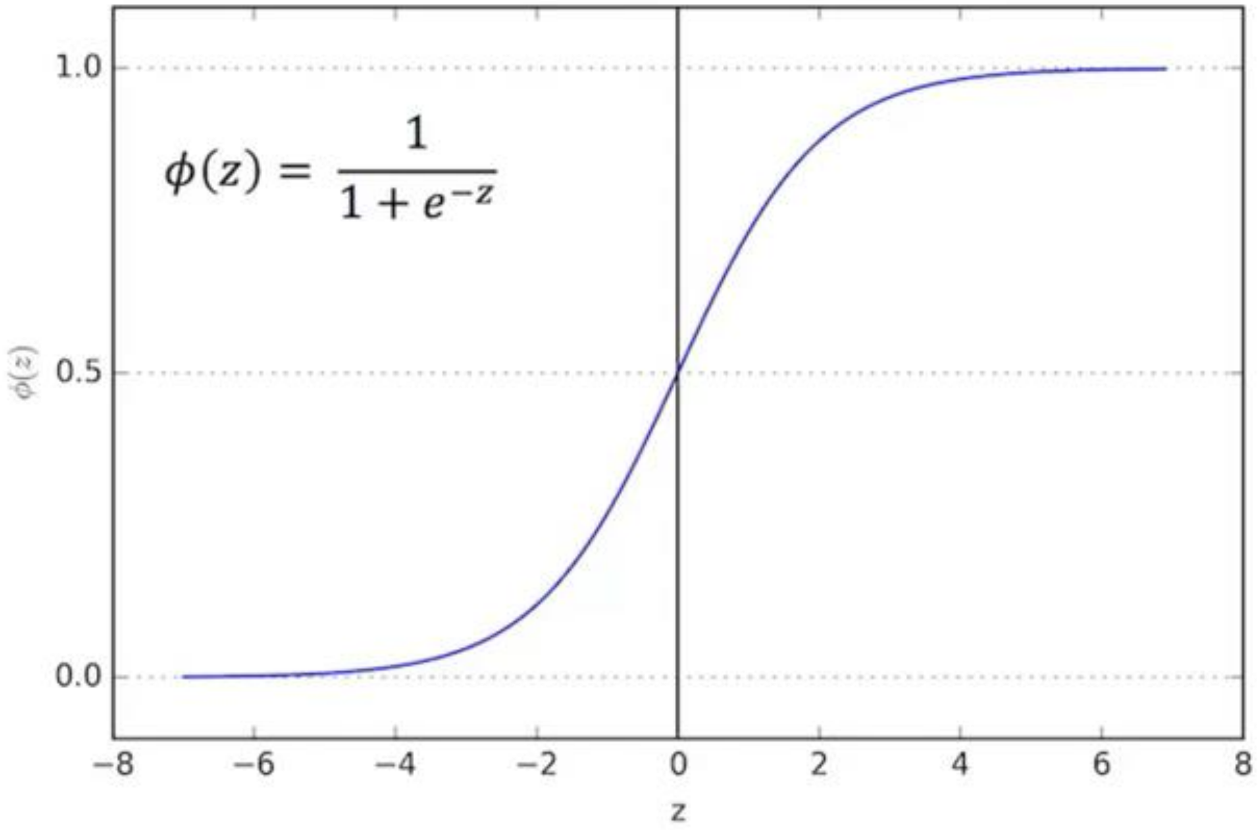


$$Y = f(x)$$



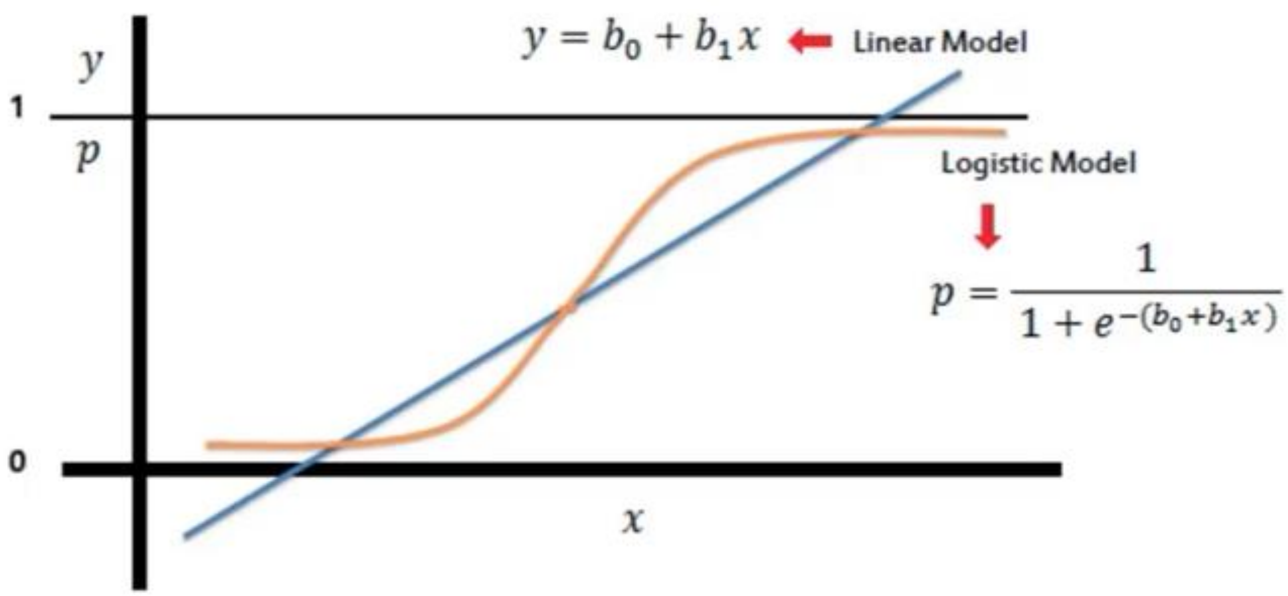
Sigmoid Function

The Sigmoid (aka Logistic) Function takes in any value and outputs it to be between 0 and 1.



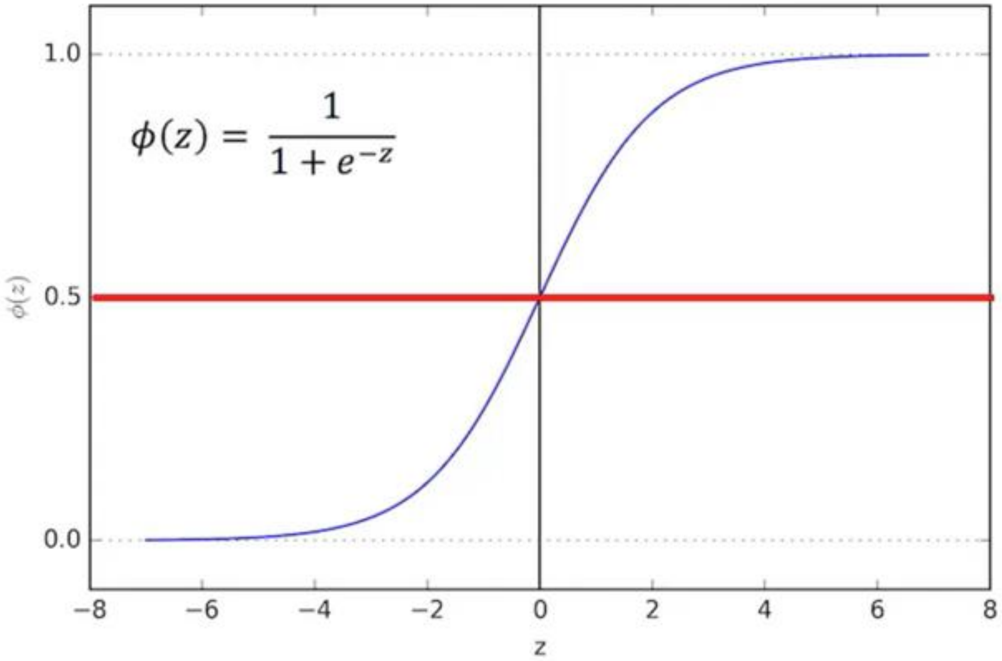
Logistic Regression

- This means we can take our Linear Regression solution and place it into the Sigmoid Function



Logistic Regression

- We can set a cutoff point at 0.5, anything below it results in class 0, anything above is class 1.
- Based on this probability we assign a class for the prediction.



Classification Model Evaluation

- After we train a logistic regression model, we need to evaluate our model's performance
- We use a confusion matrix to evaluate classification models

n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

Example: Test for presence of disease
NO = negative test = False = 0
YES = positive test = True = 1

Confusion Matrix

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

Basic Terminology:

- True Positives (TP)
- True Negatives (TN)
- False Positives (FP)
- False Negatives (FN)

Accuracy:

- Overall, how often is it correct?
- (TP + TN) / total = 150/165 = 0.91

Misclassification Rate (Error rate):

- Overall, how often is it wrong?
- (FP + FN) / total = 15/165 = 0.09

Activity – Hand-on Lab
(~30 mins)

- Logistic Regression

Lecture 8 & 9: Linear Model Development

Wrap-up

Summary

In summary, in this lecture, you learned:

- Linear Regression
- Polynomial Regression
- Logistic Regression for Classification Problem
- Model evaluation using visualization
- Measure for evaluation

Thank You !

TRƯỜNG ĐẠI HỌC NGOẠI THƯƠNG
Foreign Trade University