In this module, you will explore two essential regression techniques used in machine learning—linear and logistic regression. You'll explain the role of regression in predicting outcomes, describe the differences between simple and multiple linear regression, and apply both using scikit-learn on real-world data. You will also interpret how polynomial and non-linear regression models capture complex patterns. The module introduces logistic regression as a classification method and guides you in training and testing classification models effectively. To support your learning, you'll receive a Cheat Sheet: Linear and Logistic Regression that summarizes key concepts, formulas, and use cases.

Learning Objectives

- Explain the purpose of regression analysis in predicting continuous outcomes.
- Describe how simple linear regression works and when it is appropriate to apply it.
- Apply simple linear regression using scikit-learn to train and test a model on real data
- Differentiate multiple linear regression from simple linear regression based on input features and use cases.
- Apply multiple linear regression using scikit-learn to train and test a model on real data
- Interpret the behavior of polynomial and non-linear regression models in capturing complex relationships in data.
- Explain the use of logistic regression for classification problems and how it differs from linear regression.
- Apply logistic regression to classify real-world data using scikit-learn

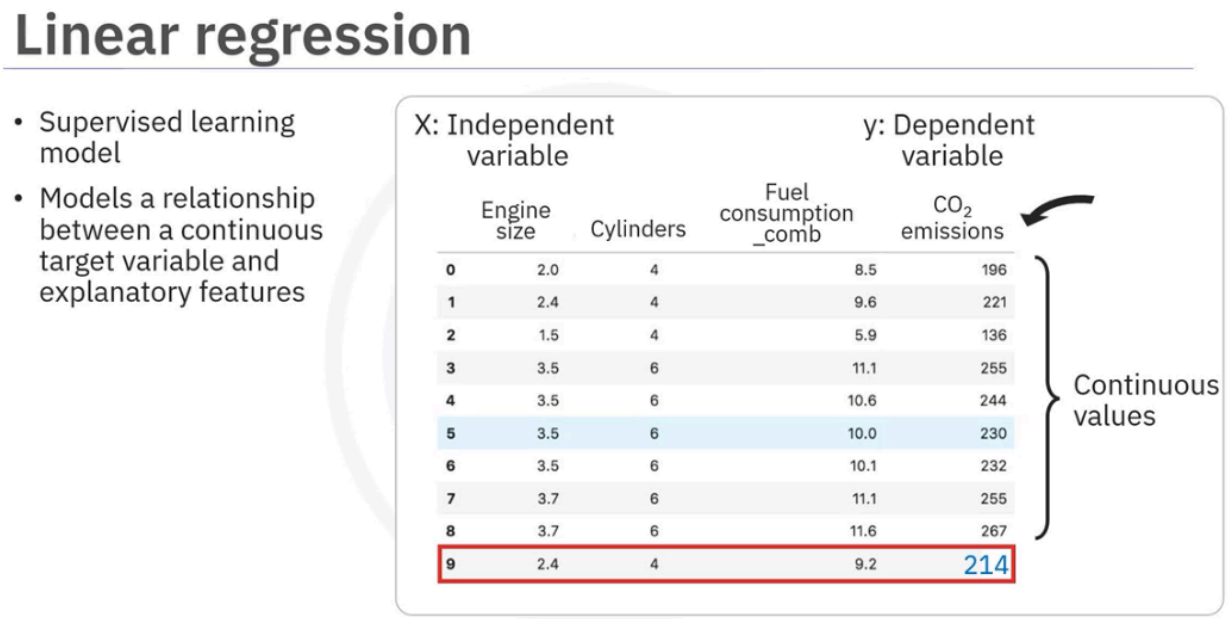
Here's a clear, structured summary of your **Introduction to Regression** script, cleaned of repetition and highlighting the key learning points:

🎯 Learning Objectives

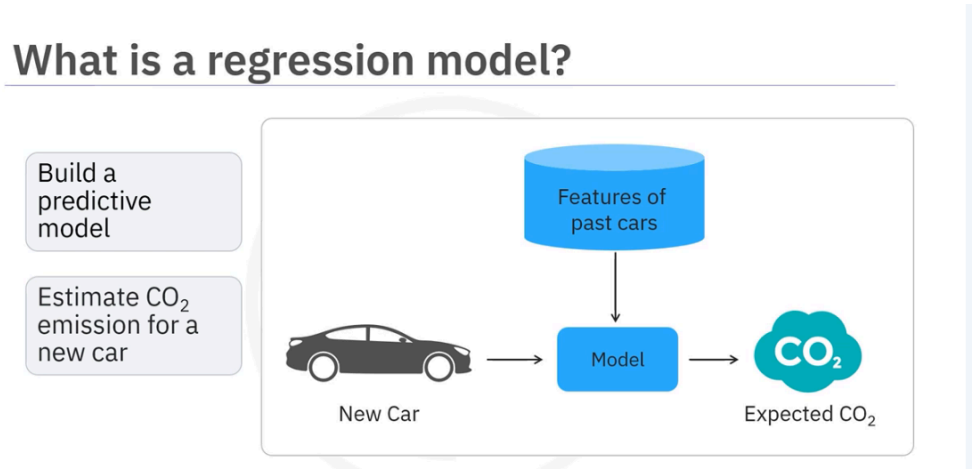
After watching this video, you will be able to:

- Define **regression**
- Compare **simple** and **multiple regression**
- Explain **applications of regression**

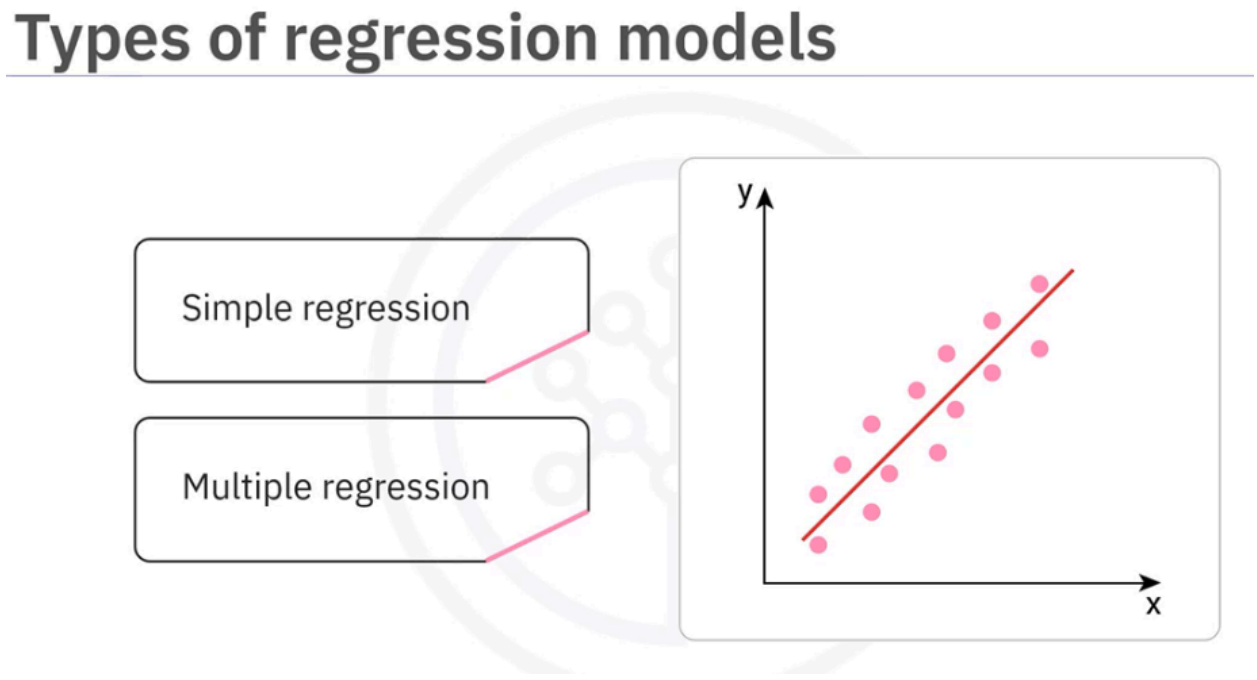
📌 What is Regression?



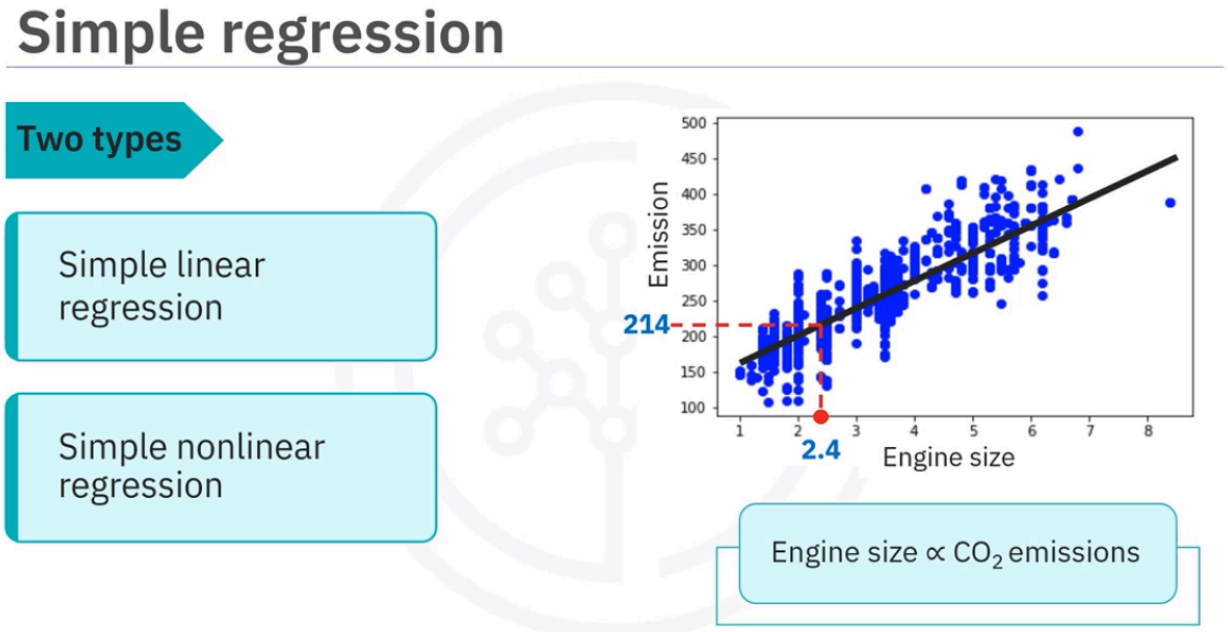
- A type of **supervised learning** model.
- Models the relationship between a **continuous target variable** and **explanatory features**.
- Example: Predicting **CO2 emissions** of cars using features like engine size, number of cylinders, and fuel consumption.



📊 Types of Regression



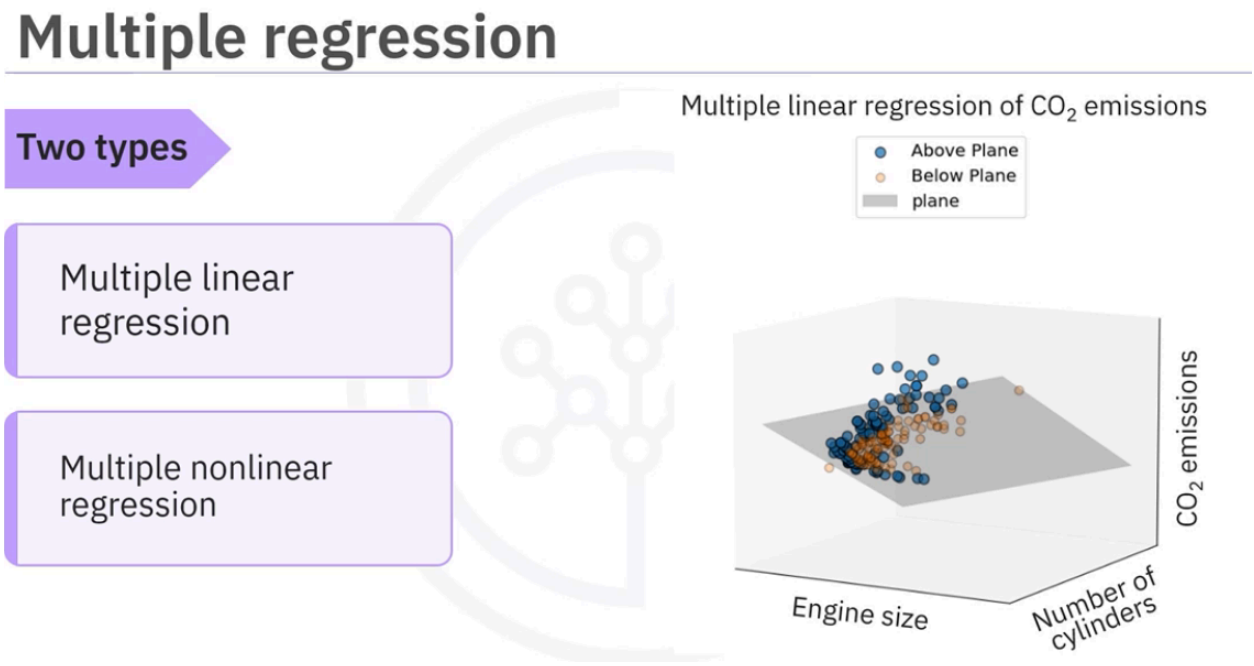
1. **Simple Regression**



   - Uses **one independent variable** to predict a dependent variable.
   - Can be **linear** (straight-line relationship) or **nonlinear**.

- Example: Predicting CO2 emission from **engine size** only.

2. **Multiple Regression**

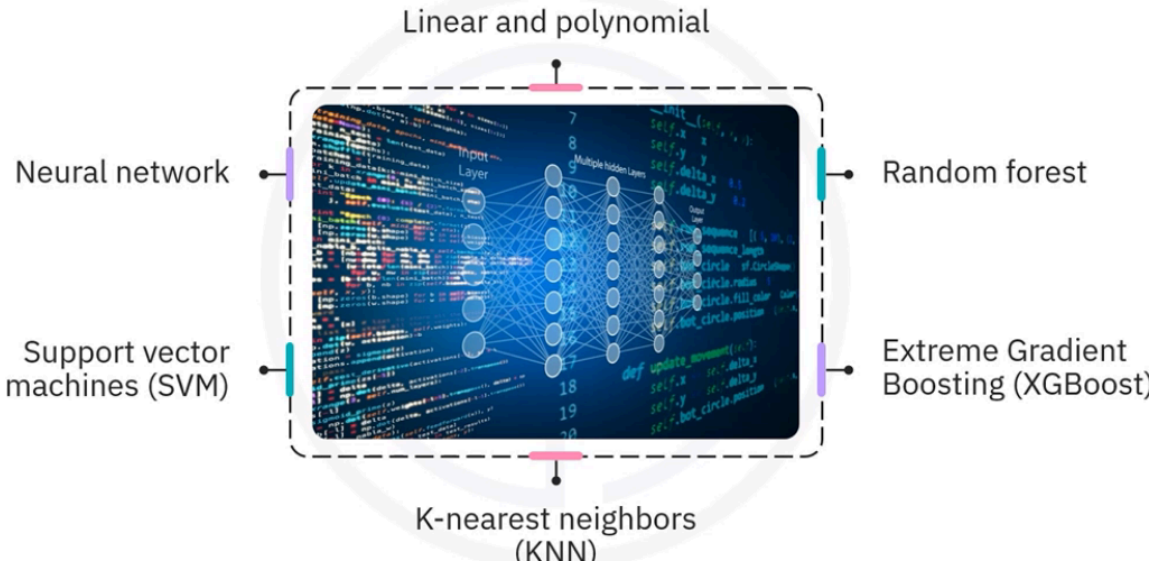


- Uses **two or more independent variables**.
- Can also be linear or nonlinear.
- Example: Predicting CO2 emission using **engine size + number of cylinders**.

🌍 Applications of Regression

- **Sales Forecasting** → Predicting a salesperson's yearly sales.
- **Real Estate** → Estimating house prices.
- **Predictive Maintenance** → Forecasting when a machine will fail.
- **Income Prediction** → Based on education, work hours, experience, etc.
- **Weather & Environment** → Estimating rainfall or wildfire severity.
- **Public Health** → Predicting disease spread or likelihood of developing conditions.

🧠 Regression Algorithms



- **Classical methods** → Linear Regression, Polynomial Regression
- **Modern ML models** → Random Forest, XGBoost, KNN, Support Vector Machines, Neural Networks

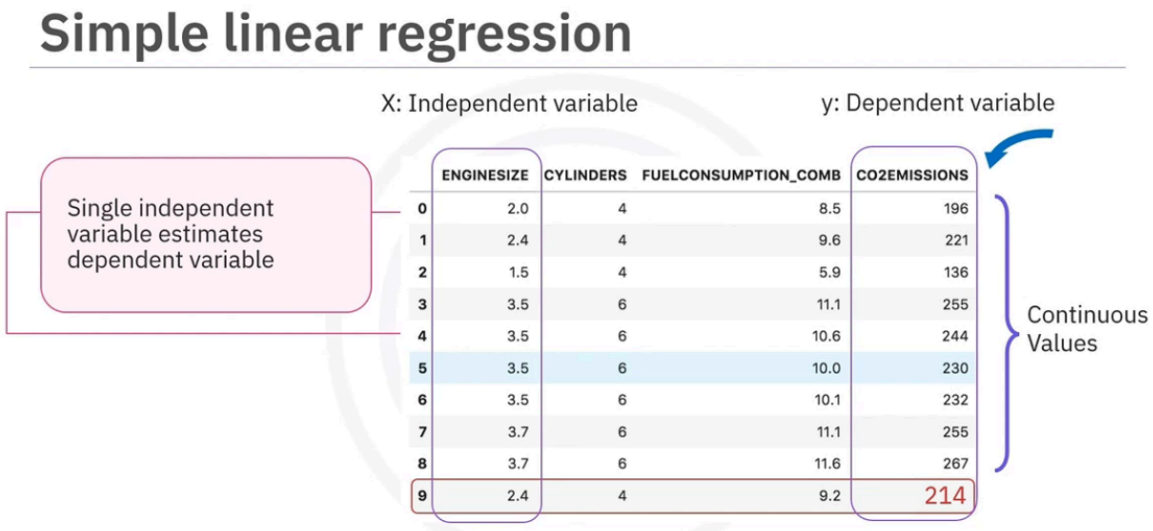
---

✅ **Key Takeaway:**
Regression is a powerful supervised learning technique for predicting continuous outcomes. It can be simple or multiple, linear or nonlinear, and is widely used across industries such as **finance, healthcare, retail, environment, and public health**.

Here's a structured and simplified **summary of your "Introduction to Simple Linear Regression" video script**, focusing on the key points while keeping it clear and engaging

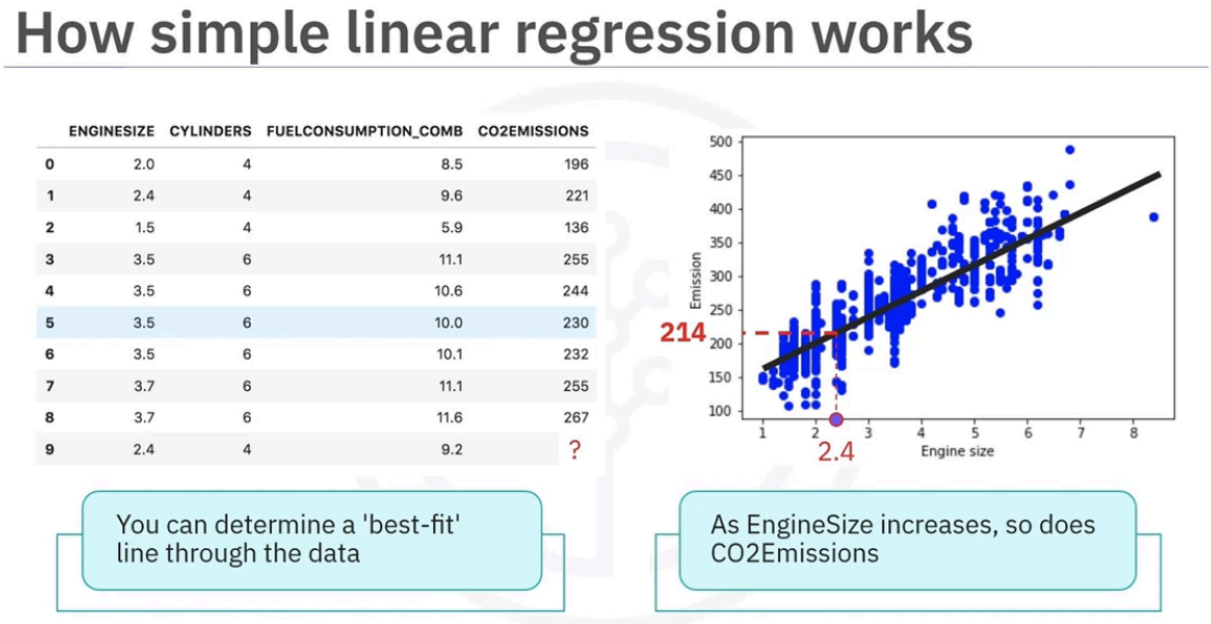
---

📌 What is Simple Linear Regression?

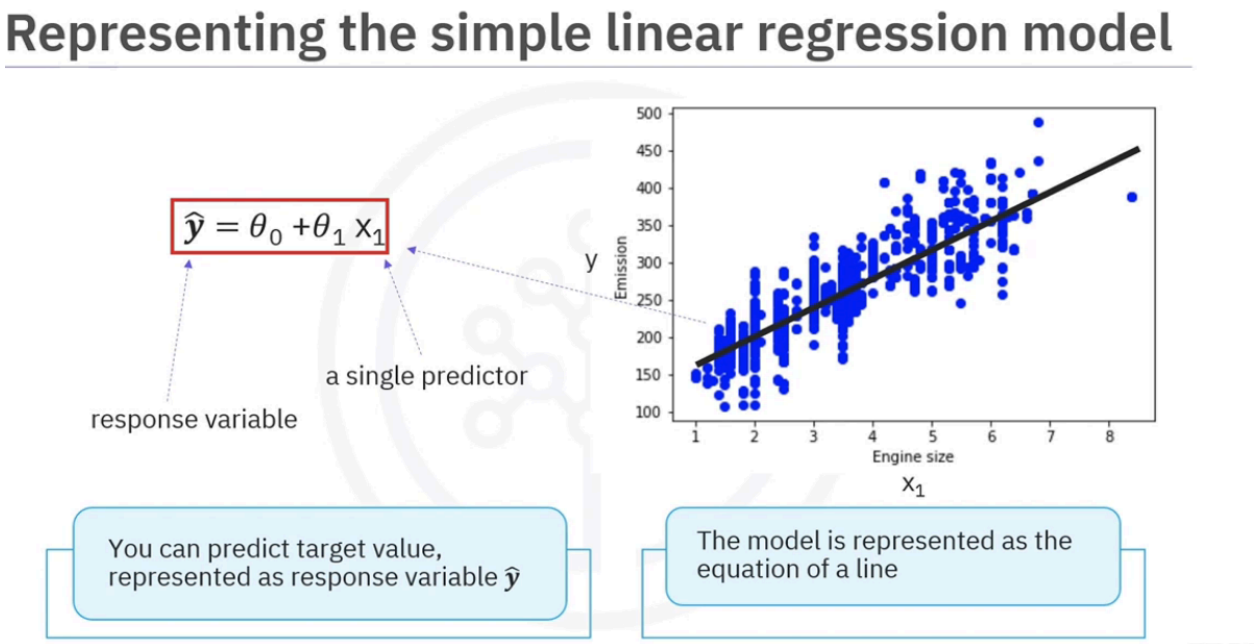


- A **supervised learning** technique.
- Models a **linear relationship** between one independent variable (**x**) and one dependent (target) variable (**y**).
- Example: Predicting **CO2 emissions** of a car based on its **engine size**.

📊 How It Works



1. **Scatter Plot**
    - Engine size (x) vs. CO2 emissions (y).
    - Shows correlation between variables.
2. **Best-Fit Line (Regression Line)**



3.
    - Equation:
      y^=θ0+θ1x\hat{y} = \theta_0 + \theta_1 x
    - **θ₀ (theta zero):** y-intercept (bias)
    - **θ₁ (theta one):** slope (coefficient)
4. **Prediction Example**
    - Engine size = 2.4 → Predicted CO2 = 214

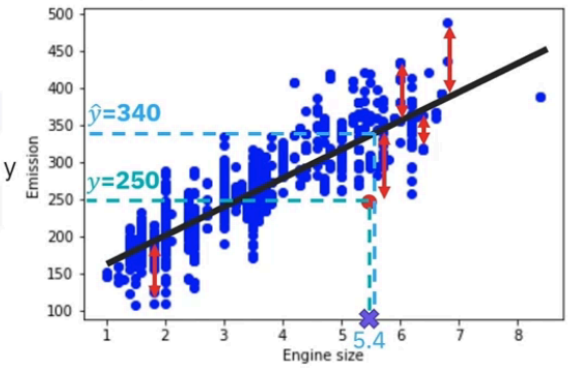
📐 Residuals & Error

Finding the best fit

Given a car with EngineSize $X_1 = 5.4$

The actual CO2 emission is 250

The predicted emission is $\hat{y} = 340$



- **Residual error** = actual value – predicted value.
- **Mean Squared Error (MSE):** average of squared residuals.
- Goal of regression → find line that minimizes MSE.
- This method is called **Ordinary Least Squares (OLS)** regression.

🧱 Coefficient Calculation

Estimating the coefficients of the linear regression model

	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_COMB	CO2EMISSIONS
0	2.0	4	8.5	196
1	2.4	4	9.6	221
2	1.5	4	5.9	136
3	3.5	6	11.1	255
4	3.5	6	10.6	244
5	3.5	6	10.0	230
6	3.5	6	10.1	232
7	3.7	6	11.1	255
8	3.7	6	11.6	267

X_1 (ENGINESIZE) y (CO2EMISSIONS)

We can use two formulas to calculate coefficients θ_0 and θ_1

It requires that we calculate the means, y bar $[\bar{y}]$ and x bar $[\bar{x}]$, of the independent and dependent variables

- Derived by **Gauss and Legendre** in the 1800s.
- Requires calculating the mean of x ($\bar{x}$) and y ($\bar{y}$).
- Example result from dataset:
  - θ_1 (slope) = 39
  - θ_0 (intercept) = 125.7

Predictions with linear regression

	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_COMB	CO2EMISSIONS
0	2.0	4	8.5	196
1	2.4	4	9.6	221
2	1.5	4	5.9	136
3	3.5	6	11.1	255
4	3.5	6	10.6	244
5	3.5	6	10.0	230
6	3.5	6	10.1	232
7	3.7	6	11.1	255
8	3.7	6	11.6	267
9	2.4	4	9.2	214

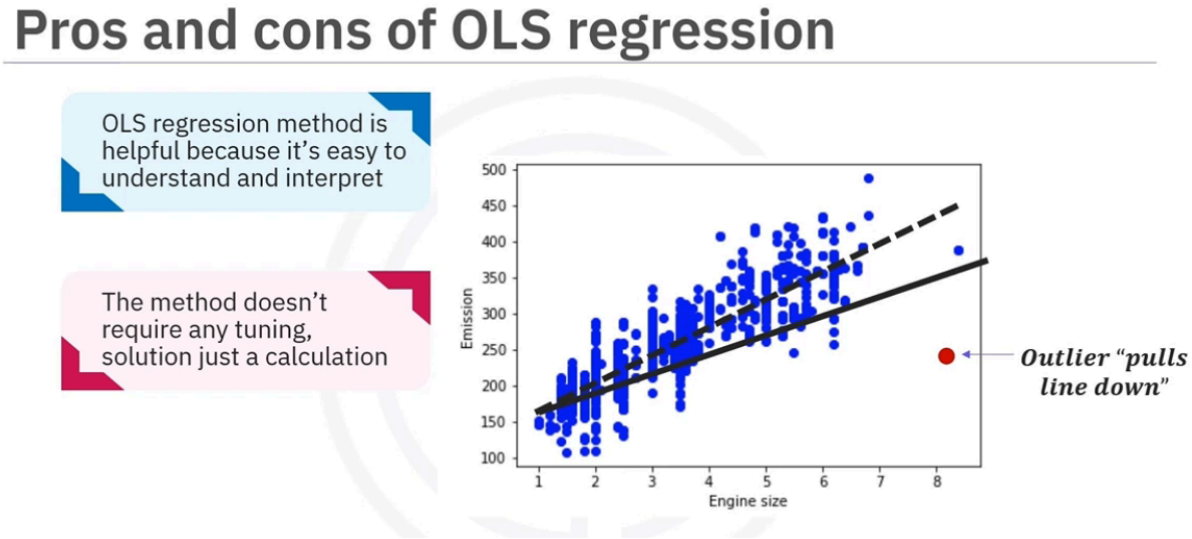
You can predict CO2Emission from EngineSize using the following equation:

$$CO2Emission = 125 + 39 \times EngineSize$$

Equation becomes:

y^=125.7+39x\hat{y} = 125.7 + 39x

✅ Advantages of OLS Regression



- Easy to understand and interpret.
- Fast (just calculations, no tuning needed).
- Works well on small datasets.

⚠️ Limitations

- **Too simplistic** for complex or nonlinear relationships.
- **Sensitive to outliers**, which can distort predictions.

📝 Key Takeaways

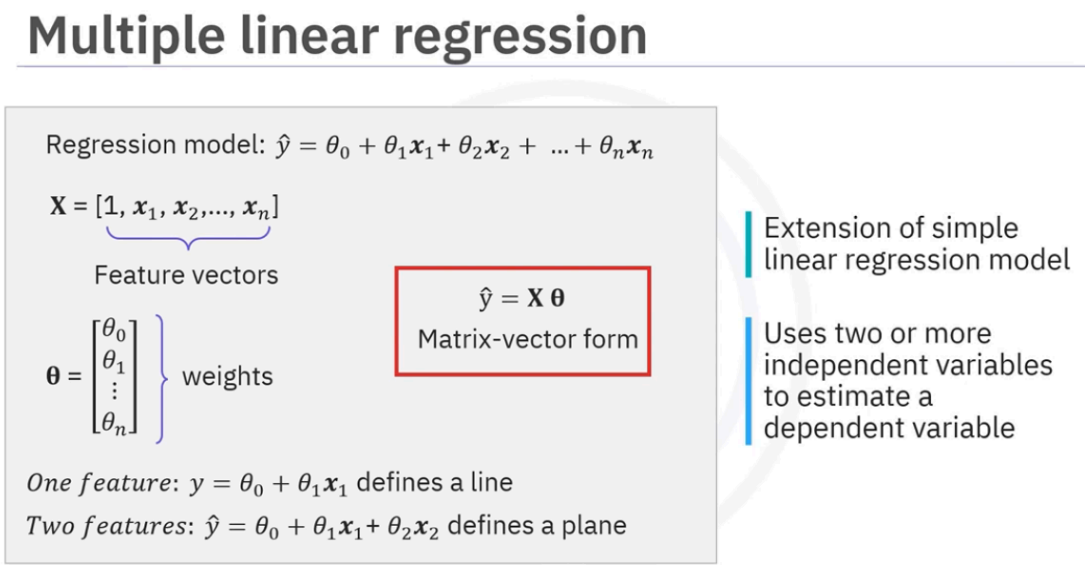
- Simple linear regression uses **one predictor variable** to estimate a continuous target variable.
- It finds a **best-fit line** that minimizes error (MSE).
- Method = **Ordinary Least Squares (OLS)**.
- Great for simple, linear data — but struggles with outliers and complex patterns.

📘 Notes: Introduction to Multiple Linear Regression

🔹 Definition

- **Multiple Linear Regression (MLR):** Extension of simple linear regression.
- Uses **two or more independent variables** to estimate a **dependent variable**.

Mathematical Form

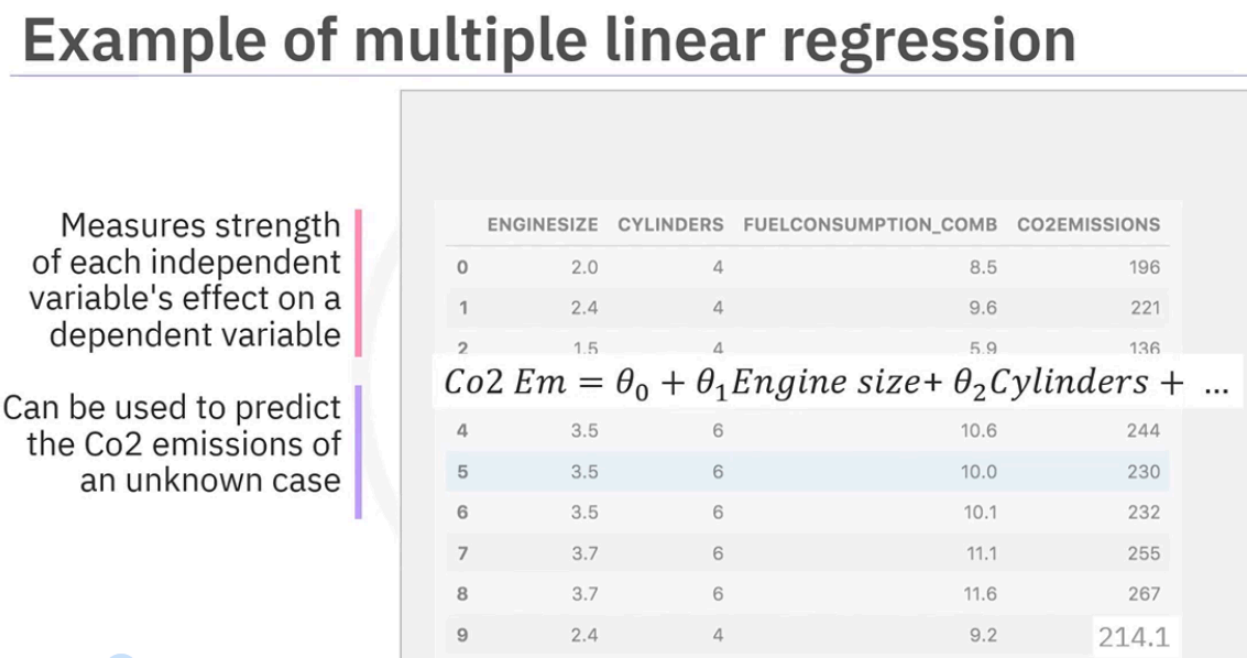


y^=θ0+θ1x1+θ2x2+...+θnxn\hat{y} = \theta_0 + \theta_1x_1 + \theta_2x_2 + ... + \theta_nx_n

- θ0\theta_0: Bias/intercept term.

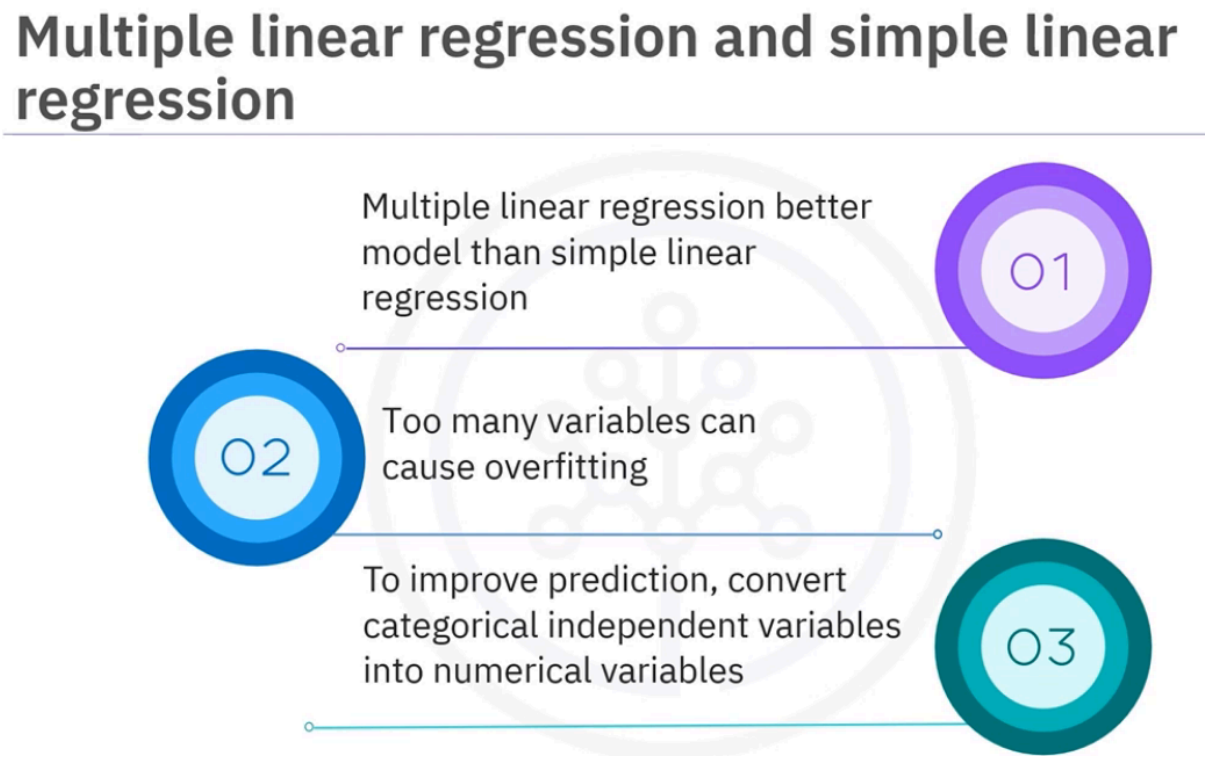
- θi\theta_i: Coefficients (weights).
- xix_i: Features (independent variables).
- Matrix form: y^=Xθ\hat{y} = X\theta.

🔹 Example



- Predict **CO2 emissions** from features like:
    - Engine size
    - Number of cylinders
    - Fuel consumption

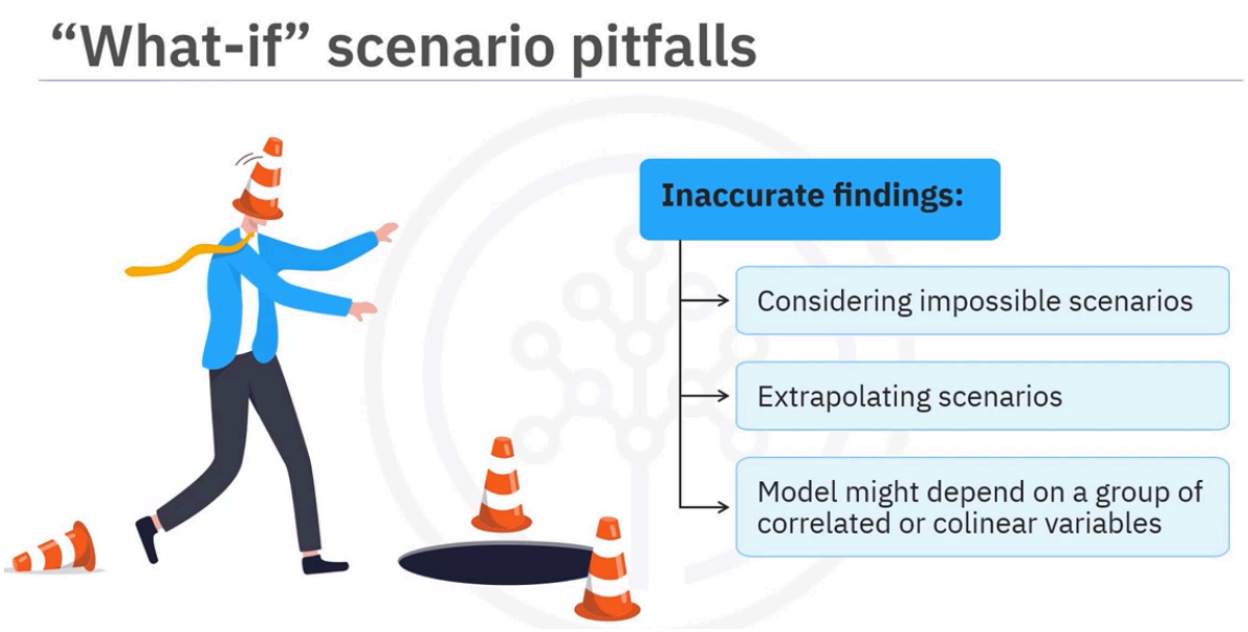
- Produces a **better model** than simple linear regression.



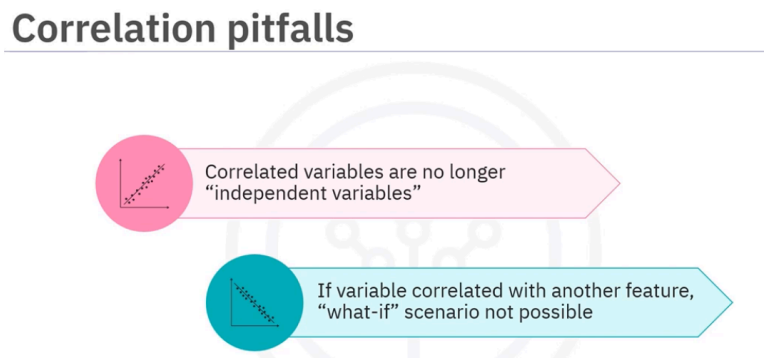
🔹 Advantages

- Measures the **strength of each variable's effect** on the dependent variable.
- Handles **categorical variables** by encoding:
  - **Binary variable:** Manual = 0, Automatic = 1.
  - **Multi-class:** Convert into dummy (Boolean) features.
- Useful for **what-if scenarios** (e.g., effect of BMI change on blood pressure).

🔹 Pitfalls



-
- **Overfitting:** Adding too many variables → memorises training data → poor generalisation.
- **Multicollinearity:** When independent variables are correlated with each other.
    - Predictors are no longer independent.
    - Solution: Remove redundant variables.
- **Invalid what-if scenarios:**



    - Unrealistic feature changes.

- Extrapolating beyond training data.

🔹 Visualization

Predictions with multiple linear regression

	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_COMB	CO2EMISSIONS
0	2.0	4	8.5	196
1	2.4	4	9.6	221
2	1.5	4	5.9	136
3	3.5	6	11.1	255
4	3.5	6	10.6	244
5	3.5	6	10.0	230
6	3.5	6	10.1	232
7	3.7	6	11.1	255
8	3.7	6	11.6	267
9	2.4	4	9.2	214.1

$\theta = (125, 6.2, 14, \ldots)$

$\hat{y} = 125 + 6.2x_1 + 14x_2 +$

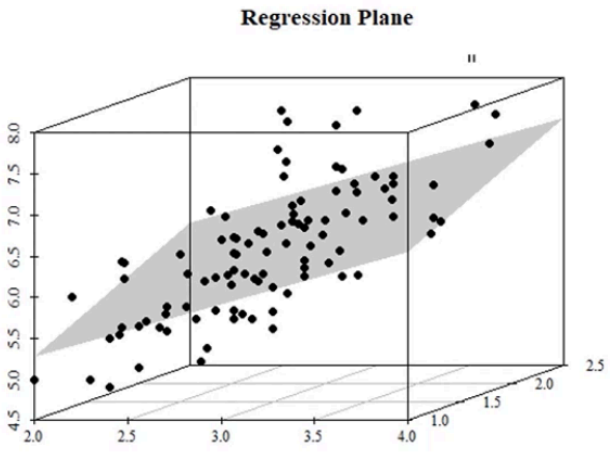
$Co2Em = 125 + 6.2EngSize + 14$ Cylinders + ...

Multiple linear regression assigns relative importance to features

Choice of features suggests cylinder number has more impact on Co2

- **Simple Linear Regression:** Line.
- **Multiple Linear Regression:**

Fitting a hyperplane



Simple linear regression: Regression equation defines a line

Multiple linear regression using two features: Solution describes a plane

Beyond two dimensions: Solution describes a hyperplane

-

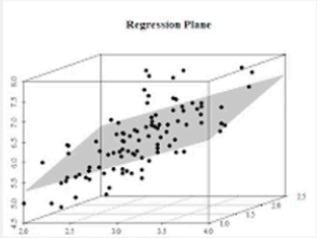
- 2 features → Plane.
- 2 features → Hyperplane.

🔹 Error Measurement

Model error

One feature: $y = \theta_0 + \theta_1 \boldsymbol{x}_1$ defines a line

Two features: $\hat{y} = \theta_0 + \theta_1 \boldsymbol{x}_1 + \theta_2 \boldsymbol{x}_2$ defines a plane



N features: $y = \theta_0 + \theta_1 \boldsymbol{x}_1 + \theta_2 \boldsymbol{x}_2 + \ldots + \theta_N \boldsymbol{x}_N$ an N−dimensional hyperplane

Residual error for each car in the data set = Difference between its true Co2 emission value and predicted value

Average of all residual errors indicates how poorly model predicts the actual values

- **Residual error:** Difference between actual and predicted values.
- **Mean Squared Error (MSE):**

Least squares solution

$$\text{Minimizing MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Same as minimizing $\text{SE} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

==> least-squares linear regression

Factor of 1/n in the MSE equation unnecessary to minimize the error

Method is called least-squares linear regression

MSE=1n∑i=1n(yi−y^i)2MSE = \frac{1}{n}\sum_{i=1}^n (y_i - \hat{y}_i)^2

- Goal: Find parameters (θ\theta) that **minimise MSE**.
- Method: **Least Squares Regression**.

🔹 Parameter Estimation Methods

Estimating parameters for multiple linear regression

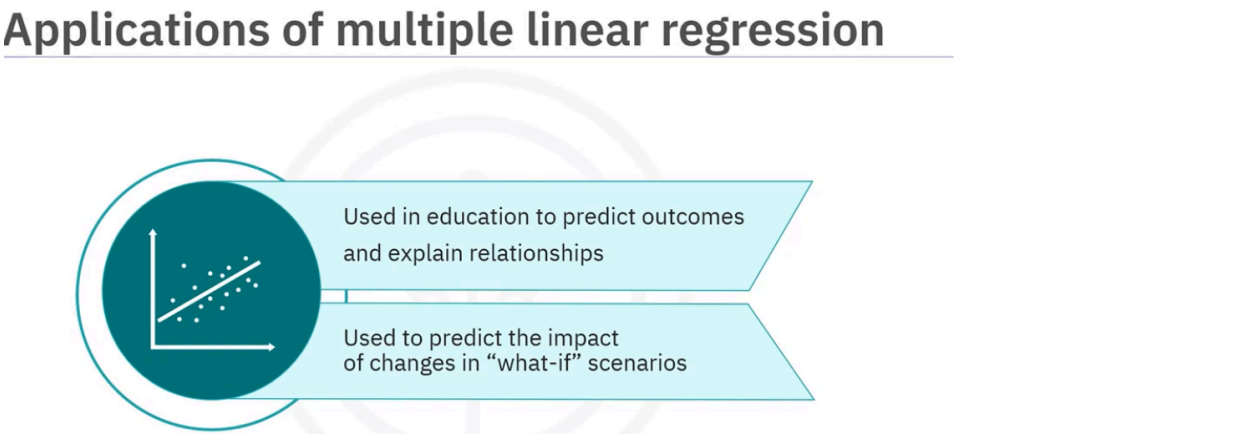
- Gradient descent starts optimization with random values for each coefficient
- Useful for a large data set

1. **Ordinary Least Squares (OLS):**

- Uses linear algebra to compute optimal θ\theta.
- Best for small/medium datasets.

2. **Optimisation (e.g., Gradient Descent):**
    - Iteratively updates coefficients.
    - Best for **large datasets**.

🔹 Applications



- **Education:** Predict exam performance from revision time, test anxiety, attendance, and gender.
- **Healthcare:** Predict blood pressure changes from BMI.
- **Industry:** Predict CO2 emissions in automobiles.
- **What-if Scenarios:** Estimate outcomes of hypothetical changes.

🔹 Summary

Recap

- Extension of simple linear regression model
- Uses two or more independent variables to estimate a dependent variable
- Widely used in the education sector to predict outcomes
- To build your multiple regression model, you must use uncorrelated variables.
- The most common methods to estimate multiple regression parameters are ordinary least square and optimization with random values.

- MLR uses **multiple features** to predict a target.
- Improves accuracy compared to simple regression.
- Must handle **overfitting** and **multicollinearity** carefully.
- Parameters estimated using **OLS** or **Gradient Descent**.
- Widely applied across industries for **prediction and explanation**.

Got it ✅ I'll turn your provided text into **well-structured notes with code snippets** so you can quickly revise Multiple Linear Regression (MLR) in Python.

📘 Notes on Multiple Linear Regression (MLR) Labs

🔹 1. Objective

By the end of this lab, you will be able to:

- Use **Scikit-learn** to implement multiple linear regression.
- Create, train, and test a multiple linear regression model on real data.

🔹 2. Required Packages

Install and import the necessary libraries:

```
!pip install numpy==2.2.0 pandas==2.2.3 scikit-learn==1.6.0 matplotlib==3.9.3

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import preprocessing, linear_model
from sklearn.model_selection import train_test_split
%matplotlib inline
```

🔹 3. Load Dataset

Dataset: **FuelConsumption.csv** (Canada's car CO2 emissions & fuel consumption data)

```
url =
"https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%202/data/FuelConsumptionCo2.csv"
df = pd.read_csv(url)
df.sample(5)
```

Columns of interest:

- `ENGINESIZE`, `CYLINDERS`, `FUELCONSUMPTION_*`, `CO2EMISSIONS`

🔹 4. Data Cleaning & Feature Selection

- Drop **categorical columns** & unused fields:

```
df = df.drop(['MODELYEAR', 'MAKE', 'MODEL', 'VEHICLECLASS',
'TRANSMISSION','FUELTYPE'], axis=1)
```

- Check correlations:

```
df.corr()
```

➡ Drop redundant features (`CYLINDERS`, `FUELCONSUMPTION_CITY`, `FUELCONSUMPTION_HWY`, `FUELCONSUMPTION_COMB`):

```
df = df.drop(['CYLINDERS',
'FUELCONSUMPTION_CITY','FUELCONSUMPTION_HWY','FUELCONSUMPTION_COMB'],
axis=1)
```

Now the dataset has:

- **ENGINESIZE**, **FUELCONSUMPTION_COMB_MPG**, **CO2EMISSIONS**

🔹 5. Feature Extraction

```
X = df.iloc[:, [0,1]].to_numpy()   # Independent vars
y = df.iloc[:, [2]].to_numpy()     # Dependent var (CO2 emissions)
```

🔹 6. Standardization

```
std_scaler = preprocessing.StandardScaler()
X_std = std_scaler.fit_transform(X)
```

🔹 7. Train-Test Split

```
X_train, X_test, y_train, y_test = train_test_split(X_std, y, test_size=0.2, random_state=42)
```

🔹 8. Build Multiple Linear Regression Model

```
regressor = linear_model.LinearRegression()
regressor.fit(X_train, y_train)

print("Coefficients:", regressor.coef_)
print("Intercept:", regressor.intercept_)
```

🔹 9. Convert Back to Original Scale

```
means_ = std_scaler.mean_
std_devs_ = np.sqrt(std_scaler.var_)

coef_original = regressor.coef_ / std_devs_
intercept_original = regressor.intercept_ - np.sum((means_ * regressor.coef_) / std_devs_)

print("Coefficients (original scale):", coef_original)
print("Intercept (original scale):", intercept_original)
```

🔹 10. Visualization

3D Regression Plane

```
fig = plt.figure(figsize=(20, 8))
ax = fig.add_subplot(111, projection='3d')

X1, X2 = X_test[:, 0], X_test[:, 1]
x1_surf, x2_surf = np.meshgrid(np.linspace(X1.min(), X1.max(), 100),
                               np.linspace(X2.min(), X2.max(), 100))
y_surf = regressor.intercept_ + regressor.coef_[0,0]*x1_surf + regressor.coef_[0,1]*x2_surf

ax.scatter(X1, X2, y_test, c='blue', s=50, alpha=0.5, edgecolors='k')
ax.plot_surface(x1_surf, x2_surf, y_surf, color='k', alpha=0.2)
ax.set_xlabel("Engine Size")
ax.set_ylabel("Fuel Consumption (MPG)")
ax.set_zlabel("CO2 Emissions")
plt.show()
```

Simple Regression Lines (slices of 3D plot)

```
plt.scatter(X_train[:,0], y_train, color='blue')
plt.plot(X_train[:,0], regressor.coef_[0,0]*X_train[:,0] + regressor.intercept_[0], '-r')
plt.xlabel("Engine Size")
plt.ylabel("CO2 Emission")
plt.show()
```

🔹 11. Exercises (Practice)

Exercise 1 – Fit regression using only `ENGINESIZE`

```
X_train_1 = X_train[:,0].reshape(-1, 1)
regressor_1 = linear_model.LinearRegression()
regressor_1.fit(X_train_1, y_train)

print("Coefficient:", regressor_1.coef_)
print("Intercept:", regressor_1.intercept_)
```

Exercise 2 – Plot regression line for training set

```
plt.scatter(X_train_1, y_train, color='blue')
plt.plot(X_train_1, regressor_1.coef_[0] * X_train_1 + regressor_1.intercept_, '-r')
plt.xlabel("Engine Size")
plt.ylabel("CO2 Emission")
plt.show()
```

Exercise 3 – Same plot, but test set

```
X_test_1 = X_test[:,0]
plt.scatter(X_test_1, y_test, color='blue')
plt.plot(X_test_1, regressor_1.coef_[0] * X_test_1 + regressor_1.intercept_, '-r')
plt.xlabel("Engine Size")
plt.ylabel("CO2 Emission")
plt.show()
```

Exercise 4 – Fit regression using `FUELCONSUMPTION_COMB_MPG`

```
X_train_2 = X_train[:,1].reshape(-1, 1)
```

```
regressor_2 = linear_model.LinearRegression()
regressor_2.fit(X_train_2, y_train)

print("Coefficient:", regressor_2.coef_)
print("Intercept:", regressor_2.intercept_)
```

Exercise 5 – Plot test results

```
X_test_2 = X_test[:,1]
plt.scatter(X_test_2, y_test, color='blue')
plt.plot(X_test_2, regressor_2.coef_[0] * X_test_2 + regressor_2.intercept_, '-r')
plt.xlabel("Fuel Consumption (MPG)")
plt.ylabel("CO2 Emission")
plt.show()
```

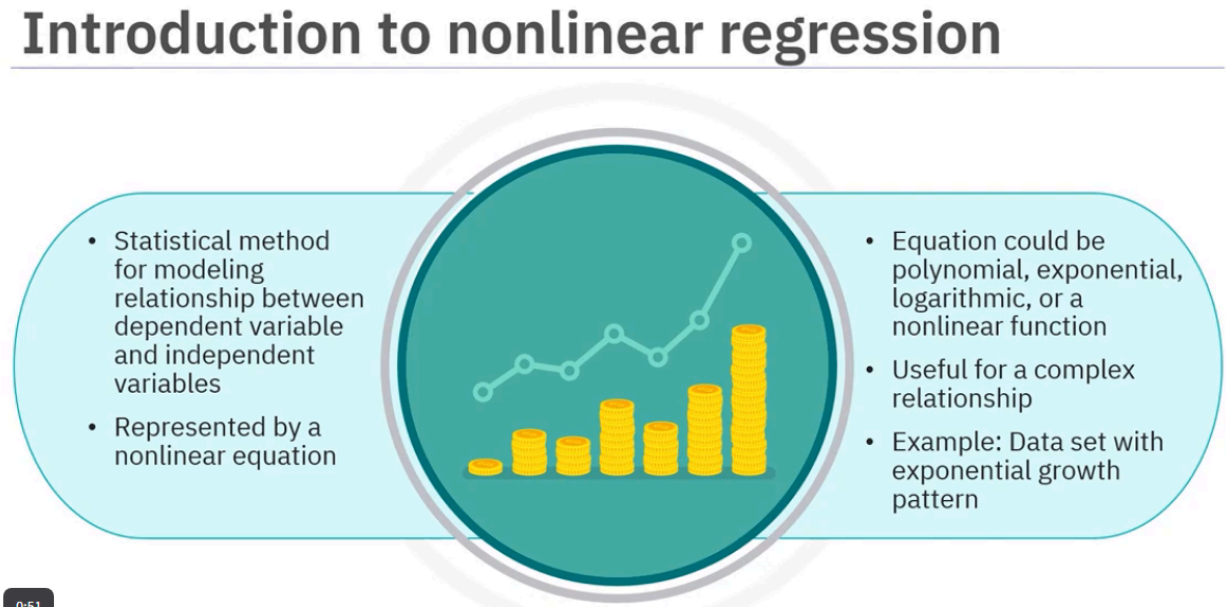
✅ Key Takeaways

- **Multiple Linear Regression** uses several predictors (features) to model a dependent variable.
- Standardization ensures fair contribution of features.
- Feature correlation analysis helps remove redundancy.
- Even with multiple features, **non-linear relationships** may limit model performance.
- Simple regression (single predictor) can sometimes fit better than MLR if features are strongly correlated.

Here are **well-structured notes** from your transcript on **Polynomial & Nonlinear Regression** 👇

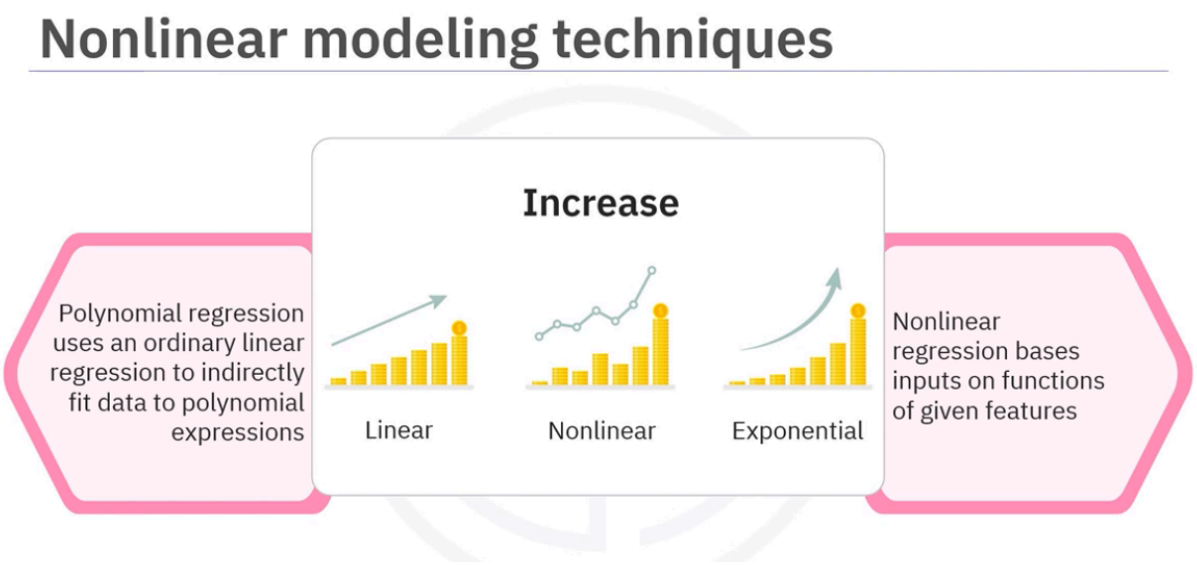
📘 Notes: Polynomial & Nonlinear Regression

🔹 Nonlinear Regression

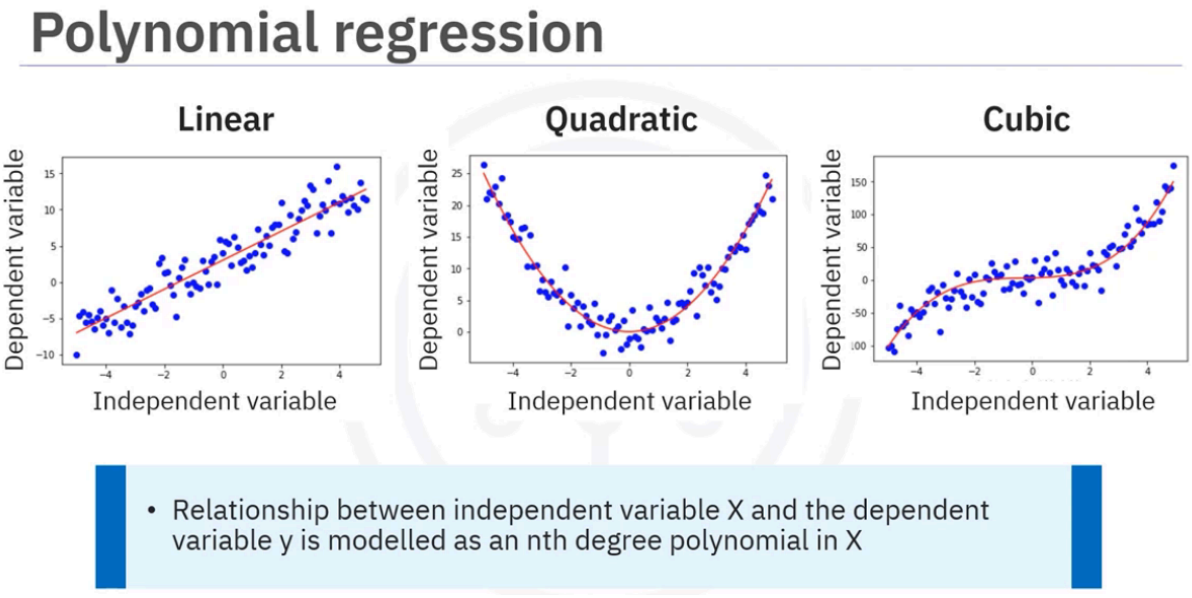


- **Definition:** Models the relationship between dependent (Y) and independent variables (X) using **nonlinear equations** (not limited to straight lines).
- Types of nonlinear functions:
    - **Polynomial**
    - **Exponential**
    - **Logarithmic**
    - **Sinusoidal (periodic)**
- **Use case:** When relationships are complex and cannot be captured by a straight line.
    - Example: **Exponential growth** → GDP increase over time.

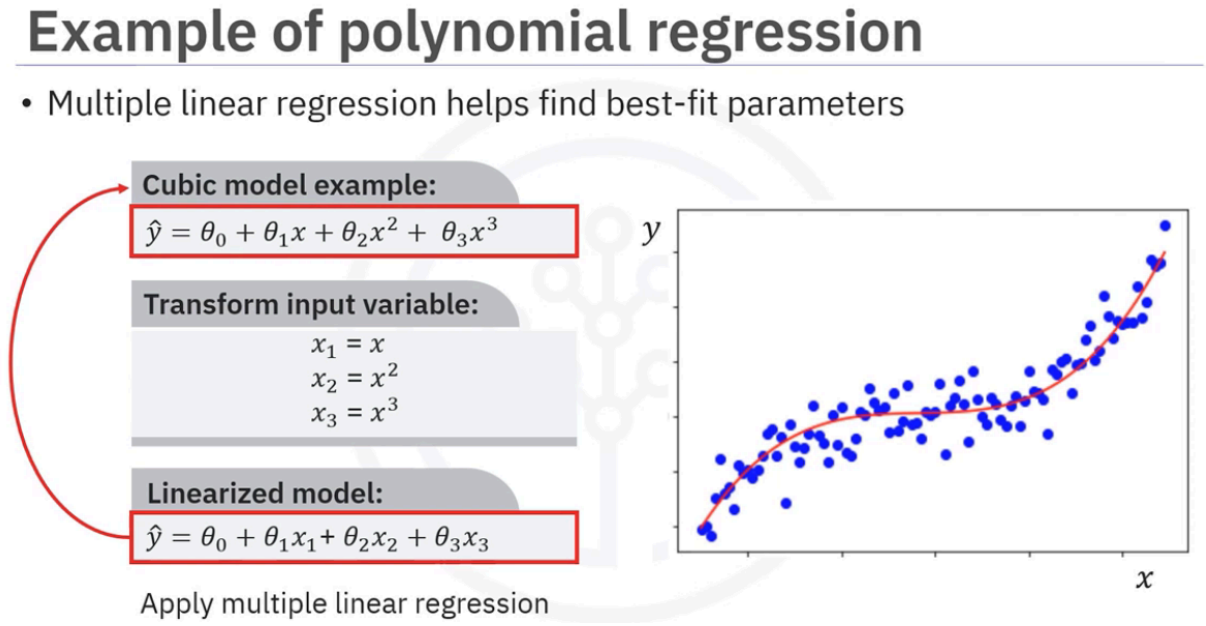
🔹 Polynomial Regression



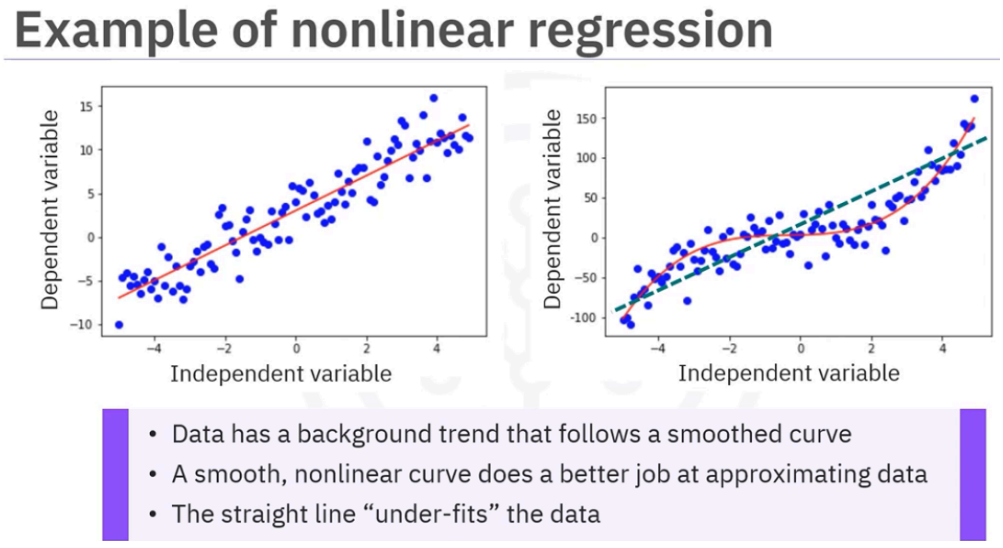
- **Special case of nonlinear regression.**



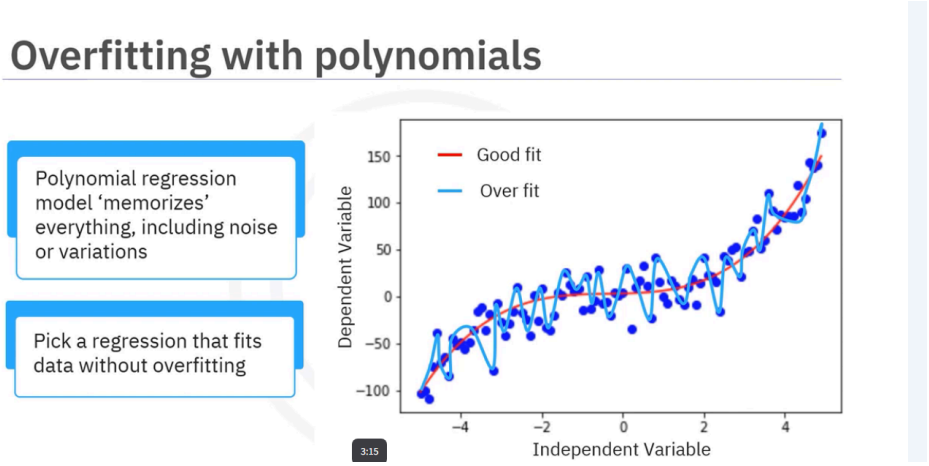
- Models Y as an **n-th degree polynomial in X**:



- y=θ0+θ1x+θ2x2+θ3x3+...+θnxny = \theta_0 + \theta_1x + \theta_2x^2 + \theta_3x^3 + ... + \theta_nx^n
- **Key idea:**
    - Transform features into polynomial terms (x1=x,x2=x2,x3=x3x_1=x, x_2=x^2, x_3=x^3)
    - Then apply **ordinary linear regression** (since coefficients remain linear).
- **Visualization:**

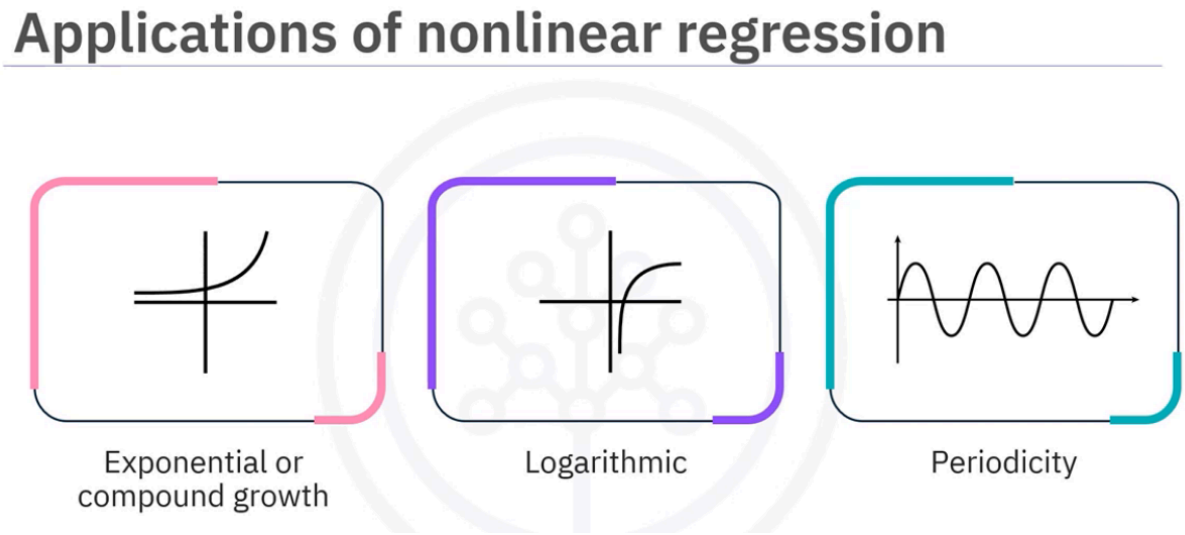


- Linear (straight line) → underfits.
- Quadratic/cubic → better fit.
- Very high degree polynomial → **overfitting** (captures noise, not trend).



- **Summary:** Polynomial regression = nonlinear in features, **linear in parameters**.

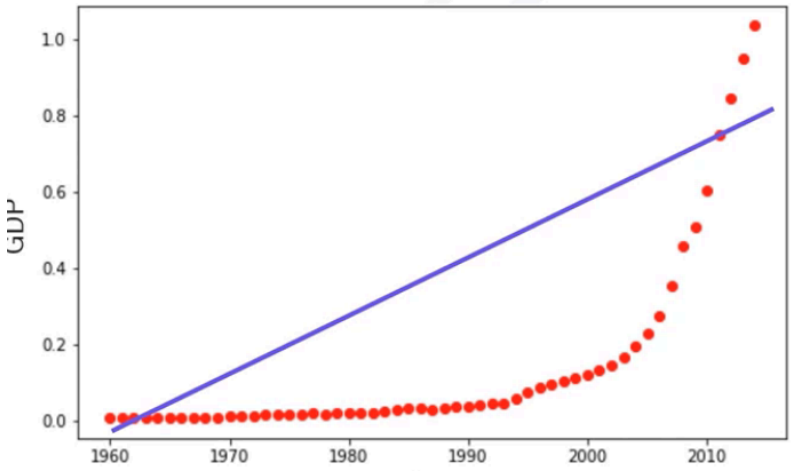
🔹 Common Nonlinear Relationships & Applications



1. **Exponential growth:**

Compound growth example

- Scatterplot displays strong dependence of GDP on time, but relationship is nonlinear



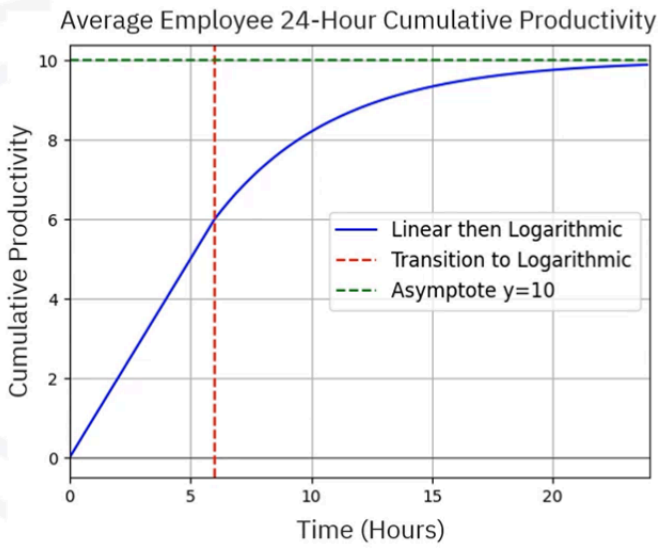
	Year	GDP (B USD)
0	1960	59.2
1	1961	49.6
2	1962	46.7
3	1963	50.1
4	1964	59.1
5	1965	69.8
6	1966	75.9
7	1967	72.1
...	.....	.....

   - Example: GDP growth, compound interest.
   - Formula: y^=θ0+θ1ex\hat{y} = \theta_0 + \theta_1 e^x.

2. **Logarithmic (diminishing returns):**

Productivity by work hours example

- Working more hours per day on average increases productivity
- After a reasonable limit, each additional hour generates less productivity



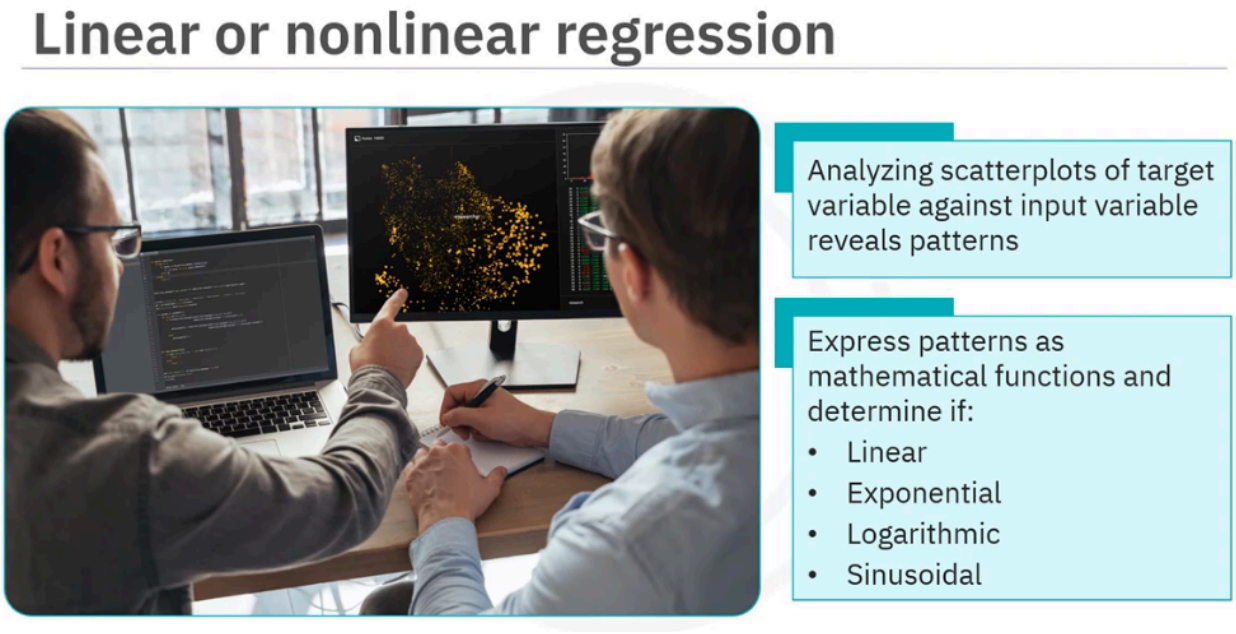
- Example: Productivity vs hours worked → increases initially, then slows down.

3. **Periodic (sinusoidal):**
    - Example: Seasonal rainfall, temperature cycles.

🔹 Pitfalls

- **Overfitting in polynomial regression:**
    - High-degree polynomials pass through every point → memorize noise.
    - Must balance fit vs generalization.

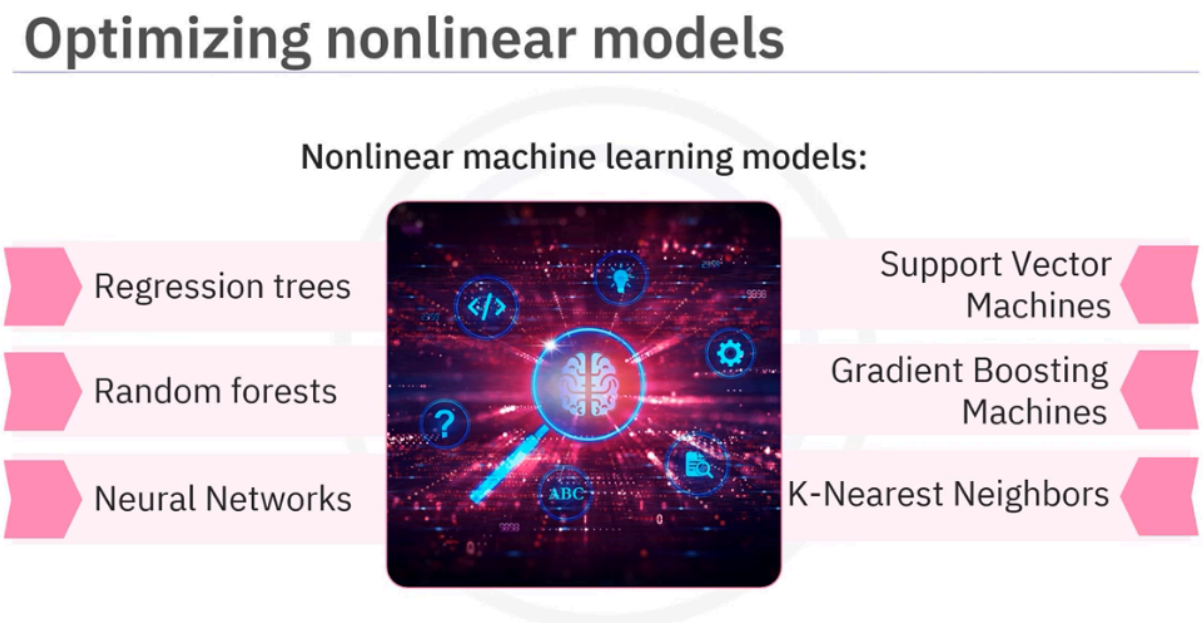
🔹 Model Selection Process



- **Step 1:** Visualize data (scatter plots).
- **Step 2:** Identify trend type (linear, exponential, logarithmic, periodic).

- **Step 3:** Choose regression model accordingly.
- **Step 4:** Validate fit by comparing predictions vs actuals.

🔹 Finding Optimal Nonlinear Model



1. If **mathematical equation known** → optimize parameters (e.g., Gradient Descent).
2. If **no equation assumed** → try ML models:
    - Regression Trees
    - Random Forests
    - Gradient Boosting Machines
    - Support Vector Machines
    - Neural Networks
    - k-Nearest Neighbors

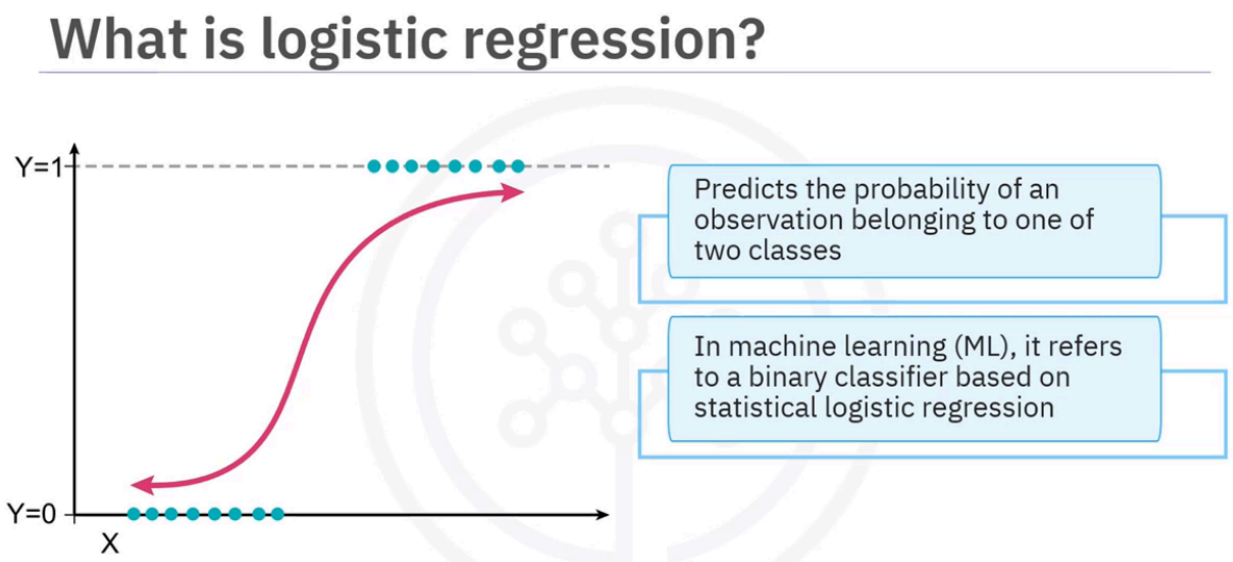
🔹 Summary

- **Nonlinear regression:** Models complex, non-straight relationships.
- **Polynomial regression:** Special case → nonlinear features, linear parameters.
- **Common nonlinear models:** Exponential, logarithmic, sinusoidal.
- **Overfitting risk** in high-degree polynomials.
- **Model selection** relies on visualization + optimization.
- Advanced ML models can approximate nonlinear patterns if no explicit formula exists.

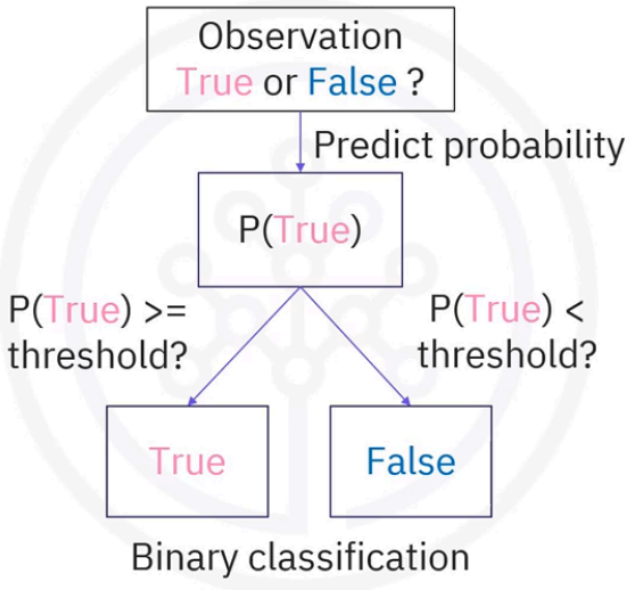
Here are **clean, structured notes** from your transcript on **Logistic Regression** 👇

📘 Notes: Logistic Regression

🔹 Definition

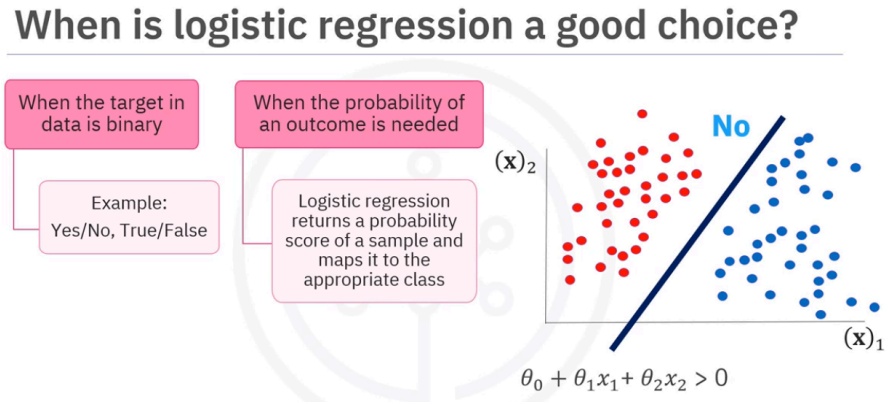


What is logistic regression?



- **Logistic Regression** = Statistical + ML method for **binary classification**.
- Predicts the **probability** of an observation belonging to class 1 (yes/true) or class 0 (no/false).
- Uses a **sigmoid (logit) function** to map values into the range **0–1**.

🔹 When to Use Logistic Regression



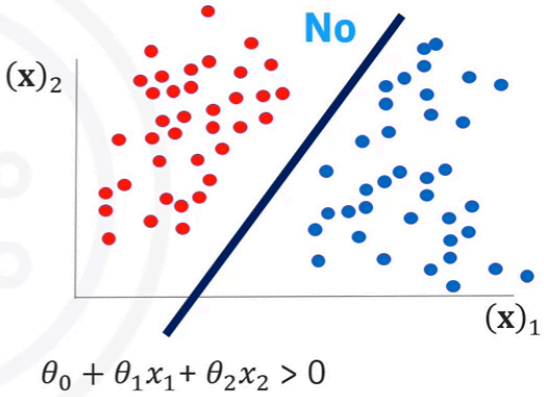
When is logistic regression a good choice?

- If the data is linearly separable, the decision boundary of logistic regression is a line, a plane, or a hyperplane

Example: $\theta_0 + \theta_1 x_1 + \theta_2 x_2 > 0$

- To understand the impact of an independent feature

Example: Select features based on model coefficient size or weights



1. **Binary target variable** (0/1, Yes/No, True/False).
2. **Need probability of outcome** (not just classification).
    - Example: Probability of a customer buying a product.
3. **Linearly separable data** → decision boundary = line, plane, or hyperplane.
4. **Feature impact analysis** → coefficients show how strongly features affect the outcome.

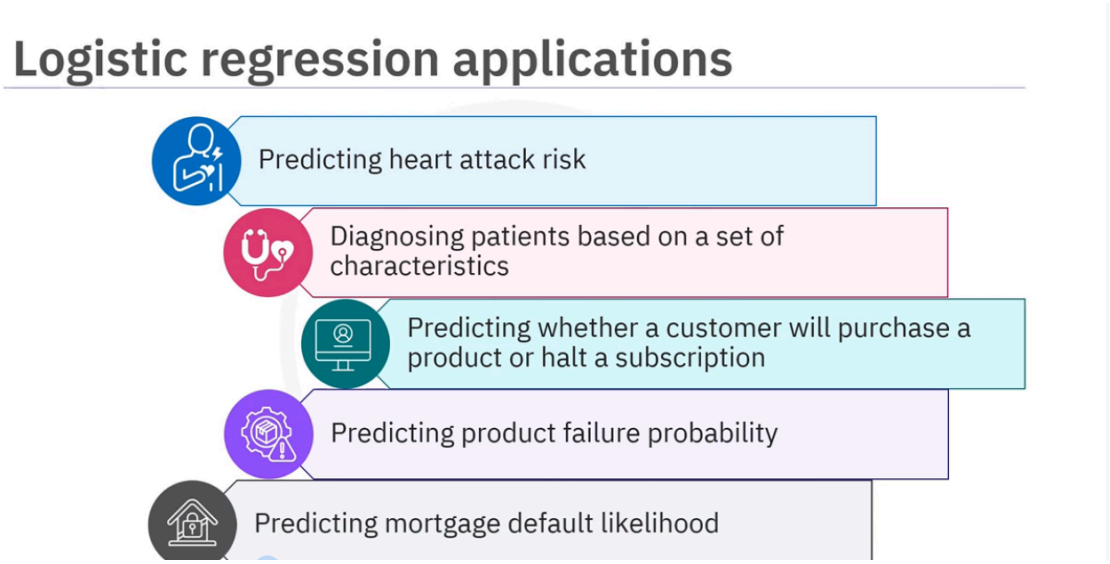
InIn

🔹 Logistic Regression Workflow

1. Input: Independent features (e.g., age, income).
2. Compute linear combination:
   y=θ0+θ1x1+θ2x2y = \theta_0 + \theta_1 x_1 + \theta_2 x_2
3. Apply **Sigmoid Function**:
   σ(y)=11+e−y\sigma(y) = \frac{1}{1 + e^{-y}}
    - Compresses outputs to **[0,1] range**.
    - Represents **predicted probability (p̂)**.

4. Apply **Threshold (Decision Boundary)**:
    - If p^≥0.5p̂ \geq 0.5 → Class = 1
    - If p^<0.5p̂ < 0.5 → Class = 0

🔹 Example Applications



- Predicting **heart attack risk** (based on age, sex, BMI).
- Identifying the likelihood of **disease occurrence** (e.g., diabetes).
- **Customer churn prediction** (telecom, subscription businesses).
- **Fraud detection, credit scoring** (mortgage default, loan repayment).
- **System/process failure probability**.

- **Example: Customer Churn**

Logistic regression example

Scenario

Telecommunication data set:

- Services that customers have signed up for
- Customer account information
- Demographic information
- Customers who've left in the last month

Logistic regression example

Independent variables

Dependent variable

	tenure	age	address	income	ed	employ	equip	callcard	wireless	churn
0	11.0	33.0	7.0	136.0	5.0	5.0	0.0	1.0	1.0	Yes
1	33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	0.0	Yes
2	23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	0.0	No
3	38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	1.0	No
4	7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0	?

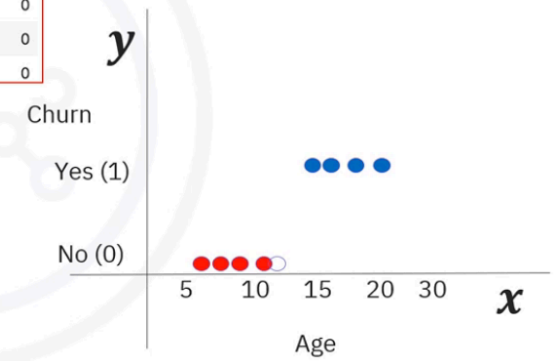
Binary Variable

Predicting churn using linear regression

	tenure	age	address	income	ed	employ	equip	callcard	wireless	churn
0	11.0	33.0	7.0	136.0	5.0	5.0	0.0	1.0	1.0	1
1	33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	0.0	1
2	23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	0.0	0
3	38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	1.0	0
4	7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0	0

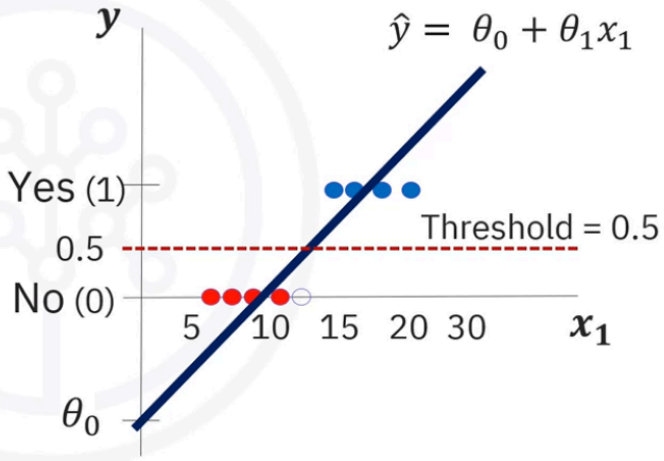
y = Actual churn classes

$\hat{y}$ = Predicted churn classes



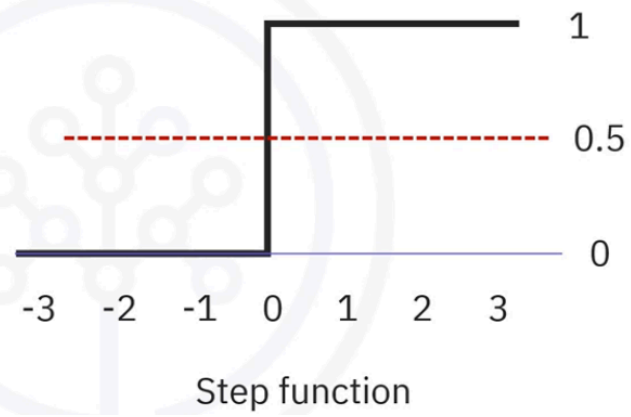
Predicting churn using linear regression

$$\hat{y} \rightarrow \begin{cases} 0 & \text{if } \hat{y} < 0.5 \\ 1 & \text{if } \hat{y} \geq 0.5 \end{cases}$$



Challenges of linear regression

$$\hat{y} \rightarrow \begin{cases} 0 & \text{if } \hat{y} < 0.5 \\ 1 & \text{if } \hat{y} \geq 0.5 \end{cases}$$

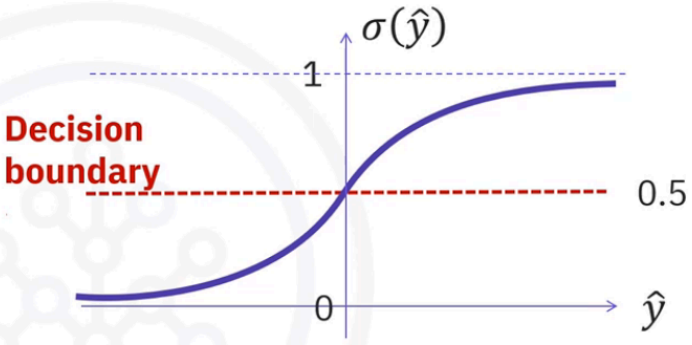


Step function

Probabilities to class predictions

$$\hat{p} = \sigma(\hat{y}) = \frac{1}{1 + e^{-\hat{y}}}$$

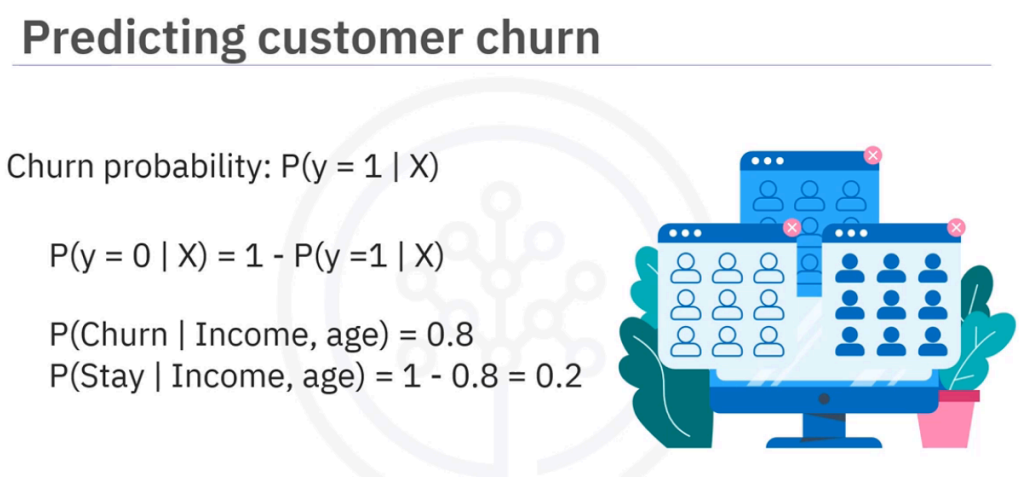
Probability that class is 1



$$\sigma(\hat{y}) \rightarrow \begin{cases} 0 & \text{if } \sigma(\hat{y}) < 0.5 \\ 1 & \text{if } \sigma(\hat{y}) \geq 0.5 \end{cases}$$

- Dataset: Customer info (services, account data, demographics, churn history).
- Target = Churn (Yes=1, No=0).
- Logistic regression predicts the probability of churn.
- Example:
    - P(churn)=0.8P(\text{churn}) = 0.8

- Then, P(not churn)=1−0.8=0.2P(\text{not churn}) = 1 - 0.8 = 0.2.



🔹 Why Not Linear Regression?

- Linear regression outputs values that can exceed 0–1.
- Example: Age vs Churn → line grows indefinitely.
- Step function thresholding (0/1) is **too rigid**, no probability output.
- **Solution:** Sigmoid function → smooth curve between 0 and 1.

🔹 Key Concepts

- **Sigmoid function:** Converts linear values → probabilities.
- **Decision boundary:** Threshold (commonly 0.5) used to classify outputs.
- **Probabilistic interpretation:** Provides both class prediction & probability.
- **Model coefficients (θ):** Indicate feature impact (positive/negative relationship).

🔹 Summary

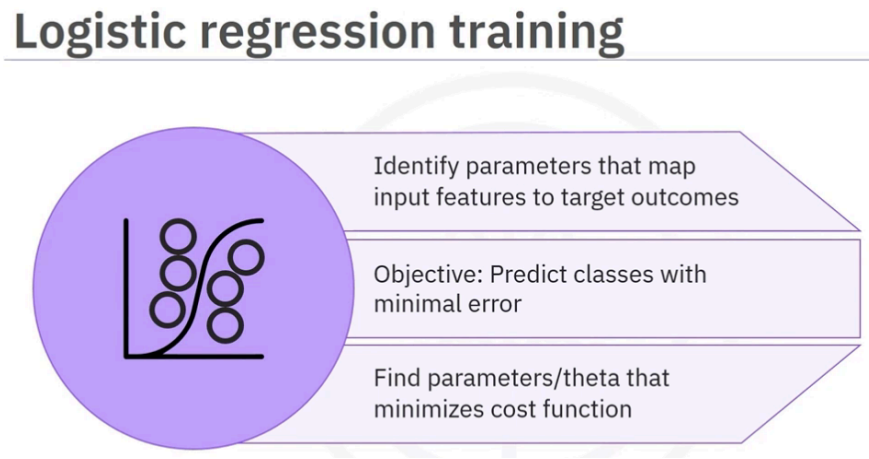
- Logistic regression = **binary classifier + probability predictor**.
- Works best with **binary targets**, **linearly separable data**, and **when probabilities are needed**.
- Uses **sigmoid function** to map predictions into [0,1].
- Widely used in **healthcare, finance, telecom, marketing** for classification problems.

Here are **clear, structured notes** from your lesson on *Training a Logistic Regression Model*:

📘 Training a Logistic Regression Model – Notes

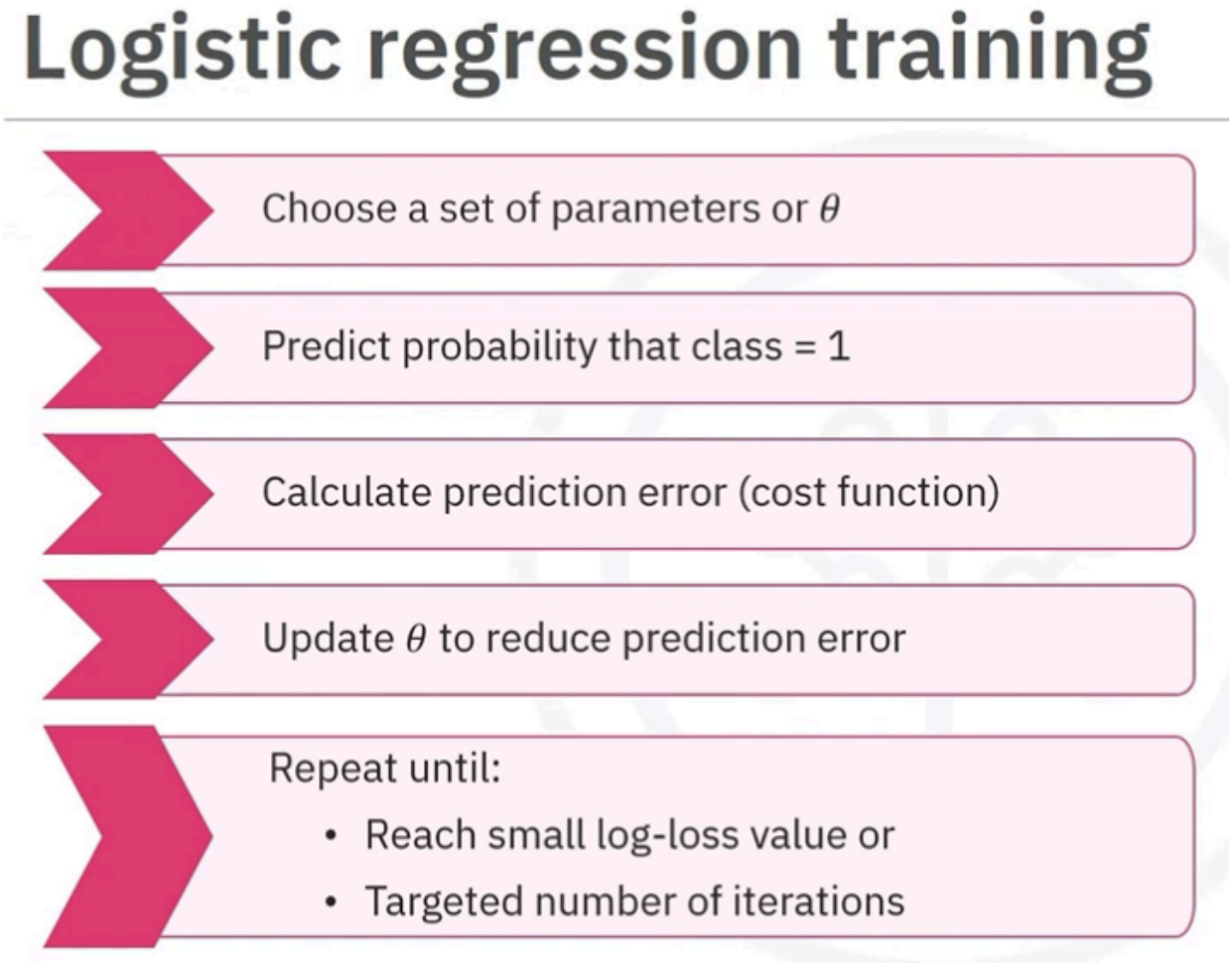
🎯 Objective

- Find the **best parameters (θ / theta)** to map input features to target outcomes.



- Goal: **Predict classes with minimal error**.

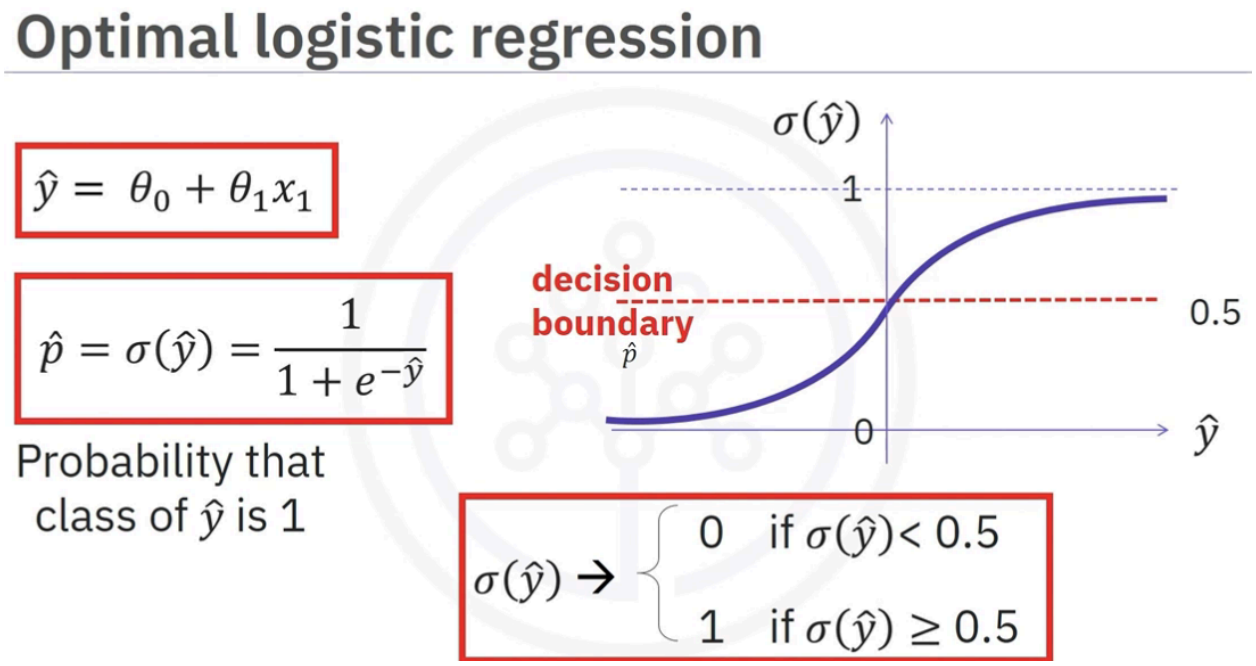
🔑 Steps in Training



1. **Initialize parameters (θ)** – random or chosen values.
2. **Predict probability** that class = 1 for each observation.
3. **Measure error** between prediction and actual (cost function).
4. **Update θ** to reduce error.
5. **Repeat** until:
    - Log loss is small enough, OR

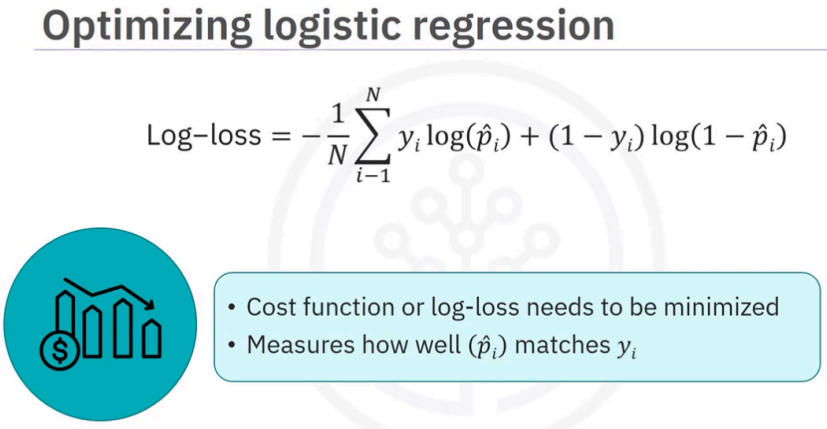
- Maximum iterations reached.

⚡ Optimization in Logistic Regression



- Logistic regression = linear model (ŷ) + **sigmoid function** → binary classifier.
- The **best model** requires optimization to find **optimal θ**.

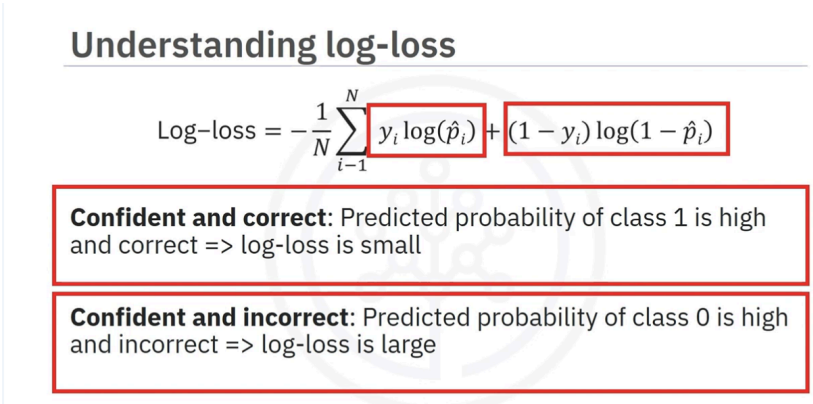
📉 Cost Function – Log Loss



Log Loss=−1n∑i=1n[yi⋅log(p^i)+(1−yi)⋅log(1−p^i)]Log\ Loss = -\frac{1}{n} \sum_{i=1}^{n} \Big[y_i \cdot \log(\hat{p}_i) + (1 - y_i)\cdot \log(1 - \hat{p}_i) \Big]

Where:

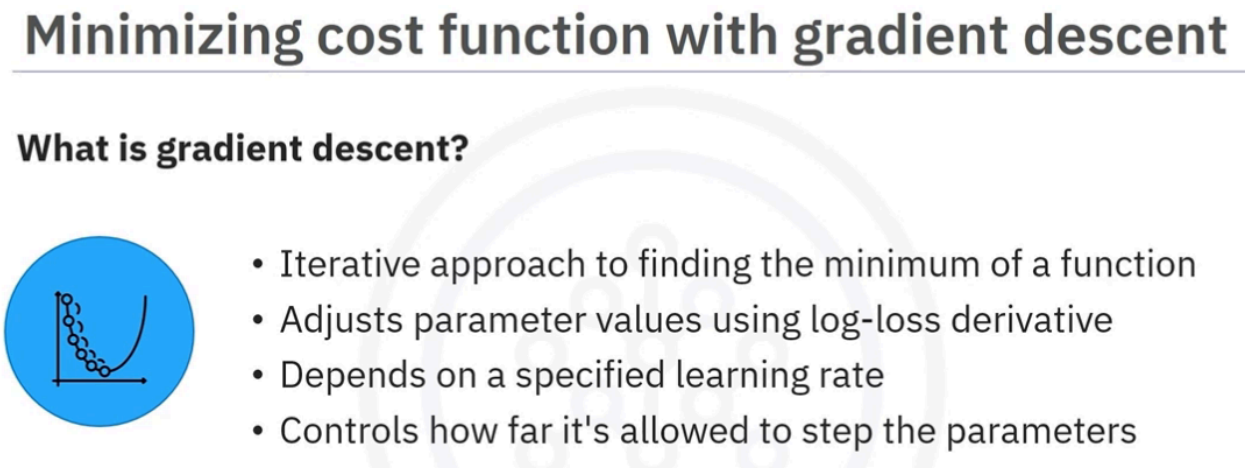
- yiy_i = actual class (0 or 1)
- p^i\hat{p}_i = predicted probability



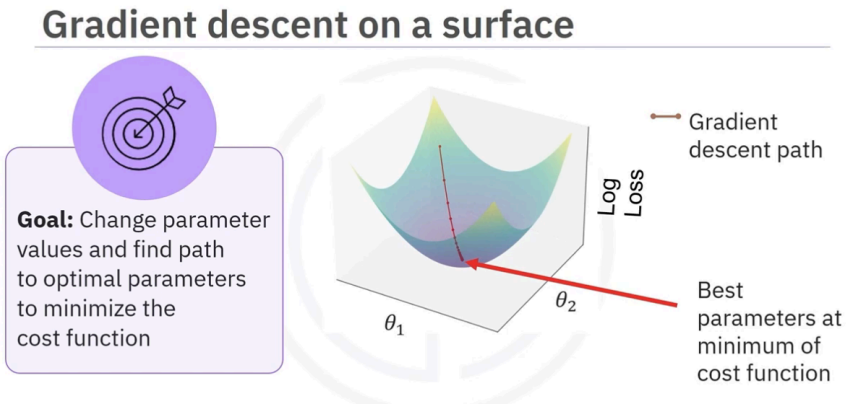
✅ Properties:

- **Small log loss** → good fit.
- Penalises **confident, wrong predictions** heavily.

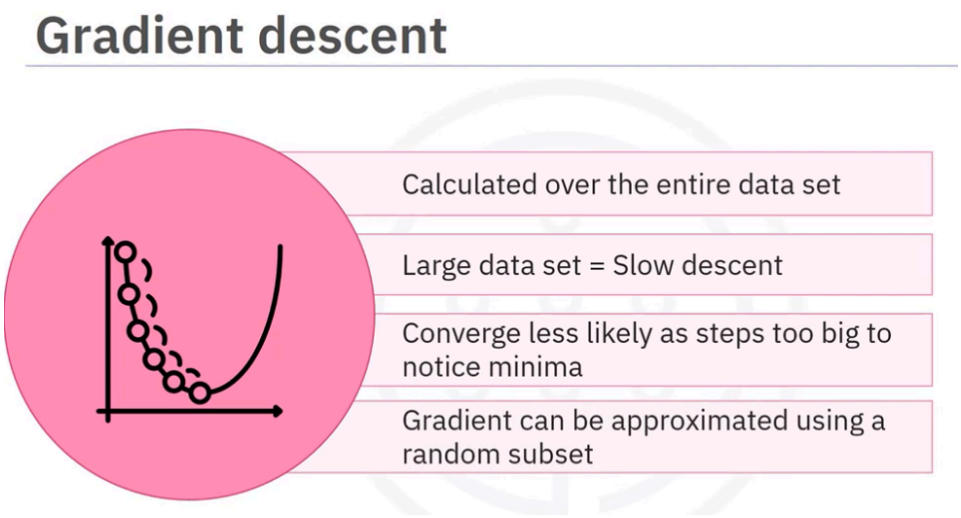
🌀 Gradient Descent (GD)



- Iterative optimization method.
- Adjusts θ in the **direction of steepest descent** (negative gradient).

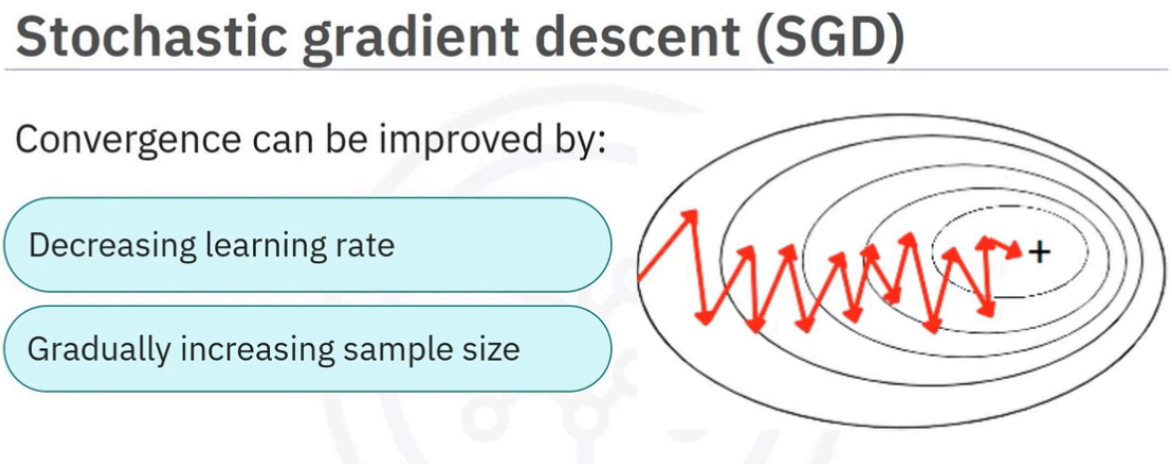


⚙️ Key Concepts:



- **Learning rate (α):** controls step size.
    - Too small → slow convergence.
    - Too big → may overshoot minima.
- As slope → 0 → algorithm reaches optimal θ.

⚡ Stochastic Gradient Descent (SGD)



- Variant of GD.
- Uses **random subset of data** instead of the whole dataset.

✅ Advantages:

- Faster for large datasets.
- Scales well.
- Can escape **local minima** → more likely to find global minimum.

⚠️ Drawbacks:

- Less accurate (wanders around minimum).
- Convergence improved by:
    - **Decreasing learning rate** near minimum.
    - **Increasing sample size** gradually.

📌 Key Takeaways

- Logistic regression training = **find θ that minimises log loss**.
- **Log Loss** is the main cost function.
- **Gradient Descent** → standard optimization.
- **SGD** → faster, scalable, often preferred for large data.

📘 Logistic Regression Notes (with Python)

1. Introduction

- **Logistic Regression** → Used for **classification problems** (binary or multi-class).
- Predicts **probability** of belonging to a class (e.g., churn vs. not churn).
- Suitable for customer churn prediction, medical diagnosis, fraud detection, etc.

---

2. Objectives

After learning this lab, you should be able to:

- Preprocess data for modeling.
- Train and evaluate a Logistic Regression model.
- Interpret model coefficients and performance.

---

3. Setup

Install required libraries:

```
!pip install numpy==2.2.0 pandas==2.2.3 scikit-learn==1.6.0 matplotlib==3.9.3
```

Import libraries:

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import log_loss

import matplotlib.pyplot as plt

%matplotlib inline


import warnings

warnings.filterwarnings('ignore')
```

---

4. Dataset: Telco Churn

- **Goal** → Predict whether a customer will **churn** (leave the company).
- Each row = 1 customer (demographics + service details).
- Focused on **reducing customer turnover**.

Load dataset:

```
url = k-ML0101EN-SkillsNetwork/labs/Module%203/data/ChurnData.csv"
```

churn_df =
pd.read_csv(url)"https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwor

---

5. Data Preprocessing

Select features + convert target:

churn_df = churn_df[['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equip', 'churn']]

churn_df['churn'] = churn_df['churn'].astype('int')

Define X (features) & y (target):

X = np.asarray(churn_df[['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equip']])

y = np.asarray(churn_df['churn'])

Normalize features:

X_norm = StandardScaler().fit(X).transform(X)

Train-test split:

X_train, X_test, y_train, y_test = train_test_split(X_norm, y, test_size=0.2, random_state=4)

---

6. Logistic Regression Modeling

Train model:

```
LR = LogisticRegression().fit(X_train, y_train)
```

Predictions:

```
yhat = LR.predict(X_test)

yhat[:10]
```

Prediction probabilities:

```
yhat_prob = LR.predict_proba(X_test)

yhat_prob[:10]
```

---

7. Feature Importance

Visualize coefficients:

```
coefficients = pd.Series(LR.coef_[0], index=churn_df.columns[:-1])

coefficients.sort_values().plot(kind='barh')

plt.title("Feature Coefficients in Logistic Regression Churn Model")

plt.xlabel("Coefficient Value")

plt.show()
```

🔎 **Interpretation:**

- Positive coefficient → Higher value increases chance of churn.
- Negative coefficient → Higher value decreases chance of churn.
- Small absolute values → Weak effect on prediction.

---

8. Performance Evaluation

Log Loss

- Measures **probability-based prediction accuracy**.
- Lower log loss → Better model.

log_loss(y_test, yhat_prob)

---

9. Practice Exercises (with Expected Results)

Modify the feature set and recompute **log loss**.

a) Add `callcard`
👉 Expected log loss = **0.6039**

b) Add `wireless`
👉 Expected log loss = **0.7227**

c) Add `callcard` + `wireless`
👉 Expected log loss = **0.7761**

d) Remove `equip`
👉 Expected log loss = **0.5302**

e) Remove `income` + `employ`
👉 Expected log loss = **0.6529**

✅ **Key Takeaways:**

- Logistic Regression is effective for churn prediction.
- Feature scaling improves performance.

- Coefficients help interpret feature importance.
- Log Loss is a good metric for evaluating classification models.

Module 2 Summary and Highlights

Congratulations! You have completed this lesson. At this point in the course, you know:

- Regression models relationships between a continuous target variable and explanatory features, covering simple and multiple regression types.
- Simple regression uses a single independent variable to estimate a dependent variable, while multiple regression involves more than one independent variable.
- Regression is widely applicable, from forecasting sales and estimating maintenance costs to predicting rainfall and disease spread.
- In simple linear regression, a best-fit line minimizes errors, measured by Mean Squared Error (MSE); this approach is known as Ordinary Least Squares (OLS).
- OLS regression is easy to interpret but sensitive to outliers, which can impact accuracy.
- Multiple linear regression extends simple linear regression by using multiple variables to predict outcomes and analyze variable relationships.
- Adding too many variables can lead to overfitting, so careful variable selection is necessary to build a balanced model.
- Nonlinear regression models complex relationships using polynomial, exponential, or logarithmic functions when data does not fit a straight line.
- Polynomial regression can fit data but mayoverfit by capturing random noise rather than underlying patterns.

- Logistic regression is a probability predictor and binary classifier, suitable for binary targets and assessing feature impact.
- Logistic regression minimizes errors using log-loss and optimizes with gradient descent or stochastic gradient descent for efficiency.
- Gradient descent is an iterative process to minimize the cost function, which is crucial for training logistic regression models.

Cheat Sheet: Linear and Logistic Regression