Unit-3: Understanding Supervised Learning

3.1 Overview of Supervised Learning

Supervised Learning is like learning with a teacher. Imagine a child learning to identify fruits: a teacher shows the child an apple and says, "This is an apple." The teacher shows a banana and says, "This is a banana." After seeing enough examples, the child can look at a new fruit they've never seen before and correctly identify it.

In technical terms, Supervised Learning is a type of Machine Learning where the algorithm is trained on a **labeled dataset**.

- **Input Data (X):** The features or variables we provide to the model (e.g., the color and shape of a fruit).
- **Output Labels (Y):** The correct answers or targets (e.g., the name of the fruit).

The goal of the algorithm is to learn a mapping function (f) that connects the input to the output:

$$Y = f(X)$$

It is called supervised because the learning happens under **guidance**, the correct output is already given during training, like a teacher providing answers while a student learns.

Supervised learning is used when you want to **predict a known outcome**:

- Predict a **number** (e.g., house price, marks, temperature)
- Predict a **category/label** (e.g., spam/not spam, disease/no disease)

Example
- Input: Email text
- Output: Spam or Not Spam
  The model learns from emails that are already labeled.

Common Uses
- Predicting house prices
- Identifying handwritten digits
- Detecting fraud in banking
- Disease diagnosis

Supervised learning is basically **learning from mistakes**:

1. Model predicts an output.

2. Compare prediction with the actual correct output (label).
3. Compute **error (loss).**
4. Adjust model parameters to reduce error.
5. Repeat until error becomes small and stable.

This process is called **training**.

3.1.1 Concepts of Supervised Learning

1. Labeled Dataset: A dataset where each input has a corresponding correct output.

Examples:

- An email classified as "spam" or "not spam".
- A customer review tagged with a sentiment label: "positive", "negative", or "neutral".
- A recording of a sound with the label "breaking glass" or "dog barking".
- A bank transaction record with a flag indicating "fraudulent" or "legitimate" activity.
- A house price in a spreadsheet, with columns for features like location, size, and number of rooms used as inputs to predict the price (the label).

2. Input Variables (Features)

- Independent variables used to make predictions
- Represent characteristics or attributes of data

Example: Age, salary, temperature, size

3. Output Variable (Target/Label)

- Dependent variable
- The value to be predicted

Example: Price, category, class label

4. Training Phase

- The model learns patterns from labeled data
- Errors are calculated and minimized

5. Testing Phase

- The trained model is tested on new data
- Used to evaluate performance

6. Learning Algorithm

Algorithms used in supervised learning include:

- Linear Regression
- Logistic Regression
- Decision Trees
- Support Vector Machines (SVM)
- k-Nearest Neighbors (KNN)

Advantages of Supervised Learning

- High accuracy
- Clear performance evaluation
- Easy to understand results

Limitations

- Requires large labeled datasets
- Labeling data can be time-consuming
- Less flexible to unseen patterns

3.1.2 Difference between Classification and Regression

Supervised Learning is generally divided into two main categories based on the type of output you want to predict: **Classification** and **Regression**.

1. Classification (Predicting a Category)

In Classification, the output variable is a **category** or a **discrete label**. You are asking the model to put data into specific "buckets."

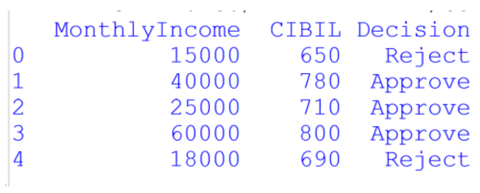
- **Binary Classification:** Only two possible outcomes (e.g., Spam vs. Not Spam).
- **Multi-class Classification:** More than two categories (e.g., classifying an image as a "dog," "cat," or "bird").

Example: Loan Approval (Approve/Reject) is a classification problem because the output is a **category**, not a number.

- **Inputs (features):** Monthly_Income, CIBIL
- **Output (label):** Decision → **Approve** or **Reject**

```
import pandas as pd
import numpy as np

df = pd.DataFrame({
  "MonthlyIncome": [15000, 40000, 25000, 60000, 18000],
  "CIBIL":        [650,   780,   710,   800,   690]
})

# Rule: Approve if Income >= 25000 AND CIBIL >= 700

df["Decision"] = np.where((df["MonthlyIncome"] >= 25000) & (df["CIBIL"] >= 700), "Approve", "Reject")

print(df)
```

Output:



```
   MonthlyIncome  CIBIL Decision
0          15000    650   Reject
1          40000    780  Approve
2          25000    710  Approve
3          60000    800  Approve
4          18000    690   Reject
```

Rule used:

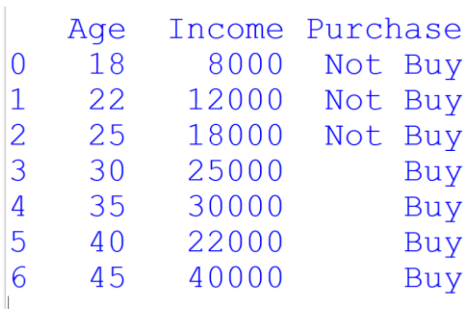
- **Approve** if MonthlyIncome ≥ 25000 **AND** CIBIL ≥ 700
- Otherwise **Reject**

np.where() applies this rule row-wise and writes the final class in the Decision column.

Example: Customer Purchase (Buy/Not Buy) based on **Age** and **Income**.

```
import pandas as pd
import numpy as np

df = pd.DataFrame({
   "Age":    [18, 22, 25, 30, 35, 40, 45],
   "Income": [8000, 12000, 18000, 25000, 30000, 22000, 40000]
})

# Rule-based classification:
# Buy if Age >= 25 AND Income >= 20000, else Not Buy

df["Purchase"] = np.where((df["Age"] >= 25) & (df["Income"] >= 20000), "Buy", "Not Buy")

print(df)
```

Output:



```
   Age  Income Purchase
0   18    8000  Not Buy
1   22   12000  Not Buy
2   25   18000  Not Buy
3   30   25000      Buy
4   35   30000      Buy
5   40   22000      Buy
6   45   40000      Buy
```

It is classification: the output is a **category** (“Buy” or “Not Buy”).

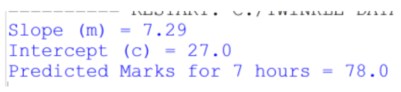
2. Regression (Predicting a Number)

In Regression, the output variable is a **continuous, numerical value.** You are asking the model to predict "how much" or "how many."

Example: Predict Marks **from** Study Hours

```
import numpy as np
import pandas as pd

# 1) Small dataset (Regression: output is a number)
df = pd.DataFrame({
   "StudyHours": [1, 2, 3, 4, 5, 6],
   "Marks":      [35, 40, 50, 55, 65, 70]
})

# 2) Fit a straight line: Marks = m*Hours + c
m, c = np.polyfit(df["StudyHours"], df["Marks"], 1)

print("Slope (m) =", round(m, 2))
print("Intercept (c) =", round(c, 2))

# 3) Predict marks for a new student
new_hours = 7
predicted_marks = m * new_hours + c

print("Predicted Marks for", new_hours, "hours =", round(predicted_marks, 2))
```

Output:



```
Slope (m) = 7.29
Intercept (c) = 27.0
Predicted Marks for 7 hours = 78.0
```

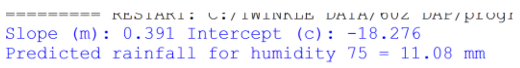
It is regression as the output (**Marks**) is a **continuous numeric value** (e.g., 72.5), not a category.

Example: Predicting the amount of rainfall (Regression)

- **Goal:** Predict **rainfall amount in mm** (a number), so it is **regression**.
- **Inputs (features):** temperature, humidity, wind speed, cloud cover, air pressure (etc.).
- **Output (target):** rainfall amount (e.g., **0 mm, 5.2 mm, 18 mm**).

```
import numpy as np
import pandas as pd

# Rainfall Regression (predict rainfall in mm)

df = pd.DataFrame({
   "Humidity": [40, 50, 60, 70, 80, 90],
   "Rain_mm":  [0,  1,  3,  7,  12, 20]
})

# Fit line: Rain_mm = m*Humidity + c
m, c = np.polyfit(df["Humidity"], df["Rain_mm"], 1)

# Predict rainfall for new humidity value
new_humidity = 75
pred_rain = m * new_humidity + c

print("Slope (m):", round(m, 3), "Intercept (c):", round(c,
3))

print("Predicted rainfall for humidity", new_humidity,
"=", round(pred_rain, 2), "mm")
```

Output:



```
========= RESTART: C:/TWINKLE DATA/602 DAP/progr
Slope (m): 0.391 Intercept (c): -18.276
Predicted rainfall for humidity 75 = 11.08 mm
```

Difference between Classification & Regression

Feature	**Classification**	**Regression**
Output Type	Discrete (Categories/Labels)	Continuous (Numbers)
Nature of O/P	Discrete classes	Numeric values
Used for	Grouping	Prediction
Goal	To find a decision boundary to separate classes.	To find a trend line that fits the data points.
Example	Is this email spam? (Yes/No)	What will the gold price be tomorrow?
Common Algorithms	Logistic Regression, Decision Trees, SVM.	Linear Regression, Polynomial Regression.

- **Classification:** Choosing the correct *bucket*
- **Regression:** Finding the *exact value*

Unit-3: Understanding Supervised Learning

3.2 Basic Terminologies in Machine Learning

Before building machine learning models, it is important to understand some basic terms. These terminologies help in understanding how data is prepared, how models learn, and how performance is evaluated.

3.2.1 Dataset, Features, Labels

1. Dataset

A **dataset** is a collection of data used to train and test a machine learning model.
It is usually organized in **rows and columns** (like a table).

- Each **row** represents one data item (example/record)
- Each **column** represents a variable or attribute

Example:
A dataset of students containing their marks, attendance, and final result.

Marks	Attendance	Result
80	90%	Pass
45	60%	Fail

2. Features

Features are the **input variables** used by the model to make predictions.

- Also called **independent variables**
- They describe the characteristics of the data

Examples of features:

- Age
- Salary
- Marks
- Height
- Attendance

In the student dataset above: **Marks** and **Attendance** are features

3. Labels

Labels are the **output values** that the model tries to predict.

- Also called **target variable** or **dependent variable**
- Present only in **supervised learning**

Examples of labels:

- Pass / Fail
- Price of a house
- Yes / No
- Disease / No disease

In the student dataset: **Result (Pass/Fail)** is the label.

Difference between Features and Labels

Features	Labels
Input to the model	Output of the model
Used for prediction	Value to be predicted
Independent variables	Dependent variable

3.2.2 Training Data, Test Data, Validation Data

1. Training Data

Training data is the portion of data used to **train the model**.

- The model learns patterns from this data
- Usually the **largest portion** of the dataset

Typical split: 60–70%

Example:
Using past student records to teach the model how marks and attendance affect results.

2. Test Data

Test data is used to **evaluate the final performance** of the trained model.

- The model has **never seen this data before**
- Used to check how well the model performs on new data

Typical split: 20–30%

3. Validation Data

Validation data is used to **tune and improve the model** during training.

- Helps in selecting the best model
- Used to adjust parameters (hyper parameters)

Typical split: 10–20%

Data Type	Purpose
Training Data	Learn patterns
Validation Data	Improve and tune model
Test Data	Final performance evaluation

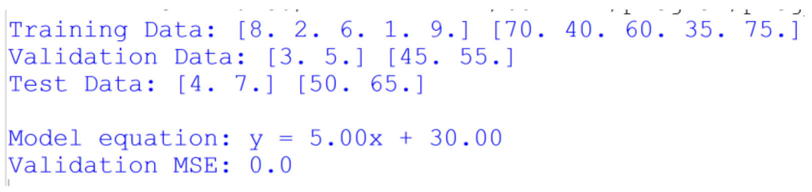
Example: **Suppose you have 100 student records** (StudyHours, Attendance → Marks/Pass).

- **Training Data (70 records):**
  Used to **teach** the model (learn patterns).

- **Validation Data (15 records):**
  Used to **tune/improve** the model while training (choose best settings).
  The model does **not** learn directly from validation; it is used for checking during training.
- **Test Data (15 records):**
  Used at the end to **final check** performance on completely unseen data.

```
import numpy as np

# ----------------------------
# Step 1: Create simple data
# X = Hours studied
# y = Marks obtained
# ----------------------------
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=float)
y = np.array([35, 40, 45, 50, 55, 60, 65, 70, 75], dtype=float)

# ----------------------------
# Step 2: Shuffle the data
# (so that splitting is fair)
# ----------------------------
np.random.seed(42)
 # same shuffle every time
idx = np.random.permutation(len(X)) # random order of indexes
X, y = X[idx], y[idx]
# shuffle X and y together

# ----------------------------
# Step 3: Split into 3 parts
# Train = 60%
# Validation = 20%
# Test = 20%
# ----------------------------
n = len(X)
train_end = int(0.6 * n)
val_end = int(0.8 * n)

X_train, y_train = X[:train_end], y[:train_end]
X_val, y_val    = X[train_end:val_end], y[train_end:val_end]
X_test, y_test  = X[val_end:], y[val_end:]

print("Training Data:", X_train, y_train)
print("Validation Data:", X_val, y_val)
print("Test Data:", X_test, y_test)

# ----------------------------
# Step 4: Train a simple model
# Using numpy polyfit:
# y = a*x + b
# ----------------------------
a, b = np.polyfit(X_train, y_train, 1)

# ----------------------------
# Step 5: Check on Validation
# Find prediction + MSE error
# ----------------------------
val_pred = a * X_val + b
val_mse = np.mean((y_val - val_pred) ** 2)

print("\nModel equation: y = {:.2f}x + {:.2f}".format(a, b))
print("Validation MSE:", round(val_mse, 2))
```

Output:



```
Training Data: [8. 2. 6. 1. 9.] [70. 40. 60. 35. 75.]
Validation Data: [3. 5.] [45. 55.]
Test Data: [4. 7.] [50. 65.]

Model equation: y = 5.00x + 30.00
Validation MSE: 0.0
```

- **Training Data:** Model learns the pattern (study hours → marks)
- **Validation Data:** Used to choose best settings (like which model or best parameters)
- **Test Data:** Final check to see model performance on unseen dat

3.2.3 Overfitting and Underfitting

These are common problems in machine learning models.

1. Underfitting

Underfitting occurs when a model is **too simple** to learn patterns from the data.

- Performs poorly on **training data**
- Performs poorly on **test data**

Causes:

- Very simple model
- Too few features
- Insufficient training

Example: Trying to predict house prices using only the number of rooms and ignoring location, area, and amenities.

2. Overfitting

Overfitting occurs when a model learns the **training data too well**, including noise.

- Performs very well on training data
- Performs poorly on test data

Causes:

- Too complex model
- Too many features
- Very small training dataset

Example: A model that memorizes answers instead of understanding concepts.

Underfitting	Overfitting
Model is too simple	Model is too complex
Low accuracy everywhere	High training accuracy, low test accuracy
Fails to learn patterns	Learns noise and details

Python Implementation

```
import numpy as np

# ------------------------
# Simple dataset: Hours -> Marks
# ------------------------
X = np.array([1, 2, 3, 4, 5, 6, 7, 8], dtype=float)
y = np.array([30, 35, 45, 50, 60, 65, 80, 85], dtype=float)

# Train = first 6 points, Test = last 2 points
X_train, y_train = X[:6], y[:6]
X_test,  y_test  = X[6:], y[6:]

# MSE function (error)
def mse(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)

# ==========================================================
# 1) UNDERFITTING
# Very simple model: predicts SAME marks for everyone
# (mean of training marks)
# ==========================================================
mean_marks = y_train.mean()

train_pred_under = np.full(len(y_train), mean_marks)
test_pred_under  = np.full(len(y_test),  mean_marks)

print("UNDERFITTING (same prediction for all)")
print("Train MSE:", round(mse(y_train, train_pred_under), 2))
print("Test  MSE:", round(mse(y_test,  test_pred_under),  2))

# ==========================================================
# 2) OVERFITTING
# Very complex model: high-degree polynomial
# Fits training too closely, may fail on test
# ==========================================================
degree = 5
coef = np.polyfit(X_train, y_train, degree)

train_pred_over = np.polyval(coef, X_train)
test_pred_over  = np.polyval(coef, X_test)

print("\nOVERFITTING (complex polynomial)")
print("Train MSE:", round(mse(y_train, train_pred_over), 2))
print("Test  MSE:", round(mse(y_test,  test_pred_over),  2))
```

Output:

```
UNDERFITTING (same prediction for all)
Train MSE: 156.25
Test  MSE: 1231.25

OVERFITTING (complex polynomial)
Train MSE: 0.0
Test  MSE: 85625.0
```

Balanced (Good Fit) Model

A good model:

- Learns important patterns
- Performs well on both training and test data

RULES:

Underfitting: Train MSE high + Test MSE high
Overfitting: Train MSE very low + Test MSE high

3.3 Loss Functions

In machine learning, a **loss function** measures **how wrong a model's predictions are.** It compares **actual values** with **predicted values** and returns a numerical value.

Lower loss = Better model performance

Loss functions are mainly used in **regression problems**.

3.3.1 Mean Squared Error (MSE)

Mean Squared Error (MSE) calculates the **average of the squared differences** between actual and predicted values.

Formula

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

Where:

- yi = actual value
- y^i = predicted value
- n = number of observations

3.3.2 Definition of MSE

MSE measures the **average of the squares of the errors**. An "error" is simply the distance between the actual value (y) and the predicted value (y^).

- Find the difference between actual and predicted values
- Square the difference
- Take the average of all squared values

Real-World Example: House Price Prediction

Actual Price	Predicted Price

Actual Price	Predicted Price
50	45
60	65
55	52

Errors:
$(50-45)^2 = 25$
$(60-65)^2 = 25$
$(55-52)^2 = 9$

MSE = (25+25+9)/3 = 19.67

3.3.2.1 Computing MSE and its Properties

To calculate MSE, you follow these steps:

1. Find the difference between the actual and predicted value for every data point.
2. Square each of those differences.
3. Sum them all up.
4. Divide by the total number of data points (n).

The Formula:

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

Key Properties:

- **Always Positive:** Because we square the errors, MSE can never be negative.
- **Penalizes Outliers:** Because the error is squared, a big mistake is penalized much more heavily than a small one. (e.g., an error of 10 becomes a loss of 100, while an error of 2 is only 4).
- **Differentiable:** This makes it very "math-friendly" for optimization algorithms.

Example of Computing MSE

Actual (y)	Predicted (ŷ)	Error (y – ŷ)	Squared Error $(y - \hat{y})^2$
50	45	5	25
60	70	-10	100
80	75	5	25

$$MSE = \frac{25 + 100 + 25}{3} = \frac{150}{3} = 50$$

So, **MSE = 50**.

It measures the average of the **squared difference** between actual values and predicted values.
It is used to check how well a regression model is performing (lower MSE = better model).

3.3.3 Mean Absolute Error (MAE)

MAE is a simpler, more intuitive way to look at error.

3.3.3.1 Definition of MAE

MAE is the **average of the absolute differences** between predicted and actual values. It treats all errors equally regardless of their size.
Lower MAE = better model.

3.3.3.2 Computing MAE and its Properties

1. Calculate the difference between actual and predicted values.
2. Take the absolute value (ignore the negative sign).
3. Average these values.

The Formula:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

Example:

Actual	Predicted	Absolute Error
50	45	5
60	65	5
55	52	3

MAE = (5+5+3)/3 = 4.33

Key Properties:

- **Robust to Outliers ():** Since we don't square the error, a single massive mistake doesn't "blow up" the total loss as much as it does in MSE.
- **Easy Interpretation:** The result is in the same units as your data. If you are predicting house prices and your MAE is 5,000, it means your predictions are off by $5,000 on average.
- **Linear Penalty:** Treats all errors equally
- **Non-negative:** MAE ≥ 0

Python Implementation

```
import numpy as np

# Actual values and Predicted values
y = np.array([50, 60, 70, 80])        # actual
y_hat = np.array([52, 58, 75, 78])     # predicted

# ------------------------
# MAE = mean(|y - y_hat|)
# ------------------------
mae = np.mean(np.abs(y - y_hat))

# ------------------------
# MSE = mean((y - y_hat)^2)
# ------------------------
mse = np.mean((y - y_hat) ** 2)
```

Output:

```
MAE = 2.75

MSE = 9.25
```

print("MAE =", mae)
print("MSE =", mse)	

Difference between MSE (Mean Squared Error) VS MAE (Mean Absolute Error)

Point	**MSE (Mean Squared Error)**	**MAE (Mean Absolute Error)**
Formula	$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$	$MAE = \frac{1}{n}\sum_{i=1}^{n}\lvert y_i - \hat{y}_i\rvert$
Error type	Squared error	Absolute error
Effect of large errors	**Penalizes more** (large errors become very big after squaring)	**Penalizes less** (large error increases linearly)
Outlier sensitivity	**More sensitive** to outliers	**Less sensitive** to outliers
Units	Output unit becomes **squared** (e.g., marks^2)	Same unit as output (e.g., marks)
When preferred	When you want to strongly punish big mistakes	When you want a simple, robust average error

3.3.3.3 Understanding Regression and R²

Regression is used when the output is a **number** (continuous value).
Example: Predicting **marks**, **house price**, **rainfall (mm)**.

While MSE and MAE tell us the "error," **R² (R-Squared)**, or the **Coefficient of Determination**, tells us about the "goodness of fit."

R² score measures **how well the regression model explains the variance** in the data.

- **What it represents:** It measures the proportion of the variance in the dependent variable that is predictable from the independent variables.
- **The Scale: R²** usually ranges from **0 to 1** (or 0% to 100%).
  - **R² = 0:** The model explains none of the variability (it's as good as just guessing the average every time).
  - **R² = 1:** The model perfectly predicts every single data point.
  - **R² < 0**: The model is worse than predicting the average

The Formula:

$$R^2 = 1 - \frac{\text{Residual Sum of Squares}}{\text{Total Sum of Squares}}$$

R² Value Interpretation:

R² Value	**Meaning**
1.0	Perfect prediction
0.8	Very good model
0.0	No improvement over mean
Negative	Very poor model

Example for Actual vs Predicted Marks

Student	Actual (y)	Predicted (y^)
1	50	52
2	60	58
3	70	72

Step 1: Find mean of actual values $\bar{y}$

$$\bar{y} = \frac{50 + 60 + 70}{3} = 60$$

Step 2: Compute SS_{res}

$$SS_{res} = (50 - 52)^2 + (60 - 58)^2 + (70 - 72)^2$$
$$= (-2)^2 + (2)^2 + (-2)^2 = 4 + 4 + 4 = 12$$

Step 3: Compute SS_{tot}

$$SS_{tot} = (50 - 60)^2 + (60 - 60)^2 + (70 - 60)^2$$
$$= (-10)^2 + 0^2 + 10^2 = 100 + 0 + 100 = 200$$

Step 4: Compute R^2

$$R^2 = 1 - \frac{12}{200} = 1 - 0.06 = 0.94$$

Python Implementation:

```
import numpy as np

# Actual values (true marks)
y = np.array([50, 60, 70])

# Predicted values (model output)
y_hat = np.array([52, 58, 72])

# Step 1: Find average of actual values
y_mean = y.mean()

# Step 2: Calculate total error in predictions (SSE)
sse = np.sum((y - y_hat) ** 2)

# Step 3: Calculate total variation in actual data (SST)
sst = np.sum((y - y_mean) ** 2)

# Step 4: R² formula
r2 = 1 - (sse / sst)

print("R² =", round(r2, 2))
```

So output, R^2=0.94
Meaning: the model explains about **94%** of the variation in marks (very good fit)

If you predict a student's grade using only **study hours**, an **R^2=0.85** means **85% of the differences in grades** among students can be explained by **how much they studied.** The remaining **15%** is due to other factors like sleep, health, stress, teaching quality, prior knowledge, or exam difficulty.

Example: House Price Prediction (Actual vs Predicted House Prices (in lakhs))

House	Actual (y)	Predicted (y^)
1	50	52
2	60	59
3	70	68

Python Implementation:

```
import numpy as np

# Actual and predicted house prices (in lakhs)
actual = np.array([50, 60, 70])
pred   = np.array([52, 59, 68])

# Step 1: Mean (average) of actual values
avg = actual.mean()

# Step 2: Prediction error (how far predictions are from actual)
ss_res = np.sum((actual - pred) ** 2)

# Step 3: Total variation in actual data (how far actual values are from average)
ss_tot = np.sum((actual - avg) ** 2)

# Step 4: R² score
r2 = 1 - (ss_res / ss_tot)

print("R² =", round(r2, 3))
```

Output: R^2=0.955

Meaning: the model explains about **95.5%** of the variation in house prices (very good).

Day	Actual Temp (°C) (y)	Predicted Temp (°C) (y^) *(mean only)*
1	20	25
2	25	25
3	30	25

Actual temperature (°C)

$$y = [20,\ 25,\ 30]$$

Mean temperature:

$$\bar{y} = \frac{20+25+30}{3} = 25$$

Predicted temperature (model predicts only the mean)

$$\hat{y} = [25,\ 25,\ 25]$$

R^2 calculation

$$SS_{res} = (20-25)^2 + (25-25)^2 + (30-25)^2 = 25 + 0 + 25 = 50$$

$$SS_{tot} = (20-25)^2 + (25-25)^2 + (30-25)^2 = 50$$

$$R^2 = 1 - \frac{50}{50} = 0$$

Python Implementation:

```
import numpy as np

# Actual temperatures
actual = np.array([20, 25, 30])

# Predicted temperatures (model predicts the same value for all)
pred = np.array([25, 25, 25])

# Step 1: Average of actual temperatures
avg = actual.mean()

# Step 2: Error in prediction (how far predictions are from actual)
ss_res = np.sum((actual - pred) ** 2)

# Step 3: Total variation in actual values (how far actual values are from average)
ss_tot = np.sum((actual - avg) ** 2)

# Step 4: R² score
r2 = 1 - (ss_res / ss_tot)

print("R² =", round(r2, 2))
```

So, R^2=0
Meaning: predicting only the average temperature is **not better** than this model

Important Key Points

- **Supervised Learning**: Learns from **labeled data** (input + correct output).
- **Classification**: Predicts **categories/classes** (e.g., spam/ham, pass/fail).
- **Regression**: Predicts **numbers/continuous values** (e.g., price, marks, temperature).
- **Dataset**: Complete data used for learning.
- **Features (X)**: Input columns used for prediction.
- **Labels/Target (y)**: Output column to be predicted.
- **Training Data**: Used to **fit/learn** the model.
- **Validation Data**: Used to **tune** model settings and reduce overfitting.
- **Test Data**: Used for **final performance checking** on unseen data.
- **Underfitting**: Model is **too simple** → poor on train and test.
- **Overfitting**: Model **memorizes** training data → good on train, poor on test.
- **Loss Function**: Measures prediction error (smaller is better).
- **MSE (Mean Squared Error)**: Squares errors → **penalizes large errors more**, sensitive to outliers.
- **MAE (Mean Absolute Error)**: Absolute errors → **easier to understand**, less effect of outliers.
- **R^2 (Regression Score)**: Measures model fit (**1 = best**, **0 = weak**, negative = worse than mean).