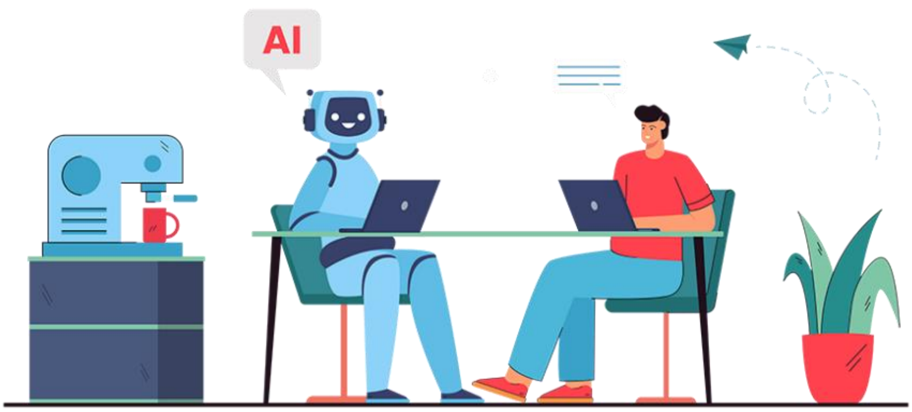
Reliance Foundation

N·S·D·C RE-IMAGINE FUTURE

Supervised Model Evaluation

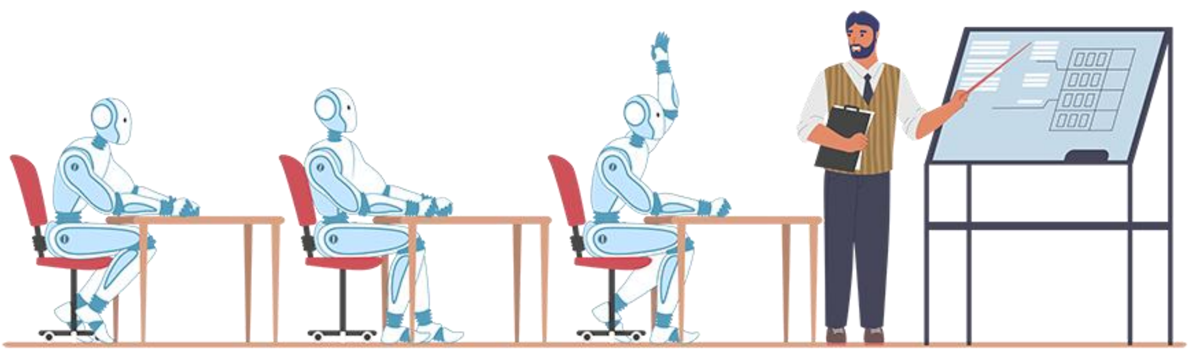
The supervised model is trained on labelled data, meaning each input (features) is paired with the correct output (labels). The goal is to learn a mapping from input to output. For example, predicting house prices (regression) or classifying emails as spam/not spam (classification).



Alt text: *Supervised learning*

Training Process

- **Goal:** Minimise the error between the predicted and actual output by adjusting the model's parameters.
- **Process:**



Alt text: *Training process*

- **Prepare the Data:** Collect and pre-process the dataset, ensuring it has labelled input-output pairs. Clean the data, handle missing values, and split it into features (X) and labels (y).

- **Split the Data:** Divide the dataset into training sets and test sets (commonly 80/20 or 70/30). The training set is used to train the model, while the test set evaluates its performance.
- **Choose the Model:** Select an appropriate supervised learning algorithm (e.g., linear regression, decision trees, or support vector machines) based on the problem type (classification or regression).
- **Train the Model:** Use the training data to fit the model. The algorithm learns the patterns by adjusting its internal parameters to minimise error.

a. Evaluating the Models

- **Use a Test Set:** After training, test the model on a separate dataset (test set) that wasn't used during training.
- **Choose Evaluation Metrics:**
  - **Classification:** Use metrics like accuracy, precision, recall, **F1 score, and the ROC-AUC score**.
  - **Regression:** Use metrics like **mean squared error** (MSE), **mean absolute error** (MAE), and **R-squared**.
- **Confusion Matrix:** For classification, a **confusion matrix** helps visualize true positives, false positives, true negatives, and false negatives.
- **Cross-Validation:** Apply techniques like **k-fold cross-validation** to assess the model's performance across different subsets of the data and reduce overfitting.
- **Compare Baselines:** Compare the model's performance against simple baseline models or previous versions to assess improvements.

This process trains and evaluates a supervised model to make predictions based on labelled data.

Example:

Let us develop a supervised model and evaluate it.

Scenario: Predicting Housing Prices

Objective: Train a model to predict house prices based on features like the size of the house in square feet.

Here is the code for the above objective:

```
# Import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Data: Features (house sizes in square feet) and Target
(house prices)
X = [[800], [1000], [1200], [1500], [2000]]  # Features
(house sizes)
y = [200000, 250000, 300000, 350000, 500000]  # Target
(house prices)

# Split the data into training and test sets (80% training,
20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Initialize the Linear Regression model
model = LinearRegression()

# Train the model using the training data
model.fit(X_train, y_train)

# Predict the house prices for the test set
y_pred = model.predict(X_test)

# Print the test set house sizes and corresponding
predicted prices
print("Test Set House Sizes and Predicted Prices:")
for size, price in zip(X_test, y_pred):
    print(f"House Size: {size[0]} sq ft, Predicted Price:
Rs {price:,.2f}")

# Evaluate the model using Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print(f"\nMean Squared Error: {mse:.2f}")
```

Explanation:

Here is a breakdown of the code step by step:

1. **Import necessary libraries**
   - train_test_split is used to split the dataset into training and testing sets.
   - LinearRegression is the machine learning algorithm we use for predicting house prices.

```
# Import necessary libraries
from sklearn.model_selection import train_test_split
    from sklearn.linear_model import LinearRegression
```

2. **Prepare the Data**
   - **Data Collection:** Gather historical data on housing prices. For this case study, the dataset includes features such as house size (in square feet) and corresponding prices.
   - **Pre-processing:** Clean the dataset by handling missing values and ensuring all data is in a usable format. Split the dataset into features (X) and labels (y).

```
# Example data

X = [[800], [1000], [1200], [1500], [2000]]  # Features
(house sizes in sq ft)

y = [200000, 250000, 300000, 350000, 500000]  # Target
(house prices in Rs)
```

3. **Split the Data**
   - **Training and Test Sets:** Divide the dataset into training (80%) and test sets (20%) to evaluate the model's performance.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

4. **Choose the Model**
   - **Model Selection:** Choose a linear regression model for this case, as it is suitable for predicting continuous values like house prices.

```
model = LinearRegression()
```

5. Train the Model

- **Model Training:** Fit the linear regression model using the training data.

```
model.fit(X_train, y_train)
```

6. Predicting House Prices

```
# Predict the house prices for the test set
y_pred = model.predict(X_test)
```

- The trained model is used to predict house prices for the test set X_test.
- The predictions (y_pred) represent the model's estimated prices based on the test data (house sizes).

7. Displaying the Predictions

```
# Print the test set house sizes and corresponding predicted prices
print("Test Set House Sizes and Predicted Prices:")
for size, price in zip(X_test, y_pred):
    print(f"House Size: {size[0]} sq ft, Predicted Price: Rs {price:,.2f}")
```

- This loop prints the predicted house prices along with the corresponding house sizes.
- zip(X_test, y_pred) pairs each house size with its predicted price.
- The output is formatted to show the house size in square feet and the price in rupees (Rs).

8. Evaluating the Model using Mean Squared Error (MSE)

```
# Evaluate the model using Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print(f"\nMean Squared Error: {mse:.2f}")
```

- **mean_squared_error(y_test, y_pred):** This function compares the actual house prices (y_test) with the predicted prices (y_pred) and calculates the Mean Squared Error (MSE). MSE gives a measure of how well the model has performed. Lower MSE means the predictions are closer to the actual values.

Outcome: At this point, the model is trained, evaluated and can be used for prediction.

This case study illustrates the step-by-step process of creating a supervised model using linear regression for predicting housing prices.