Independent Skill Development Mission

Independent Skill Development Mission NXT
EMPOWERING FUTURES, SHAPING TOMORROW

ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION

📖 INTRODUCTION TO MACHINE LEARNING (ML) & SUPERVISED LEARNING

📌 CHAPTER 1: UNDERSTANDING MACHINE LEARNING

1.1 What is Machine Learning?

Machine Learning (ML) is a **subfield of artificial intelligence (AI)** that enables computers to learn from data, identify patterns, and make decisions **without being explicitly programmed**. It differs from traditional programming, where a human defines all the rules, in that ML models are trained to find relationships and patterns in **large datasets**.

Machine Learning works by training a model using **historical data** and then using that model to **predict outcomes or classify new data**. The accuracy of an ML model improves over time as it is exposed to more data and optimized for better performance.

Machine Learning is widely used in various fields, including:

✓ **Healthcare** – Predicting diseases based on symptoms.

✓ **Finance** – Detecting fraudulent transactions.

✓ **Retail & E-Commerce** – Recommending products based on browsing history.

✓ **Marketing** – Optimizing ad campaigns and targeting users based on behavior.

📌 **Example Use Case:**
Consider a **spam detection system** in an email application. Instead of manually writing rules for every possible spam keyword, an ML algorithm can **analyze past spam messages**, learn from patterns, and **automatically classify** new emails as spam or not spam.

💡 **Conclusion:**
Machine Learning **enables automation and enhances decision-making** by processing vast amounts of data quickly and efficiently. It plays a crucial role in the advancement of AI-powered applications.

📌 CHAPTER 2: TYPES OF MACHINE LEARNING

There are **three primary categories of Machine Learning**, each designed for different types of learning problems.

2.1 Supervised Learning

Supervised learning is the most widely used type of ML, where the model is trained on **labeled data** (input-output pairs). The system learns the relationship between inputs (features) and their corresponding outputs (labels) so that it can make predictions on unseen data.

🔹 **Key Features of Supervised Learning:**
✓ Requires a **dataset with labeled outputs**.
✓ The model **learns from past data** to make future predictions.
✓ Can be used for **both classification and regression tasks**.

📌 **Example:**
A **loan approval system** learns from past customer applications

(including features like credit score, income, and debt) to predict whether a **new applicant will be approved or denied**.

📌 **Common Algorithms for Supervised Learning:**

✓ **Linear Regression** – Predicts continuous values like house prices.

✓ **Logistic Regression** – Classifies binary outcomes (spam vs. not spam).

✓ **Decision Trees** – Makes decisions based on feature importance.

✓ **Support Vector Machines (SVM)** – Classifies complex datasets efficiently.

✓ **Neural Networks** – Mimic human brain structure for deep learning.

🔍 **Key Advantages:**

- **High accuracy** when sufficient labeled data is available.
- **Easy to interpret and evaluate model performance**.
- **Reliable predictions in controlled environments**.

🔍 **Challenges:**

- Requires **a large dataset with labeled examples,** which may not always be available.
- **Overfitting risk,** where models memorize training data rather than generalizing patterns.

2.2 Unsupervised Learning

Unsupervised learning deals with **unlabeled data** where the model identifies patterns and structures **without predefined categories**. Instead of predicting specific outcomes, it **groups similar data points together**.

🔹 **Key Features of Unsupervised Learning:**

✓ **No labeled data is required**; the model finds patterns on its own.

✓ Best for **data exploration, clustering, and anomaly detection**.

✓ Used for **customer segmentation, recommendation systems, and fraud detection**.

📌 **Example:**

An e-commerce company wants to categorize its customers based on purchasing behavior but doesn't have predefined segments. An **unsupervised clustering algorithm** can group similar customers based on spending habits, enabling the company to personalize marketing campaigns.

📌 **Common Algorithms for Unsupervised Learning:**

✓ **K-Means Clustering** – Groups similar data points into clusters.

✓ **Hierarchical Clustering** – Creates a tree of nested clusters.

✓ **Principal Component Analysis (PCA)** – Reduces dataset dimensionality for visualization.

✓ **Autoencoders** – Used for anomaly detection in network security.

🔍 **Key Advantages:**

- **Useful for finding hidden patterns** in large datasets.
- **No need for labeled data,** making it cost-effective.
- Can process **large, unstructured datasets like customer data and social media trends**.

🔍 **Challenges:**

- **Less interpretable than supervised models**.
- May find **patterns that are not meaningful** if not used correctly.

📌 CHAPTER 3: SUPERVISED LEARNING IN DETAIL

3.1 How Supervised Learning Works

Supervised learning follows a structured process:

✓ **Step 1: Collect & Label Data** – Gather historical data with known labels.
✓ **Step 2: Split Data into Training & Test Sets** – Typically, **80% for training, 20% for testing**.
✓ **Step 3: Train the Model** – Apply an ML algorithm to **learn from data patterns**.
✓ **Step 4: Evaluate Performance** – Use accuracy, precision, recall, or RMSE to assess results.
✓ **Step 5: Make Predictions** – The model classifies or predicts new data points.

📌 **Example:**
A **credit card fraud detection system** is trained on past transaction data (labeled as fraud or not fraud). It learns patterns and **flags suspicious transactions in real-time**.

🔍 **Key Benefits:**

- Provides **accurate predictions** when sufficient labeled data is available.
- **Used across industries,** from **healthcare** (disease prediction) to **finance** (loan approval).

🔍 **Challenges:**

- Requires **high-quality labeled data,** which may be expensive to obtain.

- May suffer from **overfitting,** where the model performs well on training data but poorly on new data.

💡 **Conclusion:**

Supervised learning is **the foundation of AI-driven decision-making,** enabling automation in various industries.

📌 CHAPTER 4: APPLICATIONS OF SUPERVISED LEARNING

🔹 **Healthcare:**

✓ Predicts diseases based on medical history.

✓ Analyzes X-rays and MRI scans for abnormalities.

🔹 **Finance & Banking:**

✓ Fraud detection for credit card transactions.

✓ Approving or rejecting loan applications.

🔹 **E-Commerce & Retail:**

✓ Customer segmentation (identifying high-value customers).

✓ Product recommendations based on past purchases.

🔹 **Self-Driving Cars:**

✓ Classifies road signs, pedestrians, and obstacles.

✓ Predicts collision risks based on real-time data.

📌 **Example:**

Amazon uses supervised learning to recommend products based on user preferences.

💡 **Conclusion:**

Supervised learning is the foundation of modern AI applications, helping businesses make accurate predictions and automate decision-making.

📌 CHAPTER 5: EXERCISES & ASSIGNMENTS

5.1 Multiple Choice Questions

✅ **What is the main characteristic of supervised learning?**

☐ (a) It does not require labeled data

☐ (b) It learns from labeled data ✅

☐ (c) It finds hidden patterns in unlabeled data

☐ (d) It only works for text-based data

✅ **Which algorithm is best suited for predicting house prices?**

☐ (a) Logistic Regression

☐ (b) Linear Regression ✅

☐ (c) Naïve Bayes

☐ (d) Decision Trees

5.2 Practical Assignment

📌 **Task:**

1. Choose a dataset with labeled data (House Prices, Customer Purchases, Loan Approval).
2. Apply Supervised Learning (Classification or Regression).
3. Train and evaluate the model using Python (Scikit-learn).
4. Interpret the results and discuss improvements.

📖 Regression Techniques: Linear, Multiple & Polynomial Regression

📌 CHAPTER 1: UNDERSTANDING REGRESSION ANALYSIS

1.1 What is Regression?

Regression is a **fundamental statistical and machine learning technique** used to model the relationship between **independent variables (predictors)** and a **dependent variable (target)**. It is widely used in forecasting, risk assessment, and data analysis across industries.

Regression analysis helps in:
✓ **Predicting future values** based on historical data.
✓ **Understanding relationships** between variables.
✓ **Identifying trends and patterns** in data.

📌 **Example Use Case:**
A retail business wants to predict future sales based on advertising spending. Regression analysis can model this relationship and forecast how much sales will increase with additional advertising investment.

💡 **Conclusion:**
Regression is a powerful tool for **quantifying relationships** between variables and making **data-driven decisions**.

📌 CHAPTER 2: LINEAR REGRESSION

2.1 What is Linear Regression?

Linear Regression is the **simplest form of regression analysis,** where a straight line is fitted to the data to model the relationship between the **independent variable (X)** and the **dependent variable (Y)**.

📌 **Formula for Simple Linear Regression:**

$Y=mX+bY = mX + b$

where:

- YY = Predicted value (dependent variable)
- XX = Predictor variable (independent variable)
- mm = Slope of the line (rate of change)
- bb = Intercept (value of Y when $X = 0$)

📌 **Example:**
A company analyzes data on **years of experience (X)** and **salary (Y)**. The model can predict salary based on experience using a best-fit line.

2.2 How Linear Regression Works

1️⃣ **Collect Data:** Historical data of predictors and target variables.
2️⃣ **Split Data into Training & Testing Sets:** Typically **80% for training, 20% for testing**.
3️⃣ **Fit the Regression Model:** Calculate the best-fit line that minimizes errors.
4️⃣ **Evaluate Model Performance:** Use metrics like **R^2 Score, Mean Squared Error (MSE)**.
5️⃣ **Make Predictions:** Apply the trained model to new data points.

2.3 Assumptions of Linear Regression

✓ **Linearity** – The relationship between X and Y should be linear.
✓ **Independence** – Data points should be independent of each other.
✓ **Homoscedasticity** – Variability of errors should be constant.
✓ **No Multicollinearity** – Independent variables should not be highly correlated.

📌 **Example in Python:**

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error


# Sample Data (Years of Experience vs. Salary)

data = {'Experience': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

    'Salary': [30000, 35000, 40000, 45000, 50000, 60000, 65000, 70000, 75000, 80000]}

df = pd.DataFrame(data)


# Splitting data

X = df[['Experience']]

y = df['Salary']
```

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


# Train Linear Regression Model

model = LinearRegression()

model.fit(X_train, y_train)


# Predictions

y_pred = model.predict(X_test)


# Model Performance

mse = mean_squared_error(y_test, y_pred)

r2 = model.score(X_test, y_test)


print(f"Mean Squared Error: {mse}")

print(f"R² Score: {r2}")


# Visualization

plt.scatter(X, y, color="blue", label="Actual Data")

plt.plot(X, model.predict(X), color="red", label="Regression Line")

plt.xlabel("Years of Experience")
```

```
plt.ylabel("Salary")

plt.legend()

plt.show()
```

💡 **Conclusion:**
Linear regression is useful for **predicting numerical values** when a **clear linear relationship** exists between variables.

📌 CHAPTER 3: MULTIPLE LINEAR REGRESSION

3.1 What is Multiple Linear Regression?

Multiple Linear Regression is an **extension of Simple Linear Regression,** where more than **one independent variable (X)** is used to predict the dependent variable (Y).

📌 **Formula for Multiple Linear Regression:**

Y=bo+b1X1+b2X2+...+bnXnY = b_0 + b_1X_1 + b_2X_2 + ... + b_nX_n

where:

- YY = Predicted value
- X1,X2,...,XnX_1, X_2, ..., X_n = Multiple predictor variables
- bob_0 = Intercept
- b1,b2,...,bnb_1, b_2, ..., b_n = Coefficients of independent variables

📌 **Example Use Case:**
A real estate company predicts **house prices** based on multiple factors like **square footage, number of bedrooms, location, and age of the house**.

3.2 Assumptions of Multiple Regression

✓ **No Multicollinearity:** Independent variables should not be highly correlated.
✓ **Homoscedasticity:** Variance of residuals should be constant.
✓ **Linearity:** Relationship between independent and dependent variables should be linear.

📌 **Example in Python:**

```
# Load dataset
X = df[['Square_Feet', 'Bedrooms', 'Bathrooms']]
y = df['House_Price']

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Model
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Model Performance
```

print(f"R² Score: {model.score(X_test, y_test)}")

💡 **Conclusion:**
Multiple regression is more powerful than simple regression, allowing **more factors to be considered** for better accuracy.

📌 CHAPTER 4: POLYNOMIAL REGRESSION

4.1 What is Polynomial Regression?

Polynomial Regression is a **type of regression analysis** that captures the **non-linear relationship** between the independent variable(s) and the dependent variable by introducing polynomial terms.

📌 **Formula for Polynomial Regression:**

Y=b0+b1X+b2X2+b3X3+...+bnXnY = b_0 + b_1X + b_2X^2 + b_3X^3 + ... + b_nX^n

where **higher-degree polynomial terms** allow capturing **curved trends** in data.

📌 **Example Use Case:**
Predicting **population growth, demand forecasting, or stock price trends**, where the relationship between variables is **non-linear**.

📌 **Example in Python:**

from sklearn.preprocessing import PolynomialFeatures

from sklearn.pipeline import make_pipeline

Generate polynomial features

```
degree = 2  # Quadratic

poly_model = make_pipeline(PolynomialFeatures(degree), LinearRegression())


# Train Model

poly_model.fit(X_train, y_train)


# Predictions

y_pred = poly_model.predict(X_test)


# Model Performance

print(f"R² Score: {poly_model.score(X_test, y_test)}")
```

💡 **Conclusion:**
Polynomial Regression **fits complex, non-linear data trends,** making it ideal for **real-world forecasting problems**.

📌 CHAPTER 5: COMPARISON OF REGRESSION TECHNIQUES

Regression Type	**Use Case**	**Best For**
Linear Regression	Predicts continuous values	Simple relationships
Multiple Regression	Uses multiple variables	Complex, multi-factor problems

Polynomial Regression	Captures curved trends	Non-linear relationships

💡 **Key Takeaways:**

- **Linear Regression** works for **simple relationships** between variables.
- **Multiple Regression** considers **multiple factors** for better accuracy.
- **Polynomial Regression** is useful when **data shows non-linear patterns**.

📌 **Next Steps:**

🔹 Work on **real-world datasets** like house price prediction.

🔹 Learn **advanced regression techniques** (Ridge, Lasso).

🔹 Explore **Machine Learning model tuning techniques** to improve accuracy.

📖 Classification Techniques: Logistic Regression, Decision Trees, Random Forest

📌 CHAPTER 1: UNDERSTANDING CLASSIFICATION IN MACHINE LEARNING

1.1 What is Classification?

Classification is a **supervised machine learning technique** that involves predicting the category or class of a given input based on **previously labeled training data**. The output of a classification model is discrete, meaning it assigns inputs to predefined groups.

🔹 Why is Classification Important?

✓ **Automates Decision-Making:** Used in fraud detection, spam filtering, and medical diagnosis.

✓ **Handles Categorical Outputs:** Unlike regression (which predicts continuous values), classification deals with **Yes/No, True/False, or multiple categories**.

✓ **Improves Business Operations:** Helps in **customer segmentation, sentiment analysis, and product recommendation.**

📌 Example Use Case:

A bank uses classification to determine **whether a loan applicant will default** based on their credit history, income, and employment status.

💡 **Conclusion:**
Classification is widely used in various fields like healthcare, finance, and marketing, making it **a key area in machine learning.**

📌 CHAPTER 2: LOGISTIC REGRESSION FOR CLASSIFICATION

2.1 What is Logistic Regression?

Logistic Regression is a **statistical method** used for **binary classification** (Yes/No, 0/1, Spam/Not Spam). Unlike Linear Regression, which predicts continuous values, **Logistic Regression outputs probabilities between 0 and 1**, which are mapped to class labels.

🔹 Key Features of Logistic Regression:

✓ Predicts **binary or multi-class classification problems**.
✓ Uses a **sigmoid function** to transform output values into probabilities.
✓ Easy to interpret and implement in real-world scenarios.

📌 Mathematical Representation:

Logistic Regression uses the **sigmoid function** to map predictions between **0 and 1**:

P(Y=1)=11+e−(bo+b1X1+b2X2+...+bnXn)P(Y=1) = \frac{1}{1+e^{-(b_0 + b_1X_1 + b_2X_2 + ... + b_nX_n)}}

where:

- P(Y=1)P(Y=1) is the probability of the event occurring.
- ee is Euler's number (approximately 2.718).
- bo,b1,b2b_0, b_1, b_2 are coefficients learned from data.

- X1,X2,XnX_1, X_2, X_n are input features.

📌 **Example:**
A Logistic Regression model can predict **whether an email is spam or not** based on **word frequency, sender details, and message length**.

2.2 Implementing Logistic Regression in Python

📌 **Python Code Example:**

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score


# Load dataset

df = pd.read_csv("customer_churn.csv")  # Example dataset


# Selecting features and target variable

X = df[['monthly_charges', 'tenure', 'total_charges']]

y = df['churn']  # 1 = Churn, 0 = No Churn


# Splitting data
```

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


# Train model

model = LogisticRegression()

model.fit(X_train, y_train)


# Predict on test data

y_pred = model.predict(X_test)


# Evaluate accuracy

print("Accuracy:", accuracy_score(y_test, y_pred))
```

🔍 **Key Insights:**

- Logistic Regression is **easy to implement and works well for binary classification problems**.
- If the dataset has **non-linear relationships**, Logistic Regression may not perform well, requiring more advanced techniques.

💡 **Conclusion:**

Logistic Regression is a powerful tool for **binary classification problems**, but it may not be suitable for **complex, non-linear relationships**.

📌 CHAPTER 3: DECISION TREES FOR CLASSIFICATION

3.1 What is a Decision Tree?

A **Decision Tree** is a **tree-like model** used for classification tasks. It splits data into **branches** based on feature conditions, making it **easy to interpret**.

🔹 Key Features of Decision Trees:

✓ **Rule-based classification** – Breaks down decisions into step-by-step logic.
✓ **Handles categorical & numerical data**.
✓ **Interpretable model** – Decision paths can be visualized.

📌 Example:

A **decision tree for loan approvals** might have rules like:

1. If **income > $50,000**, approve the loan.
2. If **income < $50,000** but **credit score > 700**, approve the loan.
3. If **income < $50,000** and **credit score < 700**, deny the loan.

📌 Tree Structure:

```
Income > 50K?
/     \
Yes       No
Approve   Credit Score > 700?
          /       \
      Approve     Deny
```

3.2 Implementing Decision Trees in Python

📌 Python Code Example:

```
from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score


# Train Decision Tree Model

tree_model = DecisionTreeClassifier(max_depth=3)

tree_model.fit(X_train, y_train)


# Predict on test data

y_pred_tree = tree_model.predict(X_test)


# Evaluate model

print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_tree))
```

🔍 Key Insights:

- Decision Trees **work well for both classification & regression problems**.
- They are **prone to overfitting,** meaning they can perform **too well on training data but poorly on unseen data**.
- Setting a **maximum depth** helps control overfitting.

💡 Conclusion:

Decision Trees are highly **interpretable and useful for rule-based decision-making,** but **they may overfit if not pruned properly**.

📌 CHAPTER 4: RANDOM FOREST FOR CLASSIFICATION

4.1 What is Random Forest?

Random Forest is an **ensemble learning technique** that creates **multiple decision trees** and combines their predictions to improve accuracy and **reduce overfitting**.

🔹 Key Features of Random Forest:

✓ **Uses multiple decision trees** to enhance predictions.
✓ **Reduces overfitting** compared to individual decision trees.
✓ **Can handle large datasets efficiently**.

📌 How It Works:

- The dataset is **randomly split into multiple subsets**.
- Multiple **decision trees are trained** on different subsets.
- The final prediction is **based on the majority vote** from all trees.

📌 Example:

A **credit risk prediction model** uses multiple trees, each trained on different sets of customer data. The final model combines all tree predictions to determine **whether a customer is likely to default on a loan**.

4.2 Implementing Random Forest in Python

📌 Python Code Example:

```
from sklearn.ensemble import RandomForestClassifier
```

```
# Train Random Forest Model

rf_model = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=42)

rf_model.fit(X_train, y_train)


# Predict on test data

y_pred_rf = rf_model.predict(X_test)


# Evaluate accuracy

print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
```

🔍 **Key Insights:**

- Random Forest is **more accurate than a single Decision Tree**.
- Works well for **large datasets** but is **computationally expensive**.
- The more trees, the **higher the accuracy**, but too many trees **increase computation time**.

💡 **Conclusion:**

Random Forest is an excellent **high-accuracy classification model** that **reduces overfitting** while maintaining strong predictive performance.

📌 CHAPTER 5: SUMMARY & NEXT STEPS

✅ Key Takeaways:

✓ **Logistic Regression** is best for **binary classification problems**.

✓ **Decision Trees** are **rule-based classifiers** that work well for interpretable decision-making.

✓ **Random Forest** is an **ensemble method** that reduces overfitting and improves accuracy.

📌 Next Steps:

- Work with **real-world datasets** to apply classification models.
- Tune hyperparameters using **GridSearchCV** to improve performance.
- Implement **deep learning classification models** for complex problems.

📖 Model Evaluation: Accuracy, Precision, Recall, F1-Score

📌 CHAPTER 1: INTRODUCTION TO MODEL EVALUATION

1.1 What is Model Evaluation?

Model evaluation is the process of **assessing the performance of a machine learning model** to determine how well it generalizes to unseen data. A good model should make **accurate and reliable predictions** while minimizing errors.

🔹 Why is Model Evaluation Important?

✓ **Prevents Overfitting** – Ensures that the model does not just memorize training data but performs well on new data.
✓ **Measures Performance** – Helps understand whether a model is good enough for real-world applications.
✓ **Guides Model Selection** – Helps in choosing between different algorithms and tuning hyperparameters.

📌 Example Use Case:

A bank develops a machine learning model to predict **loan approvals**. Model evaluation ensures that **predictions are accurate** and that the model does not misclassify **high-risk customers as safe applicants**.

💡 Conclusion:

Without proper evaluation, a model may appear to work well but fail when deployed in real-world scenarios. Evaluation metrics help **quantify a model's effectiveness** and guide improvements.

📌 CHAPTER 2: COMMON METRICS FOR MODEL EVALUATION

2.1 Key Evaluation Metrics for Classification Models

Classification models predict categorical outcomes (e.g., spam/not spam, disease/no disease). The **four most important classification evaluation metrics** are:

✓ **Accuracy** – Measures overall correctness.
✓ **Precision** – Measures how many predicted positives are actually correct.
✓ **Recall (Sensitivity)** – Measures how well the model identifies actual positives.
✓ **F1-Score** – A balance between Precision and Recall.

📌 **Example:**
A machine learning model predicts whether a customer will default on a loan. We must ensure that:

- **High-risk customers are not misclassified as safe applicants**.
- **The bank doesn't reject too many genuine applicants**.

Different evaluation metrics help **balance these trade-offs**.

2.2 Confusion Matrix: Understanding True Positives and Negatives

A **confusion matrix** helps visualize a model's predictions. It contains:

	Predicted Positive	**Predicted Negative**
Actual Positive	**True Positive (TP)**	**False Negative (FN)**

Actual Negative	**False Positive (FP)**	**True Negative (TN)**

🔹 **Definitions:**

✓ **True Positive (TP):** Model correctly predicts positive cases (e.g., detects fraud when it actually happened).

✓ **True Negative (TN):** Model correctly predicts negative cases (e.g., correctly classifies non-fraudulent transactions).

✓ **False Positive (FP):** Model wrongly predicts positive (e.g., falsely flags a legitimate transaction as fraud).

✓ **False Negative (FN):** Model wrongly predicts negative (e.g., fails to detect an actual fraud case).

📌 **Example:**

A hospital uses ML to detect cancer. If a patient has cancer but the model predicts "No Cancer," this is a **False Negative** and could have **serious consequences**.

💡 **Conclusion:**

The confusion matrix provides the foundation for calculating accuracy, precision, recall, and F1-score.

📌 CHAPTER 3: ACCURACY – OVERALL CORRECTNESS

3.1 What is Accuracy?

Accuracy measures the **percentage of correct predictions** made by the model out of all predictions.

📌 **Formula:**

Accuracy=TP+TNTP+TN+FP+FN\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}

📌 Example Calculation:

A model predicts **100 loan applications**:

- **True Positives (TP) = 40** (correctly predicted approvals)
- **True Negatives (TN) = 30** (correctly predicted rejections)
- **False Positives (FP) = 10** (wrongly approved risky customers)
- **False Negatives (FN) = 20** (missed actual risky customers)

$$\text{Accuracy} = \frac{40 + 30}{40 + 30 + 10 + 20} = \frac{70}{100} = 70\%$$

🔍 When to Use Accuracy:

✓ Works well when **positive and negative classes are balanced** (equal number of labels).

✓ Simple and intuitive to understand.

🔍 Limitations of Accuracy:

- **Misleading in imbalanced datasets** (e.g., fraud detection where only 1% of transactions are fraud).
- **Does not differentiate between false positives and false negatives**.

📌 Example:

In a **fraud detection system**, if 99% of transactions are legitimate and only 1% are fraudulent, a model that predicts "Not Fraud" for every transaction will have **99% accuracy but be completely useless**.

💡 Conclusion:

Accuracy is useful but should be complemented with **Precision, Recall, and F1-Score** for imbalanced datasets.

📌 CHAPTER 4: PRECISION, RECALL & F1-SCORE

4.1 Precision – How Many Positives Were Correct?

Precision measures **how many of the positive predictions were actually correct**.

📌 Formula:

Precision=TPTP+FP\text{Precision} = \frac{TP}{TP + FP}

📌 Example Calculation:

If a fraud detection model predicts **50 transactions as fraud** but **10 were false alarms,** precision would be:

Precision=4040+10=4050=80%\text{Precision} = \frac{40}{40 + 10} = \frac{40}{50} = 80\%

🔍 When to Use Precision:

✓ Important when **False Positives are costly** (e.g., legal cases, medical tests).
✓ Used in **spam filters, fraud detection, and medical diagnostics**.

💡 Conclusion:

High precision means fewer **false alarms,** but it might **miss some actual fraud cases**.

4.2 Recall (Sensitivity) – How Many Actual Positives Were Detected?

Recall measures **how well the model identifies all actual positive cases**.

📌 **Formula:**

$$Recall = \frac{TP}{TP + FN}$$

📌 **Example Calculation:**
If there were **50 actual fraud cases**, but the model only detected **40**, recall would be:

$$Recall = \frac{40}{40 + 10} = \frac{40}{50} = 80\%$$

🔍 **When to Use Recall:**
✓ Important when **False Negatives are costly** (e.g., cancer detection, fraud detection).
✓ High recall means the model **detects most actual fraud cases**, but may have **more false alarms**.

💡 **Conclusion:**
High recall means catching **more fraud cases**, but it might also **flag legitimate transactions incorrectly**.

4.3 F1-Score – The Balance Between Precision and Recall

F1-score is the **harmonic mean** of Precision and Recall, balancing both metrics.

📌 **Formula:**

$$F1\text{-}Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

📌 **Example Calculation:**
If **Precision = 80%** and **Recall = 80%**,

$F1 = 2 \times \frac{0.8 \times 0.8}{0.8 + 0.8} = 0.8$

🔍 When to Use F1-Score:

✓ Best for **imbalanced datasets** where both Precision and Recall matter.

✓ Used in **fraud detection, medical diagnosis, and spam filtering**.

💡 Conclusion:

A **high F1-Score** indicates a well-balanced model that catches **positives correctly** while minimizing false alarms.

📌 CHAPTER 5: EXERCISES & ASSIGNMENTS

✅ Which metric should be prioritized for medical tests?

☐ (a) Accuracy

☐ (b) Recall ✅

☐ (c) Precision

☐ (d) F1-Score

📌 Practical Assignment:

1. ☐ Train a classification model using Python (e.g., Logistic Regression on fraud detection).
2. ☐ Compute **Accuracy, Precision, Recall, and F1-Score** using Scikit-learn.
3. ☐ Interpret which metric is **most important** for the dataset.

📖 HYPERPARAMETER TUNING & MODEL OPTIMIZATION

📌 CHAPTER 1: UNDERSTANDING HYPERPARAMETERS IN MACHINE LEARNING

1.1 What Are Hyperparameters?

Hyperparameters are **external settings** that are defined before a machine learning model starts training. Unlike model parameters (such as weights in neural networks), hyperparameters **are not learned from data** but are instead manually set to control how a model learns.

These settings influence model training, impacting **accuracy, speed, and performance**. Choosing the right hyperparameters can significantly **improve model performance and prevent overfitting or underfitting**.

🔹 **Key Differences Between Parameters & Hyperparameters:**

Feature	Parameters	Hyperparameters
Definition	Learned from training data	Set before training starts
Examples	Weights in Neural Networks	Learning Rate, Number of Trees, Batch Size
Optimization	Adjusted by the model during training	Manually tuned using cross-validation

📌 **Example:**

- In a **decision tree model**, the **depth of the tree** is a hyperparameter.
- In a **neural network**, the **learning rate and number of hidden layers** are hyperparameters.

💡 **Conclusion:**
Hyperparameter tuning is crucial for **optimizing a machine learning model**, ensuring it generalizes well to new data.

📌 CHAPTER 2: COMMON HYPERPARAMETERS IN MACHINE LEARNING

Different machine learning models have **different hyperparameters**. Below are the most common hyperparameters across various algorithms:

2.1 Hyperparameters in Decision Trees & Random Forests

✓ **Max Depth (max_depth)** – Limits the depth of the tree to prevent overfitting.
✓ **Min Samples Split (min_samples_split)** – Minimum number of samples required to split a node.
✓ **Min Samples Leaf (min_samples_leaf)** – Minimum number of samples required in a leaf node.
✓ **Number of Trees (n_estimators)** – Only for Random Forests; defines how many trees are combined.

📌 **Example:**

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100, max_depth=10, min_samples_split=5)
```

2.2 Hyperparameters in Neural Networks

✓ **Learning Rate (learning_rate)** – Controls how much model weights change during training.
✓ **Batch Size (batch_size)** – Defines how many samples are processed before updating weights.
✓ **Number of Layers (n_layers)** – Determines the depth of the neural network.
✓ **Dropout Rate (dropout)** – Prevents overfitting by randomly deactivating some neurons.

📌 **Example:**

```
from keras.models import Sequential
from keras.layers import Dense, Dropout


model = Sequential([
  Dense(128, activation='relu', input_shape=(10,)),
  Dropout(0.2),
  Dense(64, activation='relu'),
  Dense(1, activation='sigmoid')
])
```

2.3 Hyperparameters in Gradient Boosting (XGBoost, LightGBM, CatBoost)

✓ **Learning Rate (eta)** – Controls step size when updating weights.
✓ **Number of Trees (n_estimators)** – Defines how many trees to use in boosting.
✓ **Max Depth (max_depth)** – Limits how deep trees can grow.
✓ **Subsample (subsample)** – Percentage of data used for each boosting round.

📌 **Example:**

```
from xgboost import XGBClassifier

model = XGBClassifier(n_estimators=100, learning_rate=0.1, max_depth=5, subsample=0.8)
```

💡 **Conclusion:**
Choosing the right hyperparameters for each model is **critical to optimizing performance** while avoiding overfitting.

📌 CHAPTER 3: HYPERPARAMETER TUNING METHODS

Hyperparameter tuning helps find **the best combination of hyperparameters** for a given model. Here are the most commonly used techniques:

3.1 Manual Search (Trial & Error Approach)

✓ Users manually test different hyperparameter values and compare model performance.
✓ Suitable for small models but **inefficient** for complex models.

📌 **Example:**

```
model = RandomForestClassifier(n_estimators=50, max_depth=5)
```

🔍 **Pros:**

✓ Quick for small models.

✓ Easy to implement.

🔍 **Cons:**

❌ Time-consuming for large datasets.

❌ Requires expert knowledge.

3.2 Grid Search Cross-Validation

✓ Tries **all possible combinations** of hyperparameters and selects the best one based on model performance.

📌 **Example:**

```
from sklearn.model_selection import GridSearchCV

from sklearn.ensemble import RandomForestClassifier


# Define hyperparameters

param_grid = {

  'n_estimators': [50, 100, 200],

  'max_depth': [5, 10, 15]

}


# Perform Grid Search

grid_search = GridSearchCV(RandomForestClassifier(), param_grid, cv=5)
```

```
grid_search.fit(X_train, y_train)


# Best Parameters

print(grid_search.best_params_)
```

🔍 **Pros:**
✓ Finds the **best combination of hyperparameters**.
✓ Uses **cross-validation** for accuracy.

🔍 **Cons:**
❌ Computationally expensive for **large parameter spaces**.

3.3 Random Search

✓ **Randomly selects combinations of hyperparameters** rather than trying all possible values.
✓ More efficient than Grid Search for **large hyperparameter spaces**.

📌 **Example:**

```
from sklearn.model_selection import RandomizedSearchCV

import numpy as np


# Define hyperparameter distribution

param_dist = {

  'n_estimators': np.arange(50, 500, 50),

  'max_depth': np.arange(3, 20, 2)
```

}

Perform Random Search

random_search = RandomizedSearchCV(RandomForestClassifier(), param_dist, n_iter=10, cv=5)

random_search.fit(X_train, y_train)

Best Parameters

print(random_search.best_params_)

🔍 **Pros:**
✓ Faster than Grid Search.
✓ Works well when **computing power is limited**.

🔍 **Cons:**
❌ May **miss the optimal combination** due to random selection.

3.4 Bayesian Optimization (Advanced Tuning)

✓ Uses **probabilistic models** to find the best hyperparameters efficiently.
✓ Suitable for **deep learning and complex models**.

📌 **Example (Using Optuna):**

import optuna

from sklearn.ensemble import RandomForestClassifier

```
# Define objective function
def objective(trial):
    n_estimators = trial.suggest_int('n_estimators', 50, 500)
    max_depth = trial.suggest_int('max_depth', 3, 20)

    model = RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth)
    model.fit(X_train, y_train)
    return model.score(X_test, y_test)

# Run Optimization
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=20)

print(study.best_params_)
```

🔍 **Pros:**
✓ Finds optimal hyperparameters efficiently.
✓ Suitable for **complex and deep learning models**.

🔍 **Cons:**
❌ More difficult to implement.

💡 **Conclusion:**
Each tuning method has its **advantages and trade-offs,** and the

best approach depends on **dataset size, computational power, and time constraints**.

📌 CHAPTER 4: SUMMARY & NEXT STEPS

✅ Key Takeaways:

✓ Hyperparameter tuning improves **model accuracy and generalization**.

✓ Common hyperparameters include **learning rate, max depth, batch size, and dropout**.

✓ Tuning methods include **Grid Search, Random Search, and Bayesian Optimization**.

📌 Next Steps:

- Practice **hyperparameter tuning on different datasets**.
- Use **Grid Search for small models** and **Bayesian Optimization for deep learning**.
- Implement **automated hyperparameter tuning using libraries like Optuna and Hyperopt**.

📌 ASSIGNMENT:

☑ BUILD A CUSTOMER CHURN PREDICTION MODEL USING LOGISTIC REGRESSION

📌 Solution: Building a Customer Churn Prediction Model using Logistic Regression

🎯 **Objective:**

The goal of this assignment is to **build a Customer Churn Prediction Model using Logistic Regression**. The model will help businesses predict whether a customer is likely to leave (churn) or stay based on historical customer data.

🔹 STEP 1: Understanding the Problem Statement

Customer churn occurs when **a customer stops using a company's service**. Businesses in telecom, banking, and e-commerce use churn prediction models to **retain customers before they leave**.

1.1 Why Logistic Regression?

Logistic Regression is a **classification algorithm** that predicts a binary outcome (Yes/No, Churn/No Churn). Since churn prediction is a **binary classification problem**, Logistic Regression is an excellent choice.

✔ **Simple & Interpretable** – Easy to understand and explain.
✔ **Works Well with Small Datasets** – Suitable for structured business data.
✔ **Provides Probability Scores** – Helps businesses take preventive actions based on risk levels.

🔹 STEP 2: Importing Required Libraries

Before building the model, install and import necessary libraries.

```
# Install required libraries

pip install pandas numpy matplotlib seaborn scikit-learn


# Import Libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

🔍 Key Insights:

- pandas – Handles data manipulation.
- numpy – Supports numerical operations.
- matplotlib & seaborn – Used for data visualization.
- sklearn.model_selection – Splits data into training and testing sets.

- sklearn.linear_model.LogisticRegression – Implements Logistic Regression.
- sklearn.metrics – Evaluates model performance.

🔹 STEP 3: Load and Explore the Dataset

For this assignment, we use a **Customer Churn Dataset** with the following columns:

Column Name	Description
customer_id	Unique ID of the customer
tenure	Number of months the customer has stayed
monthly_charges	Monthly fee paid by the customer
total_charges	Total amount paid by the customer
contract	Type of contract (Month-to-Month, One Year, Two Year)
payment_method	Mode of payment (Credit Card, Bank Transfer, etc.)
churn	Target variable (Yes = Churn, No = Not Churn)

📌 Load the dataset:

```
# Load the dataset

df = pd.read_csv("customer_churn.csv")
```

```
# Display first 5 rows
df.head()
```

🔍 **Key Insights:**

- Check for missing values and incorrect data types.
- Identify **categorical and numerical features**.

🔹 STEP 4: Data Cleaning & Preprocessing

4.1 Handling Missing Values

Check if there are missing values in the dataset:

```
# Check for missing values
print(df.isnull().sum())
```

📌 **Fill missing numerical values with median and categorical values with mode:**

```
df['total_charges'] = df['total_charges'].replace(" ", np.nan).astype(float)  # Convert to numeric
df['total_charges'].fillna(df['total_charges'].median(), inplace=True)
```

4.2 Encoding Categorical Variables

Machine learning models **cannot process categorical data** directly. We convert categorical columns into numeric format.

```
# Convert 'Yes'/'No' in 'churn' column to 1/0
```

```
df['churn'] = df['churn'].map({'Yes': 1, 'No': 0})


# Convert categorical columns using one-hot encoding
df = pd.get_dummies(df, columns=['contract', 'payment_method'], drop_first=True)
```

🔍 **Key Insights:**

- **One-hot encoding** creates binary columns for categorical features.
- **'drop_first=True'** avoids dummy variable trap.

4.3 Splitting Data into Training & Testing Sets

We split the data into **80% training and 20% testing**.

```
# Define features and target variable
X = df.drop(columns=['customer_id', 'churn'])
y = df['churn']


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


print(f"Training set size: {X_train.shape}, Testing set size: {X_test.shape}")
```

🔍 Key Insights:

- **Training data** is used to train the model.
- **Testing data** is used to evaluate model performance.

🔹 STEP 5: Model Training & Evaluation

5.1 Feature Scaling

Logistic Regression performs better when **numerical features are scaled**.

```
# Scale numerical features

scaler = StandardScaler()

X_train[['tenure', 'monthly_charges', 'total_charges']] = scaler.fit_transform(X_train[['tenure', 'monthly_charges', 'total_charges']])

X_test[['tenure', 'monthly_charges', 'total_charges']] = scaler.transform(X_test[['tenure', 'monthly_charges', 'total_charges']])
```

5.2 Training the Logistic Regression Model

Now, we fit the **Logistic Regression model** on the training data.

```
# Train Logistic Regression Model

model = LogisticRegression()

model.fit(X_train, y_train)
```

```
# Predict on test set
y_pred = model.predict(X_test)
```

🔍 **Key Insights:**

- The model **learns from training data** and applies this knowledge to **make predictions on new customers**.
- The **predict() function** outputs either **0 (No Churn) or 1 (Churn)**.

5.3 Evaluating Model Performance

We measure the performance of our model using:

✔ **Accuracy** – How often the model predicts correctly.

✔ **Confusion Matrix** – Shows true positives, true negatives, false positives, and false negatives.

✔ **Precision & Recall** – Important for imbalanced datasets.

```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")


# Display confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)


# Generate classification report
```

```
print("Classification Report:\n", classification_report(y_test, y_pred))
```

🔹 STEP 6: Understanding the Model Insights

6.1 Interpreting the Confusion Matrix

	Predicted Churn	Predicted No Churn
Actual Churn	True Positives (TP)	False Negatives (FN)
Actual No Churn	False Positives (FP)	True Negatives (TN)

6.2 Business Interpretation

✔ A **high precision score** means the model correctly identifies churners.

✔ A **high recall score** ensures fewer actual churners are missed.

✔ If accuracy is **low**, consider **tuning hyperparameters or adding more features**.

📌 SUMMARY OF STEPS

✅ **Loaded and cleaned dataset** (handled missing values, converted categorical features).

✅ **Split data into training (80%) and testing (20%) sets.**

✅ **Applied feature scaling to numerical columns.**

✅ **Trained a Logistic Regression model** to predict customer churn.

✅ **Evaluated model performance using accuracy, confusion matrix, and classification report.**

📌 Next Steps & Improvements

📌 Improve Model Performance:

- Tune hyperparameters like **regularization strength (C)**.
- Try different ML models (Random Forest, XGBoost) for better accuracy.
- Handle class imbalance using **oversampling (SMOTE) or undersampling**.

📌 Deploy Model:

- Integrate with a **real-time customer support system** to prevent churn.
- Use dashboards (Power BI, Tableau) for visual analysis.