ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION)

📖 Regression Models: Linear Regression & Polynomial Regression

📌 Chapter 1: Introduction to Regression Models

1.1 What is Regression?

Regression is a statistical method used in machine learning and data science to model relationships between variables. It helps predict a dependent variable (output) based on one or more independent variables (inputs).

1.2 Why is Regression Important?

Regression analysis is widely used in various industries: ✓ **Finance** – Predicting stock prices and financial trends.
✓ **Healthcare** – Diagnosing diseases based on symptoms and test results.
✓ **Marketing** – Understanding consumer behavior and sales forecasting.
✓ **Engineering** – Estimating product performance based on input factors.

1.3 Types of Regression Models

Regression models can be categorized as: 🔹 **Linear Regression** – Models a straight-line relationship.

🔹 **Polynomial Regression** – Captures non-linear relationships.

🔹 **Logistic Regression** – Used for classification problems.

📌 Chapter 2: Linear Regression

2.1 Understanding Linear Regression

Linear regression is the simplest form of regression, where the relationship between input (X) and output (Y) is modeled using a straight line.

Mathematical Formula:

$Y = mX + bY = mX + b$

Where:

- YY = Dependent variable (Predicted Output)
- XX = Independent variable (Input)
- MM = Slope of the line
- BB = Y-intercept (where the line crosses the Y-axis)

2.2 Example of Linear Regression

A company wants to predict sales based on advertising spending:

Advertising Spend (X)	Sales (Y)
10,000	20,000

20,000	40,000
30,000	60,000

A LINEAR REGRESSION MODEL WILL IDENTIFY A RELATIONSHIP BETWEEN ADVERTISING SPEND AND SALES, PREDICTING FUTURE SALES BASED ON NEW SPENDING VALUES.

2.3 ASSUMPTIONS OF LINEAR REGRESSION

✓ THERE IS A LINEAR RELATIONSHIP BETWEEN X AND Y.
✓ DATA HAS MINIMAL NOISE (OUTLIERS ARE RARE).
✓ VARIABLES ARE INDEPENDENT OF EACH OTHER.

2.4 IMPLEMENTING LINEAR REGRESSION IN PYTHON

```
IMPORT NUMPY AS NP

IMPORT PANDAS AS PD

IMPORT MATPLOTLIB.PYPLOT AS PLT

FROM SKLEARN.LINEAR_MODEL IMPORT LINEARREGRESSION


# SAMPLE DATA

X = NP.ARRAY([10, 20, 30, 40, 50]).RESHAPE(-1,1)

Y = NP.ARRAY([20, 40, 60, 80, 100])


# CREATE AND TRAIN MODEL

MODEL = LINEARREGRESSION()

MODEL.FIT(X, Y)
```

```
# Predict and plot
predictions = model.predict(X)
plt.scatter(X, Y, color='blue')
plt.plot(X, predictions, color='red')
plt.title("Linear Regression Example")
plt.xlabel("Advertising Spend")
plt.ylabel("Sales")
plt.show()
```

📌 Chapter 3: Polynomial Regression

3.1 What is Polynomial Regression?

Polynomial Regression extends Linear Regression by modeling a curved relationship between variables. Instead of a straight line, it fits a **parabolic or higher-order curve** to the data.

Mathematical Formula:

$Y=aX2+bX+cY = aX^2 + bX + c$

Where:

- YY = Dependent variable (Predicted Output)
- XX = Independent variable (Input)
- A,B,CA, B, C = Coefficients

3.2 When to Use Polynomial Regression?

✓ When data has a **non-linear** relationship.

✓ When a straight line cannot accurately represent the data trend.

✓ When higher-degree features improve model accuracy.

3.3 Example of Polynomial Regression

A company's website traffic is increasing in a non-linear pattern:

Ad Spend (X)	Website Visits (Y)
10,000	5,000
20,000	15,000
30,000	35,000
40,000	70,000

A polynomial regression model can **curve-fit** this data better than linear regression.

3.4 Implementing Polynomial Regression in Python

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline


# Sample data
X = np.array([10, 20, 30, 40, 50]).reshape(-1,1)
Y = np.array([5, 15, 35, 70, 120])  # Non-linear growth
```

```
# Create polynomial regression model
poly_model = make_pipeline(PolynomialFeatures(degree=2), LinearRegression())
poly_model.fit(X, Y)

# Predict and plot
predictions = poly_model.predict(X)
plt.scatter(X, Y, color='blue')
plt.plot(X, predictions, color='red')
plt.title("Polynomial Regression Example")
plt.xlabel("Advertising Spend")
plt.ylabel("Website Visits")
plt.show()
```

📌 Chapter 4: Comparison of Linear & Polynomial Regression

Feature	Linear Regression	Polynomial Regression
Relationship	Straight-line (linear)	Curved (non-linear)
Complexity	Simple	More complex

Accuracy	Limited for non-linear data	Fits complex patterns
Formula	Y=mX+bY = mX + b	Y=aX2+bX+cY = aX^2 + bX + c

4.1 When to Choose Which Model?

✓ **Linear Regression** – When data shows a simple increasing or decreasing trend.

✓ **Polynomial Regression** – When data has **curvature** or non-linearity.

📌 Chapter 5: Exercises & Assignments

5.1 Multiple Choice Questions

1. **What does linear regression predict?**

- ☐ (a) Categories
- ☐ (b) Numerical values ✅
- ☐ (c) Text
- ☐ (d) Images

2. **Which formula represents a polynomial regression?**

- ☐ (a) Y=mX+bY = mX + b
- ☐ (b) Y=aX2+bX+cY = aX^2 + bX + c ✅
- ☐ (c) Y=X×ZY = X \times Z
- ☐ (d) Y=X/ZY = X/Z

3. **When should you use polynomial regression?**

- ☐ (a) When data has a linear pattern
- ☐ (b) When data has a curved trend ✅

☐ (C) WHEN THERE IS NO PATTERN

☐ (D) ALWAYS

5.2 PRACTICAL ASSIGNMENTS

📌 **TASK 1:** IMPLEMENT LINEAR AND POLYNOMIAL REGRESSION ON A DATASET (E.G., HOUSING PRICES, SALES DATA).

📌 **TASK 2:** COMPARE THE ACCURACY OF BOTH MODELS AND EXPLAIN WHICH ONE WORKS BEST FOR YOUR DATA.

📌 CHAPTER 6: SUMMARY

✅ **LINEAR REGRESSION** MODELS A **STRAIGHT-LINE** RELATIONSHIP.

✅ **POLYNOMIAL REGRESSION** CAPTURES **CURVED** RELATIONSHIPS.

✅ **POLYNOMIAL MODELS** IMPROVE ACCURACY WHEN DATA IS NON-LINEAR.

✅ **BOTH MODELS** HAVE DIFFERENT APPLICATIONS DEPENDING ON DATA TRENDS.

🌟 CONCLUSION: THE FUTURE OF REGRESSION MODELS

AS MACHINE LEARNING ADVANCES, REGRESSION MODELS ARE BEING ENHANCED USING AI AND DEEP LEARNING. HYBRID MODELS COMBINING LINEAR AND NON-LINEAR TECHNIQUES ARE IMPROVING PREDICTIVE ACCURACY IN **FINANCE, HEALTHCARE, AND ENGINEERING**.

☑ Classification Models: Logistic Regression, Decision Trees, Support Vector

📌 Introduction to Classification Models

Classification is a supervised learning technique used in **machine learning (ML)** to categorize data into predefined classes. It is widely applied in **spam detection, fraud detection, medical diagnosis, sentiment analysis, and more.**

🔹 Types of Classification Models

There are several classification models, but in this guide, we will focus on:

1. **Logistic Regression**
2. **Decision Trees**
3. **Support Vector Machines (SVMs)**

📌 1. Logistic Regression

🔹 What is Logistic Regression?

- Logistic Regression is a **statistical method** used for **binary classification (Yes/No, 0/1, True/False)**.
- It is an extension of linear regression but **uses a sigmoid function** to predict probabilities.

The Sigmoid Function

The sigmoid function maps input values to a probability range between **0 and 1**:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

where z is the weighted sum of input features.

When to Use Logistic Regression?

- When the target variable is **binary** (e.g., spam/not spam, fraud/not fraud).
- When you need **interpretable** probability outputs.
- Works well when features are **linearly separable**.

Advantages

✓ Simple and easy to interpret.
✓ Works well with small datasets.
✓ Outputs **probabilities,** which help in decision-making.

Disadvantages

✗ Assumes **linear relationship** between independent variables and the log odds.
✗ Not effective when dealing with **non-linear** relationships.

Example: Predicting Customer Churn

```
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")


# SELECT FEATURES AND TARGET VARIABLE

X = DF[['MONTHLY_CHARGES', 'TENURE']]

Y = DF['CHURN']  # 0: NO, 1: YES


# SPLIT DATASET INTO TRAINING AND TESTING SETS

X_TRAIN, X_TEST, Y_TRAIN, Y_TEST = TRAIN_TEST_SPLIT(X, Y,
TEST_SIZE=0.2, RANDOM_STATE=42)


# TRAIN LOGISTIC REGRESSION MODEL

MODEL = LOGISTICREGRESSION()

MODEL.FIT(X_TRAIN, Y_TRAIN)


# PREDICTIONS

Y_PRED = MODEL.PREDICT(X_TEST)
```

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

📌 2. Decision Trees

🔹 What is a Decision Tree?

- A **tree-like model** where data is split into branches based on conditions.
- Uses a **hierarchical decision-making process**.

🔹 How It Works?

1. Starts from the **root node** (entire dataset).
2. Splits into **branches** using conditions (e.g., Is age > 30?).
3. Ends at **leaf nodes**, which contain the final class labels.

🔹 Key Concepts

- **Entropy**: Measures impurity in a dataset.
- **Gini Index**: Measures how well a node is split.
- **Pruning**: Removing unnecessary branches to **prevent overfitting**.

🔹 When to Use Decision Trees?

- When the dataset has **complex and non-linear relationships**.
- If interpretability is important (**easy to visualize**).

Advantages

✓ Works for both **classification** and **regression** problems.
✓ Handles **non-linear relationships** effectively.
✓ No need for feature scaling.

Disadvantages

✗ **Overfitting** if the tree is too deep.
✗ **Sensitive to noisy data,** leading to instability.

Example: Predicting Loan Approval

```
from sklearn.tree import DecisionTreeClassifier


# Train decision tree classifier
model = DecisionTreeClassifier(max_depth=3,
random_state=42)
model.fit(X_train, y_train)


# Predictions
y_pred = model.predict(X_test)


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

Visualizing the Decision Tree

```
from sklearn import tree
```

```
IMPORT MATPLOTLIB.PYPLOT AS PLT



PLT.FIGURE(FIGSIZE=(12,6))

TREE.PLOT_TREE(MODEL, FILLED=TRUE,
FEATURE_NAMES=['MONTHLY_CHARGES', 'TENURE'],
CLASS_NAMES=['NO', 'YES'])

PLT.SHOW()
```

📌 3. Support Vector Machines (SVMs)

🔹 What is SVM?

- **A powerful classification algorithm** that separates data using a hyperplane.
- **Maximizes the margin** between different classes.
- Can be **linear or non-linear**.

🔹 Key Concepts

- **Hyperplane**: A decision boundary that separates different classes.
- **Support Vectors**: Data points closest to the hyperplane that influence its position.
- **Kernel Trick**: Used when data is not linearly separable (e.g., **RBF Kernel**).

🔹 When to Use SVM?

- When the dataset is **high-dimensional**.

- Works well when data is **not linearly separable**.
- Effective for **small to medium-sized datasets**.

🔹 Advantages

✓ Works well with **high-dimensional data**.
✓ Can classify **non-linearly separable data** (with kernel trick).
✓ Robust to **outliers**.

🔹 Disadvantages

✗ Computationally **expensive for large datasets**.
✗ Choosing the **right kernel** requires tuning.

🔹 Example: Classifying Tumors (Malignant or Benign)

```
from sklearn.svm import SVC


# Train an SVM classifier
svm_model = SVC(kernel='rbf', C=1.0, gamma='scale')
svm_model.fit(X_train, y_train)

# Predictions
y_pred = svm_model.predict(X_test)


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

Visualizing the SVM Decision Boundary

```
IMPORT NUMPY AS NP

IMPORT MATPLOTLIB.PYPLOT AS PLT


# CREATE MESHGRID

X_MIN, X_MAX = X_TRAIN['MONTHLY_CHARGES'].MIN() - 1,
X_TRAIN['MONTHLY_CHARGES'].MAX() + 1

Y_MIN, Y_MAX = X_TRAIN['TENURE'].MIN() - 1,
X_TRAIN['TENURE'].MAX() + 1

XX, YY = NP.MESHGRID(NP.LINSPACE(X_MIN, X_MAX, 100),
NP.LINSPACE(Y_MIN, Y_MAX, 100))


# PREDICT ON THE GRID

Z = SVM_MODEL.PREDICT(NP.C_[XX.RAVEL(), YY.RAVEL()])

Z = Z.RESHAPE(XX.SHAPE)


# PLOT DECISION BOUNDARY

PLT.CONTOURF(XX, YY, Z, ALPHA=0.3)

PLT.SCATTER(X_TRAIN['MONTHLY_CHARGES'], X_TRAIN['TENURE'],
C=Y_TRAIN, EDGECOLORS='K')

PLT.TITLE('SVM DECISION BOUNDARY')

PLT.XLABEL('MONTHLY CHARGES')

PLT.YLABEL('TENURE')
```

```
PLT.SHOW()
```

📌 Summary

Algorithm	Best For	Advantages	Disadvantages
Logistic Regression	Binary classification, interpretable results	Simple, probability-based, efficient	Only works with **linear data**
Decision Trees	Complex datasets, easy visualization	Handles non-linearity, interpretable	Prone to **overfitting**
SVM	High-dimensional and non-linear data	Works well for **complex boundaries**, robust	Computationally expensive for large datasets

---

📌 Conclusion

- **Logistic Regression** is great for simple, linearly separable problems.
- **Decision Trees** are useful when interpretability is important.

- **SVMs** WORK WELL FOR COMPLEX DECISION BOUNDARIES BUT CAN BE SLOW.

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

📌 Chapter 1: Introduction to Model Evaluation Metrics

1.1 What is Model Evaluation?

Model evaluation is the process of assessing how well a machine learning model performs on a dataset. It helps determine whether a model is **accurate, reliable, and suitable** for real-world applications.

1.2 Why is Model Evaluation Important?

✓ Helps avoid **overfitting** (when a model memorizes training data but fails on new data).
✓ Ensures the model **generalizes well** to unseen data.
✓ Helps in selecting the **best model** among multiple models.

1.3 Common Model Evaluation Metrics

- **Accuracy** – Measures the proportion of correct predictions.
- **Precision** – Evaluates how many predicted positives were actually correct.
- **Recall** – Measures how well the model detects actual positives.
- **F1-Score** – A balance between precision and recall.

📌 CHAPTER 2: ACCURACY METRIC

2.1 WHAT IS ACCURACY?

ACCURACY IS THE MOST BASIC EVALUATION METRIC, MEASURING THE PERCENTAGE OF CORRECTLY CLASSIFIED INSTANCES.

FORMULA FOR ACCURACY:

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}} \times 100$$

2.2 EXAMPLE OF ACCURACY CALCULATION

CONSIDER A **SPAM EMAIL CLASSIFIER** PREDICTING WHETHER AN EMAIL IS SPAM OR NOT.

ACTUAL	PREDICTED
SPAM	SPAM
NOT SPAM	NOT SPAM
SPAM	NOT SPAM
NOT SPAM	NOT SPAM

- **CORRECT PREDICTIONS** = 3
- **TOTAL PREDICTIONS** = 4

$$\text{Accuracy} = \frac{3}{4} \times 100 = 75\%$$

2.3 WHEN IS ACCURACY USEFUL?

✓ WHEN THE DATASET IS **BALANCED** (EQUAL POSITIVES AND NEGATIVES).

✓ In simple **classification problems** where all errors have equal consequences.

2.4 When is Accuracy Misleading?

🚫 If the dataset is **imbalanced** (e.g., detecting fraud, where fraud cases are rare), accuracy may be **misleading**.

🚫 If predicting **only the majority class** (e.g., always predicting "Not Spam"), accuracy might be high but the model is useless.

📌 Chapter 3: Precision Metric

3.1 What is Precision?

Precision measures the proportion of **correct positive predictions** out of all positive predictions made by the model.

Formula for Precision:

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

3.2 Example of Precision Calculation

Consider a **medical test** for detecting a disease.

Actual	Predicted
Diseased	Diseased
Healthy	Diseased
Diseased	Diseased

HEALTHY	HEALTHY

- **TRUE POSITIVES (TP)** = 2 (CORRECTLY DETECTED DISEASED CASES)
- **FALSE POSITIVES (FP)** = 1 (HEALTHY PERSON MISCLASSIFIED AS DISEASED)

\TEXT{PRECISION} = \FRAC{2}{2+1} = \FRAC{2}{3} = 0.66 \TEXT{(66%)}

3.3 WHEN IS PRECISION IMPORTANT?

✓ WHEN **FALSE POSITIVES ARE COSTLY** (E.G., FRAUD DETECTION, CANCER DIAGNOSIS).
✓ IN APPLICATIONS WHERE **INCORRECT POSITIVE PREDICTIONS** SHOULD BE MINIMIZED (E.G., RECOMMENDING INAPPROPRIATE PRODUCTS TO USERS).

📌 CHAPTER 4: RECALL METRIC

4.1 WHAT IS RECALL?

RECALL (ALSO CALLED **SENSITIVITY** OR **TRUE POSITIVE RATE**) MEASURES THE PROPORTION OF ACTUAL POSITIVE CASES THAT WERE **CORRECTLY PREDICTED**.

FORMULA FOR RECALL:

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

4.2 EXAMPLE OF RECALL CALCULATION

USING THE **SAME MEDICAL TEST EXAMPLE**:

Actual	Predicted
Diseased	Diseased
Healthy	Diseased
Diseased	Diseased
Diseased	Healthy

- **True Positives (TP)** = 2 (correctly detected diseased cases)
- **False Negatives (FN)** = 1 (missed diseased case)

\text{Recall} = \frac{2}{2+1} = \frac{2}{3} = 0.66 \text{ (66%)}

4.3 When is Recall Important?

✓ When **False Negatives are costly** (e.g., detecting cancer, safety alarms).
✓ In scenarios where **missing a positive case** has serious consequences (e.g., detecting terrorist threats).

📌 Chapter 5: F1-Score Metric

5.1 What is F1-Score?

F1-Score is the **harmonic mean** of Precision and Recall. It **balances** the two when one is high and the other is low.

Formula for F1-Score:

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

5.2 Example of F1-Score Calculation

USING OUR PREVIOUS PRECISION **(66%)** AND RECALL **(66%)**:

F1 = 2 \TIMES \FRAC{0.66 \TIMES 0.66}{0.66 + 0.66} = 2 \TIMES \FRAC{0.4356}{1.32} = 0.66 \TEXT{ (66%)}

5.3 WHEN IS F1-SCORE IMPORTANT?

✓ WHEN **BOTH FALSE POSITIVES AND FALSE NEGATIVES MATTER** (E.G., SPAM DETECTION).

✓ IN **IMBALANCED DATASETS,** WHERE ACCURACY IS MISLEADING.

📌 CHAPTER 6: COMPARISON OF METRICS

METRIC	DEFINITION	WHEN TO USE?
ACCURACY	% OF CORRECT PREDICTIONS	BALANCED DATASETS, GENERAL CLASSIFICATION
PRECISION	% OF PREDICTED POSITIVES THAT ARE CORRECT	WHEN FALSE POSITIVES ARE COSTLY (E.G., FRAUD DETECTION)
RECALL	% OF ACTUAL POSITIVES CORRECTLY IDENTIFIED	WHEN FALSE NEGATIVES ARE COSTLY (E.G., MEDICAL TESTS)
F1-SCORE	BALANCE BETWEEN PRECISION AND RECALL	WHEN BOTH FP AND FN ARE IMPORTANT

📌 CHAPTER 7: IMPLEMENTING METRICS IN PYTHON

7.1 USING SCIKIT-LEARN

```
FROM SKLEARN.METRICS IMPORT ACCURACY_SCORE,
PRECISION_SCORE, RECALL_SCORE, F1_SCORE


# EXAMPLE PREDICTIONS

Y_TRUE = [1, 0, 1, 1, 0, 1, 0, 0, 1, 1] # ACTUAL LABELS

Y_PRED = [1, 0, 1, 0, 0, 1, 1, 0, 1, 1] # PREDICTED LABELS


# CALCULATE METRICS

ACCURACY = ACCURACY_SCORE(Y_TRUE, Y_PRED)

PRECISION = PRECISION_SCORE(Y_TRUE, Y_PRED)

RECALL = RECALL_SCORE(Y_TRUE, Y_PRED)

F1 = F1_SCORE(Y_TRUE, Y_PRED)


# PRINT RESULTS

PRINT(F"ACCURACY: {ACCURACY:.2F}")

PRINT(F"PRECISION: {PRECISION:.2F}")

PRINT(F"RECALL: {RECALL:.2F}")

PRINT(F"F1-SCORE: {F1:.2F}")
```

📌 CHAPTER 8: EXERCISES & ASSIGNMENTS

8.1 MULTIPLE CHOICE QUESTIONS

1. What does Precision measure?

- ☐ (A) Correctly classified instances
- ☐ (B) Correct positive predictions ✅
- ☐ (C) Misclassified instances

2. Which metric is best when missing a positive case is costly?

- ☐ (A) Accuracy
- ☐ (B) Recall ✅
- ☐ (C) Precision

3. What does F1-Score balance?

- ☐ (A) Accuracy and Precision
- ☐ (B) Precision and Recall ✅
- ☐ (C) False Positives and True Negatives

8.2 Practical Assignments

📌 **Task 1:** Implement model evaluation metrics on a dataset (e.g., spam detection).

📌 **Task 2:** Compare model performance using Accuracy, Precision, Recall, and F1-Score.

📌 CHAPTER 9: SUMMARY

✅ **Accuracy** is useful for balanced datasets.

✅ **Precision** is important when False Positives are costly.

✅ **Recall** is needed when False Negatives are costly.

✅ **F1-Score** balances Precision and Recall for imbalanced data.

📖 Hyperparameter Tuning & Model Optimization

📌 Chapter 1: Introduction to Hyperparameter Tuning & Model Optimization

1.1 What is Hyperparameter Tuning?

Hyperparameter tuning refers to the process of selecting the best hyperparameters for a machine learning model to improve its performance. Unlike model parameters (which are learned from data), hyperparameters are set before training begins.

- **Example:** In a Decision Tree, hyperparameters include:
  - Maximum depth of the tree
  - Minimum samples per leaf
  - Splitting criterion (Gini impurity vs. entropy)

1.2 Why is Hyperparameter Tuning Important?

Proper hyperparameter tuning can: ✓ Improve model accuracy and generalization

✓ Prevent overfitting or underfitting

✓ Optimize training time and computational resources

1.3 Difference Between Parameters & Hyperparameters

Feature	Model Parameters	Hyperparameters
Definition	Values learned by the model from data	Values set before training
Examples	Weights in Neural Networks, Coefficients in Linear Regression	Learning Rate, Number of Layers, Number of Neurons
Adjusted by	Training algorithm	Manually or via tuning techniques

---

📌 Chapter 2: Common Hyperparameters in Machine Learning

Different ML algorithms have different hyperparameters. Here are some common ones:

2.1 Hyperparameters in Decision Trees

✓ max_depth – Limits tree depth to avoid overfitting

✓ min_samples_split – Minimum samples required to split a node

✓ criterion – Measures split quality (Gini impurity vs. entropy)

2.2 Hyperparameters in Random Forest

✓ n_estimators – Number of decision trees

✓ max_features – Number of features used per tree

✓ bootstrap – Whether samples are drawn with replacement

2.3 Hyperparameters in Neural Networks

✓ learning_rate – Controls weight updates during training

✓ batch_size – Number of samples per training iteration

✓ epochs – Number of times the model sees the full dataset

📌 Chapter 3: Hyperparameter Tuning Techniques

3.1 Manual Tuning

- **What it is:** Selecting hyperparameters manually based on domain knowledge
- **Pros:** Simple for small models
- **Cons:** Time-consuming and inefficient

3.2 Grid Search

- **What it is:** Tries all possible hyperparameter combinations
- **Pros:** Ensures the best hyperparameters are found
- **Cons:** Computationally expensive

Example: Grid Search in Python

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# Define model
model = RandomForestClassifier()
```

```
# Define hyperparameter grid
param_grid = {'n_estimators': [50, 100, 200], 'max_depth': [5, 10, None]}

# Perform Grid Search
grid_search = GridSearchCV(model, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Get best parameters
print("Best parameters:", grid_search.best_params_)
```

3.3 Random Search

- **What it is:** Randomly selects hyperparameters instead of trying all combinations
- **Pros:** Faster than grid search
- **Cons:** May not find the absolute best parameters

Example: Random Search in Python

```
from sklearn.model_selection import RandomizedSearchCV
import numpy as np

# Define hyperparameter distributions
```

```
PARAM_DIST = {'N_ESTIMATORS': NP.ARANGE(50, 200, 50),
'MAX_DEPTH': [5, 10, NONE]}


# PERFORM RANDOM SEARCH

RANDOM_SEARCH = RANDOMIZEDSEARCHCV(MODEL, PARAM_DIST,
N_ITER=5, CV=5)

RANDOM_SEARCH.FIT(X_TRAIN, Y_TRAIN)


PRINT("BEST PARAMETERS:", RANDOM_SEARCH.BEST_PARAMS_)
```

3.4 Bayesian Optimization

- **What it is:** Uses probability models to find the best hyperparameters
- **Pros:** More efficient than Grid and Random Search
- **Cons:** Requires additional libraries like scikit-optimize

Example: Bayesian Optimization in Python

```
FROM SKOPT IMPORT BAYESSEARCHCV


# PERFORM BAYESIAN OPTIMIZATION

BAYES_SEARCH = BAYESSEARCHCV(MODEL, PARAM_GRID,
N_ITER=10, CV=5)

BAYES_SEARCH.FIT(X_TRAIN, Y_TRAIN)
```

print("Best parameters:", bayes_search.best_params_)

3.5 Automated Hyperparameter Tuning (AutoML)

- **What it is:** Uses AI to optimize hyperparameters automatically
- **Pros:** Hands-off approach for complex models
- **Cons:** Limited control over tuning process

Popular AutoML Tools ✓ Google AutoML
✓ TPOT (Tree-based Pipeline Optimization Tool)
✓ H2O AutoML

📌 Chapter 4: Model Optimization Techniques

Hyperparameter tuning alone isn't enough. Here's how to further improve model performance:

4.1 Feature Engineering

✓ Removing irrelevant features
✓ Creating new informative features
✓ Normalizing or scaling data

4.2 Cross-Validation

✓ Splits data into training & validation sets multiple times
✓ Ensures model generalizes well to new data

Example of **k-Fold Cross-Validation**:

```
FROM SKLEARN.MODEL_SELECTION IMPORT CROSS_VAL_SCORE


SCORES = CROSS_VAL_SCORE(MODEL, X_TRAIN, Y_TRAIN, CV=5)

PRINT("MEAN ACCURACY:", SCORES.MEAN())
```

4.3 EARLY STOPPING (FOR NEURAL NETWORKS)

✓ STOPS TRAINING WHEN MODEL PERFORMANCE STARTS DEGRADING

✓ PREVENTS OVERFITTING

EXAMPLE IN **TENSORFLOW**:

```
FROM TENSORFLOW.KERAS.CALLBACKS IMPORT EARLYSTOPPING


EARLY_STOPPING = EARLYSTOPPING(MONITOR='VAL_LOSS', PATIENCE=3)

MODEL.FIT(X_TRAIN, Y_TRAIN, VALIDATION_DATA=(X_VAL, Y_VAL), CALLBACKS=[EARLY_STOPPING])
```

📌 CHAPTER 5: EXERCISES & ASSIGNMENTS

5.1 MULTIPLE CHOICE QUESTIONS

1. WHAT IS HYPERPARAMETER TUNING USED FOR?

☐ (A) CHANGING MODEL STRUCTURE

☐ (B) FINDING THE BEST MODEL PARAMETERS ✅

☐ (C) Increasing training time

☐ (D) Removing missing values

2. **Which method exhaustively tries all hyperparameter combinations?**

☐ (A) Grid Search ✅

☐ (B) Random Search

☐ (C) Bayesian Optimization

☐ (D) Early Stopping

3. **Which optimization technique is used in neural networks to prevent overfitting?**

☐ (A) Feature Engineering

☐ (B) Cross-Validation

☐ (C) Early Stopping ✅

☐ (D) Random Search

5.2 Practical Assignments

📌 **Task 1:** Perform Grid Search and Random Search on a dataset and compare results.

📌 **Task 2:** Apply Bayesian Optimization to a Random Forest model and report the best hyperparameters.

📌 Chapter 6: Summary

✅ **Hyperparameter tuning** improves model performance by selecting the best configurations.

✅ **Grid Search** is exhaustive but slow, while **Random Search** is faster.

✅ **Bayesian Optimization** balances efficiency and accuracy.

✅ **Model optimization techniques** like cross-validation and early stopping prevent overfitting.

🌟 Conclusion: The Future of Hyperparameter Tuning

Automated tools like **Google AutoML** and **TPOT** are making hyperparameter tuning faster and more efficient. With advancements in AI-driven optimization, model performance will continue to improve across industries.

📖 Implementing Supervised ML Models Using Scikit-Learn

📌 Chapter 1: Introduction to Supervised Machine Learning

1.1 What is Supervised Learning?

Supervised learning is a type of machine learning where a model learns from **labeled data**. It maps input variables (X) to an output variable (Y).

1.2 Types of Supervised Learning

- **Classification** – Predicting categories (e.g., spam or not spam).
- **Regression** – Predicting continuous values (e.g., house prices).

1.3 Popular Supervised Learning Algorithms

✓ **Logistic Regression** – Used for binary classification.
✓ **Decision Trees** – Rule-based decision making.
✓ **Random Forest** – A collection of decision trees for better accuracy.
✓ **Support Vector Machines (SVMs)** – Works well for classification.
✓ **K-Nearest Neighbors (KNN)** – Finds the nearest neighbors for prediction.
✓ **Naïve Bayes** – Works well for text classification.

✓ **Neural Networks** – Advanced model for complex patterns.

📌 Chapter 2: Implementing Classification using Scikit-learn

2.1 Steps to Build a Classification Model

1. Load the dataset.
2. Split data into training and testing sets.
3. Train the model using a supervised learning algorithm.
4. Make predictions.
5. Evaluate the model using metrics like Accuracy and F1-Score.

2.2 Example: Implementing Logistic Regression

```
PYTHON

COPYEDIT

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 (EXAMPLE: BREAST CANCER DATASET)

FROM SKLEARN.DATASETS IMPORT LOAD_BREAST_CANCER

DATA = LOAD_BREAST_CANCER()
```

```
X = PD.DATAFRAME(DATA.DATA,
COLUMNS=DATA.FEATURE_NAMES)

Y = DATA.TARGET


# SPLIT DATASET

X_TRAIN, X_TEST, Y_TRAIN, Y_TEST = TRAIN_TEST_SPLIT(X, Y,
TEST_SIZE=0.2, RANDOM_STATE=42)


# TRAIN LOGISTIC REGRESSION MODEL

MODEL = LOGISTICREGRESSION(MAX_ITER=1000)

MODEL.FIT(X_TRAIN, Y_TRAIN)


# MAKE PREDICTIONS

Y_PRED = MODEL.PREDICT(X_TEST)


# EVALUATE ACCURACY

PRINT("ACCURACY:", ACCURACY_SCORE(Y_TEST, Y_PRED))
```

📌 CHAPTER 3: IMPLEMENTING REGRESSION USING SCIKIT-LEARN

3.1 EXAMPLE: IMPLEMENTING LINEAR REGRESSION

```
PYTHON
```

```
COPYEDIT
FROM SKLEARN.LINEAR_MODEL IMPORT LINEARREGRESSION
IMPORT NUMPY AS NP

# SAMPLE DATA
X = NP.ARRAY([1, 2, 3, 4, 5]).RESHAPE(-1, 1)
Y = NP.ARRAY([2, 4, 5, 4, 5])

# TRAIN MODEL
REGRESSOR = LINEARREGRESSION()
REGRESSOR.FIT(X, Y)

# PREDICT NEW VALUES
PREDICTIONS = REGRESSOR.PREDICT(X)
PRINT("PREDICTIONS:", PREDICTIONS)
```

📌 CHAPTER 4: EXERCISES & ASSIGNMENTS

📌 **TASK 1:** IMPLEMENT A DECISION TREE CLASSIFIER USING SCIKIT-LEARN.

📌 **TASK 2:** TRAIN A RANDOM FOREST MODEL AND COMPARE IT WITH DECISION TREES.

📌 CHAPTER 5: SUMMARY

- ✅ **SUPERVISED LEARNING** REQUIRES LABELED DATA.
- ✅ **CLASSIFICATION MODELS** PREDICT CATEGORIES.
- ✅ **REGRESSION MODELS** PREDICT CONTINUOUS VALUES.
- ✅ **SCIKIT-LEARN** MAKES MODEL TRAINING EASY.

🌟 CONCLUSION: THE POWER OF SCIKIT-LEARN

SCIKIT-LEARN IS A POWERFUL TOOL FOR **QUICKLY BUILDING ML MODELS**. WITH MORE DATA AND **HYPERPARAMETER TUNING,** WE CAN IMPROVE ACCURACY AND GENERALIZATION.

📌 ✨ ASSIGNMENT 1: 🎯 BUILD A HOUSE PRICE PREDICTION MODEL USING LINEAR REGRESSION.

📌✨ Assignment Solution 1: Build a House Price Prediction Model using Linear Regression

🎯 Objective

The goal of this assignment is to build a **House Price Prediction Model** using **Linear Regression**. You will learn to **load data, clean it, perform exploratory data analysis (EDA), build a regression model, and evaluate its performance**.

🛠️ Step 1: Import Necessary Libraries

We will use **Python** and its data science libraries to build the model.

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

📂 Step 2: Load the Dataset

We will use a publicly available **House Price Dataset** from **Kaggle** or generate sample data.

```
# Load the dataset (Replace 'house_prices.csv' with your actual dataset)
df = pd.read_csv("house_prices.csv")


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

🔍 Step 3: Exploratory Data Analysis (EDA)

3.1 Check Dataset Information

```
# Get dataset info
df.info()


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

✓ Action Plan:

- If missing values exist, fill them using the median or drop them.

3.2 Visualizing House Prices Distribution

```
# Histogram of House Prices
plt.figure(figsize=(8,5))
```

```
SNS.HISTPLOT(DF['PRICE'], BINS=30, KDE=TRUE, COLOR='BLUE')
PLT.TITLE("HOUSE PRICE DISTRIBUTION")
PLT.XLABEL("PRICE")
PLT.YLABEL("COUNT")
PLT.SHOW()
```

3.3 IDENTIFY CORRELATIONS

```
# CORRELATION HEATMAP
PLT.FIGURE(FIGSIZE=(10,6))
SNS.HEATMAP(DF.CORR(), ANNOT=TRUE, CMAP="COOLWARM")
PLT.TITLE("FEATURE CORRELATION MATRIX")
PLT.SHOW()
```

✓ INSIGHT:

- IDENTIFY THE MOST IMPORTANT FEATURES AFFECTING **HOUSE PRICES**.

📊 STEP 4: SELECT FEATURES & TARGET VARIABLE

4.1 CHOOSING THE RIGHT FEATURES

WE SELECT **IMPORTANT INDEPENDENT VARIABLES (X)** AFFECTING HOUSE PRICES.

```
# DEFINE INDEPENDENT VARIABLES (X) AND DEPENDENT VARIABLE (Y)
```

```
X = DF[['SQFT_LIVING', 'BEDROOMS', 'BATHROOMS', 'FLOORS',
'SQFT_LOT']]

Y = DF['PRICE']
```

📉 Step 5: Split the Data into Training & Testing Sets

We divide the dataset into **training (80%)** and **testing (20%)** sets.

```
# Split into training and testing sets (80-20 ratio)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.2, random_state=42)
```

📌 Step 6: Build and Train the Linear Regression Model

We train a **Linear Regression Model** using **Scikit-Learn**.

```
# Create Linear Regression model

model = LinearRegression()


# Train the model

model.fit(X_train, Y_train)
```

📊 Step 7: Make Predictions

Now, we use the trained model to predict **house prices**.

```
# Make predictions on the test set
```

```
Y_PRED = MODEL.PREDICT(X_TEST)
```

📏 Step 8: Evaluate Model Performance

We evaluate the model using **Mean Absolute Error (MAE), Mean Squared Error (MSE), and R^2 Score**.

```
# Calculate performance metrics
MAE = mean_absolute_error(Y_test, Y_pred)
MSE = mean_squared_error(Y_test, Y_pred)
RMSE = np.sqrt(MSE)
R2 = r2_score(Y_test, Y_pred)

# Display results
print(f"Mean Absolute Error (MAE): {MAE}")
print(f"Mean Squared Error (MSE): {MSE}")
print(f"Root Mean Squared Error (RMSE): {RMSE}")
print(f"R² Score: {R2}")
```

✓ Interpretation:

- **MAE & RMSE** should be **low** (less error means better predictions).
- **R^2 Score** should be **close to 1** (higher means better accuracy).

📊 Step 9: Visualizing the Model Predictions

We compare **actual vs predicted house prices**.

```
plt.figure(figsize=(8,5))
plt.scatter(Y_test, Y_pred, color='blue', alpha=0.5)
plt.plot([min(Y_test), max(Y_test)], [min(Y_test), max(Y_test)], color='red', linestyle='dashed')
plt.title("Actual vs Predicted House Prices")
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.show()
```

✓ Insight:

- If points **align with the red dashed line**, the predictions are accurate.

📌 Step 10: Conclusion

✅ Key Takeaways

- We **loaded and explored** the dataset.
- We **identified correlations** and selected the best features.
- We **trained a Linear Regression Model** to predict house prices.
- We **evaluated performance** using MAE, MSE, RMSE, and R^2 Score.

- **Model predictions were visualized** to assess accuracy.

🎯 Next Steps

- Try adding more features (e.g., location, year_built).
- Experiment with **Polynomial Regression** for better predictions.
- Optimize the model using **feature engineering** and **hyperparameter tuning**.

📌 ✨ ASSIGNMENT 2:

🎯 IMPLEMENT A SPAM EMAIL CLASSIFIER USING LOGISTIC REGRESSION.

📌 ✨ Assignment Solution 2: Implement a Spam Email Classifier using Logistic Regression

🎯 Objective

The goal of this assignment is to build a **Spam Email Classifier** using **Logistic Regression**. The classifier will analyze an email's text and predict whether it is **Spam (1)** or **Not Spam (0)**.

🛠️ Step 1: Install and Import Necessary Libraries

We will use the following Python libraries:

- **pandas**: For handling datasets.
- **numpy**: For numerical operations.
- **sklearn**: For machine learning and text processing.

🔹 Install Dependencies (if not installed)

```
!pip install pandas numpy scikit-learn
```

🔹 Import Required Libraries

```
import pandas as pd

import numpy as np

import re

import string
```

```
FROM SKLEARN.MODEL_SELECTION IMPORT TRAIN_TEST_SPLIT

FROM SKLEARN.FEATURE_EXTRACTION.TEXT IMPORT
TFIDFVECTORIZER

FROM SKLEARN.LINEAR_MODEL IMPORT LOGISTICREGRESSION

FROM SKLEARN.METRICS IMPORT ACCURACY_SCORE,
CLASSIFICATION_REPORT, CONFUSION_MATRIX
```

📥 Step 2: Load and Explore the Dataset

We will use the **Spam Email dataset**, which consists of labeled emails.

🔹 Load the Dataset

```
# LOAD THE DATASET (ASSUMING A CSV FILE WITH 'MESSAGE' AND
'LABEL' COLUMNS)

DF = PD.READ_CSV('SPAM.CSV', ENCODING='LATIN-1')


# DISPLAY FIRST FEW ROWS

PRINT(DF.HEAD())
```

🔹 Check Dataset Information

```
DF.INFO()

DF.ISNULL().SUM()  # CHECK FOR MISSING VALUES
```

🔹 Understanding the Data

- **message**: The actual email text.

- **LABEL**: THE CLASSIFICATION (SPAM = 1, NOT SPAM = 0).

CONVERT LABELS INTO BINARY VALUES

```
DF = DF[['V1', 'V2']]  # SELECT RELEVANT COLUMNS
DF.COLUMNS = ['LABEL', 'MESSAGE']  # RENAME COLUMNS


# CONVERT LABELS TO NUMERIC VALUES (SPAM = 1, NOT SPAM = 0)
DF['LABEL'] = DF['LABEL'].MAP({'SPAM': 1, 'HAM': 0})


# DISPLAY UPDATED DATA
PRINT(DF.HEAD())
```

🧹 STEP 3: PREPROCESS THE DATA

BEFORE TRAINING THE MODEL, WE NEED TO CLEAN AND PREPROCESS THE TEXT.

🔹 DEFINE A FUNCTION TO CLEAN TEXT

```
DEF PREPROCESS_TEXT(TEXT):
  TEXT = TEXT.LOWER()  # CONVERT TO LOWERCASE
  TEXT = RE.SUB(R'\D+', '', TEXT)  # REMOVE NUMBERS
  TEXT = RE.SUB(R'\S+', ' ', TEXT)  # REMOVE EXTRA SPACES
  TEXT = TEXT.TRANSLATE(STR.MAKETRANS('', '',
STRING.PUNCTUATION))  # REMOVE PUNCTUATION
  RETURN TEXT.STRIP()
```

🔹 Apply Cleaning Function

```
DF['CLEANED_MESSAGE'] =
DF['MESSAGE'].APPLY(PREPROCESS_TEXT)

PRINT(DF.HEAD())  # DISPLAY CLEANED DATA
```

📓 Step 4: Convert Text to Numerical Features

Since machine learning models do not understand text, we need to convert the emails into **numerical features** using **TF-IDF Vectorization**.

🔹 Convert Text to TF-IDF Features

```
VECTORIZER = TFIDFVECTORIZER(STOP_WORDS='ENGLISH',
MAX_FEATURES=5000)


# TRANSFORM TEXT DATA INTO NUMERICAL FORMAT

X = VECTORIZER.FIT_TRANSFORM(DF['CLEANED_MESSAGE'])


# TARGET VARIABLE (SPAM OR NOT SPAM)

Y = DF['LABEL']
```

📊 Step 5: Split the Data into Training and Testing Sets

```
X_TRAIN, X_TEST, Y_TRAIN, Y_TEST = TRAIN_TEST_SPLIT(X, Y,
TEST_SIZE=0.2, RANDOM_STATE=42)
```

```
PRINT("TRAINING DATA SHAPE:", X_TRAIN.SHAPE)

PRINT("TESTING DATA SHAPE:", X_TEST.SHAPE)
```

🤖 STEP 6: TRAIN THE LOGISTIC REGRESSION MODEL

```
# INITIALIZE AND TRAIN THE LOGISTIC REGRESSION MODEL

MODEL = LOGISTICREGRESSION()

MODEL.FIT(X_TRAIN, Y_TRAIN)


PRINT("MODEL TRAINING COMPLETE ✅ ")
```

📝 STEP 7: MAKE PREDICTIONS

```
# PREDICT ON THE TEST SET

Y_PRED = MODEL.PREDICT(X_TEST)
```

📈 STEP 8: EVALUATE THE MODEL

WE EVALUATE THE MODEL USING **ACCURACY, CONFUSION MATRIX, AND CLASSIFICATION REPORT**.

🔹 ACCURACY SCORE

```
ACCURACY = ACCURACY_SCORE(Y_TEST, Y_PRED)

PRINT(F"MODEL ACCURACY: {ACCURACY:.2F}")
```

🔹 CLASSIFICATION REPORT

```
PRINT("CLASSIFICATION REPORT:\N",
CLASSIFICATION_REPORT(Y_TEST, Y_PRED))
```

🔹 CONFUSION MATRIX

```
IMPORT SEABORN AS SNS

IMPORT MATPLOTLIB.PYPLOT AS PLT


# GENERATE CONFUSION MATRIX

CM = CONFUSION_MATRIX(Y_TEST, Y_PRED)


# VISUALIZING THE CONFUSION MATRIX

PLT.FIGURE(FIGSIZE=(5,4))

SNS.HEATMAP(CM, ANNOT=TRUE, FMT="D", CMAP="BLUES",
XTICKLABELS=['NOT SPAM', 'SPAM'], YTICKLABELS=['NOT SPAM',
'SPAM'])

PLT.XLABEL("PREDICTED")

PLT.YLABEL("ACTUAL")

PLT.TITLE("CONFUSION MATRIX")

PLT.SHOW()
```

📝 STEP 9: TEST ON NEW EMAILS

NOW, LET'S TEST THE MODEL WITH SOME SAMPLE EMAILS.

```
DEF PREDICT_SPAM(TEXT):
```

```
    TEXT = PREPROCESS_TEXT(TEXT)  # PREPROCESS THE TEXT
    TEXT_TFIDF = VECTORIZER.TRANSFORM([TEXT])  # CONVERT TO TF-IDF
    PREDICTION = MODEL.PREDICT(TEXT_TFIDF)[0]  # MAKE PREDICTION
    RETURN "SPAM" IF PREDICTION == 1 ELSE "NOT SPAM"

# EXAMPLE EMAILS
EMAIL1 = "CONGRATULATIONS! YOU HAVE WON A LOTTERY. CLICK HERE TO CLAIM YOUR PRIZE."
EMAIL2 = "HI JOHN, CAN WE SCHEDULE A MEETING FOR TOMORROW?"

PRINT("EMAIL 1 PREDICTION:", PREDICT_SPAM(EMAIL1))
PRINT("EMAIL 2 PREDICTION:", PREDICT_SPAM(EMAIL2))
```

✅ FINAL SUMMARY

STEP	DESCRIPTION
STEP 1	INSTALL AND IMPORT NECESSARY LIBRARIES
STEP 2	LOAD AND EXPLORE THE DATASET
STEP 3	PREPROCESS THE TEXT (CLEANING)

Step 4	Convert text to numerical features using TF-IDF
Step 5	Split data into training and testing sets
Step 6	Train a **Logistic Regression model**
Step 7	Make predictions on the test set
Step 8	Evaluate the model using accuracy and confusion matrix
Step 9	Test the model on new email samples

🌟 Conclusion

- We successfully built a **Spam Email Classifier** using **Logistic Regression**.
- The model can predict whether an email is **Spam** or **Not Spam** based on its text content.
- The **TF-IDF** technique helped convert text into numerical values.
- This classifier can be further **improved** by using **advanced models** like **Naive Bayes** or **Deep Learning**.