# Day 6 – Logistic Regression and Introduction to K-Nearest Neighbors (KNN)

---

## 📘 Today's Highlights:

Today, we extended our understanding of **Logistic Regression** by working with the **Iris dataset** loaded from a CSV file using **pandas**.
We implemented a complete training, testing, and evaluation workflow. This included saving the model using **joblib** and predicting the species based on user input.

We also discussed the **basics of K-Nearest Neighbors (KNN)**, a simple yet powerful classification algorithm that uses feature similarity for prediction.

---

## □ Logistic Regression (Recap):

Logistic Regression is a **supervised machine learning algorithm** used for **classification**.
It models the probability that an input belongs to a particular class using a **logistic function** (sigmoid).
It is widely used in binary and multi-class classification problems such as spam detection, disease prediction, etc.

---

## 📚 Introduction to K-Nearest Neighbors (KNN):

KNN is a **non-parametric classification algorithm** that predicts the class of a data point by looking at the 'k' closest data points in the training set.
It relies on distance metrics such as **Euclidean distance** to measure similarity.
KNN is simple to implement and works well for smaller datasets.

---

## 🛠□ Practical Implementation (Python Code):

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
import joblib

# Load the Iris dataset
df = pd.read_csv('IRIS.csv')
print("First few rows of the dataset:")
print(df.head())

# Features and target
```

```python
X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']].values
y = df['species'].values

# Split 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)

# Train logistic regression model
model = LogisticRegression(multi_class='ovr', random_state=42)
model.fit(X_train, y_train)

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

# Evaluate model
accuracy = accuracy_score(y_test, y_pred)
print("\nModel Evaluation:")
print(f"Accuracy: {accuracy:.2f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Save model
joblib.dump(model, 'iris_model.pkl')
print("\nModel saved as 'iris_model.pkl'")

# Function to predict species from user input
def predict_iris_species(model, feature_names):
    print("\nEnter values (in cm):")
    user_input = []
    for feature in feature_names:
        while True:
            try:
                value = float(input(f"{feature}: "))
                user_input.append(value)
                break
            except ValueError:
                print("Enter a valid number.")

    # Convert input to array and predict
    user_input = np.array(user_input).reshape(1, -1)
    predicted_species = model.predict(user_input)[0]
    print(f"\nPredicted Species: {predicted_species}")

# Test with user input
feature_names = ['Sepal Length', 'Sepal Width', 'Petal Length', 'Petal Width']
print("\nTesting with user input:")
predict_iris_species(model, feature_names)
)
```

---

## 📊 Output & Summary:

- The model achieved **high accuracy** in predicting species.
- It successfully outputted a **classification report** showing precision, recall, and F1-score.

- A sample input from the user was correctly classified by the model.
- The model was **saved for future use** using `joblib`.

This activity reinforced the full machine learning workflow:
**Load data → Train → Evaluate → Predict → Save model**

```
First few rows of the dataset:
   sepal_length  sepal_width  petal_length  petal_width      species
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa

Model Evaluation:
Accuracy: 0.97
```

```
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      0.89      0.94         9
 Iris-virginica       0.92      1.00      0.96        11

       accuracy                           0.97        30
      macro avg       0.97      0.96      0.97        30
   weighted avg       0.97      0.97      0.97        30


Model saved as 'iris_model.pkl'
```

```
Testing with user input:

Enter values (in cm):
Sepal Length: 5.1
Sepal Width: 3.5
Petal Length: 1.4
Petal Width: 0.2

Predicted Species: Iris-setosa
```