🔥 What is Scikit-Learn?

Scikit-learn is a **machine learning library** for Python. It provides:

✅ **Data Preprocessing** → Handling missing values, scaling, encoding
✅ **Machine Learning Models** → Regression, Classification, Clustering
✅ **Model Evaluation** → Accuracy, Precision, Recall
✅ **Feature Engineering** → Selecting important features
✅ **Pipelines & Automation** → Make ML workflows efficient

🔥 1. Install & Import Scikit-Learn

If you haven't installed it yet, run:

```
pip install scikit-learn
```

Now import the essential modules:

```
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.preprocessing import StandardScaler, LabelEncoder

from sklearn.linear_model import LinearRegression

from sklearn.metrics import accuracy_score, mean_squared_error
```

🔥 2. Load and Prepare Data

Let's use a simple dataset:

```
from sklearn.datasets import load_diabetes

# Load dataset

data = load_diabetes()

df = pd.DataFrame(data.data, columns=data.feature_names)

df['target'] = data.target  # Add target column

print(df.head())
```

✅ **Diabetes dataset** has medical records of patients

🔥 3. Data Preprocessing

Before training an ML model, **data must be cleaned & scaled**.

✅ **Handle Missing Values**

```
df.fillna(df.mean(), inplace=True)  # Replace NaN with mean
```

✅ **Fills missing values with the mean**

✅ **Feature Scaling**

```
scaler = StandardScaler()

df_scaled = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
```

✅ **Standardizes the dataset for better performance**

✅ **Convert Categorical Data**

```
encoder = LabelEncoder()

df["target"] = encoder.fit_transform(df["target"])
```

✅ **Encodes categorical values into numbers**

🔥 4. Train-Test Split

Divide the dataset into **training (80%) and testing (20%)**:

```
X = df.drop(columns=["target"])  # Features

y = df["target"]  # Target variable

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

print(X_train.shape, X_test.shape)  # Check sizes
```

✅ **Ensures model doesn't overfit by testing on unseen data**

🔥 5. Train a Machine Learning Model

Let's train a **Linear Regression Model**:

```
model = LinearRegression()

model.fit(X_train, y_train)  # Train the model


y_pred = model.predict(X_test)  # Make predictions
```

✅ **Finds the best-fit line for regression problems**

🔥 6. Evaluate Model Performance

Check how well the model performs:

```
mse = mean_squared_error(y_test, y_pred)

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

✅ **Lower MSE = Better model performance**

🔥 7. Classification with Scikit-Learn

Classification is used when the output is **categorical** (e.g., Spam/Not Spam, Pass/Fail).

We'll use the **Iris dataset** 📊

```
from sklearn.datasets import load_iris

# Load dataset

data = load_iris()

df = pd.DataFrame(data.data, columns=data.feature_names)

df["target"] = data.target  # Add target column

print(df.head())  # Check dataset
```

✅ **Iris dataset** has 3 flower species (**Setosa, Versicolor, Virginica**)

🔥 8. Train-Test Split for Classification

```
X = df.drop(columns=["target"])  # Features

y = df["target"]  # Target (0, 1, 2 for each species)

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

print(X_train.shape, X_test.shape)  # Check sizes
```

✅ **80% Training, 20% Testing**

🔥 9. Train a Classification Model (Decision Tree)

```
from sklearn.tree import DecisionTreeClassifier

# Initialize & Train

model = DecisionTreeClassifier()

model.fit(X_train, y_train)

# Predict

y_pred = model.predict(X_test)
```

✅ **Decision Tree finds patterns to classify species**

🔥 10. Evaluate Model Performance

```
from sklearn.metrics import accuracy_score, classification_report

acc = accuracy_score(y_test, y_pred)

print(f"Accuracy: {acc}")

print(classification_report(y_test, y_pred))
```

✅ **Accuracy** → Measures how many predictions were correct
✅ **Classification Report** → Shows precision, recall, F1-score

---

🔥 11. Try a Different Model (Support Vector Machine - SVM)

```
from sklearn.svm import SVC
model = SVC()  # Support Vector Classifier
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
```

✅ **SVM works well for complex patterns**

🔥 12. Clustering with Scikit-Learn

Clustering is **unsupervised learning**, meaning we don't have labeled data. It groups similar data points together.

🔥 13. K-Means Clustering (Customer Segmentation Example)

```
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# Generate synthetic data
from sklearn.datasets import make_blobs
X, _ = make_blobs(n_samples=300, centers=3, cluster_std=1.0, random_state=42)
# Apply K-Means
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)
labels = kmeans.labels_
# Plot the clusters
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap="viridis")
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
c="red", marker="X", s=200, label="Centers")
plt.legend()
plt.title("K-Means Clustering")
plt.show()
```

✅ **Automatically groups data into 3 clusters**
✅ **Red 'X' marks are cluster centers**

🔥 14. DBSCAN (Density-Based Clustering)

Unlike K-Means, **DBSCAN** finds clusters based on density, good for non-spherical clusters.

```
from sklearn.cluster import DBSCAN
# Apply DBSCAN
```

```
dbscan = DBSCAN(eps=0.5, min_samples=5)
labels = dbscan.fit_predict(X)
# Plot Clusters
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap="rainbow")
plt.title("DBSCAN Clustering")
plt.show()
```

✅ **Can detect noise points (outliers)**

🔥 15. Hierarchical Clustering (Dendrogram Example)

```
from scipy.cluster.hierarchy import dendrogram, linkage
# Compute linkage
linked = linkage(X, method="ward")
# Plot Dendrogram
plt.figure(figsize=(10, 5))
dendrogram(linked)
plt.title("Hierarchical Clustering Dendrogram")
plt.show()
```

✅ **Dendrogram shows how clusters merge**

🔥 16. Model Selection (Choosing the Best Model Automatically)

Different models perform differently depending on the dataset. **Scikit-Learn's GridSearchCV** and **RandomizedSearchCV** help find the best model and parameters.

✅ **GridSearchCV (Exhaustive Search for Best Parameters)**

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
# Define parameter grid
param_grid = {
    "n_estimators": [10, 50, 100],
    "max_depth": [None, 5, 10],
    "min_samples_split": [2, 5, 10],
}
# Initialize classifier
model = RandomForestClassifier()
# Apply GridSearch
grid_search = GridSearchCV(model, param_grid, cv=5, scoring="accuracy")
```

```
grid_search.fit(X_train, y_train)
print(f"Best Parameters: {grid_search.best_params_}")
print(f"Best Score: {grid_search.best_score_}")
```

✅ **Finds the best combination of hyperparameters**

🔥 17. Pipelines (Automating the ML Workflow)

Instead of writing separate steps for **preprocessing, training, and prediction**, we can use **Pipelines**.

✅ **Example: Automate a Classification Task**

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
# Create a pipeline
pipeline = Pipeline([
    ("scaler", StandardScaler()),  # Normalize Data
    ("svm", SVC(kernel="linear"))  # Train SVM Model
])
# Train & Predict
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
# Evaluate
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
```

✅ **Automates Scaling + Model Training in one step**

🔥 18. Cross-Validation (Avoid Overfitting)

Instead of training & testing only once, **cross-validation** ensures the model performs well on different data splits.

```
from sklearn.model_selection import cross_val_score


# Apply cross-validation
scores = cross_val_score(RandomForestClassifier(), X, y, cv=5,
scoring="accuracy")


print(f"Cross-Validation Accuracy: {scores.mean():.2f}")
```

✅ **Reduces the risk of overfitting**