Dhirubhai Ambani University

(Formerly known as DA-IICT)

Topic: Implementation of ML models

Course: Programming Lab

Course Code- PC503

Dr. Ankit Vijayvargiya

Assistant Professor

Room No. 4205, Faculty Block 4

Email: ankit_Vijayvargiya[at]dau.ac.in

Phone: 079-68261628(O), 7877709590(M)

Dhirubhai Ambani University
Formerly known as DA-IICT

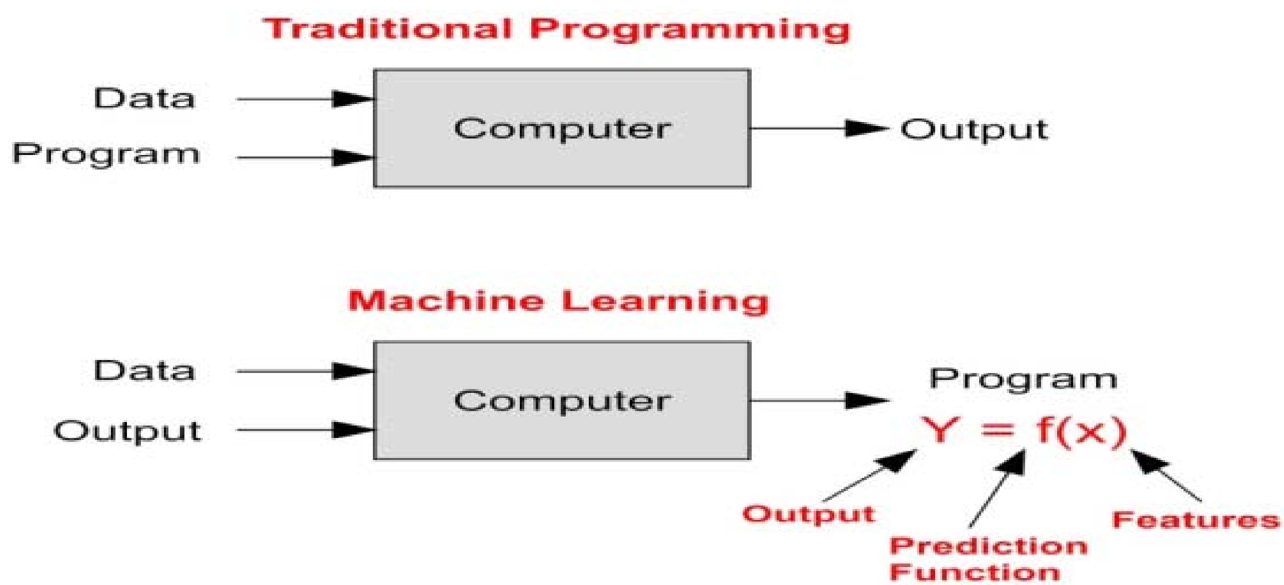
Python Library: Scikit-learn

- Scikit-learn is a third party machine learning module for Python.
- Key Features: Simple and efficient tools for data analysis.
- Built on NumPy, SciPy, and Matplotlib.
- It features various machine learning models for classification, regression, and clustering algorithms.

Dhirubhai Ambani University
Formerly known as DA-IICT

Python Library: Scikit-learn

What is the Machine Learning:

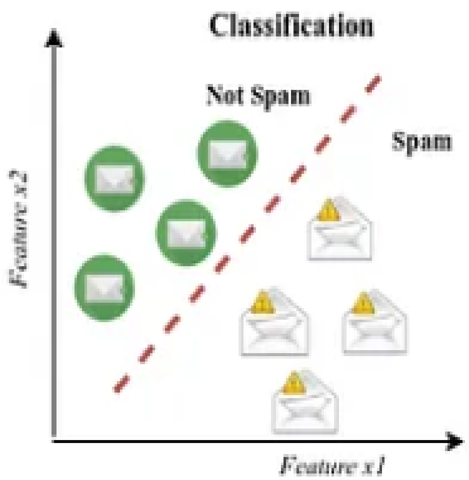


- Training: given a training set of labeled examples $(X_1, Y_1), \ldots, (X_n, Y_n)$, estimate the prediction function $f(x)$ by minimizing the prediction error on the training set.
- Testing: apply f to a never before seen test example x and output the predicted value $Y = f(x)$.

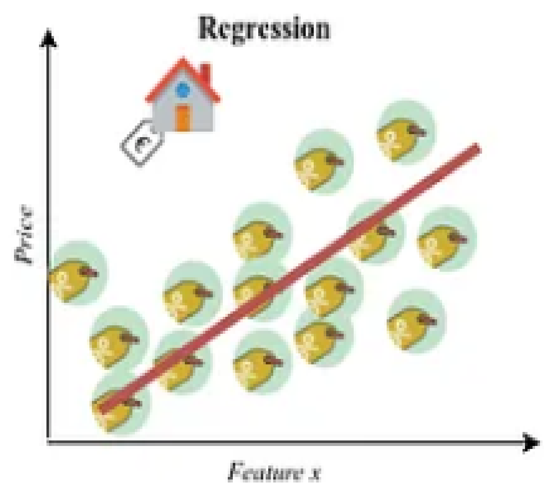
Python Library: Scikit-learn

ML Problem formulated as:

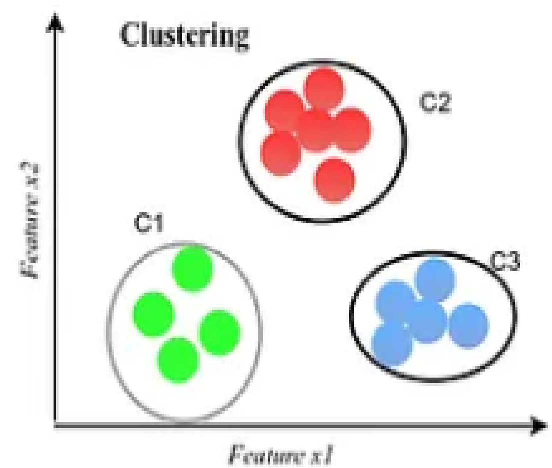
Classification



Regression



Clustering



Python Library: Scikit-learn

Classification

- **Definition:** Predicting a discrete category or class label.
- **Goal:** To classify new data points into predefined groups.
- **Example:** Spam detection (spam/not spam), image recognition (cat/dog), disease diagnosis.
- **Key concept:** Uses labeled training data to learn the mapping from input to output.
- **Algorithms:** Decision Trees, Logistic Regression, Random Forests, Gradient Boosting.

Python Library: Scikit-learn

Regression

- **Definition:** Predicting a continuous numerical value.
- **Goal:** To find the relationship between variables to forecast a value.
- **Example:** Predicting house prices, stock prices, or future sales.
- **Key concept:** Uses labeled historical data to predict continuous outcomes.
- **Algorithms:** Linear Regression, Polynomial Regression, Lasso, Ridge.

Dhirubhai Ambani University
Formerly known as DA-IICT

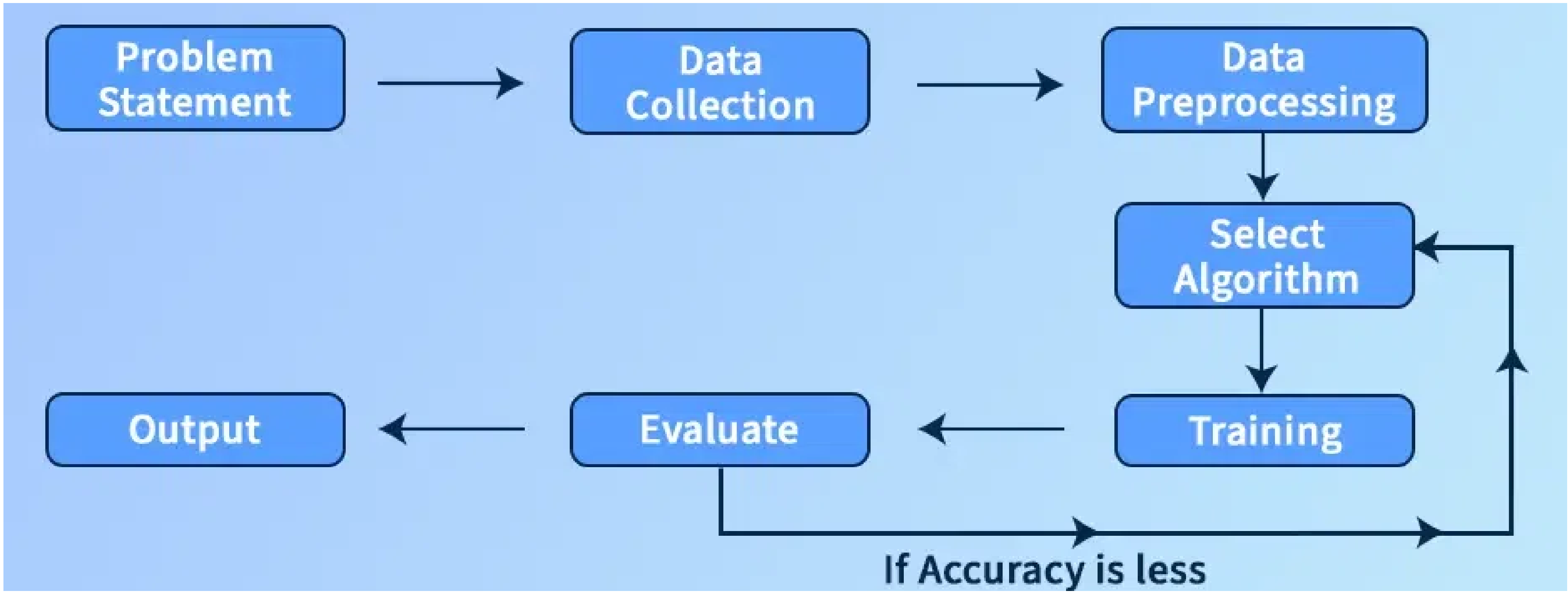
Python Library: Scikit-learn

Clustering

- **Definition:** Grouping similar data points together without any prior labels.
- **Goal:** To discover hidden patterns and structures in unlabeled data.
- **Example:** Customer segmentation, grouping similar documents, or identifying regions in an image.
- **Key concept:** The algorithm discovers the "classes" or clusters on its own.
- **Algorithms:** K means clustering, Hierarchical Clustering.

Python Library: Scikit-learn

ML model implementation process



Dhirubhai Ambani University
Formerly known as DA-IICT

Python Library: Scikit-learn

Pre-processing: Standardization

Preprocessing is essential to prepare data for machine learning models, ensuring they perform effectively.

Standardization: Standardization rescales the data so that each feature has a mean of 0 and a standard deviation of 1. This process transforms the distribution to resemble a standard normal distribution (Gaussian).

$$z = \frac{(x - \mu)}{\sigma}$$

When to Use:

- When data follows a normal (Gaussian) distribution.
- Common for algorithms like Linear/Logistic Regression, SVM, and PCA.

```
from sklearn.preprocessing import StandardScaler

# Apply Standardization
scaler = StandardScaler()
scaled_data = scaler.fit_transform(X)
```

Dhirubhai Ambani University
Formerly known as DA-IICT

Python Library: Scikit-learn

Pre-processing: Normalization

Normalization: Normalization rescales all feature values into a fixed range between 0 and 1. This ensures that all features contribute proportionally to model training.

$$x' = \frac{(x - x_{min})}{(x_{max} - x_{min})}$$

When to Use:

- When features have different units or scales (e.g., age in years, income in dollars).
- Often used for neural networks and distance-based models (KNN, clustering).

```python
from sklearn.preprocessing import MinMaxScaler

# Apply Normalization
scaler = MinMaxScaler()
normalized_data = scaler.fit_transform(X)
```

Dhirubhai Ambani University
Formerly known as DA-IICT

Python Library: Scikit-learn

Train/Test Split

- Parameters:
  - X: Features (input variables)
  - y: Target variable (output)
  - test_size: Proportion of the dataset to include in the test split (e.g., 0.2 for 20%)
  - train_size: Proportion of the dataset to include in the train split (optional)
  - random_state: Seed for random number generator (ensures reproducibility)
  - shuffle: Whether to shuffle the data before splitting (default is True).

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

Python Library: Scikit-learn

Model: SVM classifier

```
#import
from sklearn.svm import SVC

#Create a LR model
model = SVC()

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

# Make predictions on the test set
y_predict=model.predict(X_test)
```

Model: Support Vector Regression

```
#import
from sklearn.svm import SVR

#Create a LR model
model = SVR()

#Train the model
model.fit(Xr_train,yr_train)

# Make predictions on the test set
y_predict=model.predict(Xr_test)
```

Dhirubhai Ambani University
Formerly known as DA-IICT

Python Library: Scikit-learn

Sklearn.metrics

- Provides functions to evaluate the model performance

- Key metrics:
  - Classification: accuracy, precision, recall, F1score
  - Regression: Mean absolute error, mean squared error

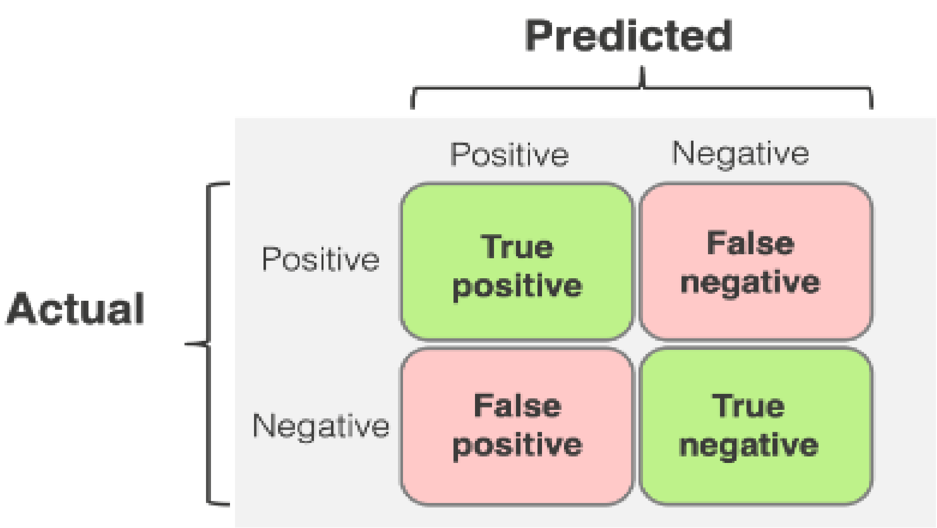
Dhirubhai Ambani University
Formerly known as DA-IICT

Python Library: Scikit-learn

Classification Evaluation Metrics:

Confusion metrics

- A table used to evaluate the performance of a classification model
- Components: TP, TN, FP, FN



Accuracy: The ratio of correctly predicted data samples to the total samples.

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

Precision: The ratio of positive predictions to total predicted positives

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

Recall: The ratio of true positive predictions to total actual positives

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

F1-Score: The harmonic mean of precision and recall, balancing both the metrics

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

Python Library: Scikit-learn

Evaluation Metrics for classification

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report

# Calculate metrics
accuracy = accuracy_score(y_test, y_predict)
precision = precision_score(y_test, y_predict, average='weighted')  # or 'macro', 'micro' depending on need
recall = recall_score(y_test, y_predict, average='weighted')
f1 = f1_score(y_test, y_predict, average='weighted')

# Print results
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
```

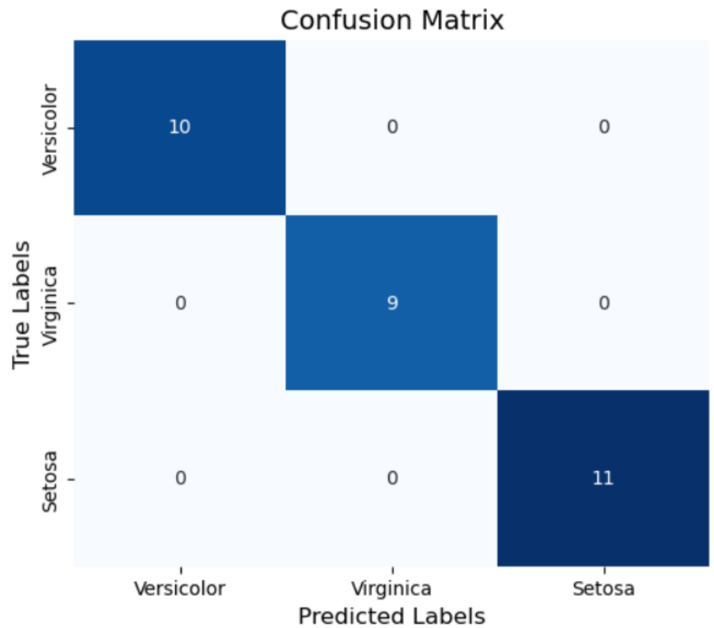
Dhirubhai Ambani University
Formerly known as DA-IICT

Confusion Metrics for classification

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# ---- Confusion Matrix ----
cm = confusion_matrix(y_test, y_predict)
print("\nConfusion Matrix:\n", cm)

# ---- Seaborn Heatmap ----
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, cmap='Blues',
            xticklabels=set(y_test), yticklabels=set(y_test))
plt.title("Confusion Matrix", fontsize=14)
plt.xlabel("Predicted Labels", fontsize=12)
plt.ylabel("True Labels", fontsize=12)
plt.show()
```



Python Library: Scikit-learn

Regression Evaluation Metrics:

- Mean Absolute Error: Average absolute difference between predicted and actual values.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- Mean Squared Error: Average of squared differences between predicted and actual values.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- Root Mean Squared Error (RMSE): Square root of MSE — brings error back to original units of target variable.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- R^2 Score (Coefficient of Determination): Measures how well predictions approximate real data.

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

Interpretation:

- $R^2 = 1 \rightarrow$ Perfect fit.
- $R^2 = 0 \rightarrow$ Model is as good as mean prediction.
- $R^2 < 0 \rightarrow$ Model is worse than predicting the mean.

Dhirubhai Ambani University
Formerly known as DA-IICT

Python Library: Scikit-learn

Evaluation Metrics for Regression

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

#import
from sklearn.svm import SVR
#Create a LR model
model = SVR()
#Train the model
model.fit(Xr_train,yr_train)
# Make predictions on the test set
y_predict=model.predict(Xr_test)

# Calculate metrics
mse = mean_squared_error(yr_test, y_predict)
mae = mean_absolute_error(yr_test, y_predict)
rmse = np.sqrt(mse)
r2 = r2_score(yr_test, y_predict)

print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("Root Mean Squared Error (RMSE):", rmse)
print("R^2 Score:", r2)
```

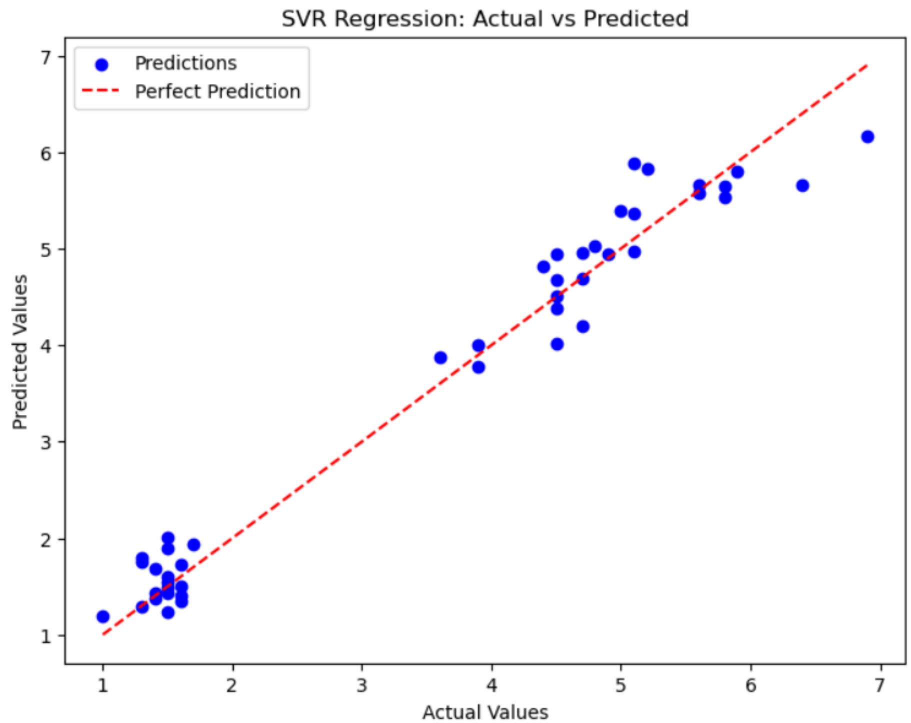
```
Mean Squared Error (MSE): 0.10788323379073202
Mean Absolute Error (MAE): 0.251139218061077
Root Mean Squared Error (RMSE): 0.32845583232868925
R^2 Score: 0.968005971004878
```

Dhirubhai Ambani University
Formerly known as DA-IICT

Python Library: Scikit-learn

Plot Actual vs Predicted

```
# Plot actual vs predicted
plt.figure(figsize=(8,6))
plt.scatter(yr_test, y_predict, color='blue', label='Predictions')
plt.plot([min(yr_test), max(yr_test)], [min(yr_test), max(yr_test)],
         color='red', linestyle='--', label='Perfect Prediction')
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("SVR Regression: Actual vs Predicted")
plt.legend()
plt.show()
```



Dhirubhai Ambani University
Formerly known as DA-IICT

Python Library: Scikit-learn

K-Mean Clustering

```python
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import pandas as pd
import matplotlib.pyplot as plt

# K-Means clustering (3 clusters)
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)
clusters = kmeans.predict(X)

# Plot using first two features
plt.figure(figsize=(8,6))
plt.scatter(X.iloc[:,0], X.iloc[:,1], c=clusters, cmap='viridis', s=50, label='Data Points')
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1],
            color='red', marker='X', s=200, label='Centroids')

plt.xlabel("Sepal Length (cm)")
plt.ylabel("Sepal Width (cm)")
plt.title("K-Means Clustering on Iris Data")
plt.legend()
plt.show()
```