EXP NO: 3: Study of KNN, Decision Tree, Random Forest, and SVM classifiers on Iris Dataset.

Aim:- To implement and compare the performance of KNN, Decision Tree, Random Forest, and SVM classifiers using the iris dataset, based on statistical evaluation metrics such as accuracy, precision, recall, and F1-score

Algorithm:-

1. Load Dataset: Import the Iris dataset from sklearn.datasets
2. Data splitting: Divide the datasets into training and testing sets using train_test_split
3. Feature scaling: Normalize the feature data using standard scaler to ensure equal treatment of feature
4. Model Definition: Define the classification models
   → KNN
   → Decision Tree
   → Random Forest
   → SVM
5. Training and prediction:
   → For each model, train it on training data
   → predict the output using test data
6. Evaluation metrics: Compute Accuracy, precision, Recall & F1-Score using sklearn.metrics.
7. Comparision:- print and Compare the performance metrics

Code:-

```
        from sklearn.datasets import load_iris
    from sklearn, model_selection import train_test_split
   from sklearnpreprocessing import standard scaler
   from sklearn.neighbors import kNeighbors classifier
   from sklearn. tree import Decision Tree classifier
   from sklearn ensemble import RandomForest classifier
   from   sklearn, svm  import   svc
   from   sklearn. metrics import (
          accuracy_score, precision_score, recall_score,
        f1_score    )

  # step 1 : Load data
    iris  = load_iris()
      x   = iris.data
      y   = iris.target

  # step 2 : Train_test split
  x_train , x_test, y_train, y_test = train_test_
     split (x, y, test_size=0.3, random_state=42)

  # step 3 : scaling
    scaler = standard scaler()
      x_train = scaler. fit_transform(x_train)
      x_test  = scaler. transform (x_test)

  # step 4 : Define models
      models = {
        "KNN": kNeighborsclassifier (n_neighbors=3)
```

```
"Decision Tree" : Decision Tree classifier (),
"Random Forest" : Random Forest classifier(),
  "SVM" : SVC()  }
#step 5: Train, predict, evaluate each model
print("Model performance Comparision : \n")
for name, model in models.items():
   model.fit (x_train, y_train)
   y_pred = model.predict(x_test)

accuracy = accuracy_score (y_test, y_pred)
precision = precision_score (y_test, y_pred, average='macro')
recall = recall_score (y_test, y_pred, average ='macro')
 f1 = f1_score (y_test, y_pred, average ='macro')
print (f"{name} Results:")
print (f" Accuracy: {accuracy :.2f}")
print(f" precision : {precision: .2f}")
print (f" Recall : {recall:.2f}")
print (f" F1 score : {f1: .2f}")
 print ("-" *35)
```

Conclusion:- All four classifiers - KNN, Decision Tree, Random Forest, SVM - performed well on the iris dataset due to its simple and well separated classes. In pratical scenarios with complex or noisy data, such Comparisions help in selecting the most suitable model based on performance & efficiency

Definitions :-

1. Accuracy : The proportion of Correct prediction out of all predictions made

$$\text{Accuracy} = \frac{\text{True positives} + \text{True Negatives}}{\text{Total predictions}}$$

2. Precision :- of all the samples the model predicted as positive, how many were actually positive

$$\text{precision} = \frac{\text{True positives}}{\text{True positives} + \text{False positives}}$$

3. Recall :- of all the actual positive samples how many did the model correctly identified

$$\text{Recall} = \frac{\text{True positives}}{\text{True positives} + \text{False Negatives}}$$

4. F1 Score :- The harmonic mean of precision and recall. it balances both metrics into a single number.

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

Libraries used and why:-

1. pandas :- used for data manipulation and analysis

2. matplotlib.pyplot :- A foundational plotting library

3. seaborn :- Built on top of matplotlib, seaborn provides a higher level interface for drawing attractive and informative

4. scikit-learn Components:-

- Load_iris from sklearn.datasets
→ load iris dataset

- train_test_split
→ split data into training and testing

- standard scaler
→ Normalize by removing the mean and scaling to unit variance.

- Metrics
→ used to evaluate model performance on test data using different criterias.

macro :- Averaging means equal weight to each class.

(Stratify = y) → Not in use

output:

Model performance Comparision:

kNN Results:

Accuracy : 1.00

precision : 1.00

Recall : 1.00

F1 score : 1.00

Decision Tree Results:

Accuracy : 1.00

precision : 1.00

Recall : 1.00

F1 score : 1.00

Random Forest Results:

Accuracy : 1.00

precision : 1.00

Recall : 1.00

F1 score : 1.00

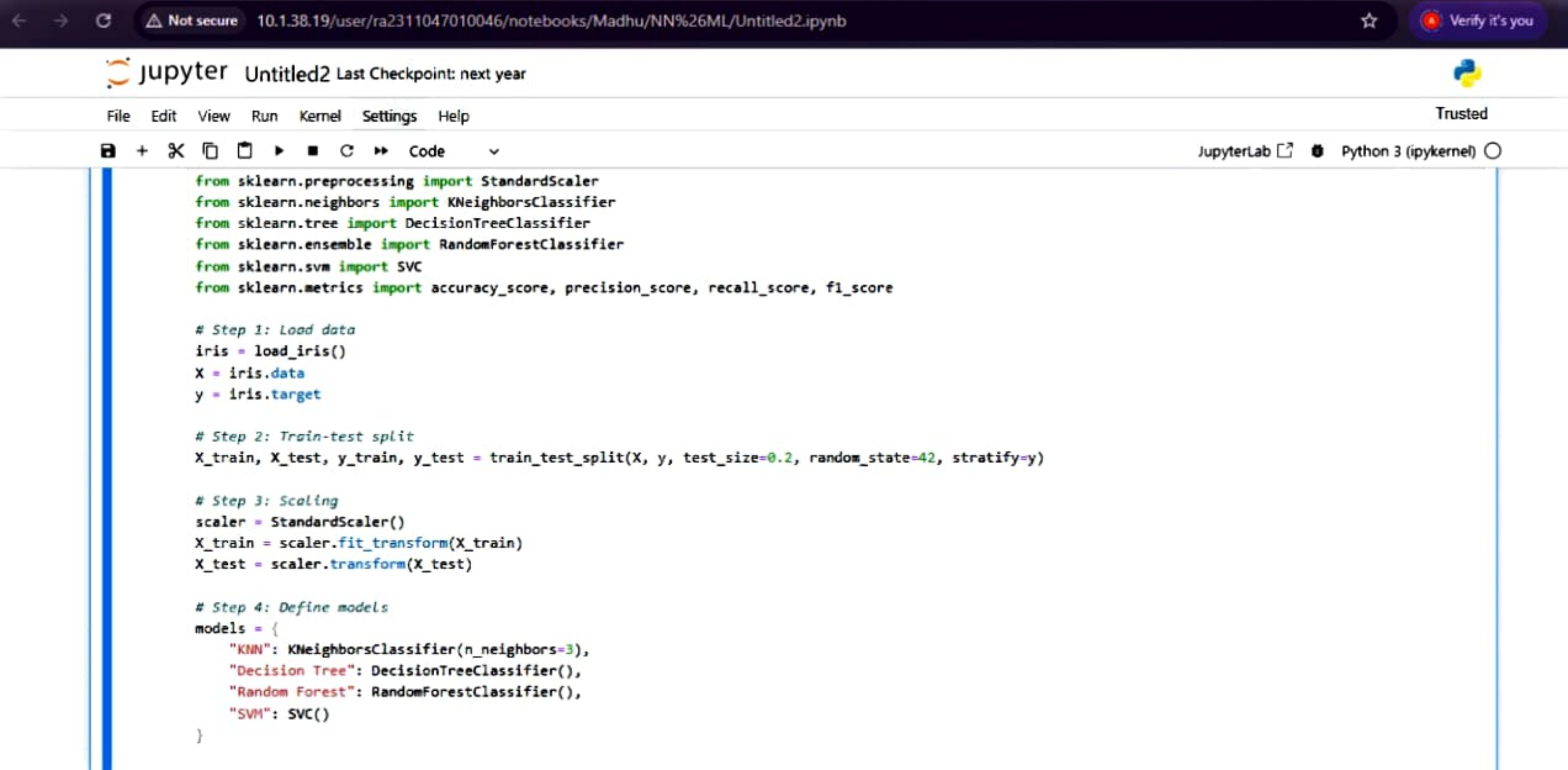
SVM Results:

Accuracy : 1.00

precision : 1.00

Recall : 1.00

F1 score : 1.00


Not secure 10.1.38.19/user/ra2311047010046/notebooks/Madhu/NN%26ML/Untitled2.ipynb

Verify it's you
jupyter Untitled2 Last Checkpoint: next year
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)

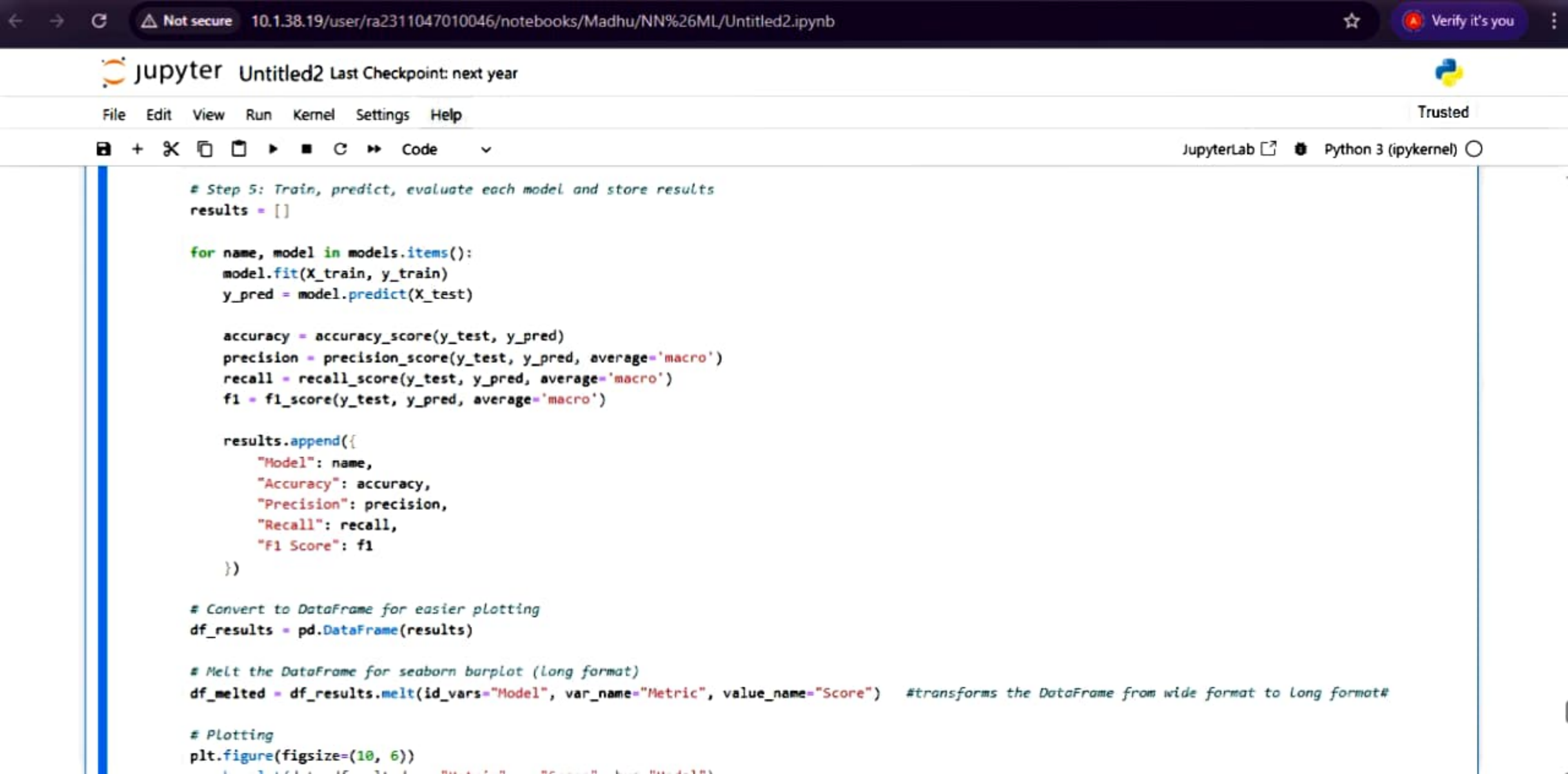
```python
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Step 1: Load data
iris = load_iris()
X = iris.data
y = iris.target

# Step 2: Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Step 3: Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 4: Define models
models = {
    "KNN": KNeighborsClassifier(n_neighbors=3),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "SVM": SVC()
}
```


Not secure
10.1.38.19/user/ra2311047010046/notebooks/Madhu/NN%26ML/Untitled2.ipynb
Verify it's you
jupyter
Untitled2 Last Checkpoint: next year
File
Edit
View
Run
Kernel
Settings
Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)

```python
# Step 5: Train, predict, evaluate each model and store results
results = []

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='macro')
    recall = recall_score(y_test, y_pred, average='macro')
    f1 = f1_score(y_test, y_pred, average='macro')

    results.append({
        "Model": name,
        "Accuracy": accuracy,
        "Precision": precision,
        "Recall": recall,
        "F1 Score": f1
    })

# Convert to DataFrame for easier plotting
df_results = pd.DataFrame(results)

# Melt the DataFrame for seaborn barplot (long format)
df_melted = df_results.melt(id_vars="Model", var_name="Metric", value_name="Score")   #transforms the DataFrame from wide format to long format#

# Plotting
plt.figure(figsize=(10, 6))
```

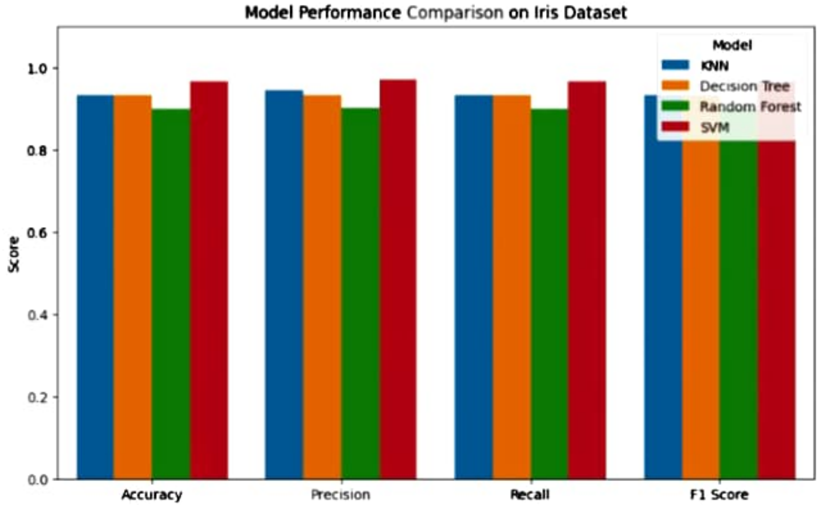
10.1.38.19/user/ra2311047010046/notebooks/Madhu/NN%26ML/Untitled2.ipynb

jupyter Untitled2 Last Checkpoint: next year

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```python
# Plotting
plt.figure(figsize=(10, 6))
sns.barplot(data=df_melted, x="Metric", y="Score", hue="Model")
plt.ylim(0, 1.1)
plt.title("Model Performance Comparison on Iris Dataset")
plt.ylabel("Score")
plt.legend(title="Model")
plt.show()
```


Model Performance Comparison on Iris Dataset
Score
Accuracy
Precision
Recall
F1 Score
Model
KNN
Decision Tree
Random Forest
SVM