Step 2: Working with Data and Basic ML Concepts

Objective

Learn the basics of machine learning using scikit-learn and understand how to work with datasets for training ML models.

Context

Machine Learning (ML) allows computers to learn from data and make predictions without being explicitly programmed. In this step, you'll learn how to handle data, explore basic datasets, and create a simple classification model using scikit-learn.

Why it is required

Understanding basic ML concepts is essential before integrating ML models into an API. This step provides the foundation for:

- Data handling and preprocessing
- Model selection and training
- Evaluation of model performance
- Making predictions with trained models

These skills are necessary for building intelligent APIs that can provide predictions based on input data.

How to achieve this

1. Set up your environment

Update your requirements to include scikit-learn, numpy, and pandas:

```
# Install the required packages
pip install scikit-learn numpy pandas matplotlib
```

2. Create a new file named `ml_basics.py`

This will contain our ML-related code:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
```

```
def main():
    # Load the Iris dataset (a classic beginner ML dataset)
    print("Loading Iris dataset...")
    iris = load_iris()
    X = iris.data  # Features
    y = iris.target  # Target variable
    feature_names = iris.feature_names
    target_names = iris.target_names

    # Print basic information about the dataset
    print(f"Dataset shape: {X.shape}")
    print(f"Number of classes: {len(target_names)}")
    print(f"Classes: {target_names}")
    print(f"Features: {feature_names}")

    # Create a pandas DataFrame for easier data handling
    df = pd.DataFrame(X, columns=feature_names)
    df['target'] = y
    df['species'] = df['target'].apply(lambda x: target_names[x])

    # Display the first 5 rows of the dataset
    print("\nFirst 5 rows of the dataset:")
    print(df.head())

    # Basic data analysis
    print("\nBasic statistics:")
    print(df.describe())

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3, random_state=42
    )
    print(f"\nTraining set size: {X_train.shape}")
    print(f"Testing set size: {X_test.shape}")

    # Standardize the features (important for many ML algorithms)
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Train a simple K-Nearest Neighbors classifier
    print("\nTraining a K-Nearest Neighbors classifier...")
    knn = KNeighborsClassifier(n_neighbors=5)
    knn.fit(X_train_scaled, y_train)

    # Make predictions on the test set
    y_pred = knn.predict(X_test_scaled)

    # Evaluate the model
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Model accuracy: {accuracy:.2f}")

    # Print a detailed classification report
    print("\nClassification Report:")
```

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

    # Make a prediction for a new sample
    new_sample = np.array([[5.1, 3.5, 1.4, 0.2]])  # Example: likely a setosa
    new_sample_scaled = scaler.transform(new_sample)
    prediction = knn.predict(new_sample_scaled)
    predicted_species = target_names[prediction[0]]

    print(f"\nPrediction for new sample {new_sample[0]}: {predicted_species}")

    # Plot the data for visualization (only using 2 features for simplicity)
    plt.figure(figsize=(10, 6))
    colors = ['blue', 'green', 'red']

    for i, species in enumerate(target_names):
        plt.scatter(
            df[df['target'] == i]['sepal length (cm)'],
            df[df['target'] == i]['sepal width (cm)'],
            c=colors[i],
            label=species
        )

    plt.xlabel('Sepal Length (cm)')
    plt.ylabel('Sepal Width (cm)')
    plt.title('Iris Dataset: Sepal Length vs Sepal Width')
    plt.legend()
    plt.savefig('iris_visualization.png')
    plt.close()
    print("\nVisualization saved as 'iris_visualization.png'")

    return knn, scaler  # Return the trained model and scaler for later use

if __name__ == "__main__":
    main()
```

3. Run the ML basics script

```
python ml_basics.py
```

4. Analyze the output and visualization

- Look at the dataset information and statistics
- Review the model's performance metrics
- Examine the visualization of the dataset

Examples of usage

Running the ML code and analyzing results

```
python ml_basics.py
```

Sample output:

```
Loading Iris dataset...
Dataset shape: (150, 4)
Number of classes: 3
Classes: ['setosa' 'versicolor' 'virginica']
Features: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal
width (cm)']

First 5 rows of the dataset:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
target species
0                5.1               3.5                1.4               0.2
0  setosa
1                4.9               3.0                1.4               0.2
0  setosa
2                4.7               3.2                1.3               0.2
0  setosa
3                4.6               3.1                1.5               0.2
0  setosa
4                5.0               3.6                1.4               0.2
0  setosa

Basic statistics:
       sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
target
count         150.000000        150.000000         150.000000        150.000000
150.000000
mean            5.843333          3.057333           3.758000          1.199333
1.000000
std             0.828066          0.435866           1.765298          0.762238
0.819232
min             4.300000          2.000000           1.000000          0.100000
0.000000
25%             5.100000          2.800000           1.600000          0.300000
0.000000
50%             5.800000          3.000000           4.350000          1.300000
1.000000
75%             6.400000          3.300000           5.100000          1.800000
2.000000
max             7.900000          4.400000           6.900000          2.500000
2.000000

Training set size: (105, 4)
Testing set size: (45, 4)

Training a K-Nearest Neighbors classifier...
Model accuracy: 0.98
```

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

      setosa       1.00      1.00      1.00        16
  versicolor       0.94      1.00      0.97        16
   virginica       1.00      0.92      0.96        13

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.98        45
weighted avg       0.98      0.98      0.98        45

Prediction for new sample [5.1 3.5 1.4 0.2]: setosa

Visualization saved as 'iris_visualization.png'
```

Using the trained model for predictions

You can modify the code to make predictions for other samples:

```python
# Make a prediction for multiple new samples
new_samples = np.array([
    [5.1, 3.5, 1.4, 0.2],  # Likely setosa
    [6.2, 2.9, 4.3, 1.3],  # Likely versicolor
    [7.2, 3.6, 6.1, 2.5]   # Likely virginica
])
new_samples_scaled = scaler.transform(new_samples)
predictions = knn.predict(new_samples_scaled)

for i, pred in enumerate(predictions):
    print(f"Sample {i+1}: {new_samples[i]} - Predicted: {target_names[pred]}")
```

Tasks for students

1. Run the `ml_basics.py` script and analyze the output
2. Modify the script to:
   - Use a different classifier (e.g., `RandomForestClassifier` or `LogisticRegression`)
   - Try different values for test_size (e.g., 0.2, 0.4) and observe the impact on performance
   - Create an additional visualization showing a different pair of features
3. Create a function that loads a model and makes predictions for new input data
4. Try with a different dataset from scikit-learn (e.g., `load_wine` or `load_breast_cancer`)
5. Research and implement one method for improving model performance (hint: parameter tuning, feature scaling, etc.)