Section 1: Data Loading and Overview

```
import time
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from sklearn.metrics import accuracy_score
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split

# Load the Iris dataset
print('Loading data...')
file_path = "IRIS_Dataset.csv"
iris_df = pd.read_csv(file_path)
print('Data is ready!')

# Display the first few rows of the dataset
iris_df.head()

# Overview of the dataset
iris_df.describe()
# Number of types of flowers
iris_df['species'].nunique()
```

Explanation:

- In this section, you load the Iris dataset, display the first few rows, and provide a statistical overview. You also check the number of unique flower species in the dataset.
- Ensure that the dataset is loaded correctly, and the basic statistics (mean, std, etc.) make sense for each feature. Verify that the number of unique species matches your expectations.

	sepal_length	sepal_width	petal_length	petal_width	species
0	6.3	2.7	1.4	0.3	I.Â setosa
1	6.5	2.7	6.1	1.9	I.Â virginica
2	5.9	3.6	1.1	0.3	I.Â setosa
3	5.4	2.6	1.1	1.4	I.Â versicolor
4	6.5	3.2	1.4	1.8	I.Â versicolor

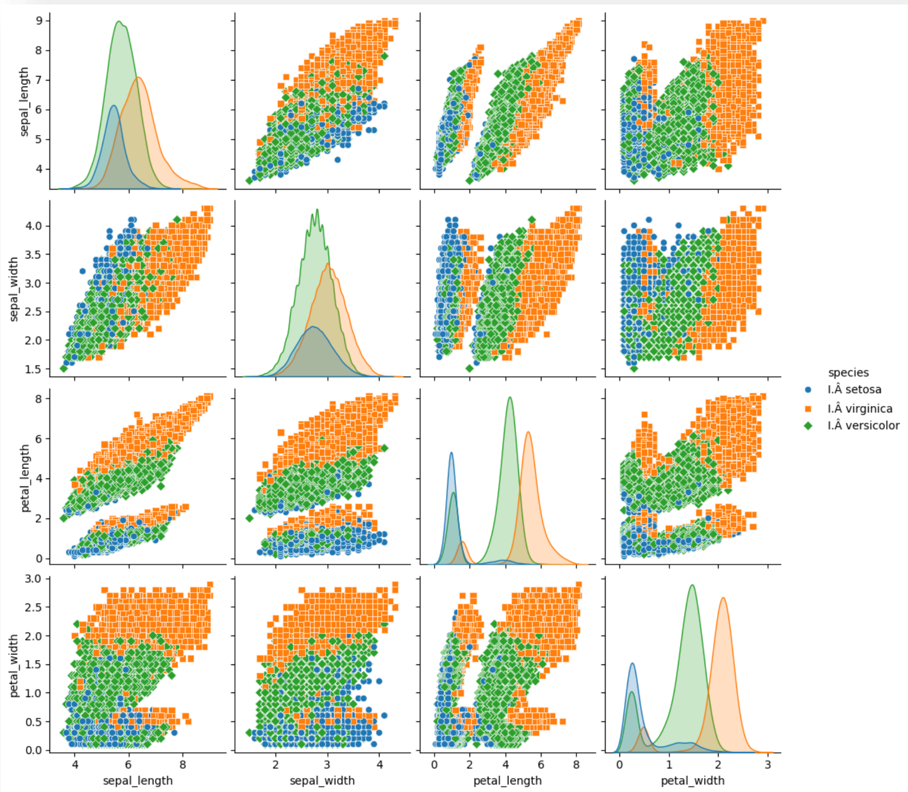
	sepal_length	sepal_width	petal_length	petal_width
count	49736.000000	49736.000000	49736.000000	49736.000000
mean	5.918906	2.848186	3.717969	1.380107
std	0.694449	0.360276	1.804229	0.687765
min	3.600000	1.500000	0.100000	0.100000
25%	5.400000	2.600000	1.500000	0.900000
50%	5.900000	2.800000	4.300000	1.500000
75%	6.300000	3.100000	5.100000	1.900000
max	9.000000	4.300000	8.100000	2.900000

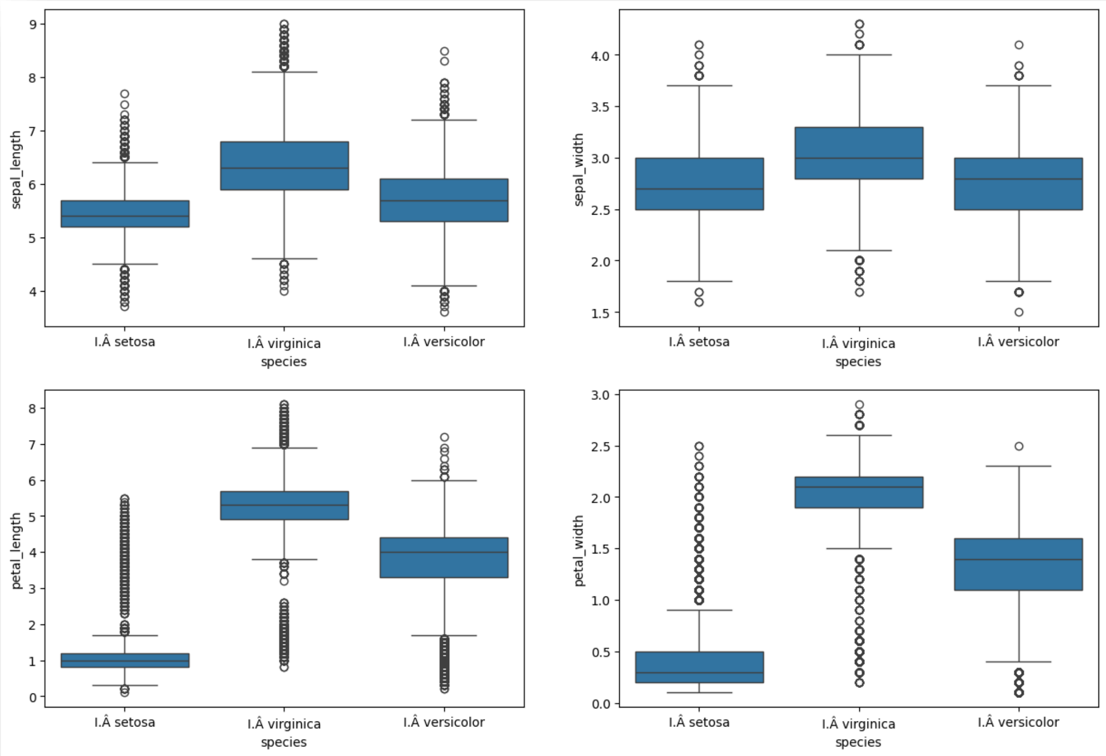
Section 2: Data Visualization

```
# Visualizations
# Pairplot for a quick overview
sns.pairplot(iris_df, hue='species', markers=["o", "s", "D"])

# Boxplot for each feature with respect to species
plt.figure(figsize=(15, 10))
plt.subplot(2, 2, 1)
sns.boxplot(x='species', y='sepal_length', data=iris_df)
plt.subplot(2, 2, 2)
sns.boxplot(x='species', y='sepal_width', data=iris_df)
plt.subplot(2, 2, 3)
sns.boxplot(x='species', y='petal_length', data=iris_df)
plt.subplot(2, 2, 4)
sns.boxplot(x='species', y='petal_width', data=iris_df)

# Show the plots
plt.show()
```

Explanation:

- This section generates pair plots and box plots for better visualization of the dataset.
- Ensure that the pair plot provides insights into the relationships between features, and the box plots show how each feature varies across different species.





Section 3: Data Preprocessing and Model Training (SDBP)

```
# Features (X) and target variable (y)
X = iris_df.drop('species', axis=1)
y = iris_df['species']

# Split the 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)

# Create the MLPClassifier with the SDBP solver
mlp_sdbp = MLPClassifier(solver='sgd', max_iter=1000, random_state=42)

# Training
# ...
# Print the final results
print("\nResults for Steepest Descent Backpropagation (SDBP):")
# ...
```

Explanation:

- This part preprocesses the data, splits it into training and testing sets, creates an MLP classifier with Steepest Descent Backpropagation (SDBP) solver, trains the model, and prints the results.
- Check that the training process completes without errors, and review the results, including the number of iterations, time taken, and accuracy.

```
Training started...
Epoch 1000/1000 - 100.00% complete - Elapsed time: 251.82 seconds
Results for Steepest Descent Backpropagation (SDBP):
Number of iterations: 1000
Time taken: 251.8737 seconds
Accuracy: 0.8821
```

Section 4: Model Training (LM)

```python
# ...
# Create the MLPClassifier with the LBFGS solver and verbose=True
mlp_lm = MLPClassifier(solver='lbfgs', max_iter=1000, random_state=42,
verbose=True)

# Training
mlp_lm.fit(X_train, y_train)

# Predictions
y_pred_lm = mlp_lm.predict(X_test)

# Accuracy
accuracy_lm = accuracy_score(y_test, y_pred_lm)

# Print the results
print("\nResults for Levenberg-Marquardt (LM):")
# ...
```

Explanation:

- This section trains another MLP classifier using the Levenberg-Marquardt (LM) solver and prints the results.
- Confirm that the training process with LM completes successfully and review the accuracy and the number of iterations.

```
Results for Levenberg-Marquardt (LM):
Number of iterations: 18
Accuracy: 0.9667
```

Section 5: TensorFlow Model Training (Powell-Beale-CG)

```
# ...
# Instantiate the model
# ...
# Train the model using Powell-Beale-CG
model = powell_beale_cg_optimization(model, X_train_tensor, y_train_tensor,
max_iter=200, learning_rate=0.01)

# Inference
# ...
# Accuracy
accuracy_powell_beale_cg_tf = accuracy_score(y_test_numeric,
predictions.numpy())

# Print the results
print("\nResults for Powell-Beale Conjugate Gradient (CG) using TensorFlow:")
# ...
```

Explanation:

- This section uses TensorFlow to implement a custom Powell-Beale Conjugate Gradient optimization algorithm and trains the model.
- Ensure that the training process completes, and review the accuracy. Note that the time taken is marked as "Not applicable" since it's not explicitly measured.

```
Progress: 100.00%
Results for Powell-Beale Conjugate Gradient (CG) using TensorFlow:
Time taken: Not applicable
Accuracy: 0.7104
```