```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler

# Load the dataset
url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.d
ata"
df = pd.read_csv(url, header=None)

print(df.head)

# Define feature and target variables
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

# Encode target labels
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=100)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
<bound method NDFrame.head of        0    1    2    3               4
0    5.1  3.5  1.4  0.2     Iris-setosa
1    4.9  3.0  1.4  0.2     Iris-setosa
2    4.7  3.2  1.3  0.2     Iris-setosa
3    4.6  3.1  1.5  0.2     Iris-setosa
4    5.0  3.6  1.4  0.2     Iris-setosa
..   ...  ...  ...  ...             ...
145  6.7  3.0  5.2  2.3  Iris-virginica
146  6.3  2.5  5.0  1.9  Iris-virginica
147  6.5  3.0  5.2  2.0  Iris-virginica
148  6.2  3.4  5.4  2.3  Iris-virginica
149  5.9  3.0  5.1  1.8  Iris-virginica

[150 rows x 5 columns]>
```

Conclusions:

I choose Iris dataset from the UCI repository. I made label encoding to convert categorical variables to numerical variables. I split the data into training and testing sets. I used 20% of the data for testing. I applied standart scaler to standart the features.

PART 2

```
!pip install scikeras

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from scikeras.wrappers import KerasClassifier

def create_mlp():
    model = Sequential()
    model.add(Dense(10, input_dim=4, activation='relu'))
    model.add(Dense(3, activation='softmax'))
    model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

# Wrap the Keras model for use in scikit-learn
mlp = KerasClassifier(model=create_mlp, epochs=50, batch_size=5,
verbose=0)

print(mlp)

KerasClassifier(
	model=<function create_mlp at 0x792907602ef0>
	build_fn=None
	warm_start=False
	random_state=None
	optimizer=rmsprop
	loss=None
	metrics=None
	batch_size=5
	validation_batch_size=None
	verbose=0
	callbacks=None
	validation_split=0.0
	shuffle=True
	run_eagerly=False
	epochs=50
	class_weight=None
)

from sklearn.ensemble import AdaBoostClassifier
```

```
# Initialize AdaBoost with the MLP classifier as the base estimator
ada_boost = AdaBoostClassifier(estimator=mlp, n_estimators=50,
learning_rate=1.0)


# Train AdaBoost
ada_boost.fit(X_train, y_train)

from sklearn.metrics import accuracy_score, classification_report

# Predict on the test set
y_pred = ada_boost.predict(X_test)

# Evaluate the performance
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred,
target_names=label_encoder.classes_)

print(f"Accuracy: {accuracy}")
print("Classification Report:\n", report)
```

```
Accuracy: 0.9666666666666667
Classification Report:
                  precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       0.86      1.00      0.92         6
 Iris-virginica       1.00      0.92      0.96        13

       accuracy                           0.97        30
      macro avg       0.95      0.97      0.96        30
   weighted avg       0.97      0.97      0.97        30
```

Conclusions:

create_mlp function defines the structure of the multi-layer perceptron. **Sequential** indicates a linear stack of layers. **Dense(10, input_dim=4, activation='relu')** adds a dense (fully connected) layer with 10 neurons, input dimension of 4 (since the Iris dataset has 4 features), and ReLU activation function. **Dense(3, activation='softmax')** adds an output layer with 3 neurons (since there are 3 classes in the Iris dataset) and softmax activation for multi-class classification. **compile** method specifies the optimizer (adam), loss function (sparse_categorical_crossentropy), and evaluation metric (accuracy).

Then, I used **KerasClassifier** to create a simple multi-layer perceptron (MLP) with one hidden layer.

Keras models can be easily wrapped for use in scikit-learn with the KerasClassifier wrapper from scikeras. This integration allows the Keras model to be used as a base estimator in scikit-learn's ensemble methods, like AdaBoost in this case.

Then I used this mlp as the base classifier for the **AdaBoost** ensemble and trained the model.

Finally I calculated the accuracy and report the performance with **classification_report**.

PART 3

```
def create_perceptron():
    model = Sequential()
    model.add(Dense(1, input_dim=4, activation='linear'))
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model

from sklearn.base import BaseEstimator, ClassifierMixin
from scikeras.wrappers import KerasRegressor
import numpy as np

class PerceptronTree(BaseEstimator, ClassifierMixin):
    def __init__(self, depth=1):
        self.depth = depth
        self.model = KerasRegressor(model=create_perceptron,
epochs=10, batch_size=5, verbose=0)
        self.left = None
        self.right = None
        self.is_leaf = True
        self.label = None

    def fit(self, X, y):
        if self.depth > 1 and len(np.unique(y)) > 1:
            self.model.fit(X, y)
            self.is_leaf = False
            predictions = self.model.predict(X)
            median = np.median(predictions)
            left_indices = predictions <= median
            right_indices = predictions > median
            self.left = PerceptronTree(depth=self.depth - 1)
            self.right = PerceptronTree(depth=self.depth - 1)
            self.left.fit(X[left_indices], y[left_indices])
            self.right.fit(X[right_indices], y[right_indices])
        else:
            self.is_leaf = True
            self.label = np.argmax(np.bincount(y))

    def predict(self, X):
        if self.is_leaf:
            return np.full(X.shape[0], self.label)
```

```
        else:
            predictions = self.model.predict(X)
            median = np.median(predictions)
            left_indices = predictions <= median
            right_indices = predictions > median
            y_pred = np.zeros(X.shape[0])
            y_pred[left_indices] = self.left.predict(X[left_indices])
            y_pred[right_indices] =
self.right.predict(X[right_indices])
            return y_pred

class PerceptronForest(BaseEstimator, ClassifierMixin):
    def __init__(self, n_estimators=10, max_depth=3):
        self.n_estimators = n_estimators
        self.max_depth = max_depth
        self.trees = [PerceptronTree(depth=max_depth) for _ in
range(n_estimators)]

    def fit(self, X, y):
        for tree in self.trees:
            indices = np.random.choice(X.shape[0], X.shape[0],
replace=True)
            tree.fit(X[indices], y[indices])

    def predict(self, X):
        predictions = np.zeros((self.n_estimators, X.shape[0]))
        for i, tree in enumerate(self.trees):
            predictions[i] = tree.predict(X)
        return np.round(np.mean(predictions, axis=0))

from sklearn.metrics import accuracy_score, classification_report
import tensorflow as tf

# Set random seed for determinism
tf.random.set_seed(1)

# Initialize and train the perceptron forest
perceptron_forest = PerceptronForest(n_estimators=10, max_depth=3)
perceptron_forest.fit(X_train, y_train)

# Predict on the test set
y_pred = perceptron_forest.predict(X_test)


# Evaluate the performance
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred,
target_names=label_encoder.classes_)

print(f"Accuracy: {accuracy}")
print("Classification Report:\n", report)
```

```
Accuracy: 0.8
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       0.50      1.00      0.67         6
 Iris-virginica       1.00      0.54      0.70        13

       accuracy                           0.80        30
      macro avg       0.83      0.85      0.79        30
   weighted avg       0.90      0.80      0.80        30
```

Conclusions:

I implemented a class called **PerceptronForest** which constructs a forest of decision trees.

I defined a class called **PerceptronTree** which represents *a decision tree where each node is a perceptron*. I used the **KerasRegressor** wrapper from scikeras to integrate the perceptron.

In the fit method of PerceptronTree, I trained a perceptron at each node to predict the target. Then, I split the data based on the predictions of the perceptron. The split was determined by the median of the perceptron's predictions. In the predict method of PerceptronTree, I used the perceptron's predictions to decide which branch to follow for each input sample.

Finally I calculated the accuracy and report the performance with classification_report.