Assignment: Practicing Ensemble Learning in Python

Objective:

Learn to implement various ensemble methods and understand their differences through practice on real datasets.

Step 0: Setup

Import them in your notebook:

```python
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns


from sklearn.model_selection import train_test_split,
cross_val_score

from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix


from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier,
AdaBoostClassifier, GradientBoostingClassifier,
VotingClassifier, BaggingClassifier

from sklearn.linear_model import LogisticRegression

from sklearn.svm import SVC
```

Step 1: Load Dataset

Choose one of the following datasets:

1. Heart Disease - Kaggle Link
2. Diabetes - Kaggle Link

```python
df = pd.read_csv("path_to_dataset.csv")

df.head()
```

Tasks:

- Check for missing values
- Explore the data (e.g., df.describe(), df.info())

Step 2: Preprocessing

- Split into features and target variable:

```python
X = df.drop('target_column', axis=1)

y = df['target_column']
```

Split into training and testing sets:

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 3: Practice Bagging & Pasting

1. Train a Decision Tree with Bagging:

```python
bagging_model = BaggingClassifier(
    base_estimator=DecisionTreeClassifier(),
    n_estimators=10,
    max_samples=0.8,  # 80% samples
    bootstrap=True,    # Bagging (with replacement)
    random_state=42
)
bagging_model.fit(X_train, y_train)
y_pred = bagging_model.predict(X_test)
print("Bagging Accuracy:", accuracy_score(y_test, y_pred))
```

2. Change bootstrap=False to practice Pasting. Compare accuracy.

Step 4: Practice Random Forest

```python
rf_model = RandomForestClassifier(n_estimators=50, random_state=42)

rf_model.fit(X_train, y_train)

y_pred = rf_model.predict(X_test)

print("Random Forest Accuracy:", accuracy_score(y_test, y_pred))
```

- Plot feature importance:

```python
importances = rf_model.feature_importances_

sns.barplot(x=importances, y=X.columns)

plt.show()
```

Step 5: Practice Boosting

1. AdaBoost:

```python
ada_model = AdaBoostClassifier(

    base_estimator=DecisionTreeClassifier(max_depth=1),

    n_estimators=50,

    learning_rate=1,

    random_state=42

)
```

```
ada_model.fit(X_train, y_train)

y_pred = ada_model.predict(X_test)

print("AdaBoost Accuracy:", accuracy_score(y_test, y_pred))
```

2. Gradient Boosting:

```
Python
gb_model = GradientBoostingClassifier(n_estimators=50, learning_rate=0.1,
random_state=42)

gb_model.fit(X_train, y_train)

y_pred = gb_model.predict(X_test)

print("Gradient Boosting Accuracy:", accuracy_score(y_test, y_pred))
```

Step 6: Practice Voting

```
Python
voting_model = VotingClassifier(
    estimators=[
        ('lr', LogisticRegression(max_iter=1000)),
        ('dt', DecisionTreeClassifier()),
        ('rf', RandomForestClassifier(n_estimators=50))
    ],
    voting='hard'  # try 'soft' as well
)
```

```
voting_model.fit(X_train, y_train)

y_pred = voting_model.predict(X_test)

print("Voting Accuracy:", accuracy_score(y_test, y_pred))
```

Step 7: Practice Stacking

Python

```
from sklearn.ensemble import StackingClassifier

stack_model = StackingClassifier(
    estimators=[
        ('lr', LogisticRegression(max_iter=1000)),
        ('rf', RandomForestClassifier(n_estimators=50)),
        ('dt', DecisionTreeClassifier())
    ],
    final_estimator=GradientBoostingClassifier(n_estimators=50),
)

stack_model.fit(X_train, y_train)

y_pred = stack_model.predict(X_test)

print("Stacking Accuracy:", accuracy_score(y_test, y_pred))
```

Step 8: Analysis & Report

1. Compare all models' accuracies.
2. Discuss:
    - Which model performed best?
    - Difference between Bagging and Boosting?
    - Advantages of Stacking?
3. Optional: Plot confusion matrices for each model.

Deliverables:

- Python notebook with all code, plots, and outputs.
- Short written summary comparing the models.