1. Part 1 DNN-XGboost:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.metrics import mean_squared_error, r2_score
import shap

# Load the raw dataset
data = pd.read_csv(r'C:\Users\Administrator\PycharmProjects\pythonProject\Data UHPC1230.csv')
X = data.iloc[:, :-1]    # Features
y = data.iloc[:, -1]    # Target variable (original values)

# Get the minimum and maximum values of the target variable
min_value = y.min()
max_value = y.max()

# Data normalization
X = (X - X.min()) / (X.max() - X.min())
y = (y - min_value) / (max_value - min_value)

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define denormalization function
def denormalize(normalized_data, min_value, max_value):
    """
    Denormalize normalized data to original units
    """
    return normalized_data * (max_value - min_value) + min_value

# Build DNN model
def build_dnn_model(input_dim):
    model = Sequential()
    model.add(Dense(128, input_dim=input_dim, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(32, activation='relu'))
```

```
    model.add(Dense(16, activation='relu'))
    model.add(Dense(1, activation='linear'))  # Regression problem, use linear activation
    model.compile(optimizer='adam', loss='mse')
    return model

# Create and train DNN model
dnn_model = build_dnn_model(X_train.shape[1])
dnn_model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)

# Extract intermediate layer outputs from DNN model as features
dnn_feature_extractor = Sequential(dnn_model.layers[:-2])  # Remove the last two layers
X_train_dnn_features = dnn_feature_extractor.predict(X_train)
X_test_dnn_features = dnn_feature_extractor.predict(X_test)

# Confirm the number of features and generate meaningful names
num_features = X_train_dnn_features.shape[1]
feature_names = [f'Layer3_Neuron{i + 1}' for i in range(num_features)]

# Build XGBoost model
xgb_model = XGBRegressor(n_estimators=100, random_state=42)
xgb_model.fit(X_train_dnn_features, y_train)

# 1. K-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = cross_val_score(xgb_model, X_train_dnn_features, y_train, cv=kf,
scoring='neg_mean_squared_error')
mse_scores = -cv_scores
print("MSE for each fold: ", mse_scores)
print("Average MSE: ", np.mean(mse_scores))

# Predict and evaluate the model
y_train_pred = xgb_model.predict(X_train_dnn_features)
y_test_pred = xgb_model.predict(X_test_dnn_features)

# Denormalize
y_train_original = denormalize(y_train, min_value, max_value)
y_test_original = denormalize(y_test, min_value, max_value)
y_train_pred_original = denormalize(y_train_pred, min_value, max_value)
y_test_pred_original = denormalize(y_test_pred, min_value, max_value)

# Add calculation of a20 index
def calculate_a20_index(y_actual, y_pred, tolerance=0.2):
    """
    Calculate the a20 index, which is the proportion of predicted values within ±tolerance of
```

```
the actual values.
    """
    relative_errors = np.abs((y_actual - y_pred) / y_actual)
    within_tolerance = (relative_errors <= tolerance).sum()
    a20 = (within_tolerance / len(y_actual)) * 100
    return a20

# Calculate a20 index
a20_train_original = calculate_a20_index(y_train_original, y_train_pred_original)
a20_test_original = calculate_a20_index(y_test_original, y_test_pred_original)

# Calculate MSE, RMSE, MAE, MAPE, and R² after denormalization
mse_train_original = mean_squared_error(y_train_original, y_train_pred_original)
mse_test_original = mean_squared_error(y_test_original, y_test_pred_original)
rmse_train_original = np.sqrt(mse_train_original)
rmse_test_original = np.sqrt(mse_test_original)
mae_train_original = np.mean(np.abs(y_train_original - y_train_pred_original))
mae_test_original = np.mean(np.abs(y_test_original - y_test_pred_original))
mape_train_original = np.mean(np.abs((y_train_original - y_train_pred_original) / y_train_original)) * 100
mape_test_original = np.mean(np.abs((y_test_original - y_test_pred_original) / y_test_original)) * 100
r2_original = r2_score(y_test_original, y_test_pred_original)

# Print results
print(f"Denormalized training set mean squared error (MSE, unit: MPa): {mse_train_original}")
print(f"Denormalized test set mean squared error (MSE, unit: MPa): {mse_test_original}")
print(f"Denormalized training set root mean squared error (RMSE, unit: MPa): {rmse_train_original}")
print(f"Denormalized test set root mean squared error (RMSE, unit: MPa): {rmse_test_original}")
print(f"Denormalized training set mean absolute error (MAE, unit: MPa): {mae_train_original}")
print(f"Denormalized test set mean absolute error (MAE, unit: MPa): {mae_test_original}")
print(f"Denormalized training set mean absolute percentage error (MAPE, unit: %): {mape_train_original}")
print(f"Denormalized test set mean absolute percentage error (MAPE, unit: %): {mape_test_original}")
print(f"Denormalized model R² score: {r2_original}")
print(f"Denormalized training set a20 index (proportion of data within ±20%, unit: %): {a20_train_original}")
print(f"Denormalized test set a20 index (proportion of data within ±20%, unit: %): {a20_test_original}")
```

```
# SHAP analysis
explainer = shap.TreeExplainer(xgb_model)
shap_values = explainer.shap_values(X_train_dnn_features)

# Plot SHAP value summary
shap.summary_plot(shap_values, X_train_dnn_features, feature_names=feature_names)

# Plot regression fit scatter plots for training and test sets
plt.figure(figsize=(14, 6))

# Training set fit plot
plt.subplot(1, 2, 1)
plt.scatter(y_train_original, y_train_pred_original, c='blue', marker='o', edgecolor='white', label='Training data')
plt.plot([min(y_train_original), max(y_train_original)], [min(y_train_original), max(y_train_original)], color='black', linestyle='--', lw=2)
plt.xlabel('Actual values (Training, MPa)')
plt.ylabel('Predicted values (Training, MPa)')
plt.title(f'Training Data: Actual vs Predicted (MPa)\na20 index: {a20_train_original:.2f}%')
plt.legend(loc='upper left')

# Test set fit plot
plt.subplot(1, 2, 2)
plt.scatter(y_test_original, y_test_pred_original, c='green', marker='s', edgecolor='white', label='Test data')
plt.plot([min(y_test_original), max(y_test_original)], [min(y_test_original), max(y_test_original)], color='black', linestyle='--', lw=2)
plt.xlabel('Actual values (Test, MPa)')
plt.ylabel('Predicted values (Test, MPa)')
plt.title(f'Test Data: Actual vs Predicted (MPa)\na20 index: {a20_test_original:.2f}%')
plt.legend(loc='upper left')

plt.tight_layout()
plt.show()p
```

2. Part 2: DNN-RF

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
```

```
import shap

# 加载数据
data = pd.read_csv(r'C:\Users\Administrator\PycharmProjects\pythonProject\Data UHPC1230.csv')
X = data.iloc[:, :-1]   # 特征
y = data.iloc[:, -1]   # 目标变量（原始值）

# 数据集原始目标变量的最小值和最大值
min_value = y.min()   # 获取目标列的最小值
max_value = y.max()   # 获取目标列的最大值

# 归一化数据
X = (X - X.min()) / (X.max() - X.min())
y = (y - min_value) / (max_value - min_value)

# 将数据集拆分为训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 构建 DNN 模型
def build_dnn_model(input_dim):
    model = Sequential()
    model.add(Dense(128, input_dim=input_dim, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(16, activation='relu'))
    model.add(Dense(1, activation='linear'))   # 回归问题，使用线性激活
    model.compile(optimizer='adam', loss='mse')
    return model

# 创建并训练 DNN 模型
dnn_model = build_dnn_model(X_train.shape[1])
dnn_model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)

# 提取 DNN 模型的中间层输出作为特征
dnn_feature_extractor = Sequential(dnn_model.layers[:-2])   # 去掉最后两层
X_train_dnn_features = dnn_feature_extractor.predict(X_train)
X_test_dnn_features = dnn_feature_extractor.predict(X_test)

# 确认特征数并生成有意义的名称
num_features = X_train_dnn_features.shape[1]
feature_names = [f'Layer3_Neuron{i + 1}' for i in range(num_features)]
```

```
# 构建随机森林模型
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train_dnn_features, y_train)

# 定义反归一化函数
def denormalize(normalized_data, min_value, max_value):
    """
    将归一化数据反归一化为原始单位
    """
    return normalized_data * (max_value - min_value) + min_value

# 将预测值和实际值反归一化为原始单位
y_train_original = denormalize(y_train, min_value, max_value)
y_test_original = denormalize(y_test, min_value, max_value)

y_train_pred = rf_model.predict(X_train_dnn_features)
y_test_pred = rf_model.predict(X_test_dnn_features)

y_train_pred_original = denormalize(y_train_pred, min_value, max_value)
y_test_pred_original = denormalize(y_test_pred, min_value, max_value)

# 计算反归一化后的 MSE、RMSE、MAE、MAPE 和 R²
mse_train_original = mean_squared_error(y_train_original, y_train_pred_original)
mse_test_original = mean_squared_error(y_test_original, y_test_pred_original)
rmse_train_original = np.sqrt(mse_train_original)
rmse_test_original = np.sqrt(mse_test_original)
mae_train_original = np.mean(np.abs(y_train_original - y_train_pred_original))
mae_test_original = np.mean(np.abs(y_test_original - y_test_pred_original))
mape_train_original = np.mean(np.abs((y_train_original - y_train_pred_original) / y_train_original)) * 100
mape_test_original = np.mean(np.abs((y_test_original - y_test_pred_original) / y_test_original)) * 100
r2_original = r2_score(y_test_original, y_test_pred_original)

# 添加 a20 index 的计算
def calculate_a20_index(y_actual, y_pred, tolerance=0.2):
    """
    计算 a20 index，即预测值与真实值的相对误差在 ±tolerance 范围内的比例。
    """
    relative_errors = np.abs((y_actual - y_pred) / y_actual)
    within_tolerance = (relative_errors <= tolerance).sum()
    a20 = (within_tolerance / len(y_actual)) * 100
    return a20
```

```
# 计算 a20 index
a20_train_original = calculate_a20_index(y_train_original, y_train_pred_original)
a20_test_original = calculate_a20_index(y_test_original, y_test_pred_original)

# 打印结果
print(f"反归一化后的训练集均方误差（MSE，单位：MPa）：{mse_train_original}")
print(f"反归一化后的测试集均方误差（MSE，单位：MPa）：{mse_test_original}")
print(f"反归一化后的训练集均方根误差（RMSE，单位：MPa）：{rmse_train_original}")
print(f"反归一化后的测试集均方根误差（RMSE，单位：MPa）：{rmse_test_original}")
print(f"反归一化后的训练集平均绝对误差（MAE，单位：MPa）：{mae_train_original}")
print(f"反归一化后的测试集平均绝对误差（MAE，单位：MPa）：{mae_test_original}")
print(f"反归一化后的训练集平均绝对百分比误差（MAPE，单位：%）：{mape_train_original}")
print(f"反归一化后的测试集平均绝对百分比误差（MAPE，单位：%）：{mape_test_original}")
print(f"反归一化后的模型 R²分数: {r2_original}")
print(f"反归一化后的训练集 a20 index（±20% 的数据比例，单位：%）：{a20_train_original}")
print(f"反归一化后的测试集 a20 index（±20% 的数据比例，单位：%）：{a20_test_original}")

# 更新性能指标总结表
performance_summary = pd.DataFrame({
    'Metric': ['MSE (Training)', 'MSE (Test)', 'RMSE (Training)', 'RMSE (Test)',
               'MAE (Training)', 'MAE (Test)', 'MAPE (Training)', 'MAPE (Test)',
               'R²', 'a20 index (Training)', 'a20 index (Test)'],
    'Value': [mse_train_original, mse_test_original,
              rmse_train_original, rmse_test_original,
              mae_train_original, mae_test_original,
              mape_train_original, mape_test_original,
              r2_original, a20_train_original, a20_test_original]
})

print(performance_summary)

# 可视化实际值和预测值与 a20 index 的对比
plt.figure(figsize=(14, 6))

# 训练集拟合图
plt.subplot(1, 2, 1)
plt.scatter(y_train_original, y_train_pred_original, c='blue', marker='o', edgecolor='white', label='Training data')
plt.plot([min(y_train_original), max(y_train_original)], [min(y_train_original), max(y_train_original)], color='black', linestyle='--', lw=2)
plt.xlabel('Actual values (Training, MPa)')
plt.ylabel('Predicted values (Training, MPa)')
plt.title(f'Training Data: Actual vs Predicted (MPa)\na20 index: {a20_train_original:.2f}%')
plt.legend(loc='upper left')
```

```
# 测试集拟合图
plt.subplot(1, 2, 2)
plt.scatter(y_test_original, y_test_pred_original, c='green', marker='s', edgecolor='white', label='Test data')
plt.plot([min(y_test_original), max(y_test_original)], [min(y_test_original), max(y_test_original)], color='black', linestyle='--', lw=2)
plt.xlabel('Actual values (Test, MPa)')
plt.ylabel('Predicted values (Test, MPa)')
plt.title(f'Test Data: Actual vs Predicted (MPa)\na20 index: {a20_test_original:.2f}%')
plt.legend(loc='upper left')

plt.tight_layout()
plt.show()

# SHAP 分析
explainer = shap.TreeExplainer(rf_model)
shap_values = explainer.shap_values(X_train_dnn_features)

# 绘制 SHAP 值的总结图
shap.summary_plot(shap_values, X_train_dnn_features, feature_names=feature_names)
```

3. Part 3: DNN-SVR

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.metrics import mean_squared_error, r2_score
import shap

# 加载原始数据集
data = pd.read_csv(r'C:\Users\Administrator\PycharmProjects\pythonProject\Data UHPC1230.csv')
X = data.iloc[:, :-1]   # 特征
y = data.iloc[:, -1]   # 目标变量（原始值）

# 获取目标变量的最小值和最大值
min_value = y.min()
max_value = y.max()

# 数据归一化
X = (X - X.min()) / (X.max() - X.min())
```

```
y = (y - min_value) / (max_value - min_value)

# 将数据集拆分为训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 定义反归一化函数
def denormalize(normalized_data, min_value, max_value):
    """
    将归一化数据反归一化为原始单位
    """
    return normalized_data * (max_value - min_value) + min_value

# 构建 DNN 模型
def build_dnn_model(input_dim):
    model = Sequential()
    model.add(Dense(128, input_dim=input_dim, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(16, activation='relu'))
    model.add(Dense(1, activation='linear'))  # 回归问题，使用线性激活
    model.compile(optimizer='adam', loss='mse')
    return model

# 创建并训练 DNN 模型
dnn_model = build_dnn_model(X_train.shape[1])
dnn_model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)

# 提取 DNN 模型的中间层输出作为特征
dnn_feature_extractor = Sequential(dnn_model.layers[:-2])  # 去掉最后两层
X_train_dnn_features = dnn_feature_extractor.predict(X_train)
X_test_dnn_features = dnn_feature_extractor.predict(X_test)

# 确认特征数并生成有意义的名称
num_features = X_train_dnn_features.shape[1]
feature_names = [f'Layer3_Neuron{i + 1}' for i in range(num_features)]

# 构建 SVR 模型
svr_model = SVR(kernel='rbf', C=1.0, epsilon=0.1)
svr_model.fit(X_train_dnn_features, y_train)

# 1. K 折交叉验证
kf = KFold(n_splits=5, shuffle=True, random_state=42)
```

```
cv_scores = cross_val_score(svr_model, X_train_dnn_features, y_train, cv=kf,
scoring='neg_mean_squared_error')
mse_scores = -cv_scores
print("每折的 MSE: ", mse_scores)
print("平均 MSE: ", np.mean(mse_scores))

# 预测并评估模型
y_train_pred = svr_model.predict(X_train_dnn_features)
y_test_pred = svr_model.predict(X_test_dnn_features)

# 反归一化
y_train_original = denormalize(y_train, min_value, max_value)
y_test_original = denormalize(y_test, min_value, max_value)
y_train_pred_original = denormalize(y_train_pred, min_value, max_value)
y_test_pred_original = denormalize(y_test_pred, min_value, max_value)

# 添加 a20 index 的计算
def calculate_a20_index(y_actual, y_pred, tolerance=0.2):
    """
    计算 a20 index，即预测值与真实值的相对误差在 ±tolerance 范围内的比例。
    """
    relative_errors = np.abs((y_actual - y_pred) / y_actual)
    within_tolerance = (relative_errors <= tolerance).sum()
    a20 = (within_tolerance / len(y_actual)) * 100
    return a20

# 计算 a20 index
a20_train_original = calculate_a20_index(y_train_original, y_train_pred_original)
a20_test_original = calculate_a20_index(y_test_original, y_test_pred_original)

# 计算反归一化后的 MSE、RMSE、MAE、MAPE 和 R²
mse_train_original = mean_squared_error(y_train_original, y_train_pred_original)
mse_test_original = mean_squared_error(y_test_original, y_test_pred_original)
rmse_train_original = np.sqrt(mse_train_original)
rmse_test_original = np.sqrt(mse_test_original)
mae_train_original = np.mean(np.abs(y_train_original - y_train_pred_original))
mae_test_original = np.mean(np.abs(y_test_original - y_test_pred_original))
mape_train_original = np.mean(np.abs((y_train_original - y_train_pred_original) /
y_train_original)) * 100
mape_test_original = np.mean(np.abs((y_test_original - y_test_pred_original) / y_test_original)) *
100
r2_original = r2_score(y_test_original, y_test_pred_original)

# 打印结果
```

```
print(f"反归一化后的训练集均方误差（MSE，单位：MPa）：{mse_train_original}")
print(f"反归一化后的测试集均方误差（MSE，单位：MPa）：{mse_test_original}")
print(f"反归一化后的训练集均方根误差（RMSE，单位：MPa）：{rmse_train_original}")
print(f"反归一化后的测试集均方根误差（RMSE，单位：MPa）：{rmse_test_original}")
print(f"反归一化后的训练集平均绝对误差（MAE，单位：MPa）：{mae_train_original}")
print(f"反归一化后的测试集平均绝对误差（MAE，单位：MPa）：{mae_test_original}")
print(f"反归一化后的训练集平均绝对百分比误差（MAPE，单位：%）：{mape_train_original}")
print(f"反归一化后的测试集平均绝对百分比误差（MAPE，单位：%）：{mape_test_original}")
print(f"反归一化后的模型 R²分数: {r2_original}")
print(f"反归一化后的训练集 a20 index（±20% 的数据比例，单位：%）：{a20_train_original}")
print(f"反归一化后的测试集 a20 index（±20% 的数据比例，单位：%）：{a20_test_original}")

# SHAP 分析
explainer = shap.KernelExplainer(svr_model.predict, X_train_dnn_features)
shap_values = explainer.shap_values(X_train_dnn_features)

# 绘制 SHAP 值的总结图
shap.summary_plot(shap_values, X_train_dnn_features, feature_names=feature_names)

# 绘制训练集和测试集的回归拟合散点图
plt.figure(figsize=(14, 6))

# 训练集拟合图
plt.subplot(1, 2, 1)
plt.scatter(y_train_original, y_train_pred_original, c='blue', marker='o', edgecolor='white', label='Training data')
plt.plot([min(y_train_original), max(y_train_original)], [min(y_train_original), max(y_train_original)], color='black', linestyle='--', lw=2)
plt.xlabel('Actual values (Training, MPa)')
plt.ylabel('Predicted values (Training, MPa)')
plt.title(f'Training Data: Actual vs Predicted (MPa)\na20 index: {a20_train_original:.2f}%')
plt.legend(loc='upper left')

# 测试集拟合图
plt.subplot(1, 2, 2)
plt.scatter(y_test_original, y_test_pred_original, c='green', marker='s', edgecolor='white', label='Test data')
plt.plot([min(y_test_original), max(y_test_original)], [min(y_test_original), max(y_test_original)], color='black', linestyle='--', lw=2)
plt.xlabel('Actual values (Test, MPa)')
plt.ylabel('Predicted values (Test, MPa)')
plt.title(f'Test Data: Actual vs Predicted (MPa)\na20 index: {a20_test_original:.2f}%')
plt.legend(loc='upper left')
```

```
plt.tight_layout()
plt.show()
```