```
import os
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,r2_score,mean_absolute_error
from sklearn.model_selection import GridSearchCV,cross_val_score,KFold
from sklearn.svm import SVR #支持向量机回归
from sklearn.ensemble import RandomForestRegressor,AdaBoostRegressor
from sklearn.preprocessing import StandardScaler,OneHotEncoder
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import accuracy_score,recall_score,f1_score,precision_score
from sklearn.tree import DecisionTreeRegressor
import xgboost as xgb
import lightgbm as lgb
import joblib
import scipy.stats as stats
from sklearn.linear_model import Lasso
import statsmodels.api as sm #  线性回归显著性检测
import shap # SHAP 解释

# sex    age       height weight bmi       tmi       trunk_fat_ratio       ln_tmi age_tmi       adtmi    age_adtmi
sex_adtmi    sex_age
features_list= [
                          ['sex','age','height','weight'],
                         ['sex','age','adtmi','age_adtmi'],
                         ['age','height','weight'],     # 2 --0
                         ['age','adtmi','age_adtmi'], # 3 -- 0
                         ['age','height','weight'],     # 4--1
                         ['age','adtmi','age_adtmi'], # 5--1
                         ['sex','age','adtmi','sex_age','sex_adtmi','age_adtmi']
                         ]
work_path = [
    r'Z:\\中国人人体成分数据集\\main_v13std',
    r'Z:\\中国人人体成分数据集\\main_v14std',
    r'Z:\\中国人人体成分数据集\\main_v15_LM_std',
    r'Z:\\中国人人体成分数据集\\main_v16_LM_std',
    r'Z:\\中国人人体成分数据集\\main_v15_LM_std',
    r'Z:\\中国人人体成分数据集\\main_v16_LM_std',
    r'Z:\\中国人人体成分数据集\\main_v17_std',
]
df =pd.read_excel(r'Z:\\中国人人体成分数据集\\bodycompsition_copy_wyt.xlsx')
```

```
print(df['躯干脂肪'].isnull().sum())
# 删除空值
df = df.dropna(subset=['躯干脂肪']) # 躯干肌肉 躯干骨质
df = df.dropna(subset=['躯干肌肉'])
df = df.dropna(subset=['躯干骨质'])
flag ='男女'
# for i in range(2,3):
# for i in range(0,len(features_list)):
for i in range(6, len(features_list)):
    print(f'{i} - {features_list[i] }-{ work_path[i]}')
    if i ==2:
        flag = '0'
    elif i==2:
        flag = '0'
    elif i==4:
        flag = '1'
    elif i==5:
        flag = '1'
    # 数据清洗 使用统计方法进行插值，例如线性插值
    #features = df[['性别','年龄','体重身高立方']]
    features = df[features_list[i]]
    # 筛选 6-17 岁 年龄的数据
    min_age = 7
    max_age = 12
    features_filtered = features[(features['age'] >= min_age) & (features['age'] <= max_age)]
    print(f"features_filtered:{type(features_filtered)}")
    # 筛选 6-17 岁 年龄的数据
    target = df['trunk_fat_ratio']   # 躯干脂肪百分比 2=躯干脂肪/(躯干肌肉+躯干骨质+躯干脂肪)*100
    target_filtered = target[(features['age'] >= min_age) & (features['age'] <= max_age )]

    with open(os.path.join(work_path[i],flag+'_results_main.txt'),'w') as f:

        # 划分训练集和测试集
        #X_train,X_test,y_train,y_test=train_test_split(features_filtered,target_filtered,
test_size=0.2,random_state=42)
        f.write("features_filtered-%s-%s: %s \n" % (min_age, max_age, features_filtered.describe()))
        f.write("target_filtered-%s-%s: %s \n" % (min_age, max_age, target_filtered.describe()))
        f.write("结果--年龄段：%s-%s \n" % (min_age, max_age))
        print("features_filtered-%s-%s: %s" % (min_age, max_age, features_filtered.describe()))
        print("target_filtered-%s-%s: %s" % (min_age, max_age, target_filtered.describe()))
        #target_filtered-6-18: count       7433.000000
        print("结果--年龄段：%s-%s" % (min_age, max_age))

        # 五折交叉验证方法进行数据集划分
```

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# 存储模型性能指标
performance_metrics_std = {
    'models_name': [],
    'MSE': [],
    'RMSE': [],
    'MAE': [],
    'MAPE': [],
    'R2': []

}


#定义模型列表
models = {
    'Linear Regression':LinearRegression(),
    'Random Forest':RandomForestRegressor(),
    'XGBoost': xgb.XGBRegressor(),
    'LightGBM':lgb.LGBMRegressor(),
    'KNN':KNeighborsRegressor(),
    'AdaBoots':AdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_depth=3)),#基模型是决策树
    # 'SVR':SVR()
    'lasso': Lasso()
}

# 参数网格
param_grids = {
    # 'Linear Regression': {'fit_intercept':[True,False],'normalize':[True,False]},
    'Random Forest': {
        'n_estimators': [100, 200],  # 森林中树林的数量
        'max_depth': [5, 10],  # 树的最大深度
    },
    'XGBoost':{
        'n_estimators': [100, 200],  # 森林中树林的数量
        'max_depth': [5, 10],  # 树的最大深度
        'learning_rate':[0.01, 0.1]
    },
    'LightGBM': {
        'n_estimators': [100, 200],  # 森林中树林的数量
        'max_depth': [5, 10],  # 树的最大深度
        'learning_rate':[0.01, 0.1]
    },
```

```
    'KNN': {
        'n_neighbors': [3, 5,7],   # 森林中树林的数量
    },
    'AdaBoots':{
        'n_estimators': [50, 100],   # 森林中树林的数量
    },
    'SVR': {
        'C':[0.1,1,10,100],
        'kernel':['linear','rbf'],
        'gamma':[0.01,0.1,1,10],
        'epsilon':[0.01, 0.1 ,0.5,1]
    },
    'lasoo':[0.1,1,10]
}

# 定义性能指标函数
def evaluate_model(y_test,y_pred):
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_test, y_pred)
    mape = (np.abs((y_test - y_pred) / y_test)).mean() * 100 #百分比
    r2 = r2_score(y_test, y_pred)
    return mse,rmse,mae,mape,r2

# 模型调优和性能评估
for model_name,model in models.items():
    print(model_name)

    # 执行五折交叉验证 features_filtered,target_filtered
    mse_scores = []
    rmse_scores = []
    mae_scores = []
    mape_scores = []
    r2_scores = []

    if model_name in ['Linear Regression']:
        coefficient_estinates = []
        standard_errors = []
        t_values = []
        p_values = []
        residual_standard_errors = []
        multiple_r_squareds = []
        adjusted_r_squareds = []
        f_statistics = []
```

```
        overall_p_values = []
        df_models =[]
        df_resids =[]


    for train_index, test_index in kf.split(target_filtered):
        X_train,X_test = features_filtered.iloc[train_index],features_filtered.iloc[test_index]
        y_train,y_test = target_filtered.iloc[train_index],target_filtered.iloc[test_index]

        # i = 2 ['age','height','weight'],      # 2 --0
        if i==2:
            X_train = X_train[df['sex'] == 0]
            X_test = X_test[df['sex'] == 0]
            y_train = y_train[df['sex'] == 0]
            y_test = y_test[df['sex'] == 0]
        # i= 3 ['age','adtmi','age_adtmi'], # 3 -- 0
        elif i==3:
            X_train = X_train[df['sex'] == 0]
            X_test = X_test[df['sex'] == 0]
            y_train = y_train[df['sex'] == 0]
            y_test = y_test[df['sex'] == 0]
        # i=4 ['age','height','weight'],      # 4--1
        elif i==4:
            X_train = X_train[df['sex'] == 1]
            X_test = X_test[df['sex'] == 1]
            y_train = y_train[df['sex'] == 1]
            y_test = y_test[df['sex'] == 1]
        # i=5 ['age','adtmi','age_adtmi'], # 5--1
        elif i==5:
            X_train = X_train[df['sex'] == 1]
            X_test = X_test[df['sex'] == 1]
            y_train = y_train[df['sex'] == 1]
            y_test = y_test[df['sex'] == 1]
        else:
            pass

        f.write(f"性别-{flag}-{model_name} - len(X_train): {len(X_train)} \n")
        f.write(f"性别-{flag}-{model_name} - len(X_test): {len(X_test)} \n")
        f.write(f"性别-{flag}-{model_name} - len(y_train): {len(y_train)} \n")
        f.write(f"性别-{flag}-{model_name} - len(y_test): {len(y_test)} \n")

        if model_name in param_grids:
            # 创建 GridSearchCV 对象，用于网格搜索
            grid_search = GridSearchCV(estimator=model, param_grid=param_grids[model_name],
```

```
                                                cv=kf, scoring='neg_mean_squared_error')
                        # 执行网格搜索
                        grid_search.fit(X_train, y_train)
                        # 使用最佳参数模型
                        model = grid_search.best_estimator_
                        # 输出最佳参数
                        print(f"性别-{flag}-{model_name} - best parameters found:
{grid_search.best_params_}")
                        f.write(f"性别-{flag}-{model_name} - best parameters found:
{grid_search.best_params_} \n")
                    else:
                        # 训练模型
                        model.fit(X_train, y_train)
                        # 获取模型参数
                        intercept_ = model.intercept_
                        coef_ = model.coef_
                        print(f'性别-{flag}-{model_name}-截距（intercept）：{intercept_},回归系数
（coefficient）:{coef_}')
                        f.write(f'性别-{flag}-{model_name}-截距（intercept）：{intercept_},回归系数
（coefficient）:{coef_} \n')


                    # 计算性能指标
                    y_pred = model.predict(X_test)
                    mse, rmse, mae, mape, r2 = evaluate_model(y_test, y_pred)
                    # 存储性能指标
                    mse_scores.append(mse)
                    rmse_scores.append(rmse)
                    mae_scores.append(mae)
                    mape_scores.append(mape)
                    r2_scores.append(r2)

                    if model_name in ['Linear Regression']:
                        # 系数显著性检测
                        X_train_sm = sm.add_constant(X_train)
                        est = sm.OLS(y_train,X_train_sm).fit()
                        coefficient_estinates.append(est.params)
                        standard_errors.append(est.bse)
                        t_values.append(est.tvalues)
                        p_values.append(est.pvalues)
                        residual_standard_errors.append(est.mse_resid)
                        multiple_r_squareds.append(est.rsquared)
                        adjusted_r_squareds.append(est.rsquared_adj)
                        f_statistics.append(est.fvalue)
```

```
                overall_p_values.append(est.f_pvalue)
                df_models.append(est.df_model)
                df_resids.append(est.df_resid)

        if model_name in ['Linear Regression']:
            # 将每一折的统计量转换成 DataFram
            df_coefficient_estinates=pd.DataFrame(coefficient_estinates)
            df_standard_errors=pd.DataFrame(standard_errors)
            df_tvalues=pd.DataFrame(t_values)
            df_pvalues=pd.DataFrame(p_values)

            # 计算汇总统计值
            sum_coefficient_estinates = df_coefficient_estinates.mean(axis=0)
            sum_standard_errors = df_standard_errors.mean(axis=0)
            sum_tvalues = df_tvalues.mean(axis=0)
            sum_pvalues = df_pvalues.mean(axis=0)


            results_summary = {
                'Coefficient Estimates':sum_coefficient_estinates,
                'Standard Errors': sum_standard_errors,
                't-values':sum_tvalues,
                'p-values':sum_pvalues,
                'Residual Standard Errors':np.mean(residual_standard_errors),
                'Multiple R Squareds':      np.mean(multiple_r_squareds),
                'Adjusted R Squareds':np.mean(adjusted_r_squareds),
                'F-statistics':      np.mean(f_statistics),
                'Overall P-values': np.mean(overall_p_values),
                'df_model': np.mean(df_models),
                'df_resid': np.mean(df_resids)

            }

            results_df = pd.DataFrame([results_summary])
            results_df.to_csv(os.path.join(work_path[i], f'性别
-{flag}-{model_name}-results_df.csv'),index=False)
            print(f'性别-{flag}-{model_name}-{results_df.to_string()}')
        print(f"****性别-{flag}-max_inde(r2)={r2_scores.index(max(r2_scores))}")
        f.write(f"\n ****性别-{flag}-max_inde(r2)={r2_scores.index(max(r2_scores))}\n")
        #计算每个指标的平均值
        mean_mse = round(np.mean(mse_scores),2)
        mean_rmse = round(np.mean(rmse_scores),2)
        mean_mae = round(np.mean(mae_scores),2)
        mean_mape =round(np.mean(mape_scores),2)
```

```
        mean_r2 = round(np.mean(r2_scores),4)

        # 计算每个指标的标准差
        std_mse = round(np.std(mse_scores),2)
        std_rmse = round(np.std(rmse_scores),2)
        std_mae = round(np.std(mae_scores),2)
        std_mape = round(np.std(mape_scores),2)
        std_r2 = round(np.std(r2_scores),2)

        # 估计 95%置信区间（使用标准差和状态分布近似）
        confidence_interval_mse = round(1.96*std_mse/np.sqrt(5),2)
        confidence_interval_rmse = round(1.96 * std_rmse / np.sqrt(5),2)
        confidence_interval_mae = round(1.96 * std_mae / np.sqrt(5),2)
        confidence_interval_mape = round(1.96 * std_mape / np.sqrt(5),2)
        confidence_interval_r2 = round(1.96 * std_r2 / np.sqrt(5),2)

        performance_metrics_std['models_name'].append(model_name)
        performance_metrics_std['MSE'].append([mean_mse,std_mse,confidence_interval_mse])
        performance_metrics_std['RMSE'].append([mean_rmse, std_rmse, confidence_interval_rmse])
        performance_metrics_std['MAE'].append([mean_mae, std_mae, confidence_interval_mae])
        performance_metrics_std['MAPE'].append([mean_mape, std_mape, confidence_interval_mape])
        performance_metrics_std['R2'].append([mean_r2, std_r2, confidence_interval_r2])
        # performance_metrics_std['coefficients'].append(pd.DataFrame(coefficients))

        if hasattr(model, 'feature_importances_'):
            feature_importances = model.feature_importances_
            importances = pd.DataFrame({
                'Feature': X_train.columns,
                'Importance':feature_importances
            }).sort_values('Importance',ascending=False)
            print(f"性别-{flag}-{model_name} feature importances:")
            print(importances)
            f.write(f"性别-{flag}-{model_name} feature importances:\n {importances} \n")

        # 保存模型
        model_path = os.path.join(work_path[i],f'{model_name.replace(" ", "_").lower()}_model.joblib')
        joblib.dump(model, model_path)
        print(f"性别-{flag}-{model_name} model saved as {model_path}")

    for metric,values in performance_metrics_std.items():
        print(f"性别-{flag}-{metric}:{values}")
        f.write(f"性别-{flag}-{metric}:{values}\n")
```