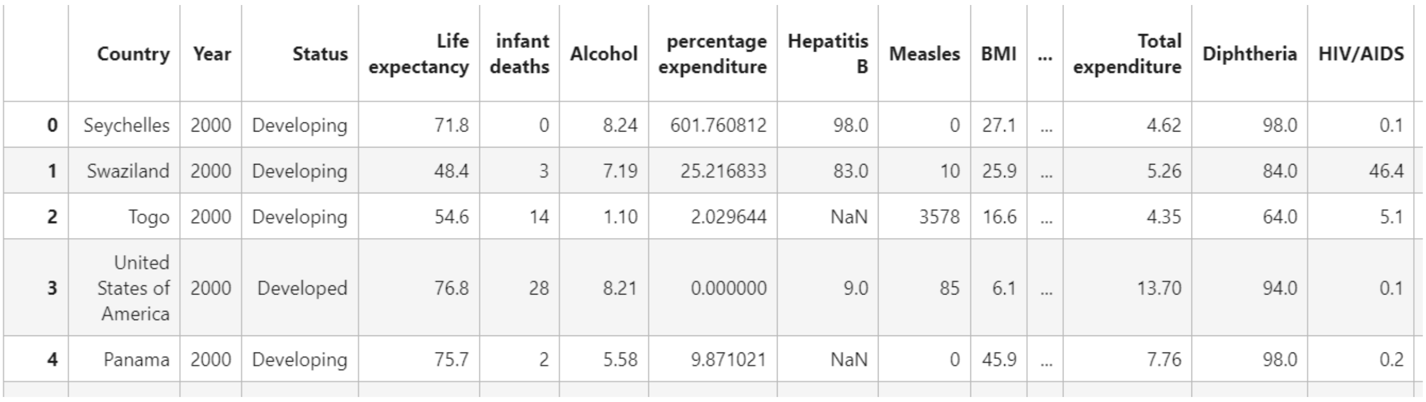
程序报告：成年人死亡率预测

学号：22451122
姓名：王中昊
专业：软件工程

数据分析

首先观察数据组成，country数据没有没有明显的意义，可以舍弃。status代表是否为发展中国家，可以用pd.factorize将其二值化，也可以直接舍弃，这里我们选择直接舍弃。

```
# Drop non-numeric columns
data = data.drop(["Country", "Status"], axis=1)
```



	Country	Year	Status	Life expectancy	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	...	Total expenditure	Diphtheria	HIV/AIDS
0	Seychelles	2000	Developing	71.8	0	8.24	601.760812	98.0	0	27.1	...	4.62	98.0	0.1
1	Swaziland	2000	Developing	48.4	3	7.19	25.216833	83.0	10	25.9	...	5.26	84.0	46.4
2	Togo	2000	Developing	54.6	14	1.10	2.029644	NaN	3578	16.6	...	4.35	64.0	5.1
3	United States of America	2000	Developed	76.8	28	8.21	0.000000	9.0	85	6.1	...	13.70	94.0	0.1
4	Panama	2000	Developing	75.7	2	5.58	9.871021	NaN	0	45.9	...	7.76	98.0	0.2

随后，可以计算各个特征与成人死亡率之间的皮尔逊相关系数

```
corr = train_data[column_name].corr()
corr.style.background_gradient(cmap='coolwarm')
```

	Year	Life expectancy	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Total expenditure	Diphtheria	HIV/AIDS	GDP	Population	thinness 1-19 years	thinness 5-9 years	Income composition of resources	Schooling	A Morta
Year	1.000000	0.173044	-0.045574	-0.059393	0.027351	0.119861	-0.089535	0.097642	-0.051892	0.099639	0.095594	0.146898	-0.144159	0.094351	0.012174	-0.049637	-0.055414	0.244558	0.219591	-0.085
Life expectancy	0.173044	1.000000	-0.200307	0.403930	0.385335	0.250026	-0.156621	0.563239	-0.227572	0.470405	0.227194	0.500060	-0.558550	0.466012	-0.028874	-0.481919	-0.476773	0.720200	0.756970	-0.695
infant deaths	-0.045574	-0.200307	1.000000	-0.117509	-0.088942	-0.233426	0.502258	-0.227356	0.996460	-0.168905	-0.127455	-0.186393	0.027106	-0.112767	0.559472	0.462870	0.466544	-0.148067	-0.197907	0.078
Alcohol	-0.059393	0.403930	-0.117509	1.000000	0.351033	0.076217	-0.045272	0.328996	-0.114326	0.223054	0.297324	0.225653	-0.053149	0.367267	-0.032890	-0.437330	-0.425520	0.451557	0.545457	-0.192
percentage expenditure	0.027351	0.385335	-0.088942	0.351033	1.000000	0.020959	-0.056564	0.243372	-0.091140	0.153229	0.196985	0.149584	-0.102281	0.908855	-0.030910	-0.257604	-0.258956	0.386001	0.399577	-0.244
Hepatitis B	0.119861	0.250026	-0.233426	0.076217	0.020959	1.000000	-0.122937	0.166267	-0.243389	0.470815	0.054130	0.610947	-0.112097	0.084776	-0.125556	-0.128664	-0.130000	0.196330	0.214588	-0.161
Measles	-0.089535	-0.156621	0.502258	-0.045272	-0.056564	-0.122937	1.000000	-0.176633	0.510569	-0.143172	-0.106506	-0.151227	0.033809	-0.078137	0.248298	0.224253	0.217004	-0.133857	-0.142477	0.024
BMI	0.097642	0.563239	-0.227356	0.328996	0.243372	0.166267	-0.176633	1.000000	-0.237654	0.287353	0.231999	0.304195	-0.243694	0.318139	-0.074910	-0.531130	-0.535272	0.507016	0.562572	-0.378
under-five deaths	-0.051892	-0.227572	0.996460	-0.114326	-0.091140	-0.243389	0.510569	-0.237654	1.000000	-0.187704	-0.128945	-0.208636	0.040501	-0.116458	0.545763	0.463884	0.467471	-0.167404	-0.214304	0.093
Polio	0.099639	0.470405	-0.168905	0.223054	0.153229	0.470815	-0.143172	0.287353	-0.187704	1.000000	0.133347	0.666200	-0.165835	0.219403	-0.036908	-0.237836	-0.237604	0.380321	0.419043	-0.289
Total expenditure	0.095594	0.227194	-0.127455	0.297324	0.196985	0.054130	-0.106506	0.231999	-0.128945	0.133347	1.000000	0.149584	-0.010628	0.171697	-0.078890	-0.273844	-0.282508	0.164861	0.256631	-0.118
Diphtheria	0.146898	0.500060	-0.186393	0.225653	0.149584	0.610947	-0.151227	0.304195	-0.208636	0.666200	0.149584	1.000000	-0.182804	0.210663	-0.034762	-0.253593	-0.245601	0.415212	0.439829	-0.304
HIV/AIDS	-0.144159	-0.558550	0.027106	-0.053149	-0.102281	-0.112097	0.033809	-0.243694	0.040501	-0.165835	-0.010628	-0.182804	1.000000	-0.141261	-0.025819	0.209907	0.214169	-0.249723	-0.228113	0.512
GDP	0.094351	0.466012	-0.112767	0.367267	0.908855	0.084776	-0.078137	0.318139	-0.116458	0.219403	0.171697	0.210663	-0.141261	1.000000	-0.036506	-0.294106	-0.299203	0.464682	0.457766	-0.296
Population	0.012174	-0.028874	0.559472	-0.032890	-0.030910	-0.125556	0.248298	-0.074910	0.545763	-0.036908	-0.078890	-0.034762	-0.025819	-0.036506	1.000000	0.262519	0.261089	-0.013043	-0.035931	-0.001
thinness 1-19 years	-0.049637	-0.481919	0.462870	-0.437330	-0.257604	-0.128664	0.224253	-0.531130	0.463884	-0.237836	-0.273844	-0.253593	0.209907	-0.294106	0.262519	1.000000	0.937984	-0.424126	-0.482056	0.295
thinness 5-9 years	-0.055414	-0.476773	0.466544	-0.425520	-0.258956	-0.130000	0.217004	-0.535272	0.467471	-0.237604	-0.282508	-0.245601	0.214169	-0.299203	0.261089	0.937984	1.000000	-0.414132	-0.471809	0.300
Income composition of resources	0.244558	0.720200	-0.148067	0.451557	0.386001	0.196330	-0.133857	0.507016	-0.167404	0.380321	0.164861	0.415212	-0.249723	0.464682	-0.013043	-0.424126	-0.414132	1.000000	0.796696	-0.452
Schooling	0.219591	0.756970	-0.197907	0.545457	0.399577	0.214588	-0.142477	0.562572	-0.214304	0.419043	0.256631	0.439829	-0.228113	0.457766	-0.035931	-0.482056	-0.471809	0.796696	1.000000	-0.462
Adult Mortality	-0.085130	-0.695485	0.078613	-0.192501	-0.244770	-0.161392	0.024766	-0.378400	0.093789	-0.289902	-0.118360	-0.304850	0.512732	-0.296923	-0.001169	0.295657	0.300346	-0.452480	-0.462488	1.000

从图中发现，life expectancy和HIV/AIDS两个特征与成年人死亡率最为相关，这也是一个符合直觉的结论。

数据清洗

对数据项中的空值（也就是np.nan），使用对应列的均值来进行填充。随后，使用RobustScaler对特征标准化，这个缩放器对异常值具有鲁棒性，缩放器移除中位数，并根据四分位数范围（默认为IQR：四分位距）对数据进行缩放。IQR 是第一四分位数和第三四分位数之间的范围。以下是数据清洗函数preprocess_data的代码。

```python
def preprocess_data(data, imputer=None, scaler=None):
    # Drop non-numeric columns
    data = data.drop(["Country", "Status"], axis=1)
    # Impute missing values if imputer is not provided
    if imputer is None:
        imputer = SimpleImputer(strategy='mean', missing_values=np.nan)
        imputer = imputer.fit(data[COLUMN_NAMES[:-1]])
    data[COLUMN_NAMES[:-1]] = imputer.transform(data[COLUMN_NAMES[:-1]])
    # Scale data if scaler is not provided
    if scaler is None:
        scaler = RobustScaler()
        scaler = scaler.fit(data)
    data_norm = pd.DataFrame(scaler.transform(data), columns=data.columns)
    data_norm = data_norm.drop(['Year'], axis=1)
    return data_norm, imputer, scaler
```

我使用IsolationForest来判定离群值，在训练过程中直接丢弃离群的数据点。

```python
def detect_and_remove_outliers(data, label):
    # Initialize the IsolationForest outlier detector
    outlier_detector = IsolationForest(contamination=0.1, random_state=42)
    # Fit the detector to the data
    outlier_detector.fit(data)
    # Predict outliers in the data
    outliers = outlier_detector.predict(data)
    # Get indices of non-outliers
    removed_indices = np.where(outliers == 1)[0]
    # Keep only non-outliers in the data and label
    data = data[outliers == 1]
    label = label[outliers == 1]
    return data, label, removed_indices
```

模型训练

以下是主训练函数代码，我选取了5个模型，分别为随机森林、AdaBoost、决策树、岭回归、支持向量回归。在训练过程中，使用10折交叉验证，一共进行$5 \times 10 = 50$次实验。最终保存下来的best_model为所有实验中泛化能力最强的模型。

```
def main():
    # Load training data
    train_data = pd.read_csv(TRAIN_DATA_PATH)

    # Initialize KFold cross-validation
    kf = KFold(n_splits=10, shuffle=True, random_state=42)

    best_model = None
    best_r2 = -np.inf
    best_imputer = None
    best_scaler = None
    best_model_name = None

    # List of models to evaluate
    model_list = ["RandomForestRegressor", "AdaBoostRegressor", "DecisionTreeRegressor", "Ridge"

    # Iterate over each model
    for model_name in model_list:
        for train_index, test_index in kf.split(train_data):
            # Split data into training and testing folds
            train_fold = train_data.iloc[train_index]
            test_fold = train_data.iloc[test_index]

            # Separate target variable
            train_y = train_fold['Adult Mortality'].values
            train_fold = train_fold.drop(["Adult Mortality"], axis=1)

            # Preprocess training data
            train_fold_norm, imputer, scaler = preprocess_data(train_fold, imputer=None, scaler=
            # train_fold_norm, train_y, rm_idx = detect_and_remove_outliers(train_fold_norm, tra
            train_x = train_fold_norm.values

            # Fit the model
            model = model_fit(model_name, train_x, train_y)

            # Separate target variable for testing data
            test_y = test_fold['Adult Mortality'].values
            test_fold = test_fold.drop(["Adult Mortality"], axis=1)
            test_fold_norm, _, _ = preprocess_data(test_fold, imputer=imputer, scaler=scaler)
            # test_fold_norm, test_y, rm_idx = detect_and_remove_outliers(test_fold_norm, test_y
            # Make predictions
            y_pred = predict(model, test_fold, imputer, scaler)
            # y_pred = y_pred[rm_idx]

            # Calculate R2 score for testing data
            r2 = r2_score(test_y, y_pred)
```

```
                # Calculate R2 score for training data
                train_pred = model.predict(train_x)
                train_r2 = r2_score(train_y, train_pred)
                print(f"Model name: {model_name}, Fold Train R2: {train_r2}, Test R2: {r2}")

                # Update the best model if current model is better
                if r2 > best_r2:
                    best_r2 = r2
                    best_model = model
                    best_model_name = model_name
                    best_imputer = imputer
                    best_scaler = scaler
                    print(f"Best model updated: {model_name}, R2: {best_r2}")

        # Save the best model, imputer, and scaler
        joblib.dump(best_model, MODEL_FILENAME)
        joblib.dump(best_imputer, IMPUTER_FILENAME)
        joblib.dump(best_scaler, SCALER_FILENAME)
        print(f'Best model: {best_model_name}, R2: {best_r2}')

    if __name__ == "__main__":
        main()
```

在每次拟合过程中，我提前定义超参数搜索区域，使用网格搜索来优化超参数。对于没有定义搜索区域的模型，直接使用默认超参数。

```python
def model_fit(model_name, train_x, train_y):
    # Define parameter grids for each model
    param_grids = {
        'MLPRegressor': {
            'hidden_layer_sizes': [(100, 100)],
            'activation': ['relu'],
            'max_iter': [1000],
            'solver': ['adam'],
        },
        'RandomForestRegressor': {
            'n_estimators': [100],
            'max_depth': [5, 10],
            'min_samples_split': [2, 5],
            'min_samples_leaf': [1, 2]
        },
        'AdaBoostRegressor': {
            'n_estimators': [50, 100],
            'learning_rate': [0.01, 0.1, 1]
        }
    }

    # Initialize the regressor based on the model name
    if model_name in param_grids:
        regressor = eval(model_name)()
        param_grid = param_grids[model_name]
        gs = GridSearchCV(regressor, param_grid, cv=5, scoring='r2', n_jobs=1)
        gs.fit(train_x, train_y)
        regressor = gs.best_estimator_
    else:
        regressor = eval(model_name)()
        regressor.fit(train_x, train_y)

    # Fit the model
    return regressor
```

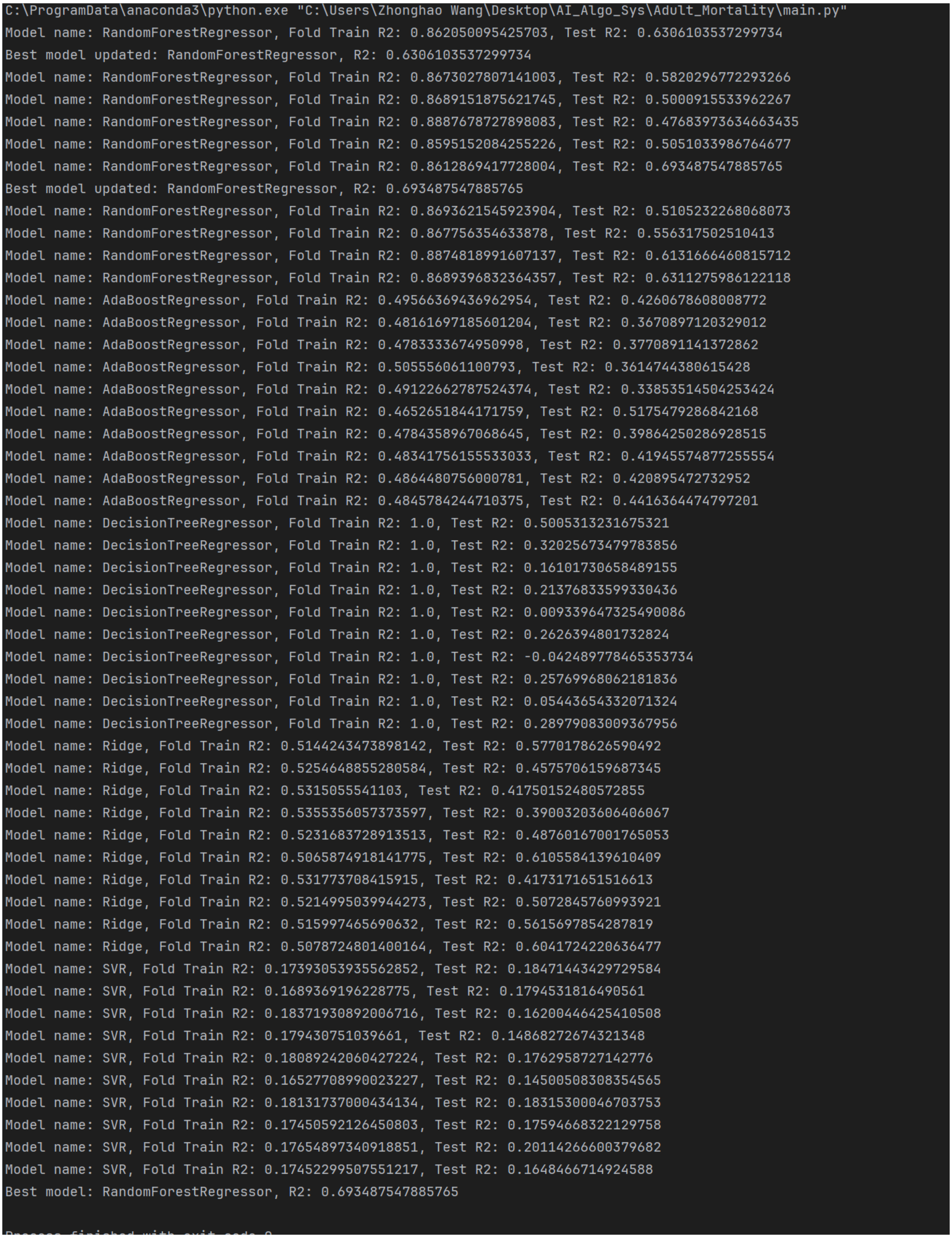
模型预测函数用于给出测试集预测结果，使用与训练集一样的超参数处理方法。

```python
def predict(model, test_data, imputer, scaler):
    # Preprocess the test data
    test_data_norm, _, _ = preprocess_data(test_data, imputer=imputer, scaler=scaler)
    test_x = test_data_norm.values

    # Make predictions
    predictions = model.predict(test_x)
    return predictions
```

测试结果

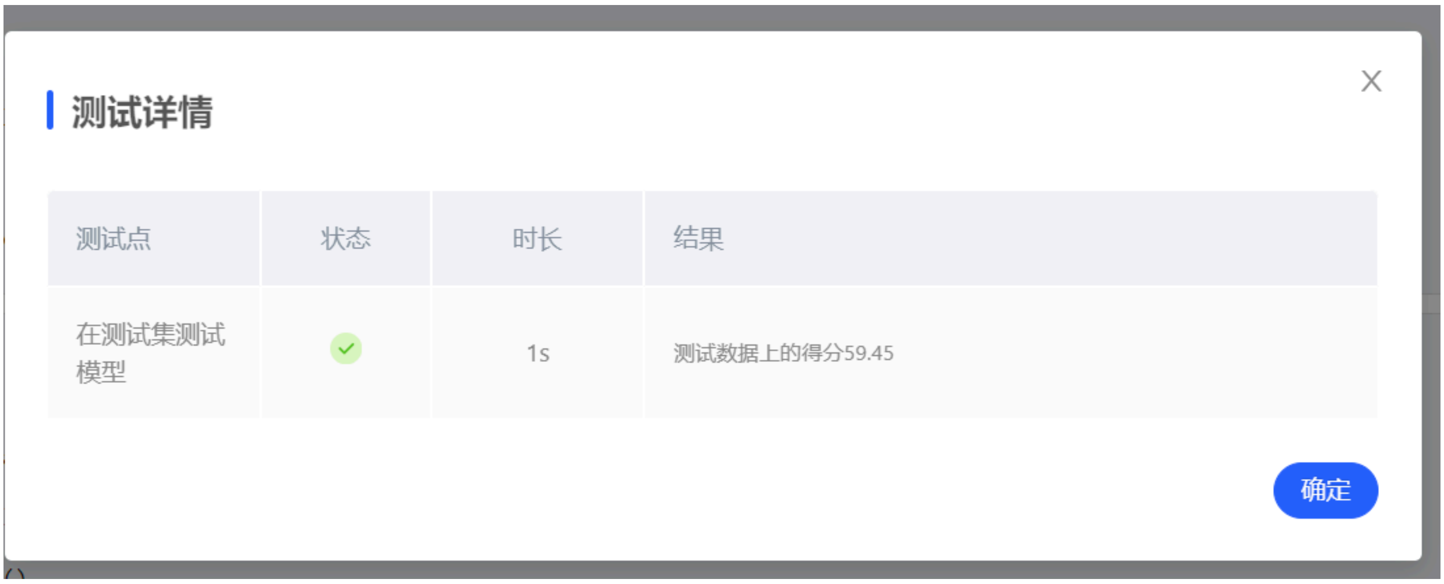
以下是程序的运行样例，在运行过程中，最佳模型会依据测试精确度不断更新。最终得到的最佳模型RandomForest在测试数据中的R2可以达到0.69。



```
C:\ProgramData\anaconda3\python.exe "C:\Users\Zhonghao Wang\Desktop\AI_Algo_Sys\Adult_Mortality\main.py"
Model name: RandomForestRegressor, Fold Train R2: 0.862050095425703, Test R2: 0.6306103537299734
Best model updated: RandomForestRegressor, R2: 0.6306103537299734
Model name: RandomForestRegressor, Fold Train R2: 0.8673027807141003, Test R2: 0.5820296772293266
Model name: RandomForestRegressor, Fold Train R2: 0.8689151875621745, Test R2: 0.5000915533962267
Model name: RandomForestRegressor, Fold Train R2: 0.8887678727898083, Test R2: 0.47683973634663435
Model name: RandomForestRegressor, Fold Train R2: 0.8595152084255226, Test R2: 0.5051033986764677
Model name: RandomForestRegressor, Fold Train R2: 0.8612869417728004, Test R2: 0.693487547885765
Best model updated: RandomForestRegressor, R2: 0.693487547885765
Model name: RandomForestRegressor, Fold Train R2: 0.8693621545923904, Test R2: 0.5105232268068073
Model name: RandomForestRegressor, Fold Train R2: 0.867756354633878, Test R2: 0.556317502510413
Model name: RandomForestRegressor, Fold Train R2: 0.8874818991607137, Test R2: 0.6131666460815712
Model name: RandomForestRegressor, Fold Train R2: 0.8689396832364357, Test R2: 0.6311275986122118
Model name: AdaBoostRegressor, Fold Train R2: 0.49566369436962954, Test R2: 0.4260678608008772
Model name: AdaBoostRegressor, Fold Train R2: 0.48161697185601204, Test R2: 0.3670897120329012
Model name: AdaBoostRegressor, Fold Train R2: 0.4783333674950998, Test R2: 0.3770891141372862
Model name: AdaBoostRegressor, Fold Train R2: 0.505556061100793, Test R2: 0.3614744380615428
Model name: AdaBoostRegressor, Fold Train R2: 0.49122662787524374, Test R2: 0.33853514504253424
Model name: AdaBoostRegressor, Fold Train R2: 0.4652651844171759, Test R2: 0.5175479286842168
Model name: AdaBoostRegressor, Fold Train R2: 0.4784358967068645, Test R2: 0.39864250286928515
Model name: AdaBoostRegressor, Fold Train R2: 0.48341756155533033, Test R2: 0.41945574877255554
Model name: AdaBoostRegressor, Fold Train R2: 0.4864480756000781, Test R2: 0.420895472732952
Model name: AdaBoostRegressor, Fold Train R2: 0.4845784244710375, Test R2: 0.4416364474797201
Model name: DecisionTreeRegressor, Fold Train R2: 1.0, Test R2: 0.5005313231675321
Model name: DecisionTreeRegressor, Fold Train R2: 1.0, Test R2: 0.32025673479783856
Model name: DecisionTreeRegressor, Fold Train R2: 1.0, Test R2: 0.16101730658489155
Model name: DecisionTreeRegressor, Fold Train R2: 1.0, Test R2: 0.21376833599330436
Model name: DecisionTreeRegressor, Fold Train R2: 1.0, Test R2: 0.009339647325490086
Model name: DecisionTreeRegressor, Fold Train R2: 1.0, Test R2: 0.2626394801732824
Model name: DecisionTreeRegressor, Fold Train R2: 1.0, Test R2: -0.042489778465353734
Model name: DecisionTreeRegressor, Fold Train R2: 1.0, Test R2: 0.25769968062181836
Model name: DecisionTreeRegressor, Fold Train R2: 1.0, Test R2: 0.05443654332071324
Model name: DecisionTreeRegressor, Fold Train R2: 1.0, Test R2: 0.28979083009367956
Model name: Ridge, Fold Train R2: 0.5144243473898142, Test R2: 0.5770178626590492
Model name: Ridge, Fold Train R2: 0.5254648855280584, Test R2: 0.4575706159687345
Model name: Ridge, Fold Train R2: 0.5315055541103, Test R2: 0.41750152480572855
Model name: Ridge, Fold Train R2: 0.5355356057373597, Test R2: 0.39003203606406067
Model name: Ridge, Fold Train R2: 0.5231683728913513, Test R2: 0.48760167001765053
Model name: Ridge, Fold Train R2: 0.5065874918141775, Test R2: 0.6105584139610409
Model name: Ridge, Fold Train R2: 0.531773708415915, Test R2: 0.4173171651516613
Model name: Ridge, Fold Train R2: 0.5214995039944273, Test R2: 0.5072845760993921
Model name: Ridge, Fold Train R2: 0.515997465690632, Test R2: 0.5615697854287819
Model name: Ridge, Fold Train R2: 0.5078724801400164, Test R2: 0.6041724220636477
Model name: SVR, Fold Train R2: 0.17393053935562852, Test R2: 0.18471443429729584
Model name: SVR, Fold Train R2: 0.1689369196228775, Test R2: 0.1794531816490561
Model name: SVR, Fold Train R2: 0.18371930892006716, Test R2: 0.16200446425410508
Model name: SVR, Fold Train R2: 0.179430751039661, Test R2: 0.14868272674321348
Model name: SVR, Fold Train R2: 0.18089242060427224, Test R2: 0.1762958727142776
Model name: SVR, Fold Train R2: 0.16527708990023227, Test R2: 0.14500508308354565
Model name: SVR, Fold Train R2: 0.18131737000434134, Test R2: 0.18315300046703753
Model name: SVR, Fold Train R2: 0.17450592126450803, Test R2: 0.17594668322129758
Model name: SVR, Fold Train R2: 0.17654897340918851, Test R2: 0.20114266600379682
Model name: SVR, Fold Train R2: 0.17452299507551217, Test R2: 0.1648466714924588
Best model: RandomForestRegressor, R2: 0.693487547885765

Process finished with exit code 0
```

该模型在molab系统中的测试集R2为0.59



测试详情

测试点	状态	时长	结果
在测试集测试模型	✓	1s	测试数据上的得分59.45