深 圳 大 学 实 验 报 告

课程名称：机器学习

实验项目名称：Machine Learning Task 1

学院：电子与信息工程学院

专业：电子信息工程

指导教师：欧阳乐

报告人：余韦藩 学号：2020285102

班级：文华班

实验时间：2022.4.1 ——2022.4.13

实验报告提交时间：2022.4.13

教务处制

Aim of Experiment:

(1) Write a linear regression algorithm by yourself.
(2) Learn how a built a whole system
(3) Learn how to explore exist datasets and analysis the predicted result.

Experiment Content:

(1) Use Boston House Dataset (506 samples and 13 feature variables), predict the value of prices of the house using the given features.
(2) Split input data into training and testing sets (the testing set includes 10% of the samples).
(3) The output of the algorithm should include the model learned from the whole training set, the average mean squared error (MSE) on training set via 10-fold cross validation. For ridge regression model, determine the value of hyper-parameter via 10-fold cross validation.

Experiment Process:

Since the task is related to data mining, I will divide the process into following four sub-processes including making data exploration, making data preprocessing, constructing prediction model and constructing evaluation system. The topological graph for the above is shown in the figure 1. In the following part, I will elaborate these four sub-processes in detail respectively. At first, I will introduce the model in corresponding part. After understanding the function of the models I build, I will discuss the whole experiment process.

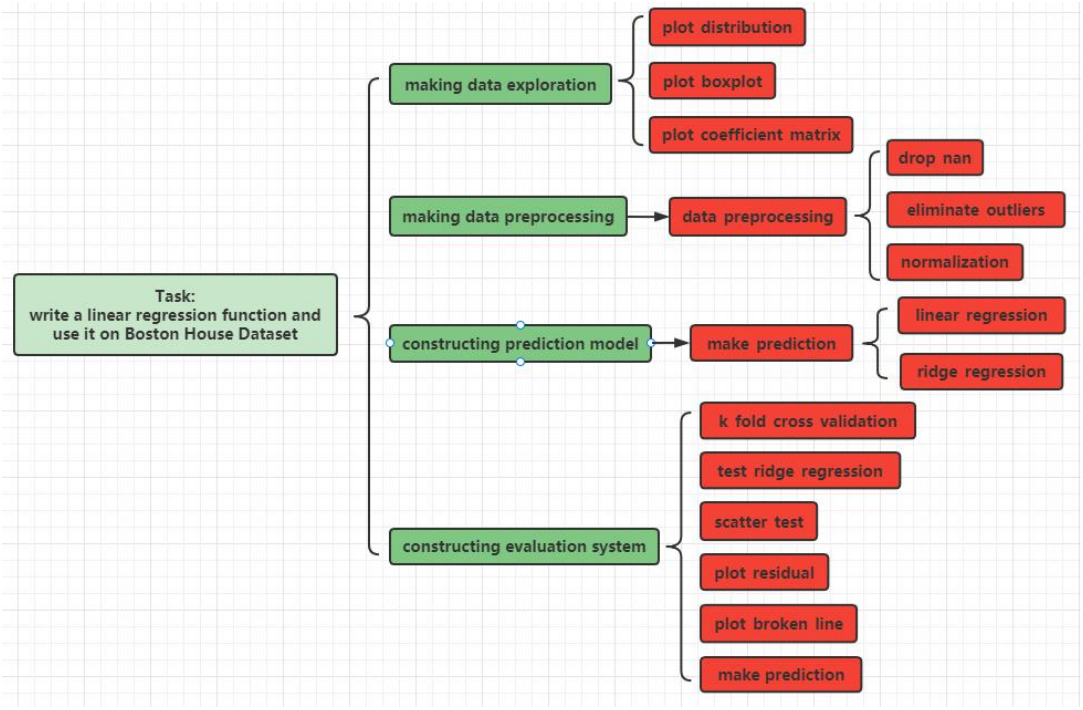

Figure1: The topological graph for splitting the task and four sub-processes
(The red box presents the function I created)

A. Make data exploration

Data exploration includes three models: Plot distribution, Plot coefficient matrix and Plot boxplot. The topological graph is shown in the figure 2.

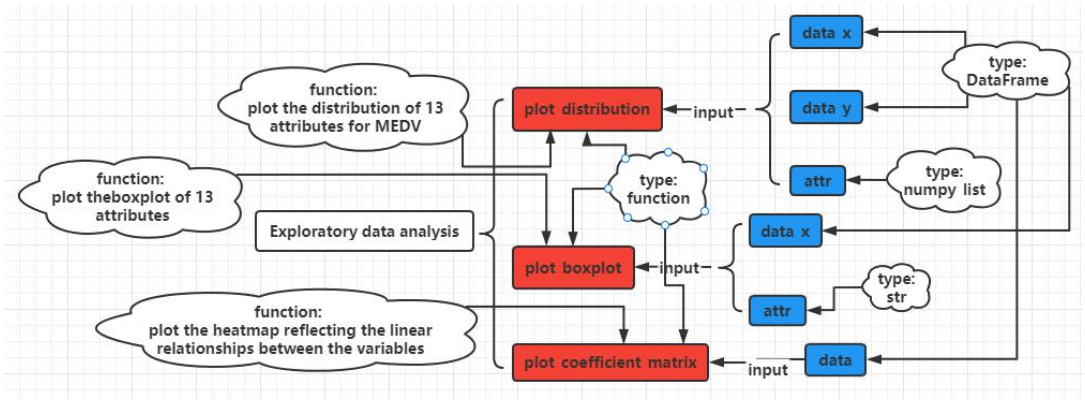

Figure 2: The topological graph for data exploration

Model one: Plot distribution

This model is built to plot the distribution of 13 attributes for price. In addition, the input data is data_x(variables), data_y(labels), and attr(feature names). The main function I use is matplotlib.pyplot.scatter( ). The codes are shown in the figure 3.

```
# plot the distribution of 13 attributes for MEDV
def plot_distribution(data_x, data_y, attr):
    for i in range(13):
        plt.subplot(3, 5, i + 1)
        plt.scatter(data_x.iloc[:, i], data_y, s=2)
        plt.title('The distribution of {} for MEDV'.format(attr[i]))
    return
```

Figure 3: The codes of Plot distribution model

Model two: Plot coefficient matrix

This model is built to plot the heatmap reflecting the linear relationship between the variables. In addition, the input data is the complete datasets. The main function I use is DataFrame.corr( ). The codes are shown in the figure 4.

```
# plot the heatmap reflecting the linear relationships between the variables
def plot_coefficient_matrix(data):
    corr = data.corr()
    sns.heatmap(corr, annot=True, cmap='YlGnBu')
    return
```

Figure 4: The codes of Plot coefficient matrix

Model three: Plot boxplot

This model is built to plot the boxplot of 13 attributes. The input data is data_x (13 attributes samples) and attr(the feature name). The main function I use is DataFrame.boxplot( ). The codes are shown in the figure 5.

```
# plot the boxplot of 13 attributes
def plot_boxplot(data_x, attr):
    data_x = pd.DataFrame(data=data_x, columns=attr)
    data_x.boxplot()
    plt.title('The boxplot of 13 attributes')
```

Figure 5: The codes of Plot boxplot

B. Make data preprocessing

Data exploration contain a module: data_preprocessing. And the this module is related to three sub-models which includes Model_drop_nan, Model_eliminate_outliers and Model_normalization. The topological graph is shown in the figure 6.

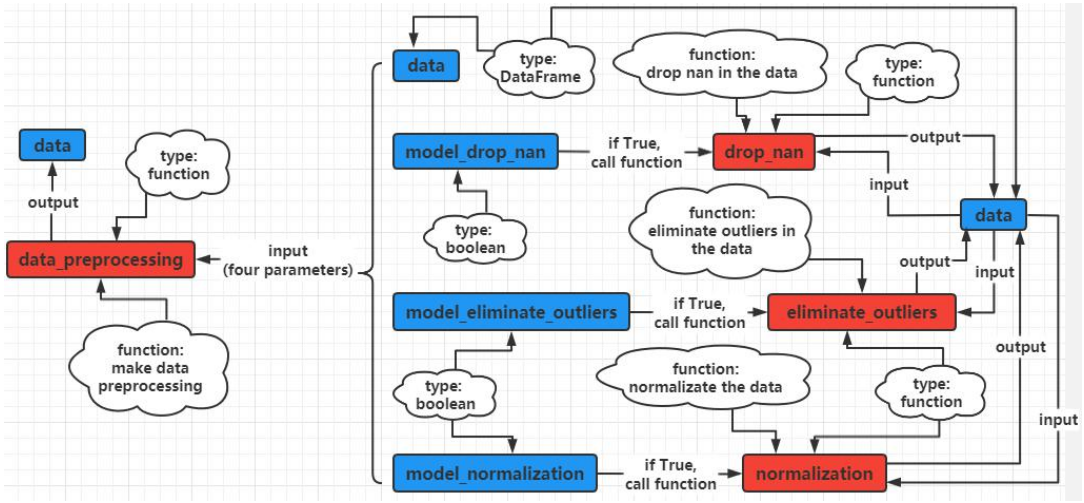

Figure 6: The topological graph for data preprocessing

Module: data_preprocessing

This module controls whether to perform the three sub-models. When the corresponding parameter is True, it implies that we perform the corresponding performance for data. The codes are shown in the figure 7.

```
# make a data preprocessing model (combine the drop_nan, eliminate_outliers, normalization model)
def data_preprocessing(data, mode_drop_nan=True, mode_eliminate_outliers=True, mode_normalization=False):
    if mode_drop_nan is True:
        data = drop_nan(data)
    if mode_eliminate_outliers is True:
        data = eliminate_outliers(data)
    if mode_normalization is True:
        data = normalization(data)
    return data
```

Figure 7: The codes of data_preprocessing module

Model one: Model_drop_nan

This model is built to drop the NaN value in the datasets. The input data is the datasets we want to deal with and the output is the processed datasets. The main function I use is DataFrame.dropna( ). The codes are shown in the figure 8.

```
# make a dropping nan value in the dataset model
def drop_nan(data):
    print('_' * 200)
    print('The data shape before performing drop nan is {} .'.format(data.shape))
    data = data.dropna()
    print('The data shape after performing drop nan is {} .'.format(data.shape))
    return data
```

Figure 8: The codes of Model_drop_nan

Model two: Model_eliminate_outliers

This model is built to remove the outliers values in the datasets. The input data is

the datasets we want to deal with and the output is the processed datasets. The criteria I use is Z score. After performing the Z transformation, if the Z data values is <=-3 or >3, the data will be delete in the original datasets. The codes are shown in the figure 9.

```
# make a removing outliers values in the dataset model
def eliminate_outliers(data):
    print('_' * 200)
    print('The data shape before performing eliminate outliers is {} .'.format(data.shape))
    row, col = data.shape
    z_data = (data - data.mean()) / data.std()
    data = data.loc[((z_data > -3).sum(axis=1) == col) & ((z_data <= 3).sum(axis=1) == col)]
    print('The data shape after performing eliminate outliers is {} .'.format(data.shape))
    return data
```

Figure 9: The codes of Model_eliminate_outliers

Model three: Model_normalization

This model is built to make normalization in the datasets. The input data is the datasets we want to deal with and the output is the processed datasets. At first, the normalization method I choose is MinMaxScaler transformation. However after thoroughly thinking, I conclude it unsuitable to use MinMaxScaler since its effect mainly depends on the min value and max value. In the data exploration process, I observe that max value of some of the variable belong to 'outlier', which causes bad normalization if using MinMaxScaler. Finally, I use the L2 normalization. The codes are shown in the figure 10.

```
# make a normalizating the dataset by MinMaxScaler model
def normalization(data):
    data = preprocessing.normalize(data)
    return data
```

Figure 10: The codes of Model_normalization

C. Constructing prediction model

Constructing prediction model contains a module: make_prediction. And this modules is related to two sub-models: linear_regression and ridge_regression. The topological graph is shown in the figure 11.

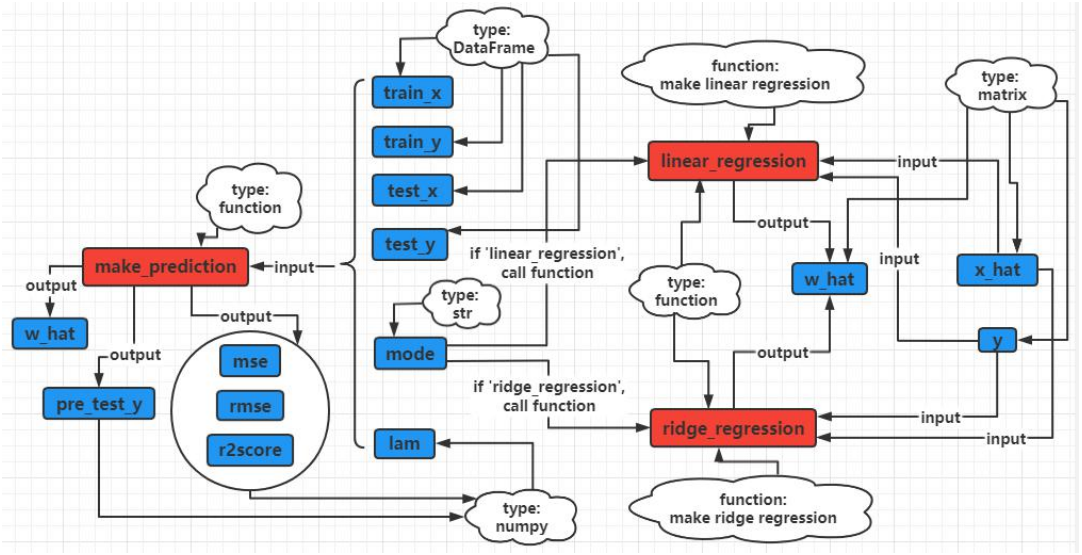

Figure 11: The topological graph for construction prediction model and the detail of two sub-models

Module: Make_prediction

This module handles the split data and choose to make linear regression and ridge regression. The input parameters are train_x(train samples), train_y(the label of train samples), test_x(test samples), test_y(the label of test samples), lam(for ridge regression) and mode. If the mode is turned to linear_regression, it conducts linear regression to make prediction; if the mode is turned to ridge_regression, it conducts ridge regression to make prediction. The output includes w_hat(can be converted to predicted best w), pre_test_y(predicted label for test samples), MSE, RMSE, R2 score. The codes are shown in the figure 12.

```
# make a prediction model (you can choose linear regression or ridge regression)
def make_prediction(train_x, train_y, test_x, test_y, model='linear_regression', lam=1.2):
    train_x = mat(train_x)
    train_y = mat(train_y).reshape(-1, 1)
    row, col = train_x.shape
    ones = np.ones((row, 1))
    train_x_hat = np.c_[train_x, ones]
    w_hat = np.zeros((1, np.shape(train_x)[1] + 1))
    # print(sample_x_add)
    if model == 'ridge_regression':
        w_hat = ridge_regression(train_x_hat, train_y, lam)
    elif model == 'linear_regression':
        w_hat = linear_regression(train_x_hat, train_y)
    w = w_hat[:-1]
    b = w_hat[-1]
    pre_test_y = np.dot(test_x, w) + b
    mse = mean_squared_error(pre_test_y, test_y)
    rmse = np.sqrt(mse)
    r2score = r2_score(test_y, pre_test_y.flatten().A[0])
    return w_hat, pre_test_y, mse, rmse, r2score
```

Figure 12: The codes of Make_prediction module

Model one: Linear_regression

This model is built to make linear regression for split data. The input parameters are x_hat(transformed samples, matrix) and y(samples label). The output is w_hat which can be converted to the best w and further reflect the predicted function. The crucial equation for linear regression is $W^* = (X^TX)^{-1}X^TY$. However, it is possible for X^TX cannot be inverse due to singular matrix, so I raise a exception to deal with the latent error and will catch this exception in practice. The codes are shown in the figure 13.

```
# make a linear regression model
def linear_regression(x_hat, y):
    partial = np.dot(x_hat.T, x_hat)
    if np.linalg.det(partial) == 0:
        raise Exception("This matrix is singular, cannot do inverse. We suggest you should make ridge regression")
    w_hat = np.dot(np.linalg.inv(partial), (np.dot(x_hat.T, y)))
    return w_hat
```

Figure 13: The codes of Linear_regression model

Model two: Ridge_regression

This model is built to make ridge regression for split data. The input parameters are x_hat(transformed samples, matrix) and y(samples label). The

output is w_hat which can be converted to the best w and further reflect the predicted function. The crucial equation for linear regression is $W^* = (X^TX+\lambda I)^{-1}X^TY$. Different from linear regression, $X^TX+\lambda I$ must be a full rank matrix in theoretical so actually there is no need to detect the singular matrix(but I just want to verify the theory). The codes are shown in the figure 14.

```
# make a ridge regression model
def ridge_regression(x_hat, y, lam=0.1):
    partial = (x_hat.T * x_hat) + np.eye(np.shape(x_hat)[1]) * lam
    if np.linalg.det(partial) == 0:
        print("This matrix is singular, cannot do inverse")
        return
    w_hat = np.linalg.inv(partial) * (x_hat.T * y)
    return w_hat
```

Figure 14: The codes of Ridge_regression model

D. Constructing evaluation system

Constructing evaluation system contains three sub-modules: k_fold_cross_ validation, test ridge regression and evaluation system. Module one use the optimal mode to call the function linear_regression or ridge_regression; Module two call the module two; Module three is illustrated by figure and statistic these two methods and includes three sub-models: plot_residual, plot broken line and scatter test.

Module one(Model one): k_fold_cross_validation

The function of this module is perform k fold cross validation. On one hand, we use it to output mean MSE and mean RMSE to evaluate linear regression and ridge regression objectively. On the other hand, this module could make contribution to find the best lambda for ridge regression. In the follow module, I will introduce in detail. The input parameters are k, data_x(samples), data_y(label), mode(choose what regression will you evaluate) and lam(for calling ridge regression model). The output are mse_list, mean_mse and mean_rmse. The topological graph for module one are shown in the figure 15.

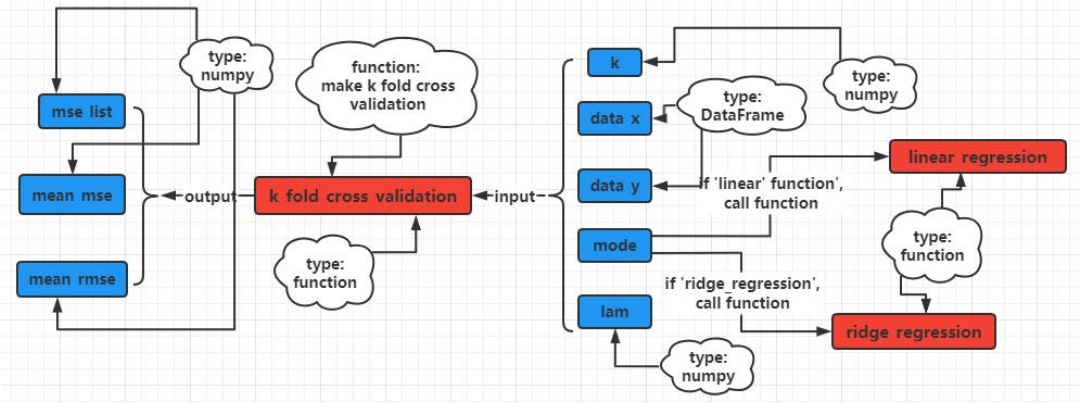

Figure 15: The topological graph for module one

The spirit of writing this function is make full use of the index for list. K fold cross validation require the percentage of k% samples be divided from total samples to play a role as test datasets. So we first calculate the length of the samples and use the principle of window sliding to record the index range in each

time. Then we just cut off this index range data and combine the rest of the datasets so that achieve splitting the test data in each cross validation. What's more, when we conducting k fold cross validation, it is crucial to perform data normalization for data_x since it eliminates the dimension for 13 attributes. In the process, I call the make_prediction function to obtain the MSE in each validation for linear regression or ridge regression. The main functions I call is numpy.concatenate. The codes are shown in the figure 16.

```
# make a k fold cross validation model
def k_fold_cross_validation(k, data_x, data_y, mode='linear_regression', lam=1.2):
    num_val_samples = len(data_x) // k
    mse_list = []
    for i in range(k):
        val_data_x = data_x[i*num_val_samples:(i+1)*num_val_samples]
        val_data_y = data_y[i*num_val_samples:(i+1)*num_val_samples]
        partial_train_data_x = np.concatenate([data_x[:i*num_val_samples], data_x[(i+1)*num_val_samples:]], axis=0)
        partial_train_data_y = np.concatenate([data_y[:i*num_val_samples], data_y[(i+1)*num_val_samples:]], axis=0)
        # val_data_x = normalization(val_data_x)
        # partial_train_data_x = normalization(partial_train_data_x)
        mse = 0
        if mode == 'linear_regression':
            _, _, mse, _, _ = make_prediction(partial_train_data_x, partial_train_data_y, val_data_x, val_data_y,
                                              'linear_regression')
        elif mode == 'ridge_regression':
            _, _, mse, _, _ = make_prediction(partial_train_data_x, partial_train_data_y, val_data_x, val_data_y,
                                              'ridge_regression', lam)
        mse_list.append(mse)
    mean_mse = np.mean(mse_list)
    mean_rmse = np.sqrt(mean_mse)
    return mse_list, mean_mse, mean_rmse
```

Figure 16: The codes for k_fold_cross_validation

Module two(Model two): test_ridge_regression

The function of this module is calculating the best lambda with the help of the built function: k_fold_cross validation. The input parameters are data_x(samples) and data_y(label). The output is best_lambda. The topological graph of test_ridge _regression is shown in the figure 17.

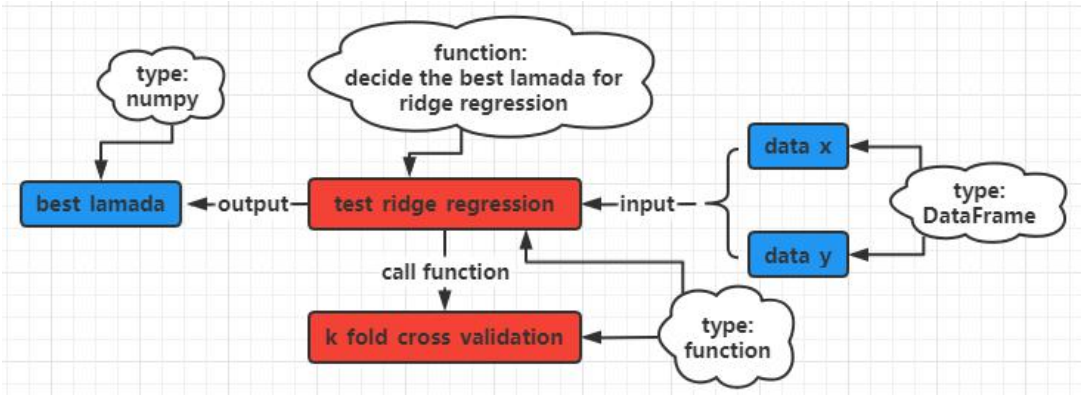

Figure 17: The topological graph of test_ridge_regression

We use 10 fold cross validation to solve the best lambda. For cover all the reasonable value of lambda, I use $e^{(i-10)}$ to update lambda for each loop. We just want to find the least mean MSE in 30 loop and record its corresponding lambda which is the best lambda we want. The codes are shown in the figure 18.

```python
# make a finding best lamdam for ridge regression model
def test_ridge_regression(data_x, data_y):
    num_test = 30
    best_lamada = 0
    min_mse = 1000000
    for i in range(num_test):
        lam = np.exp(i - 10)
        _, mean_mse, _ = k_fold_cross_validation(10, data_x, data_y, mode='ridge_regression', lam=lam)
        if mean_mse <= min_mse:
            min_mse = mean_mse
            best_lamada = lam
    return best_lamada
```

Figure 18: The codes for test_ridge_regression

Module three: evaluation system

This module is built to evaluate the performance of the train linear regression model or ridge regression model. In totally. I use two kinds of method to scale: figure and statistic. All the statistics including MSE, RMSE, R2 score, mean MSE and mean RMSE are come from the above built function: make_prediction and k_fold_cross_validation. The figure part consists of three sub-models: plot_scatter _test, plot_residual and plot_broken_line. The topological graph of evaluation system is shown in the figure 19.

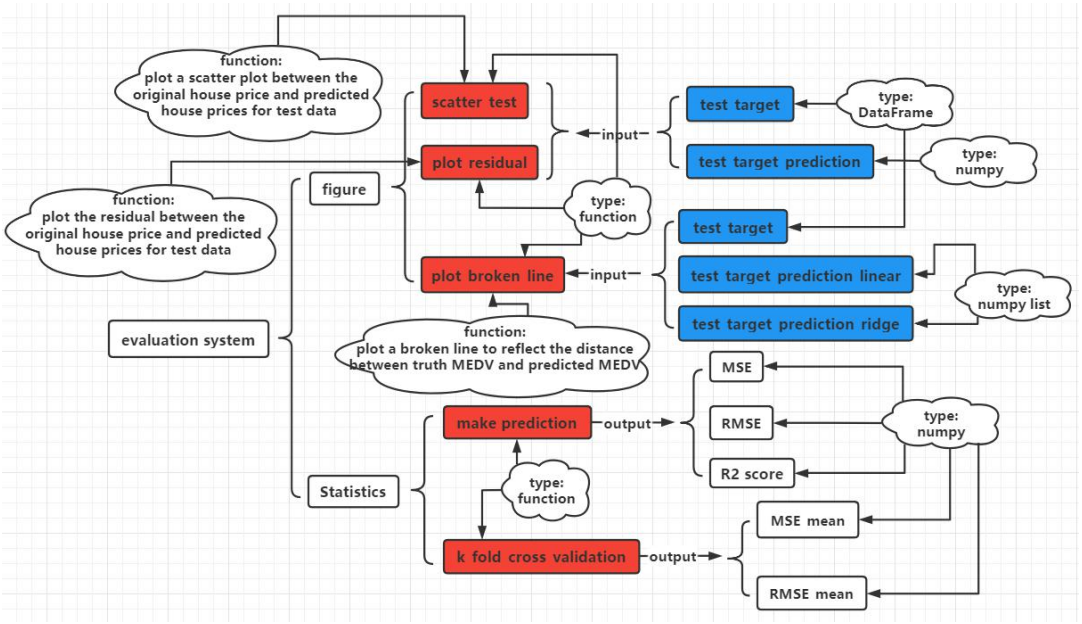

Figure 19: The topological graph for constructing evaluation system and the detail of five sub-models

Model three: Plot_scatter_test

This model is built to plot a scatter plot between the corresponding original house price and predicted price for test data. With the scatter plot, we can directly observe the distance between the truth label and the predicted label. The input parameters are test_target(label, DataFrame), test_target_prediction(label, list). It is worth to said that we should change the dimension of test_target_prediction(51,1) to the same as the dimension of test_target(51,). The key function we use is numpy.flatten( )A[0]. The codes are shown in the figure 20.

```
# plot a scatter plot between the original house price and predicted house prices for test data
def plot_scatter_test(test_target, test_target_prediction):
    plt.scatter(np.arange(len(test_target)), test_target, color='red', label='MEDV_test')
    plt.scatter(np.arange(len(test_target)), test_target_prediction.flatten().A[0], color='blue',
                label='MEDV_test_prediction')
    plt.legend(loc=2)
```

Figure 20: The codes of Plot_scatter_test

Model four: Plot_residual

This model is built to plot the residual figure for the test data and predicted test data. The residual is a further presentation of the distance between truth label and predicted label. If the residual is more farther to 0, it means that the predicted label is less accurate. So observe the distribution of residual can help us to judge whether the loss is acceptable. The main function I use is matplotlib.axhline(to record the zero line in convenience to contrast) and matplotlib.scatter( ). The codes are shown in the figure 21.

```
# plot the residual figure for the test data and predicted test data
def plot_residual(test_target, test_target_prediction):
    res = [test_target.iloc[i] - test_target_prediction[i].flatten().A[0] for i in range(0, len(test_target))]
    plt.scatter(test_target_prediction.flatten().A[0], res)
    plt.axhline(y=0, color='r', linestyle='--')
    plt.xlabel('Predicted test data')
    plt.ylabel('Residual')
    return
```

Figure 21: The codes of Plot_residual

Model five: Plot_broken_line

This model is built to plot a broken line to reflect the tendency of test label and predicted label along the dataframe index. From the broken line, we can directly observe whether the predicted label change trend is similar to the truth label change trend. The input parameters are test_target(truth label), test_target_prediction _linear(estimate linear label) and test_target_prediction_ridge(estimate ridge label). In addition, we should convert the type of test_target_prediction_linear and test_target_prediction_ridge into series assigning the test_target index for binding the corresponding group. What's more, we use solid line to present truth label and use dotted line to present estimate label. The codes are shown in the figure 22.

```
# plot a broken line to reflect the distance between truth MEDV and predicted MEDV
def plot_broken_line(test_target, test_target_prediction_linear, test_target_prediction_ridge):
    test_target_prediction_linear = pd.Series(test_target_prediction_linear.flatten().A[0],
                                              index=test_target.index).sort_index()
    test_target_prediction_ridge = pd.Series(test_target_prediction_ridge.flatten().A[0],
                                             index=test_target.index).sort_index()
    test_target = test_target.sort_index()
    plt.plot(test_target.index, test_target, label='truth', color='black', linestyle='-')
    plt.plot(test_target_prediction_linear.index, test_target_prediction_linear, label='linear', color='green',
             linestyle='--')
    plt.plot(test_target_prediction_ridge.index, test_target_prediction_ridge, label='ridge', color='blue',
             linestyle='--')
    plt.title('The predicted MEDV and truth MEDV for test data')
    plt.ylabel('MEDV')
    plt.legend(loc=2)
```

Figure 22: The codes of Plot_broken_line

Data Logging and Processing:

A. Make data exploration

In this part, I mainly use two methods to make data exploration consisting of direct observation and presenting the related figures.

a. Direct observation

I load the Boston price datasets and open it in excel and preliminary judge there is none missing values in the datasets. Furthermore, I observe that this datasets have 14 attributes and 508 samples. In addition, the MEDV attribute is the label for the Boston datasets and the other attributes is variable. The we load the data in python and analysis the data by the model we built.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1		crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
2	1	0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24
3	2	0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6
4	3	0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
5	4	0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
6	5	0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.9	5.33	36.2
7	6	0.02985	0	2.18	0	0.458	6.43	58.7	6.0622	3	222	18.7	394.12	5.21	28.7
8	7	0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.6	12.43	22.9
9	8	0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311	15.2	396.9	19.15	27.1
10	9	0.21124	12.5	7.87	0	0.524	5.631	100	6.0821	5	311	15.2	386.63	29.93	16.5
11	10	0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311	15.2	386.71	17.1	18.9
12	11	0.22489	12.5	7.87	0	0.524	6.377	94.3	6.3467	5	311	15.2	392.52	20.45	15
13	12	0.11747	12.5	7.87	0	0.524	6.009	82.9	6.2267	5	311	15.2	396.9	13.27	18.9
14	13	0.09378	12.5	7.87	0	0.524	5.889	39	5.4509	5	311	15.2	390.5	15.71	21.7
15	14	0.62976	0	8.14	0	0.538	5.949	61.8	4.7075	4	307	21	396.9	8.26	20.4
16	15	0.63796	0	8.14	0	0.538	6.096	84.5	4.4619	4	307	21	380.02	10.26	18.2
17	16	0.62739	0	8.14	0	0.538	5.834	56.5	4.4986	4	307	21	395.62	8.47	19.9
18	17	1.05393	0	8.14	0	0.538	5.935	29.3	4.4986	4	307	21	386.85	6.58	23.1
19	18	0.7842	0	8.14	0	0.538	5.99	81.7	4.2579	4	307	21	386.75	14.67	17.5
20	19	0.80271	0	8.14	0	0.538	5.456	36.6	3.7965	4	307	21	288.99	11.69	20.2
21	20	0.7258	0	8.14	0	0.538	5.727	69.5	3.7965	4	307	21	390.95	11.28	18.2
22	21	1.25179	0	8.14	0	0.538	5.57	98.1	3.7979	4	307	21	376.57	21.02	13.6
23	22	0.85204	0	8.14	0	0.538	5.965	89.2	4.0123	4	307	21	392.53	13.83	19.6
24	23	1.23247	0	8.14	0	0.538	6.142	91.7	3.9769	4	307	21	396.9	18.72	15.2
25	24	0.98843	0	8.14	0	0.538	5.813	100	4.0952	4	307	21	394.54	19.88	14.5
26	25	0.75026	0	8.14	0	0.538	5.924	94.1	4.3996	4	307	21	394.33	16.3	15.6
27	26	0.84054	0	8.14	0	0.538	5.599	85.7	4.4546	4	307	21	303.42	16.51	13.9
28	27	0.67191	0	8.14	0	0.538	5.813	90.3	4.682	4	307	21	376.88	14.81	16.6
29	28	0.95577	0	8.14	0	0.538	6.047	88.8	4.4534	4	307	21	306.38	17.28	14.8
30	29	0.77299	0	8.14	0	0.538	6.495	94.4	4.4547	4	307	21	387.94	12.8	18.4
31	30	1.00245	0	8.14	0	0.538	6.674	87.3	4.239	4	307	21	380.23	11.98	21

Figure 23: The datasets of Boston price in csv format

b. Data logging in and present the related property figures

I load the datasets and save them in a dataframe. Then I call the three model: plot_coefficient_matrix model, plot_distribution model and plot_boxplot model to plot the three figures for analysis the property of the datasets. The codes are shown in the figure 24.

```
# load boston dataset
boston = load_boston()

# arrange the data into dataframe
df_boston = pd.DataFrame(data=boston['data'], columns=boston['feature_names'])
df_boston['MEDV'] = boston.target
attribute = boston.feature_names

# plot the heatmap reflecting the linear relationships between the variables
plt.figure(0, figsize=(10, 10))
plot_coefficient_matrix(df_boston)

# plot the distribution of 13 attributes for MEDV
plt.figure(1, figsize=(25, 10))
plot_distribution(df_boston.iloc[:, 0:-1], df_boston.iloc[:, -1], attribute)

# plot the boxplot of 13 attributes
plt.figure(2, figsize=(25, 5))
df_boston_boxplot = data_preprocessing(df_boston.iloc[:, :-1], mode_eliminate_outliers=False, mode_normalization=True)
plot_boxplot(df_boston_boxplot, attribute)
```

Figure 24: The codes of plotting the data property

After we have analysis the basic property of the Boston datasets, we conduct data preprocessing.

B. Make data preprocessing

I call the module data_preprocessing and set its parameters mode_drop_nan is True, mode_eliminate_outliers is False and mode_normalization. For the reason of turning off the mode_eliminate_outliers, I support that the house price maybe unstable and could suddenly change by a wide margin. So the influential factors of house price is also unstable. What's more, although the variables of two different house price are identical, the two house price may not the same. Taking this account into consideration, I didn't deal with the outliers of the data.

Then I split the datasets into train data(90%) and test data(set random_state 65). Further, I normalize the train value of variable and test value of variable to eliminate dimension(去量纲). All the codes are shown in the figure 25.

```
# data preprocessing
df_boston = data_preprocessing(df_boston, mode_drop_nan=True, mode_eliminate_outliers=False, mode_normalization=False)

# split the datasets into data and label
sample_x = df_boston.iloc[:, 0:-1]
sample_y = df_boston.iloc[:, -1]

# split the datasets into 90% train data and 10% test data and transform the data
x_train, x_test, y_train, y_test = train_test_split(sample_x, sample_y, test_size=0.1, random_state=48)
x_train = data_preprocessing(x_train, mode_drop_nan=False, mode_normalization=True, mode_eliminate_outliers=False)
x_test = data_preprocessing(x_test, mode_drop_nan=False, mode_normalization=True, mode_eliminate_outliers=False)
```

Figure 25: The codes of making data preprocessing

C. Perform prediction model

In the rest of the part, I will perform linear regression and ridge regression at the same time and compare their output results to conclude which regression is better or how well does their performance.

For linear regression, I call the function make_prediction and set the mode parameters is 'linear_regression'. The codes are shown in the figure 26.

```
linear_w_hat, linear_pre_y_test, linear_MSE, linear_RMSE, linear_r2score = make_prediction(x_train, y_train, x_test,
                                                                                           y_test,
                                                                                           model='linear_regression'
                                                                                           )
```

Figure 26: The codes of making linear regression

However, it may encounter the matrix cannot inverse, so the process will be error. In order to deal with this situation, I have raise the exception before so I just except this exception and print 'This matrix is singular, cannot not do inverse. We suggest you should make ridge regression.' Consequently, these sample can only perform ridge regression. The codes are shown in the figure 27.

```
except Exception as e:
    print('_' * 200)
    print('This matrix is singular, cannot do inverse. We suggest you should make ridge regression')
```

Figure 27: The codes of accepting error

For ridge regression, I first call test_ridge_regression to obtain the best lambda and assign it to best_lam for ridge regression. Then I call the function make_ prediction and set the mode parameter is 'ridge_regression' and the lam parameter is best_lam. The codes are shown in the figure 27.

```
# find the best lamdam for ridge regression
best_lam = test_ridge_regression(x_train, y_train)
print('_' * 200)
print('The best lamada for ridge regression is {} .'.format(best_lam))

# train the ridge regression model and test it
ridge_w_hat, ridge_pre_y_test, ridge_MSE, ridge_RMSE, ridge_r2score = make_prediction(x_train, y_train, x_test, y_test,
                                                                                      model='ridge_regression',
                                                                                      lam=best_lam)
```

Figure 27: The codes of making ridge regression

D. Perform evaluation system

For linear regression, I record the linear_w_hat, linear_MSE, linear_RMSE, linear_r2score from make_prediction function. And record the linear_MSE_mean, linear_RMSE_mean from the k_fold_cross_validation function. With the truth label and the estimate label, I plot the scatter plot by calling the function plot_scatter_test and plot the residual by calling the function plot_residual. The codes are shown in the figure 28.

```
linear_w_hat, linear_pre_y_test, linear_MSE, linear_RMSE, linear_r2score = make_prediction(x_train, y_train, x_test,
                                                                                           y_test,
                                                                                           model='linear_regression'
                                                                                           )

print('_' * 200)
print('The coefficient of predicted linear regression function is {} .'.format(linear_w_hat.flatten()[0]))
print('_' * 200)
print('The MSE of trained linear regression model is {} .'.format(linear_MSE))
print('The RMSE of trained linear regression model is {} .'.format(linear_RMSE))
print('The r2 score of trained linear regression model is {} .'.format(linear_r2score))

# plot a scatter plot between the original house price and predicted house prices for test data in linear regression
plt.figure(3, figsize=(12, 8))
plot_scatter_test(y_test, linear_pre_y_test)
plt.title('The original house price and predicted house prices for test data in linear regression')
plt.ylabel('Price')

# plot the residual between the original house price and predicted house prices for test data in linear regression
plt.figure(4, figsize=(12, 8))
plot_residual(y_test, linear_pre_y_test)
plt.title('The residual between the original house price and predicted house prices for test data in linear '
          'regression')

# make 10 fold cross validation for linear regression model
linear_MSE_list, linear_MSE_mean, linear_RMSE_mean = k_fold_cross_validation(10, x_train, y_train,
                                                                             mode='linear_regression')
print('_' * 200)
print('The MSE list of linear regression model is {} .'.format(linear_MSE_list))
print('The MSE mean of linear regression model is {} .'.format(linear_MSE_mean))
```

Figure 28: The codes of the evaluation of linear regression

For ridge regression, the evaluative process is similar. I record the ridge_w_hat, ridge_MSE, ridge_RMSE, ridge_r2score from make_prediction function. And record the ridge_MSE_mean, ridge_RMSE_mean from the k_fold_cross_validation function with input best lambda. With the truth label and the estimate label, I plot the scatter plot by calling the function plot_scatter_test and

plot the residual by calling the function plot_residual. The codes are shown in the figure 29.

```
ridge_w_hat, ridge_pre_y_test, ridge_MSE, ridge_RMSE, ridge_r2score = make_prediction(x_train, y_train, x_test, y_test,
                                                                                      model='ridge_regression',
                                                                                      lam=best_lam)
print('_' * 200)
print('The coefficient of predicted ridge regression function is {} .'.format(ridge_w_hat.flatten()[0]))
print('_' * 200)
print('The MSE of trained ridge regression model is {} .'.format(ridge_MSE))
print('The RMSE of trained ridge regression model is {} .'.format(ridge_RMSE))
print('The r2 score of trained ridge regression model is {} .'.format(ridge_r2score))

# plot a scatter plot between the original house price and predicted house prices for test data in ridge regression
plt.figure(5, figsize=(12, 8))
plot_scatter_test(y_test, ridge_pre_y_test)
plt.title('The original house price and predicted house prices for test data in ridge regression')
plt.ylabel('Price')

# plot the residual between the original house price and predicted house prices for test data in ridge regression
plt.figure(6, figsize=(12, 8))
plot_residual(y_test, ridge_pre_y_test)
plt.title('The residual between the original house price and predicted house prices for test data in ridge regression')

# make 10 fold cross validation for ridge regression model
ridge_MSE_list, ridge_MSE_mean, ridge_RMSE_mean = k_fold_cross_validation(10, x_train, y_train,
                                                                          mode='ridge_regression', lam=best_lam)
print('_' * 200)
print('The MSE list of ridge regression model is {} .'.format(ridge_MSE_list))
print('The MSE mean of ridge regression model is {} .'.format(ridge_MSE_mean))
print('The RMSE mean of ridge regression model is {} .'.format(ridge_RMSE_mean))
```

Figure 29: The codes of the evaluation of ridge regression

Ultimately, I plot the broken line in convenience to analysis the different between truth label, estimate linear label and estimate ridge label. The codes are shown in the figure 30.

```
# plot a broken line to reflect the distance between truth MEDV and predicted MEDV
plt.figure(7, figsize=(25, 10))
plot_broken_line(y_test, linear_pre_y_test, ridge_pre_y_test)
```

Figure 30: The codes of plotting a broken line

Above are all the data logging process, next part I will focus on analysing the output data and figure in detail.

Experimental Results and Analysis:

In the process of making data exploration and performing evaluation system, there are some output results so let us analyse them respectively.

A. Make data exploration

a. plot_coefficient_matrix

Coefficient matrix reflect the correlation between the two variable. The figure 31 show that if the color is more deeper, the relationship between abscissa and ordinate is more intimate. Through the figure, we observe that RAD and Tax have a high correlation. What's more, it prove that the coefficient matrix is symmetric along the diagonal.

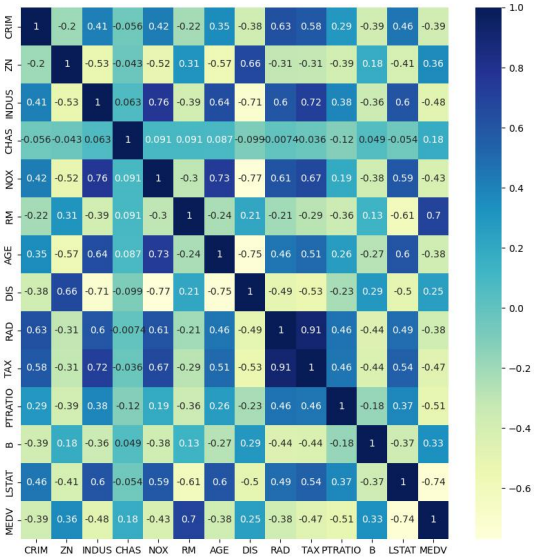

Figure 31: The coefficient matrix(heatmap) of the 13 variables

b. plot_distribution

I observe that CRIM and B data are relatively concentrated distribution so the boxplot of these two variable may have lots of outlier. I also observe that CHAS only have two values: 0 or 1.

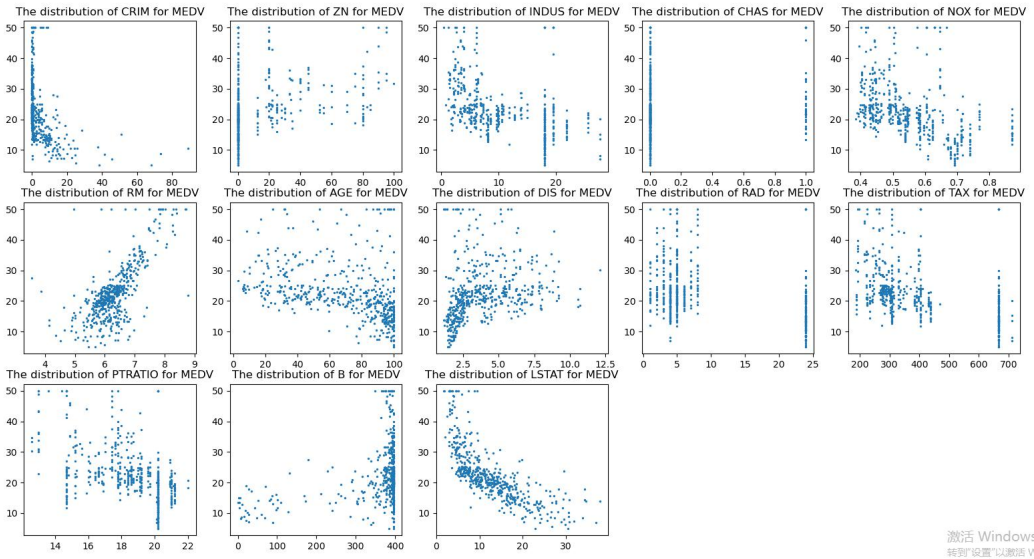

Figure 32 The distribution of 13 variables on MEDV

c. plot_boxplot

I observe that the CRIM and RM do have some 'outlier' since they are non-uniform distribution. However, it is normal for house price problem so we don't necessary to perform eliminate_outlier function.

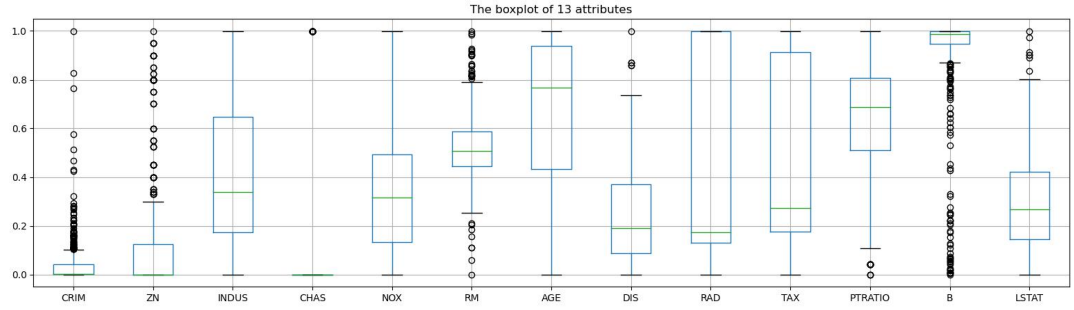

Figure 33: The boxplot of 13 variables

B. Data proprecessing

a. Drop_nan

It can see that there is no missing value in the datasets since the shape is the same no matter before normalization or after normalization.

```
The data shape before performing drop nan is (506, 14) .
The data shape after performing drop nan is (506, 14) .
```

Figure 34: The shape of datasets before and after normalization

b. Normalization

From the figure 35, I observe that all the values of variable are transformed into the range of [0,1].

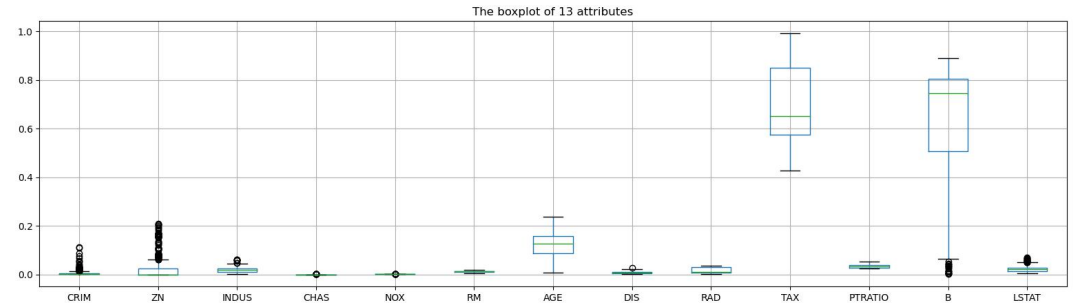

Figure 35: The boxplot of 13 variable after performing normalization

C. Perform evaluation system

- **For linear regression**

I output the coefficient of predicted linear regression function which is shown in the figure 36. The first 13 items are the w_1-w_{13} respectively and the last item is b.

```
The coefficient of predicted linear regression function is [[-9.39225636e+01  1.52750347e+01  1.50330322e+01  1.42351415e+03
  -5.99339473e+03  2.68702228e+03 -1.11567163e+00 -5.42173091e+02
   1.48763974e+02  1.02590083e+00 -3.84519228e+02  9.24308495e+00
  -2.48153907e+02  1.05246841e+01]] .
```

Figure 36: The coefficient of predicted linear regression function

I output some informative results: MSE, RMSE and R2 score respective. IThe MSE is 15.63, the RMSE is 3.95 and the R2 score is 0.80. For MSE and RMSE, the values are relatively small which reflects that our linear model perform well. For R2 score, it reaches to 0.8 which is also a relative high scores. Generally speaking, if the R2 score is more closed to 1, it prove that the model is more similar to the truth latent law. So it confirms our linear regression model have a good fitting performance. Then the results of applying our train linear function to test datasets(10%) are shown in the figure 37.

```
The MSE of trained linear regression model is 15.634547648906345 .
The RMSE of trained linear regression model is 3.9540545834505556 .
The r2 score of trained linear regression model is 0.7992195079770382 .
```

Figure 37: The results of linear regression on test datasets

I also calculate the MSE mean and RMSE mean in 10 fold cross validation to make full use of the datasets to evaluate the linear regression model as far as possible. The MSE mean is 23.96 and the RMSE mean is 4.90. Both of them are acceptable and can reflect the generated ability of fit linear regression model. The explicit output are shown in the figure 38.

```
The MSE list of linear regression model is [32.77951874945847, 40.32755831135788, 19.784309303513865, 21.559529113548336, 33.75841595607258, 15.43208377185822, 11.8263
The MSE mean of linear regression model is 23.963970867535423 .
The RMSE mean of linear regression model is 4.895300896526732 .
```

Figure 38: The results of linear regression by 10 fold cross validation on train datasets

For objectively observe the truth label and estimate label, I further plot some figures. The first one is the truth house price and predicted house price for test data in linear regression model which is shown in the figure 39. We see that most of the corresponding test data and predicted data are closed but there does exists some poor predicted points.

Figure 39: The truth house price and predicted house prices for test data in linear regression

In order to visual the distance between the truth price and the predicted price, I plot the residual for corresponding price which are shown in the figure 40. We take zero as the reference line, if the absolute of residual is more closed to zero, it proves that the estimate price is more closed to the truth price. We see that the range of the residual is [-8,15] and most of residual are relative small. It prove that the my linear regression is reasonable.

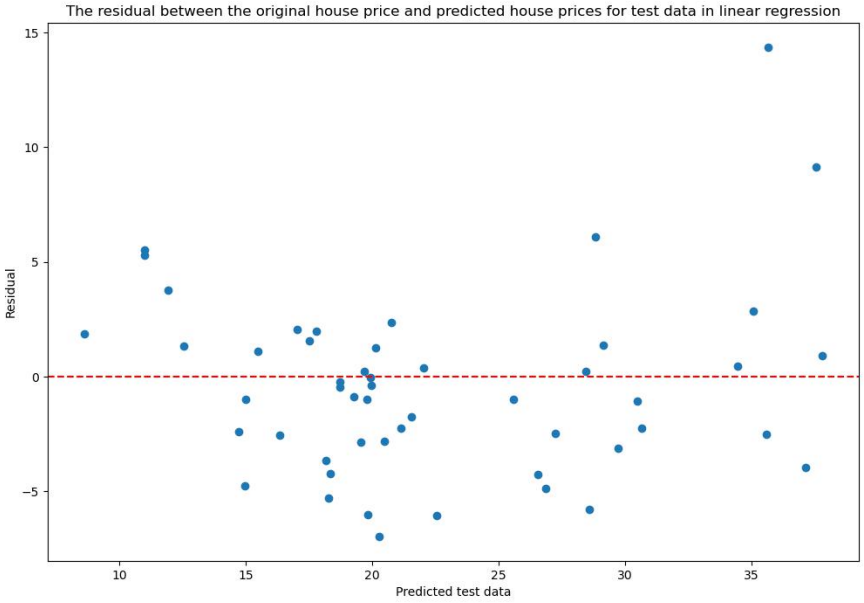

Figure 40: The residual between the truth price and the predicted price

- **For ridge regression**

The analysis for the ridge regression model is similar to the linear regression model. I first use 10 fold cross validation to find the best lambda for the ridge regression model and the best lambda is 4.54×10^{-5}. The larger lambda, the smaller w_hat. So it is reasonable for lambda to reach such a small number.

```
The best lambda for ridge regression is 4.539992976248485e-05 .
```

Figure 41: The best lambda for ridge regression model

The coefficients of predicted regression function are shown in the figure 42.

```
The coefficient of predicted ridge regression function is [[-9.37224334e+01  1.71102010e+01 -1.29303618e+01  9.69840132e+02
  -6.93907436e+02  2.31826592e+03 -3.60364625e+00 -4.88567484e+02
   1.57786019e+02 -1.56791376e+00 -3.71199011e+02  8.68419226e+00
  -2.82894541e+02  1.23175320e+01]] .
```

Figure 42: The coefficient of the predicted ridge regression model

In the process of perform my fit ridge regression model in the test datasets,the MSE is 16.36, RMSE is 4.04 and R2 score is 0.79. Both of these evaluative information confirm the good performance of my fit ridge regression model. The explicit information is shown in the figure 43.

```
The MSE of trained ridge regression model is 16.359761301898715 .
The RMSE of trained ridge regression model is 4.044720175969991 .
The r2 score of trained ridge regression model is 0.7899062385854698 .
```

Figure 43: The MSE, RMSE, R2 score of the fit ridge regression model in test data

In 10 fold cross validation for ridge regression, the MSE mean is 24.31 and the RMSE mean is 4.93. It also indicates that the fit ridge regression model has a good generated ability.

```
The MSE list of ridge regression model is [33.01062740097837, 41.7987161589962, 17.667972956541803, 22.278346721520993, 35.08864182080078, 15.731077549747965, 11.54582
The MSE mean of ridge regression model is 24.311316806097604 .
The RMSE mean of ridge regression model is 4.930650748744795 .
```

Figure 44: The MSE mean and RMSE mean for ridge regression in 10 fold cross validation

The truth price and predicted prices for test data in ridge regression is shown in the figure 45. The analysis is similar to the linear regression so we just show the figure.

Figure 45: The truth price and estimate price for test data in ridge regression

The residual between truth house price and predicted house price is shown in the figure 46. The range of residual is [-8,15] and most of residual are distributed around zero.

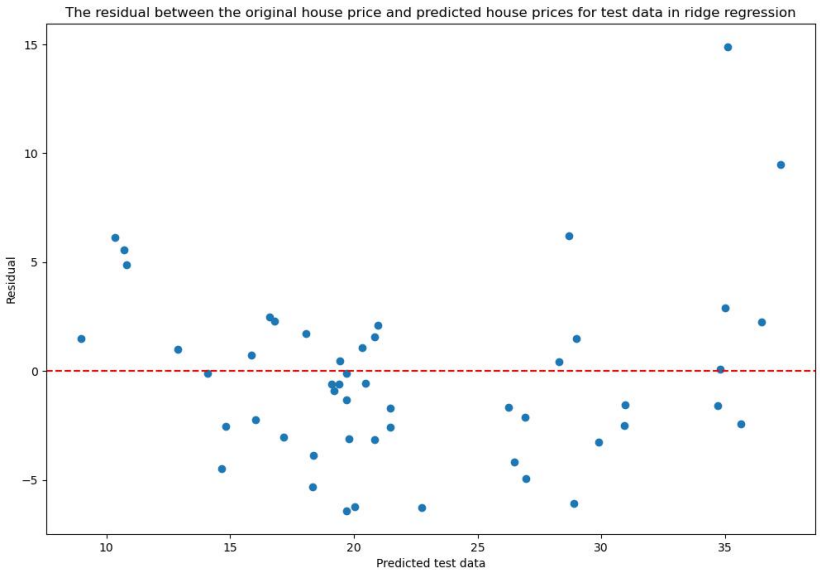

Figure 46: The residual between the corresponding truth price and the estimate price

The figure 47 shows the change tendency of truth house price, predicted house price by linear regression model and predicted house price by ridge regression model. I plot the truth house price with solid black line and plot the predicted house price by linear regression model with dotted green line and plot the predicted house price by ridge regression model with dotted blue line. It is obvious that the blue dotted line is closed to green dotted line since the principle of the ridge regression model is resemble to the linear regression model. The ridge regression model just add the penalty item to deal with the cannot inverse problem. Both of the two dotted line is basically consistent with the black solid line meaning my two linear prediction models can successfully complete the forecast task in term of the house price problem.

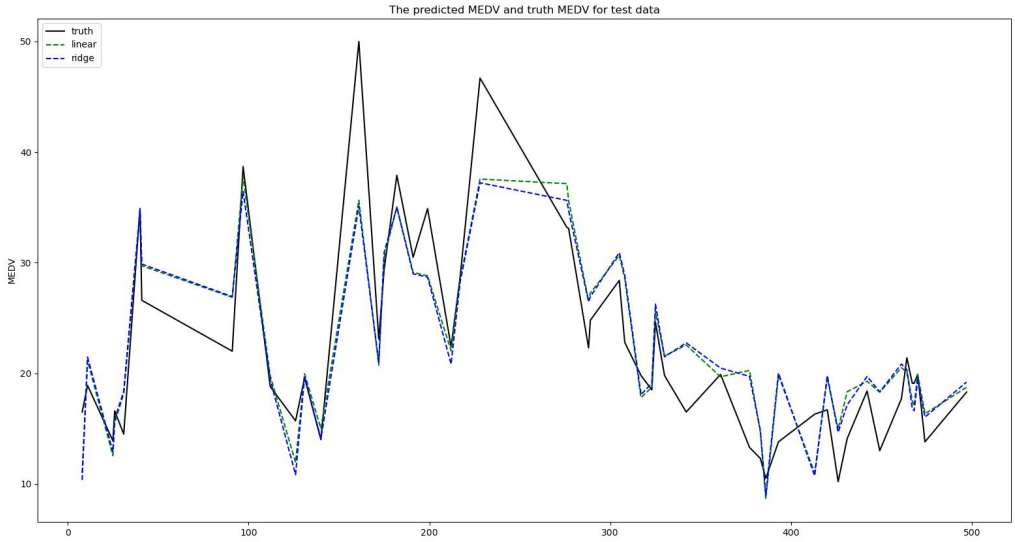

Figure 47: The tendency of truth price and estimate price using different regression

指导教师批阅意见： 成绩评定： 指导教师签字： 年　　月　　日
备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。