Homework 6

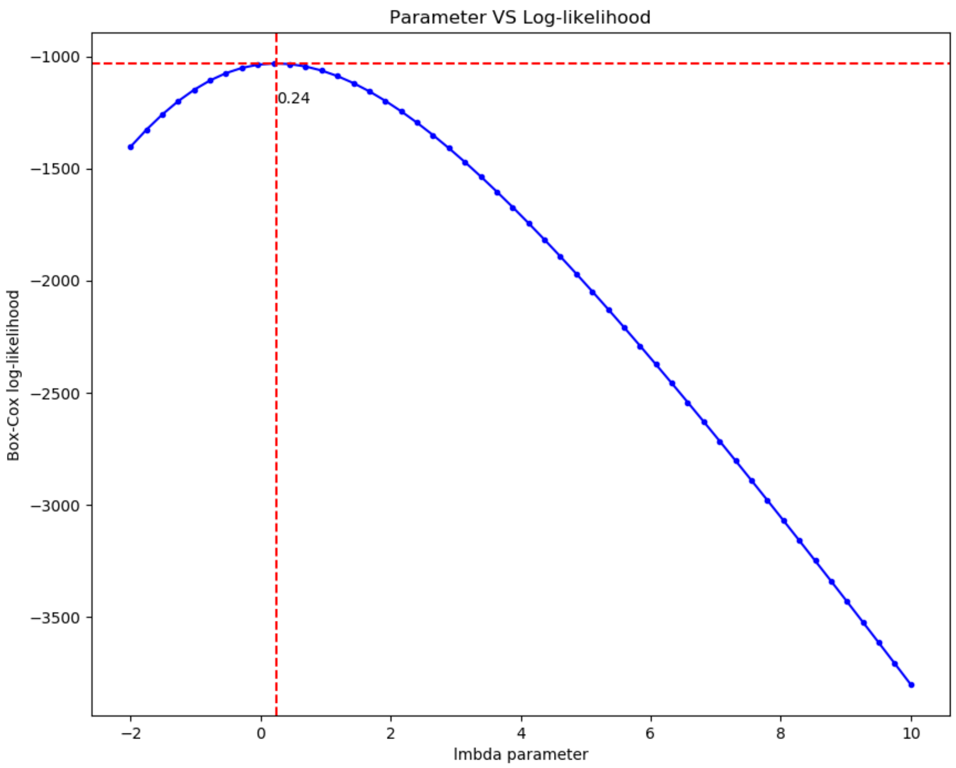
1.

(1) **List all the points (row numbers) you removed (indexed on the original dataset) as outlier points:**

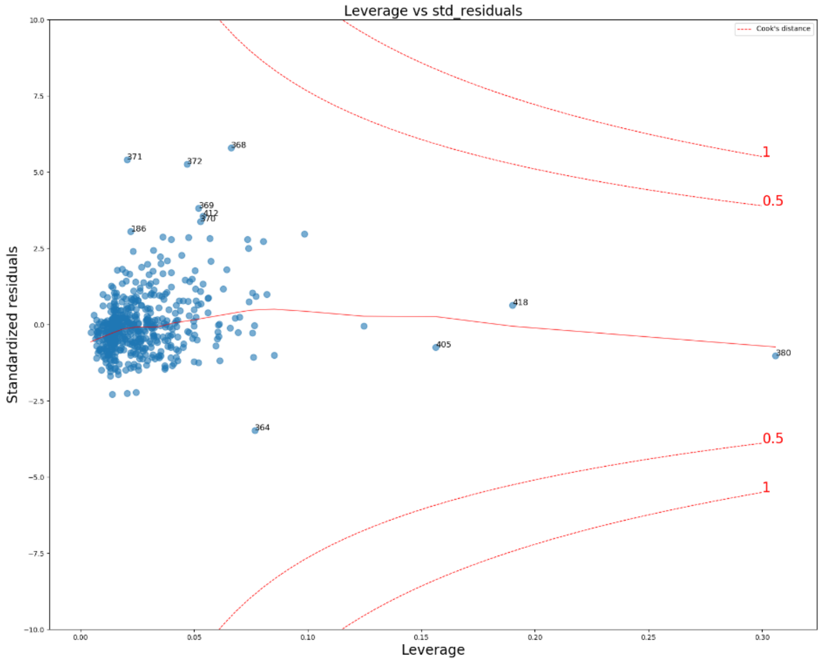
```
Outliers:  [368, 371, 372, 380, 405, 410, 418]
```

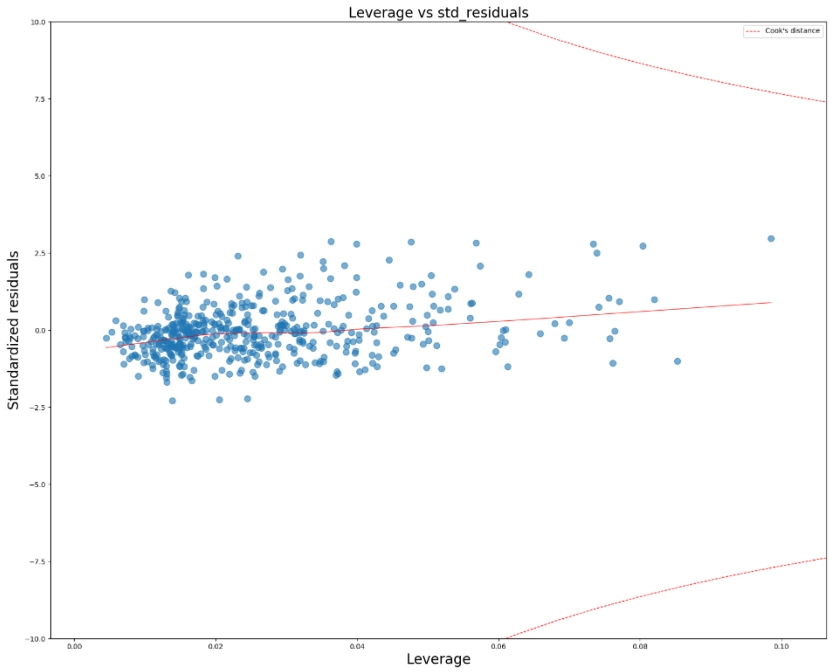
(2) **Box-Cox Transformation – Plot the Box-Cox transformation curve (Log-likelihood vs Parameter value). What is the best value of the parameter you got?**

```
Applying BOXCOX LMBDA:  0.21261640181832653
```



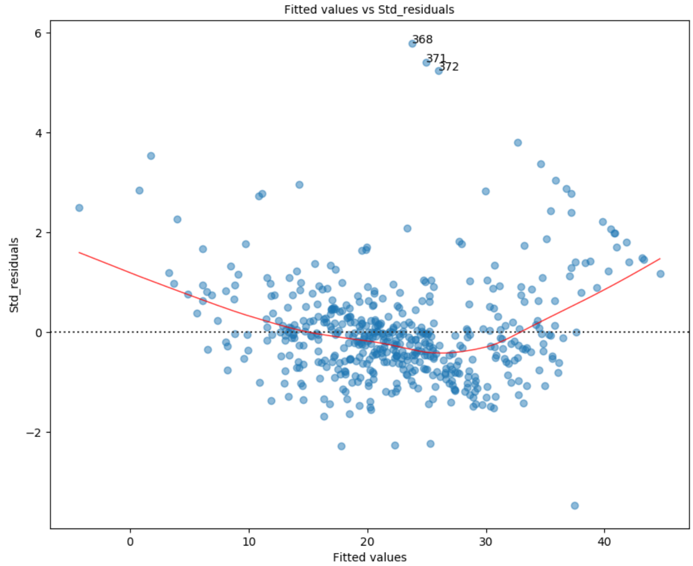
2. **Diagnostic plots used for identification of outliers. Please only include the Standard residuals vs Leverage vs Cook's distance plots (do not put other 3 plots you obtain for R). The final diagnostic plot obtained after removing all outliers should also be included:**



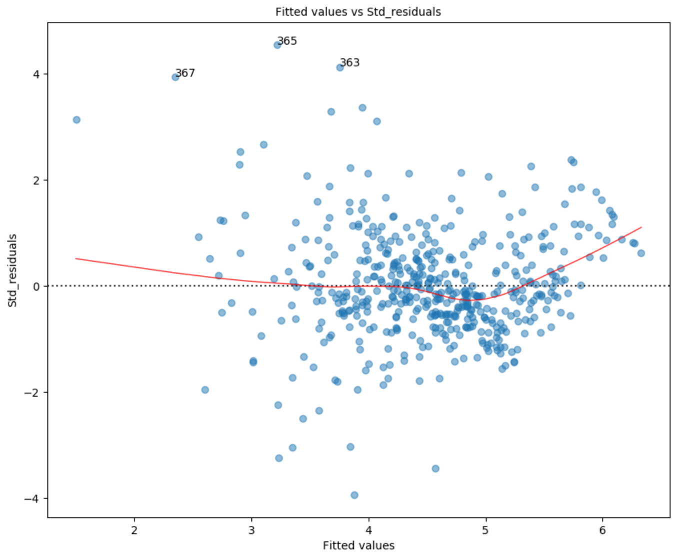


3.

(1) Plot of Standardized residuals vs Fitted values for the linear regression model obtained without any transforms (like removing outliers or transforming dependent variables).



(2) Plot of Standardized residuals vs Fitted values for the final linear regression model obtained after removing all outliers and transforming the dependent variable.



(3) Compare the two plots. What do you observe?

After removing all outliers and transforming the dependent variable, the data concentrate more in smaller scale and the range of fitted values becomes smaller.

4. Final plot of Fitted house price vs True house price. What do you observe.

5. Code screenshot

(1) Linear regression

```
# Basic linear regression
outliers, fitted_value, leverage, std_residuals, coefficient_head = getParameters(dataset_X, dataset_Y)
drawDiagnosticPlot(leverage, std_residuals, feature_number=dataset_X.shape[1]-1, path='plot_a_1.png')
drawStdResidualPlot(fitted_value, std_residuals, path='plot_d_1.png')
```

```
leverage_new, std_residuals_new = updateParameters(leverage, std_residuals, outliers)
```

```
def getParameters(dataset_X, dataset_Y):
    outliers = list()
    XTX = np.asmatrix(dataset_X.T.dot(dataset_X))
    hat_matrix = dataset_X.dot(XTX.I).dot(dataset_X.T)
    leverage = np.array(hat_matrix.diagonal())[0]
    coefficient_head = np.array(XTX.I.dot(dataset_X.T.dot(dataset_Y)))[0]
    residual_vector = np.array(dataset_Y-dataset_X.dot(coefficient_head))
    fitted_value = dataset_X.dot(coefficient_head)
    std_residuals = list()
    for item_index in range(dataset_X.shape[0]):
        std_residual = residual_vector[item_index]/ \
                  math.sqrt((residual_vector.T.dot(residual_vector)/dataset_X.shape[0])* \
                  (1-hat_matrix.item(item_index,item_index)))
        std_residuals.append(std_residual)
    std_residuals = np.array(std_residuals).reshape(1,-1)[0]
    for i in range(dataset_X.shape[0]):
        if leverage[i] > 0.1 or abs(std_residuals[i]) > 4:
            outliers.append(i)
```

```
def updateParameters(leverage, std_residuals, outliers):
    leverage_new, std_residuals_new = list(),list()
    for i in range(leverage.shape[0]):
        if i not in outliers:
            leverage_new.append(leverage[i])
            std_residuals_new.append(std_residuals[i])
    return np.array(leverage_new), np.array(std_residuals_new)
```

(2) Box-Cox transformation

```
# Transforming the dependent variable
dataset_Y_BC, lmbda = stats.boxcox(dataset_Y_new)
print('Applying BOXCOX LMBDA: ', lmbda)
drawBoxCoxTransformationPlot(dataset_Y_new, dataset_Y_BC, lmbda, path='plot_c.png')
```

```
def drawBoxCoxTransformationPlot(dataset_Y, dataset_Y_BC, lmbda_optimal, path):
    fig = plt.figure(figsize=(10,8))
    ax1 = fig.add_subplot(211)
    stats.probplot(dataset_Y, dist=stats.norm, plot=ax1)
    ax1.set_xlabel('')
    ax1.set_title('Probplot against normal distribution',fontsize=10)
    ax2 = fig.add_subplot(212)
    stats.probplot(dataset_Y_BC, dist=stats.norm, plot=ax2)
    ax2.set_title('Probplot after Box-Cox transformation',fontsize=10)
    fig.tight_layout()
    plt.savefig('plot_b.png')
    plt.close()
    print('Save figure of Probplot against normal distribution ..., path=./plot_b.png')

    lmbdas = np.linspace(-2, 10)
    llf = np.zeros(lmbdas.shape, dtype=float)
    for ii, lmbda in enumerate(lmbdas):
        llf[ii] = stats.boxcox_llf(lmbda, dataset_Y)
    fig = plt.figure(figsize=(10,8))
    ax = fig.add_subplot(111)
    ax.set_title('Parameter VS Log-likelihood')
    ax.plot(lmbdas, llf, 'b.-')
    ax.axhline(stats.boxcox_llf(lmbda_optimal, dataset_Y), color='r', ls='--')
    ax.axvline(lmbda_optimal, color='r', ls='--')
    ax.annotate("%.2f" %lmbda_optimal,xy=(lmbda_optimal,stats.boxcox_llf(lmbda_optimal, dataset_Y)-180),
                fontsize = 10)
    ax.set_xlabel('lmbda parameter',fontsize=10)
    ax.set_ylabel('Box-Cox log-likelihood',fontsize=10)
```

(3) Used the parameter value to transform the dependent variable

```
    # Transforming the dependent variable
    dataset_Y_BC, lmbda = stats.boxcox(dataset_Y_new)
    print('Applying BOXCOX LMBDA: ', lmbda)
    drawBoxCoxTransformationPlot(dataset_Y_new, dataset_Y_BC, lmbda, path='plot_c.png')
    outliers, fitted_value, leverage, std_residuals, coefficient_head = getParameters(dataset_X_new, dataset_Y
    drawStdResidualPlot(fitted_value, std_residuals, path='plot_d_2.png')
    drawFinalPrice(dataset_Y_new, dataset_X_new.dot(coefficient_head), lmbda, path='plot_e.png')
```

```
def drawFinalPrice(y, y_bc, lmbda, path):
    if lmbda:
        y_final = [math.pow(Y*lmbda+1, 1/lmbda) for Y in y_bc]
    else:
        y_final = [math.pow(Y, lmbda) for Y in y_bc]
    y_final = np.array(y_final)
    fig,ax = plt.subplots(figsize=(10,8))
    ax.scatter(y, y_final, s=20, alpha=.6)
    plt.xlabel("True house price",fontsize=10)
    plt.ylabel("Fitted house price",fontsize=10)
    plt.title("Fitted house price vs True house price")
    plt.savefig(path)
    plt.close()
    print('Save figure of Leverage vs std_residuals ..., path=./%s' %path)
```

6. Entire code

```
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.metrics import mean_squared_error, r2_score
from mpl_toolkits.axes_grid1.inset_locator import inset_axes


def loadData(path):
    dataset = pd.read_csv(path, header=None, sep=r'\s+')

    # Add one column for constant term
    dataset_X_ori = dataset.iloc[:,:13]
    dataset_X_ori['y-intercept'] = 1.
    dataset_X = np.array(dataset_X_ori)
    dataset_Y = np.array(dataset.iloc[:,13])
    return dataset_X, dataset_Y

def getParameters(dataset_X, dataset_Y):
    outliers = list()
    XTX = np.asmatrix(dataset_X.T.dot(dataset_X))
    hat_matrix = dataset_X.dot(XTX.I).dot(dataset_X.T)
    leverage = np.array(hat_matrix.diagonal())[0]
    coefficient_head = np.array(XTX.I.dot(dataset_X.T.dot(dataset_Y)))[0]
    residual_vector = np.array(dataset_Y-dataset_X.dot(coefficient_head))
    fitted_value = dataset_X.dot(coefficient_head)
    std_residuals = list()
    for item_index in range(dataset_X.shape[0]):
        std_residual = residual_vector[item_index]/ \
                    math.sqrt((residual_vector.T.dot(residual_vector)/dataset_X.shape[0])* \
                    (1-hat_matrix.item(item_index,item_index)))
        std_residuals.append(std_residual)
    std_residuals = np.array(std_residuals).reshape(1,-1)[0]
    for i in range(dataset_X.shape[0]):
        if leverage[i] > 0.1 or abs(std_residuals[i]) > 4:
            outliers.append(i)
```

```python
    return outliers, fitted_value, leverage, std_residuals, coefficient_head

def updateDataset(dataset_X, dataset_Y, outliers):
    dataset_X_new, dataset_Y_new = list(), list()
    for item_index in range(dataset_X.shape[0]):
        if item_index not in outliers:
            dataset_X_new.append(dataset_X[item_index])
            dataset_Y_new.append(dataset_Y[item_index])
    return np.array(dataset_X_new), np.array(dataset_Y_new)

def updateParameters(leverage, std_residuals, outliers):
    leverage_new, std_residuals_new = list(),list()
    for i in range(leverage.shape[0]):
        if i not in outliers:
            leverage_new.append(leverage[i])
            std_residuals_new.append(std_residuals[i])
    return np.array(leverage_new), np.array(std_residuals_new)

def drawDiagnosticPlot(leverage, std_residuals, feature_number, path):

    # Plot leverage vs std_residuals
    fig = plt.figure(figsize=(20,16))
    ax = fig.add_subplot(111)
    ax.scatter(leverage, std_residuals, s=80, alpha=.6)
    for i in range(len(leverage)):
        if leverage[i] > 0.15 or abs(std_residuals[i]) > 3:
            ax.annotate(str(i), xy=(leverage[i],std_residuals[i]), fontsize = 12)

    # Draw residuals
    sns.regplot(leverage, std_residuals,
                scatter=False,
                ci=False,
                lowess=True,
                line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})

    # cook's distance contours
    def graph(formula, x_range, label=None, line=None):
        x = x_range
        y = formula(x)
```

```python
        plt.plot(x, y, label=label, lw=1, ls='--', color='red')
        ax.annotate(str(line), xy=(x[-1],formula(x[-1])), fontsize = 20, color='red')
    graph(lambda x: np.sqrt((0.5 * feature_number * (1 - x)) / x),
        np.linspace(0.001, 0.300, 50),
        'Cook\'s distance', line=0.5) # 0.5 line
    graph(lambda x: np.sqrt((1 * feature_number * (1 - x)) / x),
        np.linspace(0.001, 0.300, 50), line=1) # 1 line
    graph(lambda x: -np.sqrt((0.5 * feature_number * (1 - x)) / x),
        np.linspace(0.001, 0.300, 50), line=0.5) # 0.5 line
    graph(lambda x: -np.sqrt((1 * feature_number * (1 - x)) / x),
        np.linspace(0.001, 0.300, 50), line=1) # 1 line

    plt.ylim(-10, 10)
    plt.xlabel("Leverage",fontsize=20)
    plt.ylabel("Standardized residuals",fontsize=20)
    plt.title("Leverage vs std_residuals",fontsize=20)
    plt.legend()
    plt.savefig(path)
    plt.close()
    print('Save figure of Leverage vs std_residuals ..., path=./%s'%path)

def drawStdResidualPlot(fitted_value, std_residuals, path):
    fig = plt.figure(1,figsize=(10,8))
    ax = fig.add_subplot(111)
    sns.residplot(fitted_value, std_residuals,
                  lowess=True,
                  scatter_kws={'alpha': 0.5},
                  line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})

    ax.set_title('Fitted values vs Std_residuals',fontsize=10)
    ax.set_xlabel('Fitted values',fontsize=10)
    ax.set_ylabel('Std_residuals',fontsize=10)

    # annotations
    abs_resid = list(std_residuals)
    abs_resid.sort(reverse=True)
    abs_resid_top_3 = abs_resid[:3]
```

```python
    for i in abs_resid_top_3:
        index = list(std_residuals).index(i)
        ax.annotate(index,
                    xy=(fitted_value[index],
                        std_residuals[index]),
                    fontsize=10)
    plt.savefig(path)
    plt.close()
    print('Save figure of Std_residuals vs Fitted ..., path=./%s'%path)

def drawBoxCoxTransformationPlot(dataset_Y, dataset_Y_BC, lmbda_optimal, path):
    fig = plt.figure(figsize=(10,8))
    ax1 = fig.add_subplot(211)
    stats.probplot(dataset_Y, dist=stats.norm, plot=ax1)
    ax1.set_xlabel('')
    ax1.set_title('Probplot against normal distribution',fontsize=10)
    ax2 = fig.add_subplot(212)
    stats.probplot(dataset_Y_BC, dist=stats.norm, plot=ax2)
    ax2.set_title('Probplot after Box-Cox transformation',fontsize=10)
    fig.tight_layout()
    plt.savefig('plot_b.png')
    plt.close()
    print('Save figure of Probplot against normal distribution ..., path=./plot_b.png')

    lmbdas = np.linspace(-2, 10)
    llf = np.zeros(lmbdas.shape, dtype=float)
    for ii, lmbda in enumerate(lmbdas):
        llf[ii] = stats.boxcox_llf(lmbda, dataset_Y)
    fig = plt.figure(figsize=(10,8))
    ax = fig.add_subplot(111)
    ax.set_title('Parameter VS Log-likelihood')
    ax.plot(lmbdas, llf, 'b.-')
    ax.axhline(stats.boxcox_llf(lmbda_optimal, dataset_Y), color='r', ls='--')
    ax.axvline(lmbda_optimal, color='r', ls='--')
    ax.annotate("%.2f" %lmbda_optimal,xy=(lmbda_optimal,stats.boxcox_llf(lmbda_optimal, dataset_Y)-180),
                fontsize = 10)
    ax.set_xlabel('lmbda parameter',fontsize=10)
    ax.set_ylabel('Box-Cox log-likelihood',fontsize=10)
    # locs = [3, 10, 4]  # 'lower left', 'center', 'lower right'
```

```python
    #     ax_inset.set_yticklabels([])
    #     ax_inset.set_title('$\lambda=%1.2f$' % lmbda)
    plt.savefig(path)
    plt.close()
    print('Save figure of Parameter VS Log-likelihood ..., path=./plot_c.png')

def drawFinalPrice(y, y_bc, lmbda, path):
    if lmbda:
        y_final = [math.pow(Y*lmbda+1, 1/lmbda) for Y in y_bc]
    else:
        y_final = [math.pow(Y, lmbda) for Y in y_bc]
    y_final = np.array(y_final)
    fig,ax = plt.subplots(figsize=(10,8))
    ax.scatter(y, y_final, s=20, alpha=.6)
    plt.xlabel("True house price",fontsize=10)
    plt.ylabel("Fitted house price",fontsize=10)
    plt.title("Fitted house price vs True house price")
    plt.savefig(path)
    plt.close()
    print('Save figure of Leverage vs std_residuals ..., path=./%s'%path)

if __name__ == "__main__":
    # Load raw data
    path = './housing.data.txt'
    dataset_X, dataset_Y = loadData(path)

    # Basic linear regression
    outliers, fitted_value, leverage, std_residuals, coefficient_head = getParameters(dataset_X, dataset_Y)
    drawDiagnosticPlot(leverage, std_residuals, feature_number=dataset_X.shape[1]-1, path='plot_a_1.png')
    drawStdResidualPlot(fitted_value, std_residuals, path='plot_d_1.png')
    print('Original MSE: ', mean_squared_error(dataset_Y,dataset_X.dot(coefficient_head)))
    print('Original R^2_SCORE: ', r2_score(dataset_Y,dataset_X.dot(coefficient_head)))
    print('Outliers: ', outliers)
    leverage_new, std_residuals_new = updateParameters(leverage, std_residuals, outliers)
    drawDiagnosticPlot(leverage_new, std_residuals_new, feature_number=dataset_X.shape[1]-1, path='plot_a_2.png
    print('='*60)

    # Removing the outliers
    dataset_X_new, dataset_Y_new = updateDataset(dataset_X, dataset_Y, outliers)
```

```python
    outliers, fitted_value, leverage, std_residuals, coefficient_head = getParameters(dataset_X_new, dataset_Y
    print('Remove Outliers MSE: ', mean_squared_error(dataset_Y_new, dataset_X_new.dot(coefficient_head)))
    print('Remove Outliers R^2_SCORE: ', r2_score(dataset_Y_new, dataset_X_new.dot(coefficient_head)))
    print('='*60)

    # Transforming the dependent variable
    dataset_Y_BC, lmbda = stats.boxcox(dataset_Y_new)
    print('Applying BOXCOX LMBDA: ', lmbda)
    drawBoxCoxTransformationPlot(dataset_Y_new, dataset_Y_BC, lmbda, path='plot_c.png')
    outliers, fitted_value, leverage, std_residuals, coefficient_head = getParameters(dataset_X_new, dataset_Y
    drawStdResidualPlot(fitted_value, std_residuals, path='plot_d_2.png')
    drawFinalPrice(dataset_Y_new, dataset_X_new.dot(coefficient_head), lmbda, path='plot_e.png')
    print('Applying BOXCOX MSE: ', mean_squared_error(dataset_Y_BC, dataset_X_new.dot(coefficient_head)))
    print('Applying BOXCOX R^2_SCORE: ', r2_score(dataset_Y_BC, dataset_X_new.dot(coefficient_head)))
    print('Applying BOXCOX Coefficient: ', coefficient_head)
    print('Applying BOXCOX Std_residuals[:5]: ', std_residuals[:5])
```