인대명 (2022311154)

2022-03-24

HW3 Report

1. First, we order the data using sort() function. We save the probability values $\frac{j-1/2}{n}$ as variable 'below_xj'. Then, we obtain the standard normal quantiles and save them as 'qj'. Finally, we plot $(q_{(j)}, x_{(j)})$ as well as the reference line which we obtained using regression.

In [2]:
```
#1
def qq_plot(xj): #assume input is a list/array
    xj.sort()
    j=np.array(range(len(xj)))+1
    below_xj=(j-1/2)/len(xj)
    qj=norm.ppf(below_xj)
    plt.plot(qj,xj) #plot Q-Q plot
    m, b = np.polyfit(qj, xj, 1)
    plt.plot(qj, m*qj+b) #plot reference line
    plt.title('Q-Q Plot')
    plt.xlabel('Theoretical Quantiles')
    plt.ylabel('Sample Quantiles')
```

2. For (a) and (b), we used the Python function boxcox to obtain the lambda that maximizes the log-likelihood function and their transformed variables. The respective Box-Cox transformations for x1 and x2 are shown below:

In [4]:
```
#(a) Box-Cox transformation for x1
transform1, lambda1= boxcox(car_prices['x1'],lmbda=None)
print("transformed variables: ", transform1, "\nlambda: ",lambda1)
```

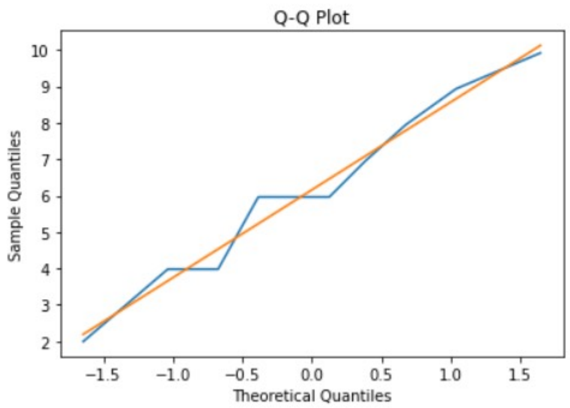
```
transformed variables:  [1.99357154 3.97994271 3.97994271 5.96225561 5.96225561 5.96225561
 6.95229389 7.9417156  8.93058988 9.91897192]
lambda:  0.9950293497559268
```

In [5]:
```
#(b) Box-Cox transformation for x2
transform2, lambda2 = boxcox(car_prices['x2'],lmbda=None)
print("transformed variables: ", transform2, "\nlambda: ",lambda2)
```

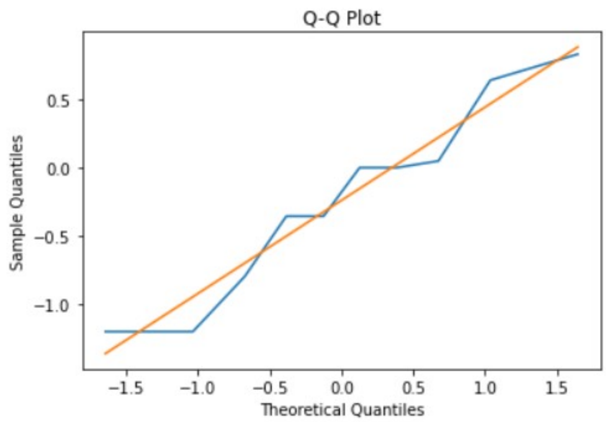
```
transformed variables:  [ 0.83302984  0.64192557  0.          -0.35665281 -1.20372064  0.
  0.04879058 -0.79839677 -0.35665281 -1.20372064]
lambda:  0.0003479745820106328
```

For (c), we used the data obtained from the Box-Cox transformations. By applying the function defined in the Python code in #1, we constructed Q-Q plots for x_1^λ and x_2^λ respectively. Comparing the two, we may say that the Q-Q plot obtained from 'x1' data after transformation is closer to a straight line than 'x2'. Thus, the first data is closer to a normal distribution.

In [6]:
```
#(c) We can use the data obtained from each Box-Cox
#x1
qq_plot(transform1)
```



In [7]:
```
#x2
qq_plot(transform2)
```



3. First, we define two functions that return the sample mean vector and the sample covariance matrix of a given set of data, respectively.

```
In [9]: #3
def mean_vector(data):
    ones=pd.DataFrame({'ones': np.ones(len(data))})
    mean=ones.transpose().dot(data)/len(data)
    return mean
```

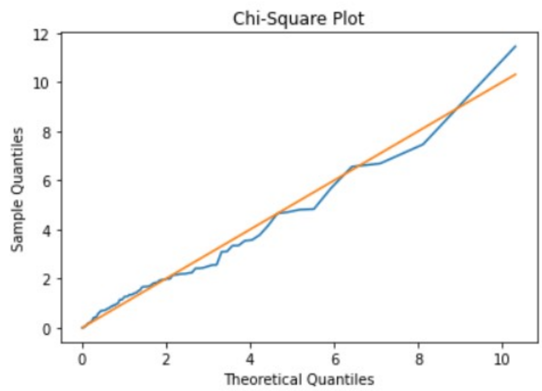
```
In [10]: def covariance_matrix(data):
    mean=mean_vector(data)
    mean_rep = pd.concat([mean]*len(data))
    mean_rep.columns=data.columns
    mean_rep.reset_index(inplace = True, drop = True)
    covariance=(data-mean_rep).transpose().dot(data-mean_rep)/(len(data)-1)
    return covariance
```

We define the following function. First, we make a list to store the squared distances ($d_{(j)}^2$). Since $d_{(j)}^2 = (\underset{\sim}{x}_{(j)} - \bar{\underset{\sim}{x}})' S^{-1} (\underset{\sim}{x}_{(j)} - \bar{\underset{\sim}{x}})$, we use the previously defined functions to obtain $\bar{\underset{\sim}{x}}$ and S^{-1} which correspond to 'mean_vector(data)' and 'S_inv' in the code, respectively. Using a for-loop, all values of $d_{(j)}^2$ are stored in the list. Similar to the function defined in problem #1, we sort the squared distances, obtain the probability values $\frac{j-\frac{1}{2}}{n}$, and then obtain the quantiles of χ_2^2 distribution. Finally, we plot $(q_{(j)}, d_{(j)}^2)$ as well as the reference line. Note that the line through the origin with slope 1 was chosen.

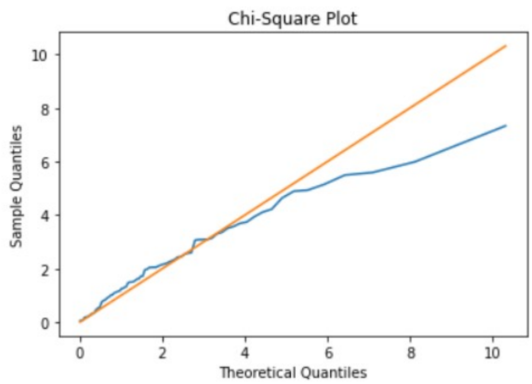
```
In [11]: def chisq_plot(data): #assume input is a dataframe
    dj=[]
    #sample covariance
    S_inv=pd.DataFrame(np.linalg.inv(covariance_matrix(data)), index=data.columns, columns=data.columns)
    #fill in values for dj
    for i in range(len(data)):
        diff=data.iloc[i]-mean_vector(data)
        val=((diff).dot(S_inv)).dot(diff.transpose())
        dj=np.append(dj,val)
    dj.sort()
    j=np.array(range(len(data)))+1
    below_dj=(j-1/2)/len(data)
    qj=chi2.ppf(below_dj, df=2)
    plt.plot(qj,dj) #plot Chi-Square Plot
    plt.plot(qj, qj) #plot reference line
    plt.title('Chi-Square Plot')
    plt.xlabel('Theoretical Quantiles')
    plt.ylabel('Sample Quantiles')
```

4. Using the code in #3, we obtained the Chi-Square plots for the three variable pairs (X1, X2), (X_1, X_3), (X_2, X_3) from the data 'college.dat'.

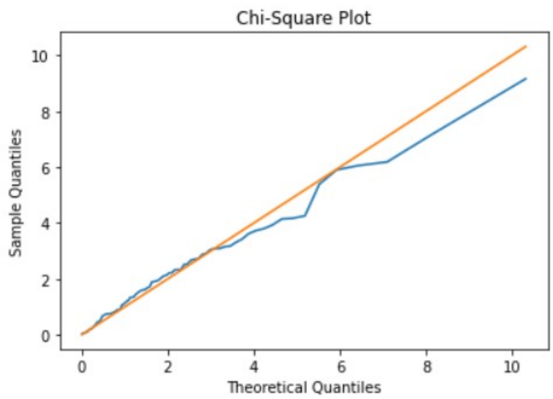
```
In [13]: chisq_plot(college[['X1','X2']])
```




```
In [14]: chisq_plot(college[['X1','X3']])
```



In [15]:
```
chisq_plot(college[['X2','X3']])
```



If the data is from a multivariate normal distribution, each variable pair should have a bivariate normal distribution. Among the Chi-Square plots of each variable pair, the plot for (X_1, X_2) has the most resemblence to the reference line. Thus, it is the closest to a bivariate normal distribution. However, the other two plots deviate from the reference line, so it is difficult to say that they have a bivariate normal distribution. So we can conclude that the data does not have a multivariate normal distribution.