The George Washington University

Laboratory Five:

Regression Models for Time Series Analysis

Fernando Zambrano

DATS 6450: Multivariate Modeling

Dr. Reza Jafari

28 February 2020

Abstract:

Perform linear regression applying the least square error method to predict how much tip to leave based on the amount of a restaurant bill. Creating custom Python functions to calculate the coefficients of the normal equation through matrix multiplication on a training dataset. The derived normal equation is then applied on the testing dataset to estimate predictions and gather forecast errors. The accuracy and performance of the model of the custom function are compared to those of a statistical package, stats.model.

Introduction:

Linear regression is calculating a line that minimizes the total error or distance between the line and each point when comparing the variable being predicted and the variable used for prediction. This subsequently captures the linear relationship between the two variables. Any point that lies on the regression line is then used as a prediction. Since the expectation is that the relationship will remain constant over time. At first this may seem trivial by just drawing a line that "looks" like it minimizes the error. But some forecasts require high precision and accuracy when stakes are high such as stock markets. Also, the computation becomes intensive as more variables and observations are added. Understanding and applying the concepts of matrix multiplication can make regression calculation more efficient.

Methods & Theory:

The fundamental concept to keep in mind for regression is the idea of linear model or linear equation that takes the values of certain variables to then predict what the value of just one other should be based upon the inputs. The name of this equation is called the normal equation which is below.

$$y_t = \beta_0 + \beta_1 x_t + \epsilon_t$$

The breakdown of this equation is as follows:

Yt is the desired prediction, B0 is the coefficient for not having a predicting variable, so it is just as effective as the mean of the predicted variable. When there are predicting variables, each variable Xi is assigned a coefficient Bi. et is the amount of deviation between actual points and the fitting line. The smaller these deviations become the lower the expected error will between actual values and predictions. Calculated errors are squared to removed signage and then summed to get the total squared error, or SSE. SEE generalizes the total error of the fitted line. The lower the SSE, the more accurate the model. Hence, this technique is called Least Squares Estimation (LSE).

The difficult part about regression is choosing the right coefficients for each variable to minimize SSE. This can be solved by rewriting the linear model as a system of linear equations, where the predicted variable, Y, the predicting variable(s) X, and the coefficients B, are written as matrices and vectors.

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

$$\boldsymbol{Y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_T \end{pmatrix} \boldsymbol{X} = \begin{pmatrix} 1 & x_{1,1} & x_{2,1} & \cdots & x_{k,1} \\ 1 & x_{1,2} & x_{2,2} & \cdots & x_{k,2} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{1,T} & x_{2,T} & \cdots & x_{k,T} \end{pmatrix}$$

$$\boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_T \end{pmatrix} \boldsymbol{\epsilon} = \begin{pmatrix} \epsilon_0 \\ \epsilon_1 \\ \vdots \\ \epsilon_T \end{pmatrix}$$

Since Y and X are known, the equation can be re-written to solve for B.

$$\boxed{\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y}}$$

Once the normal equation has been calculated, it needs be evaluated. This is done by calculating the coefficient of determination, or R-squared (R2). R2 is the squared correlation between the predicted variable and its regression predictions. R2 measures how well the model explains the future predictions based upon their variability. A model with R2 equal to 1.0 perfectly captures the variation in Y, whereas if R2 is equal to 0 fails to explain any variation. However, there are limits to the scope which R2 is reliable. Since R2 increases as more variables are added to the model without increasing performance. So a model with an R2 equal to 0.99 with several variables may be worse than a model with an R2 equal to 0.79, but with only two variables. To contemplate for this drawback in R2, the adjusted R2 was developed to penalize models with several variables. The adjusted R2 only increases if added variables improve the model or decreases if their addition does not improve the model. The formula for adjusted R2 is below.

$$\overline{R}^2 = 1 - (1 - R^2)\frac{T - 1}{T - k - 1}$$

There are two other methods the can be used to evaluate the model by assessing the predicting variables themselves. The t-test gives insight into how well individual predictors contribute to the model. All variables start off with the hypothesis that their coefficient is equal to zero. This means that there needs to be a significant amount of evidence to prove that the predictor variable is better than the average of the predicting variable. If the p-value is less than 0.05, then the predictor variable should be kept in the model.

When evaluating several predictors at the same time, the F-test can be used. The F-test assumes that the best model only uses the intercept or the mean of the predicting variable. If, the p-value

for this test is less than 0.05, then there is significant amount of evidence to suggest that model including predictors are better than the intercept only.

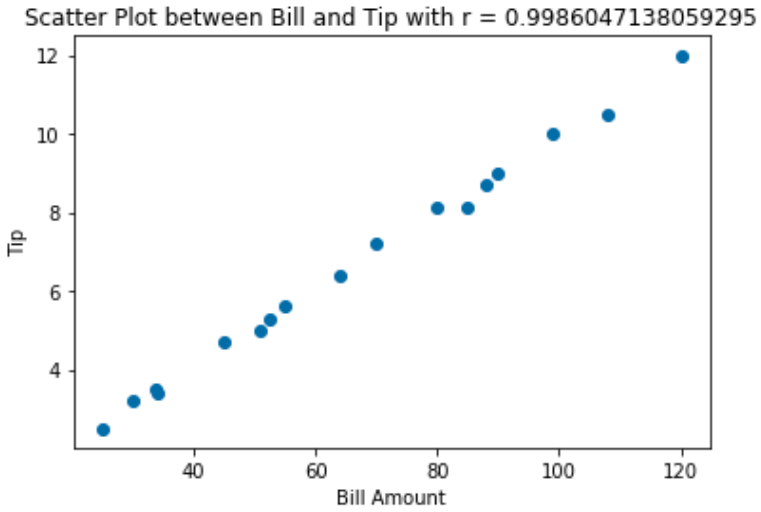
Implementation and Results:

1. Load the Bill-Tip dataset in Pandas.

2. The data is then split into a training and test set using the package Scikit-learn. The training set is about 80% of the entire data meanwhile the test set is about 20%.

```
The traing set for Bill (x-train) amount is : [ 34. 108.  64.  88.  99.  51.  70.  85.  90.  25.  30.  55.  80.]
The traing set for Tip (y-train) amount is : [ 3.4 10.5  6.4  8.7 10.   5.   7.2  8.1  9.   2.5  3.2  5.6  8.1]
The traing set for Bill (x-test) amount is : [120.   45.   52.5  33.7]
The traing set for Tip (y-test) amount is : [12.   4.7  5.3  3.5]
```

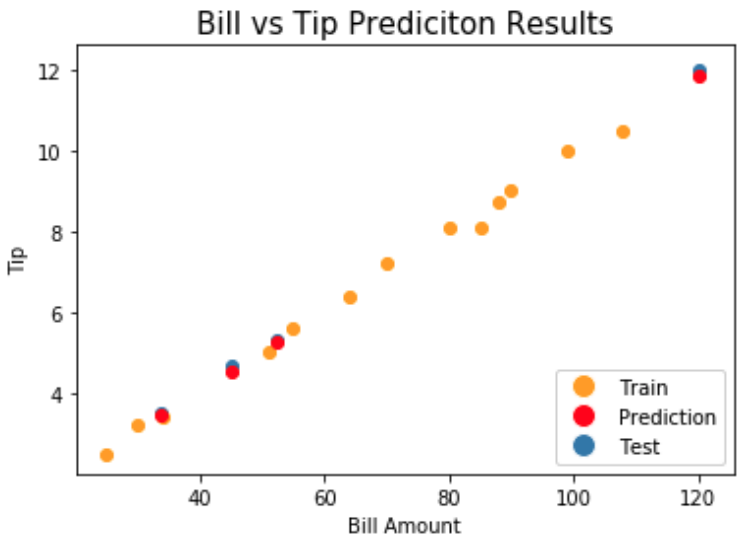
3. A scatterplot comparing bill vs tip along with their correlation coefficient. Given the strong correlation between amongst the training set of Bill and Tip, linear regression will probably be a good estimator because when there is a correlation, a relationship can be captured and modeled. Below is the scatter plot.



4. Using the LSE method and matrix multiplication, the coefficients for the linear regression equation are below.

```
B0 and B1:
 [[0.16417074]
 [0.09734446]]
The intercept for this linear regression model is:  0.16417074449492453
The slope for this linear regression model is:  0.09734445997902831
```

5. The regression equation is applied to the test set to generate predictions. The results are plotted below.



6. Calculate the forecast errors by subtracting the predicted values from the test set.

7. Once the forecast errors are calculated they can be used to further calculate the root mean squared error, or the RMSE. The RMSE is below.

```
The RMSE for prediction is:  0.11368058078991904
```

8. Calculating the mean of the forecast errors can give insight into the bias of an estimator. In this case, the linear regression. Since the mean error is 0.098 it is in a grey area if it is a good estimator or not. Good and unbiased estimators have a mean error as close as possible to zero.

```
Mean of the predictions: 0.09759716882209712
```

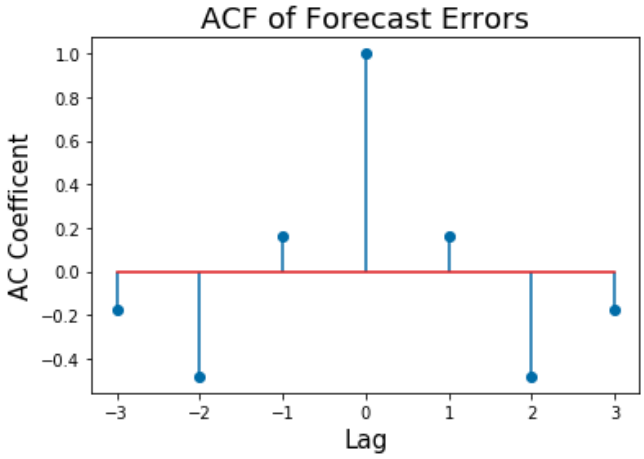
9. The standard error of the equation demonstrates the average distance that the actual values fall from the fitted line. It represents how wrong the regression model is on average using the units of the response variable.

```
Standard Error:  0.16076861913155382
```

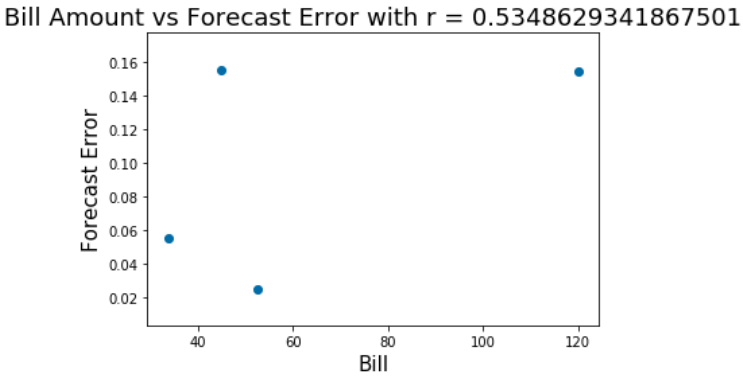
10. The coefficient of determination, or the R2 illustrates how much of the variation in tips can be explained by the variation in restaurant bill. In other words, it quantifies how well restaurant tabs can help explain tip amounts. In this particular case R2 and adjusted R2 are high which means it does great job of capturing the variation in tip amounts.

```
R2 is:  0.9810396591013221
Adjusted R2:  0.9668194034273137
```

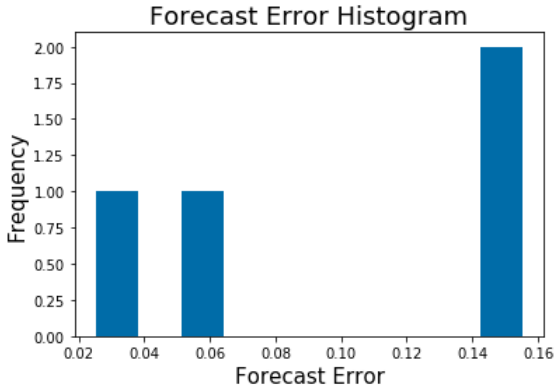
11. The autocorrelation function of the forecast errors is below. It shows that there is little correlation among the forecast errors.



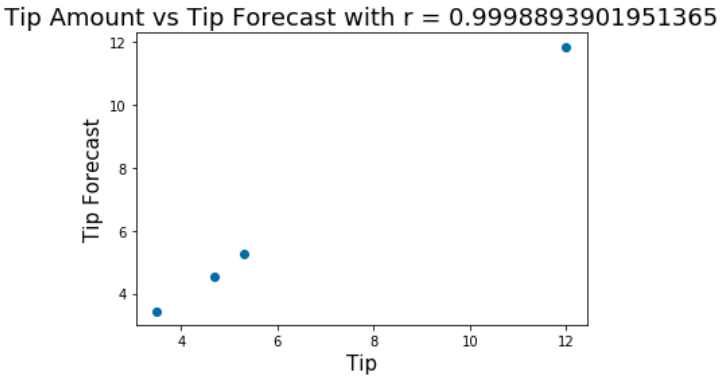
12. Plotting the forecast errors with the bill amount, there is a moderate correlation between the two sets with r = 0.535. Comparing these two sets does not give credible insight into the accuracy of the model.



13. Plotting the histogram of the forecast errors, it is clear to see that the errors are not normally distributed. This is a result of the small number of forecast errors.



14. Scatter plot between actual and predicted. Given the strong correlation between the two datasets, bill amount is a good predictor for tip amount.



15. Finding the 95% confidence interval using the equation in the methods section, the interval for each prediction is below:

```
Confidence interveal for  11.845505941978322  + - 0.44232005765017085
11.845505941978322  +  0.44232005765017085 =  12.287825999628494
11.845505941978322  -  0.44232005765017085 =  11.40318588432815
X star: (1, 2)
X star T: (2, 1)
Confidence interveal for  4.544671443551199  + - 0.36199747035537794
4.544671443551199  +  0.36199747035537794 =  4.906668913906577
4.544671443551199  -  0.36199747035537794 =  4.18267397319582
X star: (1, 2)
X star T: (2, 1)
Confidence interveal for  5.2747548933939115  + - 0.3555763809646317
5.2747548933939115  +  0.3555763809646317 =  5.630331274358543
5.2747548933939115  -  0.3555763809646317 =  4.91917851242928
X star: (1, 2)
X star T: (2, 1)
Confidence interveal for  3.4446790457881793  + - 0.377662347638467
3.4446790457881793  +  0.377662347638467 =  3.822341393426646
3.4446790457881793  -  0.377662347638467 =  3.0670166981497125
```

16. Calculating Q gives insight into the amount of autocorrelation among the forecast errors. The result for Q and other error statistics are below.

```
The mean of forcast error is =  0.09759716882209712
The SSE of forcast error is = 0.05169309779493324
The variance of forcast error is =  0.0033980670866443803
MSE is of the forcast error is =  0.01292327444873331
The mean of this dataset is:  0.09759716882209712
The total variance for this dataset is:  0.013592268346577521
ACF: [1.0, 0.1593937575575005, -0.4824265996758466, -0.17696715788165396]
ACF of the forcast errors is =  [1.0, 0.1593937575575005, -0.4824265996758466, -0.176967157
88165396]
Q value estimate is =  1.1578366759672363
```

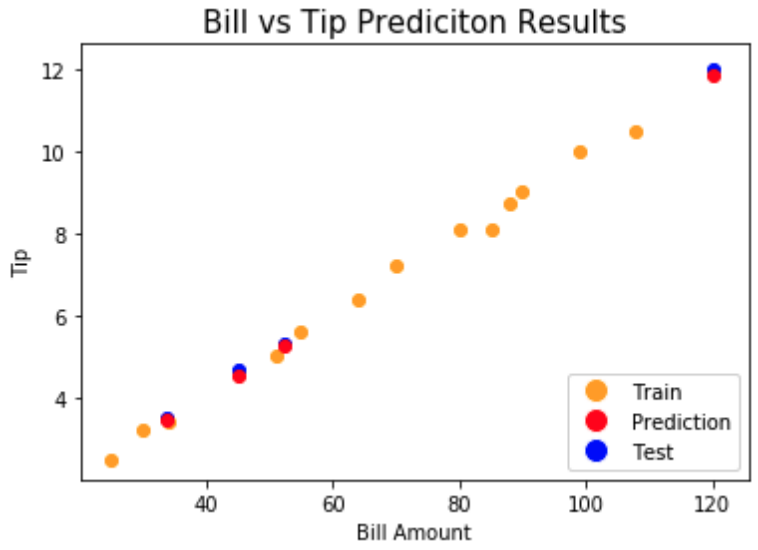
17. Using the stats.model package to run linear regression allows for a quicker method to fit the model to data and calculate the coefficients for the intercept and prediction variables. The package also allows produces a summary which is below. The summary report gives

insight to the strengths and weaknesses of the model by analyzing the t-test, F-test, R, and R2. Given the results of the summary table, the intercept has a high p-value for the test. This means there is enough evidence to support that the intercept should more likely be zero. On the other hand, the p-value for bill amount is really close to zero. This means that there is enough evidence to support that coefficient for bill amount should not be zero. The F-test analyzes all variables in the model. In this case, the p-value is extremely small. This means that the model is statistically better at fitting the data compared to the intercept-only model. Furthermore, the R and R2 values are equal at 0.996. This means that almost the entire variation in tip amount can be explained by the variation or change in the bill amount.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.996
Model:                            OLS   Adj. R-squared:                  0.996
Method:                 Least Squares   F-statistic:                     2914.
Date:                Tue, 25 Feb 2020   Prob (F-statistic):           1.09e-14
Time:                        21:38:55   Log-Likelihood:                 5.7379
No. Observations:                  13   AIC:                            -7.476
Df Residuals:                      11   BIC:                            -6.346
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1642      0.131      1.257      0.235      -0.123       0.452
x1             0.0973      0.002     53.985      0.000       0.093       0.101
==============================================================================
Omnibus:                        0.851   Durbin-Watson:                   2.838
Prob(Omnibus):                  0.653   Jarque-Bera (JB):                0.616
Skew:                          -0.476   Prob(JB):                        0.735
Kurtosis:                       2.521   Cond. No.                         202.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

18. Once a model has been fitted, it stores the coefficients of the regression equation and can then take any input to perform regression and output predictions. The training, testing, and predictions using the package are plotted below.



Conclusion:

Overall, using a dedicated statistical package like stats. model greatly increases computational efficiency for matrix multiplication. Some the most difficult parts in creating custom functions for matrix multiplication is making sure it can take matrices of various shapes since inputs can constantly change as variables are dropped and added. Even though one method is much faster than the other, the results for calculating coefficients for the normal equation are the same. One of the extra benefits of stats.model is the summary report it creates. The report calculates additional information that is important to assess the accuracy and validity of the regression model such as both adjusted R2 and R2, as well as the results of a t-test and F-test.

Appendix:

```
#!/usr/bin/env python
# coding: utf-8


# In[458]:



import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from pandas.plotting import register_matplotlib_converters
from sklearn.model_selection import train_test_split
from matplotlib.patches import Patch
from matplotlib.lines import Line2D


# In[428]:



os.listdir()


# In[429]:
```

```
# 1
# Create a function that calculates the correlation coefficent of x and y variables
def correlation_coefficent_cal(dat1,dat2,x,y):
    # calculate cross_variance between x and y
    # convert list data into numpy arrays
    dat1_mean = np.array(dat1).mean()
    print("mean of " + str(x) +":",dat1_mean)
    dat2_mean = np.array(dat2).mean()
    print("mean of " + str(y) + ":",dat2_mean)
    cross_v = sum((dat1 - dat1_mean)*(dat2-dat2_mean))
    print("Cross variance:", cross_v)
    dat1_sd = np.sqrt(sum(np.square(dat1 - dat1_mean)))
    print("standard devation of " + str(x) + ":",dat1_sd)
    dat2_sd = np.sqrt(sum(np.square(dat2 - dat2_mean)))
    print("standard devation of "+ str(y) + ":",dat2_sd)
    r = cross_v/(dat1_sd * dat2_sd)
    print("The correlation coefficent between " + str(x) + " and " + str(y) + " is:",r)
    return r


# In[430]:


# 1 load the data
df = pd.read_csv("Bill-Tip.csv")


# In[431]:


df


# In[432]:


# 2
```

```
# Create training and testing sets
x_train, x_test, y_train, y_test = train_test_split(df.Bill, df.Tip, test_size = 0.2, shuffle=False)


# In[433]:


print("The traing set for Bill (x-train) amount is :",np.array(x_train))
print("The traing set for Tip (y-train) amount is :",np.array(y_train))
print("The traing set for Bill (x-test) amount is :",np.array(x_test))
print("The traing set for Tip (y-test) amount is :",np.array(y_test))


# In[434]:


# 3
# Plot the scatter plot between Bill vs Tip
# Calcualte r
# Do you think that linear regression will be a good estimator for the set?


# In[435]:


bt_r = correlation_coefficent_cal(df.Bill,df.Tip,"Bill","Tip")


# In[436]:


plt.scatter(df.Bill,df.Tip)
plt.title("Scatter Plot between Bill and Tip with r = {}".format(bt_r))
plt.xlabel('Bill Amount')
plt.ylabel("Tip")
```

 # In[437]:

 # Yes a very good estimator becaue when there is a correlation, a realtionship can be captured and modeled

 # In[ ]:

 # In[438]:

```
# Construct Matrix X and Matrix Y using the training datasets
def matrix(x,y):
    x_ones = np.ones(len(x))
    X = np.vstack([x_ones,x])
    X = X.T
    print(X.shape)
    Y = np.vstack(y)
    print(Y.shape)
    return X,Y
```

 # In[439]:

```
X,Y = matrix(x_train,y_train)
```

 # In[484]:

```
def B_co(x,y):
    print("XT shape:",x.T.shape)
```

```
    print("X shape:",x.shape)
    x_xt = np.mat(x.T) * np.mat(x)
    print("\nXT * X :\n",x_xt)
    print("\nXT * X shape:",x_xt.shape)
    det_x_xt = np.linalg.det(x_xt)
    print("\nDeterminent:\n",det_x_xt)
    x_xtt = np.linalg.inv(x_xt)
    print("\nInverse XT * X:\n",x_xtt)
    print("\nShape Inverse XT: \n", x_xtt.shape)
    xt_x_xt = np.mat(x_xtt) * np.mat(x.T)
    print("\nInverse XT * XT:\n", xt_x_xt)
    B = np.mat(xt_x_xt) * np.mat(y)
    print("\nB0 and B1: \n",B)
    B0 = float(B[0])
    B1 = float(B[1])
    print('The intercept for this linear regression model is: ', B0)
    print('The slope for this linear regression model is: ', B1)
    #return B0,B1
    return B


# In[485]:


B = B_co(X,Y)


# In[442]:


np.array(x_test)


# In[443]:


def normal_eq(x,b):
```

```
    x = np.array(x)
    y_hat = []
    for i in range(0,len(x)):
        yh = float(b[0] + (x[i] * b[1]))
        y_hat.append(yh)
    return y_hat


# In[444]:


y_hat = normal_eq(x_test,B)


# In[445]:


y_hat


# In[481]:


# 5
# Plot train data, test, and prediciton
ax = plt.gca()
plt.scatter(x_train,y_train, color = "orange")
plt.scatter(x_test,y_test,color = "steelblue")
plt.scatter(x_test,y_hat, color = 'red')
plt.title("Bill vs Tip Prediciton Results",fontsize = 15)
plt.xlabel('Bill Amount', fontsize = 10)
plt.ylabel("Tip", fontsize = 10)
legend_elements = [Line2D([0], [0], marker = 'o',color='w',markerfacecolor = 'orange',
                          label='Train',markersize=12),
                   Line2D([0], [0], marker='o', color='w', label='Prediction',
                          markerfacecolor='r', markersize=12),
                   Line2D([0], [0], marker='o', color='w', label='Test',
```

```
                    markerfacecolor='steelblue', markersize=12)]
plt.legend(handles=legend_elements, loc='lower right')
plt.show()


# In[447]:


# 6
# Calcaualte forecast errors
def error_stats(y,yh):
    yt = np.array(y)
    error = []
    for i in range(0,len(yt)):
        ei = yt[i] - yh[i]
        error.append(ei)
    print(error)
    return error


# In[214]:


er = error_stats(y_test,y_hat)


# In[216]:


# 7
# create function to calcaulte RMSE

def rmse(error):
    error = np.array(error)
    rmse = np.sqrt(np.mean(error**2))
    print("The RMSE for prediction is: ",rmse)
    return rmse
```

```
# In[217]:


rmse = rmse(er)


# In[218]:


rmse


# In[219]:


# 8
# Calcualte mean of forecast errors and display as a message
# Is this a bias estimator

def mean_error(error):
    e = np.mean(error)
    print("Mean of the predictions:", e)


# In[220]:


mean_error(er)


# In[221]:


# 9
# Estimate standard error
```

```
# What can be said about this error
def se(error,k):
    er = np.array(error)
    T = len(error)
    den = T-k-1
    sse = np.sum(er**2)
    se = np.sqrt(sse/den)
    print("Standard Error: ",se)
    return se


# In[269]:


se = se(er,1)


# In[223]:


#10
# Calcualte R2 and adjuste r2
# Write down observations
def r2(y,yh,k):
    yt = np.array(y)
    yt_bar = yt.mean()
    yh = np.array(yh)
    yh_bar = yh.mean()
    yt_var = np.sum((yt - yt_bar)**2)
    print(yt_var)
    yh_var = np.sum((yh - yh_bar)**2)
    print(yh_var)
    r2 = yh_var/yt_var
    # Calculate adjusted R2
    T = len(y)
    ar2 = 1 - (1 - r2) * (T -1/T-k-1)
    print("R2 is: ",r2)
```

```
    print("Adjusted R2: ", ar2)
    return r2


# In[224]:


r2(y_test,y_hat,1)


# In[225]:


# 11
# Graph ACF of residuals
def auto_corr(data,lags):
    # convert input data into a numpy array
    data = np.array(data)
    # acf will store the autocorreltion coefficent at each lag interval
    # the first datapoint is always 1.0 since anything correlated with itsself is = 1
    acf = [1.0]
    # calculate the mean for the entire dataset
    y_bar = data.mean()
    print("The mean of this dataset is: ",y_bar)
    # subtract the mean from each observation
    yy_bar = data - y_bar
    # clacualte the total variance for the data set
    total_variance = sum(np.square(yy_bar))
    print("The total variance for this dataset is: ", total_variance)
    # perform a forloop over the dataset with the desired number of lags
    # range is 1,lags b/c the first iteration calcualtes T1
    for i in range(1,lags):
        # first nparray is removing the last element each iteration
        yy_bar_bottom = yy_bar[:-i]
        # second nparray removes the first element each interation
        yy_bar_top = yy_bar[i:]
        # take the sum of of the product of each nparray each iteration
```

```
        yy = sum(yy_bar_top * yy_bar_bottom)
        # divide the sum by total variance and append to resulting acf list
        acf.append(yy/total_variance)
    print("ACF:",acf)
    return acf


# In[226]:


def acf_plot(y):
    #y = y.tolist()
    y_rev = y[::-1]
    y_rev.extend(y[1:])
    print(len(y_rev))
    return y_rev


# In[227]:


er


# In[228]:


acf_error = auto_corr(er,4)


# In[229]:


acfp = acf_plot(acf_error)


# In[230]:
```

```
# Graph the ACF of the forecast errors and justify the predictor accuracy using AFC.
x = np.array(list(range(-3,4)))
figure = plt.stem(x,acfp,use_line_collection=True)
#plt.figure(figsize=(20,20))
plt.xlabel('Lag', fontsize=15)
plt.ylabel('AC Coefficent', fontsize=15)
plt.title('ACF of Forecast Errors',fontsize=18)
plt.show()


# In[231]:


#12

#Plot the scatter plot between forecast errors and predictor variable
#display the correlation coefficient between them on the title.
#Are they related?
#Justify the accuracy of this predictor by observing the correlation coefficient between them.


# In[232]:


r_bfe = correlation_coefficent_cal(x_test,er,"Bill Amount","Forecast Error")


# In[233]:


figure = plt.scatter(x_test,er)
plt.xlabel('Bill', fontsize=15)
plt.ylabel('Forecast Error', fontsize=15)
plt.title('Bill Amount vs Forecast Error with r = {}'.format(r_bfe),fontsize=18)
plt.show()
```

```
# In[234]:


# 13
# Plot the histogram for the forecast errors.
# Is it a normal distribution? Justify your answer.

plt.hist(er)
plt.xlabel("Forecast Error", fontsize=15)
plt.ylabel('Frequency', fontsize=15)
plt.title('Forecast Error Histogram',fontsize=18)
plt.show()


# In[235]:


# 14
# Plot the scatter plot between y-test and ŷ
# display the correlation coefficient between them on the title.
# Justify the accuracy of this predictor by observing the correlation coefficient between y-test and ŷ.

r_yyh = correlation_coefficent_cal(y_test,y_hat,"Tip"," Tip Forecast")


# In[236]:


figure = plt.scatter(y_test,y_hat)
plt.xlabel('Tip', fontsize=15)
plt.ylabel('Tip Forecast', fontsize=15)
plt.title('Tip Amount vs Tip Forecast with r = {}'.format(r_yyh),fontsize=18)
plt.show()
```

 # In[237]:

 # 15

 # Find a 95% prediction interval for this predictor using the following equation

 # In[352]:

```
def prediciton_95(y_hat,se,x):
    # Calculate inverse XT*X
    print("XT shape:",x.T.shape)
    print("X shape:",x.shape)
    x_xt = np.mat(x.T) * np.mat(x)
    print("\nXT * X :\n",x_xt)
    print("\nXT * X shape:",x_xt.shape)
    det_x_xt = np.linalg.det(x_xt)
    print("\nDeterminent:\n",det_x_xt)
    x_xtt = np.linalg.inv(x_xt)
    print("\nInverse XT * X:\n",x_xtt)
    print("\nShape Inverse XT: \n", x_xtt.shape)

    # Select x*
    cil = []
    for i in range(0,len(x)):
        xstar1 = X_t[i].reshape(1,2)
        print("X star:",xstar1.shape)
        xstar2 = xstar1.T
        print("X star T:",xstar2.shape)
        xx = xstar1 * xstar2
        ci = 1.96 * se * np.sqrt(1 + np.mat(xstar1) * np.mat(x_xtt) * np.mat(xstar2))
        print("Confidence interveal for ",y_hat[i]," + -", float(ci))
        print(y_hat[i]," + ",float(ci),"= ",y_hat[i] + float(ci))
        print(y_hat[i]," - ",float(ci),"= ",y_hat[i] - float(ci))
        cil.append(float(ci))
```

```
    return cil


# In[354]:


X_t, Y_t = matrix(x_test,y_test)


# In[355]:


prediciton_95(y_hat,se,X_t)


# In[384]:


# 16
# calcaute Q
# Make ACF plot double sided
def error_stats(mt,x,n,h):
    mean_error = x.mean()
    print("The mean of forcast error is = ", mean_error)
    sse = sum((x) ** 2)
    print("The SSE of forcast error is =", sse)
    variance_error = sum((x - mean_error) ** 2)/n
    print("The variance of forcast error is = ",variance_error)
    mse = sse/n
    print("MSE is of the forcast error is = ", mse)
    acf = auto_corr(x,h)
    print("ACF of the forcast errors is = ", acf)
    Q = 0
    for i in range(1,len(acf)):
        Q += acf[i]**2
    Q = n*Q
    print("Q value estimate is = ", Q)
```

```
    method = mt
    return method,mean_error,sse,variance_error,mse,acf,Q
```

```
# In[385]:
```

```
er = np.array(er)
```

```
# In[386]:
```

```
error_sts = error_stats("OLS",er,len(er),len(y_test))
```

```
# In[396]:
```

```
# In[399]:
```

```
# Part II
```

```
model_fit = sm.OLS(Y,X).fit()
```

```
# In[401]:
```

```
print(model_fit.summary())
```

```
# In[477]:
```

```
y_hato = model_fit.predict(X_t)
```

 # In[478]:

```
y_hato
```

 # In[479]:

```
# Plot train data, test, and prediciton
ax = plt.gca()
plt.scatter(x_train,y_train, color = "orange")
plt.scatter(x_test,y_test,color = "blue")
plt.scatter(x_test,y_hato, color = 'red')
plt.title("Bill vs Tip Prediciton Results",fontsize = 15)
plt.xlabel('Bill Amount', fontsize = 10)
plt.ylabel("Tip", fontsize = 10)
legend_elements = [Line2D([0], [0], marker = 'o',color='w',markerfacecolor = 'orange',
                          label='Train',markersize=12),
                   Line2D([0], [0], marker='o', color='w', label='Prediction',
                          markerfacecolor='r', markersize=12),
                   Line2D([0], [0], marker='o', color='w', label='Test',
                          markerfacecolor='blue', markersize=12)]
plt.legend(handles=legend_elements, loc='lower right')
plt.show()
```

 # In[ ]:

References: