Freanu A. Peria 1710126 January 7, 2019

BSCS CS 302 – Modeling and Simulation CS32FC1 Sir Rey Castillo

Prelim Exam

1.a.

Naïve Forecasting Technique

This forecasting technique simply uses the last period's actual output value as the forecast for a current period's forecast, without any attempt of adjusting or establishing any mathematical improvements. This technique is generated only for comparison to other higher-level forecasting techniques. What I did was that I implemented the technique in python 3. The data that I used, "Peso-Dollar exchange rate in the Philippines", was taken from the site www.data.gov.ph.

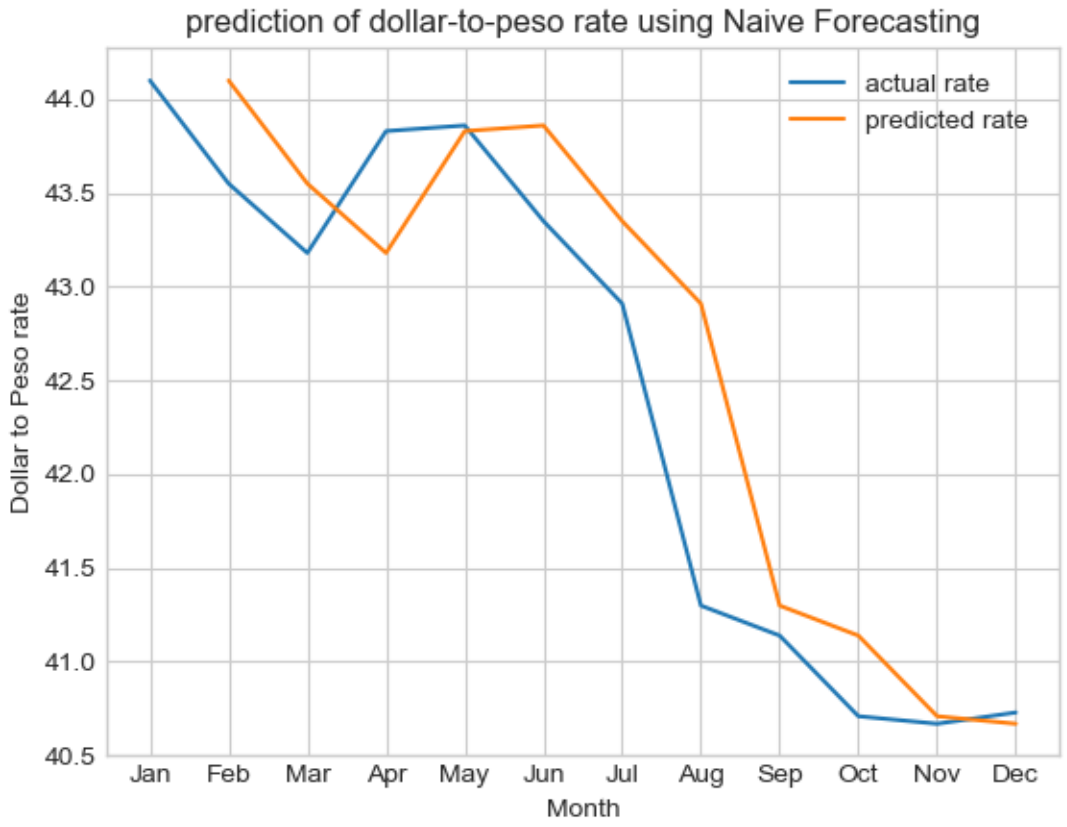
(note: on my subsequent implementation of forecasting techniques, i will be using Numpy and Pandas for manipulating data. Matplotlib is used for plotting data on a graph)

1.a. Source Code:

```
import pandas
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
#      Naive Forecasting
#   this type of forecasting uses the previous data (n-1)
#   to predict the next outcome (n-1)


#take values from the CSV
df = pandas.read_csv("ph_peso_dollar.csv")
print(df)
months = ['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']
#manipulation of data frames
actual_data = list(df.iloc[1:13,2])
predicted = np.zeros(12)
#naive forecast
ctr = 1
for x in actual_data:
    try:
        predicted[ctr] = x
        ctr += 1
    except:
        pass

predicted[0] = None

#plotting in graph
style.use('seaborn-whitegrid')
fig = plt.figure()
plt.plot(months, actual_data, label = "actual rate")
plt.plot(months, predicted, label = "predicted rate ")
plt.xlabel('Month')
plt.ylabel('Dollar to Peso rate')
plt.title('prediction of dollar-to-peso rate using Naive Forecasting')
plt.legend()
plt.show()
```

1.a

Graph



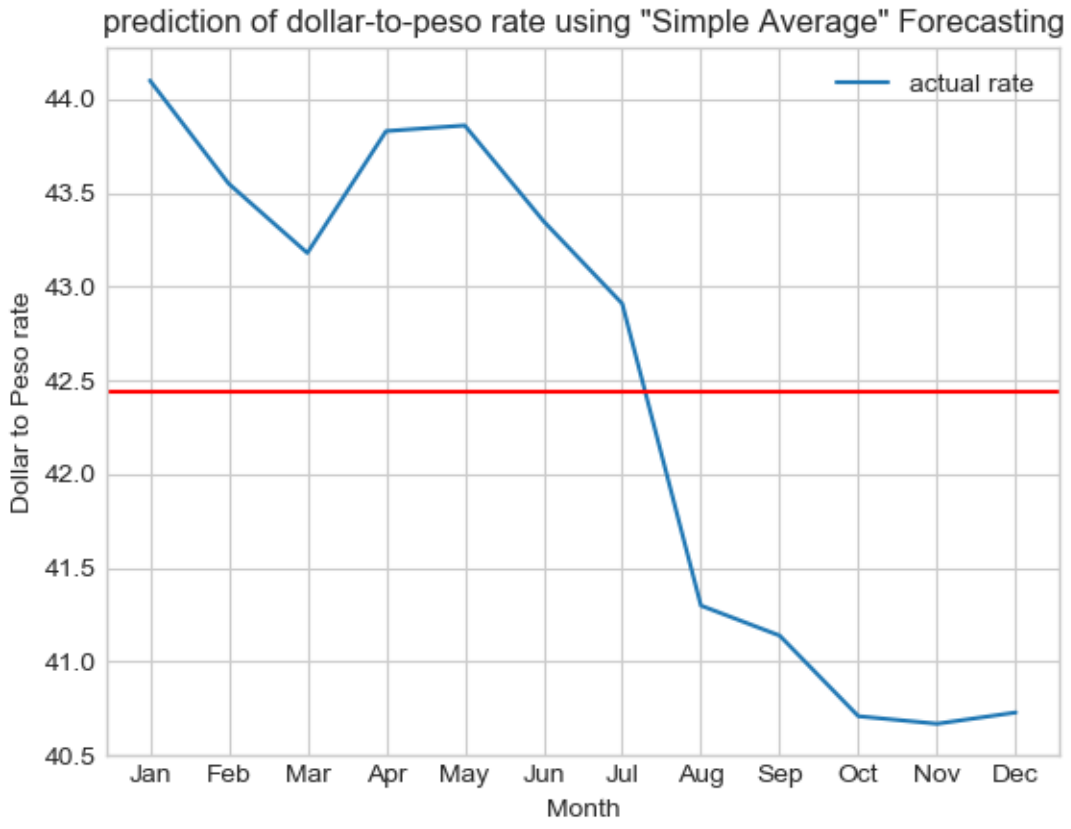
1.b – <u>Simple Average Technique</u>

On this implementation, I simply aggregated all of the data and took its average, and used the average as the forecasted amount, since there I can't find any "Simple Average Forecasting Technique" on the internet and books. I used the same data from the previous forecasting technique.

1.b. – Source Code

```
#implementation of "simple average"
sum = 0
for x in actual_data:
    sum += x
average = sum/len(actual_data)
print(average)

#plotting in graph
style.use('seaborn-whitegrid')
fig = plt.figure()
plt.plot(months, actual_data, label = "actual rate")
plt.axhline(y = average, color = 'r', linestyle='-')
plt.xlabel('Month')
plt.ylabel('Dollar to Peso rate')
plt.title('prediction of dollar-to-peso rate using "Simple Average" Forecasting')
plt.legend()
plt.show()
```

1.b. – Graph



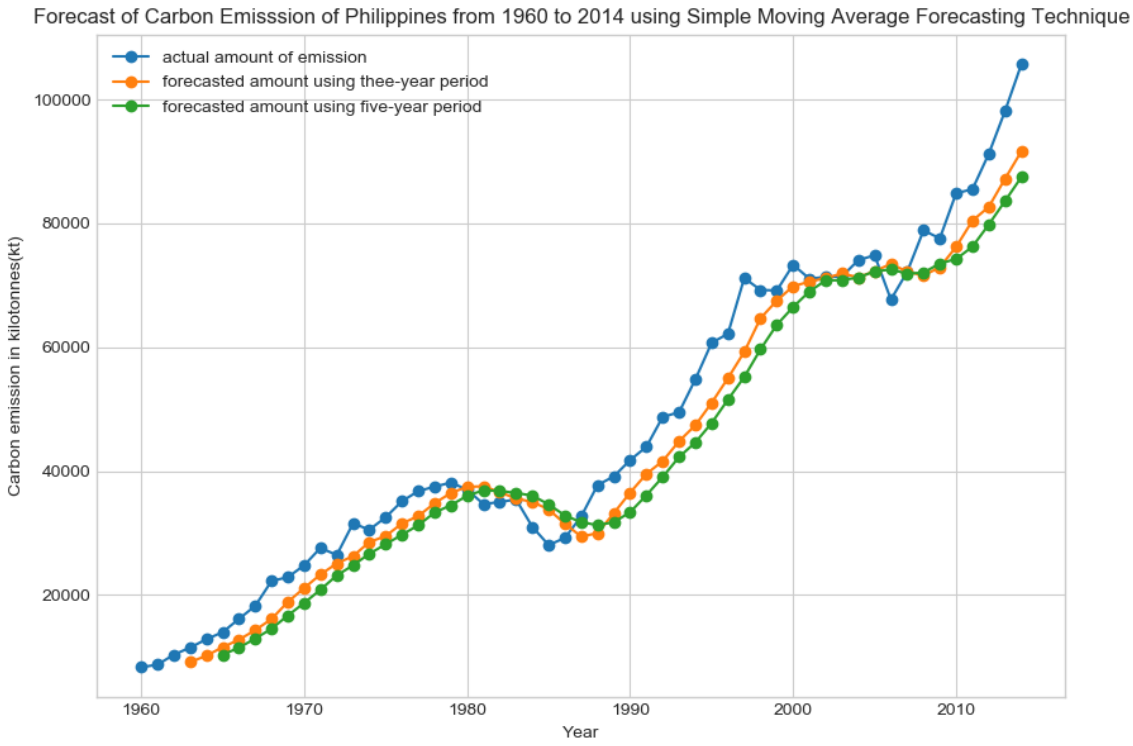
1.c. Moving Average

This forecasting technique uses the n previous values, takes the average of it and uses it as the forecasted value. It reduces the randomness of the forecast by averaging values over a period of time. In my implementation, the data that I used, “Carbon Emission of Countries from 1960 to 2014”, was taken from World Bank Open Data (https://data.worldbank.org/). I used three-year and five-year period for my implementation, resulting in two lines of forecast. The line of the five-year period is more sensitive to sudden changes in trend in the actual data. It is quite so, as seen on the graph.

1.c. Source Code

```
from matplotlib import style

#Simple Moving Average

#take values from the CSV (note that the CSV is comma-delimited)
df = pandas.read_fwf("worldbank_co2emission/API_EN.ATM.CO2E.KT_DS2_en_csv_v2_10224872.csv")
data =df.iloc[187]              #take the data in the 187th row
s = data[0]                     #put to string the 0th element in data list
s = s.replace('"','')           #remove the double quotes in every element
actual_data = s.split(',')      #make a list, 'actual data', by splitting the string s by the commas

for x in range(4):
    actual_data.pop(0)          #remove the first four elements, which is just some useless labels
for x in range(4):
    actual_data.pop(55)         #remove the trailing empty strings
actual_data = list(map(float, actual_data))     #convert the strings to numbers (the numbers in the list are enclosed by '')
print(actual_data)
print(len(actual_data))
#make a list for the years
years = [ i for i in range(1960, 2015) ]



#Simple moving average forecast - 3 year period
sma_three_year_period = np.zeros(55)
for index, value in enumerate(actual_data):
    if index in [0,1,2]:
        sma_three_year_period[index] = None
    else:
        sma_three_year_period[index] = (actual_data[index - 1] + actual_data[index-2] + actual_data[index-3])/3


#Simple moving average forecast - 5 year period
sma_five_year_period = np.zeros(55)
for index, value in enumerate(actual_data):
    if index in [0,1,2,3,4]:
        sma_five_year_period[index] = None
    else:
        sma_five_year_period[index] = (actual_data[index-1] + actual_data[index-2] + actual_data[index-3] +
                actual_data[index-4] + actual_data[index-5])/5


#plotting in graph
style.use('seaborn-whitegrid')
fig = plt.figure()
plt.plot(years, actual_data, marker = 'o',label = "actual amount of emission")
plt.plot(years, sma_three_year_period, marker = 'o', label = "forecasted amount using thee-year period")
plt.plot(years, sma_five_year_period, marker = 'o',label = "forecasted amount using five-year period")
plt.xlabel('Year')
plt.ylabel('Carbon emission in kilotonnes(kt)')
plt.title('Forecast of Carbon emisssion of Philippines from 1960 to 2014 using Simple Moving Average Forecasting Technique')
plt.legend()
plt.show()
```

1.c. Graph



1.d. <u>Weighted Average</u>

Like the moving average forecasting, it relies on the preceding values in order for it to forecast the current value. In this technique, for every actual value that contributes to the forecasted value, it has its own bearing depending on how far it is from the forecasted value. That is, it will give more "weight" to the most recent value. In my implementation, I took values in a four-year period, and I distributed the alpha as shown:

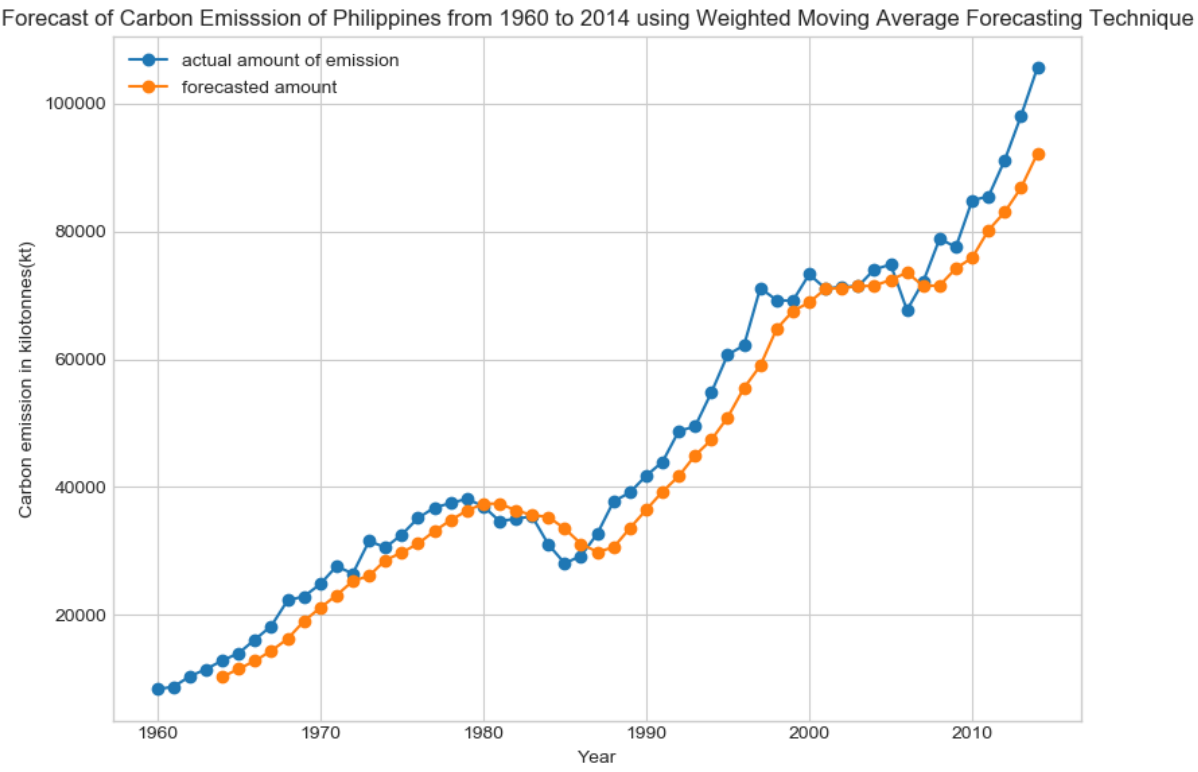
	1	2	3	4
Alpha	0.4	0.3	0.2	0.1

1.d Source Code

(I removed the block of code that dealt with processing data and plotting)

```
#Weighted moving average forecast
wma = np.zeros(55)
for index, value in enumerate(actual_data):
    if index in [0,1,2,3]:
        wma[index] = None
    else:
        wma[index] = (
                (actual_data[index-1]*0.4) + (actual_data[index-2]*0.3 ) +
                (actual_data[index-3]*0.2) + (actual_data[index-4]*0.1)
                )
```

1.d. Graph



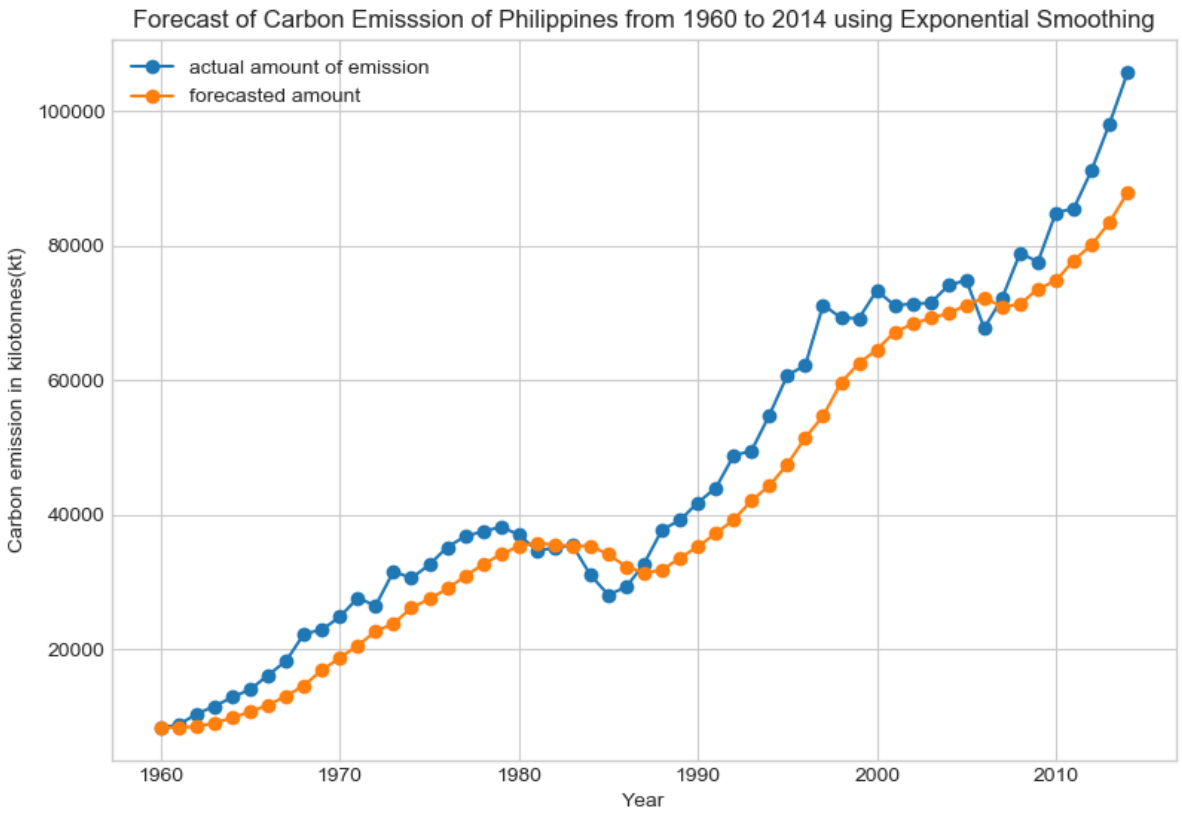
1.e. Exponential Smoothing

Exponential Smoothing technique uses the previous actual value and forecasted value in order to forecast the current value. The actual and forecasted value will be affected by the alpha, which is between 0 and 1. It uses the formula Ft+1 = Ft + α(At – Ft). The forecast of Exponential Smoothing usually starts at the third term, since it assumes that the first predicted value = actual value, then carried over to the second term. Notice that the forecasting technique “smooths out” the line of the actual data.

1.e. Source Code

```
1
2   #Exponential smoothing
3   alpha = 0.3
4   exponential_smoothing = np.zeros(55)
5   for index, value in enumerate(actual_data):
6       if index == 0:
7           exponential_smoothing[index] = actual_data[index]
8       else:
9           exponential_smoothing[index] = actual_data[index-1]*alpha + exponential_smoothing[index-1]*(1-alpha)
10
```

1.e. Graph



1.f. Linear Trend Line

This forecasting technique is as simple as a linear equation; it aggregates all of the actual value and uses it to formulate the slope and the y-intercept of the line. It uses the formula Y = a + bt , where

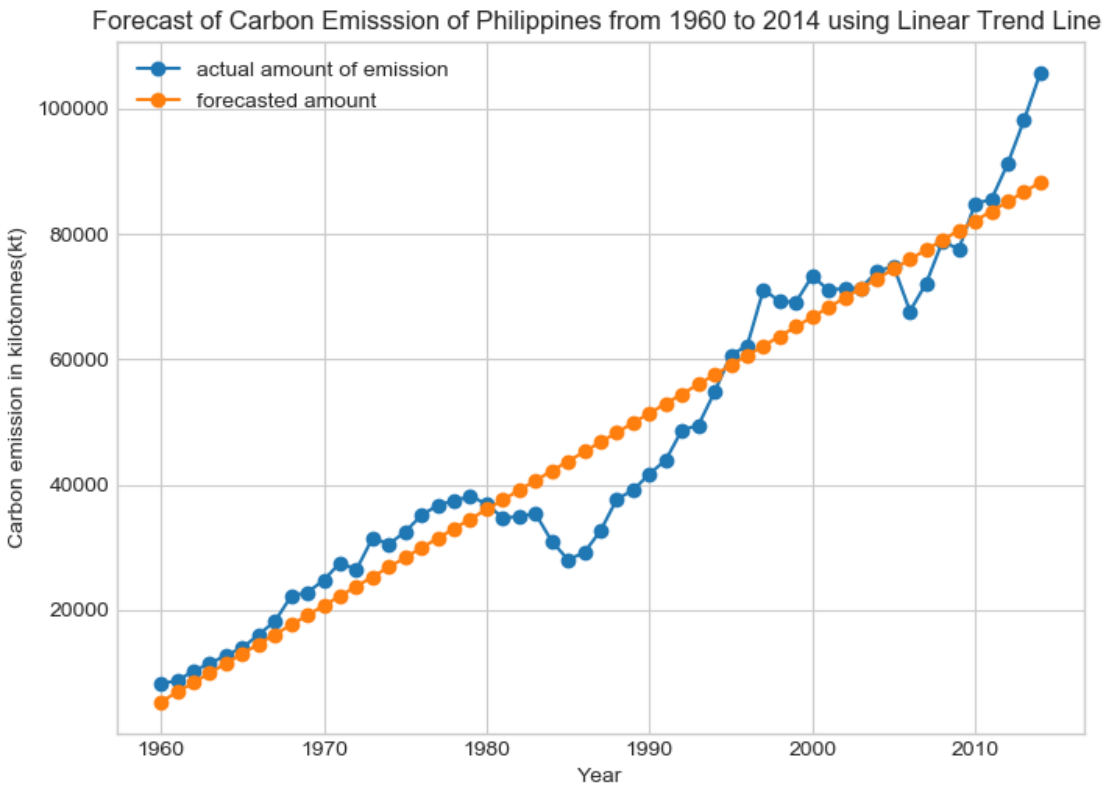
$$b = \frac{n\sum(ty) - \sum t\sum y}{n\sum t^2 - (\sum t)^2}$$

$$a = \frac{\sum y - b\sum t}{n}$$

1.f. Source Code

```
1
2   yrs = np.zeros(len(years))
3   num = 1
4   for index, value in enumerate(yrs):
5       yrs[index] = num
6       num += 1
7
8   #taking the slope (y = a + bt)
9   #taking the summation values...
10  sum_xtimesy = 0
11  for index, value in enumerate(yrs):
12      sum_xtimesy += yrs[index] * actual_data[index]
13  #
14  sum_x = 0
15  for x in yrs:
16      sum_x += x
17  #
18  sum_y = 0
19  for x in actual_data:
20      sum_y += x
21  #
22  sum_x_squared = 0
23  for x in yrs:
24      sum_x_squared += x*x
25  #
26  print(sum_xtimesy,sum_x,sum_y,sum_x_squared)
27
28  numerator = (55* sum_xtimesy) - sum_x*sum_y
29  denominator =  (55* sum_x_squared) - (sum_x*sum_x)
30  b = numerator/ denominator
31  print(b)
32
33  a = (sum_y - (b*(sum_x)))/55
```

1.f. Graph



1.g. <u>Forecasting Seasonality Technique</u>

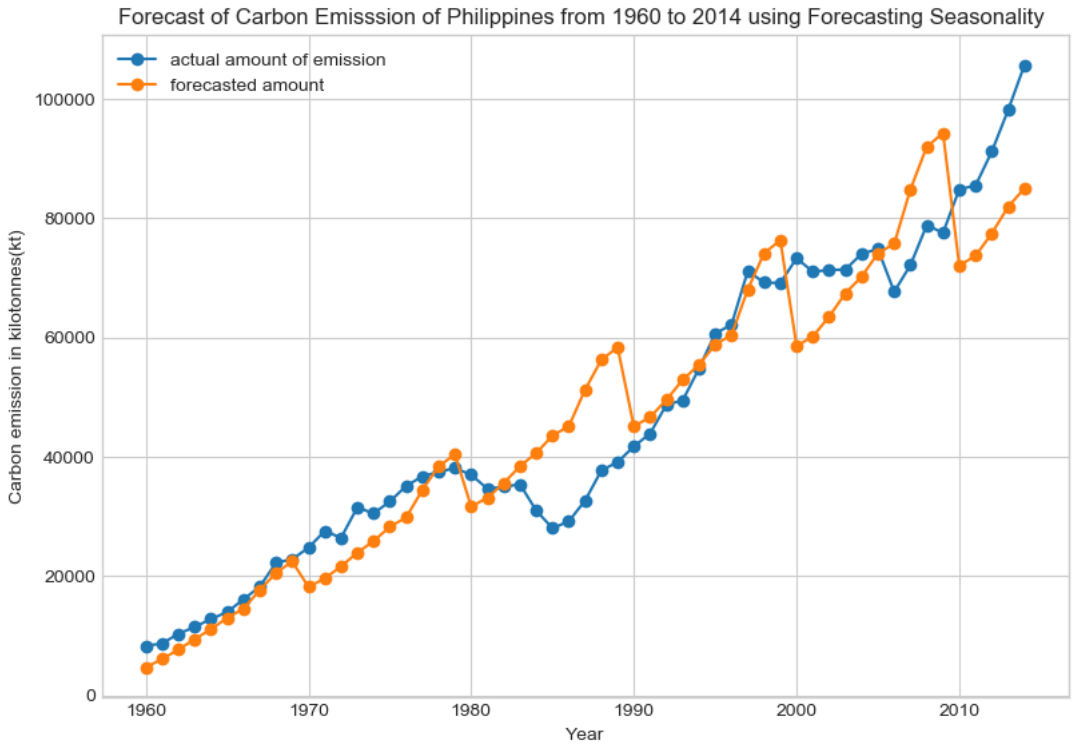
Again, I was not able to find "Forecasting Trend" and "Forecasting Seasonality" in my search. What I have learned from my study was that, you can take into account while implementing the linear trend line the seasonality of a dependent variable depending on the period, thus *forecasting the trend and seasonality*. In my implementation, I have chosen to observe the data by decades, thus having five cycles of ten-year periods. For every nth year, I took the sum and averaged it. Then each yearly average is then divided by the grand average (average of all the yearly average) – these values shall be used to adjust the value forecasted by the linear trend line. The implementation of the Forecasting Seasonality Technique was not quite effective, as shown in the graph; perhaps observing the data by decade is a wrong decision.

1.g. Source Code

```
1
2   yrs = np.zeros(len(years))
3   num = 1
4   for index, value in enumerate(yrs):
5       yrs[index] = num
6       num += 1
7   #taking the slope (y = a + bt)
8   #taking the summation values...
9   sum_xtimesy = 0
10  for index, value in enumerate(yrs):
11      sum_xtimesy += yrs[index] * actual_data[index]
12  #
13  sum_x = 0
14  for x in yrs:
15      sum_x += x
16  #
17  sum_y = 0
18  for x in actual_data:
19      sum_y += x
20  #
21  sum_x_squared = 0
22  for x in yrs:
23      sum_x_squared += x*x
24  #
25  numerator = (55* sum_xtimesy) - sum_x*sum_y
26  denominator =  (55* sum_x_squared) - (sum_x*sum_x)
27  b = numerator/ denominator
28  print(b)
29  #taking the y - intercept (y= a + bt)
30  a = (sum_y - (b*(sum_x)))/55
31  print(a)
32  #taking the seasonal relatives (every decade, for half a century)
33  cycle1 =[]
34  cycle2 =[]
35  cycle3 =[]
36  cycle4 =[]
37  cycle5 =[]
38  for index,value in enumerate(actual_data):
39      if 0<=index<=9:
40          cycle1.append(value)
41      elif 10<=index<=19:
42          cycle2.append(value)
43      elif 20<=index<=29:
44          cycle3.append(value)
45      elif 30<=index<=39:
46          cycle4.append(value)
47      elif 40<=index<=49:
48          cycle5.append(value)
49      else:
50          pass
```

```
    elif 40<=index<=49:
        cycle5.append(value)
    else:
        pass
averages = {
    1:0, 2:0, 3:0, 4:0, 5:0,
    6:0, 7:0,8:0, 9:0, 10:0
}
for x in [cycle1,cycle2,cycle3,cycle4,cycle5]:
    for index, value in enumerate(x):
        averages[index+1] += value

wantuten = [1,2,3,4,5,6,7,8,9,10]
for x in wantuten:
    averages[x] = averages[x]/5
grand_average = 0
for x in wantuten:
    grand_average += averages[x]
grand_average = grand_average/10
for x in wantuten:
    averages[x] = averages[x]/grand_average
print(averages)

#combining the linear trend line and seasonality
forecasting_seasonality = np.zeros(55)
for index, value in enumerate(forecasting_seasonality):
    forecasting_seasonality[index] = a + b * yrs[index]

for index, value in enumerate(forecasting_seasonality):
    forecasting_seasonality[index] = forecasting_seasonality[index] * averages[wantuten[index%10]]

```

1.g. Graph



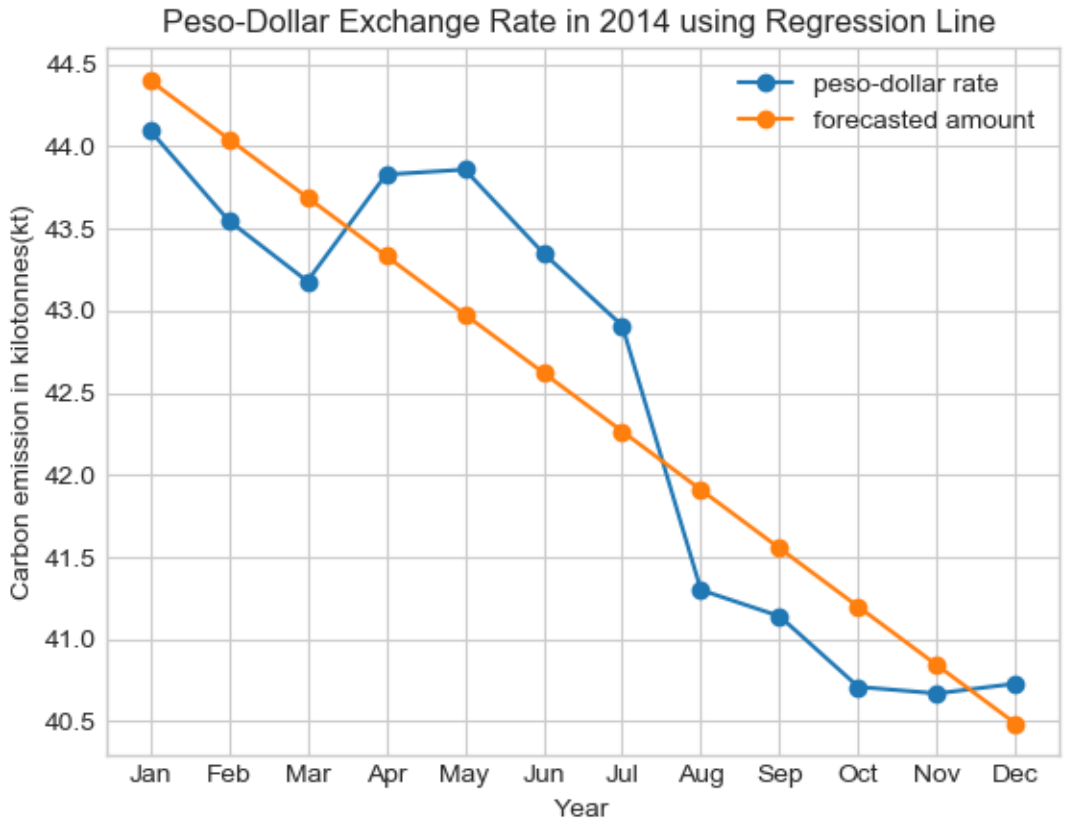
2.a. <u>Linear Regression</u>

Linear Regression is simply a line that forecasts value of a dependent variable based on the independent variable. It explains the relation and causality between two variables that are related (i.e. healthcare and mortality rate, poverty and crime rate). In my model, I used the same data set from the first forecasting technique, assuming that the peso-dollar rates are dependent on the month of the year, possibly because of Filipinos who work as OCW bringing home Foreign currencies, thus adjusting the value of the Philippine Peso.

2.a. Source Code

```
#take values from the CSV
df = pandas.read_csv("ph_peso_dollar.csv")
print(df)
months = ['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']
#manipulation of data frames
actual_data = list(df.iloc[1:13,2])
predicted = np.zeros(12)

mnths = [1,2,3,4,5,6,7,8,9,10,11,12]
#making the regression line
sum_xtimesy = 0
for index, value in enumerate(mnths):
    sum_xtimesy += mnths[index] * actual_data[index]
#
sum_x = 0
for x in mnths:
    sum_x += x
#
sum_y = 0
for x in actual_data:
    sum_y += x
#
sum_x_squared = 0
for x in mnths:
    sum_x_squared += x*x
#
print(sum_xtimesy,sum_x,sum_y,sum_x_squared)

numerator = (12* sum_xtimesy) - sum_x*sum_y
denominator =  (12* sum_x_squared) - (sum_x*sum_x)
b = numerator/ denominator
print(b)

a = (sum_y - (b*(sum_x)))/12
print(a)
predicted = []
for x in mnths:
    predicted.append(x*b + a)

#plotting in graph
style.use('seaborn-whitegrid')
fig = plt.figure()
plt.plot(months, actual_data, marker = 'o',label = "peso-dollar rate")
plt.plot(months, predicted ,marker = 'o', label = 'forecasted amount')
plt.xlabel('Year')
plt.ylabel('Carbon emission in kilotonnes(kt)')
plt.title('Peso-Dollar Exchange Rate in 2014 using Regression Line ')
plt.legend()
plt.show()
```

2.a. Graph


Peso-Dollar Exchange Rate in 2014 using Regression Line
peso-dollar rate
forecasted amount
Carbon emission in kilotonnes(kt)
44.5
44.0
43.5
43.0
42.5
42.0
41.5
41.0
40.5
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Year