Data Science Assignment (Final)

August 26, 2019

1 Data Science Assignment: Sales Prediction Model for Mamonde, a Skincare Company: Introduction

1.1 Company Background

Our company name is called Mamonde. It means combination of "my" and "world". It is launched by Amore Pacific which is a corporation of South Korea and cosmetics conglomerate. It has launched its first store in Malaysia which is in 2016. It is so famous as it used natural ingredient and it had the smell of flowers such as jasmine, camellia and lotus which every girl will like it. Mamonde has wide range of skincare and make-up product.

1.2 Problem Statement

Mamonde has requested us to analyse the sales of skincare products in different age groups, particularly in Setapak. They would want to understand the trend of sales of their products in Setapak. They would like to understand how their products fare in Setapak, and what they can do in order to improve their sales.

1.3 Project Objectives

In order to achieve the following aims, we have come up with several objectives where we need to achieve:

1) To visualise the data so that the data can be further understood and analysed,
2) Create a model so the trend of the sales of products can be replicated in the best manner, with decent accuracy,
3) Predict the sales for the next few months of the products.

1.4 Project Benefits

After the implementation of this project, Mamonde will be able to:

1) Understand the current sales situation of their products in Setapak,
2) Able to make better decisions in terms of stock production and storage, as they can know a predicted value of sales for the next few months,
3) Analyse which products that do not sell well in Setapak, and find ways in order to clear the remaining stock.

2 Data understanding

2.1 Raw Data

In order to achieve the following objectives, data is scrapped and mined from various sources. We have obtained the data of the products from the website of the company, while the sales data is created via a random data generator. The data has been inputed into one big dataframe. The following code imports the necessary packages, and calling the dataframe created.

```
In [2]: #import the main csv first
        import pandas as pd
        import matplotlib.pyplot as plt
        import numpy as np
        data = pd.read_csv("Main_table.csv")
        #display whole table
        data
```

```
Out[2]:        Month  Product Code         Product Name Category  \
        0     Jan-08             1     Rose Water Toner    Toner
        1     Feb-08             1     Rose Water Toner    Toner
        2     Mar-08             1     Rose Water Toner    Toner
        3     Apr-08             1     Rose Water Toner    Toner
        4     May-08             1     Rose Water Toner    Toner
        5     Jun-08             1     Rose Water Toner    Toner
        6     Jul-08             1     Rose Water Toner    Toner
        7     Aug-08             1     Rose Water Toner    Toner
        8     Sep-08             1     Rose Water Toner    Toner
        9     Oct-08             1     Rose Water Toner    Toner
        10    Nov-08             1     Rose Water Toner    Toner
        11    Dec-08             1     Rose Water Toner    Toner
        12    Jan-09             1     Rose Water Toner    Toner
        13    Feb-09             1     Rose Water Toner    Toner
        14    Mar-09             1     Rose Water Toner    Toner
        15    Apr-09             1     Rose Water Toner    Toner
        16    May-09             1     Rose Water Toner    Toner
        17    Jun-09             1     Rose Water Toner    Toner
        18    Jul-09             1     Rose Water Toner    Toner
        19    Aug-09             1     Rose Water Toner    Toner
        20    Sep-09             1     Rose Water Toner    Toner
        21    Oct-09             1     Rose Water Toner    Toner
        22    Nov-09             1     Rose Water Toner    Toner
        23    Dec-09             1     Rose Water Toner    Toner
        24    Jan-10             1     Rose Water Toner    Toner
        25    Feb-10             1     Rose Water Toner    Toner
        26    Mar-10             1     Rose Water Toner    Toner
        27    Apr-10             1     Rose Water Toner    Toner
        28    May-10             1     Rose Water Toner    Toner
        29    Jun-10             1     Rose Water Toner    Toner
        ..       ...           ...                  ...      ...
```

690	Jul-15	6	First Energy Essence	Essence
691	Aug-15	6	First Energy Essence	Essence
692	Sep-15	6	First Energy Essence	Essence
693	Oct-15	6	First Energy Essence	Essence
694	Nov-15	6	First Energy Essence	Essence
695	Dec-15	6	First Energy Essence	Essence
696	Jan-16	6	First Energy Essence	Essence
697	Feb-16	6	First Energy Essence	Essence
698	Mar-16	6	First Energy Essence	Essence
699	Apr-16	6	First Energy Essence	Essence
700	May-16	6	First Energy Essence	Essence
701	Jun-16	6	First Energy Essence	Essence
702	Jul-16	6	First Energy Essence	Essence
703	Aug-16	6	First Energy Essence	Essence
704	Sep-16	6	First Energy Essence	Essence
705	Oct-16	6	First Energy Essence	Essence
706	Nov-16	6	First Energy Essence	Essence
707	Dec-16	6	First Energy Essence	Essence
708	Jan-17	6	First Energy Essence	Essence
709	Feb-17	6	First Energy Essence	Essence
710	Mar-17	6	First Energy Essence	Essence
711	Apr-17	6	First Energy Essence	Essence
712	May-17	6	First Energy Essence	Essence
713	Jun-17	6	First Energy Essence	Essence
714	Jul-17	6	First Energy Essence	Essence
715	Aug-17	6	First Energy Essence	Essence
716	Sep-17	6	First Energy Essence	Essence
717	Oct-17	6	First Energy Essence	Essence
718	Nov-17	6	First Energy Essence	Essence
719	Dec-17	6	First Energy Essence	Essence

	Usage	Price	Youth Sales	Middle age sales \
0	Toning, Soothing & Moisturising	56	20	32
1	Toning, Soothing & Moisturising	56	47	33
2	Toning, Soothing & Moisturising	56	11	45
3	Toning, Soothing & Moisturising	56	35	36
4	Toning, Soothing & Moisturising	56	13	28
5	Toning, Soothing & Moisturising	56	7	49
6	Toning, Soothing & Moisturising	56	38	32
7	Toning, Soothing & Moisturising	56	48	36
8	Toning, Soothing & Moisturising	56	6	26
9	Toning, Soothing & Moisturising	56	46	33
10	Toning, Soothing & Moisturising	56	12	22
11	Toning, Soothing & Moisturising	56	44	38
12	Toning, Soothing & Moisturising	56	19	35
13	Toning, Soothing & Moisturising	56	74	9
14	Toning, Soothing & Moisturising	56	28	41
15	Toning, Soothing & Moisturising	56	9	29

```
16    Toning, Soothing & Moisturising      56          24           11
17    Toning, Soothing & Moisturising      56          60            6
18    Toning, Soothing & Moisturising      56          41           28
19    Toning, Soothing & Moisturising      56           7           31
20    Toning, Soothing & Moisturising      56          67           11
21    Toning, Soothing & Moisturising      56          11           38
22    Toning, Soothing & Moisturising      56          73           46
23    Toning, Soothing & Moisturising      56          73           38
24    Toning, Soothing & Moisturising      56          26           20
25    Toning, Soothing & Moisturising      56          36           25
26    Toning, Soothing & Moisturising      56          58           29
27    Toning, Soothing & Moisturising      56          66            6
28    Toning, Soothing & Moisturising      56          43           32
29    Toning, Soothing & Moisturising      56          34           29
..                                ...     ...         ...          ...
690  Wrinkle Improvement & brightening     49          16           43
691  Wrinkle Improvement & brightening     49          35           16
692  Wrinkle Improvement & brightening     49          53           27
693  Wrinkle Improvement & brightening     49          30           36
694  Wrinkle Improvement & brightening     49          76           35
695  Wrinkle Improvement & brightening     49          36            6
696  Wrinkle Improvement & brightening     49          22           51
697  Wrinkle Improvement & brightening     49           8           19
698  Wrinkle Improvement & brightening     49          69           25
699  Wrinkle Improvement & brightening     49          89           41
700  Wrinkle Improvement & brightening     49          69           37
701  Wrinkle Improvement & brightening     49          67           24
702  Wrinkle Improvement & brightening     49          28           10
703  Wrinkle Improvement & brightening     49          61           43
704  Wrinkle Improvement & brightening     49          74            5
705  Wrinkle Improvement & brightening     49          46           44
706  Wrinkle Improvement & brightening     49          45           54
707  Wrinkle Improvement & brightening     49          41            6
708  Wrinkle Improvement & brightening     49          85            2
709  Wrinkle Improvement & brightening     49           5           40
710  Wrinkle Improvement & brightening     49           4           23
711  Wrinkle Improvement & brightening     49          49           19
712  Wrinkle Improvement & brightening     49          67           40
713  Wrinkle Improvement & brightening     49          34           15
714  Wrinkle Improvement & brightening     49          40            7
715  Wrinkle Improvement & brightening     49          77           20
716  Wrinkle Improvement & brightening     49          38           37
717  Wrinkle Improvement & brightening     49          81            9
718  Wrinkle Improvement & brightening     49           9           15
719  Wrinkle Improvement & brightening     49          47           54

     Senior citizen sales  Total Sales  Unnamed: 10
0                       5           57          NaN
```

1	40	120	NaN
2	23	79	NaN
3	31	102	NaN
4	43	84	NaN
5	18	74	NaN
6	30	100	NaN
7	19	103	NaN
8	53	85	NaN
9	30	109	NaN
10	23	57	NaN
11	10	92	NaN
12	13	67	NaN
13	53	136	NaN
14	49	118	NaN
15	44	82	NaN
16	54	89	NaN
17	5	71	NaN
18	46	115	NaN
19	20	58	NaN
20	29	107	NaN
21	32	81	NaN
22	18	137	NaN
23	7	118	NaN
24	54	100	NaN
25	34	95	NaN
26	37	124	NaN
27	11	83	NaN
28	17	92	NaN
29	43	106	NaN
..	...	...	...
690	40	99	NaN
691	55	106	NaN
692	48	128	NaN
693	29	95	NaN
694	39	150	NaN
695	11	53	NaN
696	37	110	NaN
697	33	60	NaN
698	52	146	NaN
699	30	160	NaN
700	26	132	NaN
701	36	127	NaN
702	40	78	NaN
703	20	124	NaN
704	49	128	NaN
705	17	107	NaN
706	36	135	NaN
707	24	71	NaN

```
708                            28        115        NaN
709                            18         63        NaN
710                            38         65        NaN
711                            14         82        NaN
712                             5        112        NaN
713                            52        101        NaN
714                            53        100        NaN
715                            25        122        NaN
716                            19         94        NaN
717                            13        103        NaN
718                            30         54        NaN
719                            26        127        NaN

[720 rows x 11 columns]
```

In the following output, we can see the columns available in this dataset. It is found that there is a unknown column to the data without any value. It will be better if we remove the column.

```
In [3]: data.rename({"Unnamed: 10":'a'}, axis="columns", inplace=True)
        # Then, drop the column as usual.
        data.drop('a', axis=1, inplace=True)
        #this is better to call the head only, it's too long
        data.head()
```

```
Out[3]:     Month  Product Code      Product Name Category  \
        0  Jan-08             1  Rose Water Toner    Toner
        1  Feb-08             1  Rose Water Toner    Toner
        2  Mar-08             1  Rose Water Toner    Toner
        3  Apr-08             1  Rose Water Toner    Toner
        4  May-08             1  Rose Water Toner    Toner

                                  Usage  Price  Youth Sales  Middle age sales  \
        0  Toning, Soothing & Moisturising     56           20                32
        1  Toning, Soothing & Moisturising     56           47                33
        2  Toning, Soothing & Moisturising     56           11                45
        3  Toning, Soothing & Moisturising     56           35                36
        4  Toning, Soothing & Moisturising     56           13                28

           Senior citizen sales  Total Sales
        0                     5           57
        1                    40          120
        2                    23           79
        3                    31          102
        4                    43           84
```

Now, the table no longer possess unknown data. This output currently shows us the columns of the dataframe. However, this is not enough to understand the data, so we do further analysis.

```
In [4]: #understand the size of the table
        data.shape
```

```
Out[4]: (720, 10)
```

2.2 Metadata

Thus, we are able to see that the table has 720 rows of data and 10 columns. However, this is still not enough for us to completely understand it. Thus, analysis is done, and the metadata is produced. The following table shows the metadata of the dataframe.

Column	Type of data	Measurement of data	Description
Month	Categorical	Ordinal	The month and year of the row of data. It extends from Jan-08 to Dec-17.
Product Code	Categorical	Nominal	The product code of the product. It ranges from 1 to 6.
Product Name	Categorical	Nominal	The name of the product.
Usage	Categorical	Nominal	The usage of the product.
Price	Categorical	Nominal	The price of the product.
Youth Sales	Continuous	Ratio	The sales of the product from the age group 13-25.
Middle age Sales	Continuous	Ratio	The sales of the product from the age group 26-50.
Senior Citizen Sales	Continuous	Ratio	The sales of the product from the age group 50 onwards.
Total Sales	Continuous	Ratio	The total sales of the product in the particular month.

2.3 Descriptive Analysis

2.3.1 Data Description

In order to further understand the values in the data, descriptive analysis has been done. Firstly, data description has been done, in order to understand the trend of the data.

```
In [5]: #describe the data
        data.describe()
```

```
Out[5]:        Product Code       Price  Youth Sales  Middle age sales  \
        count    720.000000  720.000000   720.000000        720.000000
        mean       3.500000   66.850000    45.013889         27.354167
```

```
std        1.709012   25.987795   24.815869          15.633702
min        1.000000   27.000000    3.000000           0.000000
25%        2.000000   48.750000   23.750000          14.000000
50%        3.500000   54.000000   43.000000          27.000000
75%        5.000000   90.000000   67.000000          41.000000
max        6.000000  106.000000   89.000000          55.000000

       Senior citizen sales  Total Sales
count            720.000000   720.000000
mean              29.191667   101.559722
std               14.545098    31.196836
min                5.000000    22.000000
25%               17.000000    79.750000
50%               28.000000   100.000000
75%               42.000000   124.000000
max               55.000000   182.000000
```

2.3.2 Descriptive Statistics in Categorical Group

In this subchapter, we will be describing data via the categorical groups, which are the youth sales, middle-ages sales, and elderly sales. Before that, we will want to know the total number of people from different categories are involved. Firstly, we will find the value for the youth.

```
In [6]: #this is for the youth
        describeData = pd.read_csv("Main_table.csv")
        describeData = describeData.rename(columns={'Youth Sales':'Youth_Sales'})
        describeData.Youth_Sales.sum()

Out[6]: 32410
```

Next, the middle aged group:

```
In [7]: describeData = describeData.rename(columns={'Middle age sales':'Middle_age_sales'})
        describeData.Middle_age_sales.sum()

Out[7]: 19695
```

Finally, the elderly:

```
In [8]: describeData = describeData.rename(columns={'Senior citizen sales':'Senior_citizen_sale
        describeData.Senior_citizen_sales.sum()

Out[8]: 21018
```

We will also compute the total sales over the past 10 years for each product.

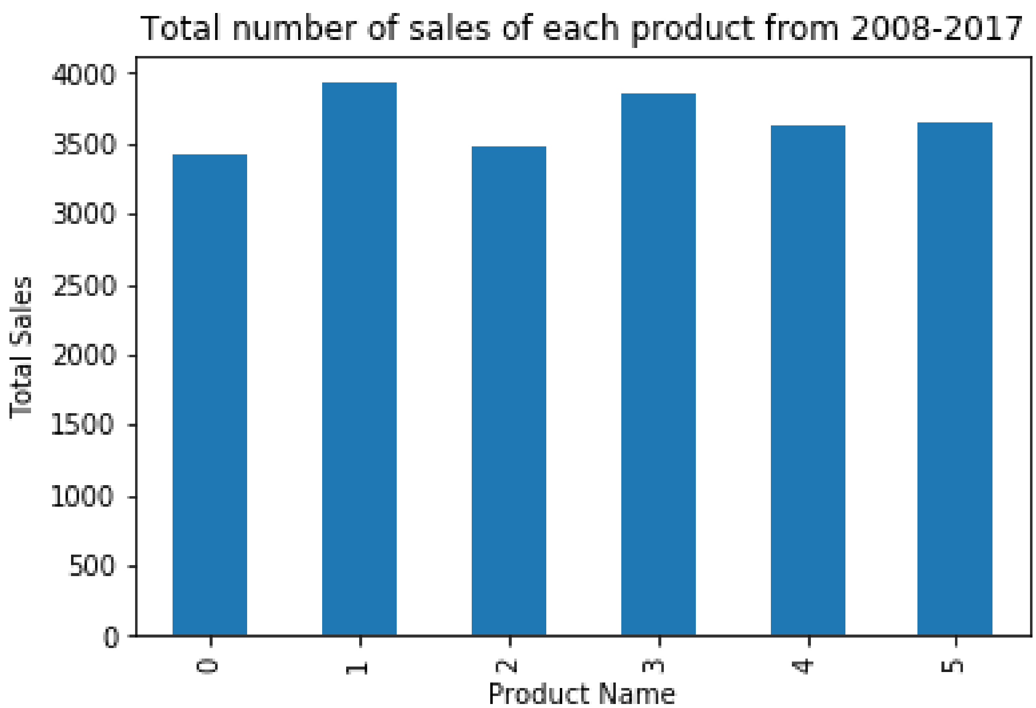
```
In [9]: data1 = pd.read_csv("Product.csv")
        data2 = pd.read_csv("Total_Number.csv")
        totalData = pd.merge(data1, data2, how="inner", left_on = 'Product Code', right_on = 'I
        totalData = totalData.rename(columns={'Total Sales_y':'Total_Sales_y'})
        totalData
```

```
Out[9]:    Product Code                         Name  Total Sales
        0             1            Rose Water Toner         3425
        1             2             Aqua Peel Toner         3925
        2             3            Pore Clean Toner         3483
        3             4    Rose Water Soothing Gel          3864
        4             5    Everyday Aqua Sun Cream          3629
        5             6         Rose Water Gel Cream        3638
```

With the dataframe, a bar chart can be plotted.

```
In [10]: %matplotlib inline
         totalData = pd.concat([data1, data2], axis=1, join='outer')

         fig = plt.figure(figsize=(6,4)) # define plot area
         ax = fig.gca() # define axis
         totalData['Total Sales'].plot.bar(ax = ax) # Use the plot.bar method on the counts da

         ax.set_title('Total number of sales of each product from 2008-2017') # Give the plot 
         ax.set_xlabel('Product Name') # Set text for the x axis
         ax.set_ylabel('Total Sales')# Set text for y axis
```

```
Out[10]: Text(0, 0.5, 'Total Sales')
```



Now, we will compute the total sales made by different categories of users.

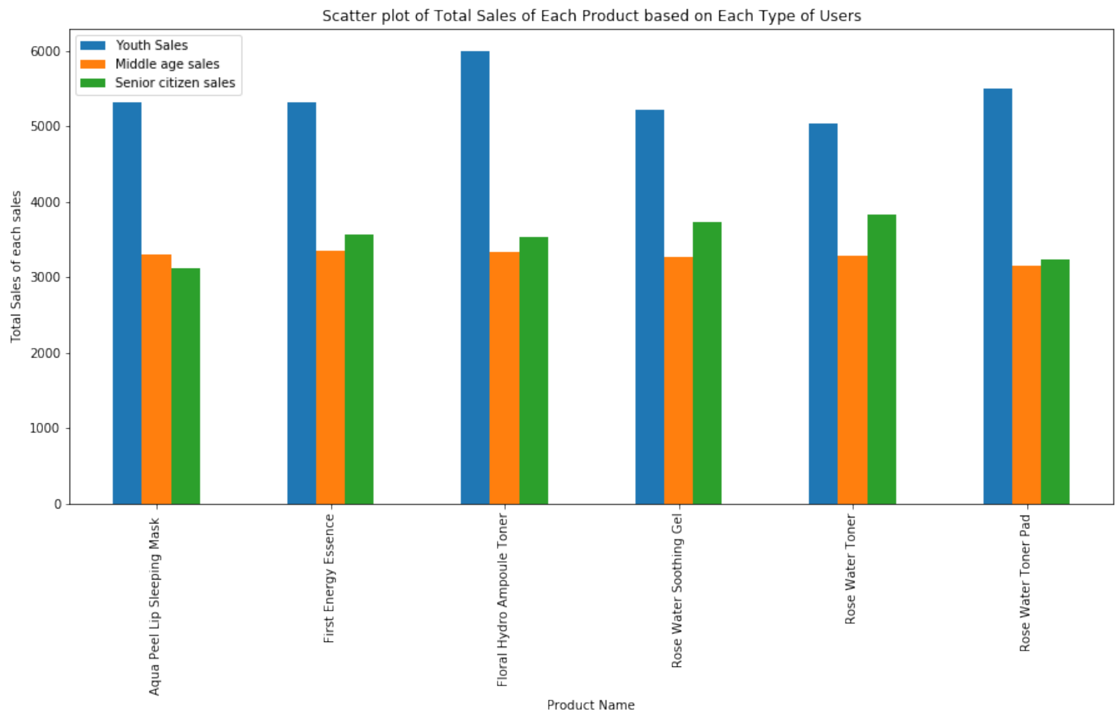
```
In [11]: usage = pd.read_csv("Main_table.csv")
         usage = usage.filter(['Product Name','Youth Sales','Middle age sales','Senior citizen

         count = usage.groupby('Product Name').sum()
         count
```

```
Out[11]:                              Youth Sales  Middle age sales  \
         Product Name
         Aqua Peel Lip Sleeping Mask         5319              3300
         First Energy Essence                5327              3361
         Floral Hydro Ampoule Toner          5995              3330
         Rose Water Soothing Gel             5220              3273
         Rose Water Toner                    5048              3279
         Rose Water Toner Pad                5501              3152

                                      Senior citizen sales
         Product Name
         Aqua Peel Lip Sleeping Mask                  3128
         First Energy Essence                         3568
         Floral Hydro Ampoule Toner                   3529
         Rose Water Soothing Gel                      3727
         Rose Water Toner                             3832
         Rose Water Toner Pad                         3234
```

With this table, we are able to create a bar chart:

```
In [12]: usage = pd.read_csv("Main_table.csv")
         usage = usage.filter(['Product Name','Youth Sales','Middle age sales','Senior citizen
         usage = usage.groupby('Product Name').sum()

         fig = plt.figure(figsize=(15, 7)) # define plot area
         ax = fig.gca() # define axis
         count.plot.bar(ax=ax)
         ax.set_title('Scatter plot of Total Sales of Each Product based on Each Type of Users
         ax.set_xlabel('Product Name') # Set text for the x axis
         ax.set_ylabel('Total Sales of each sales')# Set text for y axis
```

```
Out[12]: Text(0, 0.5, 'Total Sales of each sales')
```

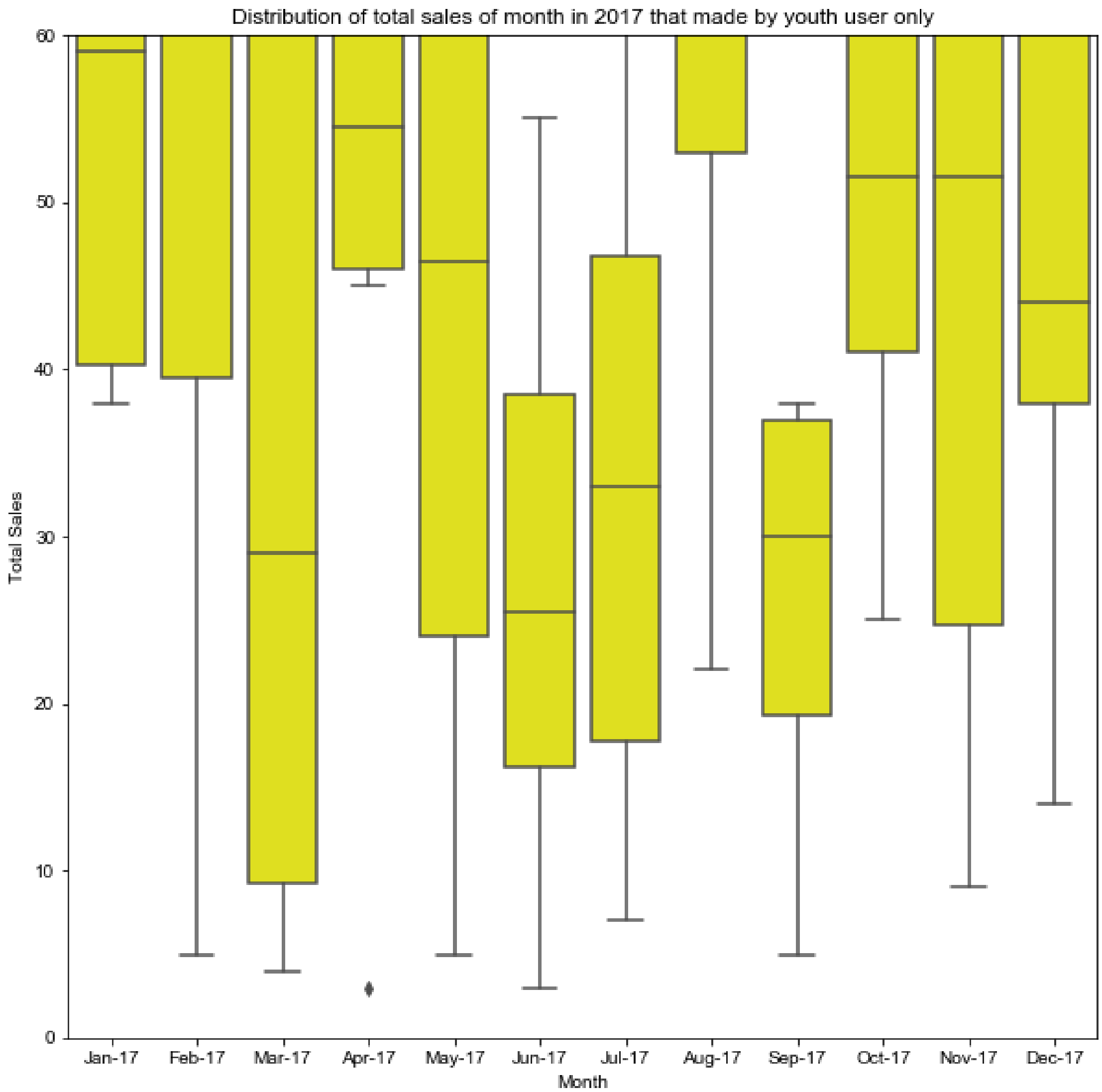


From the chart above, we can see that the youth is the main users of the company's products. In other words, the youth are the main customers that should be paid extra attention to when making business decisions.

Next, we will be showing the box plot of the youth sales for every product for every month of 2017. The reason why only one year shown is because the boxplot may get too messy to look at if all years are included.

```
In [13]: import seaborn as sns
         %matplotlib inline

         data = pd.read_csv("Main_table.csv")
         data = data[(data.Month ==  'Jan-17')| (data.Month ==  'Feb-17') |(data.Month ==  'Ma
                     (data.Month ==  'Apr-17')| (data.Month ==  'May-17')| (data.Month ==  'Ju
                     (data.Month ==  'Jul-17')| (data.Month ==  'Aug-17')|(data.Month ==  'Sep
                     (data.Month ==  'Oct-17')|(data.Month ==  'Nov-17')| (data.Month ==  'Dec

         fig = plt.figure(figsize=(10,10)) # define plot area
         ax = fig.gca() # define axis
         sns.set(style="whitegrid")
         sns.boxplot(x = 'Month', y = "Youth Sales", data = data, ax = ax, color = "Yellow")
         ax.set(ylim = (0.0, 60.0),
               title = 'Distribution of total sales of month in 2017 that made by youth user o
               xlabel = 'Month',
               ylabel = 'Total Sales')
```

Out[13]:
```
[(0.0, 60.0),
 Text(0, 0.5, 'Total Sales'),
 Text(0.5, 0, 'Month'),
 Text(0.5, 1.0, 'Distribution of total sales of month in 2017 that made by youth user
```

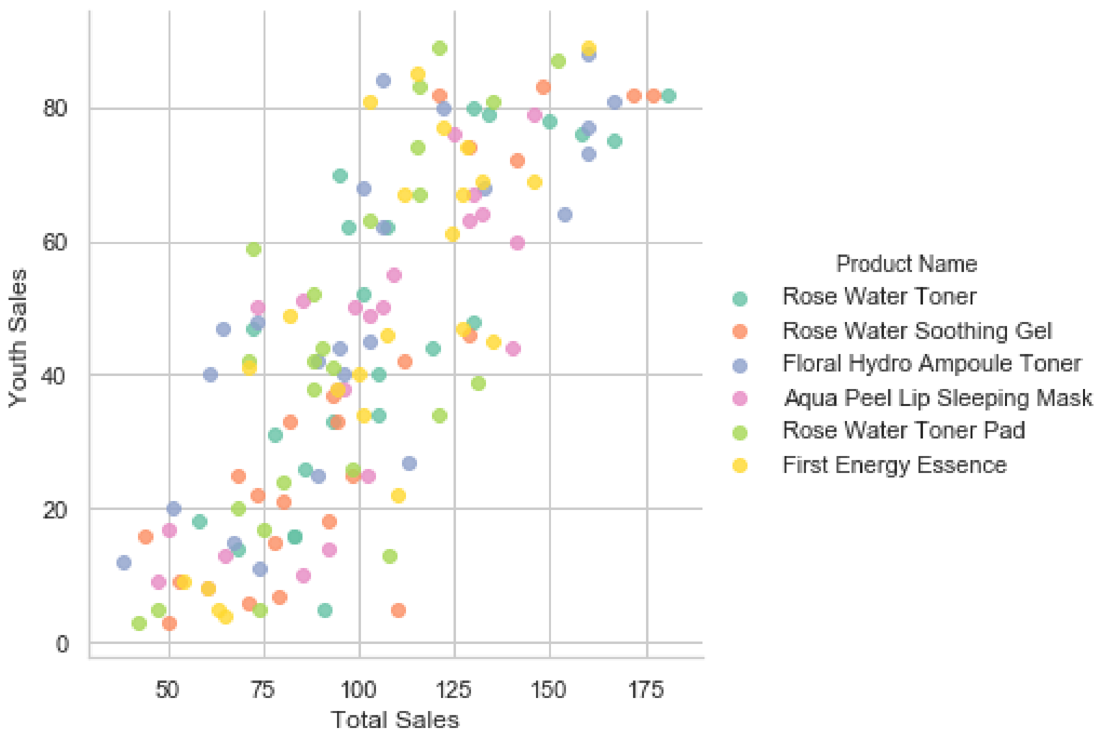


Next, we can also conduct analysis on the total sales against the youth sales with a facet graph.

In [127]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

data = pd.read_csv("Main_table.csv")
```

```
data = data[(data.Month ==  'Jan-17') | (data.Month ==  'Jan-16') |
             (data.Month ==  'Feb-17') | (data.Month ==  'Feb-16') |
             (data.Month ==  'Mar-17') | (data.Month ==  'Mar-16') |(data.Month ==
             (data.Month ==  'Apr-16')|(data.Month ==  'May-17')| (data.Month ==  'Ma
             (data.Month ==  'Jun-17')|(data.Month ==  'Jun-16')| (data.Month ==  'Ju
             (data.Month ==  'Aug-17')| (data.Month ==  'Aug-16')| (data.Month ==  'S
             (data.Month ==  'Sep-16')| (data.Month ==  'Oct-17')| (data.Month ==  'O
             (data.Month ==  'Nov-17')| (data.Month ==  'Nov-16')| (data.Month ==  'D
             (data.Month ==  'Dec-17')]

sns.lmplot(x = 'Total Sales', y = 'Youth Sales',
           data = data,
           hue = "Product Name",
           palette="Set2", fit_reg = False)
```

```
Out[127]: <seaborn.axisgrid.FacetGrid at 0x2b50d7b1240>
```



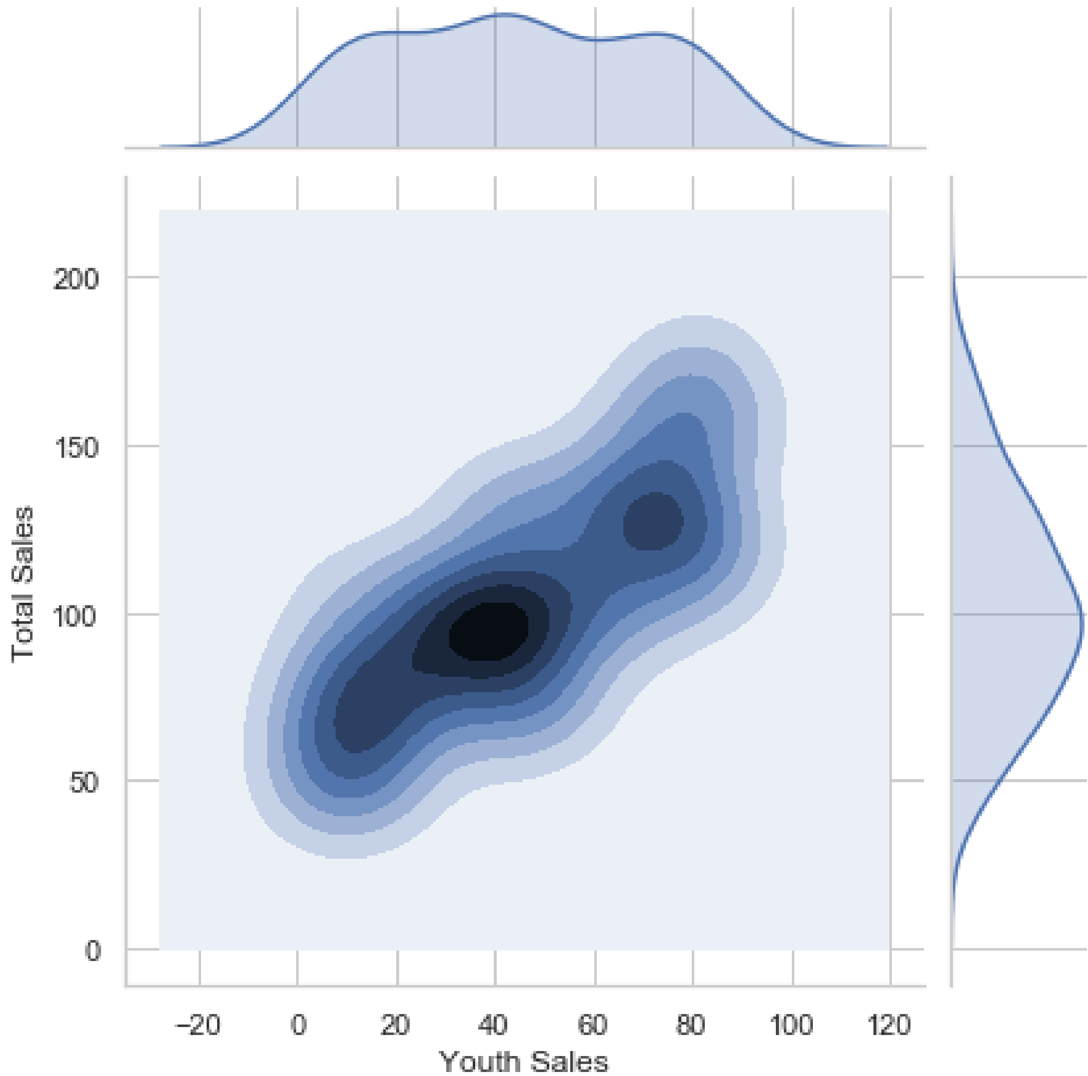
Thus, from the graph, we are able to see that when the total sales reaches 175, the youth sales reaches somewhere above 80, which contributes to close to 50% of the sales. This further strengthens the fact that the youth is the main customers of the brand.

We will also plot a seaborn plot to analyse further the data in concern.

```
In [128]: import seaborn as sns
```

```
sns.set_style("whitegrid")
sns.jointplot('Youth Sales', 'Total Sales', data = data,  kind='kde')
```

Out[128]: <seaborn.axisgrid.JointGrid at 0x2b50d6da390>



As the graph, the frequency for 40 youth sales to 100 total sales is very frequent, as the colour is shaded very dark. From all the analysis done, we are able to understand on the data we are dealing with.

3 Data Preparation

3.1 Dealing with messy data

In our scenario, every value of data is very important. Although some points may exceed the limit of (IQR*3), yet this data is the actual data and shouldn't be modified. Thus, all data shall remain as the same.

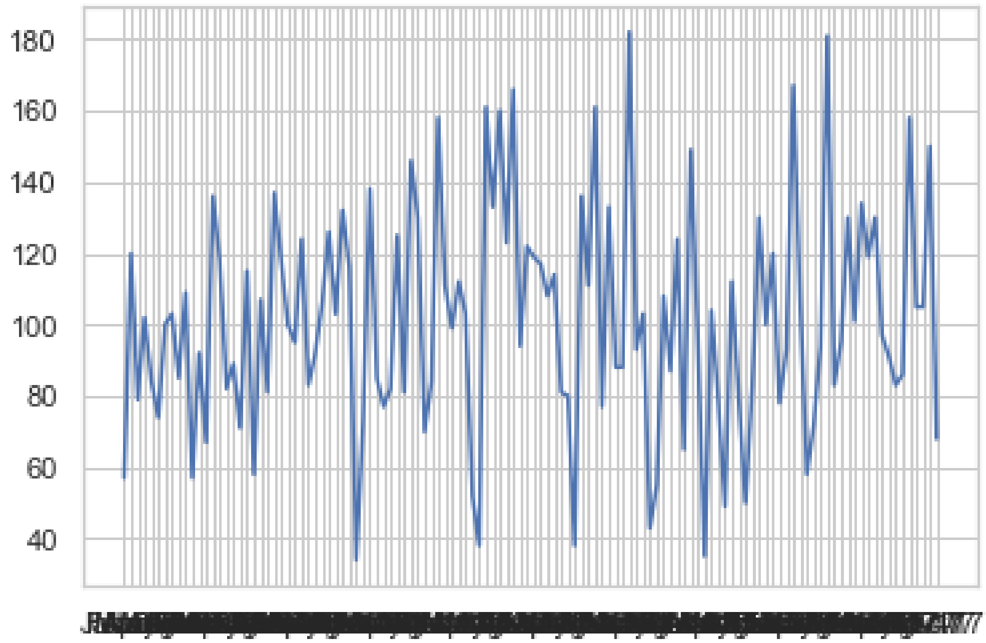
3.2 Feature Extraction

There are too many columns in the main table, which makes analysis and modelling difficult. Thus, feature extraction is done is order to take out the useful columns. We will be extracting the data into four new dataframes for the future analysis.

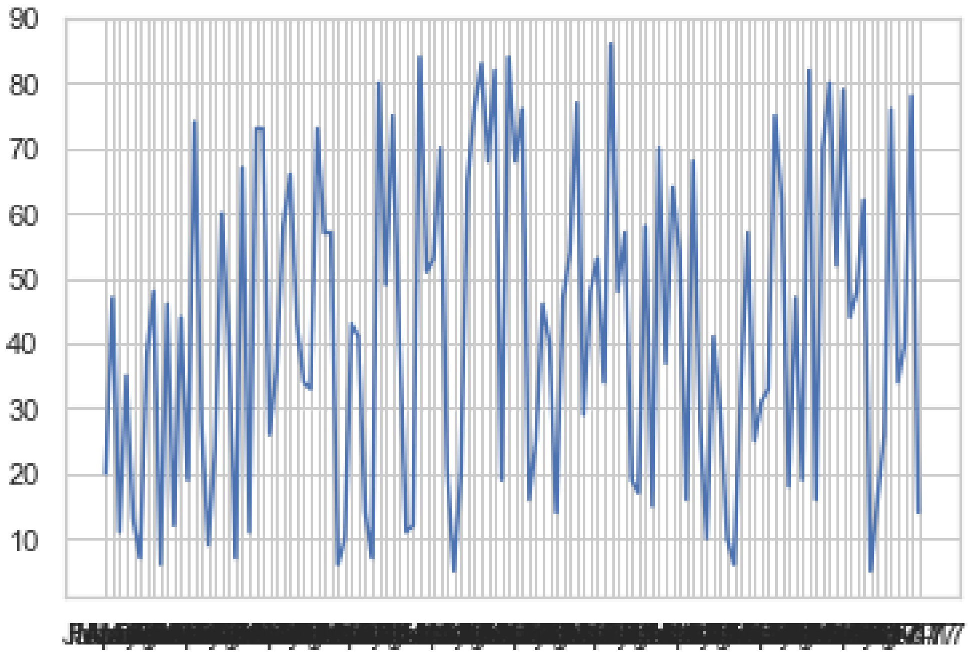
```
In [14]: #create new dataframe with new columns
         #total Sales
         data = pd.read_csv("Main_table.csv")
         totalSalesdata = data.filter(['Month','Product Code','Total Sales'],axis = 1)
         #youth Sales
         youthSalesdata = data.filter(['Month','Product Code','Youth Sales'],axis = 1)
         #middle Sales
         middleSalesdata = data.filter(['Month','Product Code','Middle age sales'],axis = 1)
         #elderly Sales
         elderlySalesdata = data.filter(['Month','Product Code','Senior citizen sales'],axis =
```

Since all dataframes have the rows for six product, it will be wise if we split the tables into 6. For this assignment, we will only take one product for research, as we plan to identify whether analysis via total sales or separate categories will yield better results.

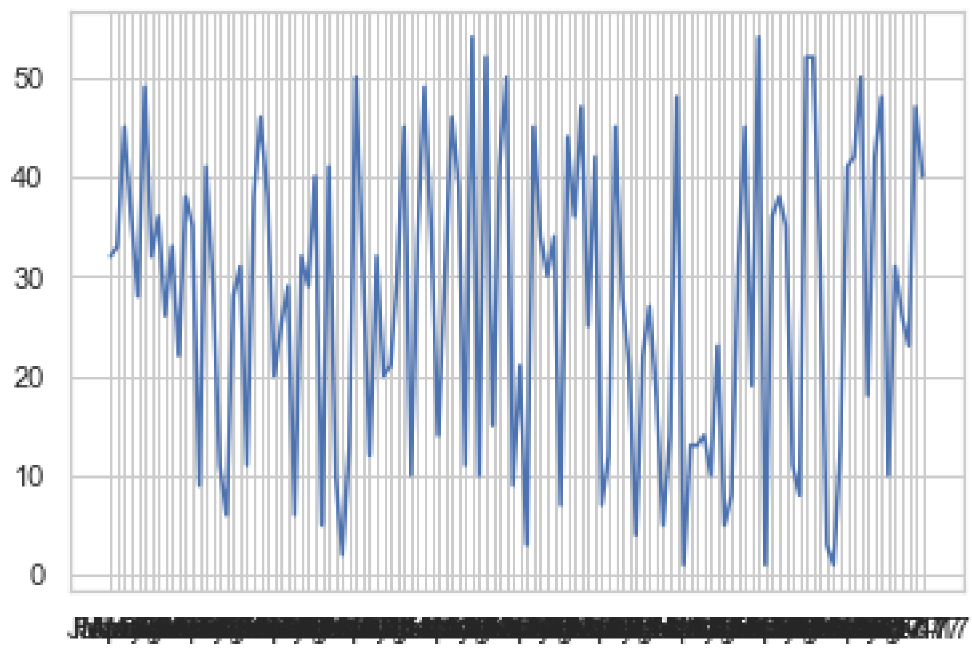
```
In [15]: #I am going to cut the tables into 6, since we have 6 products
         from pandas.plotting import autocorrelation_plot
         #total sales table
         totalSalestable1 = totalSalesdata.iloc[0:120]
         x=totalSalestable1['Month']
         y=totalSalestable1['Total Sales']
         plt.plot(x,y)
         plt.show()
```



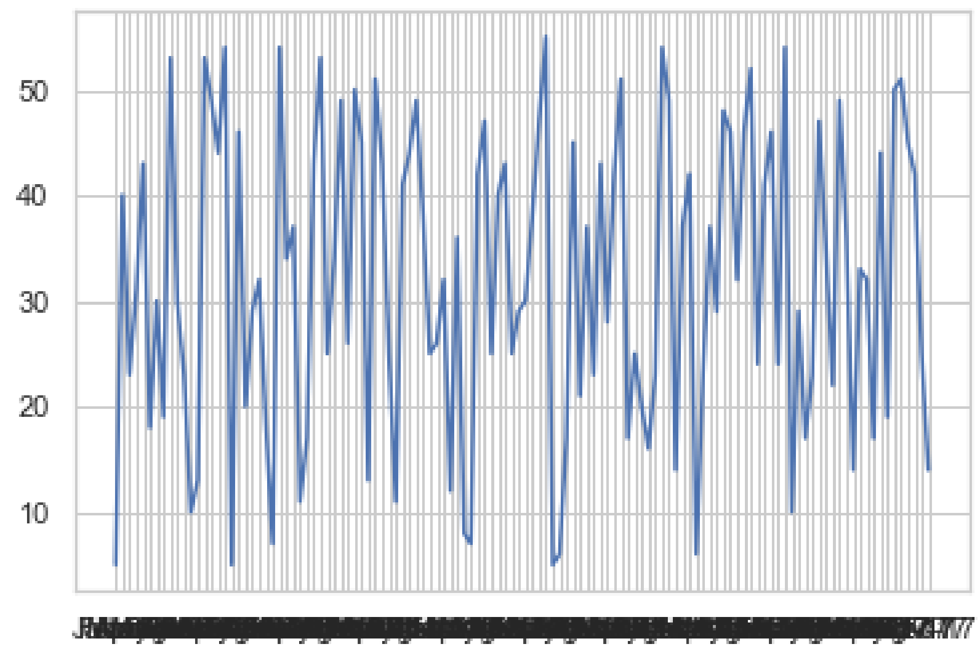

```
In [16]: #youth sales table
         youthSalesTable1 = youthSalesdata.iloc[0:120]
         x = youthSalesTable1['Month']
         y = youthSalesTable1['Youth Sales']
         plt.plot(x,y)
         plt.show()
```



```
In [17]: #middle age sales table
         middleSalesTable1 = middleSalesdata.iloc[0:120]
         x = middleSalesTable1['Month']
         y = middleSalesTable1['Middle age sales']
         plt.plot(x,y)
         plt.show()
```




```
In [18]: #elderly age sales table
         elderlySalesTable1 = elderlySalesdata.iloc[0:120]
         x = elderlySalesTable1['Month']
         y = elderlySalesTable1['Senior citizen sales']
         plt.plot(x,y)
         plt.show()
```



4 Modelling

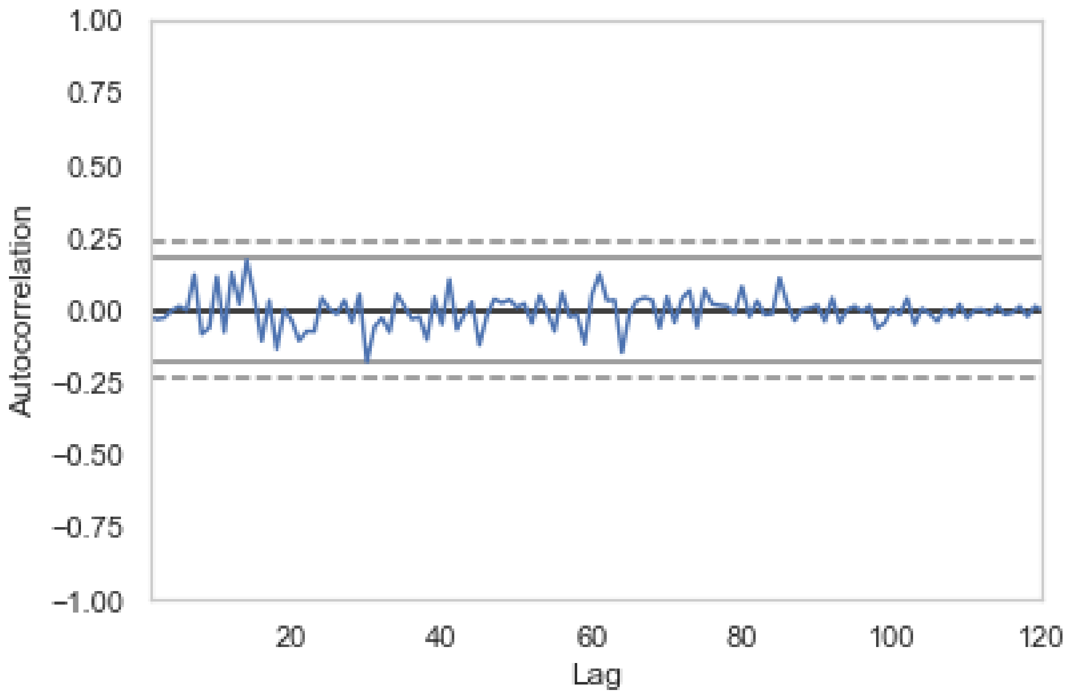
From the graphs drawn above, we will be conducting modelling on the four tables to identify whether modelling as a whole, or separate modelling produces better results.

4.1 Modelling with Total Sales

After the split, the product code is no longer useful for future analysis. Thus, we remove the column. We will start by trying to run the Autoregressive Integrated Moving Average (ARIMA) algorithm.

```
In [19]: #drop the product code since not needed
         totalSalestable1 = totalSalestable1.drop('Product Code',axis=1)
         #draw autocorrelation graph
         arimaTable = totalSalestable1.set_index(['Month'])
```

```
In [20]: autocorrelation_plot(arimaTable)
         plt.show()
         #Seems like the correlation isn't that significant. We need another method to underst
```



From the graph, it can be seen that the autocorrelation isn't significant enough in order to use the ARIMA algorithm. Thus, the Fast Fourier Transform (FFT) algorithm is considered. Before we can try the algorithm, we need to split the data into train and test datas first.

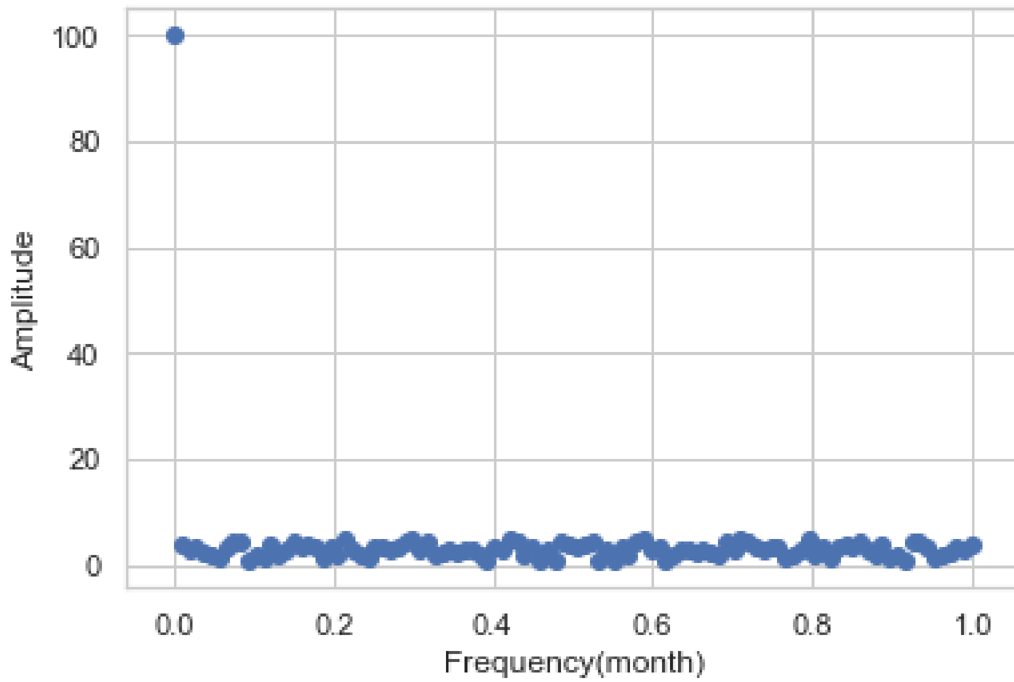
```
In [21]: #split data to train and test
         trainingData = totalSalestable1.iloc[0:108]
         testData = totalSalestable1.iloc[108:120]
         print(trainingData.head(),testData.head())
```

```
    Month  Total Sales
0   Jan-08           57
1   Feb-08          120
2   Mar-08           79
3   Apr-08          102
4   May-08           84       Month  Total Sales
108   Jan-17           134
109   Feb-17           119
110   Mar-17           130
111   Apr-17            97
112   May-17            91
```

Now, we can run the train model using the FFT algorithm.

```
In [22]: #try to run on fft
         fft = np.fft.fft(trainingData['Total Sales'])
```

```
N = 108
f = np.linspace(0,1,N)
plt.ylabel("Amplitude")
plt.xlabel("Frequency(month)") #month
frequency = f
amplitude = np.abs(fft * (1/N)) # 1 / N is a normalization factor
plt.scatter(frequency, amplitude)
plt.show()
```



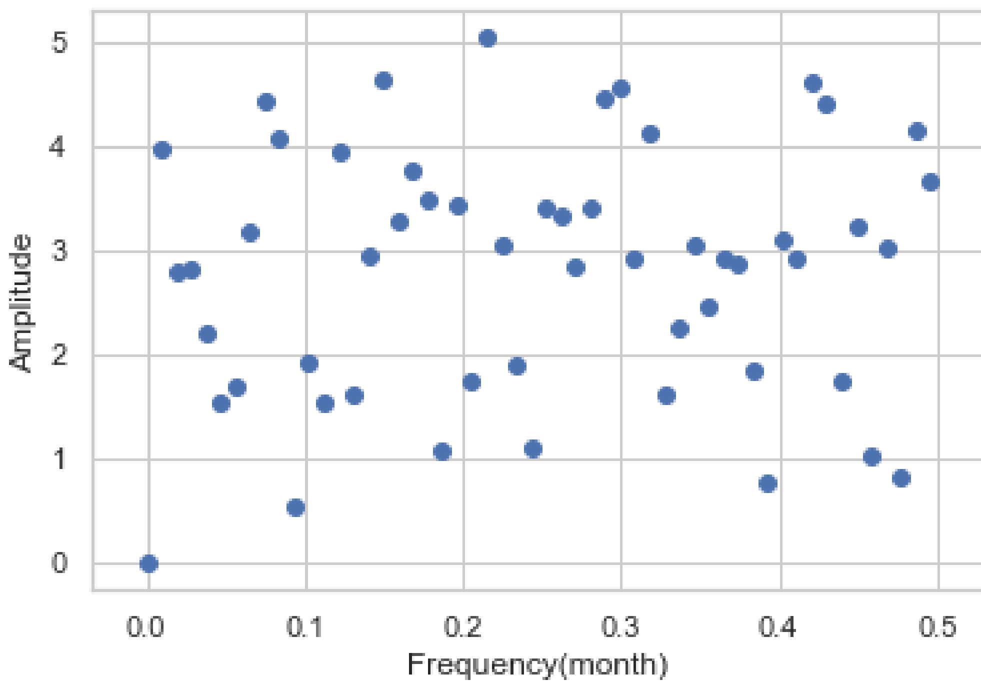
As shown, the significant frequency is 0. It can be seen that we have linear trend for this graph. Thus, detrending is needed.

```
In [23]: from scipy import signal
         #simple one line code
         trainingData_detrend = signal.detrend(trainingData['Total Sales'])
```

We can compute the FFT algorithm again with the detrended data.

```
In [24]: #compute fft again with the detrended data
         fft = np.fft.fft(trainingData_detrend)
         #this time round we have the table drawn
         N = trainingData_detrend.size
         f = np.linspace(0,1,N)
         plt.ylabel("Amplitude")
         plt.xlabel("Frequency(month)")
```

```
frequency = f[:N//2]
#divide by two as one half of it is useful
amplitude = np.abs(fft[:N//2] * (1/N)) # 1 / N is a normalization factor
plt.scatter(frequency, amplitude)
plt.show()
```



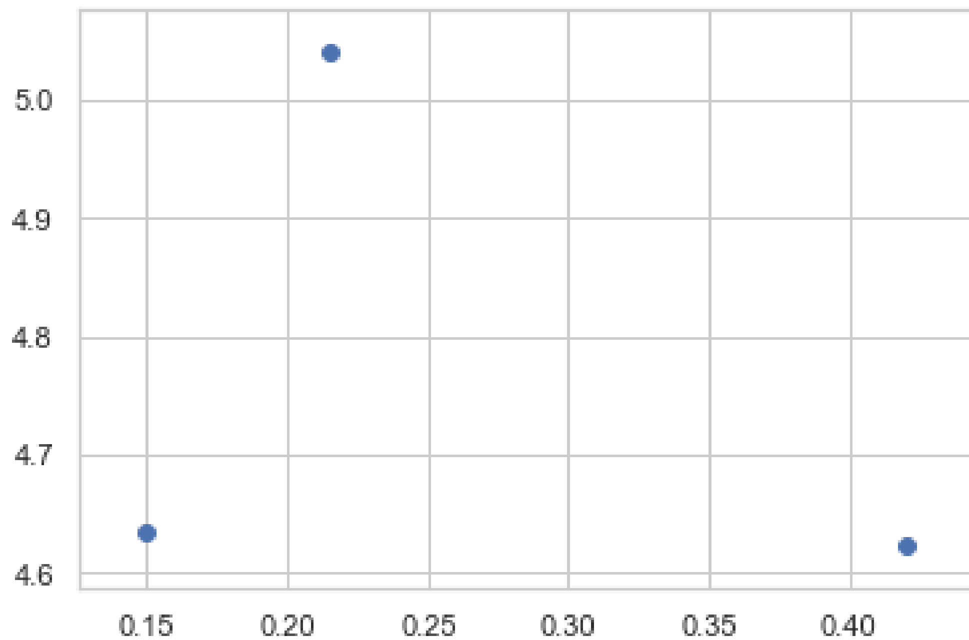
Thus, we are able to see the frequencies in different amplitudes. The frequency here is not of major concern, as the main result concerned is the highest amplitudes. Tests are run to decide which number of significant frequencies will be the best for this model. After testing, we found out that (insert number here) proves to be the best number. The following codes are the steps needed in order to produce the model.

```
In [25]: #first two seems to be ahead of others. We extract it
         frequency = pd.Series(frequency)
         amplitude = pd.Series(amplitude)
         #getting the y of the significant stuff
         significantAmplitudes = amplitude.nlargest(3)
         print(significantAmplitudes)
```

```
23    5.040985
16    4.633835
45    4.622657
dtype: float64
```

In [26]:
```
#getting the x of the significant stuff
significantFrequencies = [frequency[23],frequency[16],frequency[45]]
#ifft
plt.scatter(significantFrequencies,significantAmplitudes)
```

Out[26]:
```
<matplotlib.collections.PathCollection at 0x2b50d71af60>
```



In [27]:
```
#create a dataframe for the frequency and amplitude
processedData = frequency.to_frame()
processedData['amplitude'] = amplitude
processedData.columns = ['frequency','amplitude']
```

As the values we get for the code above is only half of the needed values, thus we need to extend the graph by another iteration. We also cleaned up the insignificant ampltiudes to reduce noise to our model.

In [28]:
```
#bring another half
processedData.amplitude[processedData.amplitude < 4.622] = 0
tempData = processedData.copy()
tempData.frequency = tempData.frequency + 0.5
#save this for testing
testTemp = processedData.copy()
#check on size to make sure didn't go wrong
tempData.size
```

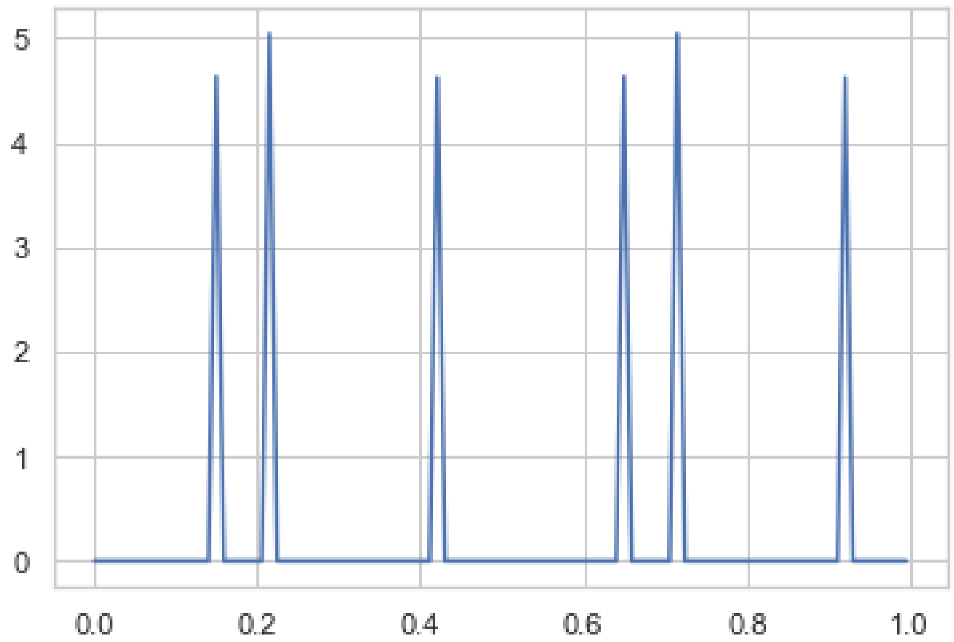
Out[28]:
```
108
```

```
In [29]: #combine graph together
         processedData = processedData.append(tempData)
         processedData.size
```

```
Out[29]: 216
```

Thus, the refined graph can be drawn.

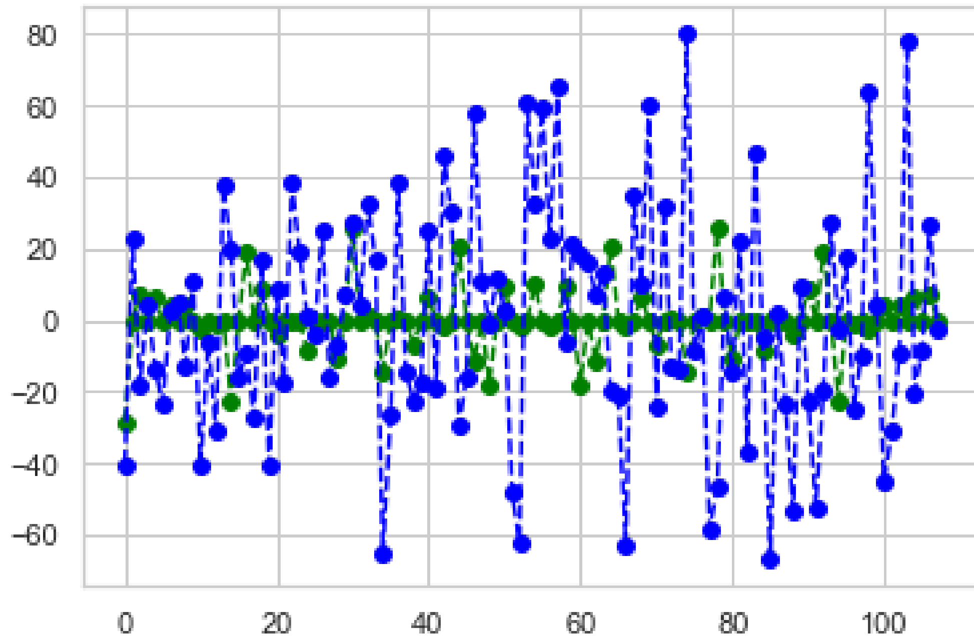
```
In [30]: #draw the refined graph
         plt.plot(processedData['frequency'],processedData['amplitude'])
         plt.show()
```



We can paste the graph with the original sales graph.

```
In [31]: #return the graph pattern
         processedGraph = -(np.fft.ifft(processedData['amplitude'])*108)
         processedSales = trainingData_detrend
         plt.plot(processedGraph,marker='o',color="green",linestyle='--')
         plt.plot(processedSales,marker='o',color="blue",linestyle='--')
         plt.show()
```

```
D:\Anaconda3\lib\site-packages\numpy\core\numeric.py:538: ComplexWarning: Casting complex valu
  return array(a, dtype, copy=False, order=order)
```



As sales tables can be too hard if the actual sales needs to be predicted, we define accuracy of the model as if the model produces a correct difference (increase or decrease), regardless of the number produced. The following steps are the code needed to obtain the difference.

In [32]:
```
#Original Data(blue line)
processedSalesTable = pd.DataFrame(processedSales).diff()
processedSalesTable.head()
```

Out[32]:

	0
0	NaN
1	62.944044
2	-41.055956
3	22.944044
4	-18.055956

In [33]:
```
#Model Data(Green Line)
processedGraphTable = pd.DataFrame(processedGraph.real).diff()
processedGraphTable.head()
```

Out[33]:

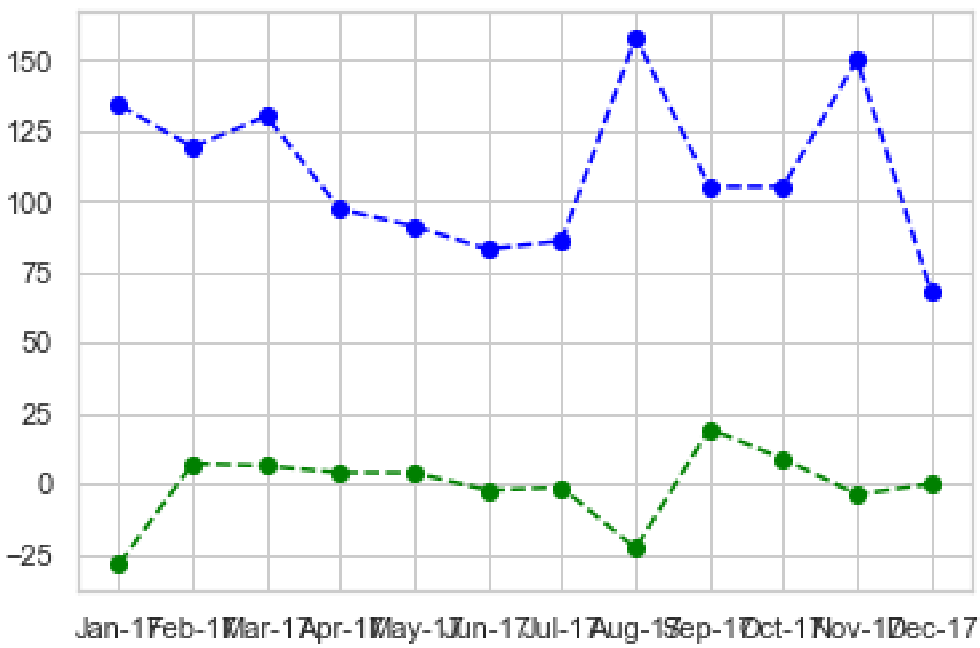
	0
0	NaN
1	28.594953
2	7.044918
3	-7.044918
4	6.345147

We export the data to a csv to do manual verification of accuracy.

```
In [34]: #produce raw results
         result=pd.concat([processedSalesTable,processedGraphTable],axis=1)
         result
         result.to_csv(r'C://Users/Leon Wong/Desktop/result.csv')
         #produce meaningful results
         #accuracy
```

Thus, the model will be tested with the drawn line to check on the accuracy.

```
In [35]: #comparisons
         testGraph = -(np.fft.ifft(testTemp['amplitude'])*108)[0:12]
         plt.plot(testGraph,marker='o',color="green",linestyle='--')
         plt.plot(testData['Month'],testData['Total Sales'],marker='o',color="blue",linestyle=
         plt.show()
```



The difference is then obtained.

```
In [36]: testDataTable = pd.DataFrame(testData['Total Sales']).diff()
         testDataTable = testDataTable.reset_index(drop = True)
         testDataTable.head()
```

```
Out[36]:    Total Sales
         0          NaN
         1        -15.0
         2         11.0
         3        -33.0
         4         -6.0
```

```
In [37]: testGraphTable = pd.DataFrame(testGraph.real).diff()
         testGraphTable.head()
```

```
Out[37]:            0
         0        NaN
         1  35.639871
         2  -0.699770
         3  -2.448565
         4  -0.053121
```

```
In [38]: #combine both graphs
         testResult=pd.concat([testDataTable,testGraphTable],axis=1)
         testResult.head()
```

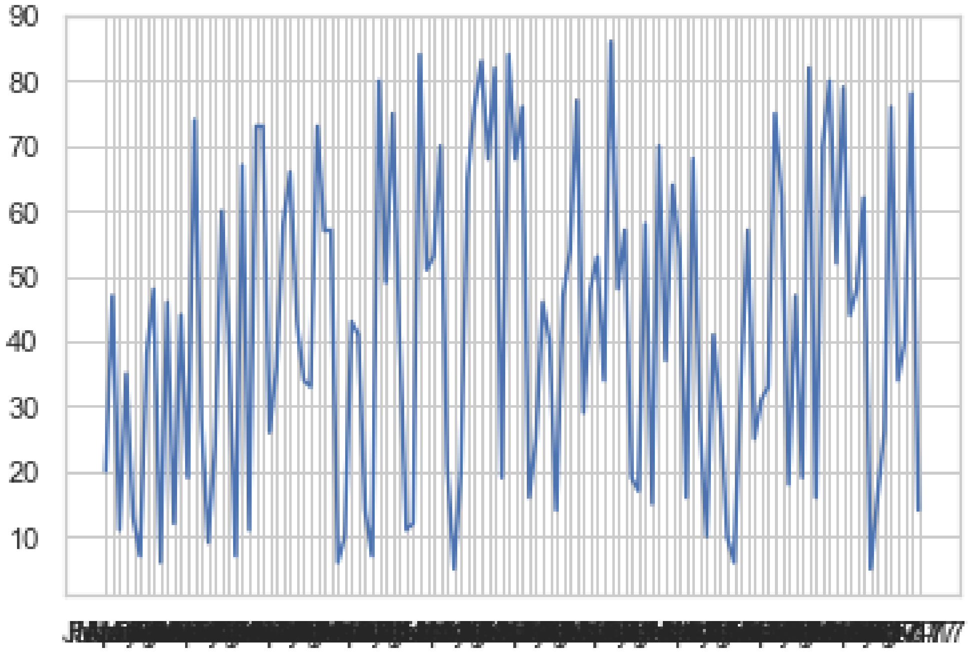
```
Out[38]:    Total Sales          0
         0          NaN        NaN
         1        -15.0  35.639871
         2         11.0  -0.699770
         3        -33.0  -2.448565
         4         -6.0  -0.053121
```

```
In [39]: #send to csv
         testResult.to_csv(r'C://Users/Leon Wong/Desktop/testResult.csv')
```

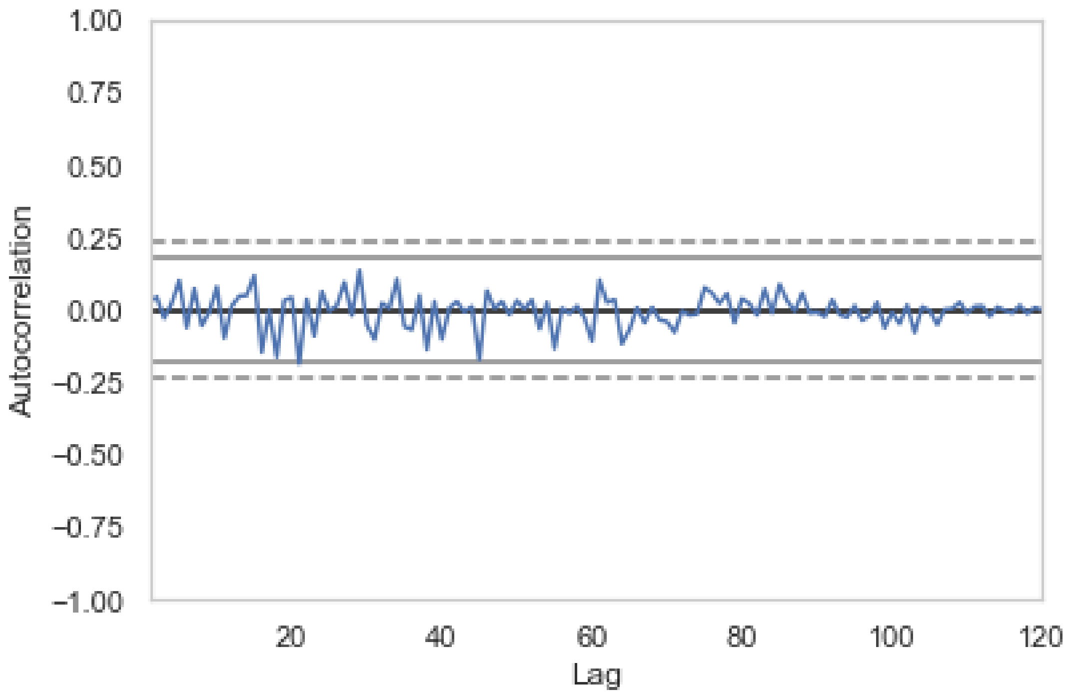
4.2 Modelling with Youth Sales

The same is done as well for the youth sales.

```
In [40]: x=youthSalesTable1['Month']
         y=youthSalesTable1['Youth Sales']
         plt.plot(x,y)
         plt.show()
```




```
In [41]: youthSalesTable1 = youthSalesTable1.drop('Product Code',axis=1)
         #draw autocorrelation graph
         arimaTable = youthSalesTable1.set_index(['Month'])

In [42]: autocorrelation_plot(arimaTable)
         plt.show()
         #Seems like the correlation isn't that significant. We need another method to underst
```

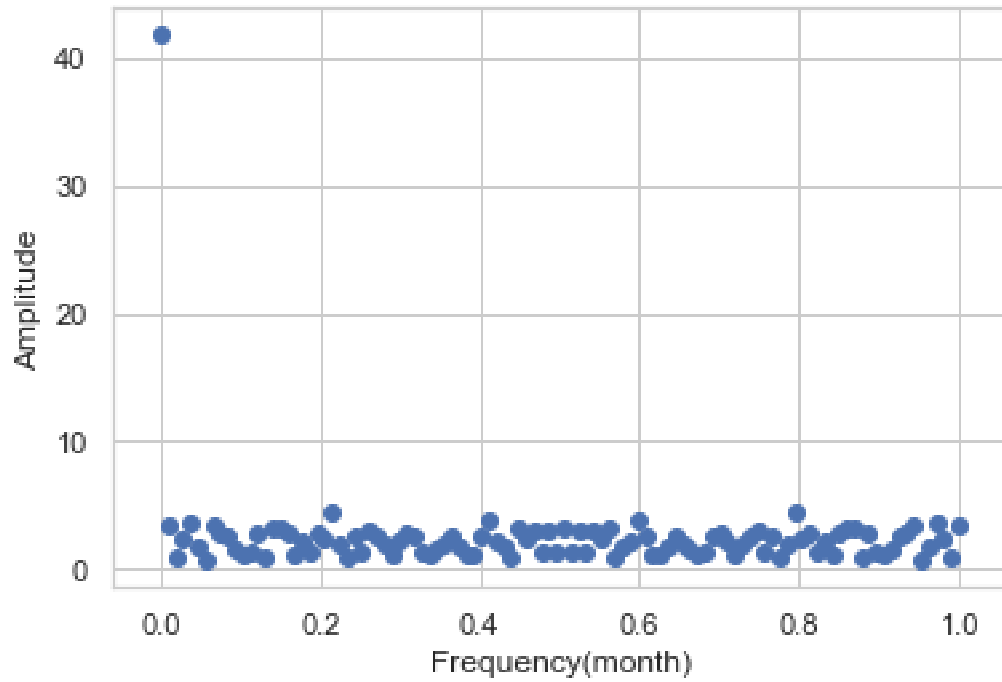


In [43]:
```
#split data to train and test
trainingData = youthSalesTable1.iloc[0:108]
testData = youthSalesTable1.iloc[108:120]
print(trainingData.head(),testData.head())
```

```
    Month  Youth Sales
0  Jan-08           20
1  Feb-08           47
2  Mar-08           11
3  Apr-08           35
4  May-08           13        Month  Youth Sales
108  Jan-17           79
109  Feb-17           44
110  Mar-17           48
111  Apr-17           62
112  May-17            5
```

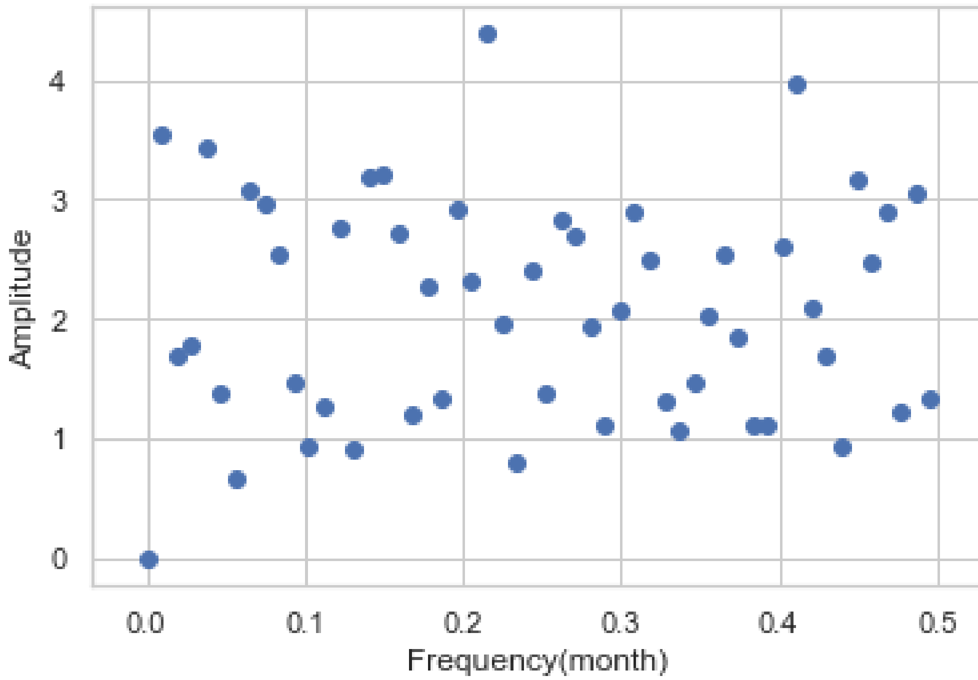
In [44]:
```
#try to run on fft
fft = np.fft.fft(trainingData['Youth Sales'])
N = 108
f = np.linspace(0,1,N)
plt.ylabel("Amplitude")
plt.xlabel("Frequency(month)") #month
```

```
frequency = f
amplitude = np.abs(fft * (1/N)) # 1 / N is a normalization factor
plt.scatter(frequency, amplitude)
plt.show()
```




```
In [45]: from scipy import signal
         #simple one line code
         trainingData_detrend = signal.detrend(trainingData['Youth Sales'])
```

```
In [46]: #compute fft again with the detrended data
         fft = np.fft.fft(trainingData_detrend)
         #this time round we have the table drawn
         N = trainingData_detrend.size
         f = np.linspace(0,1,N)
         plt.ylabel("Amplitude")
         plt.xlabel("Frequency(month)")
         frequency = f[:N//2]
         #divide by two as one half of it is useful
         amplitude = np.abs(fft[:N//2] * (1/N)) # 1 / N is a normalization factor
         plt.scatter(frequency, amplitude)
         plt.show()
```

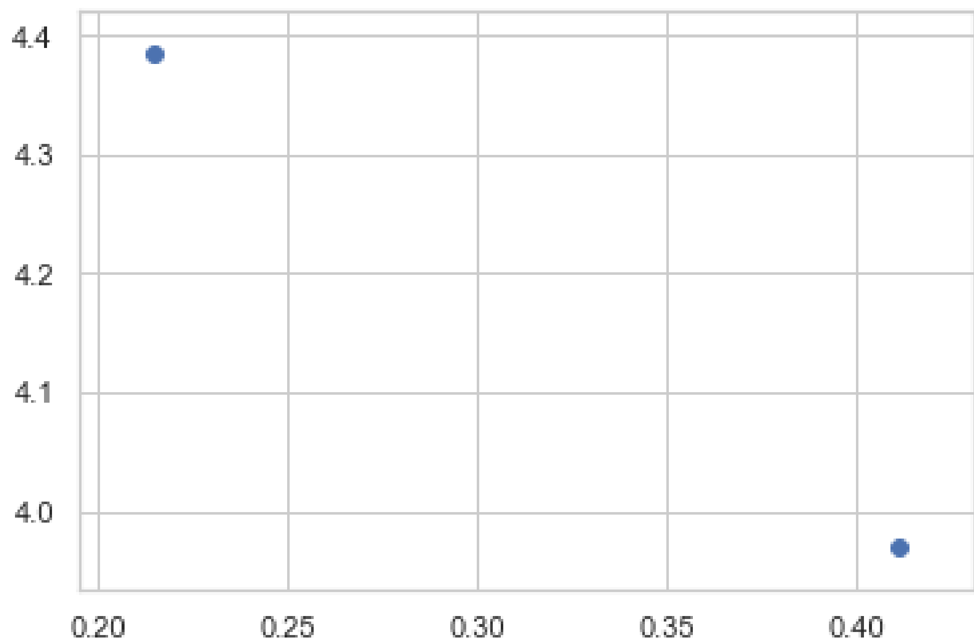



```
In [47]: #first two seems to be ahead of others. We extract it
         frequency = pd.Series(frequency)
         amplitude = pd.Series(amplitude)
         #getting the y of the significant stuff
         significantAmplitudes = amplitude.nlargest(2)
         print(significantAmplitudes)
```

```
23    4.385527
44    3.969174
dtype: float64
```

```
In [48]: #getting the x of the significant stuff
         significantFrequencies = [frequency[23],frequency[44]]
         #ifft
         plt.scatter(significantFrequencies,significantAmplitudes)
```

```
Out[48]: <matplotlib.collections.PathCollection at 0x2b50d015630>
```



In [49]:
```
#create a dataframe for the frequency and amplitude
processedData = frequency.to_frame()
processedData['amplitude'] = amplitude
processedData.columns = ['frequency','amplitude']
```

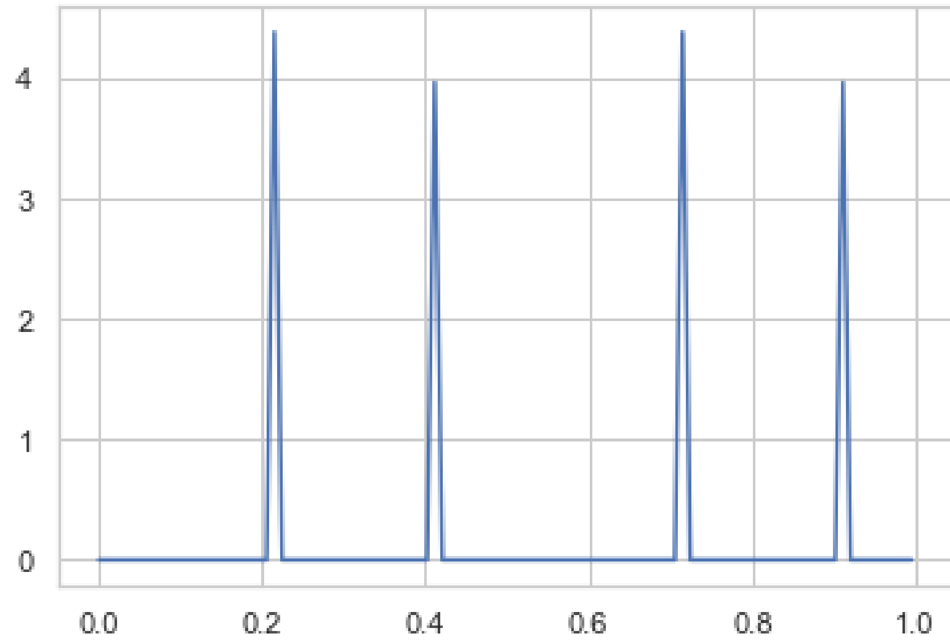
In [50]:
```
#bring another half
processedData.amplitude[processedData.amplitude < 3.969] = 0
tempData = processedData.copy()
tempData.frequency = tempData.frequency + 0.5
#save this for testing
testTemp = processedData.copy()
#check on size to make sure didn't go wrong
tempData.size
```

Out[50]: 108

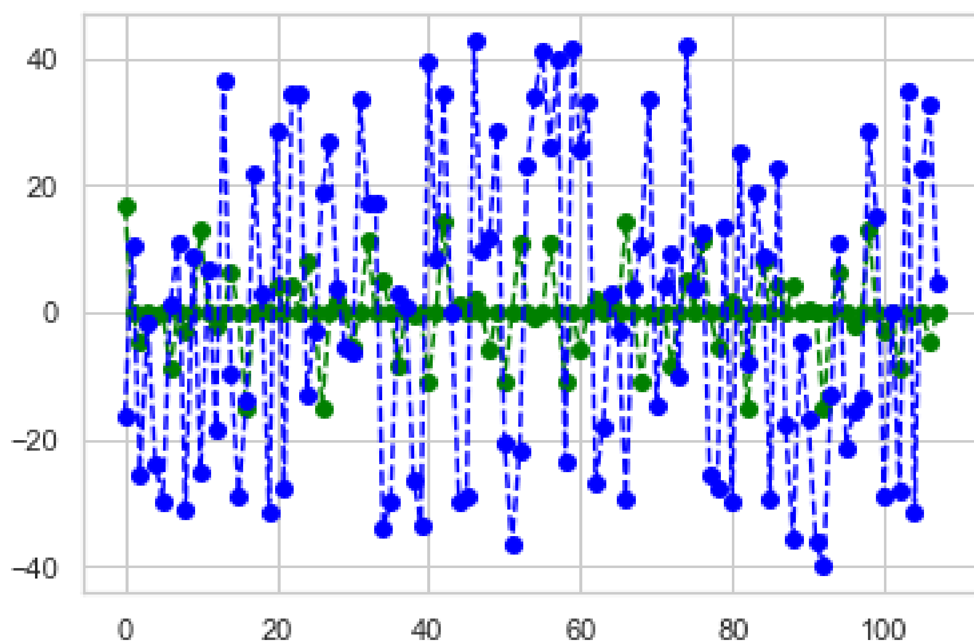
In [51]:
```
#combine graph together
processedData = processedData.append(tempData)
processedData.size
```

Out[51]: 216

In [52]:
```
#draw the refined graph
plt.plot(processedData['frequency'],processedData['amplitude'])
plt.show()
```




```
In [53]: #return the graph pattern
         processedGraph = (np.fft.ifft(processedData['amplitude'])*108)
         processedSales = trainingData_detrend
         plt.plot(processedGraph,marker='o',color="green",linestyle='--')
         plt.plot(processedSales,marker='o',color="blue",linestyle='--')
         plt.show()
```



In [54]:
```python
#Original Data(blue line)
processedSalesTable = pd.DataFrame(processedSales).diff()
processedSalesTable.head()
```

Out[54]:

	0
0	NaN
1	26.894205
2	-36.105795
3	23.894205
4	-22.105795

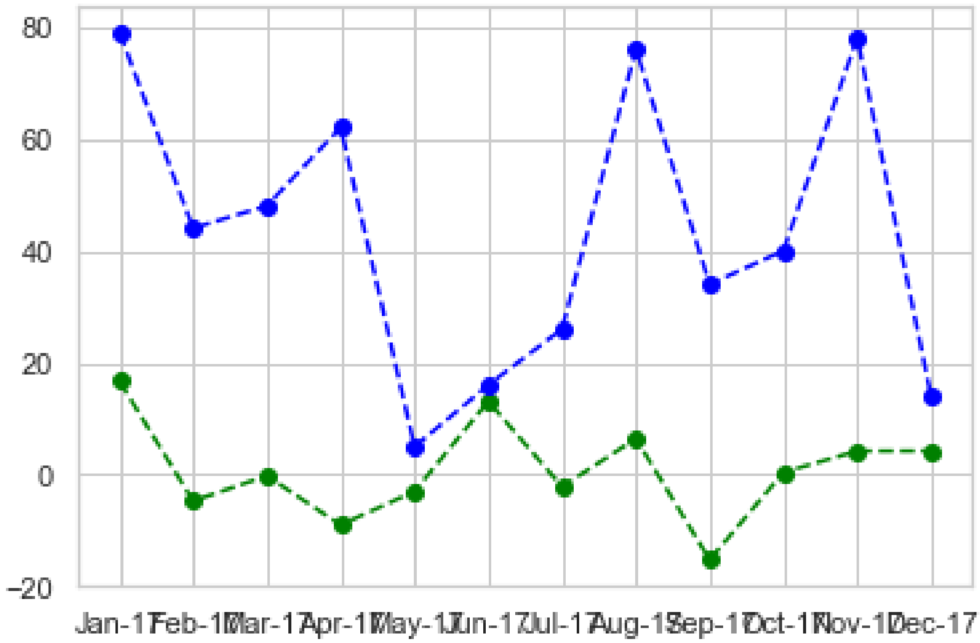
In [55]:
```python
#Model Data(Green Line)
processedGraphTable = pd.DataFrame(processedGraph.real).diff()
processedGraphTable.head()
```

Out[55]:

	0
0	NaN
1	-16.709401
2	-4.693882
3	4.693882
4	-0.209914

In [56]:
```python
#produce raw results
result=pd.concat([processedSalesTable,processedGraphTable],axis=1)
result
result.to_csv(r'C://Users/Leon Wong/Desktop/resultYouth.csv')
#produce meaningful results
#accuracy
```

In [57]:
```python
testGraph = (np.fft.ifft(testTemp['amplitude'])*108)[0:12]
plt.plot(testGraph,marker='o',color="green",linestyle='--')
plt.plot(testData['Month'],testData['Youth Sales'],marker='o',color="blue",linestyle=
plt.show()
```




```
In [58]: testDataTable = pd.DataFrame(testData['Youth Sales']).diff()
         testDataTable = testDataTable.reset_index(drop = True)
         testDataTable.head()
```

```
Out[58]:    Youth Sales
         0          NaN
         1        -35.0
         2          4.0
         3         14.0
         4        -57.0
```

```
In [59]: testGraphTable = pd.DataFrame(testGraph.real).diff()
         testGraphTable.head()
```

```
Out[59]:            0
         0        NaN
         1 -21.403283
         2   4.483968
         3  -8.772770
         4   6.005543
```

```
In [60]: #combine both graphs
         testResult=pd.concat([testDataTable,testGraphTable],axis=1)
         testResult.head()
```

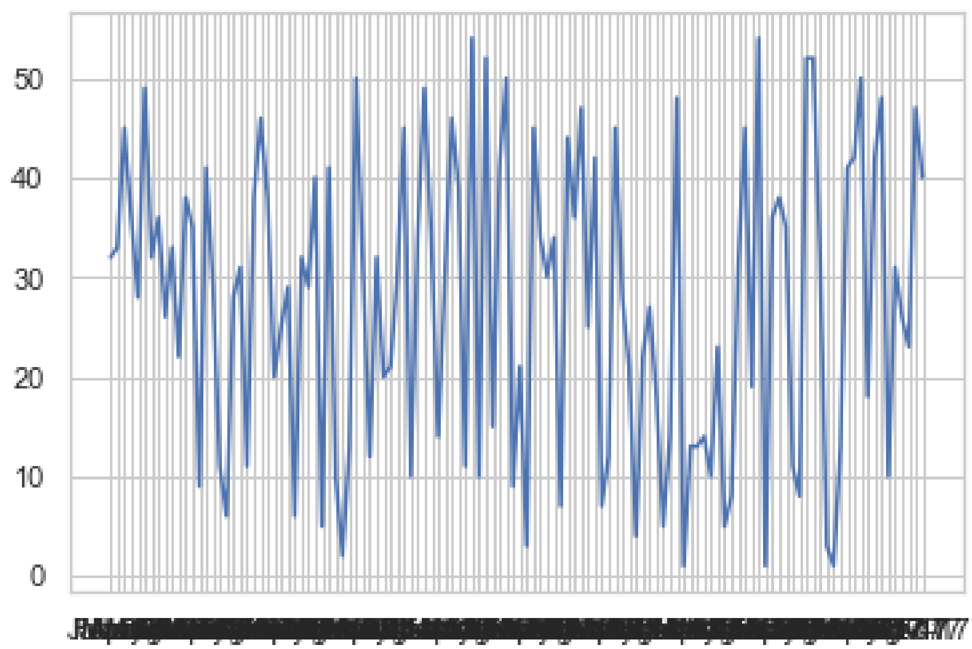
```
Out[60]:    Youth Sales          0
        0           NaN        NaN
        1         -35.0 -21.403283
        2           4.0   4.483968
        3          14.0  -8.772770
        4         -57.0   6.005543
```

```
In [61]: #send to csv
         testResult.to_csv(r'C://Users/Leon Wong/Desktop/testYouthResult.csv')
```

4.3 Modelling with Middle Age Sales

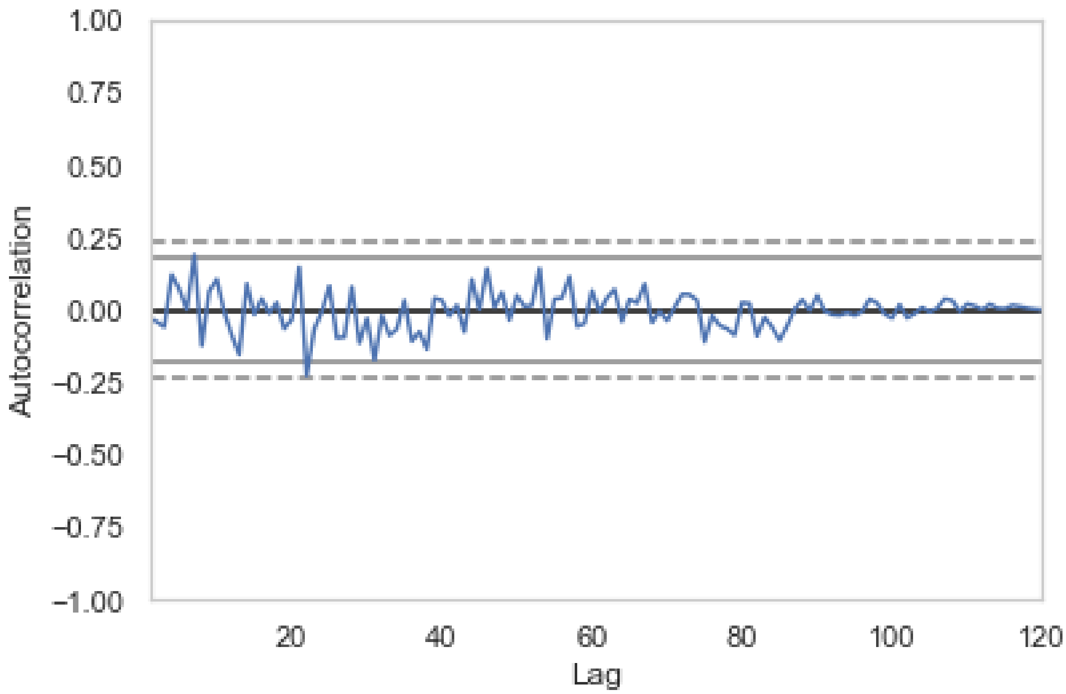
We apply the same method to the middle age sales. Firstly, we show the graph first:

```
In [62]: middleSalesTable = middleSalesdata.iloc[0:120]
         x=middleSalesTable['Month']
         y=middleSalesTable['Middle age sales']
         plt.plot(x,y)
         plt.show()
```



We will try to run the ARIMA as well.

```
In [63]: middleSalesTable = middleSalesTable.drop('Product Code',axis=1)
         arimaTable = middleSalesTable.set_index(['Month'])
         autocorrelation_plot(arimaTable)
         plt.show()
```



As it can be seen, the correlation is still not significant. Thus, we will use FFT.

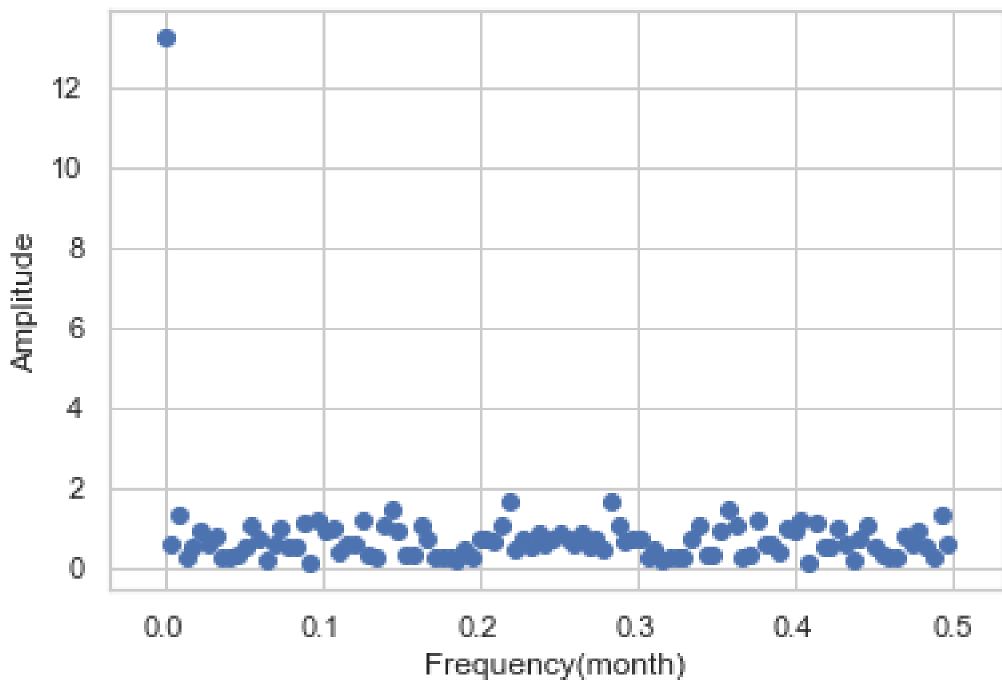
```
In [64]: #train test split
         trainingData = middleSalesTable.iloc[0:108]
         testData = middleSalesTable.iloc[108:120]
         print(trainingData.head(),testData.head())
```

```
    Month  Middle age sales
0   Jan-08                32
1   Feb-08                33
2   Mar-08                45
3   Apr-08                36
4   May-08                28        Month  Middle age sales
108   Jan-17                41
109   Feb-17                42
110   Mar-17                50
111   Apr-17                18
112   May-17                42
```

```
In [65]: fft = np.fft.fft(trainingData['Middle age sales'])
         N = trainingData.size
         print(N.size)
         f = np.linspace(0,1,N)
         print(f.size)
```

```
plt.ylabel("Amplitude")
plt.xlabel("Frequency(month)") #month
frequency = f[:N//2]
amplitude = np.abs(fft[:N//2] * (1/N)) # 1 / N is a normalization factor
plt.scatter(frequency, amplitude)
plt.show()
```

```
1
216
```

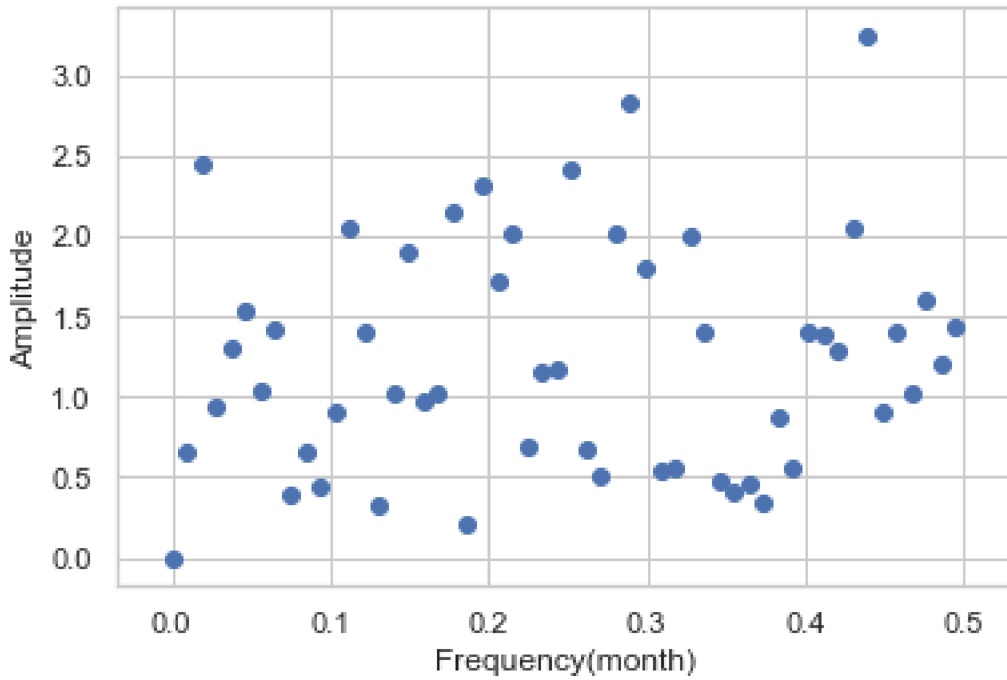


A linear trend is observed again. Thus, detrending is needed.

```
In [66]: from scipy import signal
         #simple one line code
         trainingData_detrend = signal.detrend(trainingData['Middle age sales'])
```

```
In [67]: #compute fft again with the detrended data
         fft = np.fft.fft(trainingData_detrend)
         #this time round we have the table drawn
         N = trainingData_detrend.size
         f = np.linspace(0,1,N)
         plt.ylabel("Amplitude")
         plt.xlabel("Frequency(month)")
         frequency = f[:N//2]
         amplitude = np.abs(fft[:N//2] * (1/N)) # 1 / N is a normalization factor
```

```
plt.scatter(frequency, amplitude)
plt.show()
```



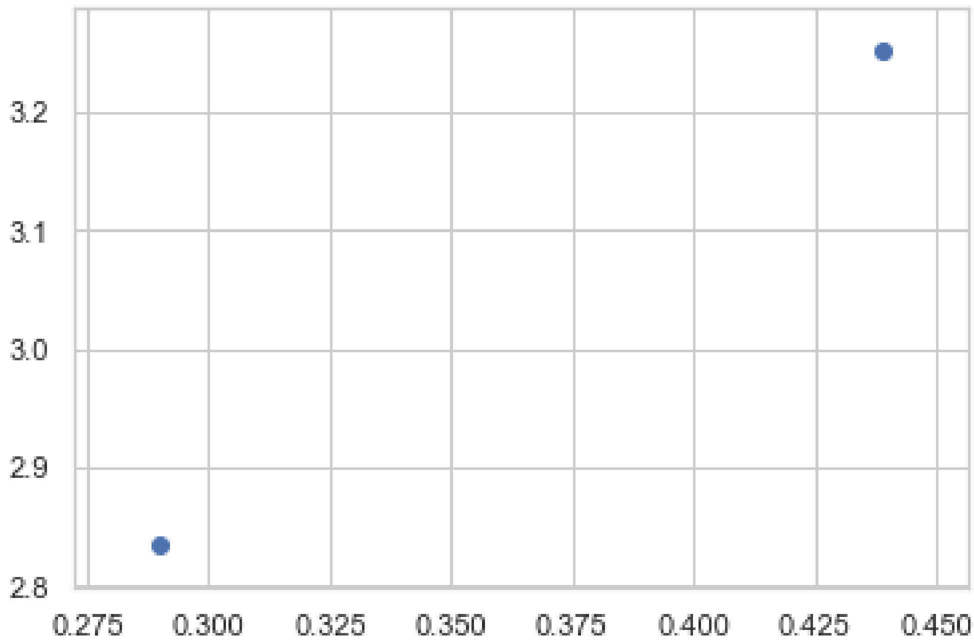

```
In [68]: #first two seems to be ahead of others. We extract it
         frequency = pd.Series(frequency)
         amplitude = pd.Series(amplitude)
         #getting the y of the significant stuff
         significantAmplitudes = amplitude.nlargest(2)
         print(significantAmplitudes)
```

```
47    3.252205
31    2.834096
dtype: float64
```

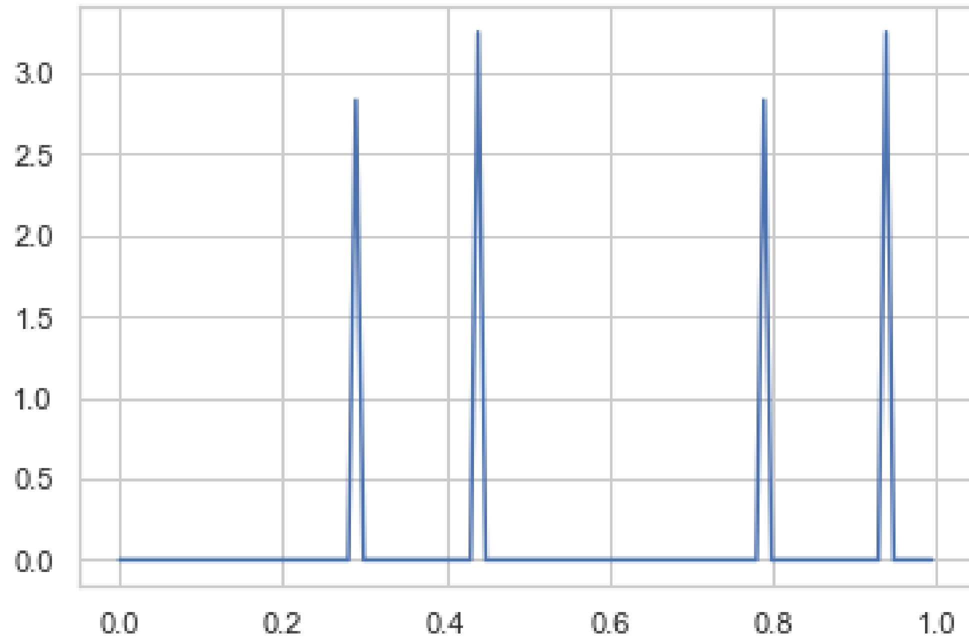
```
In [69]: #getting the x of the significant stuff,
         significantFrequencies = [frequency[47],frequency[31]]
         print(significantFrequencies)
         #ifft
         plt.scatter(significantFrequencies,significantAmplitudes)
```

```
[0.4392523364485981, 0.2897196261682243]
```

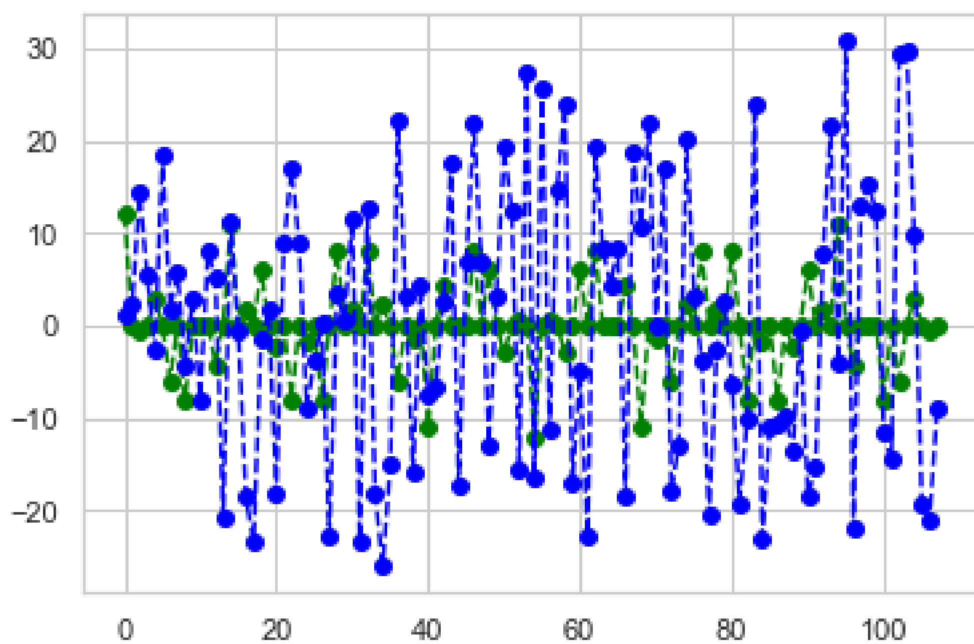
```
Out[69]: <matplotlib.collections.PathCollection at 0x2b50d165400>
```




```
In [70]: #create a dataframe for the frequency and amplitude
         processedData = frequency.to_frame()
         processedData['amplitude'] = amplitude
         processedData.columns = ['frequency','amplitude']

In [71]: #try to clear the values
         processedData.amplitude[processedData.amplitude < 2.834] = 0
         tempData = processedData.copy()
         tempData.frequency = tempData.frequency + 0.5
         #save this for testing
         testTemp = processedData.copy()
         #check on size to make sure didn't go wrong
         tempData.size

Out[71]: 108

In [72]: #combine graph together
         processedData = processedData.append(tempData)
         processedData.size

Out[72]: 216

In [73]: #draw the refined graph
         plt.plot(processedData['frequency'],processedData['amplitude'])
         plt.show()
```



In [74]:
```
#return the graph pattern
processedGraph = (np.fft.ifft(processedData['amplitude'])*108)
processedSales = trainingData_detrend
plt.plot(processedGraph,marker='o',color="green",linestyle='--')
plt.plot(processedSales,marker='o',color="blue",linestyle='--')
plt.show()
```



```
In [75]: #Original Data(blue line)
         pd.DataFrame(processedSales).head()
```

```
Out[75]:            0
         0   1.104825
         1   2.187151
         2  14.269477
         3   5.351803
         4  -2.565871
```

```
In [76]: a=pd.DataFrame(processedSales).diff()
         a.head()
```

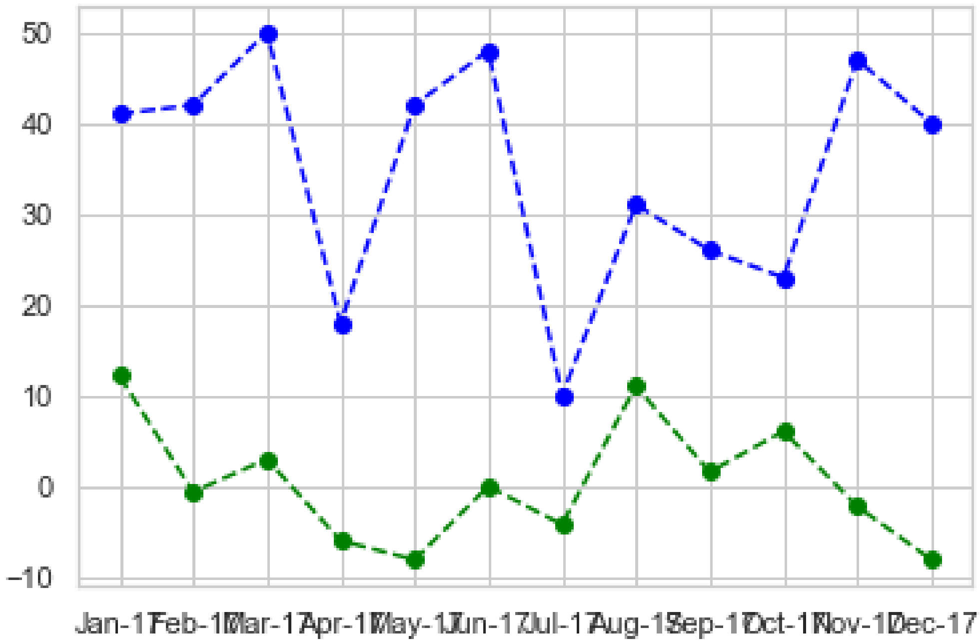
```
Out[76]:            0
         0        NaN
         1   1.082326
         2  12.082326
         3  -8.917674
         4  -7.917674
```

```
In [77]: b=pd.DataFrame(processedGraph.real).diff()
         b.head()
```

```
Out[77]:            0
         0        NaN
         1 -12.172603
         2  -0.601685
         3   0.601685
         4   3.006612
```

```
In [78]: result=pd.concat([a,b],axis=1)
         result
         result.to_csv(r'C://Users/Leon Wong/Desktop/resultMiddleAge.csv')
```

```
In [79]: testGraph = (np.fft.ifft(testTemp['amplitude'])*108)[0:12]
         plt.plot(testGraph,marker='o',color="green",linestyle='--')
         plt.plot(testData['Month'],testData['Middle age sales'],marker='o',color="blue",lines
         plt.show()
```



In [80]:
```
testDataTable = pd.DataFrame(testData['Middle age sales']).diff()
testDataTable = testDataTable.reset_index(drop = True)
testDataTable.head()
```

Out[80]:

	Middle age sales
0	NaN
1	1.0
2	8.0
3	-32.0
4	24.0

In [81]:
```
testGraphTable = pd.DataFrame(testGraph.real).diff()
testGraphTable.head()
```

Out[81]:

	0
0	NaN
1	-12.774288
2	3.608297
3	-8.973551
4	-2.119147

In [82]:
```
testResult=pd.concat([testDataTable,testGraphTable],axis=1)
testResult.head()
```

Out[82]:

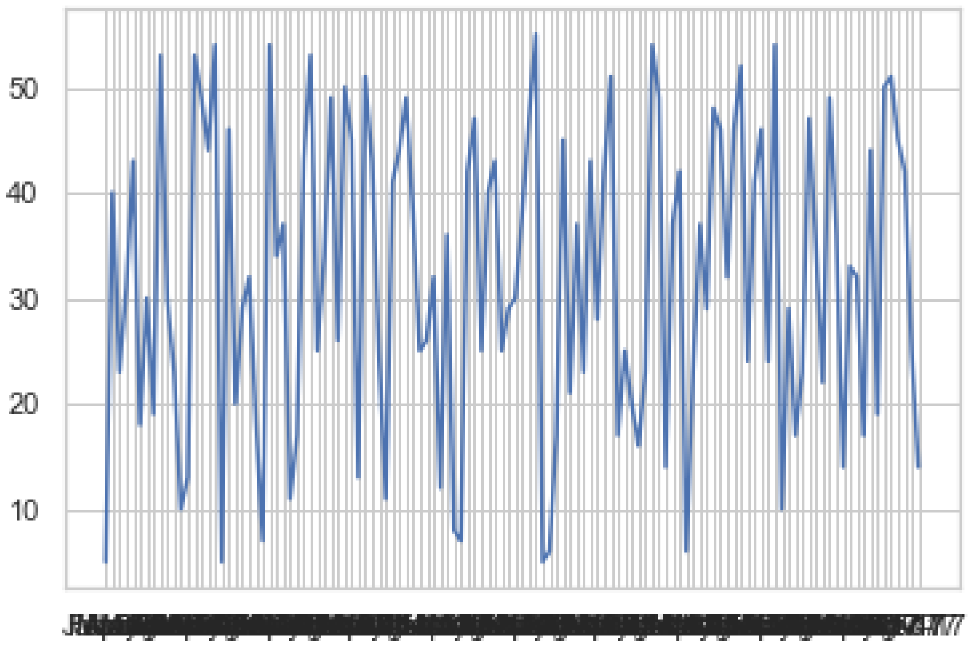
	Middle age sales	0
0	NaN	NaN

```
1                1.0 -12.774288
2                8.0   3.608297
3              -32.0  -8.973551
4               24.0  -2.119147
```

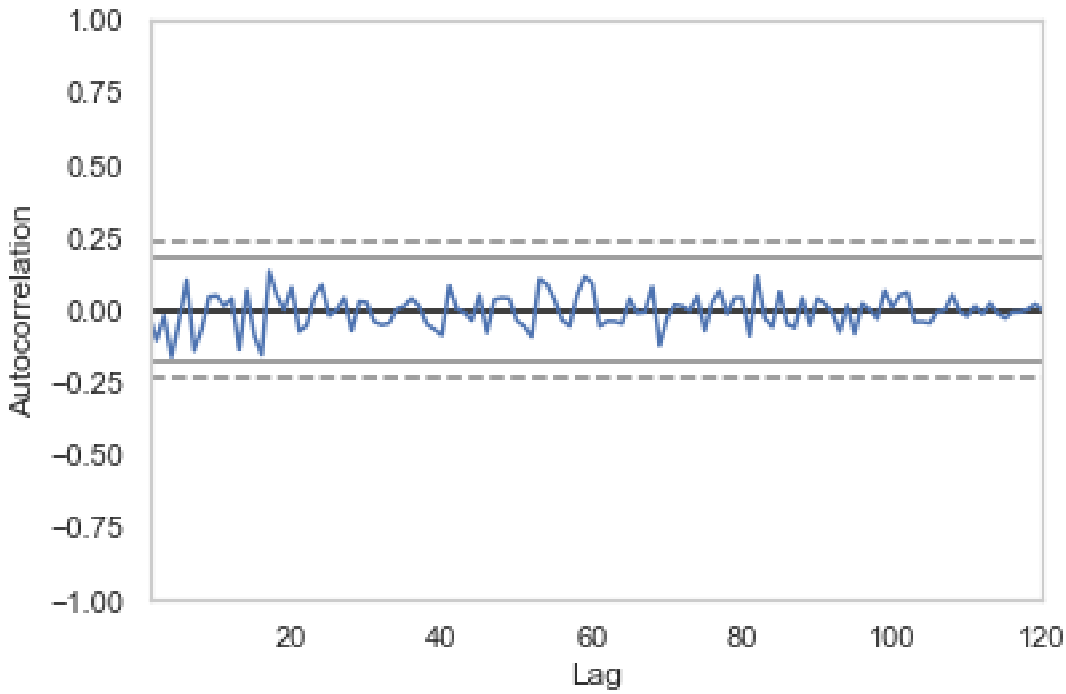
```
In [83]: testResult.to_csv(r'C://Users/Leon Wong/Desktop/testMiddleResult.csv')
```

4.4 Modelling with Senior Citizens Sales

```
In [84]: elderlySalesTable1 = elderlySalesdata.iloc[0:120]
         x = elderlySalesTable1['Month']
         y = elderlySalesTable1['Senior citizen sales']
         plt.plot(x,y)
         plt.show()
```




```
In [85]: elderlySalesTable1 = elderlySalesTable1.drop('Product Code',axis=1)
         arimaTable = elderlySalesTable1.set_index(['Month'])
         autocorrelation_plot(arimaTable)
         plt.show()
```



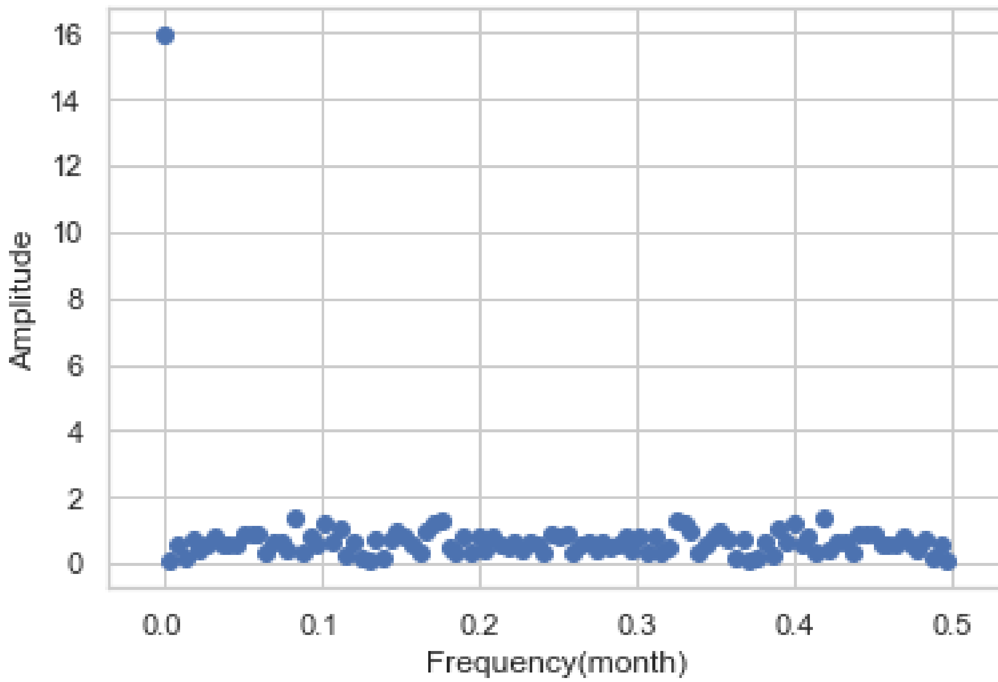

```
In [86]: #train test split
         trainingData = elderlySalesTable1.iloc[0:108]
         testData = elderlySalesTable1.iloc[108:120]
         print(trainingData.head(),testData.head())
```

```
    Month  Senior citizen sales
0  Jan-08                     5
1  Feb-08                    40
2  Mar-08                    23
3  Apr-08                    31
4  May-08                    43       Month  Senior citizen sales
108  Jan-17                    14
109  Feb-17                    33
110  Mar-17                    32
111  Apr-17                    17
112  May-17                    44
```

```
In [87]: fft = np.fft.fft(trainingData['Senior citizen sales'])
         N = trainingData.size
         print(N.size)
         f = np.linspace(0,1,N)
         print(f.size)
         plt.ylabel("Amplitude")
```

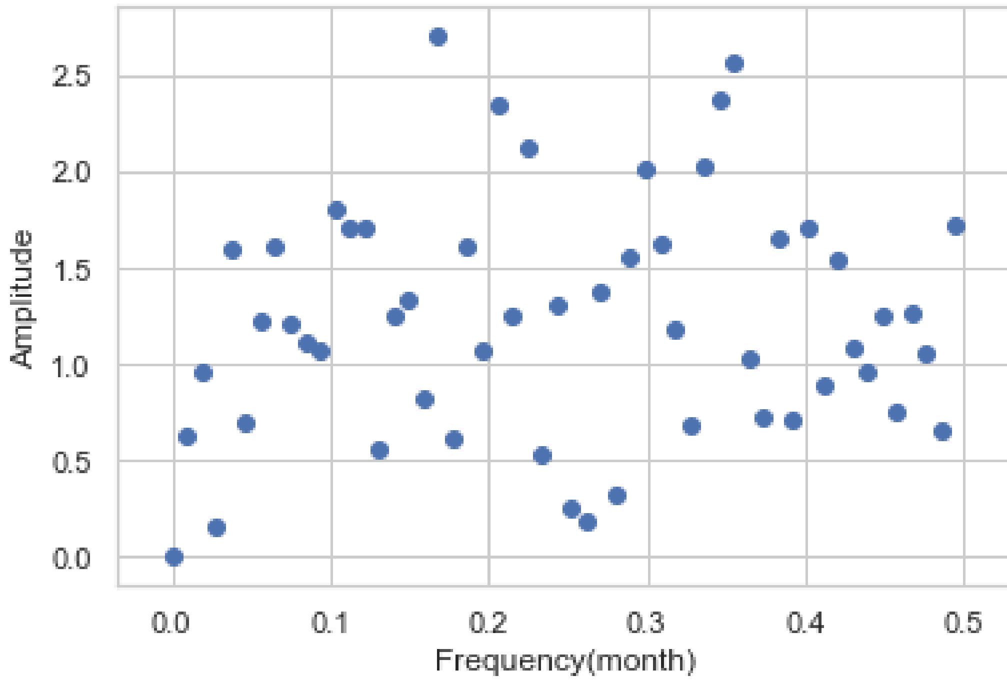
```
plt.xlabel("Frequency(month)") #month
frequency = f[:N//2]
amplitude = np.abs(fft[:N//2] * (1/N)) # 1 / N is a normalization factor
plt.scatter(frequency, amplitude)
plt.show()
```

```
1
216
```



In [88]:
```
from scipy import signal
#simple one line code
trainingData_detrend = signal.detrend(trainingData['Senior citizen sales'])
```

In [89]:
```
#compute fft again with the detrended data
fft = np.fft.fft(trainingData_detrend)
#this time round we have the table drawn
N = trainingData_detrend.size
f = np.linspace(0,1,N)
plt.ylabel("Amplitude")
plt.xlabel("Frequency(month)")
frequency = f[:N//2]
amplitude = np.abs(fft[:N//2] * (1/N)) # 1 / N is a normalization factor
plt.scatter(frequency, amplitude)
plt.show()
```




```
In [90]: #first two seems to be ahead of others. We extract it
         frequency = pd.Series(frequency)
         amplitude = pd.Series(amplitude)
         #getting the y of the significant stuff
         significantAmplitudes = amplitude.nlargest(3)
         print(significantAmplitudes)
```

```
18    2.712367
38    2.576185
37    2.377197
dtype: float64
```

```
In [91]: #create a dataframe for the frequency and amplitude
         processedData = frequency.to_frame()
         processedData['amplitude'] = amplitude
         processedData.columns = ['frequency','amplitude']
```

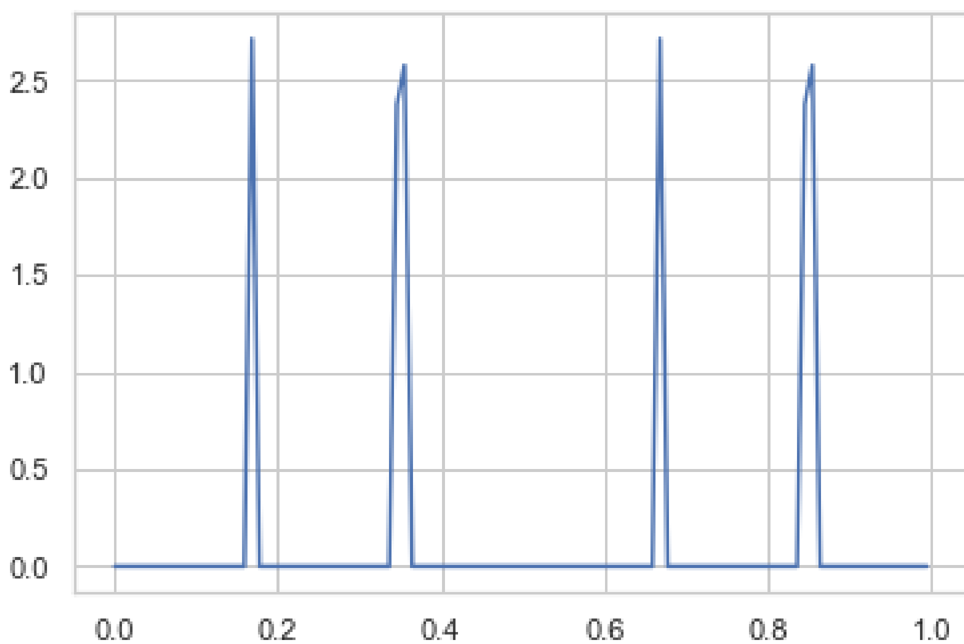
```
In [92]: #try to clear the values
         processedData.amplitude[processedData.amplitude < 2.377] = 0
         tempData = processedData.copy()
         tempData.frequency = tempData.frequency + 0.5
         #save this for testing
         testTemp = processedData.copy()
         #check on size to make sure didn't go wrong
         tempData.size
```

Out[92]: 108

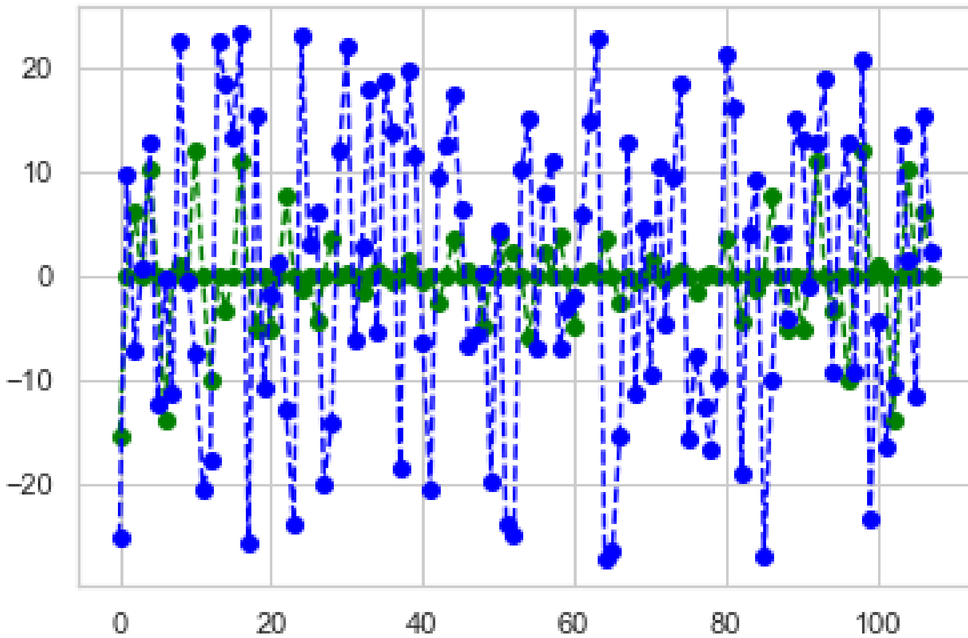
In [93]:
```
#combine graph together
processedData = processedData.append(tempData)
processedData.size
```

Out[93]: 216

In [94]:
```
#draw the refined graph
plt.plot(processedData['frequency'],processedData['amplitude'])
plt.show()
```



In [95]:
```
#return the graph pattern
processedGraph = -(np.fft.ifft(processedData['amplitude'])*108)
processedSales = trainingData_detrend
plt.plot(processedGraph,marker='o',color="green",linestyle='--')
plt.plot(processedSales,marker='o',color="blue",linestyle='--')
plt.show()
```

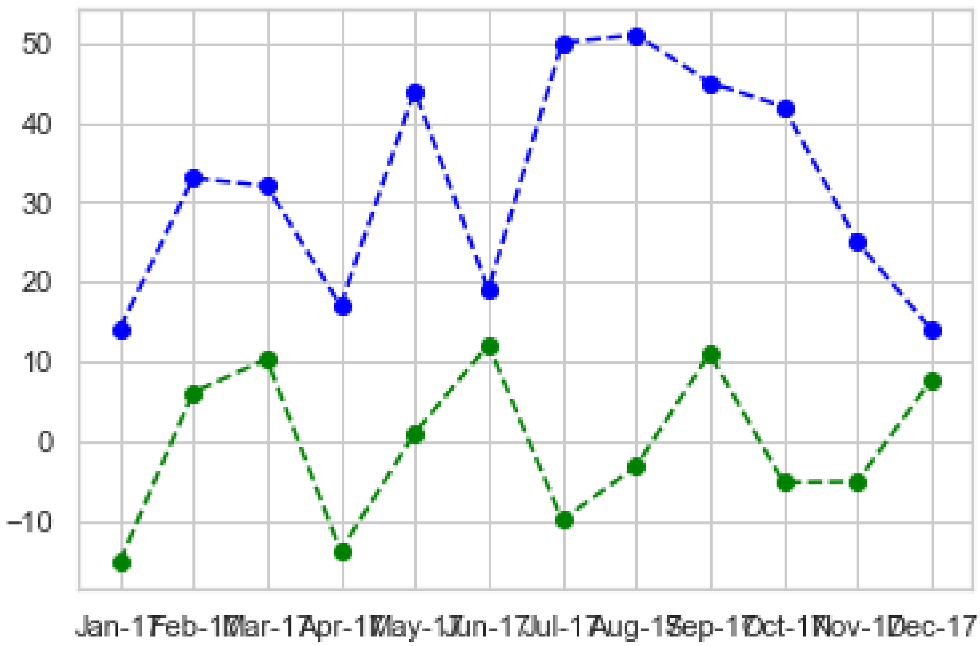



```
In [96]: #Original Data(blue line)
         pd.DataFrame(processedSales).head()

Out[96]:            0
         0 -25.169385
         1   9.798129
         2  -7.234358
         3   0.733156
         4  12.700669

In [97]: a=pd.DataFrame(processedSales).diff()
         a.head()

Out[97]:            0
         0        NaN
         1  34.967514
         2 -17.032486
         3   7.967514
         4  11.967514

In [98]: b=pd.DataFrame(processedGraph.real).diff()
         b.head()

Out[98]:            0
         0        NaN
         1  15.331498
```

```
2    6.073203
3   -6.073203
4   10.279772
```

In [99]:
```
result=pd.concat([a,b],axis=1)
result
result.to_csv(r'C://Users/Leon Wong/Desktop/resultSeniorAge.csv')
```

In [100]:
```
testGraph = -(np.fft.ifft(testTemp['amplitude'])*108)[0:12]
plt.plot(testGraph,marker='o',color="green",linestyle='--')
plt.plot(testData['Month'],testData['Senior citizen sales'],marker='o',color="blue",l
plt.show()
```



In [101]:
```
testDataTable = pd.DataFrame(testData['Senior citizen sales']).diff()
testDataTable = testDataTable.reset_index(drop = True)
testDataTable.head()
```

Out[101]:

	Senior citizen sales
0	NaN
1	19.0
2	-1.0
3	-15.0
4	27.0

In [102]:
```
testGraphTable = pd.DataFrame(testGraph.real).diff()
testGraphTable.head()
```

```
Out[102]:            0
          0        NaN
          1  21.404701
          2   4.206570
          3 -24.119120
          4  14.787409
```

```
In [103]: testResult=pd.concat([testDataTable,testGraphTable],axis=1)
          testResult.head()
```

```
Out[103]:    Senior citizen sales          0
          0                   NaN        NaN
          1                  19.0  21.404701
          2                  -1.0   4.206570
          3                 -15.0 -24.119120
          4                  27.0  14.787409
```

```
In [104]: testResult.to_csv(r'C://Users/Leon Wong/Desktop/testSeniorResult.csv')
```

5 Evaluation

5.1 Total Sales

This model has been run on different number of significant frequencies. In order to not overfit the graph, a maximum number of significant frequencies of 5 is decided. It is found that 3 significant frequencies proves to be best for this model, where the ifft method is inversed, with an training set accuracy of 57%, followed by the testing set accuracy of 36.36%. The following table shows the confusion matrices of the training set and testing set for this model.

Training set:

Attributes	Predicted positive	Predicted negative	Total
Actual positive	29	24	53
Actual negative	22	32	54
Total	41	56	107

Testing set:

Attributes	Predicted positive	Predicted negative	Total
Actual positive	1	4	5
Actual negative	3	3	6
Total	4	7	11

From this results, this shows that the total sales cannot be used to generate a useful model with significant accuracy. Thus, we have to use other attributes to build the model.

5.2 Youth Sales

After testing on different significant frequencies, it is found that 2 is the best number without inverse, with the accuracy of training set being 58.88%, while the training set being 63.63%. The confusion matrices are shown as below.

Training set:

Attributes	Predicted positive	Predicted negative	Total
Actual positive	31	22	53
Actual negative	22	32	54
Total	53	54	107

Testing set:

Attributes	Predicted positive	Predicted negative	Total
Actual positive	5	2	7
Actual negative	2	2	4
Total	7	4	11

5.3 Middle Age Sales

After testing on different significant frequencies, it is found that 2 is the best number without inverse, with the accuracy of training set being 58.88%, while the training set being 63.63%. The confusion matrices are shown as below.

Training set:

Attributes	Predicted positive	Predicted negative	Total
Actual positive	32	24	56
Actual negative	20	31	51
Total	52	55	107

Testing set:

Attributes	Predicted positive	Predicted negative	Total
Actual positive	3	3	6
Actual negative	1	4	5
Total	4	7	11

5.4 Senior Citizens Sales

After testing on different significant frequencies, it is found that 2 is the best number inverse, with the accuracy of training set being 50.46%, while the training set being 45.45%. The confusion matrices are shown as below.

Training set:

Attributes	Predicted positive	Predicted negative	Total
Actual positive	30	29	59
Actual negative	24	24	48
Total	54	53	107

Testing set:

Attributes	Predicted positive	Predicted negative	Total
Actual positive	4	1	5
Actual negative	5	1	6
Total	9	2	11

6 Conclusion

Based on the following evaluations, we have found out modelling using the different sales categories yield better results compared to the total sales. However, as all data do not have a specific trend, thus the accuracy is significantly lower. Regardless of the issue, we are able to predict the sales for the future 3 months. In this case, we will predict the youth sales.

We will need to call back our model.

```
In [122]: trainingData = youthSalesTable1.iloc[0:108]
          testData = youthSalesTable1.iloc[108:120]
          #try to run on fft
          fft = np.fft.fft(trainingData['Youth Sales'])
          N = 108
          f = np.linspace(0,1,N) #month
          frequency = f
          amplitude = np.abs(fft * (1/N)) # 1 / N is a normalization factor
          from scipy import signal
          #simple one line code
          trainingData_detrend = signal.detrend(trainingData['Youth Sales'])
          #compute fft again with the detrended data
          fft = np.fft.fft(trainingData_detrend)
          #this time round we have the table drawn
          N = trainingData_detrend.size
          f = np.linspace(0,1,N)
          frequency = f[:N//2]
          #divide by two as one half of it is useful
          amplitude = np.abs(fft[:N//2] * (1/N)) # 1 / N is a normalization factor
          #first three seems to be ahead of others. We extract it
          frequency = pd.Series(frequency)
          amplitude = pd.Series(amplitude)
          #getting the y of the significant stuff
          significantAmplitudes = amplitude.nlargest(2)
          #getting the x of the significant stuff
```

```
significantFrequencies = [frequency[23],frequency[44]]
#ifft
#create a dataframe for the frequency and amplitude
processedData = frequency.to_frame()
processedData['amplitude'] = amplitude
processedData.columns = ['frequency','amplitude']
#bring another half
```

In [123]:
```
processedData.amplitude[processedData.amplitude < 3.969] = 0
tempData = processedData.copy()
estimateData = processedData.copy()
tempData.frequency = tempData.frequency + 0.5
estimateData.frequency = estimateData.frequency + 1.0
estimateData = estimateData.iloc[0:3]
#save this for testing
testTemp = processedData.copy()
testTemp.head()
```

Out[123]:

	frequency	amplitude
0	0.000000	0.0
1	0.009346	0.0
2	0.018692	0.0
3	0.028037	0.0
4	0.037383	0.0

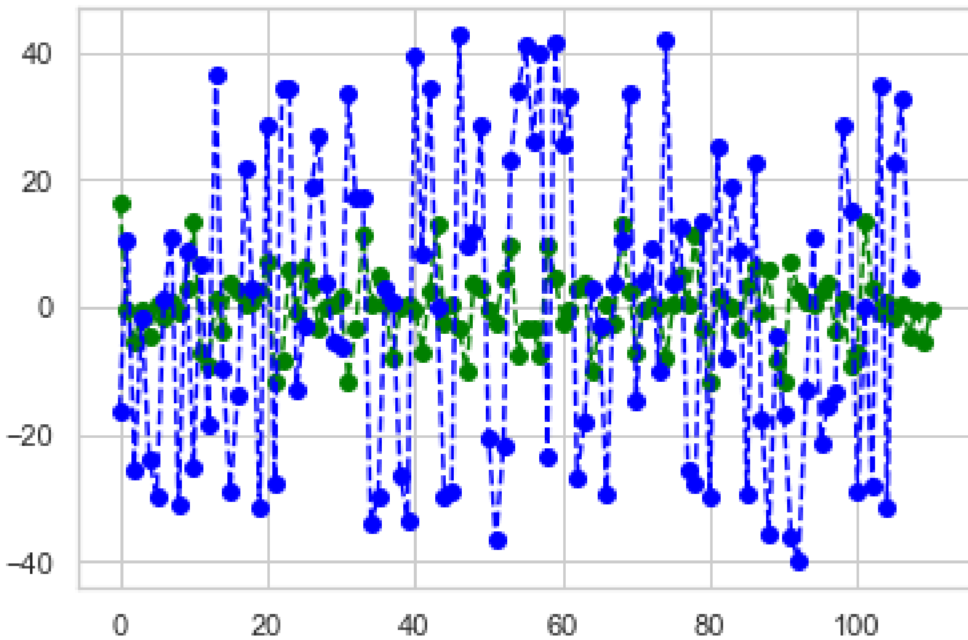
In [124]:
```
processedData = processedData.append(tempData)
processedData.size
```

Out[124]: 216

In [125]:
```
processedData = processedData.append(estimateData)
processedData.size
```

Out[125]: 222

In [126]:
```
processedGraph = (np.fft.ifft(processedData['amplitude'])*108)
processedSales = trainingData_detrend
plt.plot(processedGraph,marker='o',color="green",linestyle='--')
plt.plot(processedSales,marker='o',color="blue",linestyle='--')
plt.show()
```



Our accuracy is not significant, particularly to the lack of trend of data. Our model requires further improvement by the addition of more records, and more columns to actually get to the reason why such a data trend occurs. With this, we end our research.