In [1]:
```
# Import the modules required

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O


import matplotlib as mpl
import matplotlib.pyplot as plt   # data visualization
import seaborn as sns             # statistical data visualization

# Import the dataset required

df = pd.read_csv('Month_Value_1.csv')

print(df.head(5))
```

```
       Period       Revenue  Sales_quantity  Average_cost  \
0  01.01.2015  1.601007e+07         12729.0   1257.763541
1  01.02.2015  1.580759e+07         11636.0   1358.507000
2  01.03.2015  2.204715e+07         15922.0   1384.697024
3  01.04.2015  1.881458e+07         15227.0   1235.606705
4  01.05.2015  1.402148e+07          8620.0   1626.621765

   The_average_annual_payroll_of_the_region
0                                30024676.0
1                                30024676.0
2                                30024676.0
3                                30024676.0
4                                30024676.0
```

In [7]:
```
# Understand the data set and perform appropriate data cleaning
# (i) Check the data types
# (ii) Check for null values
# (iii) Check for outliers

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96 entries, 0 to 95
Data columns (total 5 columns):
 #   Column                                    Non-Null Count  Dtype
---  ------                                    --------------  -----
 0   Period                                    96 non-null     object
 1   Revenue                                   64 non-null     float64
 2   Sales_quantity                            64 non-null     float64
 3   Average_cost                              64 non-null     float64
 4   The_average_annual_payroll_of_the_region  64 non-null     float64
dtypes: float64(4), object(1)
memory usage: 3.9+ KB
```

In [15]:
```
# (i) Here, all the data types are appropriate
# (ii) There are some null values (with only one column filled  and the rest being nul

df = df.dropna()
df.info()
```

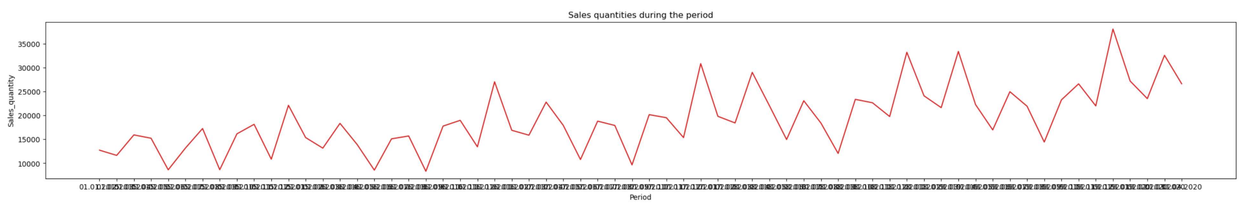
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 64 entries, 0 to 63
Data columns (total 5 columns):
 #   Column                                   Non-Null Count  Dtype
---  ------                                   --------------  -----
 0   Period                                   64 non-null     object
 1   Revenue                                  64 non-null     float64
 2   Sales_quantity                           64 non-null     float64
 3   Average_cost                             64 non-null     float64
 4   The_average_annual_payroll_of_the_region 64 non-null     float64
dtypes: float64(4), object(1)
memory usage: 3.0+ KB
```

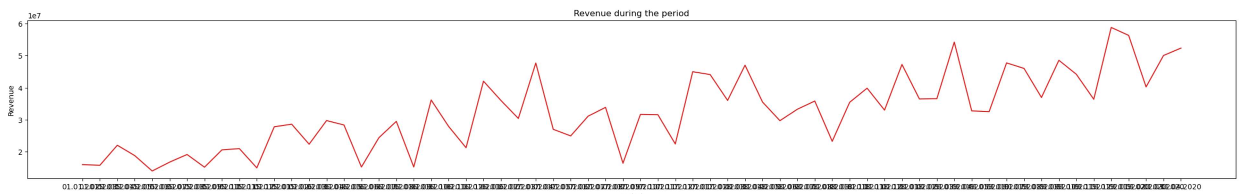
In [16]:

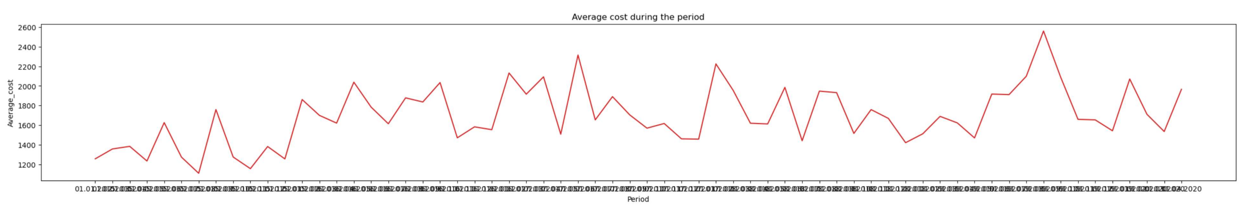
```python
# Plot the graphs of different features against time to observe any trends and seasona

def plot_df(df, x, y,  xlabel, ylabel, title = "", dpi = 100):
    plt.figure(figsize=(30, 4), dpi=dpi)
    plt.plot(x, y, color='tab:red')
    plt.gca().set(title=title, xlabel=xlabel, ylabel=ylabel)
    plt.show()


plot_df(df, x=df['Period'], y=df['Sales_quantity'], title='Sales quantities during the
plot_df(df, x=df['Period'], y=df['Revenue'], title='Revenue during the period', xlabel
plot_df(df, x=df['Period'], y=df['Average_cost'], title='Average cost during the perio
```







In [17]:

```python
# Augmented Dickey Fuller test (ADF Test)

from statsmodels.tsa.stattools import adfuller

data = df['Revenue'].values
pvalue = adfuller(data)[1]
if pvalue < 0.05:
    print("Series is stationary")
else:
    print("Series is non stationary")
```

```
Series is non stationary
```

In [18]:

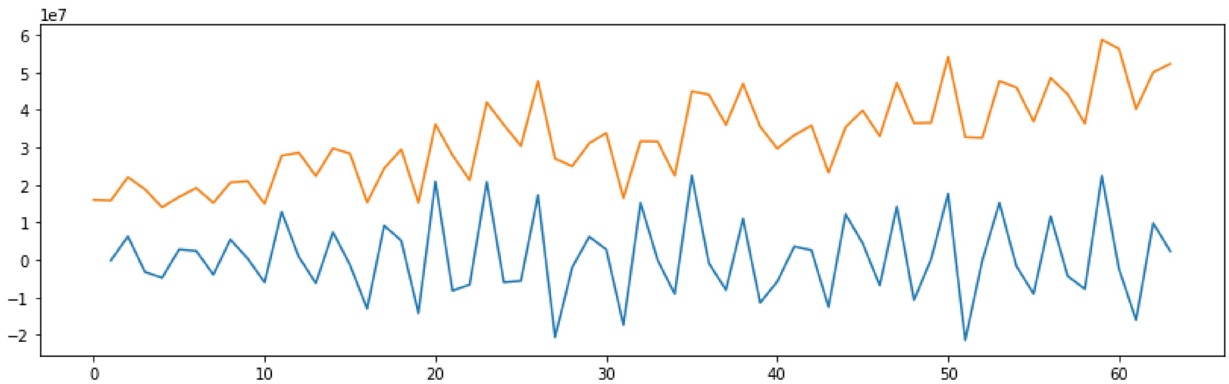
```python
# converting non stationary data to stationary using differencing
```

```
df["diff_1"] = df["Revenue"].diff(periods = 1)
df.head(6)

# checking whether the Revenue feature is now stationary or not
pvalue = adfuller(df["diff_1"].dropna())[1]
if pvalue < 0.05:
    print("Series is stationary")
else:
    print("Series is non stationary")

# Plot the Revenue feature before and after differencing
df["diff_1"].plot(figsize=(14, 4));
df["Revenue"].plot(figsize=(14, 4));
```

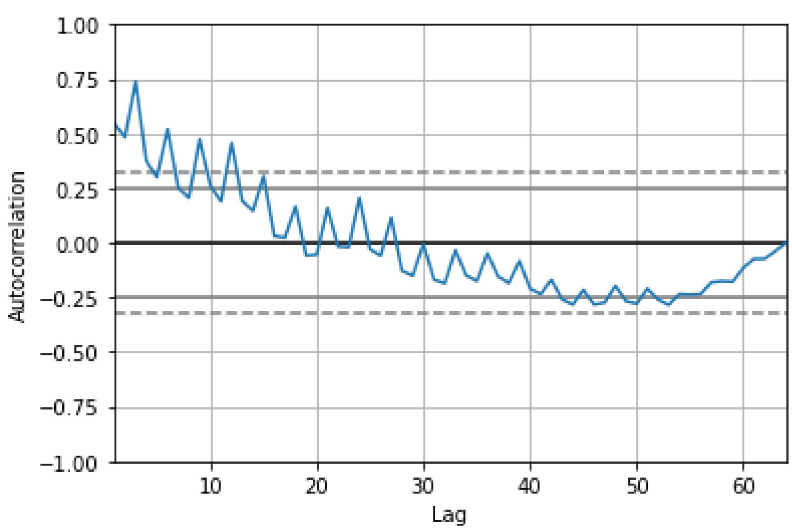
```
Series is stationary
```

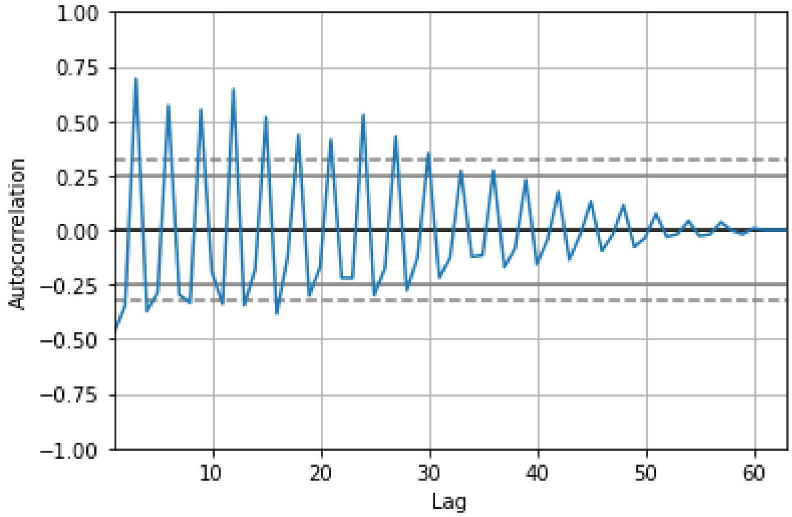


In [19]:
```
# Compare the autocorrelation plots of the Revenue feature before and after stationari

from pandas.plotting import autocorrelation_plot

autocorrelation_plot(df['Revenue'].dropna())
plt.show()
autocorrelation_plot(df['diff_1'].dropna())
plt.show()
```





In [20]:

```
# Using the ARIMA Model for forecasting

from statsmodels.tsa.arima.model import ARIMA
model=ARIMA(df['diff_1'].dropna(),order=(1,1,0))
model_fit=model.fit()
model_fit.summary()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: An unsupported index was provided and will be ignored when e.g. forecastin
g.
  self._init_dates(dates, freq)
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: An unsupported index was provided and will be ignored when e.g. forecastin
g.
  self._init_dates(dates, freq)
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: An unsupported index was provided and will be ignored when e.g. forecastin
g.
  self._init_dates(dates, freq)
```

Out[20]:

SARIMAX Results

Dep. Variable:	diff_1	**No. Observations:**	63
Model:	ARIMA(1, 1, 0)	**Log Likelihood**	-1113.880
Date:	Thu, 17 Nov 2022	**AIC**	2231.760
Time:	11:44:39	**BIC**	2236.015
Sample:	0	**HQIC**	2233.431
	- 63		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.5437	0.124	-4.383	0.000	-0.787	-0.301
sigma2	2.389e+14	1.03e-17	2.32e+31	0.000	2.39e+14	2.39e+14

Ljung-Box (L1) (Q):	13.90	**Jarque-Bera (JB):**	2.62
Prob(Q):	0.00	**Prob(JB):**	0.27
Heteroskedasticity (H):	2.18	**Skew:**	0.16
Prob(H) (two-sided):	0.08	**Kurtosis:**	2.05

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

[2] Covariance matrix is singular or near-singular, with condition number inf. Standard errors may be unstable.

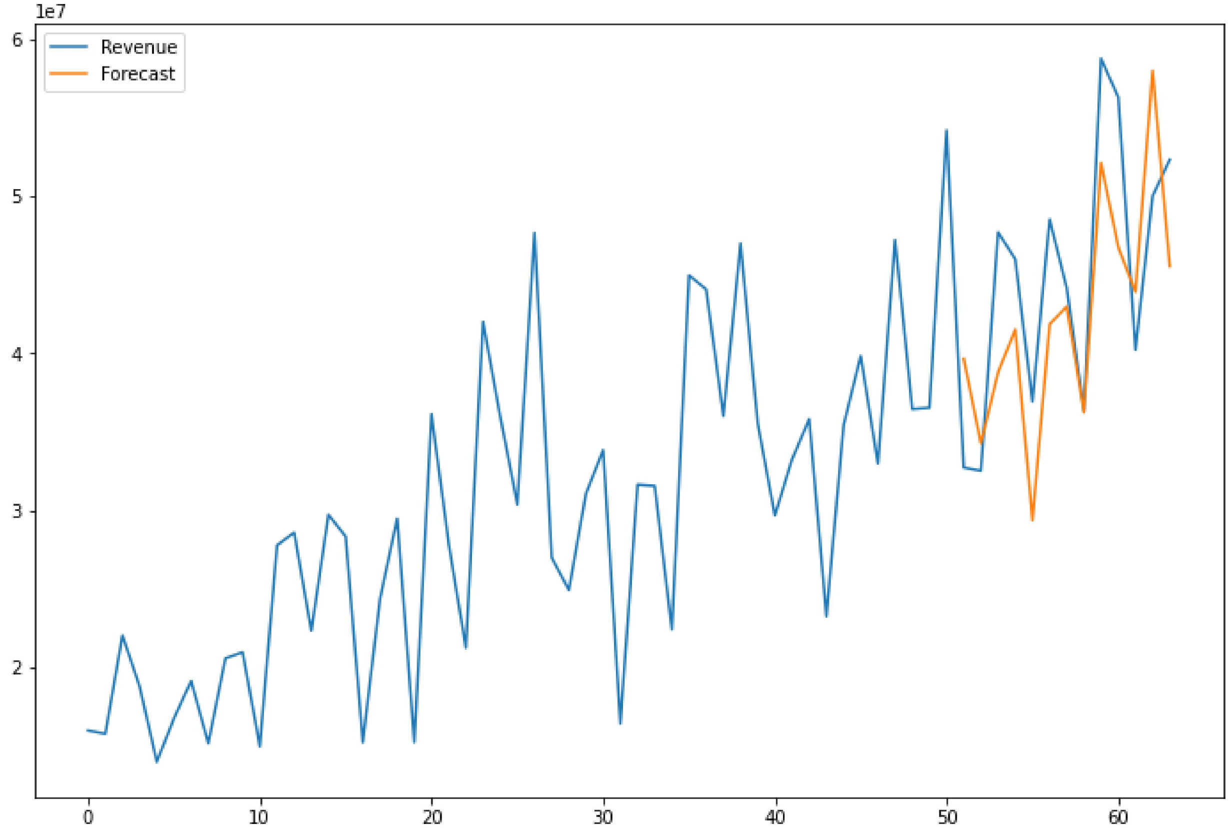
In [22]:

```
# Using the SARIMAX Model that is a seasonal variant of ARIMA to forecast Revenue feat

import statsmodels.api as sm
model = sm.tsa.statespace.SARIMAX(df['Revenue'], order=(1, 1, 1), seasonal_order=(1,1,
results = model.fit()
train_len = int(len(df['Revenue']) * 0.8)
df['Forecast'] = results.predict(start = train_len, end=len(df['Revenue']),dynamic=Tru
df[['Revenue','Forecast']].plot(figsize=(12,8))
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\statespace\sarimax.py:997:
UserWarning: Non-stationary starting seasonal autoregressive Using zeros as starting
parameters.
  warn('Non-stationary starting seasonal autoregressive'
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\statespace\sarimax.py:100
9: UserWarning: Non-invertible starting seasonal moving average Using zeros as starti
ng parameters.
  warn('Non-invertible starting seasonal moving average'
```

Out[22]: <AxesSubplot:>



In [ ]: