In [1]:

```
import pandas as pa
import numpy as np
import scipy
import statsmodels.tsa
import matplotlib.pyplot as plt
```

In [2]:

```
data = pa.read_csv(r'C:\Users\SACHIN K M\Desktop\python\data\datasets\LYNXdata.csv', header=0, names = ['year', 'trappings'], index_col = 0)
```

In [3]:

```
data.head()
```

Out[3]:

| | trappings |
|---|---|
| **year** | |
| **1821** | 269 |
| **1822** | 321 |
| **1823** | 585 |
| **1824** | 871 |
| **1825** | 1475 |

In [4]:

```
type(data)
```

Out[4]:

```
pandas.core.frame.DataFrame
```

In [5]:

```
data2 = data['trappings']
data2.head()
```

Out[5]:

```
year
1821     269
1822     321
1823     585
1824     871
1825    1475
Name: trappings, dtype: int64
```

In [6]:

```
#converting data frame into series object
#frequency aliases
#A - for year end
#B - business days
#D - day
#H - hour
#T - minutes
#S - seconds

dataseries = pa.Series(data['trappings'].values, index= pa.date_range('31/12/1821', periods = 114, freq = 'A-DEC'))
dataseries.head()
```

Out[6]:

```
1821-12-31    269
1822-12-31    321
1823-12-31    585
```

```
1824-12-31     871
1825-12-31    1475
Freq: A-DEC, dtype: int64
```

In [7]:

```
#dicky fuller test
# p value should be less than 0.05 for observation to be stationary

def stationarity_test(timeseries):
    """augumented dicky fuller test
    test for statinarity"""
    from statsmodels.tsa.stattools import adfuller
    print ("results of dicky-fuller test:")
    df_test = adfuller(timeseries, autolag = 'AIC')
    df_output = pa.Series(df_test[0:4],
                          index = ["test statestics", "p value", "#lags used", "number of observations used"])
    print (df_output)
```

In [8]:

```
stationarity_test(dataseries)
```

```
results of dicky-fuller test:
test statestics                  -2.996304
p value                           0.035241
#lags used                        7.000000
number of observations used     106.000000
dtype: float64
```

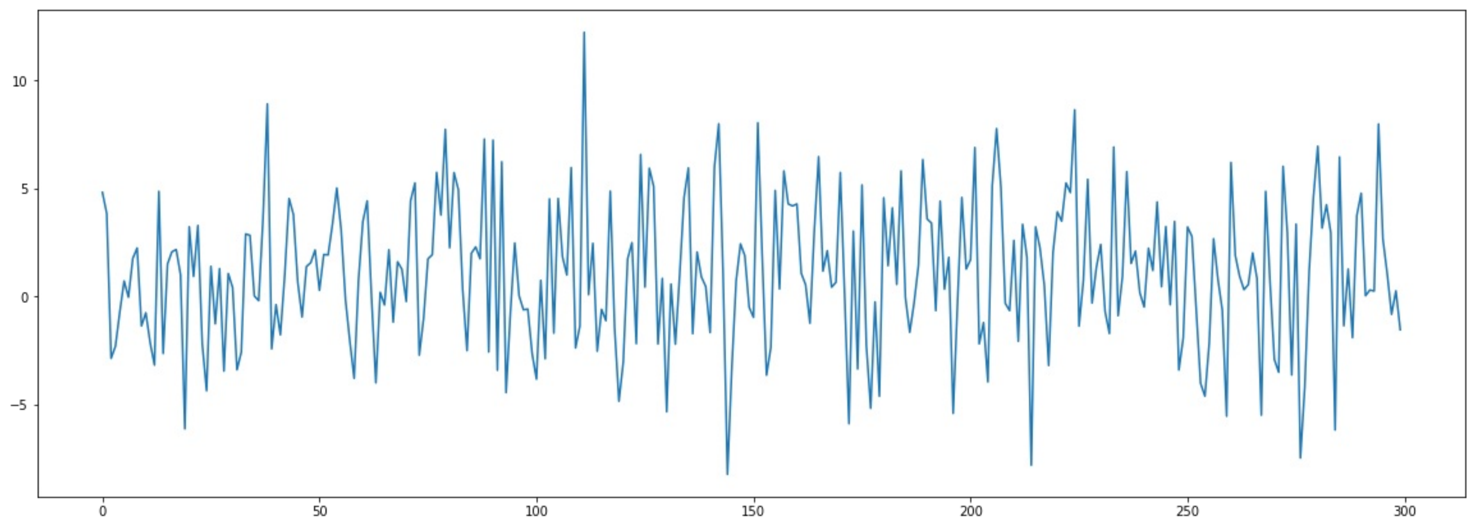
In [9]:

```
#np.random.normal(1, 3, 300) , 1 is mean, 3 standard deviation, 300 - no of observations generated randomly
stationarity_test(np.random.normal(1, 3, 300))
plt.figure(figsize=(20,7))
plt.plot(np.random.normal(1, 3, 300))
```

```
results of dicky-fuller test:
test statestics               -1.868927e+01
p value                        2.039774e-30
#lags used                     0.000000e+00
number of observations used    2.990000e+02
dtype: float64
```

Out[9]:

```
[<matplotlib.lines.Line2D at 0x13ebbca4b70>]
```

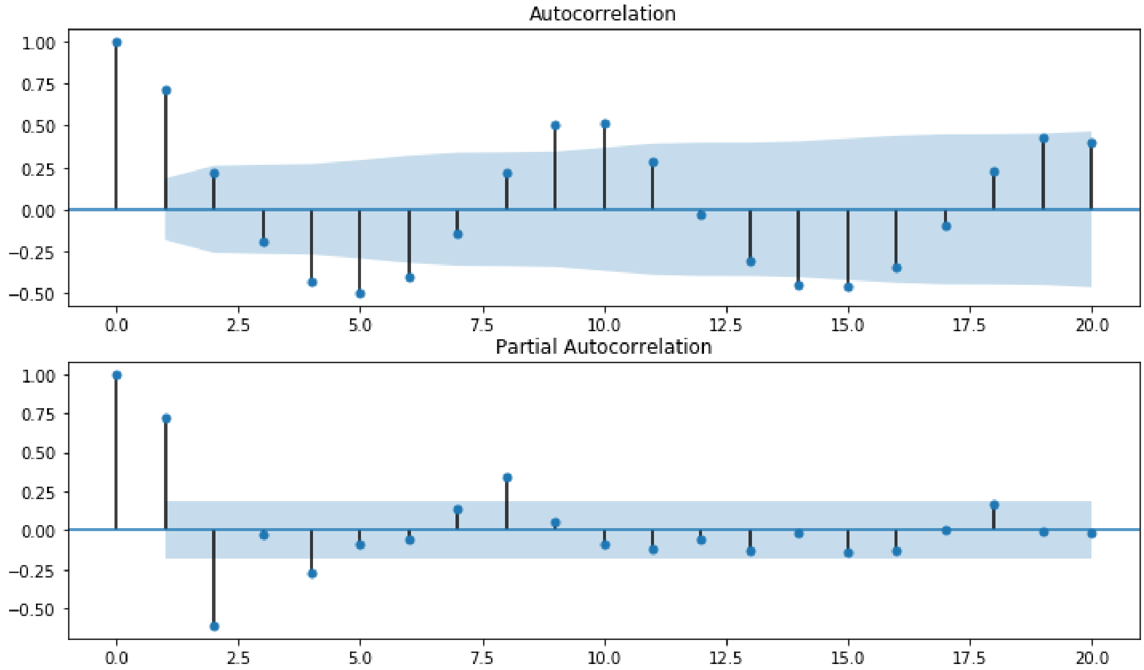



In [10]:

```
import statsmodels.graphics
from statsmodels.graphics import tsaplots
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

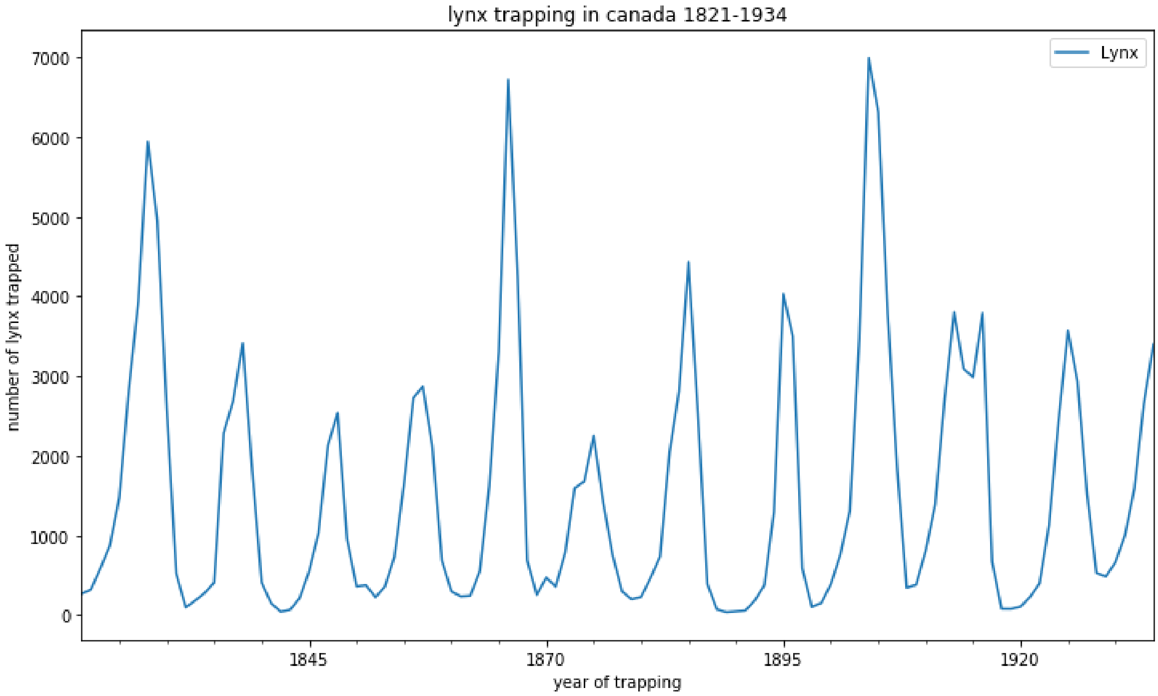
In [11]:

```
%matplotlib inline
fig = plt.figure(figsize= (12,7))
ax1 = fig.add_subplot(211)
fig = plot_acf(dataseries, lags=20, ax = ax1)
ax2 = fig.add_subplot(212)
fig = plot_pacf(dataseries, lags=20, ax = ax2)
```




In [12]:

```
plt.figure(figsize=(12,7))
dataseries.plot()
plt.title("lynx trapping in canada 1821-1934")
plt.xlabel('year of trapping')
plt.ylabel('number of lynx trapped')
plt.legend(['Lynx'])


cumsum_lynx = np.cumsum(dataseries)
```



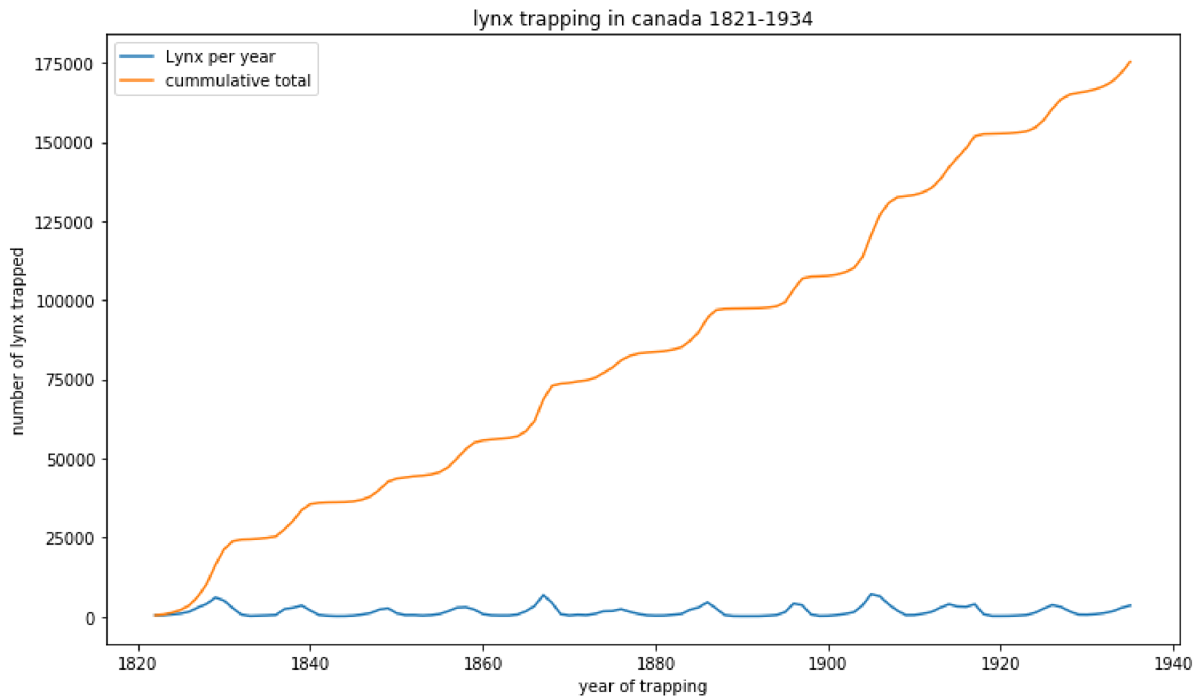

In [13]:

```
cumsum_lynx = np.cumsum(dataseries)
plt.figure(figsize=(12,7))
plt.plot(dataseries)
```

```
plt.plot(cumsum_lynx)
plt.title("lynx trapping in canada 1821-1934")
plt.xlabel('year of trapping')
plt.ylabel('number of lynx trapped')
plt.legend(['Lynx per year', 'cummulative total'])
```

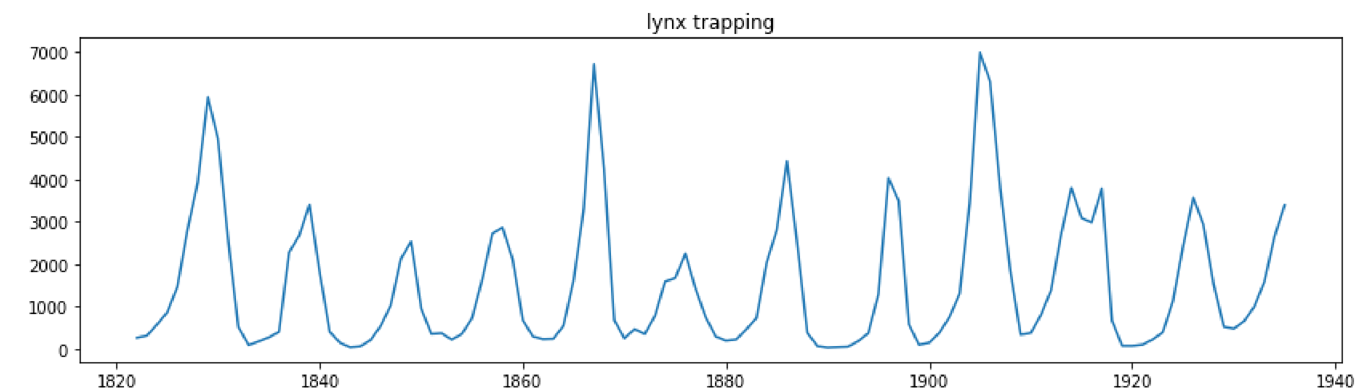
Out[13]:

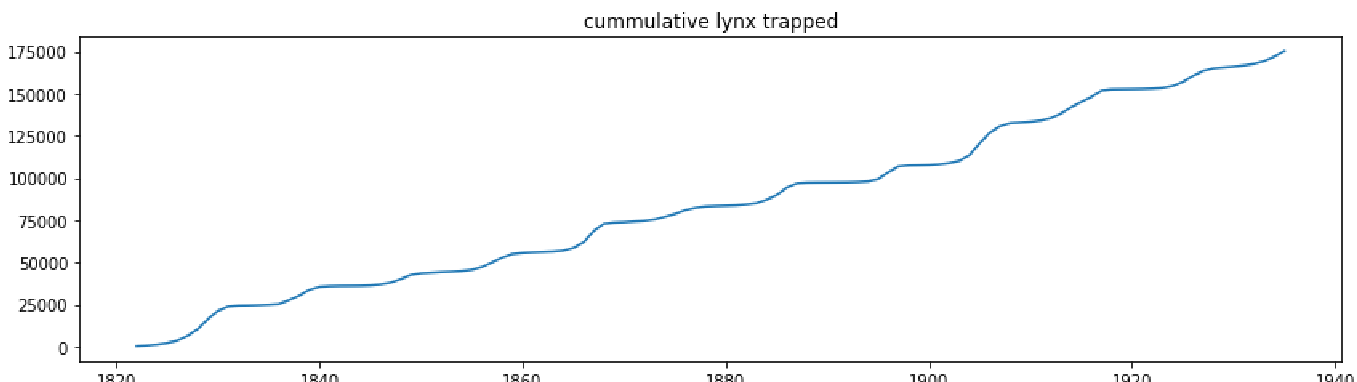
<matplotlib.legend.Legend at 0x13ebbdf78d0>




In [14]:

```
plt.figure(figsize=(12,7))

plt.subplot(2, 1, 1)
plt.plot(dataseries)
plt.title('lynx trapping')

plt.subplot(2, 1, 2)
plt.plot(cumsum_lynx)
plt.title('cummulative lynx trapped')


plt.tight_layout()
```



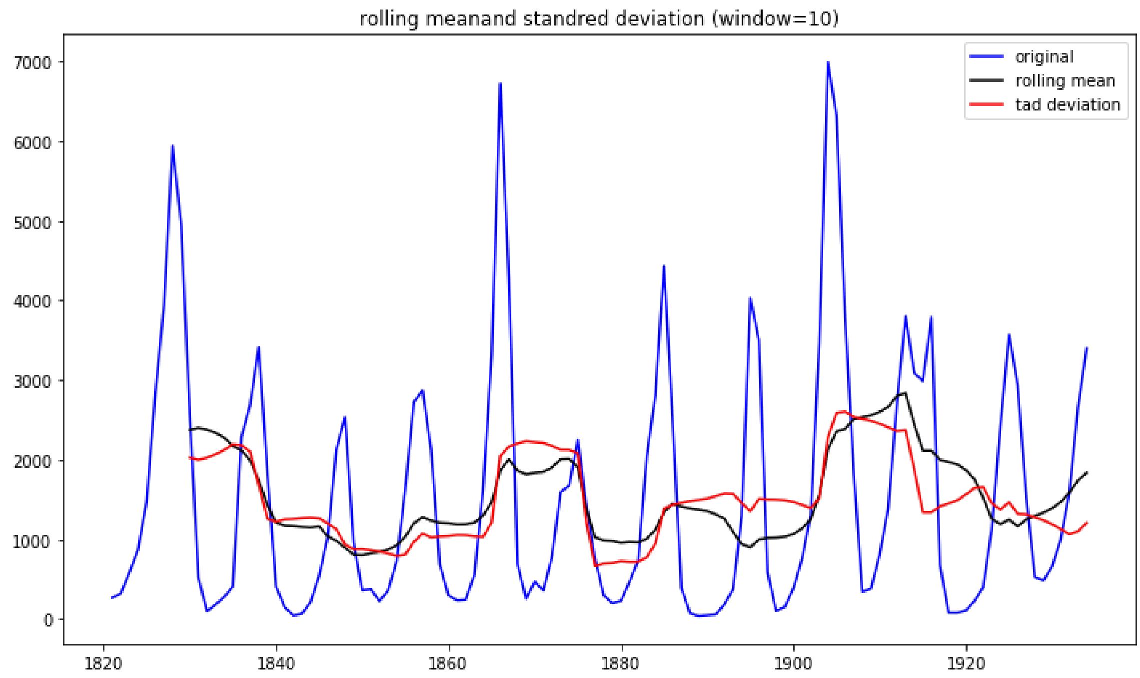



In [15]:

```
def plot_rolling(timeseries, window):
    rol_mean = timeseries.rolling(window).mean()
    rol_std = timeseries.rolling(window).std()


    plt.figure(figsize=(12,7))
    og = plt.plot(timeseries, color="blue", label="original")
    mean = plt.plot(rol_mean, color="black", label="rolling mean")
    std = plt.plot(rol_std, color="red", label="tad deviation")
    plt.legend(loc='best')
    plt.title("rolling meanand standred deviation (window="+str(window)+")")
    plt.show()
```

In [16]:

```
plot_rolling(data, 10)
```




In [17]:

```
#getting smoothing values
dataseries.rolling(10).mean()
dataseries.rolling(10, min_periods=1).mean()
```

Out[17]:

```
1821-12-31     269.000000
1822-12-31     295.000000
1823-12-31     391.666667
1824-12-31     511.500000
1825-12-31     704.200000
1826-12-31    1057.000000
1827-12-31    1467.142857
1828-12-31    2026.625000
1829-12-31    2351.444444
1830-12-31    2374.000000
1831-12-31    2399.400000
1832-12-31    2377.100000
1833-12-31    2337.000000
1834-12-31    2277.800000
1835-12-31    2171.200000
1836-12-31    2117.600000
1837-12-31    1993.300000
1838-12-31    1739.900000
1839-12-31    1427.300000
1840-12-31    1210.500000
1841-12-31    1173.300000
1842-12-31    1168.000000
1843-12-31    1156.400000
1844-12-31    1149.800000
```

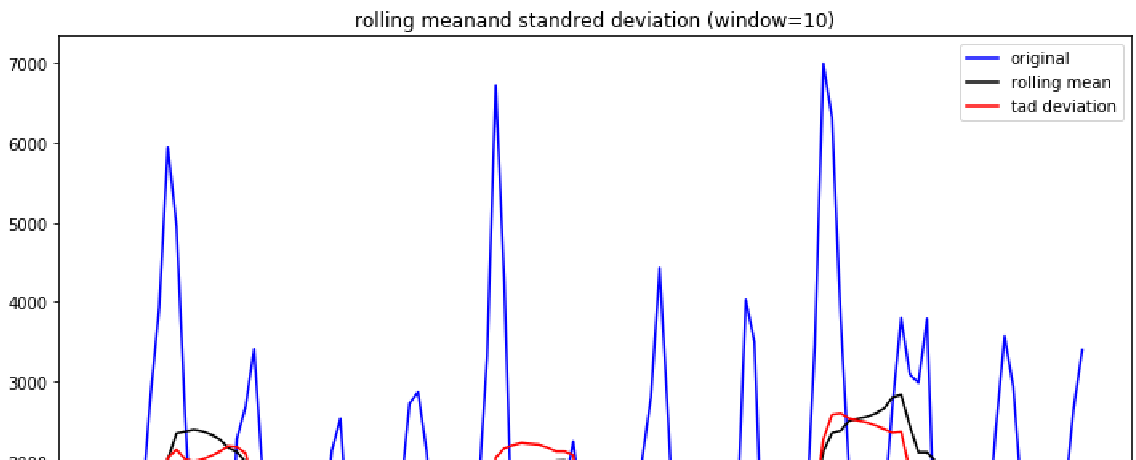
```
1845-12-31    1163.500000
1846-12-31    1038.300000
1847-12-31     982.700000
1848-12-31     895.400000
1849-12-31     808.700000
1850-12-31     803.900000
                 ...
1905-12-31    2356.100000
1906-12-31    2386.000000
1907-12-31    2510.900000
1908-12-31    2534.900000
1909-12-31    2557.800000
1910-12-31    2599.900000
1911-12-31    2662.900000
1912-12-31    2803.500000
1913-12-31    2837.000000
1914-12-31    2447.000000
1915-12-31    2114.200000
1916-12-31    2113.800000
1917-12-31    1997.600000
1918-12-31    1971.200000
1919-12-31    1941.000000
1920-12-31    1871.000000
1921-12-31    1755.100000
1922-12-31    1523.700000
1923-12-31    1256.900000
1924-12-31    1191.000000
1925-12-31    1249.900000
1926-12-31    1164.400000
1927-12-31    1250.700000
1928-12-31    1295.500000
1929-12-31    1336.000000
1930-12-31    1391.400000
1931-12-31    1468.500000
1932-12-31    1587.600000
1933-12-31    1740.100000
1934-12-31    1836.500000
Freq: A-DEC, Length: 114, dtype: float64
```

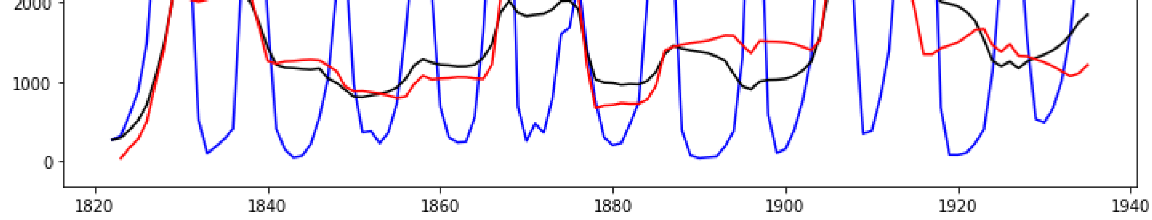
In [18]:

```python
# to avoid nan values at the wbeganing window size
dataseries.head()

# difference between plot_rolling & plot_rolling1 is min_period it will start from there

def plot_rolling1(timeseries, window):
    rol_mean = timeseries.rolling(window, min_periods=1).mean()
    rol_std = timeseries.rolling(window, min_periods=1).std()


    plt.figure(figsize=(12,7))
    og = plt.plot(timeseries, color="blue", label="original")
    mean = plt.plot(rol_mean, color="black", label="rolling mean")
    std = plt.plot(rol_std, color="red", label="tad deviation")
    plt.legend(loc='best')
    plt.title("rolling meanand standred deviation (window="+str(window)+")")
    plt.show()
```

In [19]:

```python
plot_rolling1(dataseries, 10)
```



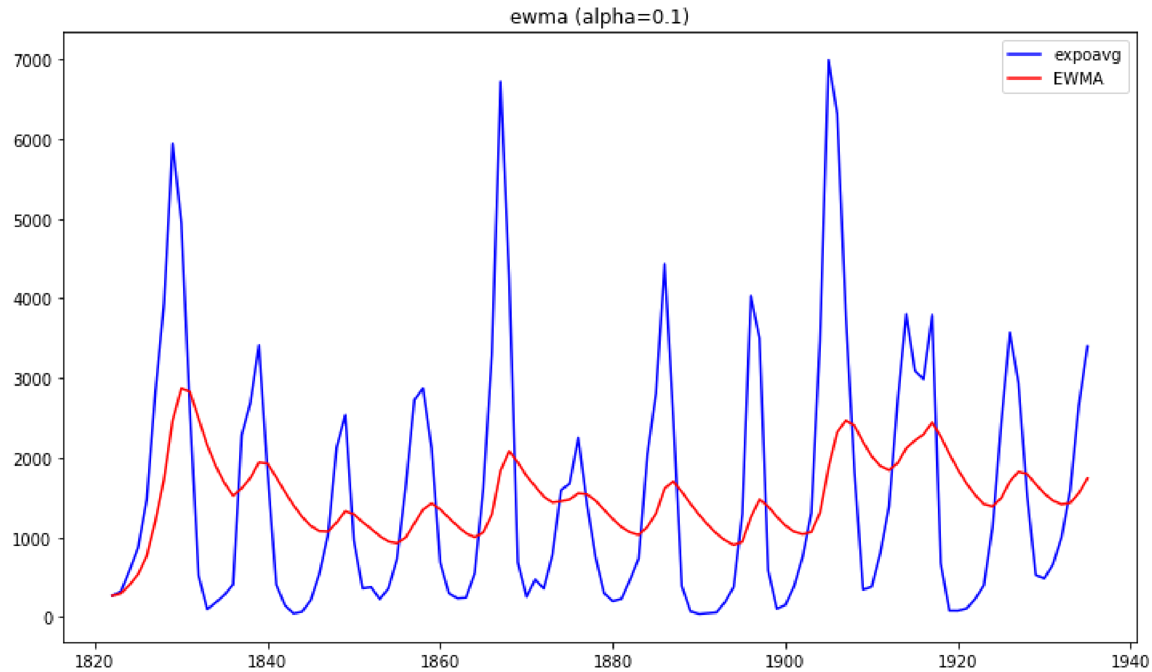


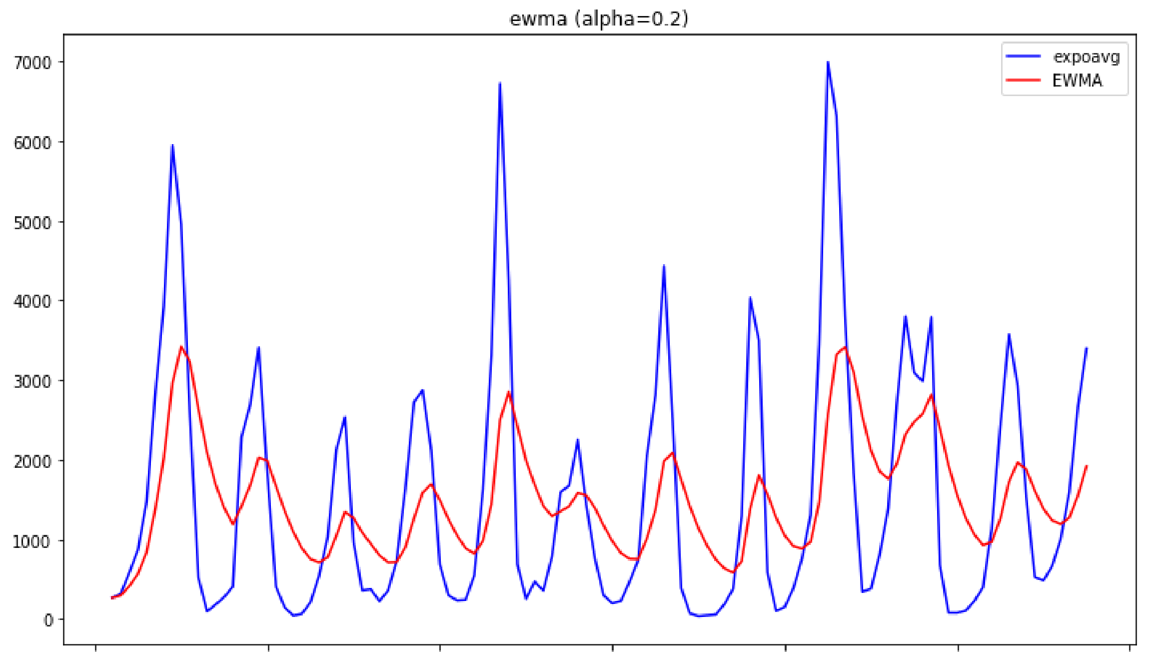

In [20]:

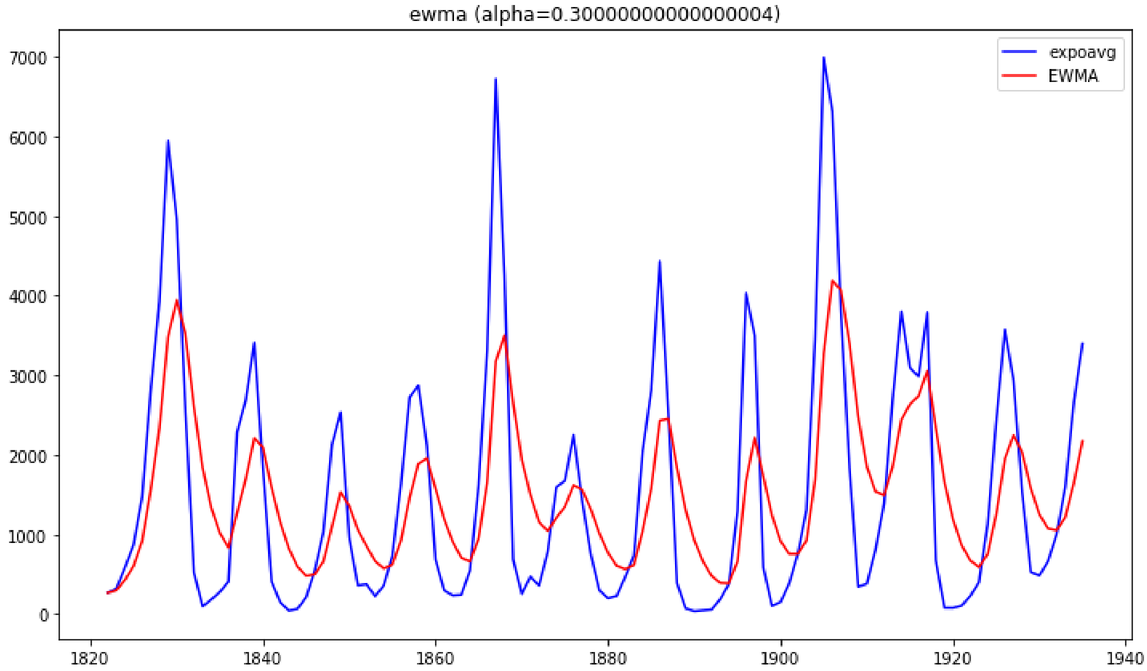
```python
#ewma (exponential weighted moving average)
#this method only for pandas data series or pandas data frame

def plot_ewma(timeseries, alpha):
    expw_ma = timeseries.ewm(alpha=alpha).mean()

    fig= plt.figure(figsize=(12,7))
    og_line = plt.plot(timeseries, color= 'blue', label='expoavg')
    exwm_avg = plt.plot(expw_ma, color='red', label='EWMA')
    plt.legend(loc='best')
    plt.title("ewma (alpha="+str(alpha)+")")
    plt.show()
```

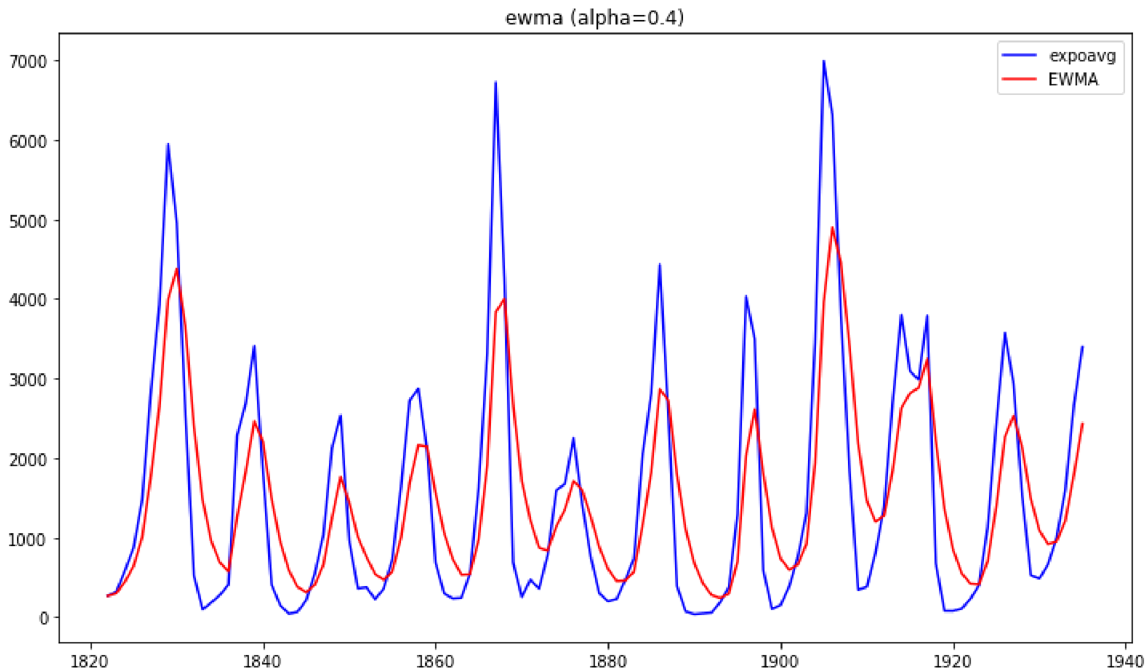
In [21]:

```python
for i in np.arange(0.1, 0.5, 0.1):
    plot_ewma(dataseries, i)
```






In [22]:

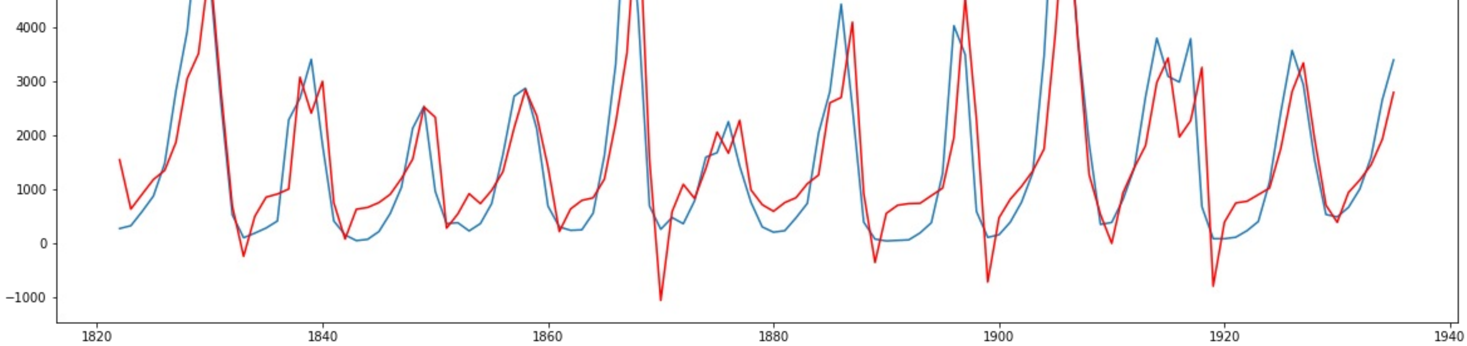
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import acf, pacf
from statsmodels.tsa.arima_model import ARIMA
```

In [23]:

```
model = ARIMA(dataseries, order=(2, 0, 0))
results_AR = model.fit()
plt.figure(figsize=(20,7))
plt.plot(dataseries)
plt.plot(results_AR.fittedvalues, color='red')
```

Out[23]:

[<matplotlib.lines.Line2D at 0x13ebc351908>]



In [24]:

```
#we will consider AIC value it should be as low as possible
#only randomness present in resudial
results_AR.summary()
```

Out[24]:

ARMA Model Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | y | **No. Observations:** | 114 |
| **Model:** | ARMA(2, 0) | **Log Likelihood** | -935.016 |
| **Method:** | css-mle | **S.D. of innovations** | 876.447 |
| **Date:** | Fri, 12 Jul 2019 | **AIC** | 1878.032 |
| **Time:** | 07:43:12 | **BIC** | 1888.977 |
| **Sample:** | 12-31-1821 | **HQIC** | 1882.474 |
| | - 12-31-1934 | | |

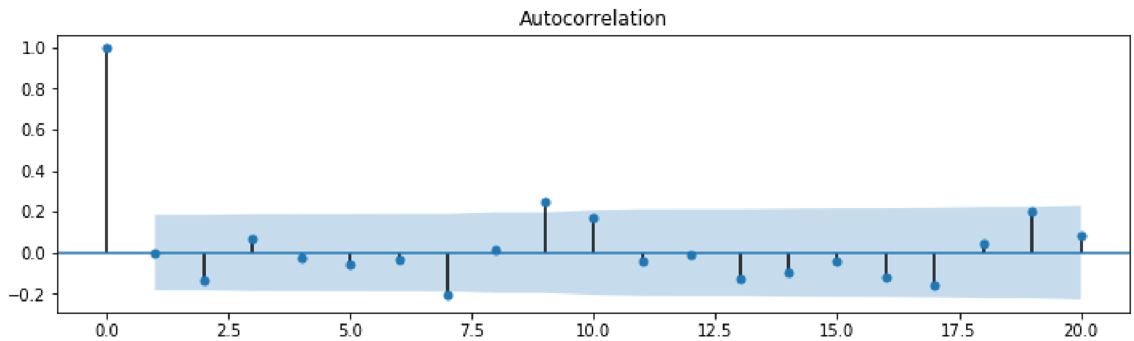
| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **const** | 1545.3385 | 181.671 | 8.506 | 0.000 | 1189.269 | 1901.408 |
| **ar.L1.y** | 1.1474 | 0.074 | 15.459 | 0.000 | 1.002 | 1.293 |
| **ar.L2.y** | -0.5997 | 0.074 | -8.110 | 0.000 | -0.745 | -0.455 |

Roots

| | Real | Imaginary | Modulus | Frequency |
|---|---|---|---|---|
| **AR.1** | 0.9566 | -0.8673j | 1.2913 | -0.1172 |
| **AR.2** | 0.9566 | +0.8673j | 1.2913 | 0.1172 |

In [25]:

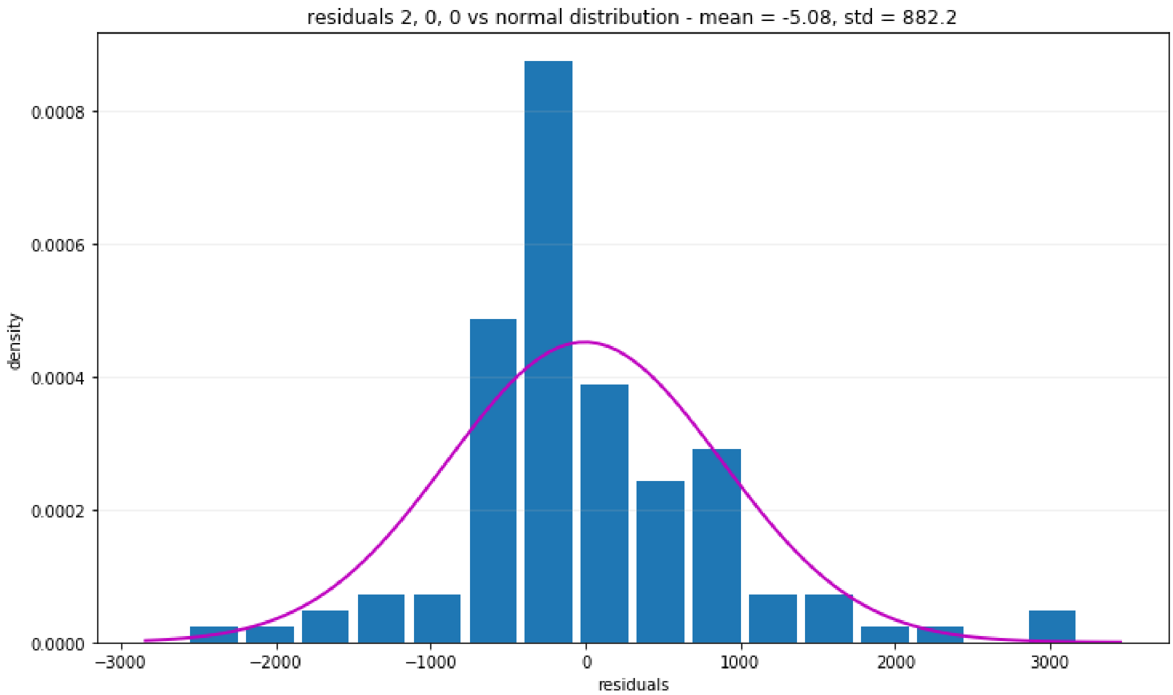
```
#ACF on our residual model
fig = plt.figure(figsize=(12,7))
ax1 = fig.add_subplot(211)
fig = plot_acf(results_AR.resid, lags=20, ax= ax1)
```




In [26]:

```
from scipy.stats import norm
```

```
plt.figure(figsize=(12,7))
plt.hist(results_AR.resid, bins = 'auto', density = True, rwidth = 0.85,
        label = 'residual')
mu, std = norm.fit(results_AR.resid)
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, mu, std)
plt.plot(x, p, 'm', linewidth = 2)
plt.grid(axis = 'y', alpha = 0.2)
plt.xlabel('residuals')
plt.ylabel('density')
plt.title('residuals 2, 0, 0 vs normal distribution - mean = '+str(round(mu, 2))+', std = '+str(round(std, 2)))
plt.show()
```

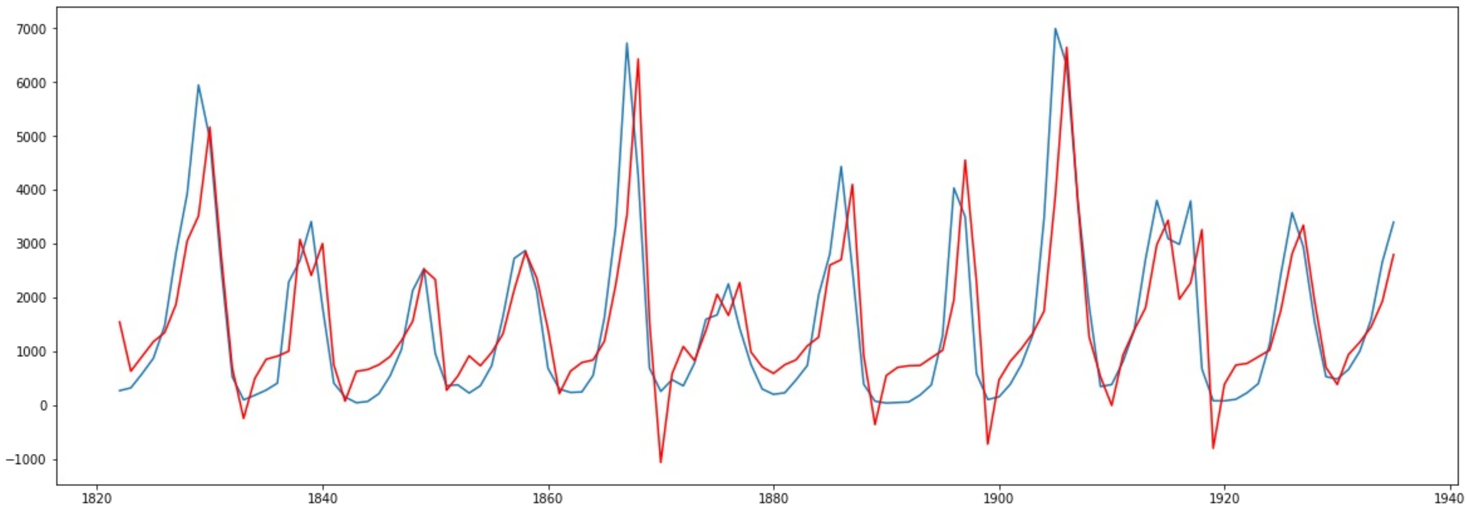



In [27]:

```
#order =(4, 0, 0) is producing the lowest value of AIC AR5 & AR3 both are higher value of AIC

model = ARIMA(dataseries, order=(4, 0, 0))
results_AR1 = model.fit()
plt.figure(figsize=(20,7))
plt.plot(dataseries)
plt.plot(results_AR.fittedvalues, color='red')
```

Out[27]:

[<matplotlib.lines.Line2D at 0x13ebbef25c0>]




In [28]:

```
results_AR1.summary()
```

Out[28]:

ARMA Model Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | y | **No. Observations:** | 114 |
| **Model:** | ARMA(4, 0) | **Log Likelihood** | -931.111 |
| **Method:** | css-mle | **S.D. of innovations** | 845.949 |
| **Date:** | Fri, 12 Jul 2019 | **AIC** | 1874.222 |
| **Time:** | 07:45:46 | **BIC** | 1890.639 |
| **Sample:** | 12-31-1821 | **HQIC** | 1880.885 |
| | - 12-31-1934 | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **const** | 1547.4367 | 136.851 | 11.307 | 0.000 | 1279.214 | 1815.659 |
| **ar.L1.y** | 1.1246 | 0.090 | 12.450 | 0.000 | 0.948 | 1.302 |
| **ar.L2.y** | -0.7174 | 0.137 | -5.250 | 0.000 | -0.985 | -0.450 |
| **ar.L3.y** | 0.2634 | 0.136 | 1.935 | 0.056 | -0.003 | 0.530 |
| **ar.L4.y** | -0.2543 | 0.090 | -2.837 | 0.005 | -0.430 | -0.079 |

Roots

| | Real | Imaginary | Modulus | Frequency |
|---|---|---|---|---|
| **AR.1** | 0.9198 | -0.6880j | 1.1486 | -0.1022 |
| **AR.2** | 0.9198 | +0.6880j | 1.1486 | 0.1022 |
| **AR.3** | -0.4020 | -1.6789j | 1.7264 | -0.2874 |
| **AR.4** | -0.4020 | +1.6789j | 1.7264 | 0.2874 |

In [29]:

```
results_AR1.resid.tail()
```

Out[29]:

```
1930-12-31    -65.572508
1931-12-31    -48.257955
1932-12-31     43.827806
1933-12-31    631.973963
1934-12-31    550.263041
Freq: A-DEC, dtype: float64
```

In [30]:

```
results_AR1.fittedvalues.tail()
```

Out[30]:

```
1930-12-31     727.572508
1931-12-31    1048.257955
1932-12-31    1546.172194
1933-12-31    2025.026037
1934-12-31    2845.736959
Freq: A-DEC, dtype: float64
```

In [31]:

```
dataseries.tail()
```

Out[31]:

```
1930-12-31     662
1931-12-31    1000
1932-12-31    1590
1933-12-31    2657
1934-12-31    3396
Freq: A-DEC, dtype: int64
```

In [32]:

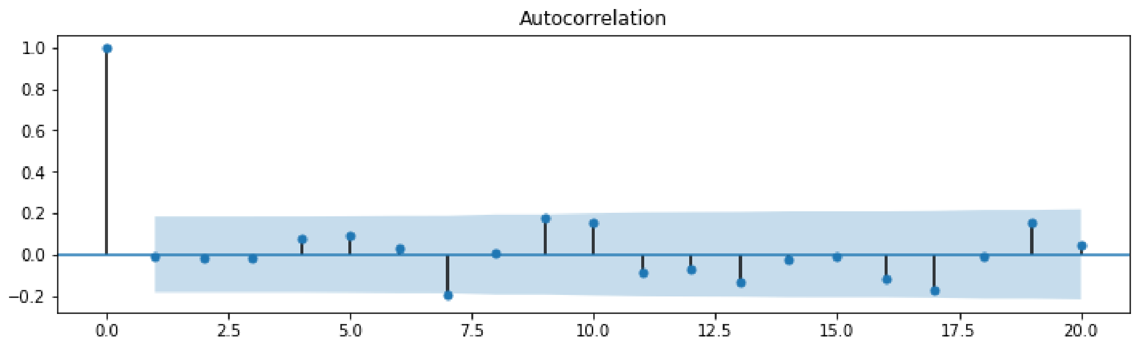
```
np.mean(results_AR1.resid)
```

Out[32]:

-9.065780174278016

In [33]:

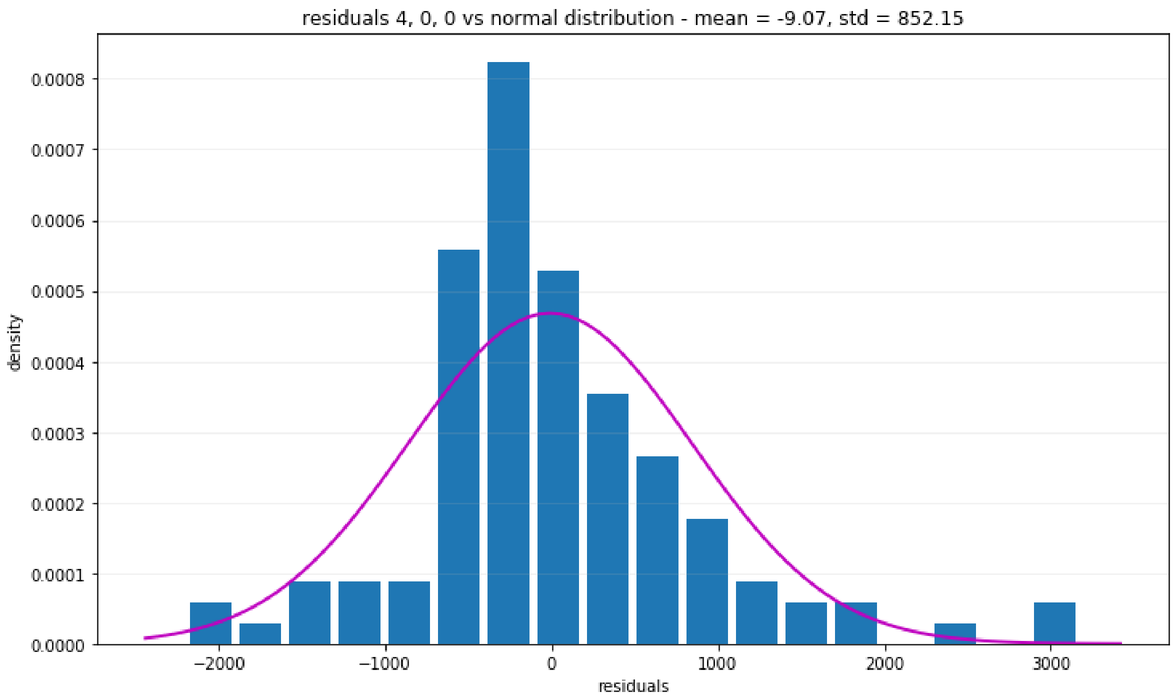
```
fig = plt.figure(figsize=(12,7))
ax1 = fig.add_subplot(211)
fig = plot_acf(results_AR1.resid, lags=20, ax= ax1)
```




In [35]:

```
#mu is mean,  pdf --> probabity densitty function
#line space is used for to equally space the bars in graph
from scipy.stats import norm

plt.figure(figsize=(12,7))
plt.hist(results_AR1.resid, bins = 'auto', density = True, rwidth = 0.85,
         label = 'residual')
mu, std = norm.fit(results_AR1.resid)
print(mu, std)
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, mu, std)
plt.plot(x, p, 'm', linewidth = 2)
plt.grid(axis = 'y', alpha = 0.2)
plt.xlabel('residuals')
plt.ylabel('density')
plt.title('residuals 4, 0, 0 vs normal distribution - mean = '+str(round(mu, 2))+', std = '+str(round(std, 2)))
plt.show()
```

-9.065780174278022 852.1504376527944



In [36]:

```
model202 = ARIMA(dataseries, order=(2, 0, 2))
results_AR2 = model202.fit()
```

In [38]:

```
Fcast400 = results_AR1.predict(start = '31/12/1935', end = '31/12/1945')
fcast202 = results_AR2.predict(start = '31/12/1935', end = '31/12/1945')
```

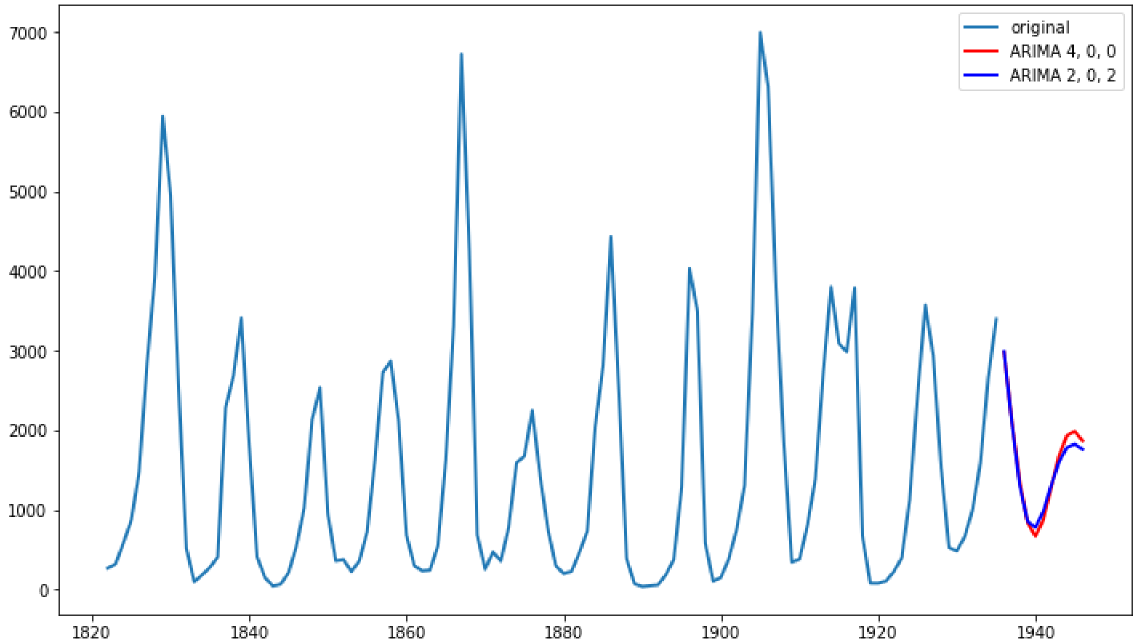
```
C:\Users\SACHIN K M\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:336: FutureWarning: Creating a DatetimeIndex by passing r
ange endpoints is deprecated.  Use `pandas.date_range` instead.
  freq=base_index.freq)
```

In [40]:

```
plt.figure(figsize=(12,7))
plt.plot(dataseries, linewidth=2, label ="original")
plt.plot(Fcast400, color='red', linewidth=2, label="ARIMA 4, 0, 0")
plt.plot(fcast202, color='blue', linewidth=2, label="ARIMA 2, 0, 2")
plt.legend()
```

Out[40]:

<matplotlib.legend.Legend at 0x13ebccce710>




In [ ]: