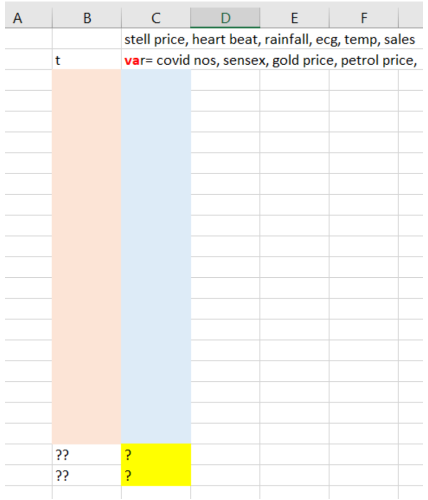
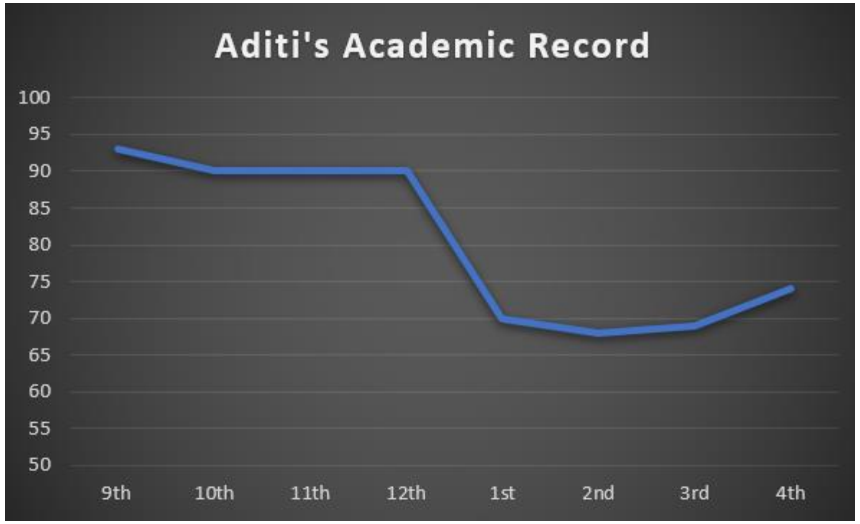
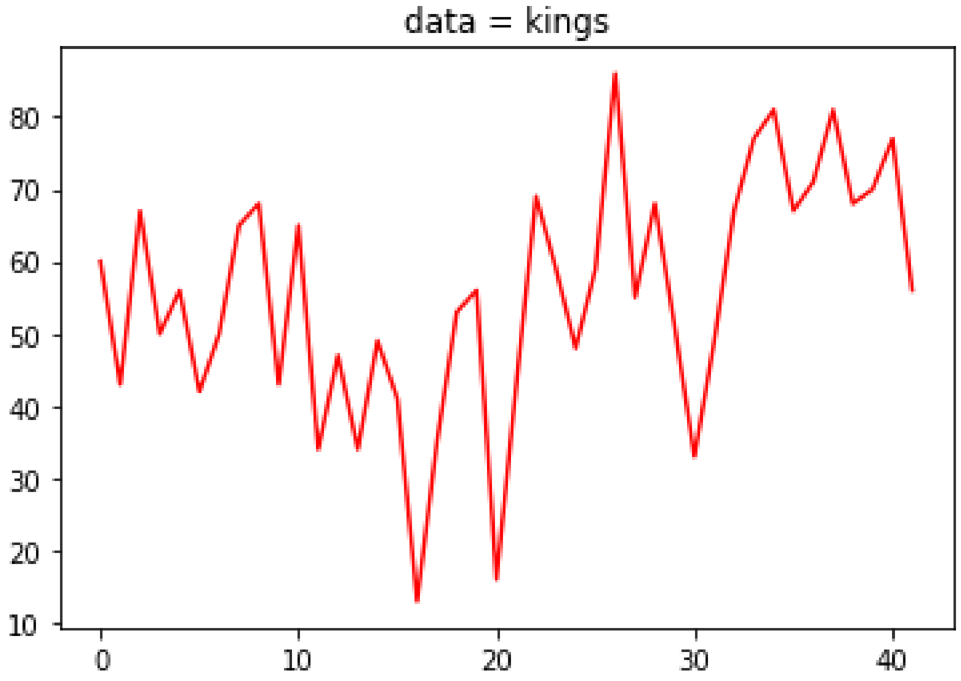
Time Series Concepts

Data sets: kings, rain, births, skirts, sovenir

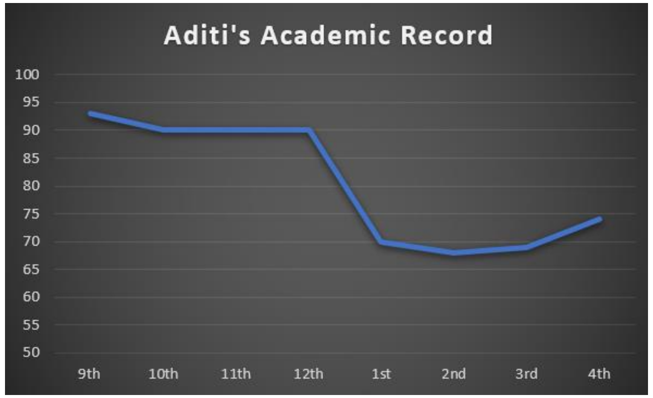


Data: kings

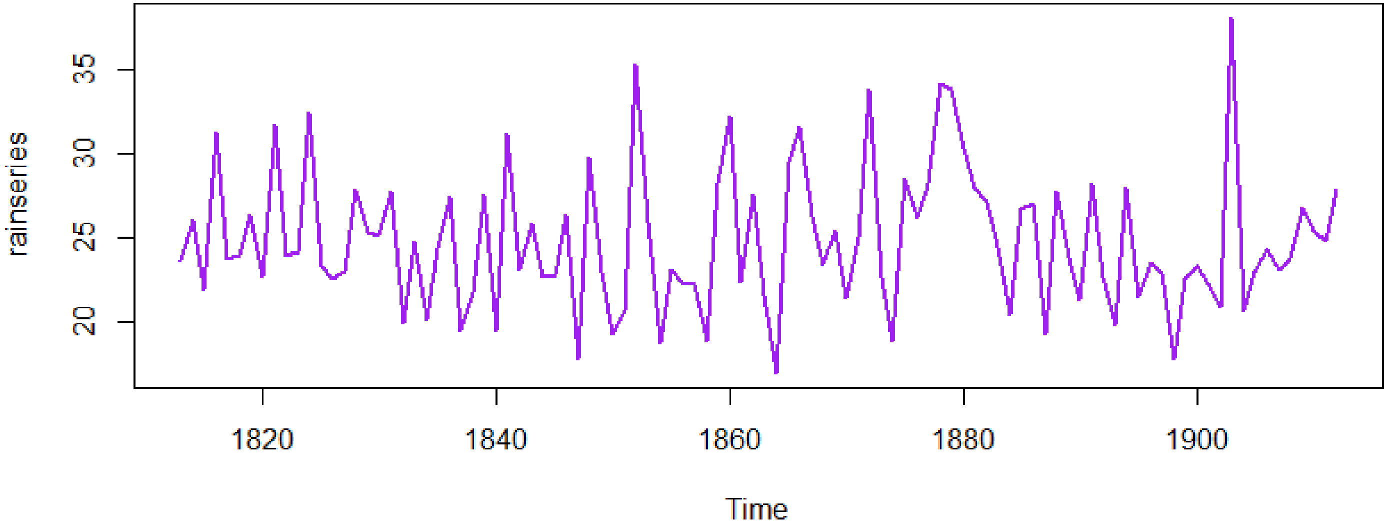

Aditi's Academic Record
100
95
90
85
80
75
70
65
60
55
50
9th
10th
11th
12th
1st
2nd
3rd
4th


data = kings
80
70
60
50
40
30
20
10
0
10
20
30
40

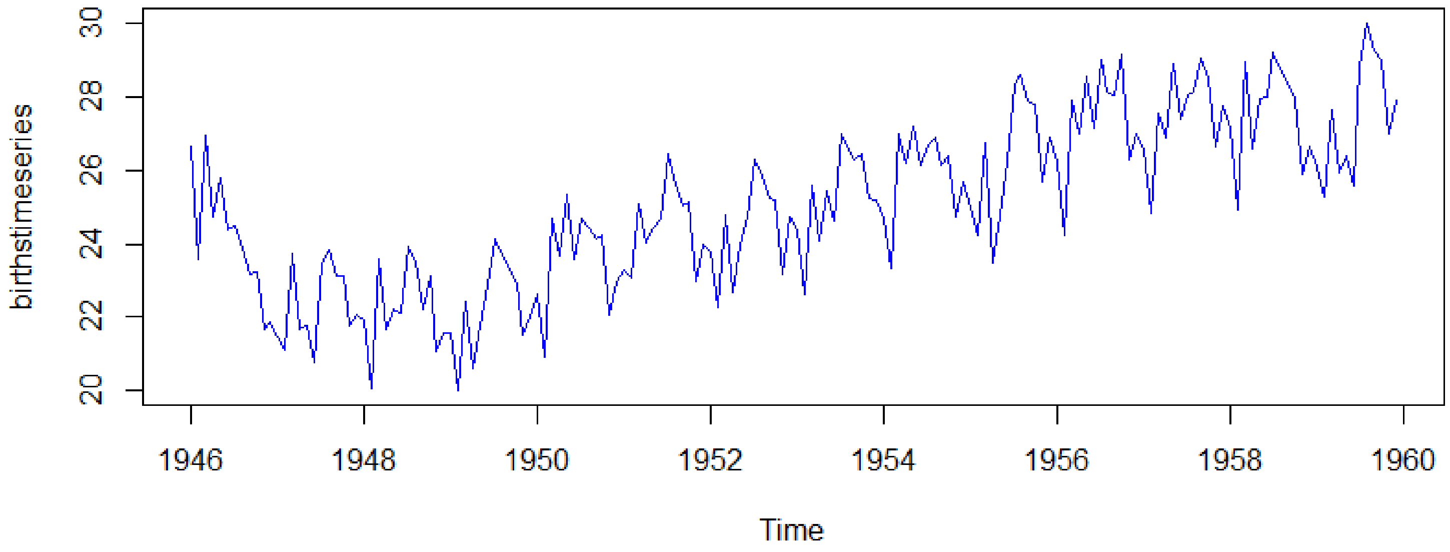
Data: rain

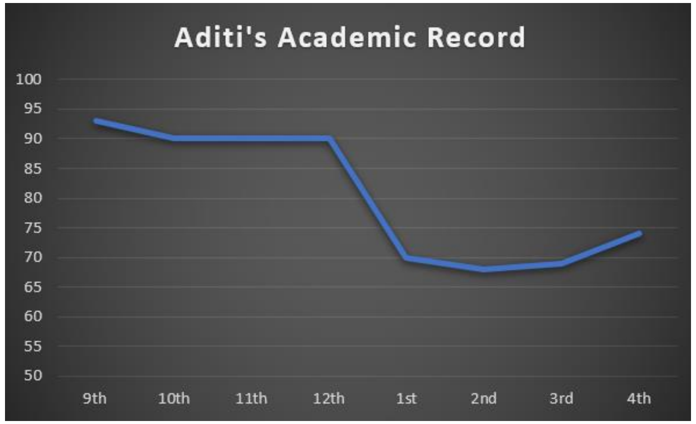


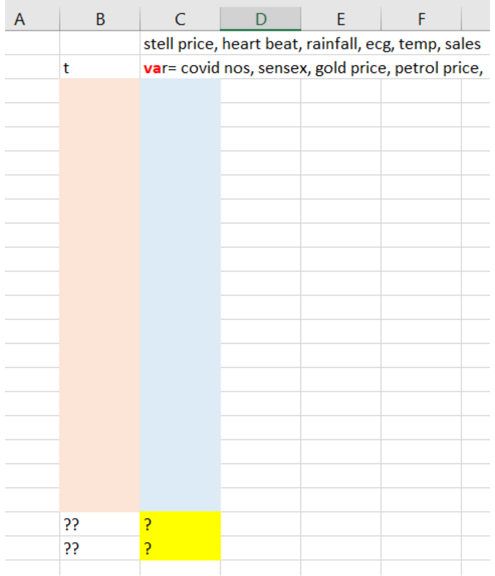
A	B	C	D	E	F
		stell price, heart beat, rainfall, ecg, temp, sales			
	t	var= covid nos, sensex, gold price, petrol price,			
	??	?			
	??	?			



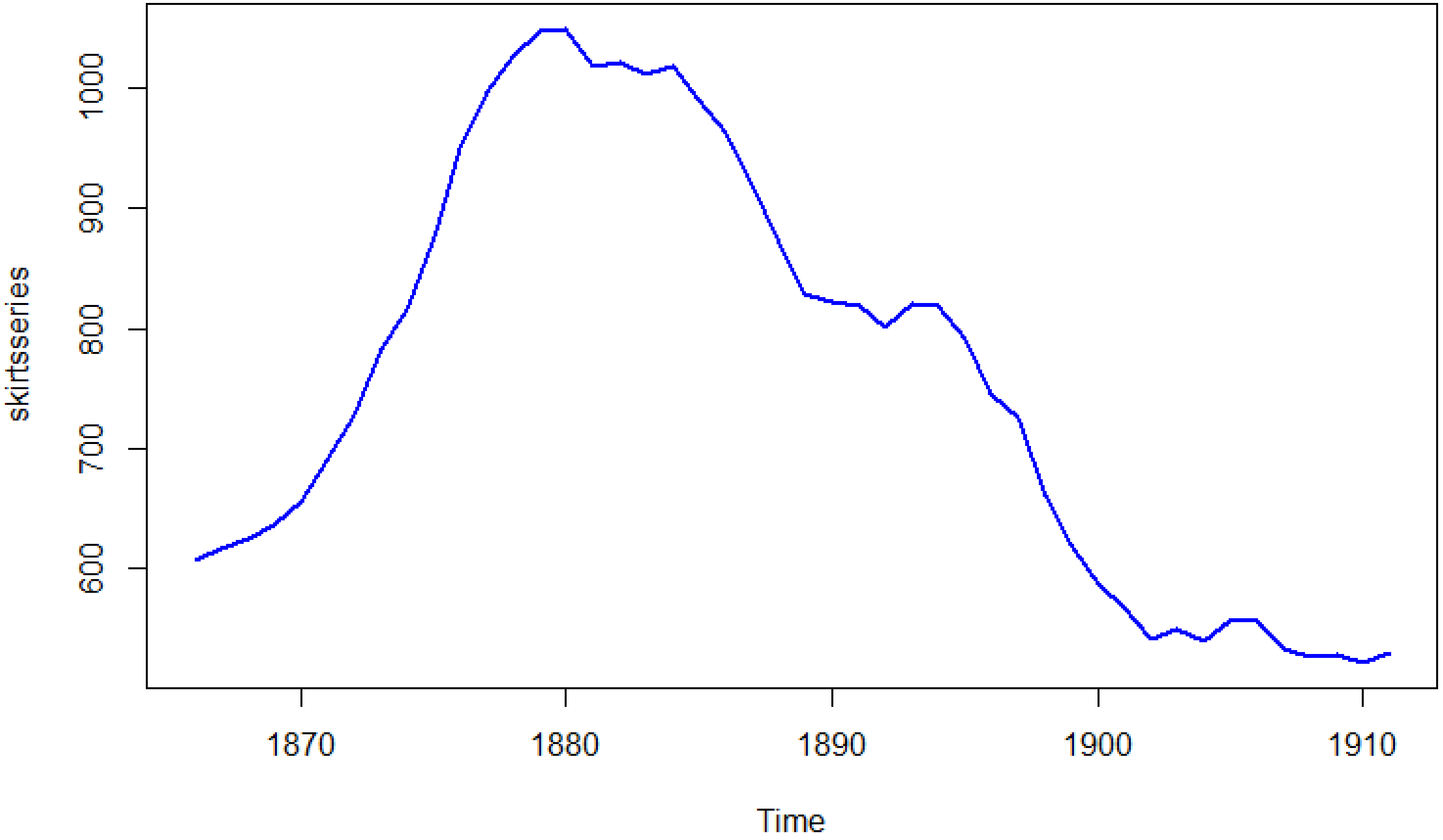
Data: births


birthstimeseries
20
22
24
26
28
30
1946
1948
1950
1952
1954
1956
1958
1960
Time

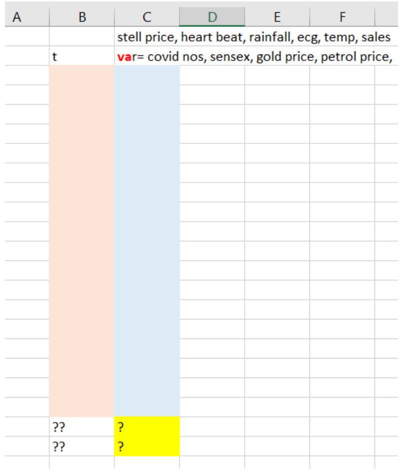

Aditi's Academic Record
100
95
90
85
80
75
70
65
60
55
50
9th
10th
11th
12th
1st
2nd
3rd
4th


A
B
C
D
E
F
stell price, heart beat, rainfall, ecg, temp, sales
t
var= covid nos, sensex, gold price, petrol price,
??
?
??
?

Data: skirts

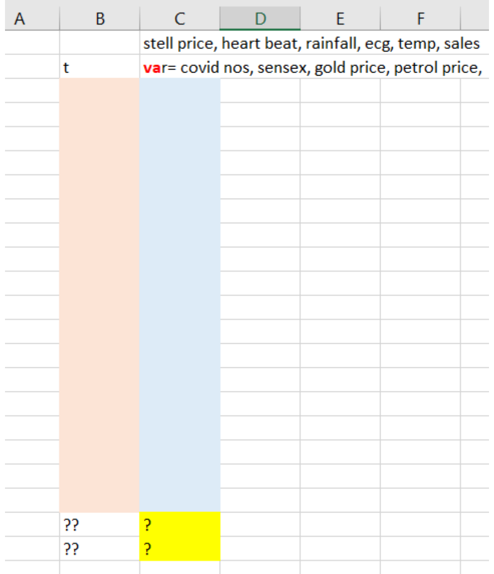


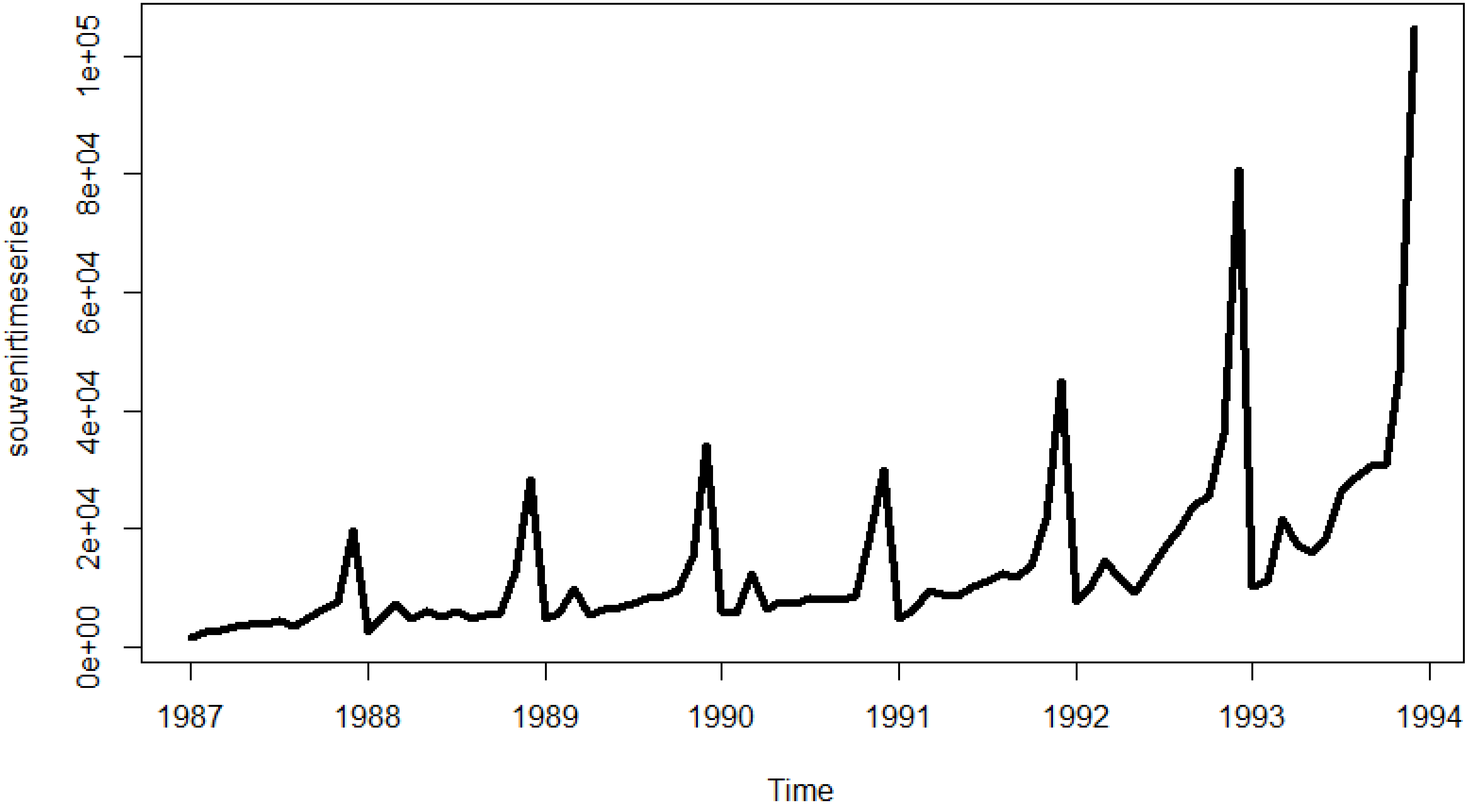
skirtsseries
600
700
800
900
1000
1870
1880
1890
1900
1910
Time



A
B
C
D
E
F
stell price, heart beat, rainfall, ecg, temp, sales
t
var= covid nos, sensex, gold price, petrol price,
??
?
??
?

Data: sovenir





Five data sets have been provided to you for applying the learnt concepts and see the difference in the results of different approaches!

Kings

Libraries

```
# Jesus is King of Kings!
import os
os.chdir('C:\\Users\\Dr Vinod\\Desktop\\WD_python')
import pandas as pd
pd.set_option('display.max_columns', None)
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import statsmodels.formula.api as smf
from statsmodels.tsa.stattools import adfuller
%matplotlib inline
from math import sqrt
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
```

Import Data

```
kings = pd.read_csv('kings.csv')
kings.info() # 42
kings = kings.drop('obsno', axis=1)
kings.info()
kings.index #RangeIndex(start=0, stop=42, step=1)
```

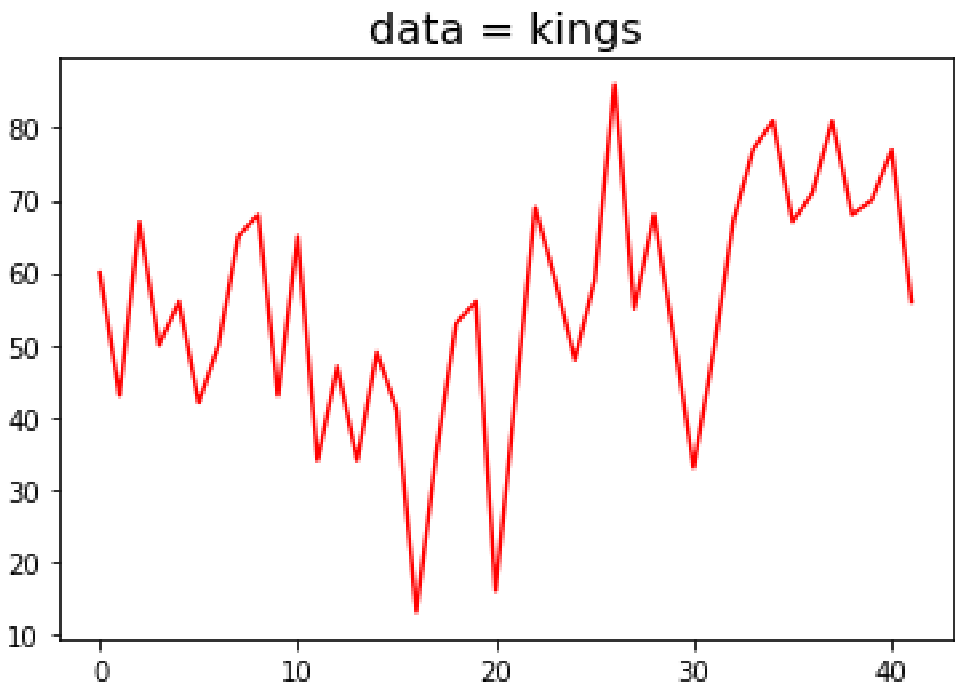
```
In [9]: kings.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42 entries, 0 to 41
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   age     42 non-null     int64
dtypes: int64(1)
memory usage: 464.0 bytes
```

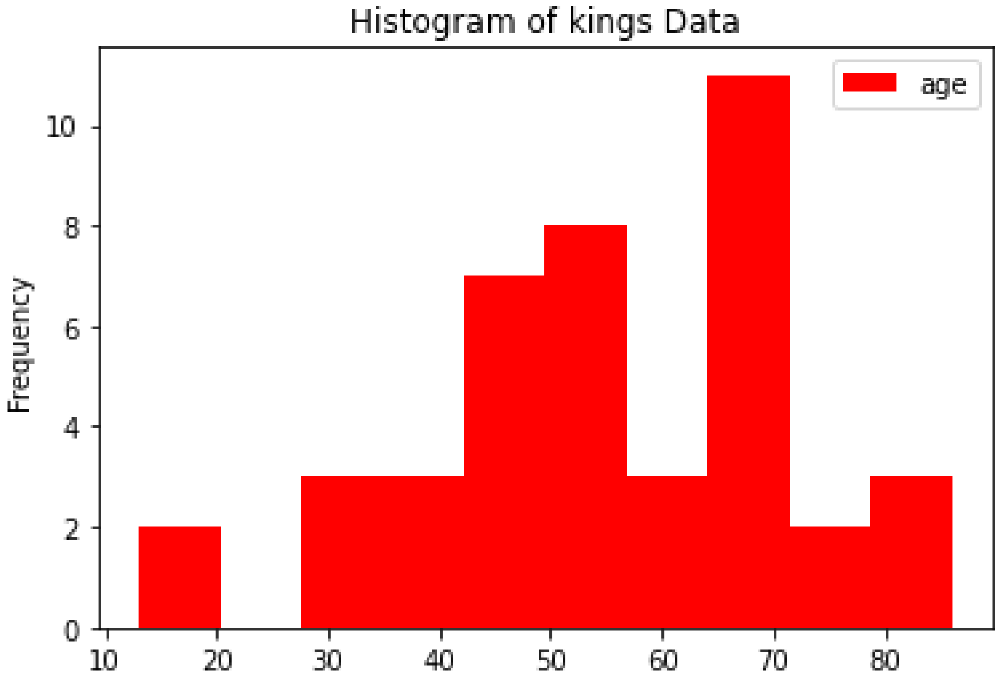
```
In [7]: kings.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42 entries, 0 to 41
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   obsno   42 non-null     int64
 1   age     42 non-null     int64
dtypes: int64(2)
memory usage: 800.0 bytes
```

```
In [10]: kings.index #RangeIndex(start=0, stop=42, step=1)
Out[10]: RangeIndex(start=0, stop=42, step=1)
```

Plots

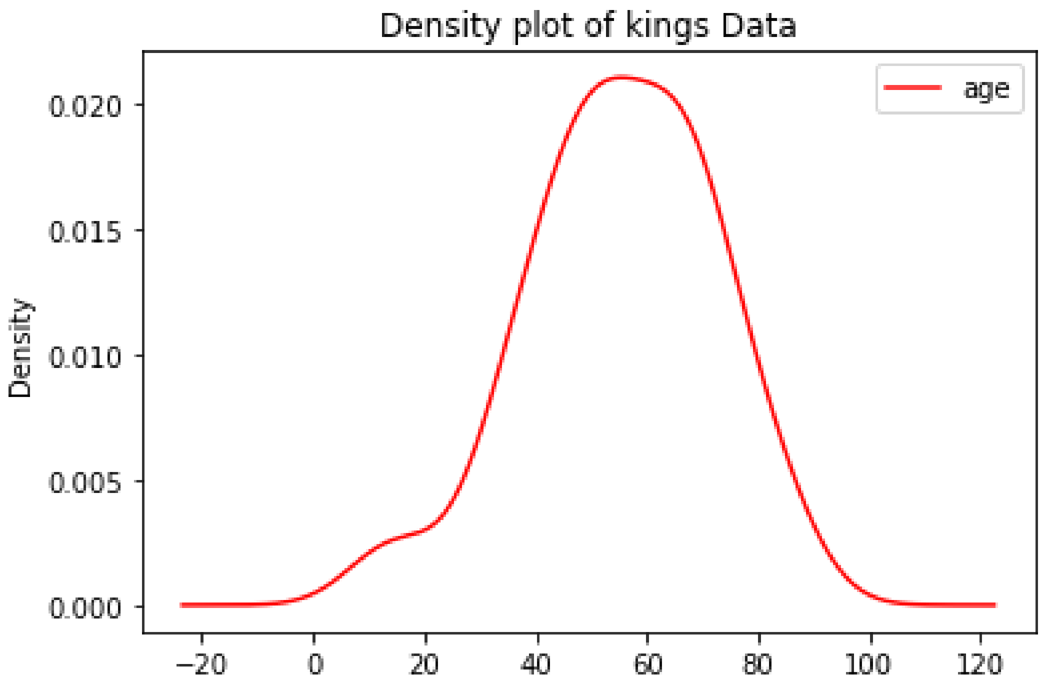
```
#lineplot
plt.plot(kings, 'r')
plt.title('data = kings', fontsize=16)

#Histogram
kings.plot(kind='hist', facecolor = 'r')
plt.title('Histogram of kings Data')
```

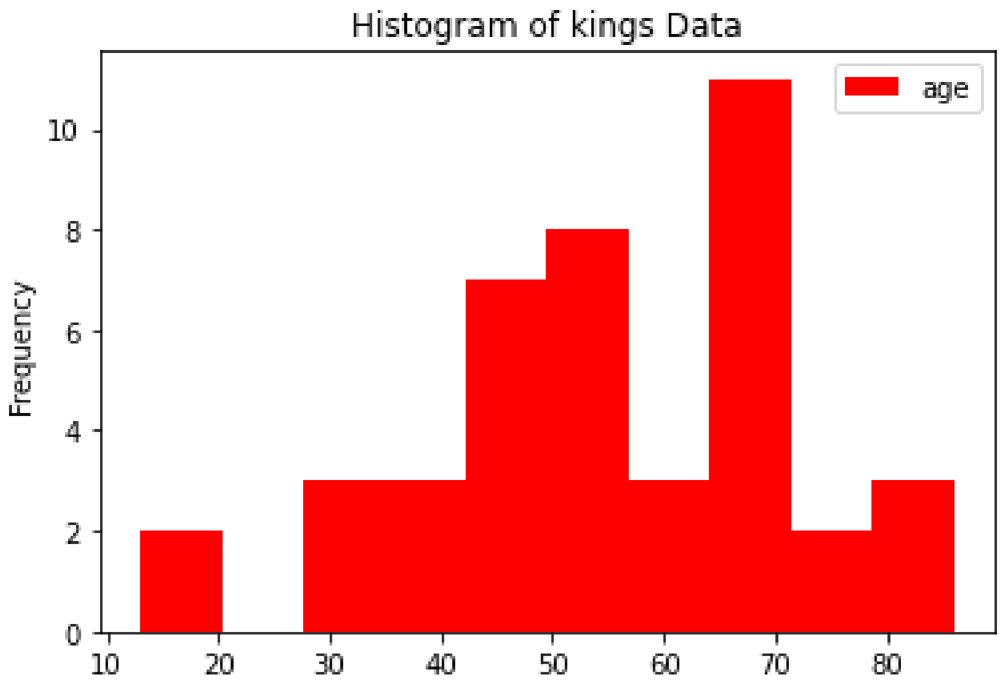




Plots

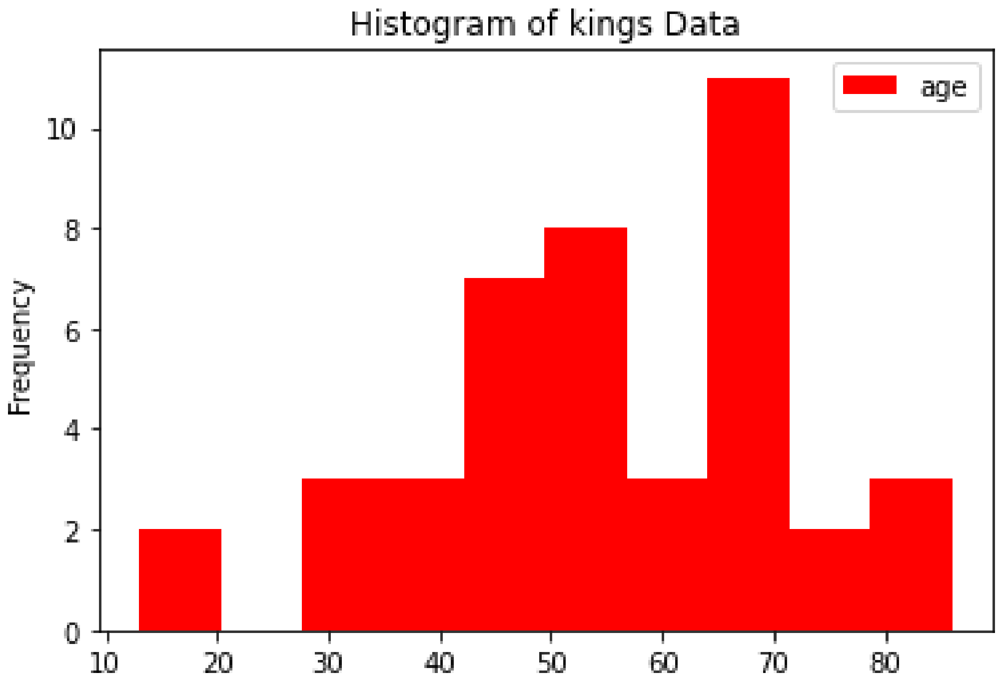
```
#Density plot
kings.plot(kind='kde', color = 'r')
plt.title('Density plot of kings Data')
```

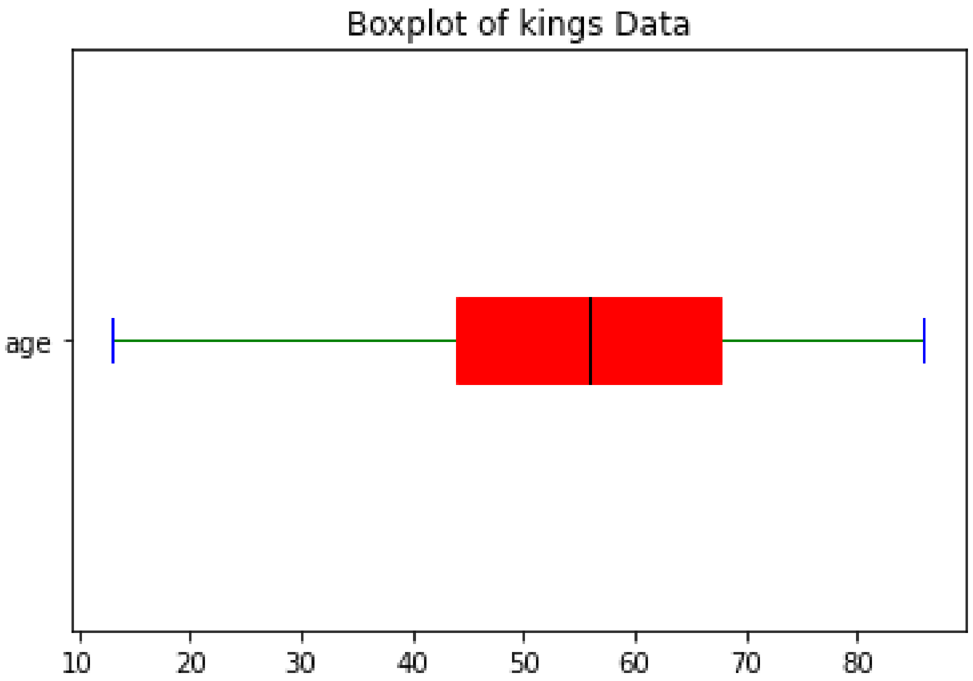




Box Plot

```
#Boxplot
props2 = dict(boxes = 'red', whiskers ='green', medians = 'black', caps = 'blue')
kings.plot.box(color = props2 , patch_artist = True, vert = False)
plt.title('Boxplot of kings Data')
```

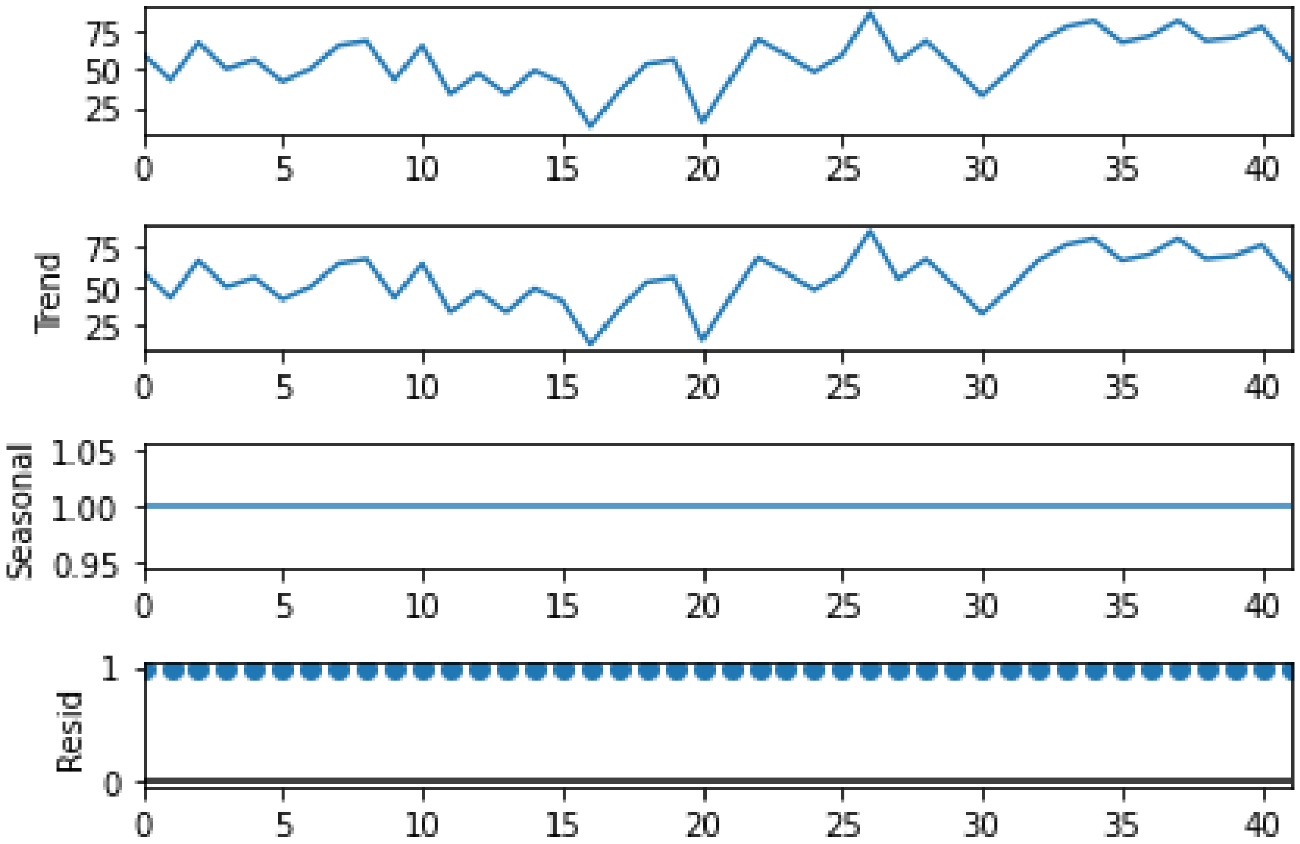




Decompose

```
#Decompose with multiplicative
from statsmodels.tsa.seasonal import seasonal_decompose
kings_decomp_m = seasonal_decompose(kings, period=1, model='mul')
#Peroid is specified being the data is not having date as index
```

```
kings_decomp_m.plot() #No Trend & no seasonality; note 1
```

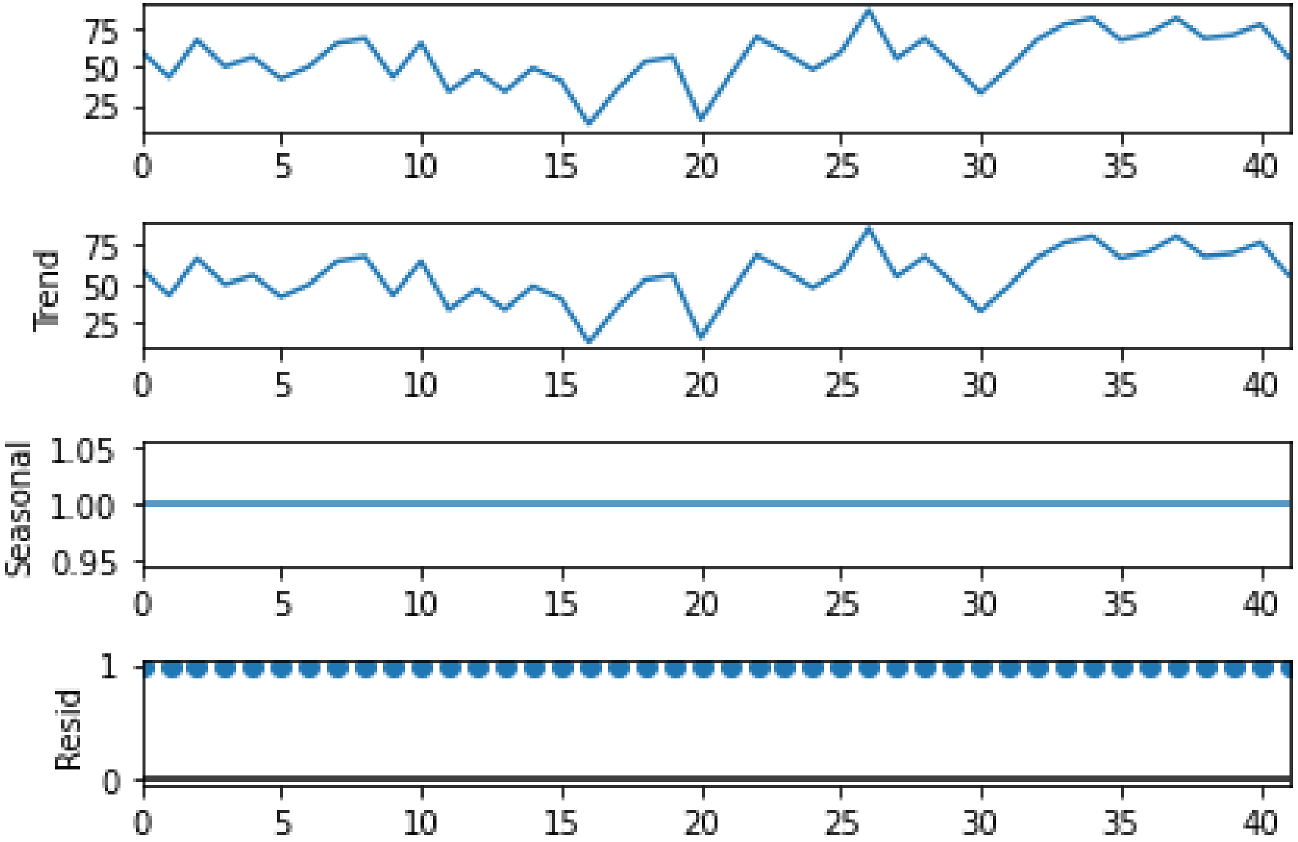


Ts = t*s*r
No seasonality
S = 1

Ts = t+s+r
No seasonality
S = 0

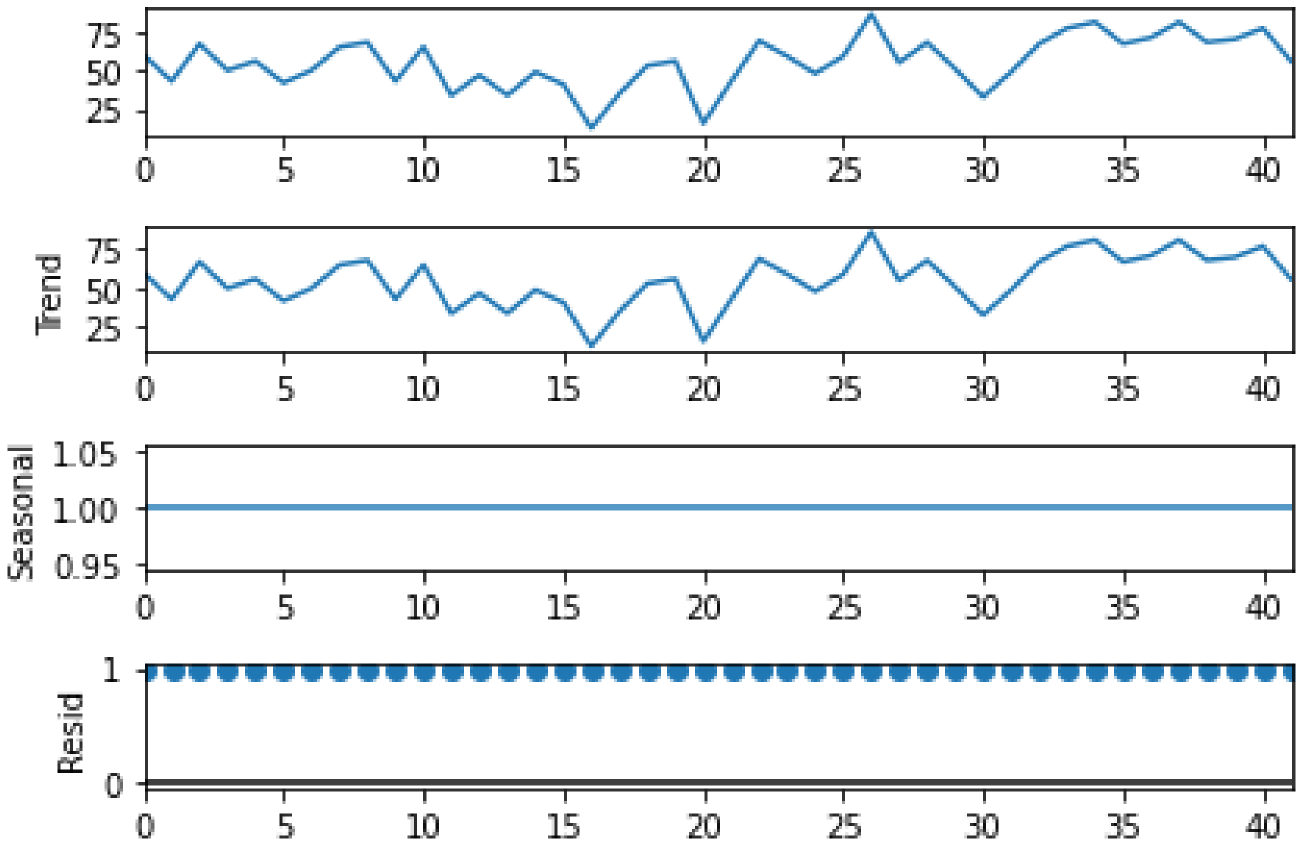
Observed

```
In [36]: kings_decomp_m.observed
Out[36]:
0       60.0
1       43.0
2       67.0
3       50.0
4       56.0
5       42.0
6       50.0
7       65.0
8       68.0
9       43.0
10      65.0
11      34.0
12      47.0
13      34.0
14      49.0
15      41.0
16      13.0
17      35.0
18      53.0
19      56.0
20      16.0
21      43.0
22      69.0
23      59.0
24      48.0
25      59.0
26      86.0
27      55.0
28      68.0
29      51.0
30      33.0
31      49.0
32      67.0
33      77.0
34      81.0
35      67.0
36      71.0
37      81.0
38      68.0
39      70.0
40      77.0
41      56.0
dtype: float64
```



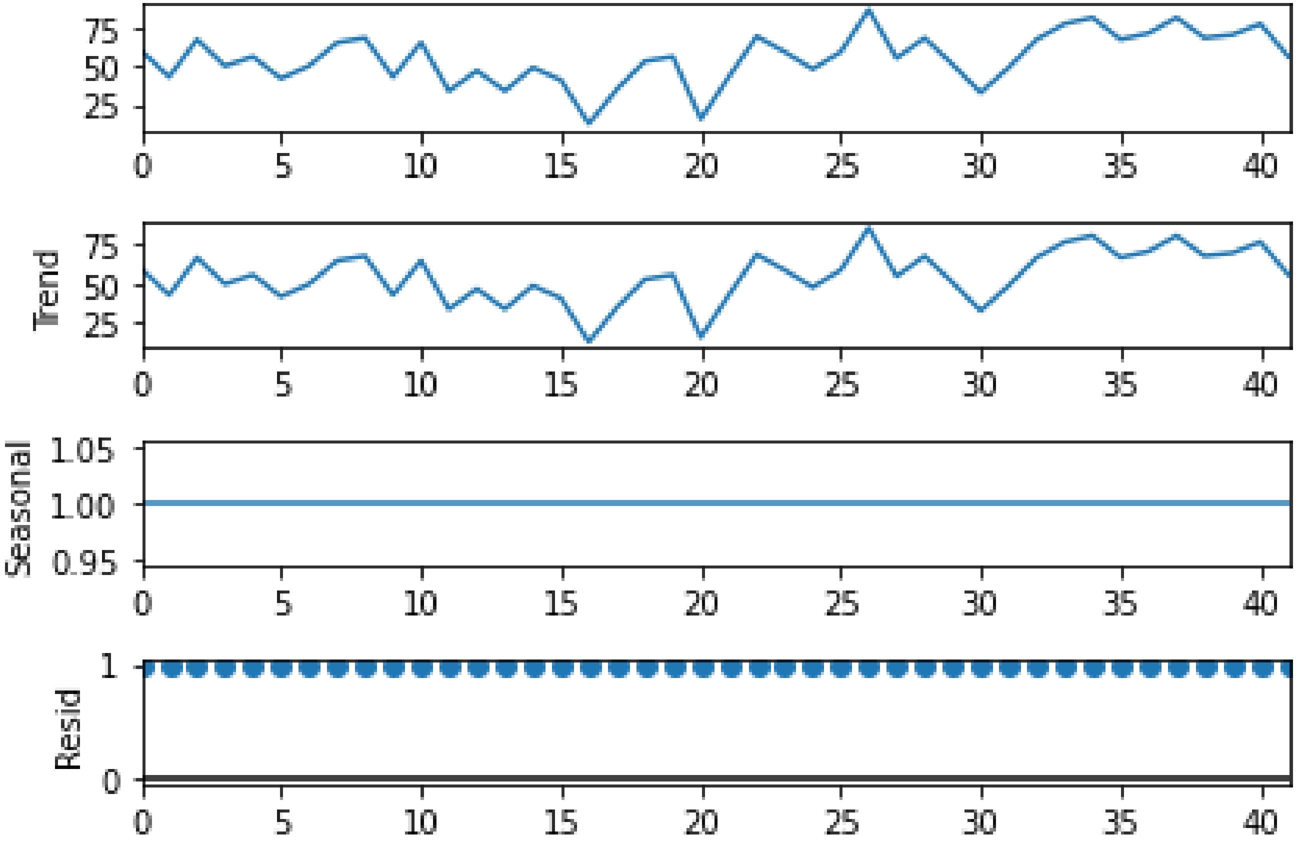
Trend

```
In [37]: kings_decomp_m.trend
Out[37]:
0      60.0
1      43.0
2      67.0
3      50.0
4      56.0
5      42.0
6      50.0
7      65.0
8      68.0
9      43.0
10     65.0
11     34.0
12     47.0
13     34.0
14     49.0
15     41.0
16     13.0
17     35.0
18     53.0
19     56.0
20     16.0
21     43.0
22     69.0
23     59.0
24     48.0
25     59.0
26     86.0
27     55.0
28     68.0
29     51.0
30     33.0
31     49.0
32     67.0
33     77.0
34     81.0
35     67.0
36     71.0
37     81.0
38     68.0
39     70.0
40     77.0
41     56.0
Name: trend, dtype: float64
```



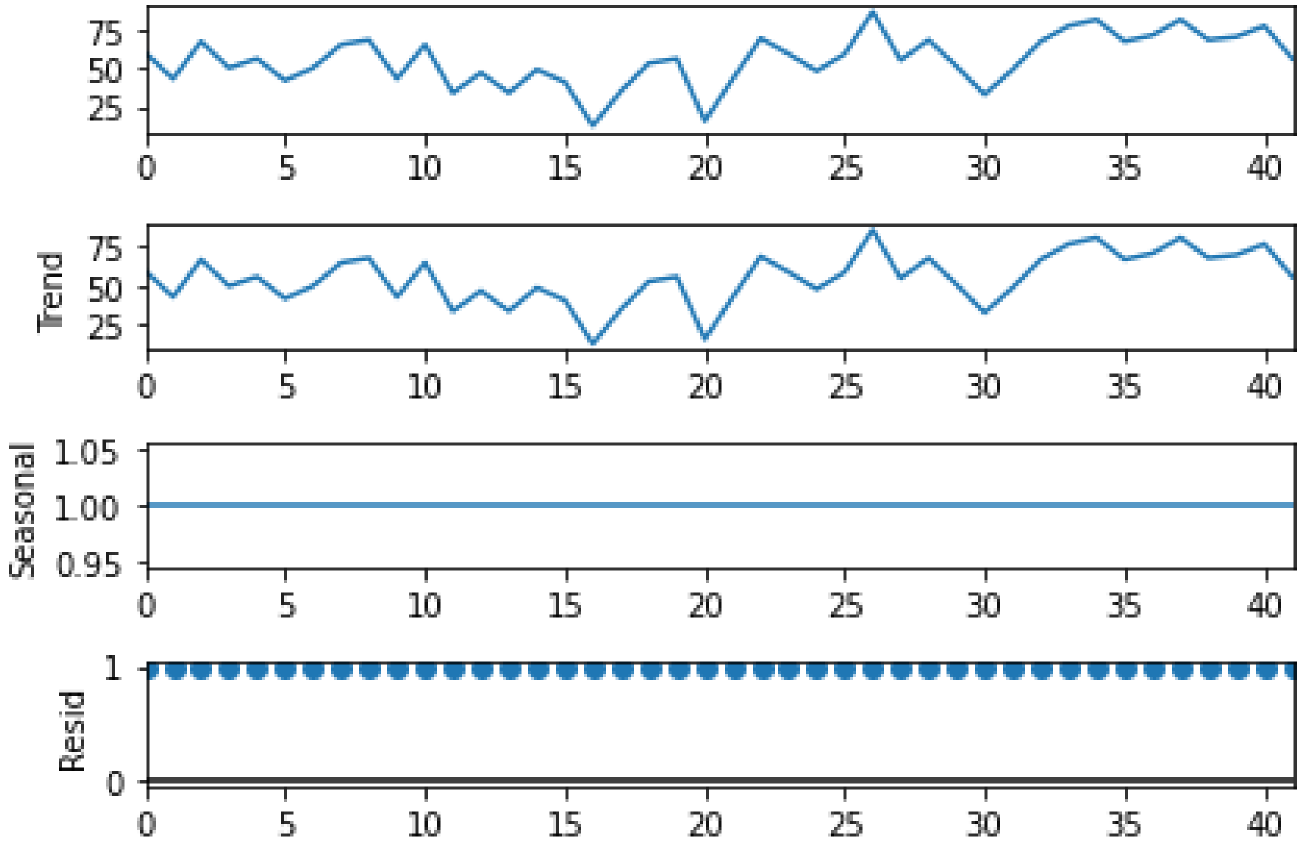
Seasonal

```
In [38]: kings_decomp_m.seasonal
Out[38]:
0      1.0
1      1.0
2      1.0
3      1.0
4      1.0
5      1.0
6      1.0
7      1.0
8      1.0
9      1.0
10     1.0
11     1.0
12     1.0
13     1.0
14     1.0
15     1.0
16     1.0
17     1.0
18     1.0
19     1.0
20     1.0
21     1.0
22     1.0
23     1.0
24     1.0
25     1.0
26     1.0
27     1.0
28     1.0
29     1.0
30     1.0
31     1.0
32     1.0
33     1.0
34     1.0
35     1.0
36     1.0
37     1.0
38     1.0
39     1.0
40     1.0
41     1.0
Name: seasonal, dtype: float64
```



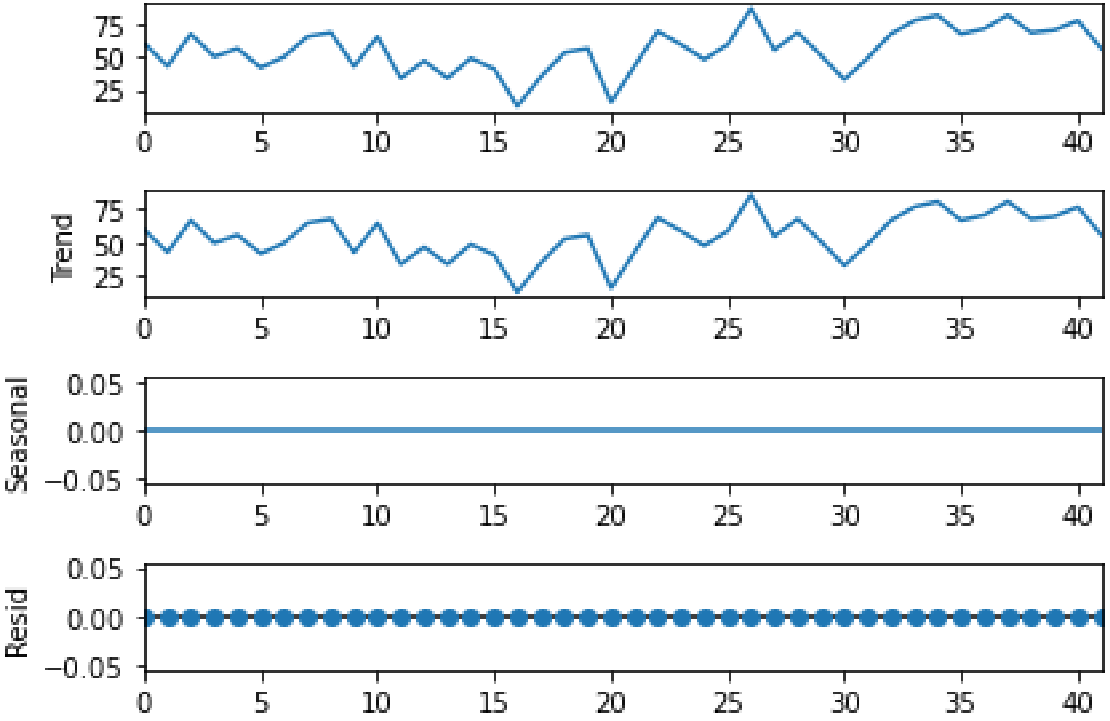
Residuals

```
In [39]: kings_decomp_m.resid
Out[39]:
0       1.0
1       1.0
2       1.0
3       1.0
4       1.0
5       1.0
6       1.0
7       1.0
8       1.0
9       1.0
10      1.0
11      1.0
12      1.0
13      1.0
14      1.0
15      1.0
16      1.0
17      1.0
18      1.0
19      1.0
20      1.0
21      1.0
22      1.0
23      1.0
24      1.0
25      1.0
26      1.0
27      1.0
28      1.0
29      1.0
30      1.0
31      1.0
32      1.0
33      1.0
34      1.0
35      1.0
36      1.0
37      1.0
38      1.0
39      1.0
40      1.0
41      1.0
Name: resid, dtype: float64
```



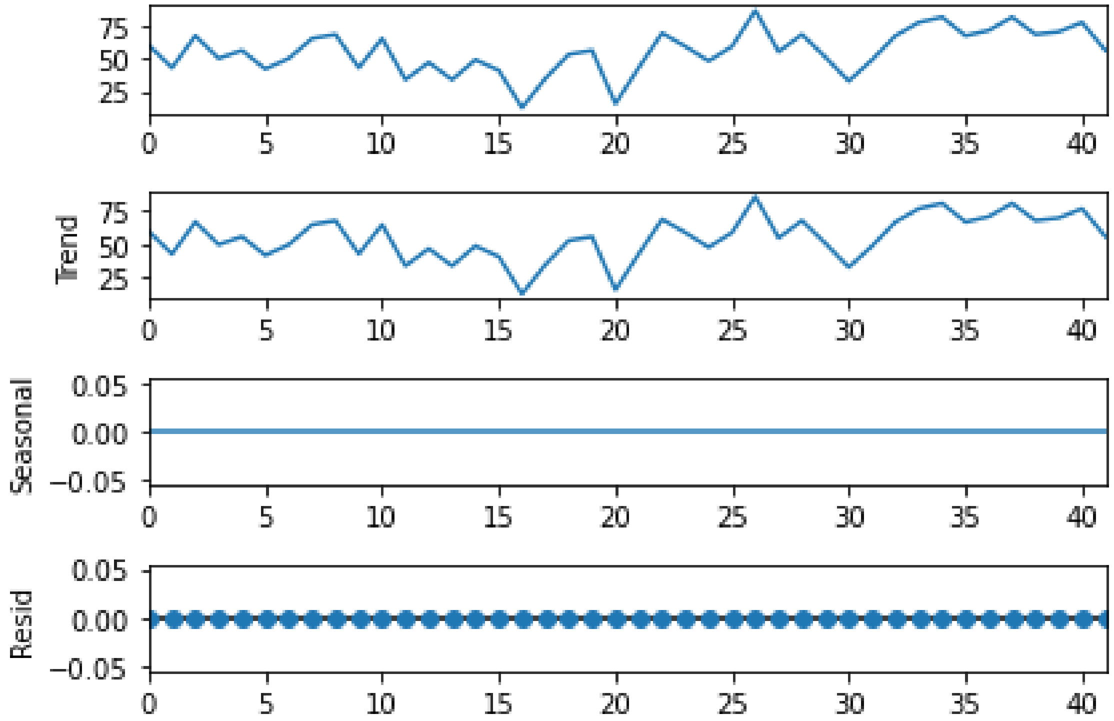
```
#Decompose with Additive
from statsmodels.tsa.seasonal import seasonal_decompose
kings_decomp_add = seasonal_decompose(kings, period=1, model='add')

kings_decomp_add.plot() #No Trend & no seasonality
```

Decompose _additive



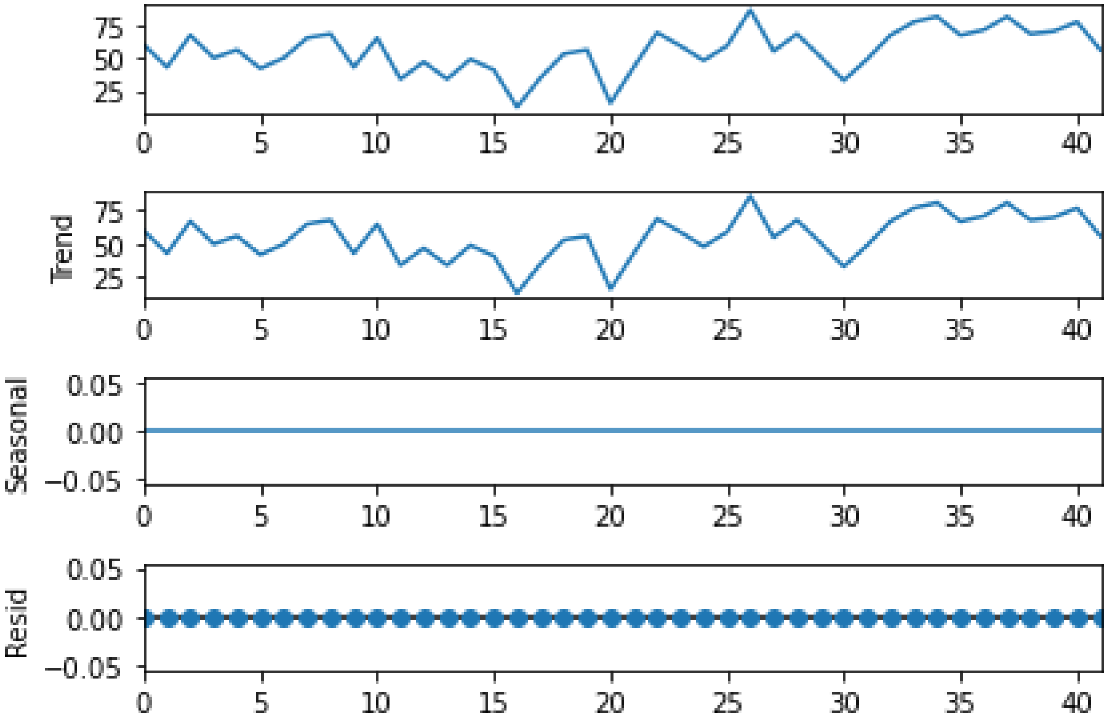
Decompose_additive Observed

```
In [43]: kings_decomp_add.observed
Out[43]:
0       60.0
1       43.0
2       67.0
3       50.0
4       56.0
5       42.0
6       50.0
7       65.0
8       68.0
9       43.0
10      65.0
11      34.0
12      47.0
13      34.0
14      49.0
15      41.0
16      13.0
17      35.0
18      53.0
19      56.0
20      16.0
21      43.0
22      69.0
23      59.0
24      48.0
25      59.0
26      86.0
27      55.0
28      68.0
29      51.0
30      33.0
31      49.0
32      67.0
33      77.0
34      81.0
35      67.0
36      71.0
37      81.0
38      68.0
39      70.0
40      77.0
41      56.0
dtype: float64
```



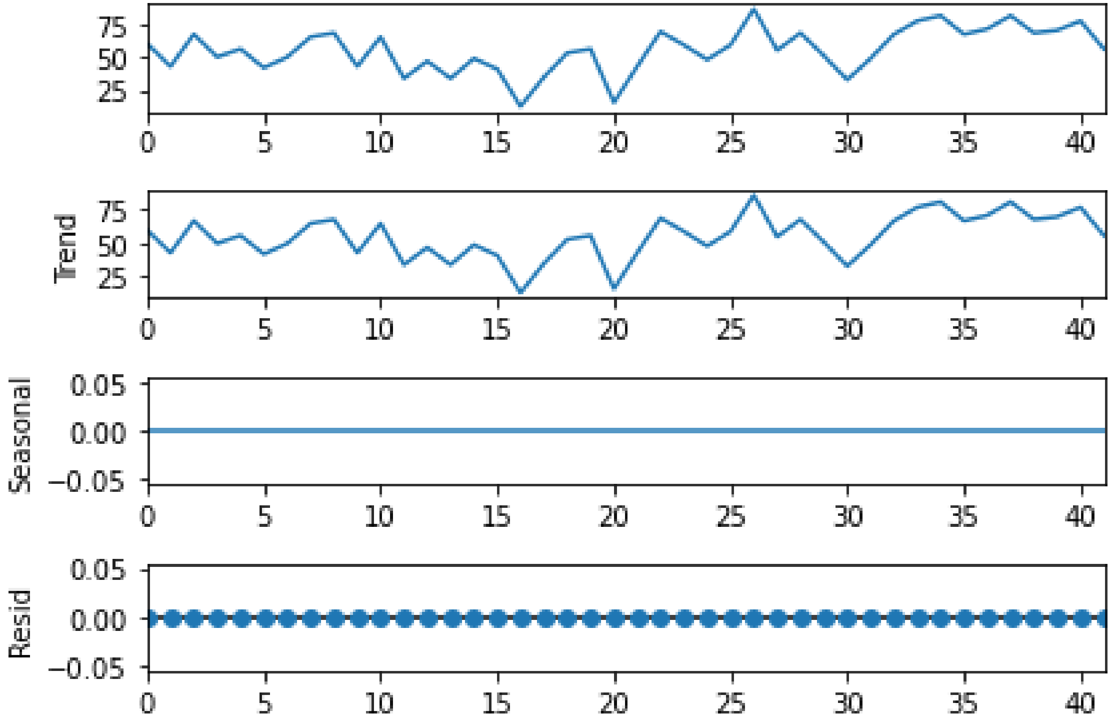
Decompose_additive Trend

```
In [44]: kings_decomp_add.trend
Out[44]:
0      60.0
1      43.0
2      67.0
3      50.0
4      56.0
5      42.0
6      50.0
7      65.0
8      68.0
9      43.0
10     65.0
11     34.0
12     47.0
13     34.0
14     49.0
15     41.0
16     13.0
17     35.0
18     53.0
19     56.0
20     16.0
21     43.0
22     69.0
23     59.0
24     48.0
25     59.0
26     86.0
27     55.0
28     68.0
29     51.0
30     33.0
31     49.0
32     67.0
33     77.0
34     81.0
35     67.0
36     71.0
37     81.0
38     68.0
39     70.0
40     77.0
41     56.0
Name: trend, dtype: float64
```



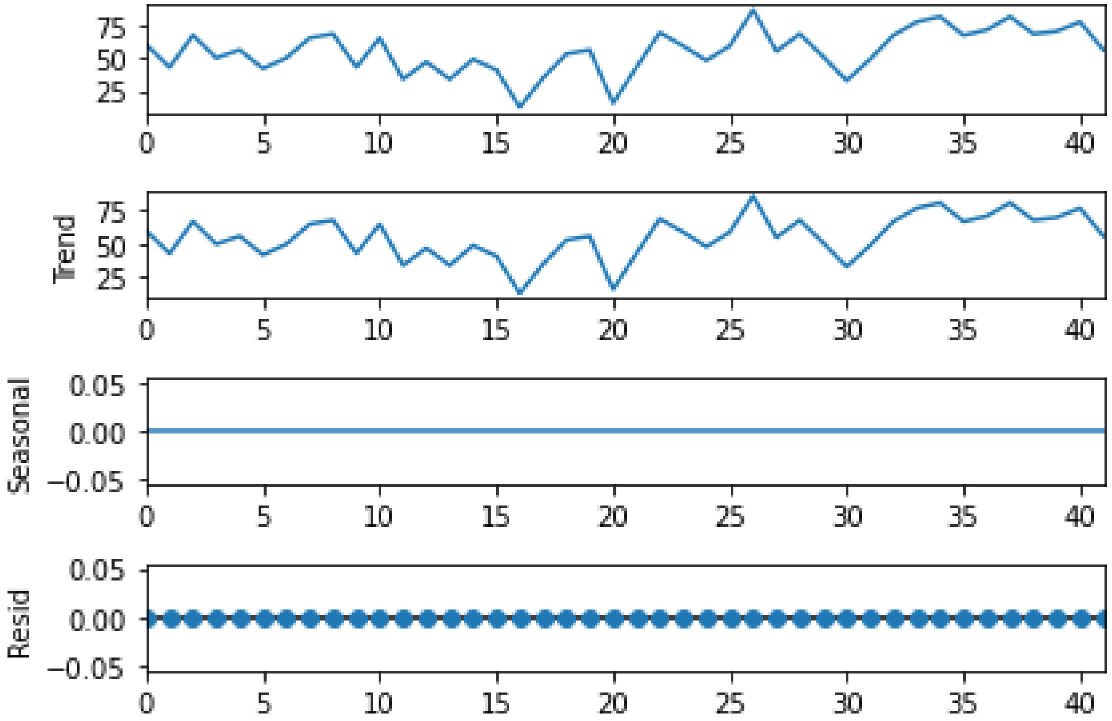
Decompose_additive Seasonal

```
In [45]: kings_decomp_add.seasonal
Out[45]:
0      0.0
1      0.0
2      0.0
3      0.0
4      0.0
5      0.0
6      0.0
7      0.0
8      0.0
9      0.0
10     0.0
11     0.0
12     0.0
13     0.0
14     0.0
15     0.0
16     0.0
17     0.0
18     0.0
19     0.0
20     0.0
21     0.0
22     0.0
23     0.0
24     0.0
25     0.0
26     0.0
27     0.0
28     0.0
29     0.0
30     0.0
31     0.0
32     0.0
33     0.0
34     0.0
35     0.0
36     0.0
37     0.0
38     0.0
39     0.0
40     0.0
41     0.0
Name: seasonal, dtype: float64
```



Decompose_additive Residuals

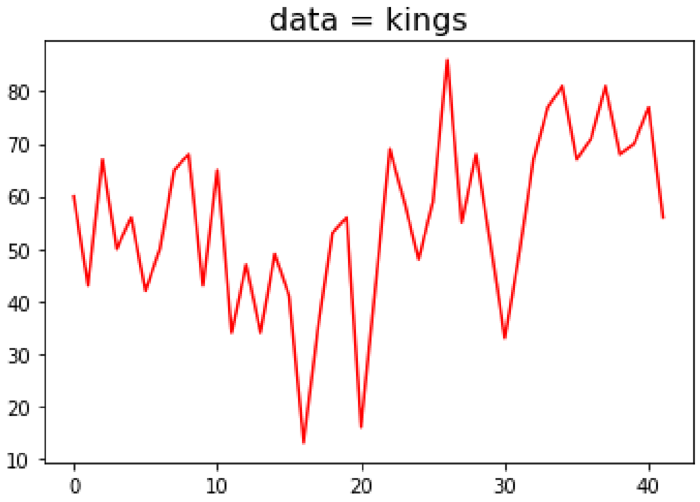
```
In [46]: kings_decomp_add.resid
Out[46]:
0       0.0
1       0.0
2       0.0
3       0.0
4       0.0
5       0.0
6       0.0
7       0.0
8       0.0
9       0.0
10      0.0
11      0.0
12      0.0
13      0.0
14      0.0
15      0.0
16      0.0
17      0.0
18      0.0
19      0.0
20      0.0
21      0.0
22      0.0
23      0.0
24      0.0
25      0.0
26      0.0
27      0.0
28      0.0
29      0.0
30      0.0
31      0.0
32      0.0
33      0.0
34      0.0
35      0.0
36      0.0
37      0.0
38      0.0
39      0.0
40      0.0
41      0.0
Name: resid, dtype: float64
```



Stationarity Test

```
#Test for stationarity
from statsmodels.tsa.stattools import adfuller
kings_adf = adfuller(kings)
kings_adf
```

```
In [49]: kings_adf
Out[49]:
(-4.090229860104913,
 0.0010051728027032974,
 0,
 41,
 {'1%': -3.60098336718852,
  '5%': -2.9351348158036012,
  '10%': -2.6059629803688282},
 263.5882454953018)
```

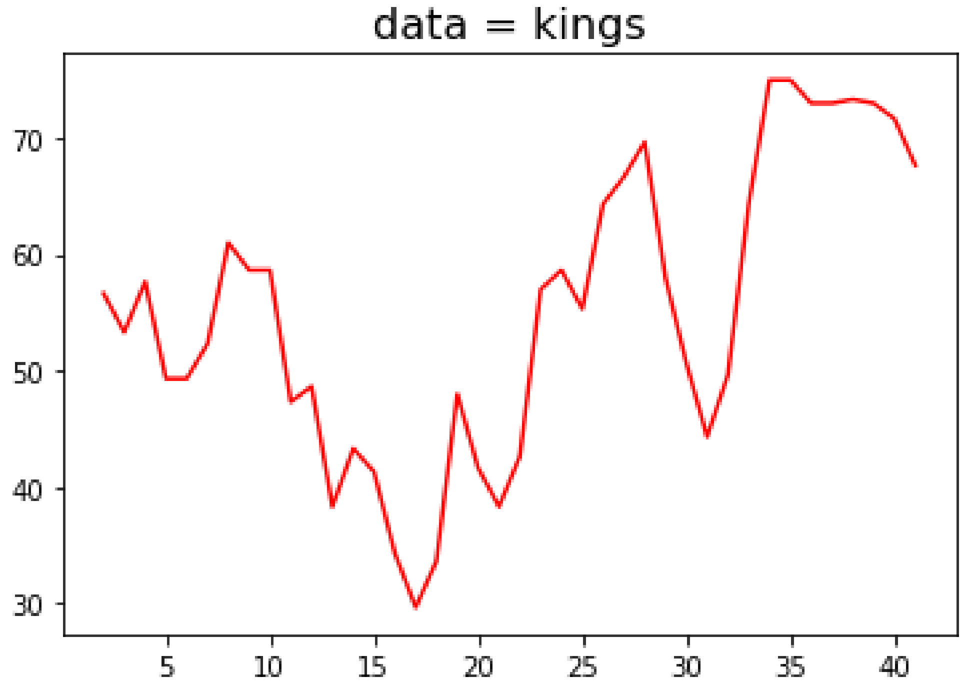



```
#H0: Data is not stationary
#p-value: 0.001005 ie <= 0.05, Null Hypothesis rejected, so, the data is stationary
```

Moving Average 3

```python
#Moving average/Rolloing average @3
kings_ma3 = kings.rolling(window=3).mean()
kings_ma3.head(10) # 1st & 2nd obs will be na

#lineplot
plt.plot(kings_ma3, 'r')
plt.title('data = kings', fontsize=16)
```

	age
0	NaN
1	NaN
2	56.666667
3	53.333333
4	57.666667
5	49.333333
6	49.333333
7	52.333333
8	61.000000
9	58.666667



Moving Average 3

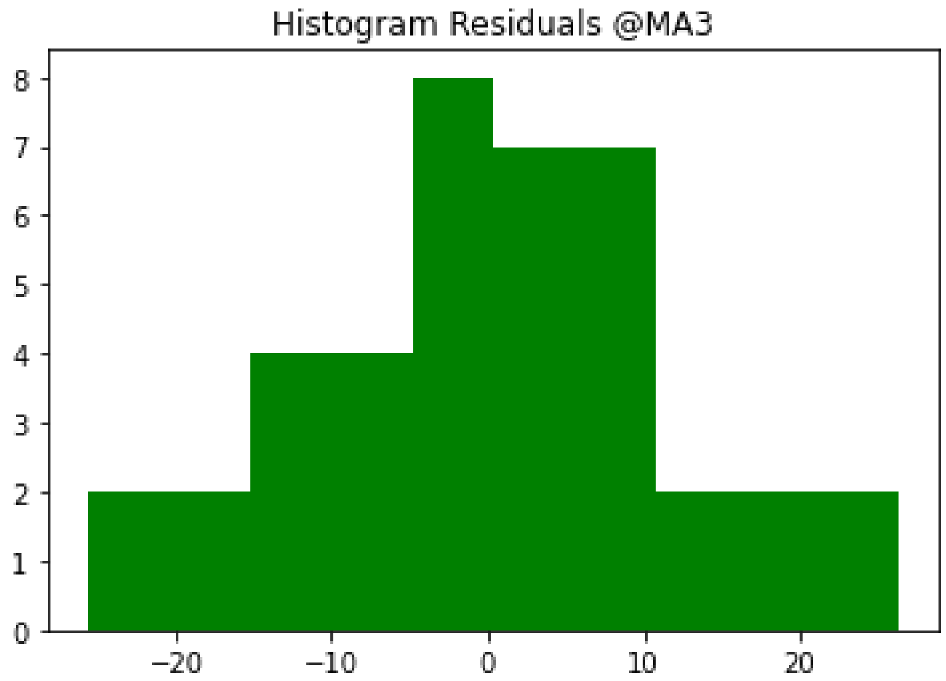
```
#Residuals / errors
kings_ma3_res = kings - kings_ma3
kings_ma3_res.head()
kings_ma3_res = kings_ma3_res.dropna()
kings_ma3_res.head()
```

```
In [65]: kings_ma3_res.head()
Out[65]:
         age
0        NaN
1        NaN
2  10.333333
3  -3.333333
4  -1.666667
```

```
In [66]: kings_ma3_res = kings_ma3_res.dropna()

In [67]: kings_ma3_res.head()
Out[67]:
         age
2  10.333333
3  -3.333333
4  -1.666667
5  -7.333333
6   0.666667
```

Moving Average 3

```
#Plotting histogram for residuals
plt.hist(kings_ma3_res, facecolor = 'g')
plt.title('Histogram Residuals @MA3')
```



RMSE_ma3

```
#average/mean of squared residuals/ errors
kings_ma3_mse = (kings_ma3_se.sum())/len(kings_ma3_se)
print(kings_ma3_mse) #128.75277777777778
```

```
#_________________________________________________
#Squaring residuals/ errors
kings_ma3_se = pow(kings_ma3_res,2)
kings_ma3_se.head()
```

```
In [87]: print(kings_ma3_mse)
age      128.752778
dtype: float64
```

```
In [65]: kings_ma3_res.head()
Out[65]:
         age
0        NaN
1        NaN
2  10.333333
3  -3.333333
4  -1.666667
```

```
In [85]: kings_ma3_se.head()
Out[85]:
          age
2  106.777778
3   11.111111
4    2.777778
5   53.777778
6    0.444444
```

```
#Root of average/mean of squared residuals/ errors
kings_ma3_rmse = sqrt(kings_ma3_mse)
print(kings_ma3_rmse) #11.346928120763689
```

```
In [89]: print(kings_ma3_rmse)
11.346928120763689
```

births

Libraries

```
# Jesus is King of Kings!
import os
os.chdir('C:\\Users\\Dr Vinod\\Desktop\\WD_python')
import pandas as pd
pd.set_option('display.max_columns', None)
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import statsmodels.formula.api as smf
from statsmodels.tsa.stattools import adfuller
%matplotlib inline
from math import sqrt
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
```

Import Data

```python
births = pd.read_csv('births.csv', date_parser=True)
births.info()
births.head()
```

```
In [4]: births.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 168 entries, 0 to 167
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   obsno       168 non-null    int64
 1   month_year  168 non-null    object
 2   births      168 non-null    float64
dtypes: float64(1), int64(1), object(1)
memory usage: 4.1+ KB
```

```
In [5]: births.head()
Out[5]:
   obsno month_year  births
0      1     Jan-46  26.663
1      2     Feb-46  23.598
2      3     Mar-46  26.931
3      4     Apr-46  24.740
4      5     May-46  25.806
```

2 digit year

births - DataFrame

Index	obsno	ıonth_yea	births
0	1	Jan-46	26.663
1	2	Feb-46	23.598
2	3	Mar-46	26.931
3	4	Apr-46	24.74
4	5	May-46	25.806
5	6	Jun-46	24.364
6	7	Jul-46	24.477
7	8	Aug-46	23.901
8	9	Sep-46	23.175
9	10	Oct-46	23.227
10	11	Nov-46	21.672

Index

```
#Indexing month_year
#2 digit year without century while converting 4 digit format,
#python considers as present century so adjusting the year
births.index = pd.DatetimeIndex(births.month_year)+pd.DateOffset(years=-100)
births.info()
births.head()
```

```
In [7]: births.info()
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 168 entries, 1946-01-01 to 1959-12-01
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   obsno       168 non-null    int64
 1   month_year  168 non-null    object
 2   births      168 non-null    float64
dtypes: float64(1), int64(1), object(1)
memory usage: 5.2+ KB
```

```
In [8]: births.head()
Out[8]:
            obsno month_year  births
month_year
1946-01-01      1     Jan-46  26.663
1946-02-01      2     Feb-46  23.598
1946-03-01      3     Mar-46  26.931
1946-04-01      4     Apr-46  24.740
1946-05-01      5     May-46  25.806
```

Data

```
#Removing unecessary variables
births = births.drop(['obsno','month_year'], axis=1)
births.info()
births.shape #168, 1
births.head()
```

```
In [8]: births.head()
Out[8]:
            obsno month_year  births
month_year
1946-01-01      1     Jan-46  26.663
1946-02-01      2     Feb-46  23.598
1946-03-01      3     Mar-46  26.931
1946-04-01      4     Apr-46  24.740
1946-05-01      5     May-46  25.806
```

old

```
In [10]: births.info()
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 168 entries, 1946-01-01 to 1959-12-01
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   births  168 non-null    float64
dtypes: float64(1)
memory usage: 2.6 KB
```

```
In [12]: births.head()
Out[12]:
            births
month_year
1946-01-01  26.663
1946-02-01  23.598
1946-03-01  26.931
1946-04-01  24.740
1946-05-01  25.806
```

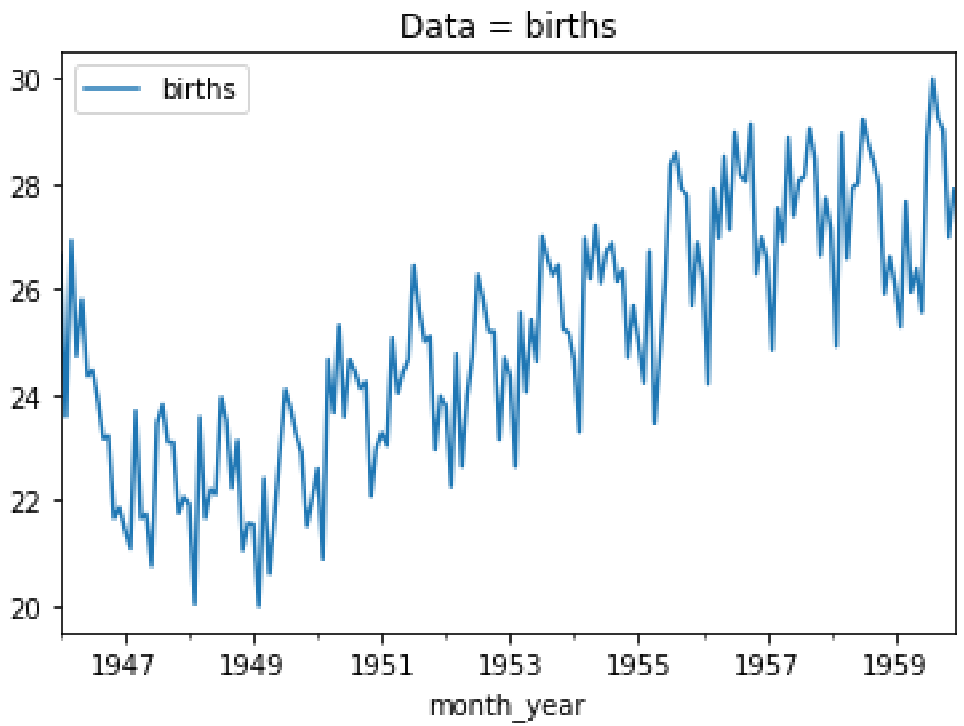
```
#Variable births
births.describe()
'''
            births
count   168.000000
mean     25.059310
std       2.318791
min      20.000000
25%      23.280750
50%      24.957000
75%      26.878750
max      30.000000'''
```

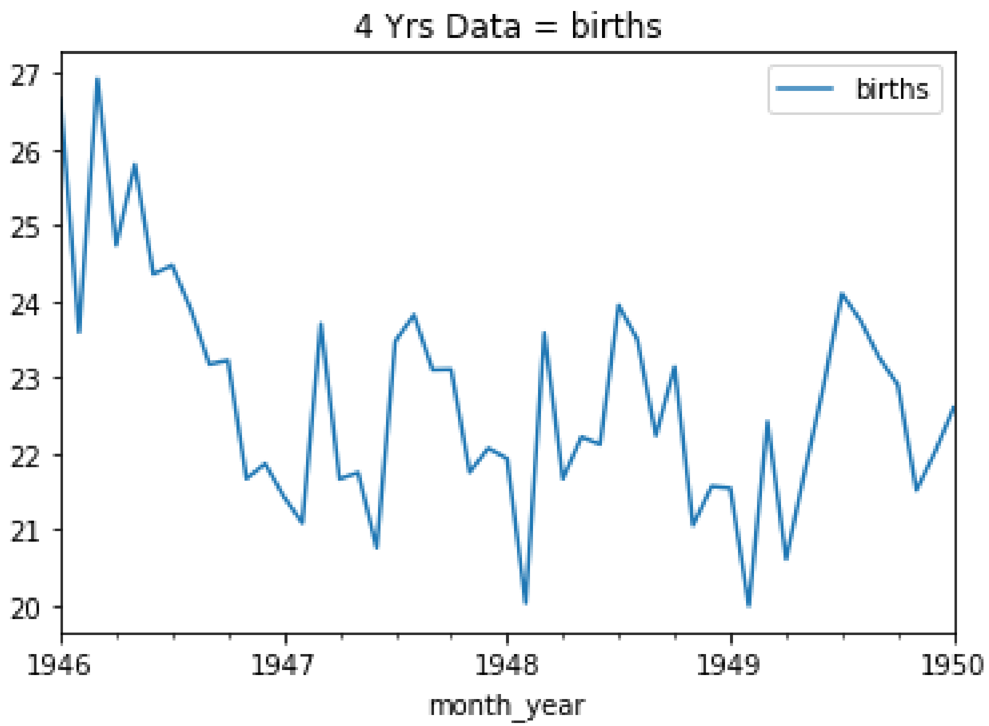
Plots

First 4 years means 12*4=48 data points, last will not be considered by python [indexing starts from 0], that's wht 49

```
#Lineplot
births.plot()
plt.title('Data = births')
```

```
#Only 4years of data
births[:49].plot()
plt.title('4 Yrs Data = births')
```

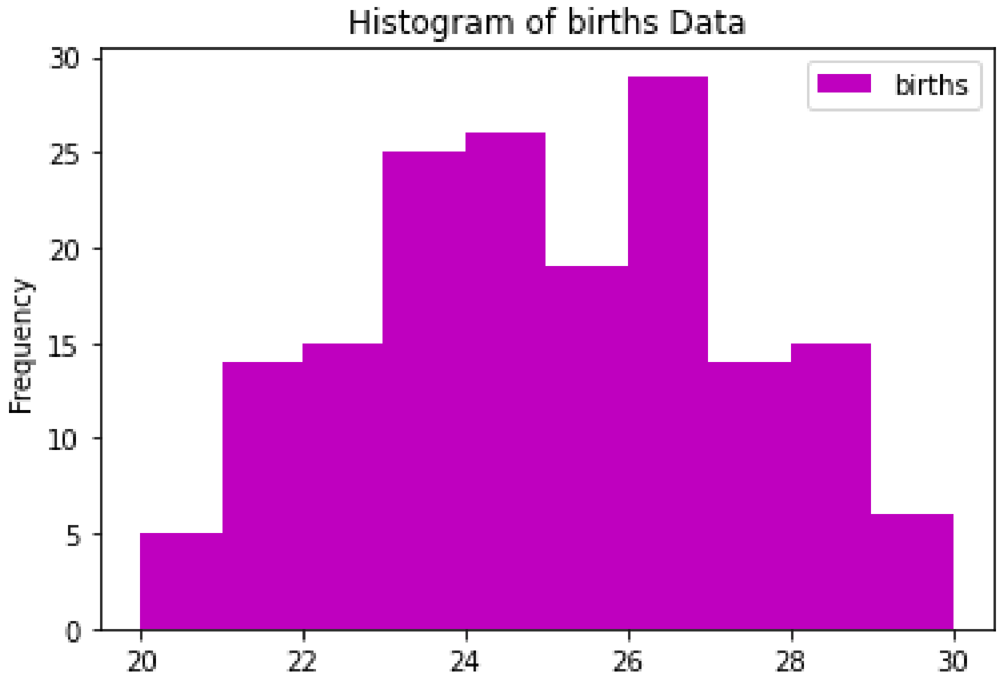


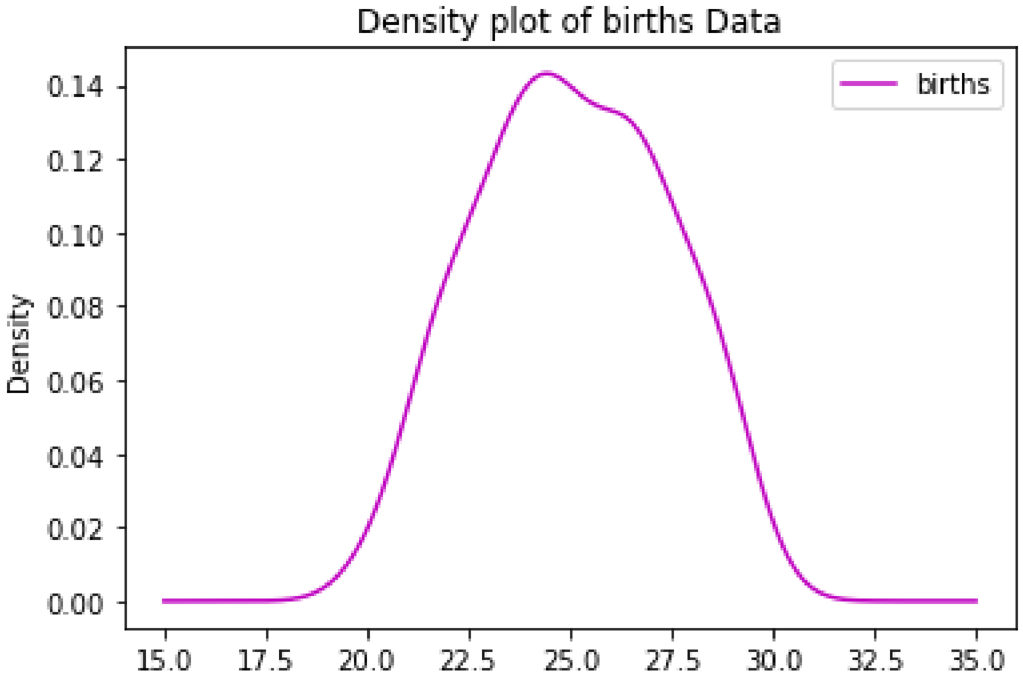


plots

```
#Histogram
births.plot(kind='hist', color = 'm')
plt.title('Histogram of births Data')
```

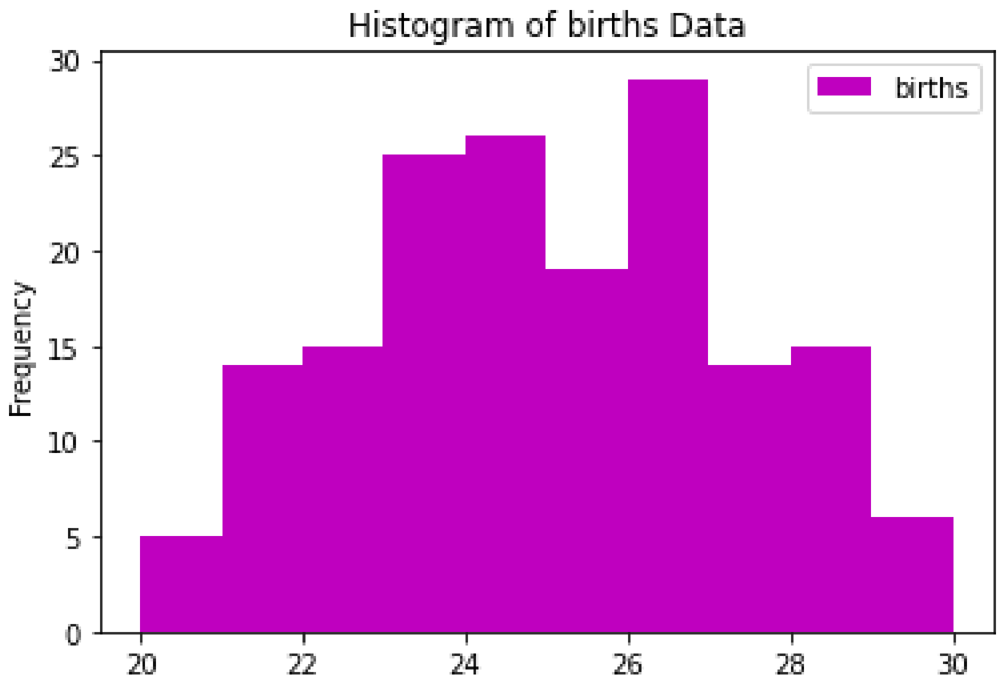
```
#Density plot
births.plot(kind='kde', color = 'm')
plt.title('Density plot of births Data')
```

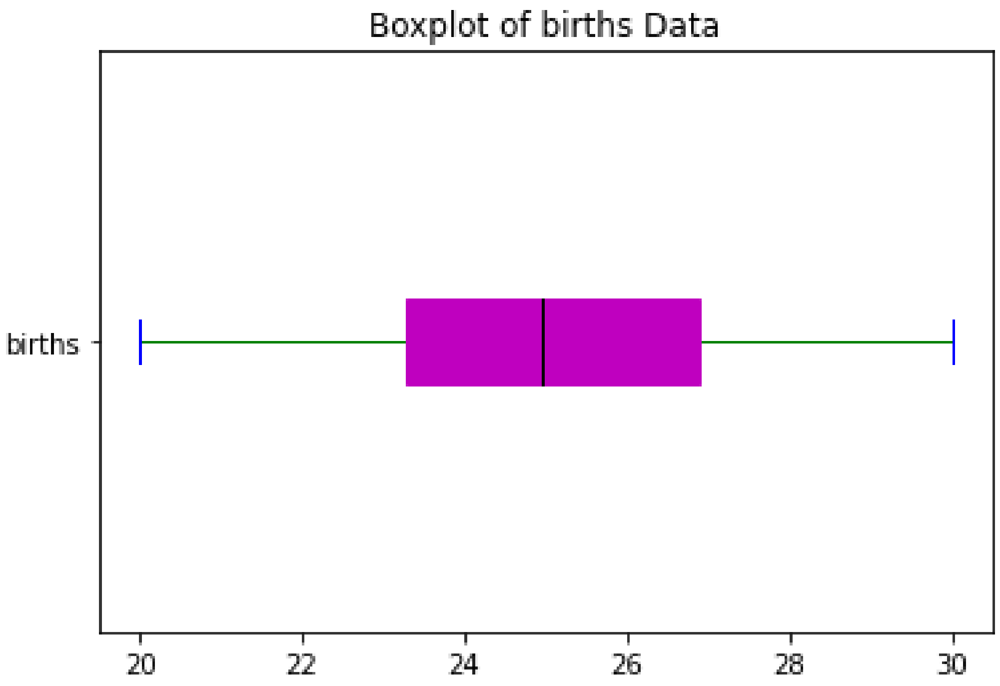




Plots

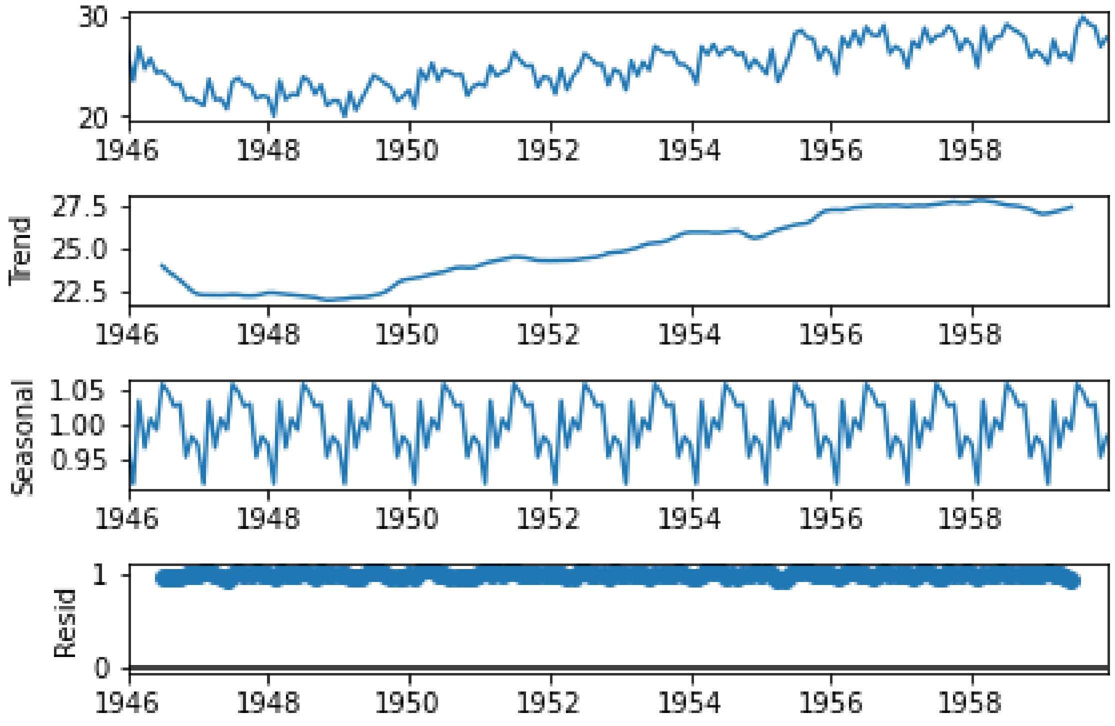
```
#Boxplot
props2  = dict(boxes = 'm', whiskers ='green', medians = 'black', caps = 'blue')
births.plot.box(color = props2 , patch_artist = True, vert = False)
plt.title('Boxplot of births Data')
```





Decompose

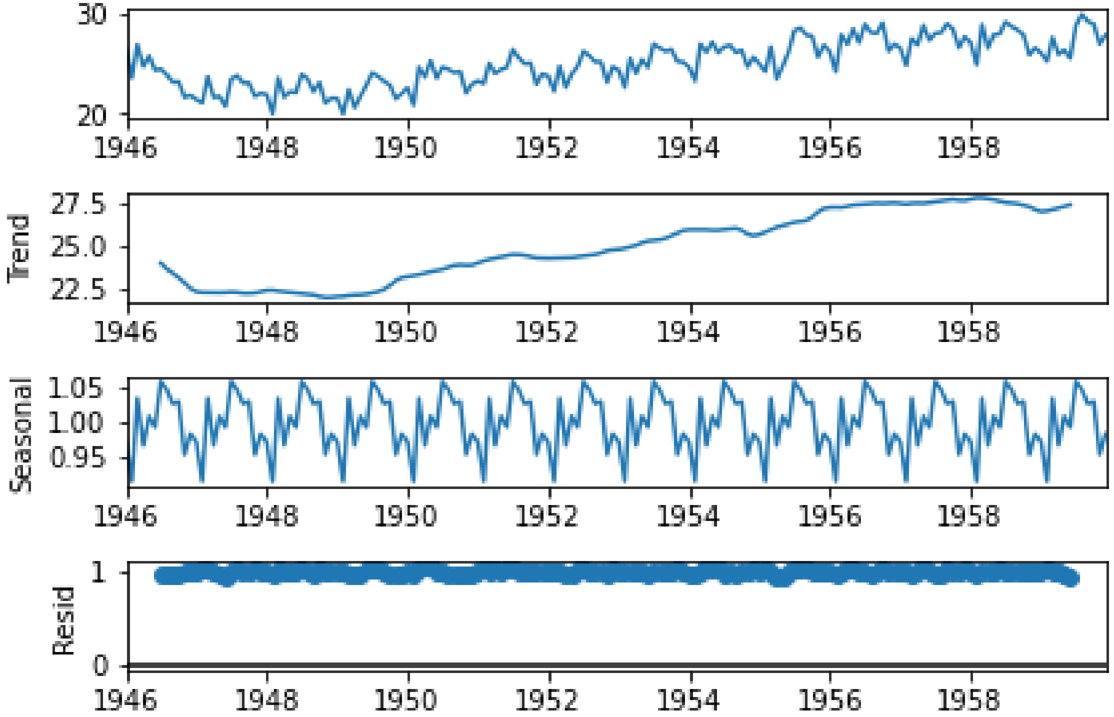
```
#Decompose
from statsmodels.tsa.seasonal import seasonal_decompose
# Season Decompose with Multiplicative model
births_dec_m = seasonal_decompose(births, model='multiplicative')
```



Trend

```
births_dec_m.trend.head(20)
#First 6 and last 6 values are Na's due calculation of seasonality indices of 12 months
```

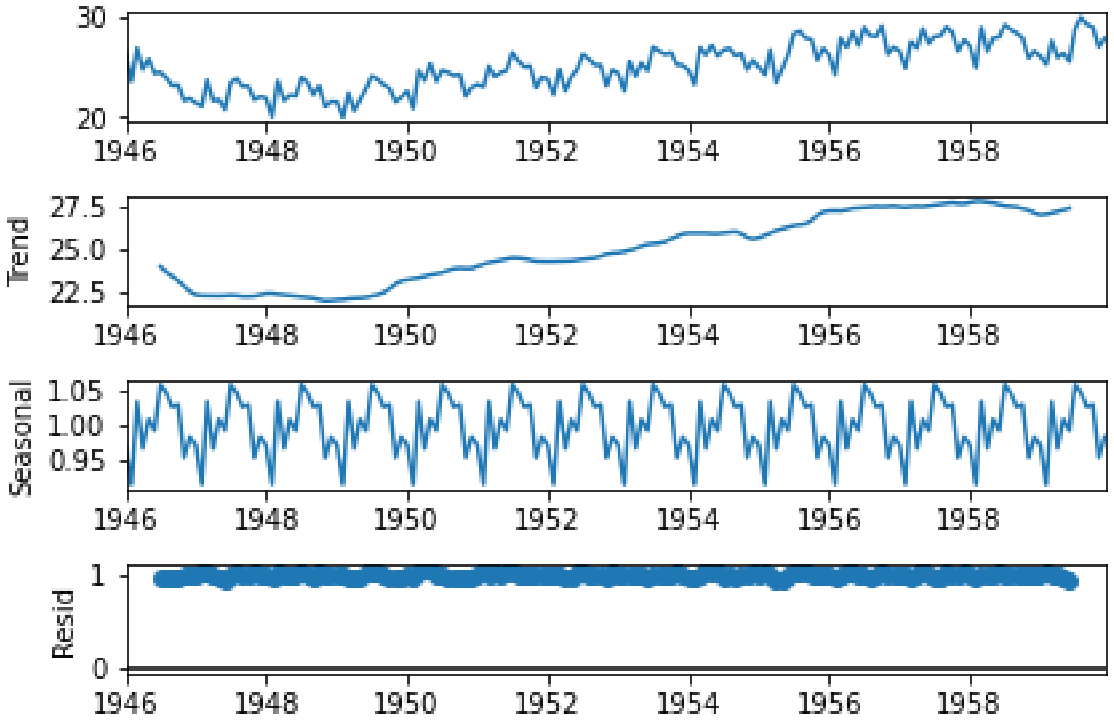
```
In [24]: births_dec_m.trend.head(20)
Out[24]:
month_year
1946-01-01          NaN
1946-02-01          NaN
1946-03-01          NaN
1946-04-01          NaN
1946-05-01          NaN
1946-06-01          NaN
1946-07-01    23.984333
1946-08-01    23.662125
1946-09-01    23.423333
1946-10-01    23.161125
1946-11-01    22.864250
1946-12-01    22.545208
1947-01-01    22.353500
1947-02-01    22.308708
1947-03-01    22.302583
1947-04-01    22.294792
1947-05-01    22.293542
1947-06-01    22.305625
1947-07-01    22.334833
1947-08-01    22.311667
Name: trend, dtype: float64
```



Seasonality

```
births_dec_m.seasonal
```

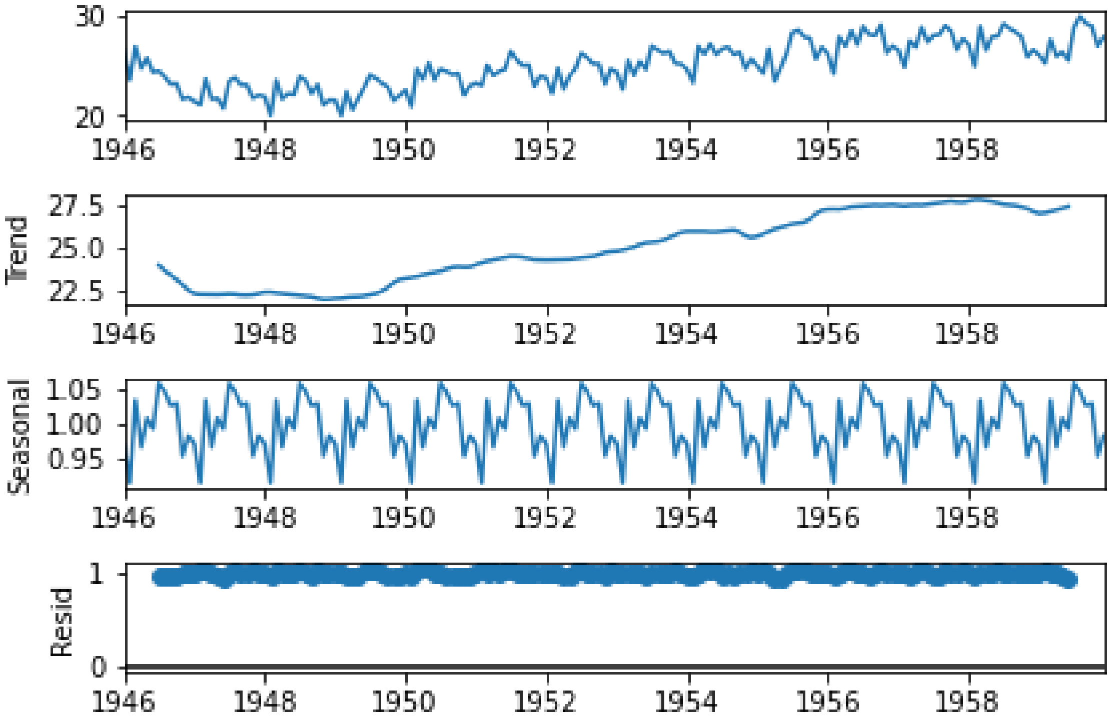
```
In [25]: births_dec_m.seasonal
Out[25]:
month_year
1946-01-01    0.972903
1946-02-01    0.916649
1946-03-01    1.035164
1946-04-01    0.967842
1946-05-01    1.009504
                ...
1959-08-01    1.047323
1959-09-01    1.027250
1959-10-01    1.030856
1959-11-01    0.955121
1959-12-01    0.984295
Name: seasonal, Length: 168, dtype: float64
```



```
births_dec_m.resid.head(20)
#First 6 and last 6 values are Na's due calculation of seasonality indices of 12 months
```

Residual

```
In [26]: births_dec_m.resid.head(20)
Out[26]:
month_year
1946-01-01         NaN
1946-02-01         NaN
1946-03-01         NaN
1946-04-01         NaN
1946-05-01         NaN
1946-06-01         NaN
1946-07-01    0.963590
1946-08-01    0.964455
1946-09-01    0.963152
1946-10-01    0.972827
1946-11-01    0.992392
1946-12-01    0.985528
1947-01-01    0.985801
1947-02-01    1.031285
1947-03-01    1.026949
1947-04-01    1.004225
1947-05-01    0.966523
1947-06-01    0.936379
1947-07-01    0.992565
```



Model!

```
#Model with triple exponential smoothing
# applicable when trend and seasonality exists
from statsmodels.tsa.holtwinters import ExponentialSmoothing

births_es = ExponentialSmoothing(births, seasonal_periods=12,
                                 trend='add', seasonal='add').fit()
births_es.summary()
```

Model Summary!

```
'''
                     ExponentialSmoothing Model Results
================================================================================
Dep. Variable:                   births   No. Observations:                  168
Model:             ExponentialSmoothing   SSE                             63.346
Optimized:                         True   AIC                           -131.859
Trend:                         Additive   BIC                            -81.875
Seasonal:                      Additive   AICC                          -127.268
Seasonal Periods:                    12   Date:                 Fri, 26 Mar 2021
Box-Cox:                          False   Time:                         00:25:39
Box-Cox Coeff.:                    None
=================================================================================
```

Model Summary!

```
================================================================================
                               coeff                 code              optimized
--------------------------------------------------------------------------------
smoothing_level            0.9401515                alpha                   True
smoothing_trend           8.5161e-11                 beta                   True
smoothing_seasonal         5.805e-12                gamma                   True
initial_level              27.155582                  l.0                   True
initial_trend              0.0062209                  b.0                   True
initial_seasons.0         -0.5917886                  s.0                   True
initial_seasons.1         -2.0910713                  s.1                   True
initial_seasons.2          0.9121398                  s.2                   True
initial_seasons.3         -0.7605586                  s.3                   True
initial_seasons.4          0.3205229                  s.4                   True
initial_seasons.5         -0.1308801                  s.5                   True
initial_seasons.6          1.4655643                  s.6                   True
initial_seasons.7          1.2730839                  s.7                   True
initial_seasons.8          0.7820954                  s.8                   True
initial_seasons.9          0.8415282                  s.9                   True
initial_seasons.10        -1.0539487                 s.10                   True
initial_seasons.11        -0.3096054                 s.11                   True
--------------------------------------------------------------------------------'''
```

residuals!

```
#Residual given by the model
births_es_res = births_es.resid
births_es_res
```

```
In [30]: births_es_res
Out[30]:
month_year
1946-01-01     0.092986
1946-02-01    -1.566373
1946-03-01     0.229823
1946-04-01    -0.510768
1946-05-01    -0.051871
                 ...
1959-08-01     1.421315
1959-09-01    -0.169169
1959-10-01    -0.324778
1959-11-01    -0.150182
1959-12-01     0.145448
Length: 168, dtype: float64
```

births_es_res - Series

month_year	0
1946-01-01 00:00:00	0.0929857
1946-02-01 00:00:00	-1.56637
1946-03-01 00:00:00	0.229823
1946-04-01 00:00:00	-0.510768
1946-05-01 00:00:00	-0.0518709
1946-06-01 00:00:00	-0.999922

Format | Resize | Background color | Column min/max | Save and Close | Close

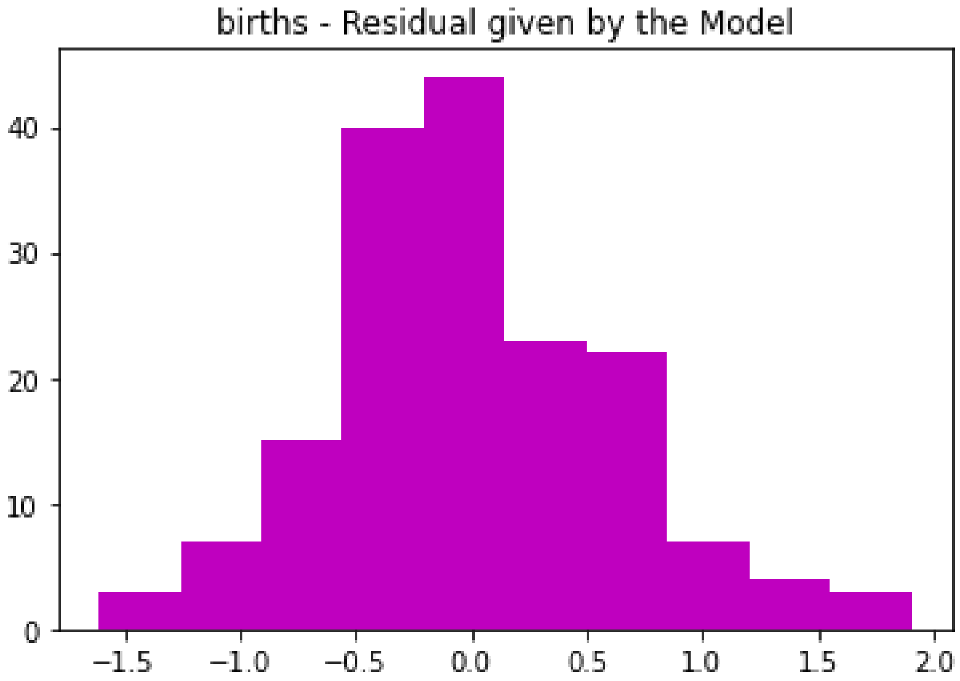
rmse!

```
#Squaring residuals/ errors
births_es_se = pow(births_es_res,2)
births_es_se.head()

#average/mean of squared residuals/ errors
births_es_mse = (births_es_se.sum())/len(births_es_se)
print(births_es_mse) #0.3770609200564516

#Root of average/mean of squared residuals/ errors
births_es_rmse = sqrt(births_es_mse)
print(births_es_rmse) #0.6140528642197279
```

```
In [31]: births_es_se = pow(births_es_res,2)

In [32]: births_es_se.head()
Out[32]:
month_year
1946-01-01    0.008646
1946-02-01    2.453525
1946-03-01    0.052819
1946-04-01    0.260884
1946-05-01    0.002691
dtype: float64

In [33]: births_es_mse = (births_es_se.sum())/len(births_es_se)

In [34]: print(births_es_mse) #0.3770609200564516
0.3770609200209467

In [35]: births_es_rmse = sqrt(births_es_mse)

In [36]: print(births_es_rmse) #0.6140528642197279
0.6140528641908176
```

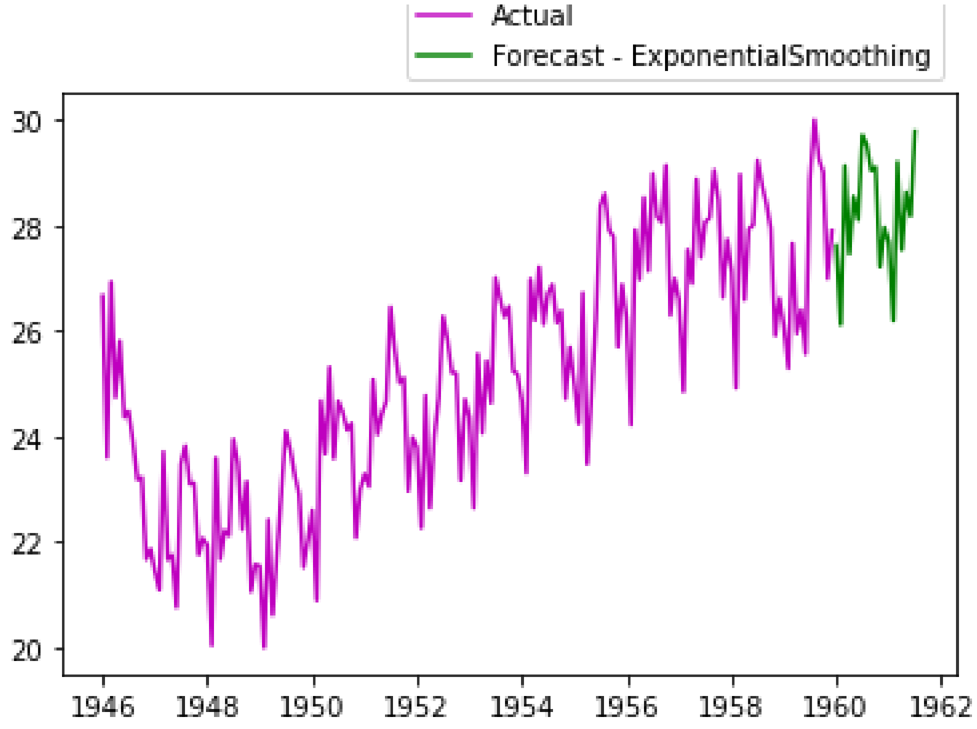
Histogram of residulas!

```
#Histogram of residuals
plt.hist(births_es_res, color = 'm')
plt.title('births - Residual given by the Model')
plt.show()
```



Forecast and plot

```
#forecasting/ predicting
births_pred2 = births_es.forecast(steps=19)
print(births_pred2)

#Plot actual and forecast
plt.plot(births, color = 'm')
plt.plot(births_pred2, color = 'g')
plt.legend(['Actual', 'Forecast - ExponentialSmoothing'],
           bbox_to_anchor=(1, 1), loc=4)
plt.show()
```



Let's try autoarima!

```
#______________________________let's go for autoarima
#Test for stationarity
from statsmodels.tsa.stattools import adfuller
births_adf = adfuller(births)
births_adf
```

```
In [121]: births_adf
Out[121]:
(-0.33128063038051875,
 0.9209557340544081,
 13,
 154,
 {'1%': -3.473542528196209,
  '5%': -2.880497674144038,
  '10%': -2.576878053634677},
 385.9083089080856)
```

Series is NOT stationary!

autoarima!

```
#________________________________Applying auto - arima to forecast
#!pip install pmdarima

from pmdarima import auto_arima

births_mod = auto_arima(births)
births_mod.summary()
```

model!

```
                                     SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  168
Model:               SARIMAX(2, 1, 1)   Log Likelihood                -271.935
Date:                Fri, 26 Mar 2021   AIC                            551.870
Time:                        00:35:55   BIC                            564.342
Sample:                             0   HQIC                           556.932
                                - 168
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.2509      0.095      2.643      0.008       0.065       0.437
ar.L2          0.3441      0.116      2.977      0.003       0.118       0.571
ma.L1         -0.9143      0.065    -14.166      0.000      -1.041      -0.788
sigma2         1.5133      0.194      7.788      0.000       1.132       1.894
===================================================================================
Ljung-Box (L1) (Q):                  0.20   Jarque-Bera (JB):                  2.24
Prob(Q):                             0.66   Prob(JB):                          0.33
Heteroskedasticity (H):              1.22   Skew:                              0.14
Prob(H) (two-sided):                 0.46   Kurtosis:                          2.51
===================================================================================

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

```python
#Residual  given by the model
births_mod_res = births_mod.resid()
births_mod_res
```

```python
#Adding index and converting to datframe
births_mod_res1 = pd.DataFrame(births_mod_res, index=births.index)
births_mod_res1
```

births_mod_res - NumPy object array

	0
0	26.663
1	-3.06497
2	1.67623
3	-0.538892
4	0.00151674
5	-0.954345
6	-0.735586
7	-0.762661
8	-1.30228
9	-0.73678
10	-1.98186

Format Resize ☑ Background color

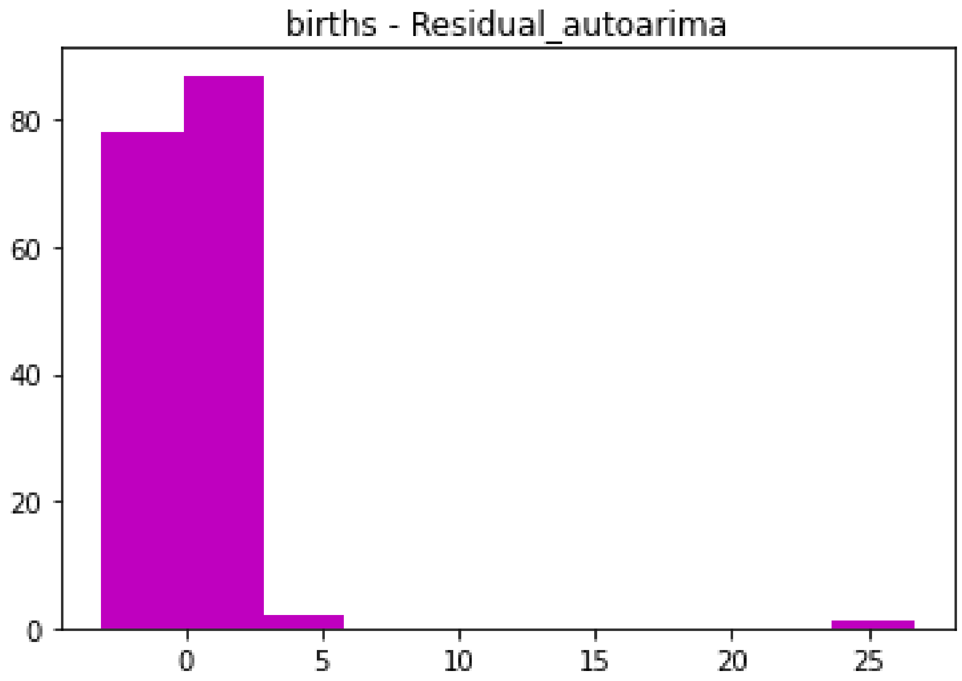
births_mod_res1 - DataFrame

month_year	0
1946-01-01 00:00:00	26.663
1946-02-01 00:00:00	-3.06497
1946-03-01 00:00:00	1.67623
1946-04-01 00:00:00	-0.538892
1946-05-01 00:00:00	0.00151674
1946-06-01 00:00:00	-0.954345
1946-07-01 00:00:00	-0.735586
1946-08-01 00:00:00	-0.762661
1946-09-01 00:00:00	-1.30228
1946-10-01 00:00:00	-0.73678
1946-11-01 00:00:00	-1.98186
1946-12-01 00:00:00	-1.21939

Format Resize ☑ Background color ☑ Column min/max

Histogram of residuals

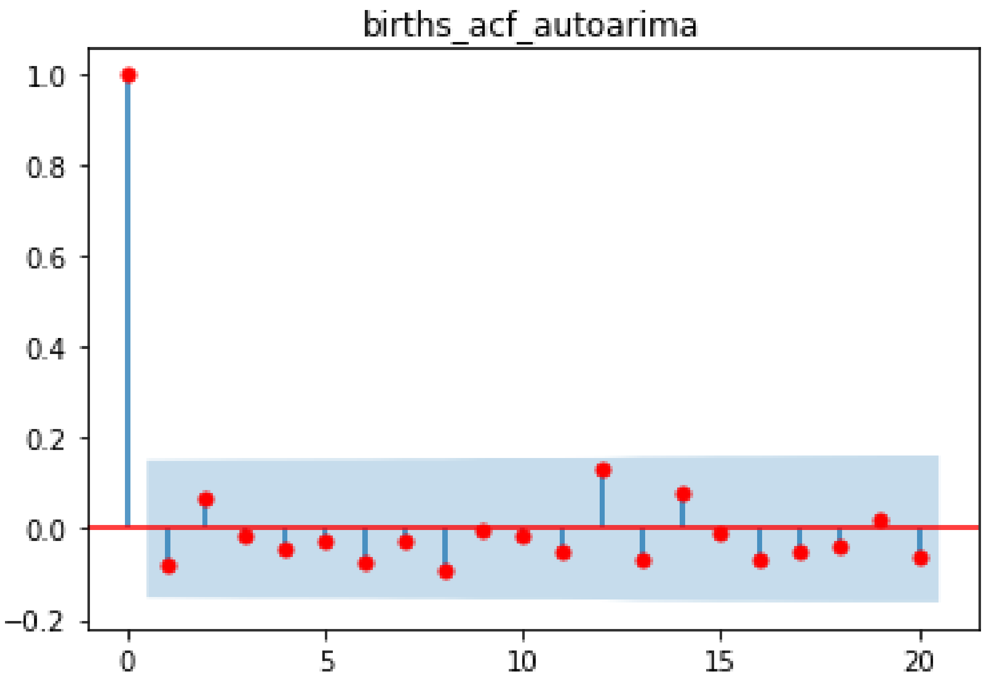
```
#Histogram of residuals
plt.hist(births_mod_res1, color = 'm')
plt.title('births - Residual_autoarima')
plt.show()
```



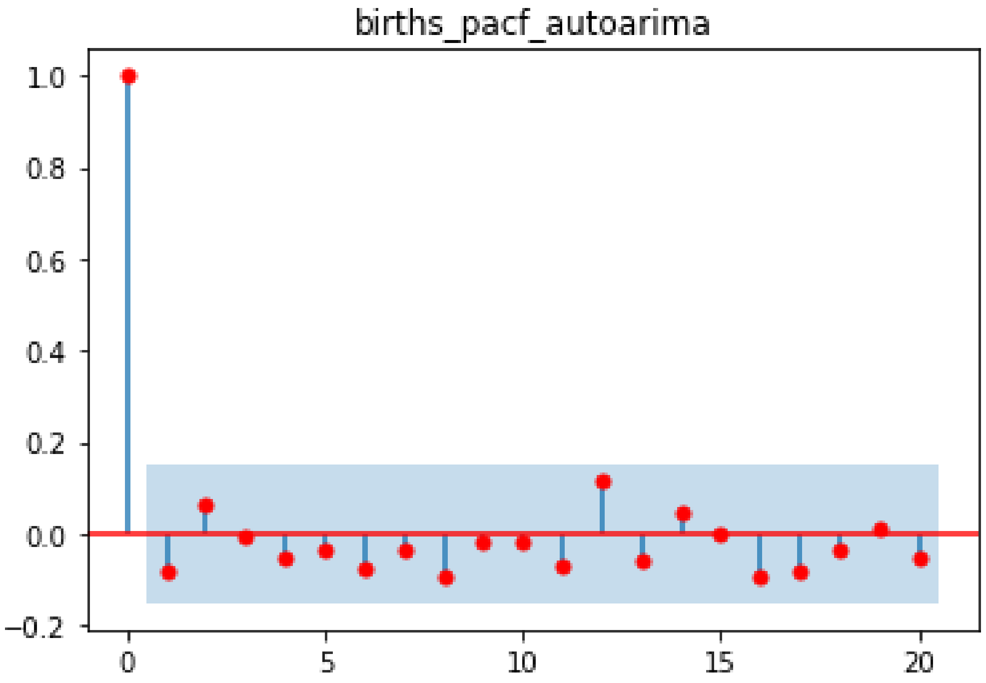
rmse!

```
#__________________________________rmse
#Squaring residuals/ errors
births_mod_se = pow(births_mod_res1,2)
births_mod_se.head()

#average/mean of squared residuals/ errors
births_mod_mse = (births_mod_se.sum())/len(births_mod_se)
print(births_mod_mse) #5.757373

#Root of average/mean of squared residuals/ errors
births_mod_rmse = sqrt(births_mod_mse)
print(births_mod_rmse) #2.399452542080286
```

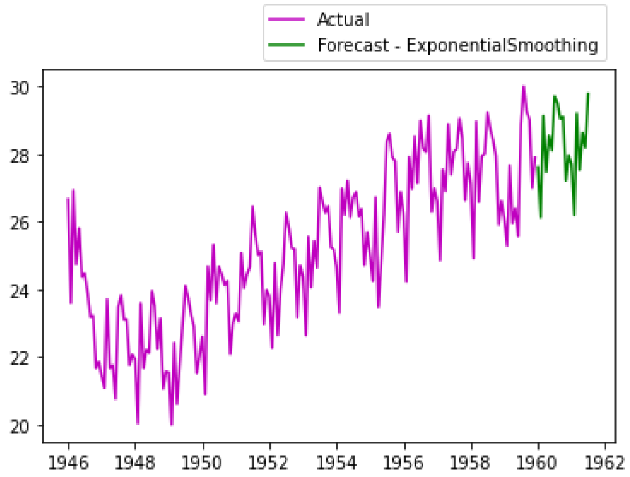
```python
#Plotting acf & pacf - residual
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf

plot_acf(births_mod_res1, lags=20, color ='r',
         title='births_acf_autoarima')
```



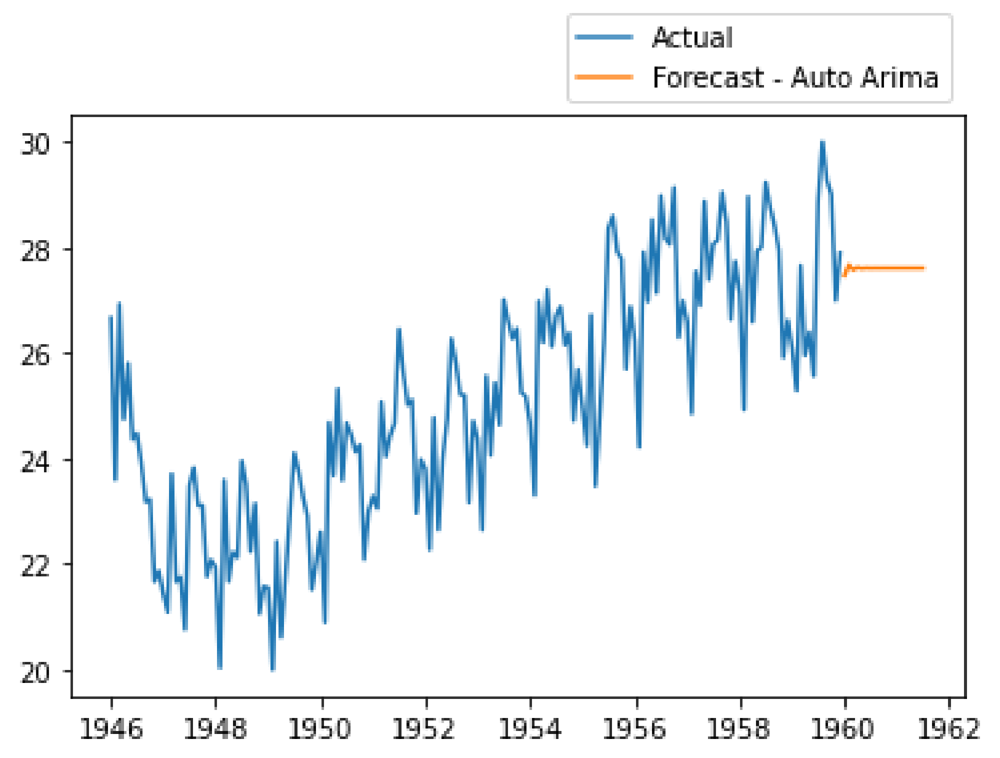
```
plot_pacf(births_mod_res1, lags=20, color = 'r',
          title='births_pacf_autoarima')
```



Forecast and plot




```
#Forecasting next 19 periods
births_mod_pred = births_mod.predict(n_periods=19)
births_mod_pred

#Adding index to forecast and converting to dataframe

births_mod_pred = pd.DataFrame(births_mod_pred,
                               index=pd.date_range(start='1960-01-01',
                                                   periods=19, freq='MS'))

births_mod_pred

#Plot actual and forecast
plt.plot(births)
plt.plot(births_mod_pred)
plt.legend(['Actual', 'Forecast - Auto Arima'],
           bbox_to_anchor=(1, 1), loc=4)
plt.show()
```



Happy Learning!