# Assignment-18-Forecasting-01-(Airlines Data)

In [121]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [122]:

```
df=pd.read_excel("C:/Users/LENOVO/Documents/Custom Office Templates/Airlines+Data.xlsx")
df
```

Out[122]:

| | Month | Passengers |
|---|---|---|
| 0 | 1995-01-01 | 112 |
| 1 | 1995-02-01 | 118 |
| 2 | 1995-03-01 | 132 |
| 3 | 1995-04-01 | 129 |
| 4 | 1995-05-01 | 121 |
| ... | ... | ... |
| 91 | 2002-08-01 | 405 |
| 92 | 2002-09-01 | 355 |
| 93 | 2002-10-01 | 306 |
| 94 | 2002-11-01 | 271 |
| 95 | 2002-12-01 | 306 |

96 rows × 2 columns

In [123]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96 entries, 0 to 95
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Month       96 non-null     datetime64[ns]
 1   Passengers  96 non-null     int64
dtypes: datetime64[ns](1), int64(1)
memory usage: 1.6 KB
```

In [127]:

```
df1=df.set_index('Month')
df1
```

Out[127]:

| | Passengers |
|---|---|
| **Month** | |
| **1995-01-01** | 112 |
| **1995-02-01** | 118 |
| **1995-03-01** | 132 |
| **1995-04-01** | 129 |
| **1995-05-01** | 121 |
| **...** | ... |
| **2002-08-01** | 405 |
| **2002-09-01** | 355 |
| **2002-10-01** | 306 |
| **2002-11-01** | 271 |
| **2002-12-01** | 306 |

96 rows × 1 columns

In [128]:

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 96 entries, 1995-01-01 to 2002-12-01
Data columns (total 1 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Passengers  96 non-null     int64
dtypes: int64(1)
memory usage: 1.5 KB
```

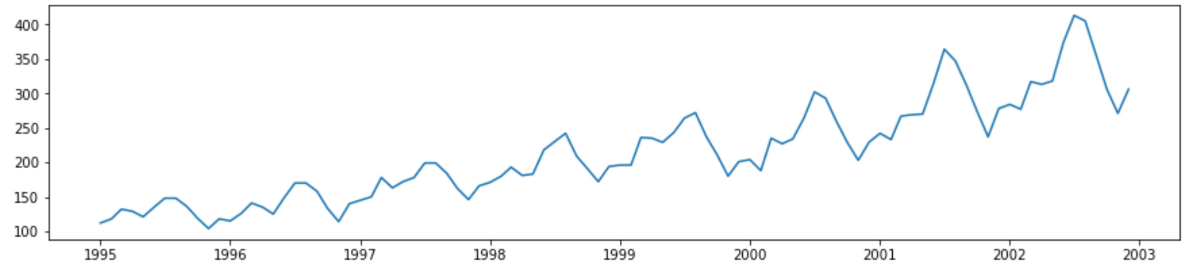
# Visualization of Data

In [129]:

```
# Visualization
plt.figure(figsize=(15,7))
# line plot
plt.subplot(211)
plt.plot(df1)
```

Out[129]:

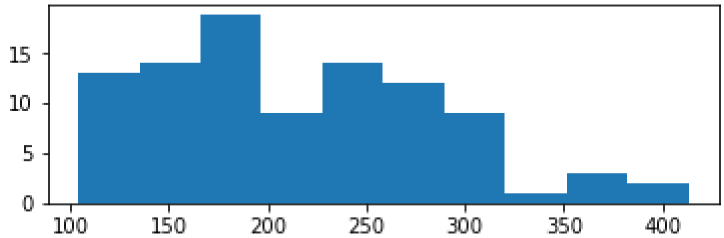
```
[<matplotlib.lines.Line2D at 0x3d85cc6d0>]
```




### From the above line diagram, the data is having Upward Exponentional Trend with Multiplicative Seasonality

In [130]:

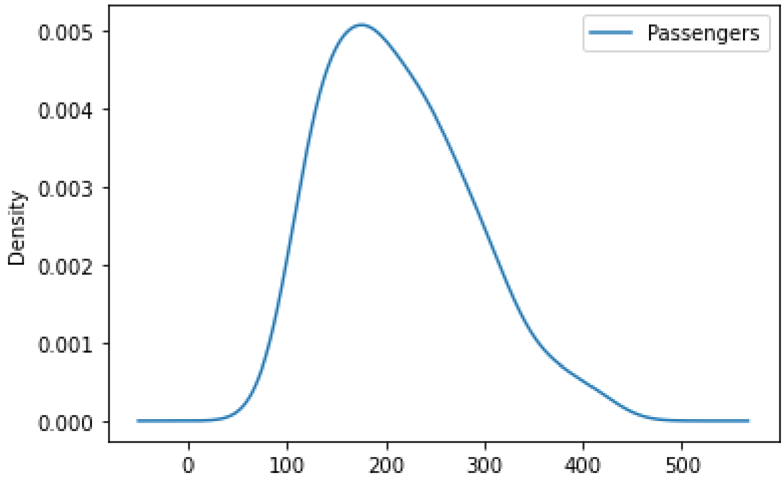
```
# histogram
plt.subplot(212)
plt.hist(df1)
plt.show()
```




### from the above histogram it is clear that the data is positively skewed.

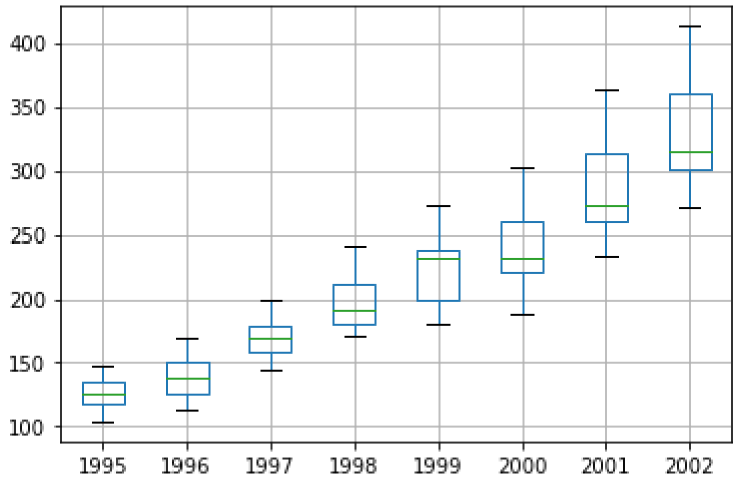
In [131]:

```
df1.plot(kind='kde')
pyplot.show()
```




In [132]:

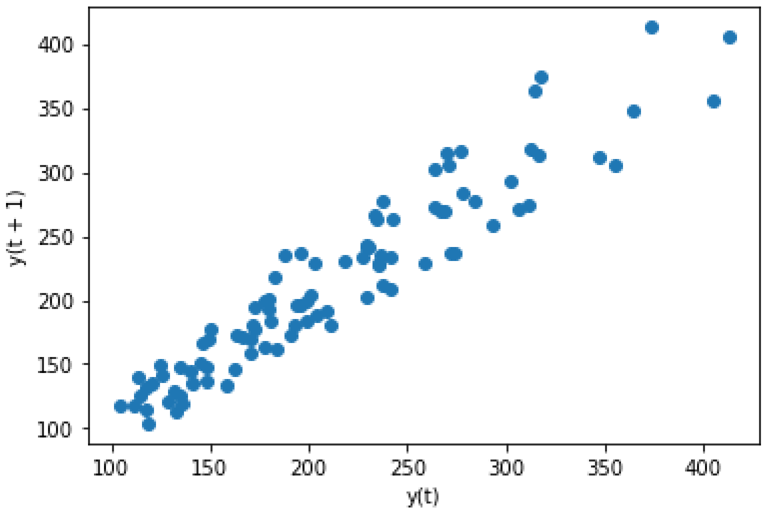
```
# create a boxplot of yearly data
from pandas import Grouper
airlines = pd.read_excel(r"C:/Users/LENOVO/Documents/Custom Office Templates/Airlines+Data.
groups = airlines.groupby(Grouper(freq = 'A'))
years = pd.DataFrame()
for name, group in groups:
    years[name.year] = group.values
years.boxplot()
plt.show()
```




**Lag plot**

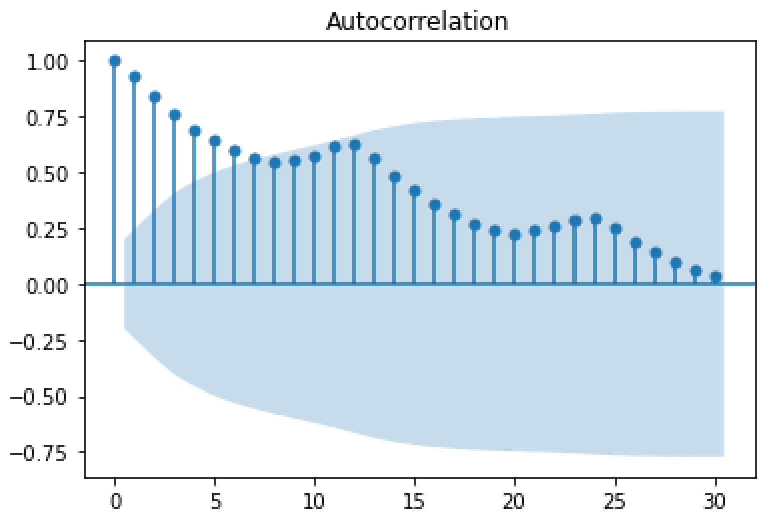
In [133]:

```
from pandas.plotting import lag_plot
lag_plot(df1)
plt.show()
```




In [134]:

```
# create an autocorrelation plot
from statsmodels.graphics.tsaplots import plot_acf
plot_acf(df1, lags = 30)
plt.show()
```




# Upsampling Data

In [135]:

```
from pandas import datetime
from matplotlib import pyplot
from pandas import read_excel
```

```
<ipython-input-135-4fecbf7532c3>:1: FutureWarning: The pandas.datetime class
is deprecated and will be removed from pandas in a future version. Import fr
om datetime module instead.
  from pandas import datetime
```

In [136]:

```
series = read_excel("C:/Users/LENOVO/Documents/Custom Office Templates/Airlines+Data.xlsx",
```

In [137]:

```
series
```

Out[137]:

```
Month
1995-01-01    112
1995-02-01    118
1995-03-01    132
1995-04-01    129
1995-05-01    121
             ...
2002-08-01    405
2002-09-01    355
2002-10-01    306
2002-11-01    271
2002-12-01    306
Name: Passengers, Length: 96, dtype: int64
```

In [138]:

```
upsampled = series.resample('D').mean()
print(upsampled.head(32))
```

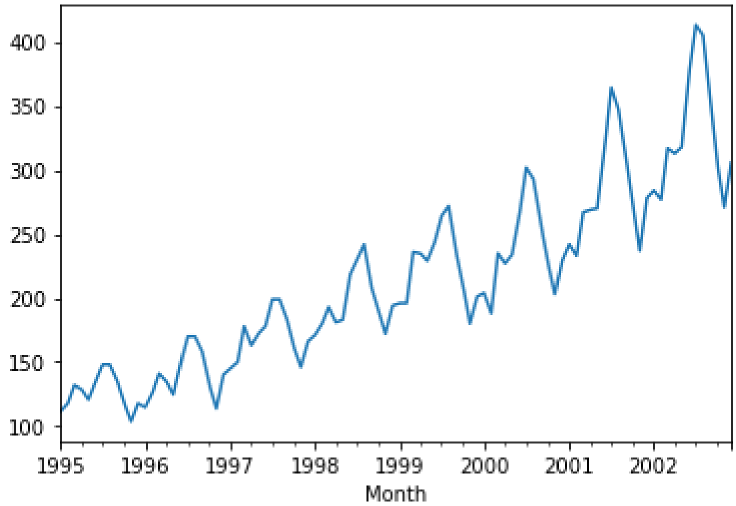
```
Month
1995-01-01    112.0
1995-01-02      NaN
1995-01-03      NaN
1995-01-04      NaN
1995-01-05      NaN
1995-01-06      NaN
1995-01-07      NaN
1995-01-08      NaN
1995-01-09      NaN
1995-01-10      NaN
1995-01-11      NaN
1995-01-12      NaN
1995-01-13      NaN
1995-01-14      NaN
1995-01-15      NaN
1995-01-16      NaN
1995-01-17      NaN
1995-01-18      NaN
1995-01-19      NaN
1995-01-20      NaN
1995-01-21      NaN
1995-01-22      NaN
1995-01-23      NaN
1995-01-24      NaN
1995-01-25      NaN
1995-01-26      NaN
1995-01-27      NaN
1995-01-28      NaN
1995-01-29      NaN
1995-01-30      NaN
1995-01-31      NaN
1995-02-01    118.0
Freq: D, Name: Passengers, dtype: float64
```

## interpolate the missing value

In [139]:

```
interpolated = upsampled.interpolate(method='linear')
print(interpolated.head(32))
interpolated.plot()
pyplot.show()
```

```
Month
1995-01-01    112.000000
1995-01-02    112.193548
1995-01-03    112.387097
1995-01-04    112.580645
1995-01-05    112.774194
1995-01-06    112.967742
1995-01-07    113.161290
1995-01-08    113.354839
1995-01-09    113.548387
1995-01-10    113.741935
1995-01-11    113.935484
1995-01-12    114.129032
1995-01-13    114.322581
1995-01-14    114.516129
1995-01-15    114.709677
1995-01-16    114.903226
1995-01-17    115.096774
1995-01-18    115.290323
1995-01-19    115.483871
1995-01-20    115.677419
1995-01-21    115.870968
1995-01-22    116.064516
1995-01-23    116.258065
1995-01-24    116.451613
1995-01-25    116.645161
1995-01-26    116.838710
1995-01-27    117.032258
1995-01-28    117.225806
1995-01-29    117.419355
1995-01-30    117.612903
1995-01-31    117.806452
1995-02-01    118.000000
Freq: D, Name: Passengers, dtype: float64
```




# Downsampling Data

In [140]:

```
# downsample to quarterly intervals
resample = series.resample('Q')
quarterly_mean_sales = resample.mean()
```

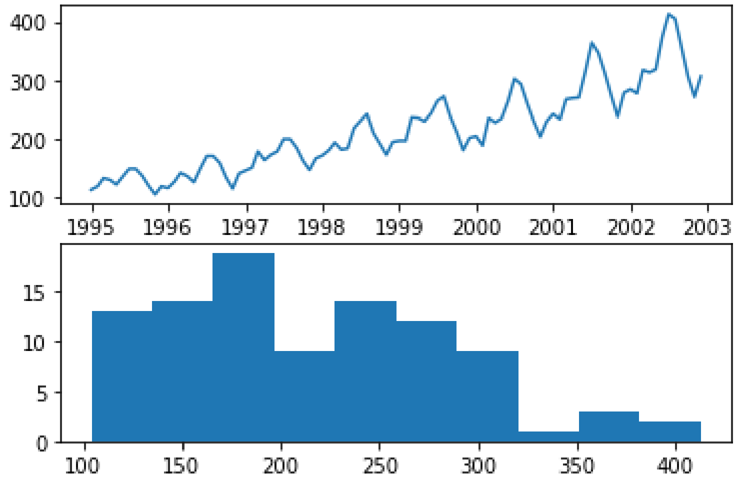
# Tranformations

In [141]:

```
ies = read_excel("C:/Users/LENOVO/Documents/Custom Office Templates/Airlines+Data.xlsx", hea
```

In [142]:

```
# line plot
pyplot.subplot(211)
pyplot.plot(series)
# histogram
pyplot.subplot(212)
pyplot.hist(series)
pyplot.show()
```




# Square Root Transform¶

In [143]:

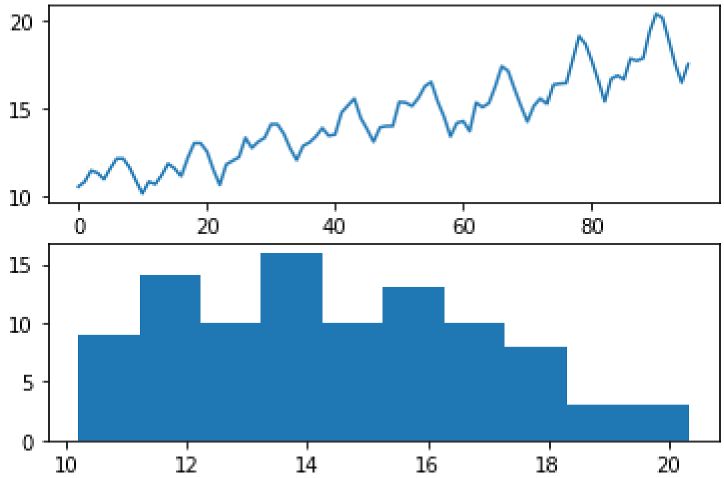
```
from pandas import DataFrame
from numpy import sqrt
```

In [144]:

```
dataframe = DataFrame(df1.values)
dataframe.columns = ['passengers']
dataframe['passengers'] = sqrt(dataframe['passengers'])
```

In [145]:

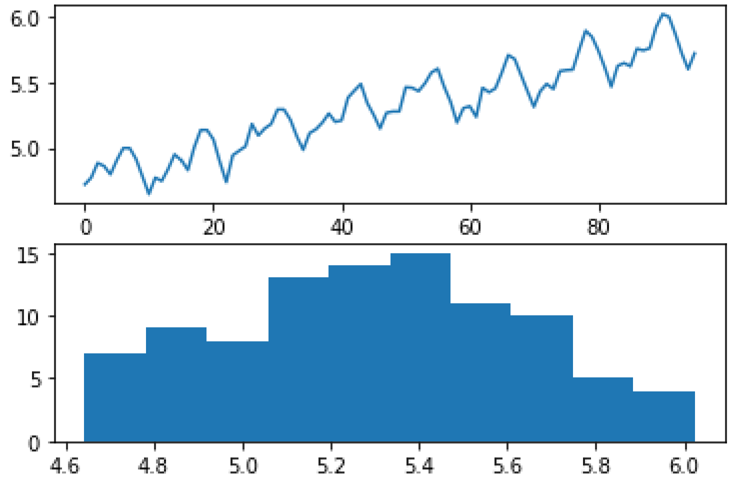
```
# line plot
pyplot.subplot(211)
pyplot.plot(dataframe['passengers'])
# histogram
pyplot.subplot(212)
pyplot.hist(dataframe['passengers'])
pyplot.show()
```




## Log Transform

In [146]:

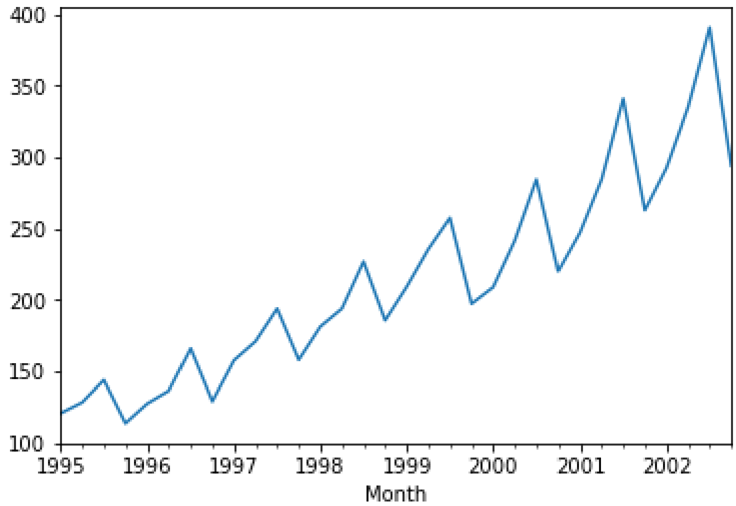
```
from numpy import log
dataframe = DataFrame(df1.values)
dataframe.columns = ['passengers']
dataframe['passengers'] = log(dataframe['passengers'])

# line plot
pyplot.subplot(211)
pyplot.plot(dataframe['passengers'])
# histogram
pyplot.subplot(212)
pyplot.hist(dataframe['passengers'])
pyplot.show()
```




In [147]:

```
print(quarterly_mean_sales.head())
quarterly_mean_sales.plot()
pyplot.show()
```

```
Month
1995-03-31    120.666667
1995-06-30    128.333333
1995-09-30    144.000000
1995-12-31    113.666667
1996-03-31    127.333333
Freq: Q-DEC, Name: Passengers, dtype: float64
```



# Forecasting_Model_Based_Methods

In [211]:

```
airline=pd.read_excel("C:/Users/LENOVO/Documents/Custom Office Templates/Airlines+Data.xlsx
airline.head()
```

Out[211]:

| | Month | Passengers |
|---|---|---|
| **0** | 1995-01-01 | 112 |
| **1** | 1995-02-01 | 118 |
| **2** | 1995-03-01 | 132 |
| **3** | 1995-04-01 | 129 |
| **4** | 1995-05-01 | 121 |

In [212]:

```
airline['Date']= pd.to_datetime(airline.Month,format='%b-%y')
airline['Months']= airline.Date.dt.strftime('%b')
airline['Year'] = airline.Date.dt.strftime('%Y')
```

In [213]:

```
airline
```

Out[213]:

| | Month | Passengers | Date | Months | Year |
|---|---|---|---|---|---|
| **0** | 1995-01-01 | 112 | 1995-01-01 | Jan | 1995 |
| **1** | 1995-02-01 | 118 | 1995-02-01 | Feb | 1995 |
| **2** | 1995-03-01 | 132 | 1995-03-01 | Mar | 1995 |
| **3** | 1995-04-01 | 129 | 1995-04-01 | Apr | 1995 |
| **4** | 1995-05-01 | 121 | 1995-05-01 | May | 1995 |
| **...** | ... | ... | ... | ... | ... |
| **91** | 2002-08-01 | 405 | 2002-08-01 | Aug | 2002 |
| **92** | 2002-09-01 | 355 | 2002-09-01 | Sep | 2002 |
| **93** | 2002-10-01 | 306 | 2002-10-01 | Oct | 2002 |
| **94** | 2002-11-01 | 271 | 2002-11-01 | Nov | 2002 |
| **95** | 2002-12-01 | 306 | 2002-12-01 | Dec | 2002 |

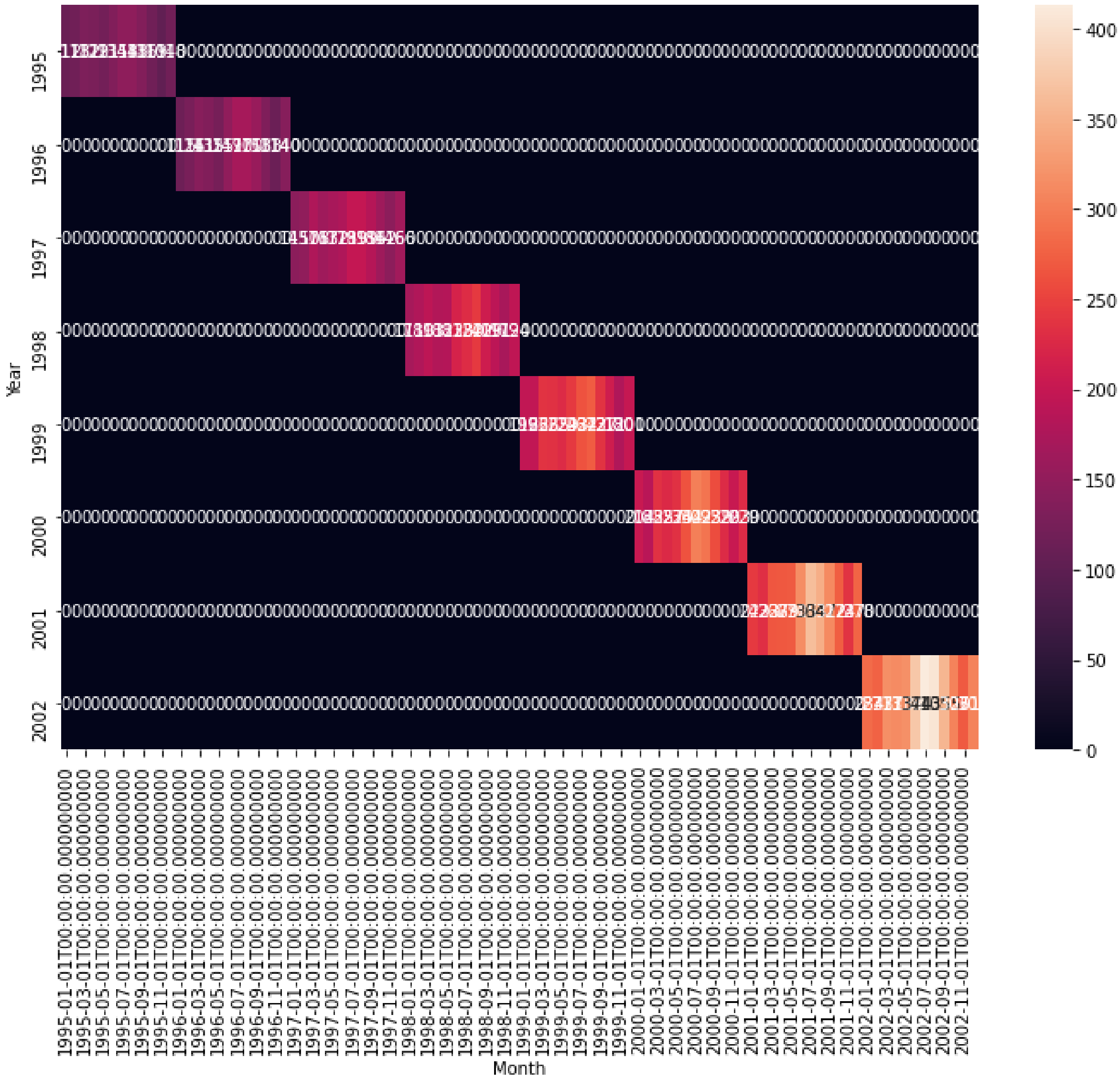
96 rows × 5 columns

In [214]:

```
# Heatmap
plt.figure(figsize=(12,8))
heatmap_y_month = pd.pivot_table(data=airline,values='Passengers',index='Year',columns='Mon
sns.heatmap(heatmap_y_month,annot=True,fmt='g') # fmt is format of the grid values
```

Out[214]:

```
<AxesSubplot:xlabel='Month', ylabel='Year'>
```

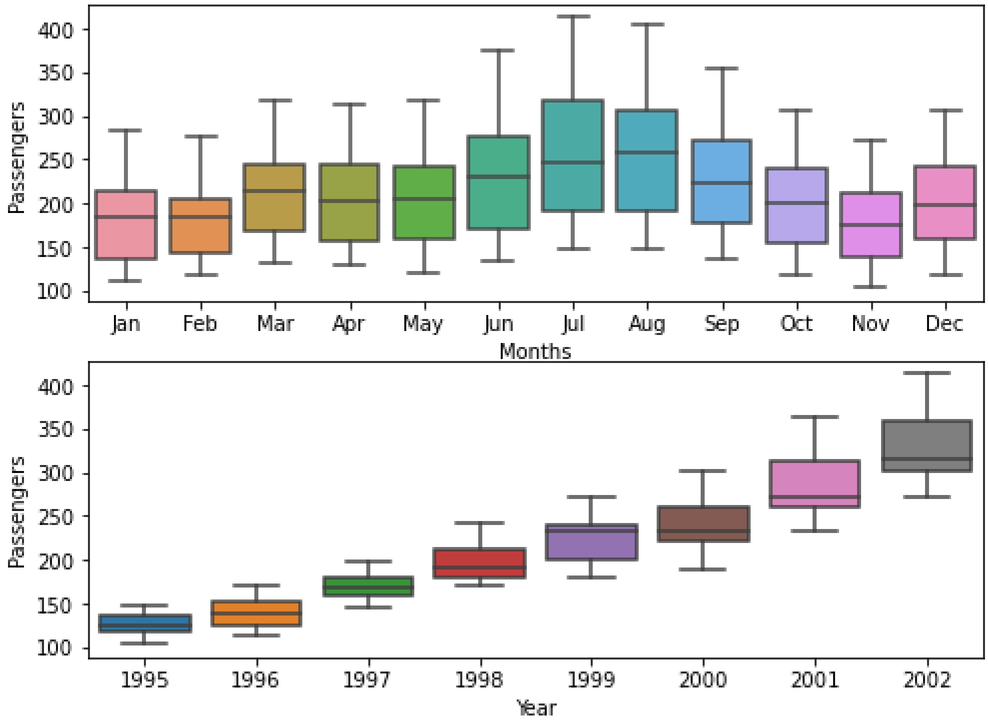


In [215]:

```
# Boxplot
plt.figure(figsize=(8,6))
plt.subplot(211)
sns.boxplot(x='Months',y='Passengers',data=airline)
plt.subplot(212)
sns.boxplot(x='Year',y='Passengers', data=airline)
```

Out[215]:

```
<AxesSubplot:xlabel='Year', ylabel='Passengers'>
```




## Preparing dummies

In [216]:

```
Month_Dummies= pd.DataFrame(pd.get_dummies(airline['Months']))
airline1 = pd.concat([airline,Month_Dummies],axis =1)
```

In [217]:

```
airline1["t"] = np.arange(1,97)
airline1["t_squared"] = airline1["t"] * airline1["t"]
airline1["Log_Passengers"] = np.log(airline1["Passengers"])
airline1
```

Out[217]:

| | Month | Passengers | Date | Months | Year | Apr | Aug | Dec | Feb | Jan | Jul | Jun | Mar | May |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1995-01-01 | 112 | 1995-01-01 | Jan | 1995 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **1** | 1995-02-01 | 118 | 1995-02-01 | Feb | 1995 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| **2** | 1995-03-01 | 132 | 1995-03-01 | Mar | 1995 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **3** | 1995-04-01 | 129 | 1995-04-01 | Apr | 1995 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **4** | 1995-05-01 | 121 | 1995-05-01 | May | 1995 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **91** | 2002-08-01 | 405 | 2002-08-01 | Aug | 2002 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **92** | 2002-09-01 | 355 | 2002-09-01 | Sep | 2002 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **93** | 2002-10-01 | 306 | 2002-10-01 | Oct | 2002 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **94** | 2002-11-01 | 271 | 2002-11-01 | Nov | 2002 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **95** | 2002-12-01 | 306 | 2002-12-01 | Dec | 2002 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

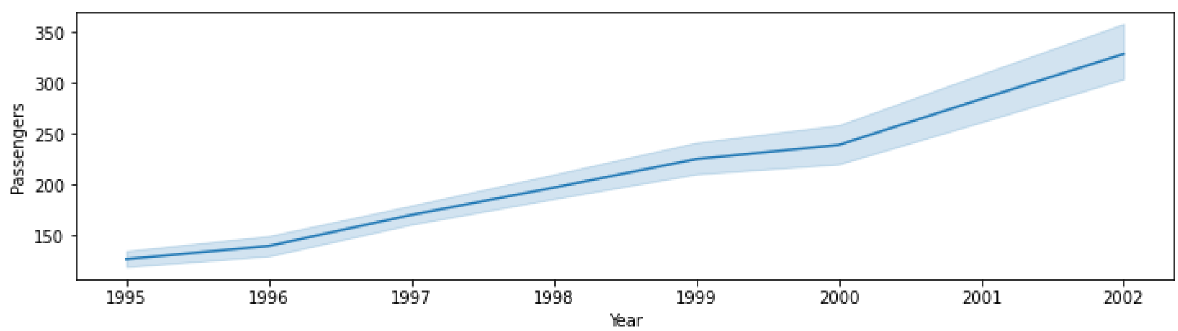
96 rows × 20 columns

In [218]:

```
plt.figure(figsize=(12,3))
sns.lineplot(x='Year', y='Passengers', data=airline)
```

Out[218]:

<AxesSubplot:xlabel='Year', ylabel='Passengers'>



In [219]:

```
# Splitting into Train & test
Train = airline1.head(80)
Test = airline1.tail(16)
```

In [220]:

```
#Linear Model
import statsmodels.formula.api as smf

linear_model = smf.ols('Passengers~t',data=Train).fit()
pred_linear =  pd.Series(linear_model.predict(pd.DataFrame(Test['t'])))
rmse_linear = np.sqrt(np.mean((np.array(Test['Passengers'])-np.array(pred_linear))**2))
rmse_linear
```

Out[220]:

```
47.54262406772677
```

In [221]:

```
#Exponential

Exp = smf.ols('Log_Passengers~t',data=Train).fit()
pred_Exp = pd.Series(Exp.predict(pd.DataFrame(Test['t'])))
rmse_Exp = np.sqrt(np.mean((np.array(Test['Passengers'])-np.array(np.exp(pred_Exp)))**2))
rmse_Exp
```

Out[221]:

```
43.79373939334308
```

In [222]:

```
#Quadratic

Quad = smf.ols('Passengers~t+t_squared',data=Train).fit()
pred_Quad = pd.Series(Quad.predict(Test[["t","t_squared"]]))
rmse_Quad = np.sqrt(np.mean((np.array(Test['Passengers'])-np.array(pred_Quad))**2))
rmse_Quad
```

Out[222]:

```
43.65440369584248
```

In [223]:

```
#Additive seasonality

add_sea = smf.ols('Passengers~Jan+Feb+Mar+Apr+May+Jun+Jul+Aug+Sep+Oct+Nov',data=Train).fit(
pred_add_sea = pd.Series(add_sea.predict(Test[['Jan','Feb','Mar','Apr','May','Jun','Jul','A
rmse_add_sea = np.sqrt(np.mean((np.array(Test['Passengers'])-np.array(pred_add_sea))**2))
rmse_add_sea
```

Out[223]:

```
129.26647641443313
```

In [224]:

```
#Additive Seasonality quadratic

add_sea_Quad = smf.ols('Passengers~t+t_squared+Jan+Feb+Mar+Apr+May+Jun+Jul+Aug+Sep+Oct+Nov'
pred_add_sea_quad = pd.Series(add_sea_Quad.predict(Test[['Jan','Feb','Mar','Apr','May','Jun
rmse_add_sea_quad = np.sqrt(np.mean((np.array(Test['Passengers'])-np.array(pred_add_sea_qua
rmse_add_sea_quad
```

Out[224]:

23.91098357010629

In [225]:

```
#Multiplicative Seasonality

Mul_sea = smf.ols('Log_Passengers~Jan+Feb+Mar+Apr+May+Jun+Jul+Aug+Sep+Oct+Nov', data=Train)
pred_Mult_sea = pd.Series(Mul_sea.predict(Test))
rmse_Mult_sea = np.sqrt(np.mean((np.array(Test['Passengers'])-np.array(np.exp(pred_Mult_sea
rmse_Mult_sea
```

Out[225]:

135.32648414621084

In [226]:

```
#Multiplicative Additive Seasonality

Mul_Add_sea = smf.ols('Log_Passengers~t+Jan+Feb+Mar+Apr+May+Jun+Jul+Aug+Sep+Oct+Nov',data=T
pred_Mult_add_sea = pd.Series(Mul_Add_sea.predict(Test))
rmse_Mult_add_sea = np.sqrt(np.mean((np.array(Test['Passengers'])-np.array(np.exp(pred_Mult
rmse_Mult_add_sea
```

Out[226]:

9.469000230305973

In [227]:

```
#Tabulating the rmse values
data= {'Model':pd.Series(['rmse_linear','rmse_Exp','rmse_Quad','rmse_add_sea','rmse_add_sea
table_rmse = pd.DataFrame(data)
table_rmse.sort_values(['RMSE_Values'])
```

Out[227]:

| | Model | RMSE_Values |
|---|---|---|
| 6 | rmse_Mult_add_sea | 9.469000 |
| 4 | rmse_add_sea_quad | 23.910984 |
| 2 | rmse_Quad | 43.654404 |
| 1 | rmse_Exp | 43.793739 |
| 0 | rmse_linear | 47.542624 |
| 3 | rmse_add_sea | 129.266476 |
| 5 | rmse_Mult_sea | 135.326484 |

## Conclusion:- From the above rmse values (rmse_Mult_ADD_sea - 9.469 ) is the best fit model

In [228]:

```
# Forecasting using Multiplicative Additive Seasonality Model
# Forecasting for next 12 months

data = [['2003-01-01','Jan'],['2003-02-01','Feb'],['2003-03-01','Mar'],['2003-04-01','Apr']
# Print(data)
forecast = pd.DataFrame(data,columns = ['Date','Months'])
forecast
```

Out[228]:

| | Date | Months |
|---|---|---|
| 0 | 2003-01-01 | Jan |
| 1 | 2003-02-01 | Feb |
| 2 | 2003-03-01 | Mar |
| 3 | 2003-04-01 | Apr |
| 4 | 2003-05-01 | May |
| 5 | 2003-06-01 | Jun |
| 6 | 2003-07-01 | Jul |
| 7 | 2003-08-01 | Aug |
| 8 | 2003-09-01 | Sep |
| 9 | 2003-10-01 | Oct |
| 10 | 2003-11-01 | Nov |
| 11 | 2003-12-01 | Dec |

In [229]:

```
# Create dummies and T and T-Squared columns

dummies = pd.DataFrame(pd.get_dummies(forecast['Months']))
forecast1 = pd.concat([forecast, dummies], axis =1)
print('After dummy\n',forecast1.head())

forecast1['t'] = np.arange(1,13)
forecast1['t_squared'] = forecast1['t'] * forecast1['t']
print('\nAfter T and T-Squared\n', forecast1.head())
```

```
After dummy
          Date Months  Apr  Aug  Dec  Feb  Jan  Jul  Jun  Mar  May  Nov  Oct  
\
0 2003-01-01    Jan    0    0    0    0    1    0    0    0    0    0    0   
1 2003-02-01    Feb    0    0    0    1    0    0    0    0    0    0    0   
2 2003-03-01    Mar    0    0    0    0    0    0    0    1    0    0    0   
3 2003-04-01    Apr    1    0    0    0    0    0    0    0    0    0    0   
4 2003-05-01    May    0    0    0    0    0    0    0    0    1    0    0   

   Sep  
0    0  
1    0  
2    0  
3    0  
4    0  

After T and T-Squared
          Date Months  Apr  Aug  Dec  Feb  Jan  Jul  Jun  Mar  May  Nov  Oct  
\
0 2003-01-01    Jan    0    0    0    0    1    0    0    0    0    0    0   
1 2003-02-01    Feb    0    0    0    1    0    0    0    0    0    0    0   
2 2003-03-01    Mar    0    0    0    0    0    0    0    1    0    0    0   
3 2003-04-01    Apr    1    0    0    0    0    0    0    0    0    0    0   
4 2003-05-01    May    0    0    0    0    0    0    0    0    1    0    0   

   Sep  t  t_squared  
0    0  1          1  
1    0  2          4  
2    0  3          9  
3    0  4         16  
4    0  5         25  
```

In [230]:

```
# Forecasting using Multiplicative Additive Seasonality Model

model_full = smf.ols('Log_Passengers~t+Jan+Feb+Mar+Apr+May+Jun+Jul+Aug+Sep+Oct+Nov+Dec',dat
pred_new  = pd.Series(model_full.predict(forecast1))
pred_new

forecast1["Forecasted_log"] = pd.Series(pred_new)
forecast1['Forecasted_Passengers'] = np.exp(forecast1['Forecasted_log'])
```

In [231]:

```
# Final Prediction

Final_predict = forecast1.loc[:, ['Months','Forecasted_Passengers']]
Final_predict
```

Out[231]:

| | Months | Forecasted_Passengers |
|---|---|---|
| **0** | Jan | 109.176148 |
| **1** | Feb | 110.331245 |
| **2** | Mar | 127.315234 |
| **3** | Apr | 123.200587 |
| **4** | May | 122.399578 |
| **5** | Jun | 138.536397 |
| **6** | Jul | 154.066959 |
| **7** | Aug | 153.741209 |
| **8** | Sep | 137.693733 |
| **9** | Oct | 120.894736 |
| **10** | Nov | 106.109309 |
| **11** | Dec | 121.633998 |

In [ ]:

```

```

# Forecasting_Data_Driven_Model

In [232]:

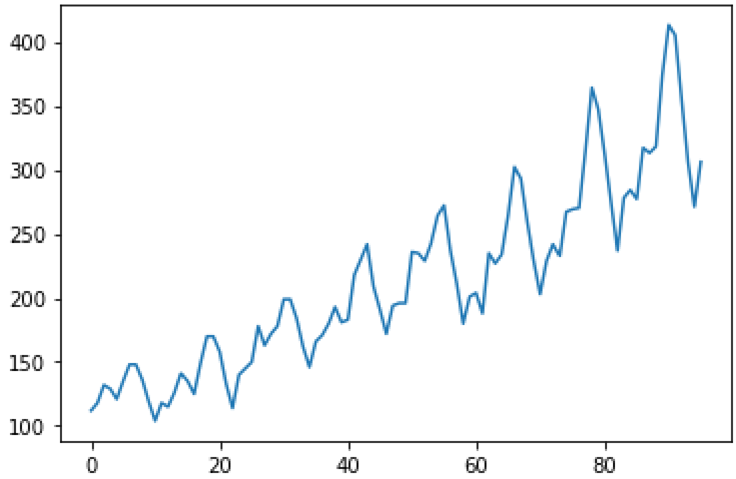
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.holtwinters import SimpleExpSmoothing # SES
from statsmodels.tsa.holtwinters import Holt # Holts Exponential Smoothing
from statsmodels.tsa.holtwinters import ExponentialSmoothing
```

In [233]:

```
airline2 = pd.read_excel("C:/Users/LENOVO/Documents/Custom Office Templates/Airlines+Data.x
airline2.Passengers.plot()
```

Out[233]:

<AxesSubplot:>




In [234]:

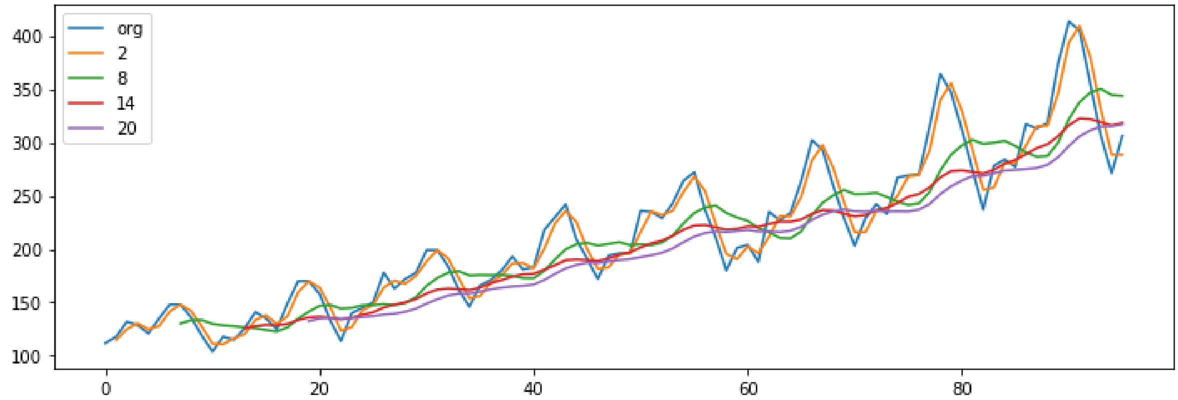
```
#Splitting data
Train = airline2.head(180)
Test = airline2.tail(16)
```

In [235]:

```
#Moving Average
plt.figure(figsize=(12,4))
airline2.Passengers.plot(label="org")
for i in range(2,24,6):
    airline2["Passengers"].rolling(i).mean().plot(label=str(i))
plt.legend(loc='best')
```

Out[235]:

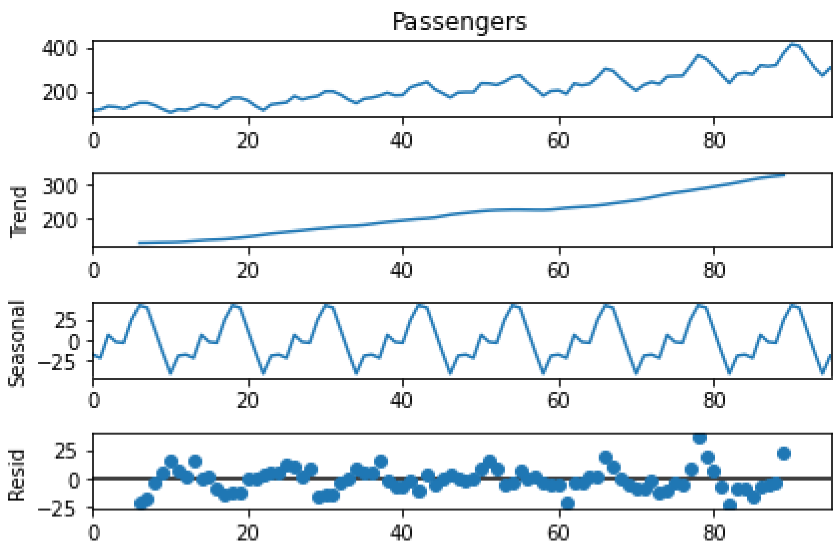
```
<matplotlib.legend.Legend at 0x3db2b3490>
```




# From the above diagram we can choose window size = 2

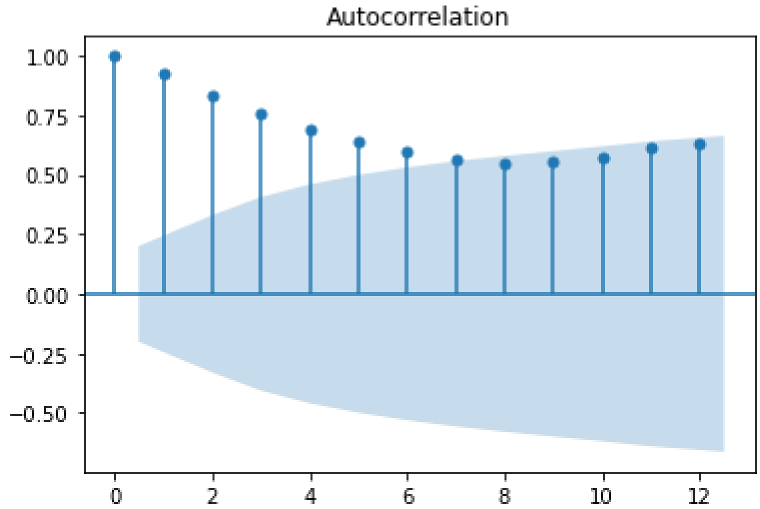
In [236]:

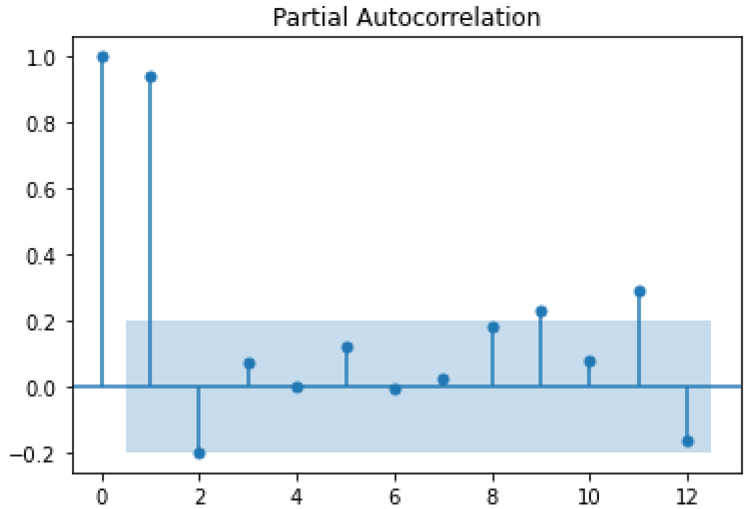
```
#Time series decomposition plot

decompose_ts_add = seasonal_decompose(airline2.Passengers,period=12)
decompose_ts_add.plot()
plt.show()
```



In [237]:

```
#ACF plots and PACF plots

import statsmodels.graphics.tsaplots as tsa_plots
tsa_plots.plot_acf(airline2.Passengers,lags=12)
tsa_plots.plot_pacf(airline2.Passengers,lags=12)
plt.show()
```



In [238]:

```
#Evaluation Metric MAPE

def MAPE(pred,org):
    temp = np.abs((pred-org)/org)*100
    return np.mean(temp)
```

In [239]:

```
def rmse(pred):
    rmse = np.sqrt(np.mean((np.array(Test['Passengers'])- np.array(pred))**2))
    return rmse
```

In [240]:

```
# Simple Exponential Method
import warnings
ses_model = SimpleExpSmoothing(Train['Passengers']).fit(smoothing_level = 0.2)
pred_ses = ses_model.predict(start = Test.index[0], end = Test.index[-1])
print('MAPE Value for the Simple Exponential Model is:',MAPE(pred_ses, Test['Passengers']))
print('rmse value for the model is:',rmse(pred_ses))
```

```
MAPE Value for the Simple Exponential Model is: 11.286601276416397
rmse value for the model is: 46.57374121940332

C:\Users\LENOVO\anaconda3\lib\site-packages\statsmodels\tsa\holtwinters\mode
l.py:427: FutureWarning: After 0.13 initialization must be handled at model
creation
  warnings.warn(
```

In [241]:

```
# Holt's Method
hw_model = Holt(Train['Passengers']).fit(smoothing_level = 0.8, smoothing_slope = 0.2)
pred_hw = hw_model.predict(start = Test.index[0], end = Test.index[-1])
print('The MAPE value for the Holt model is:', MAPE(pred_hw, Test['Passengers']))
print('The rmse value for the model is:', rmse(pred_hw))
```

```
The MAPE value for the Holt model is: 11.842736531096246
The rmse value for the model is: 42.383941460162916

<ipython-input-241-ef8a5143678d>:2: FutureWarning: the 'smoothing_slope'' ke
yword is deprecated, use 'smoothing_trend' instead
  hw_model = Holt(Train['Passengers']).fit(smoothing_level = 0.8, smoothing_
slope = 0.2)
```

In [242]:

```
# Holt's Winter exponential smoothing with addaptive seasonality and additive trend
hw_model_add_add = ExponentialSmoothing(Train['Passengers'], seasonal = 'add', trend = 'add
pred_hw_add_add = hw_model_add_add.predict(start = Test.index[0], end= Test.index[-1])
print('The MAPE value for the model is:', MAPE(pred_hw_add_add, Test['Passengers']))
print('The rmse value for the model is:', rmse(pred_hw_add_add))
```

```
The MAPE value for the model is: 2.2389155192101486
The rmse value for the model is: 9.061400169728056
```

In [243]:

```
# Holt's WInter Exponential Smoothing with Multiplicative Seasonality and Additive Trend
hw_model_mul_add = ExponentialSmoothing(Train['Passengers'], seasonal = 'mul', trend = 'add
pred_hw_mul_add = hw_model_mul_add.predict(start = Test.index[0], end = Test.index[-1])
print('The MAPE value for the model is :', MAPE(pred_hw_mul_add, Test['Passengers']))
print('The rmse value for the model is :', rmse(pred_hw_mul_add))
```

```
The MAPE value for the model is : 1.213969628181401
The rmse value for the model is : 4.999706408194285
```

In [244]:

```
data = {'MODEL': pd.Series(['MAPE_Simple_Exponential', 'MAPE_Holts', 'MAPE_holts_winter_add
        "MAPE_values":pd.Series([MAPE(pred_ses,Test.Passengers), MAPE(pred_hw,Test.Passenge
table_rmse = pd.DataFrame(data)
table_rmse.sort_values(['MAPE_values'])
```

Out[244]:

| | MODEL | MAPE_values |
|---|---|---|
| **3** | MAPE_holts_winter_mul_add | 1.213970 |
| **2** | MAPE_holts_winter_add_add | 2.238916 |
| **0** | MAPE_Simple_Exponential | 11.286601 |
| **1** | MAPE_Holts | 11.842737 |

### From the above table we can say that Holt's Winter Exponential Smoothing with multiplicative seasonality and trend model is best suitable for the Airlines Data

## Final Model by combining train and test

In [245]:

```
hwe_model_mul_add = ExponentialSmoothing(airline2['Passengers'], seasonal = 'mul', trend =
```

```
C:\Users\LENOVO\anaconda3\lib\site-packages\statsmodels\tsa\holtwinters\mode
l.py:427: FutureWarning: After 0.13 initialization must be handled at model
creation
  warnings.warn(
```

In [247]:

```
#Forecasting for next 10 time periods
hwe_model_mul_add.forecast(10)
```

Out[247]:

```
96     312.899164
97     308.170903
98     355.533272
99     345.770384
100    345.697110
101    392.472018
102    436.501550
103    429.860620
104    380.172862
105    332.318642
dtype: float64
```

In [ ]:

```

```

## Forecasting Model_Arima

In [248]:

```
# Import libraries
from pandas import read_csv
from matplotlib import pyplot
from numpy import sqrt
import warnings
import itertools
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
```

In [250]:

```
= pd.read_excel("C:/Users/LENOVO/Documents/Custom Office Templates/Airlines+Data.xlsx", hea
```

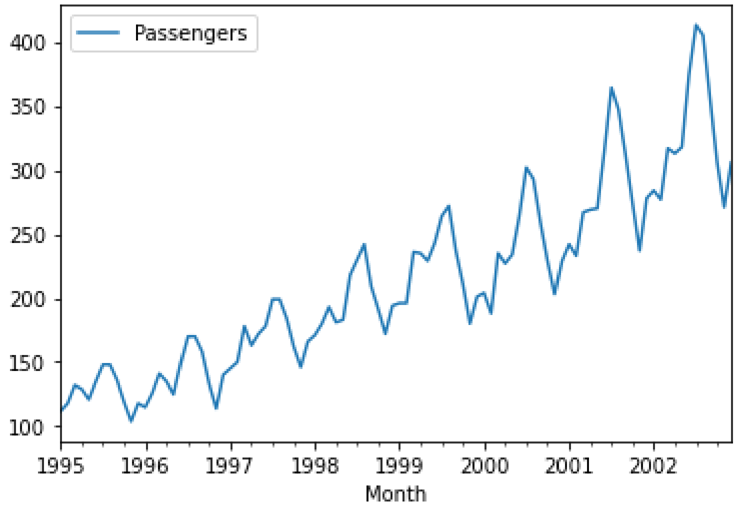
Out[250]:

| | Passengers |
|---|---|
| **Month** | |
| **1995-01-01** | 112 |
| **1995-02-01** | 118 |
| **1995-03-01** | 132 |
| **1995-04-01** | 129 |
| **1995-05-01** | 121 |
| **...** | ... |
| **2002-08-01** | 405 |
| **2002-09-01** | 355 |
| **2002-10-01** | 306 |
| **2002-11-01** | 271 |
| **2002-12-01** | 306 |

96 rows × 1 columns

In [251]:

```
# line plot of time series
from pandas import read_csv
from matplotlib import pyplot
airline3.plot()
pyplot.show()
```

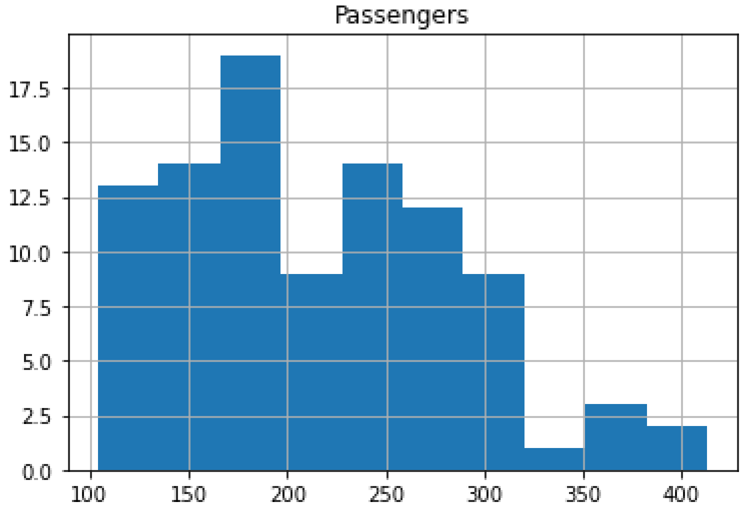


In [252]:

```
airline3.hist()
```

Out[252]:

```
array([[<AxesSubplot:title={'center':'Passengers'}>]], dtype=object)
```

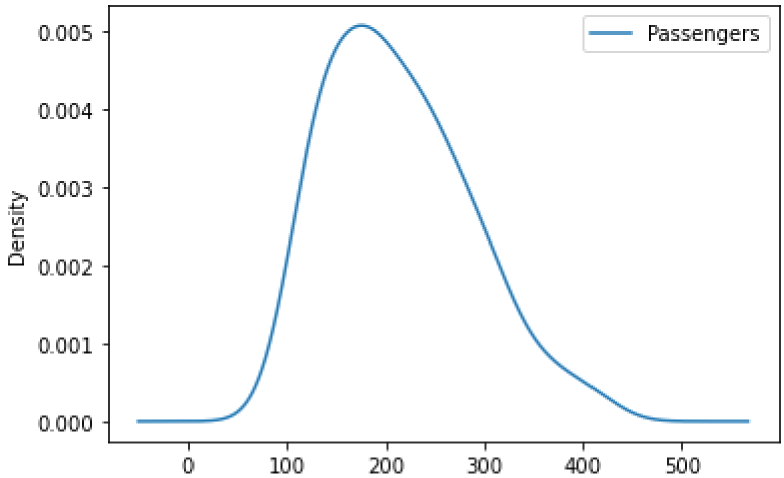



In [253]:

```
airline3.plot(kind='kde')
```

Out[253]:

```
<AxesSubplot:ylabel='Density'>
```




In [254]:

```
# separate out a validation dataset
split_point = len(airline3) - 10
dataset, validation = series[0:split_point], airline3[split_point:]
print('Dataset %d, Validation %d' % (len(dataset), len(validation)))
dataset.to_csv('dataset.csv', header=False)
validation.to_csv('validation.csv', header=False)
```

```
Dataset 86, Validation 10
```

# Persistence/ Base model

In [255]:

```
# evaluate a persistence model
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from math import sqrt
# load data
train = read_csv('dataset.csv', header=None, index_col=0, parse_dates=True, squeeze=True)
# prepare data
X = train.values
X = X.astype('float32')
train_size = int(len(X) * 0.50)
train, test = X[0:train_size], X[train_size:]
```

In [256]:

```
# walk-forward validation
history = [x for x in train]
predictions = list()
for i in range(len(test)):
    yhat = history[-1]
    predictions.append(yhat)
# observation
    obs = test[i]
    history.append(obs)
    print('>Predicted=%.3f, Expected=%.3f' % (yhat, obs))
# report performance
rmse = sqrt(mean_squared_error(test, predictions))
print('RMSE: %.3f' % rmse)
```

```
>Predicted=230.000, Expected=242.000
>Predicted=242.000, Expected=209.000
>Predicted=209.000, Expected=191.000
>Predicted=191.000, Expected=172.000
>Predicted=172.000, Expected=194.000
>Predicted=194.000, Expected=196.000
>Predicted=196.000, Expected=196.000
>Predicted=196.000, Expected=236.000
>Predicted=236.000, Expected=235.000
>Predicted=235.000, Expected=229.000
>Predicted=229.000, Expected=243.000
>Predicted=243.000, Expected=264.000
>Predicted=264.000, Expected=272.000
>Predicted=272.000, Expected=237.000
>Predicted=237.000, Expected=211.000
>Predicted=211.000, Expected=180.000
>Predicted=180.000, Expected=201.000
>Predicted=201.000, Expected=204.000
>Predicted=204.000, Expected=188.000
>Predicted=188.000, Expected=235.000
>Predicted=235.000, Expected=227.000
>Predicted=227.000, Expected=234.000
>Predicted=234.000, Expected=264.000
>Predicted=264.000, Expected=302.000
>Predicted=302.000, Expected=293.000
>Predicted=293.000, Expected=259.000
>Predicted=259.000, Expected=229.000
>Predicted=229.000, Expected=203.000
>Predicted=203.000, Expected=229.000
>Predicted=229.000, Expected=242.000
>Predicted=242.000, Expected=233.000
>Predicted=233.000, Expected=267.000
>Predicted=267.000, Expected=269.000
>Predicted=269.000, Expected=270.000
>Predicted=270.000, Expected=315.000
>Predicted=315.000, Expected=364.000
>Predicted=364.000, Expected=347.000
>Predicted=347.000, Expected=312.000
>Predicted=312.000, Expected=274.000
>Predicted=274.000, Expected=237.000
>Predicted=237.000, Expected=278.000
>Predicted=278.000, Expected=284.000
>Predicted=284.000, Expected=277.000
RMSE: 25.698
```

## ARIMA Hyperparameters

In [257]:

```
# grid search ARIMA parameters for a time series

import warnings
from pandas import read_csv
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
from math import sqrt


# evaluate an ARIMA model for a given order (p,d,q) and return RMSE
def evaluate_arima_model(X, arima_order):
# prepare training dataset
    X = X.astype('float32')
    train_size = int(len(X) * 0.50)
    train, test = X[0:train_size], X[train_size:]
    history = [x for x in train]
# make predictions
    predictions = list()
    for t in range(len(test)):
        model = ARIMA(history, order=arima_order)
# model_fit = model.fit(disp=0)
        model_fit = model.fit(disp=0)
        yhat = model_fit.forecast()[0]
        predictions.append(yhat)
        history.append(test[t])
# calculate out of sample error
    rmse = sqrt(mean_squared_error(test, predictions))
    return rmse
```

## Grid search for p,d,q values

In [258]:

```
# evaluate combinations of p, d and q values for an ARIMA model
def evaluate_models(dataset, p_values, d_values, q_values):
    dataset = dataset.astype('float32')
    best_score, best_cfg = float('inf'), None
    for p in p_values:
        for d in d_values:
            for q in q_values:
                order = (p,d,q)
                try:
                    rmse = evaluate_arima_model(train, order)
                    if rmse < best_score:
                        best_score, best_cfg = rmse, order
                    print('ARIMA%s RMSE=%.3f' % (order,rmse))
                except:
                    continue
    print('Best ARIMA%s RMSE=%.3f' % (best_cfg, best_score))
```

In [259]:

```
# load dataset
train = read_csv('dataset.csv', header=None, index_col=0, parse_dates=True, squeeze=True)
#evaluate parameters
p_values = range(0, 5)
d_values = range(0, 5)
q_values = range(0, 5)
warnings.filterwarnings("ignore")
evaluate_models(train.values, p_values, d_values, q_values)
```

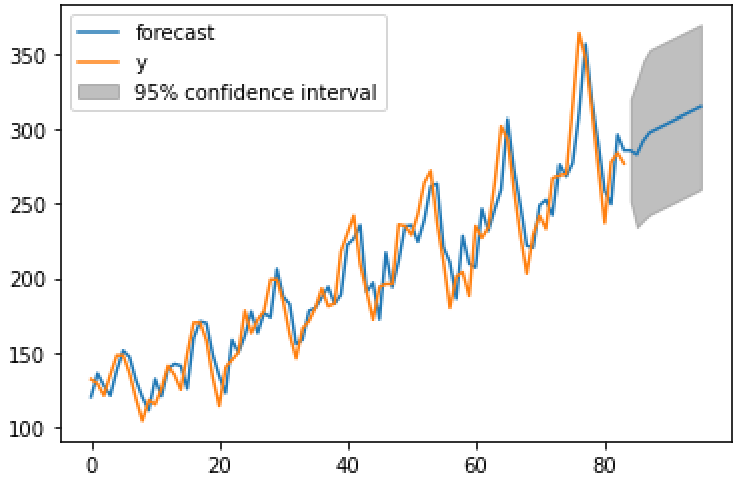
```
ARIMA(0, 0, 0) RMSE=78.563
ARIMA(0, 0, 1) RMSE=44.789
ARIMA(0, 1, 0) RMSE=25.903
ARIMA(0, 1, 1) RMSE=25.355
ARIMA(0, 1, 2) RMSE=27.772
ARIMA(0, 1, 3) RMSE=23.806
ARIMA(0, 1, 4) RMSE=22.640
ARIMA(0, 2, 0) RMSE=32.474
ARIMA(0, 2, 1) RMSE=26.640
ARIMA(0, 2, 2) RMSE=25.942
ARIMA(0, 2, 3) RMSE=27.914
ARIMA(0, 2, 4) RMSE=25.151
ARIMA(1, 0, 0) RMSE=26.036
ARIMA(1, 0, 1) RMSE=25.282
ARIMA(1, 0, 2) RMSE=455.575
ARIMA(1, 1, 0) RMSE=25.679
ARIMA(1, 2, 0) RMSE=31.603
ARIMA(2, 0, 0) RMSE=25.620
ARIMA(2, 1, 0) RMSE=25.467
ARIMA(2, 2, 0) RMSE=30.414
ARIMA(3, 0, 0) RMSE=25.510
ARIMA(3, 0, 1) RMSE=24.907
ARIMA(3, 1, 0) RMSE=25.648
ARIMA(3, 2, 0) RMSE=30.597
ARIMA(4, 0, 0) RMSE=25.764
ARIMA(4, 1, 0) RMSE=25.344
ARIMA(4, 2, 0) RMSE=29.205
ARIMA(4, 2, 1) RMSE=26.016
Best ARIMA(0, 1, 4) RMSE=22.640
```

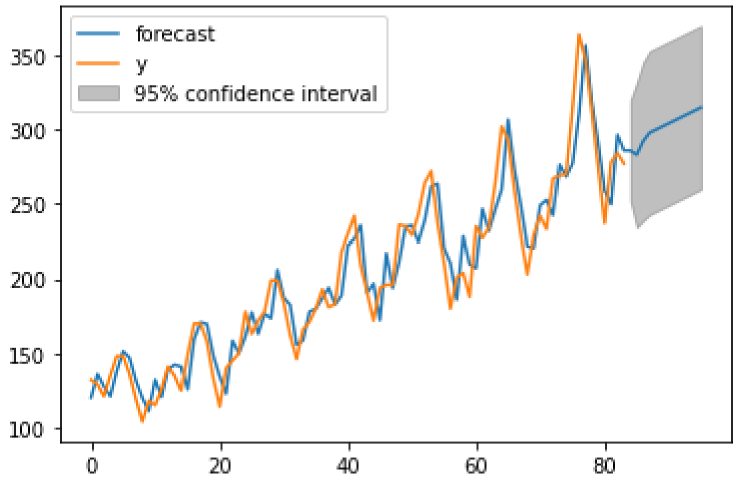
# Build Model based on the optimized values

In [260]:

```
# save finalized model to file
from pandas import read_csv
from statsmodels.tsa.arima_model import ARIMA
import numpy

# load data
train = read_csv('dataset.csv', header=0, index_col=0, parse_dates=True)
# prepare data
X = train.values
X = X.astype('float32')

# fit model
model = ARIMA(X, order=(0,1,4))
model_fit = model.fit()
forecast=model_fit.forecast(steps=10)[0]
model_fit.plot_predict(1, 96)
```

Out[260]:



In [261]:

```
#Error on the test data
val=pd.read_csv('validation.csv',header=None)
rmse = sqrt(mean_squared_error(val[1], forecast))
rmse
```

Out[261]:

```
59.81126724582207
```

## Combine train and test data and build final model

In [264]:

```
# fit model
df = pd.read_excel('C:/Users/LENOVO/Documents/Custom Office Templates/Airlines+Data.xlsx')
df1= df.set_index('Month')

# prepare data
X = df1.values
X = X.astype('float32')
```

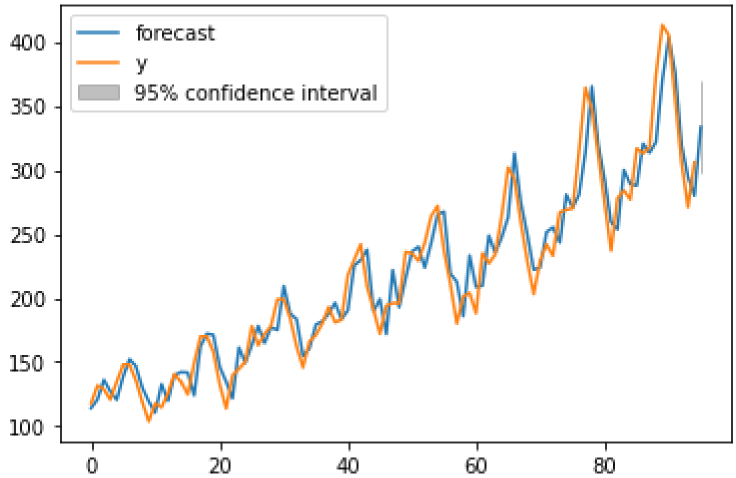
In [265]:

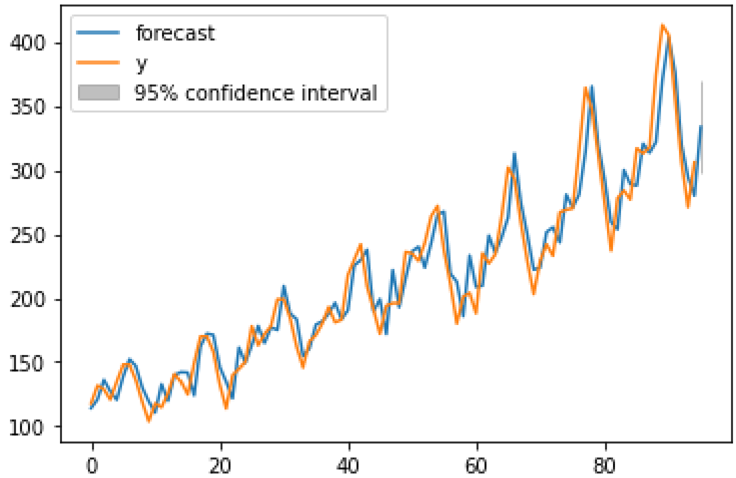
```
model = ARIMA(X, order=(0,1,4))
model_fit = model.fit()
```

In [266]:

```
forecast=model_fit.forecast(steps=10)[0]
model_fit.plot_predict(1,96)
```

Out[266]:




In [267]:

```
forecast
```

Out[267]:

```
array([333.64541868, 338.10561738, 344.47640726, 334.99091697,
       337.34157319, 339.69222941, 342.04288563, 344.39354185,
       346.74419807, 349.09485429])
```

## From the above all three models we can say that Holt's Winter Exponential Smoothing with multiplicative seasonality and trend model is best suitable for the Airlines Data (Forecasting_Data_Driven_Models)

In [ ]: