Importing necessary liabrary

In [57]:

```
import warnings
warnings.filterwarnings("ignore")
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
from statsmodels.tsa.holtwinters import Holt
from statsmodels.tsa.holtwinters import ExponentialSmoothing
import statsmodels.graphics.tsaplots as tsa_plots
import statsmodels.tsa.statespace as tm_models
from datetime import datetime,time
import itertools
plt.style.use('fivethirtyeight')
from pylab import rcParams
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
import statsmodels.formula.api as smf
from matplotlib import pyplot
```

Business Problem.

Forecast the Airlines Passengers data set. Prepare a document for each model explaining ,how many dummy variables you have created and RMSE value for each model. Finally which model you will use for Forecasting.

importing dataset

In [2]:

```
airlines_data=pd.read_excel("Airlines+Data.xlsx")
airlines_data
```

Out[2]:

	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

Performing initial analysis

In [3]:

```
airlines_data.isna().sum()
```

Out[3]:

```
Month         0
Passengers    0
dtype: int64
```

In [4]:

```
airlines_data.shape
```

Out[4]:

```
(96, 2)
```

In [5]:

```
airlines_data.describe().T
```

Out[5]:

	count	mean	std	min	25%	50%	75%	max
Passengers	96.0	213.708333	71.918216	104.0	156.0	200.0	264.75	413.0

In [6]:

```
airlines_data.dtypes
```

Out[6]:

```
Month         datetime64[ns]
Passengers             int64
dtype: object
```

In [7]:

```
airlines_data = airlines_data.set_index('Month')
```

In [8]:

```
airlines_data
```

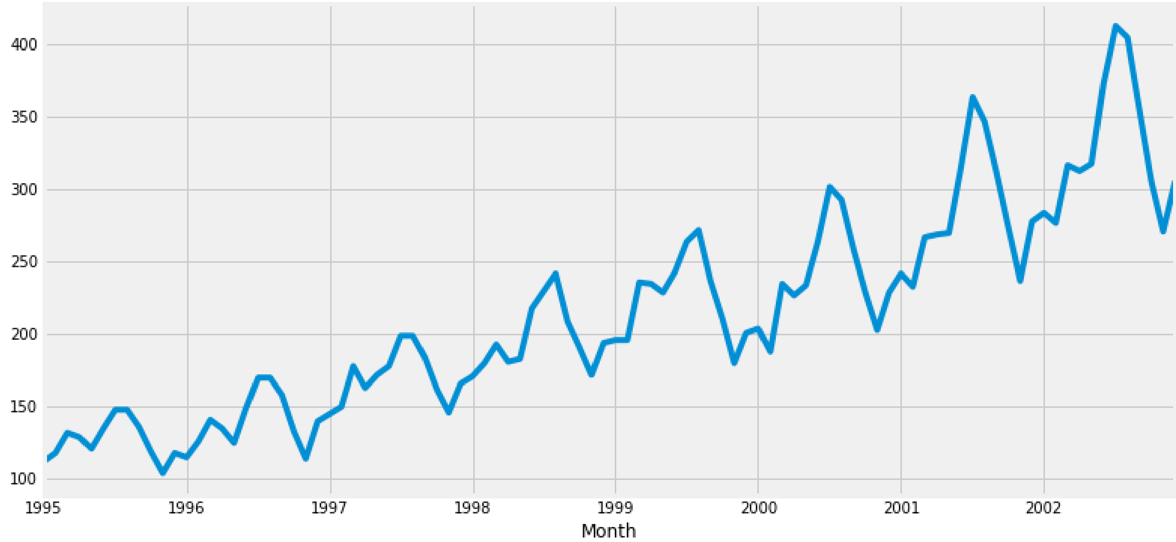
Out[8]:

	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 [9]:

```
airlines_data['Passengers'].plot(figsize=(12, 6))
plt.show()
```

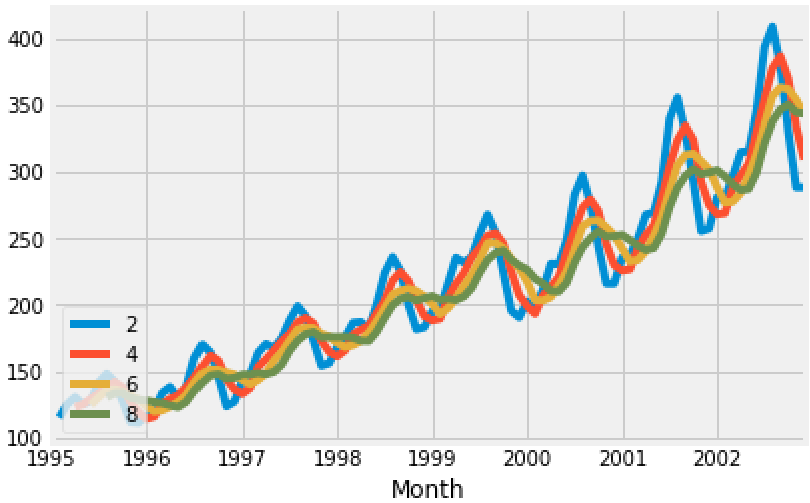


In [10]:

```
for i in range(2,10,2):
    airlines_data['Passengers'].rolling(i).mean().plot(label=str(i))
plt.legend(loc=3)
```

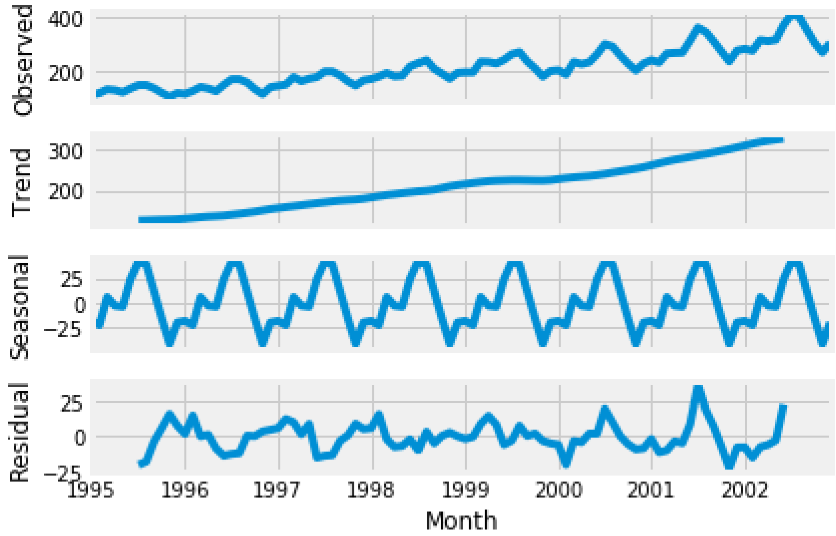
Out[10]:

```
<matplotlib.legend.Legend at 0x1c910d141c8>
```



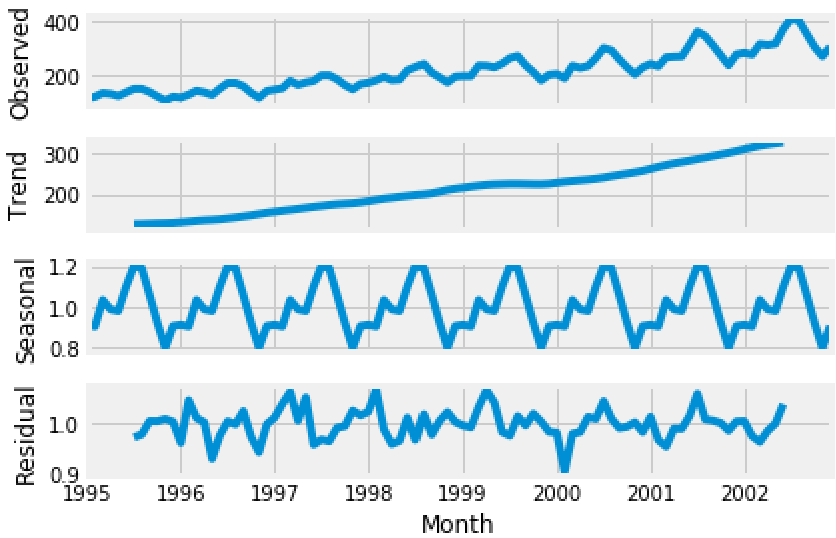
In [11]:

```
add_ts = seasonal_decompose(airlines_data['Passengers'],model="additive")
fig = add_ts.plot()
plt.show()
```



In [12]:

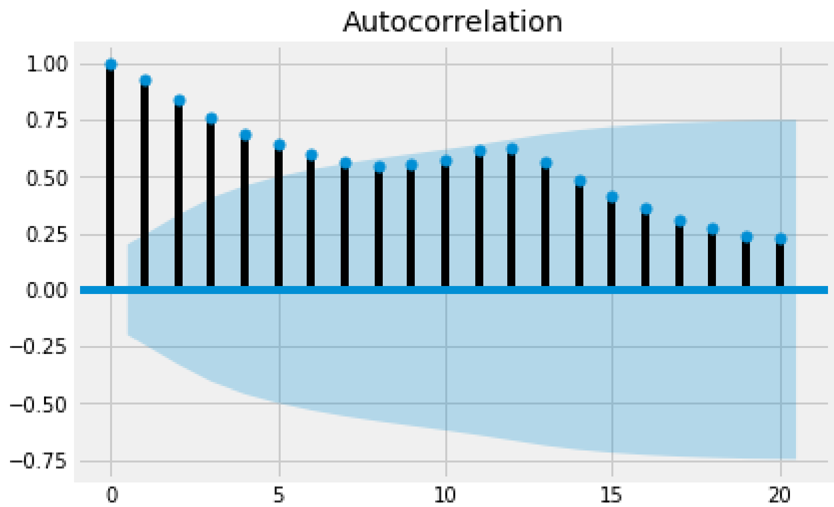
```
mul_ts = seasonal_decompose(airlines_data.Passengers,model="multiplicative")
fig = mul_ts.plot()
plt.show()
```

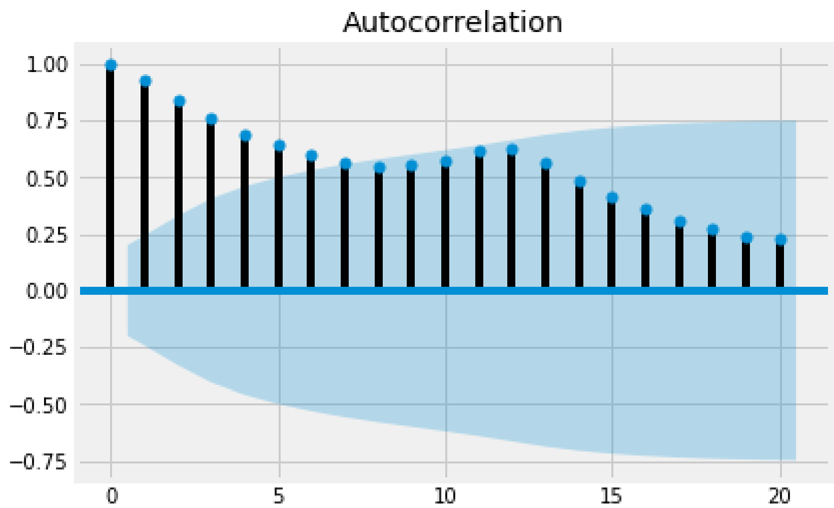


In [13]:

```
tsa_plots.plot_acf(airlines_data['Passengers'])
```

Out[13]:





Building the Time series forecasting using ARIMA.

In [14]:

```
X = airlines_data['Passengers'].values
X
```

Out[14]:

```
array([112, 118, 132, 129, 121, 135, 148, 148, 136, 119, 104, 118, 115,
       126, 141, 135, 125, 149, 170, 170, 158, 133, 114, 140, 145, 150,
       178, 163, 172, 178, 199, 199, 184, 162, 146, 166, 171, 180, 193,
       181, 183, 218, 230, 242, 209, 191, 172, 194, 196, 196, 236, 235,
       229, 243, 264, 272, 237, 211, 180, 201, 204, 188, 235, 227, 234,
       264, 302, 293, 259, 229, 203, 229, 242, 233, 267, 269, 270, 315,
       364, 347, 312, 274, 237, 278, 284, 277, 317, 313, 318, 374, 413,
       405, 355, 306, 271, 306], dtype=int64)
```

In [15]:

```
size = int(len(X) * 0.67)
size
```

Out[15]:

```
64
```

In [16]:

```
train, test = X[0:size], X[size:len(X)]
```

In [17]:

```
train
```

Out[17]:

```
array([112, 118, 132, 129, 121, 135, 148, 148, 136, 119, 104, 118, 115,
       126, 141, 135, 125, 149, 170, 170, 158, 133, 114, 140, 145, 150,
       178, 163, 172, 178, 199, 199, 184, 162, 146, 166, 171, 180, 193,
       181, 183, 218, 230, 242, 209, 191, 172, 194, 196, 196, 236, 235,
       229, 243, 264, 272, 237, 211, 180, 201, 204, 188, 235, 227],
      dtype=int64)
```

In [18]:

```
test
```

Out[18]:

```
array([234, 264, 302, 293, 259, 229, 203, 229, 242, 233, 267, 269, 270,
       315, 364, 347, 312, 274, 237, 278, 284, 277, 317, 313, 318, 374,
       413, 405, 355, 306, 271, 306], dtype=int64)
```

In [19]:

```
model = ARIMA(train, order=(5,1,0))
```

In [20]:

```
model_fit = model.fit(disp=0)
```

In [21]:

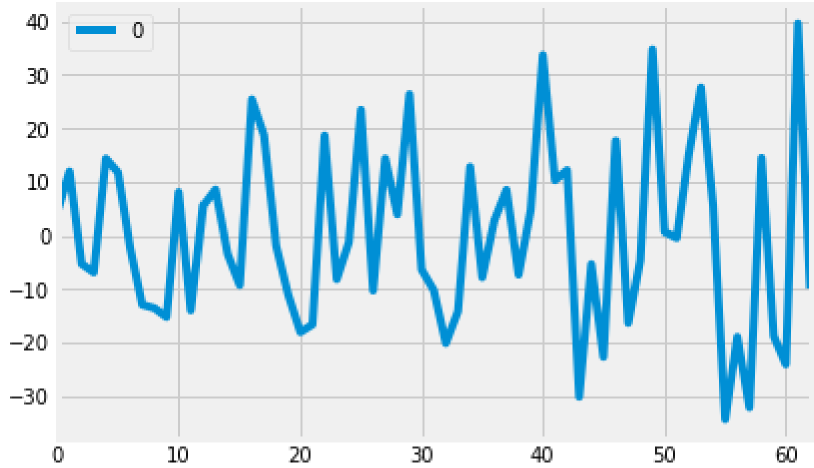
```
print(model_fit.summary())
```

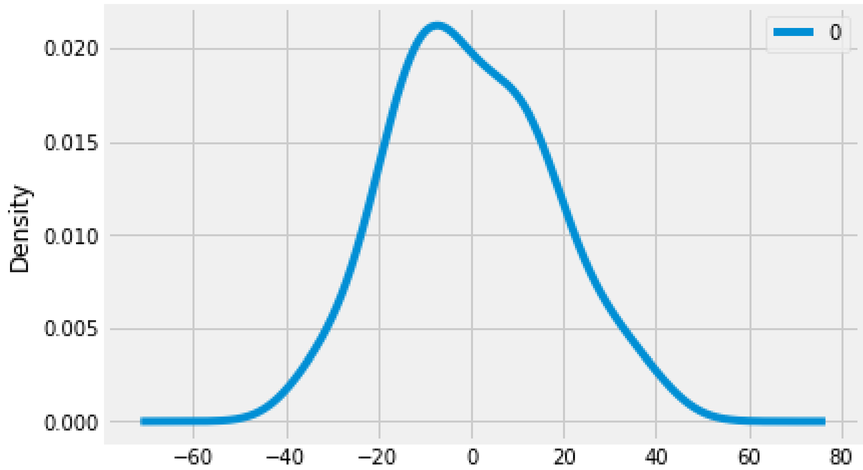
```
                             ARIMA Model Results
==============================================================================
==
Dep. Variable:                    D.y   No. Observations:
63
Model:                 ARIMA(5, 1, 0)   Log Likelihood                -266.8
69
Method:                       css-mle   S.D. of innovations             16.6
72
Date:                Fri, 24 Sep 2021   AIC                            547.7
37
Time:                        00:21:47   BIC                            562.7
39
Sample:                             1   HQIC                           553.6
38

==============================================================================
==
                 coef    std err          z      P>|z|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
const          1.6282      1.446      1.126      0.265      -1.206       4.4
62
ar.L1.D.y      0.0618      0.126      0.491      0.625      -0.185       0.3
09
ar.L2.D.y     -0.1912      0.132     -1.450      0.153      -0.450       0.0
67
ar.L3.D.y     -0.0861      0.132     -0.651      0.518      -0.345       0.1
73
ar.L4.D.y     -0.2791      0.129     -2.162      0.035      -0.532      -0.0
26
ar.L5.D.y      0.0127      0.135      0.094      0.925      -0.251       0.2
77
                                    Roots
==============================================================================
=
                  Real          Imaginary           Modulus         Frequenc
y
------------------------------------------------------------------------------
-
AR.1            0.8016          -1.0252j            1.3014           -0.144
4
AR.2            0.8016          +1.0252j            1.3014            0.144
4
AR.3           -0.9685          -1.0704j            1.4435           -0.367
1
AR.4           -0.9685          +1.0704j            1.4435            0.367
1
AR.5           22.3045          -0.0000j           22.3045           -0.000
0
------------------------------------------------------------------------------
-
```

In [22]:

```
residuals = pd.DataFrame(model_fit.resid)
residuals.plot()
pyplot.show()
residuals.plot(kind='kde')
pyplot.show()
print(residuals.describe())
```






```
               0
count  63.000000
mean    0.053576
std    16.817202
min   -34.348953
25%   -11.981893
50%    -1.847674
75%    12.079925
max    39.634564
```

Rolling Forecast ARIMA Model

In [23]:

```
history = [x for x in train]
```

In [24]:

```
predictions = list()
```

In [25]:

```
for t in range(len(test)):
    model = ARIMA(history, order=(5,1,0))
    model_fit = model.fit(disp=0)
    output = model_fit.forecast()
    ypred = output[0]
    predictions.append(ypred)
    obs = test[t]
    history.append(obs)
    print('predicted=%f, expected=%f' % (ypred, obs))
```

```
predicted=220.737312, expected=234.000000
predicted=237.815019, expected=264.000000
predicted=252.750573, expected=302.000000
predicted=306.715782, expected=293.000000
predicted=285.374653, expected=259.000000
predicted=250.264020, expected=229.000000
predicted=227.093113, expected=203.000000
predicted=211.011444, expected=229.000000
predicted=253.260276, expected=242.000000
predicted=252.490687, expected=233.000000
predicted=234.042132, expected=267.000000
predicted=268.773626, expected=269.000000
predicted=261.782251, expected=270.000000
predicted=271.798053, expected=315.000000
predicted=314.422124, expected=364.000000
predicted=368.637716, expected=347.000000
predicted=334.957866, expected=312.000000
predicted=301.161847, expected=274.000000
predicted=265.936491, expected=237.000000
predicted=244.037173, expected=278.000000
predicted=312.961794, expected=284.000000
predicted=291.748156, expected=277.000000
predicted=284.551881, expected=317.000000
predicted=316.501221, expected=313.000000
predicted=303.218126, expected=318.000000
predicted=324.834634, expected=374.000000
predicted=373.140675, expected=413.000000
predicted=415.007173, expected=405.000000
predicted=397.508422, expected=355.000000
predicted=332.087103, expected=306.000000
predicted=299.452944, expected=271.000000
predicted=279.908333, expected=306.000000
```

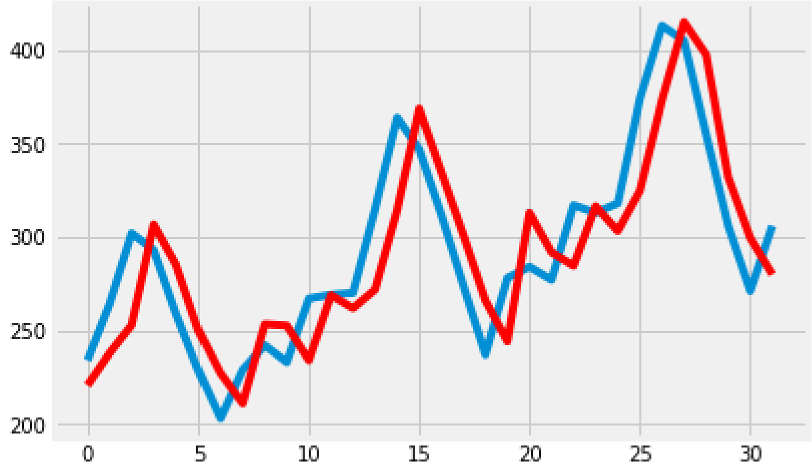
In [26]:

```
error = mean_squared_error(test, predictions)
print('Test MSE: %.3f' % error)
```

```
Test MSE: 801.863
```

In [27]:

```
pyplot.plot(test)
pyplot.plot(predictions, color='red')
pyplot.show()
```



A line plot is showing the expected values (blue) compared to the rolling forecast predictions (red). We can see the values show some trend and are in the correct scale.

Comparing the models

In [28]:

```
airlines_data1=pd.read_excel("Airlines+Data.xlsx")
airlines_data1
```

Out[28]:

	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 [29]:

```
airlines_data1 = pd.get_dummies(airlines_data1, columns = ['Month'])
```

In [30]:

```
airlines_data1
```

Out[30]:

	Passengers	Month_1995-01-01 00:00:00	Month_1995-02-01 00:00:00	Month_1995-03-01 00:00:00	Month_1995-04-01 00:00:00	Month_1995-05-01 00:00:00	Month_19 06 00:00
0	112	1	0	0	0	0	
1	118	0	1	0	0	0	
2	132	0	0	1	0	0	
3	129	0	0	0	1	0	
4	121	0	0	0	0	1	
...	...	...	...	...	...	...	
91	405	0	0	0	0	0	
92	355	0	0	0	0	0	
93	306	0	0	0	0	0	
94	271	0	0	0	0	0	
95	306	0	0	0	0	0	

96 rows × 97 columns

In [31]:

```
airlines_data1.shape
```

Out[31]:

```
(96, 97)
```

In [32]:

```
airlines_data1.columns = ['Passengers','Month_1','Month_2','Month_3','Month_4','Month_5','M
```

In [33]:

```
airlines_data1
```

Out[33]:

	Passengers	Month_1	Month_2	Month_3	Month_4	Month_5	Month_6	Month_7	Month_8
0	112	1	0	0	0	0	0	0	0
1	118	0	1	0	0	0	0	0	0
2	132	0	0	1	0	0	0	0	0
3	129	0	0	0	1	0	0	0	0
4	121	0	0	0	0	1	0	0	0
...	...	...	...	...	...	...	...	...	...
91	405	0	0	0	0	0	0	0	0
92	355	0	0	0	0	0	0	0	0
93	306	0	0	0	0	0	0	0	0
94	271	0	0	0	0	0	0	0	0
95	306	0	0	0	0	0	0	0	0

96 rows × 97 columns

In [34]:

```
airlines_data1.shape
```

Out[34]:

(96, 97)

In [35]:

```
airlines_data1=airlines_data1.iloc[:,0:13]
```

In [36]:

```
airlines_data1
```

Out[36]:

	Passengers	Month_1	Month_2	Month_3	Month_4	Month_5	Month_6	Month_7	Month_8
0	112	1	0	0	0	0	0	0	0
1	118	0	1	0	0	0	0	0	0
2	132	0	0	1	0	0	0	0	0
3	129	0	0	0	1	0	0	0	0
4	121	0	0	0	0	1	0	0	0
...	...	...	...	...	...	...	...	...	...
91	405	0	0	0	0	0	0	0	0
92	355	0	0	0	0	0	0	0	0
93	306	0	0	0	0	0	0	0	0
94	271	0	0	0	0	0	0	0	0
95	306	0	0	0	0	0	0	0	0

96 rows × 13 columns

In [37]:

```
t= np.arange(1,97)
```

In [38]:

```
airlines_data1['t'] = t
```

In [39]:

```
airlines_data1['t_sq'] = airlines_data1['t']*airlines_data1['t']
```

In [40]:

```
log_Passengers=np.log(airlines_data1['Passengers'])
```

In [41]:

```
airlines_data1['log_Passengers']=log_Passengers
```

In [42]:

```
pd.set_option("max_columns",None)
```

In [43]:

```
airlines_data1.head()
```

Out[43]:

	Passengers	Month_1	Month_2	Month_3	Month_4	Month_5	Month_6	Month_7	Month_8
0	112	1	0	0	0	0	0	0	0
1	118	0	1	0	0	0	0	0	0
2	132	0	0	1	0	0	0	0	0
3	129	0	0	0	1	0	0	0	0
4	121	0	0	0	0	1	0	0	0

In [44]:

```
train1, test1 = np.split(airlines_data1, [int(.67 *len(airlines_data1))])
```

In [45]:

```
linear= smf.ols('Passengers ~ t',data=train1).fit()
predlin=pd.Series(linear.predict(pd.DataFrame(test1['t'])))
rmselin=np.sqrt((np.mean(np.array(test1['Passengers'])-np.array(predlin))**2))
rmselin
```

Out[45]:

25.50398351648351

In [46]:

```
quad=smf.ols('Passengers~t+t_sq',data=train1).fit()
predquad=pd.Series(quad.predict(pd.DataFrame(test1[['t','t_sq']])))
rmsequad=np.sqrt(np.mean((np.array(test1['Passengers'])-np.array(predquad))**2))
rmsequad
```

Out[46]:

53.18955514415471

In [47]:

```
expo=smf.ols('Passengers~t',data=train1).fit()
predexp=pd.Series(expo.predict(pd.DataFrame(test1['t'])))
rmseexpo=np.sqrt(np.mean((np.array(test1['Passengers'])-np.array(np.exp(predexp)))**2))
rmseexpo
```

Out[47]:

1.6030945933278588e+128

In [48]:

```
:h_5+Month_6+Month_7+Month_8+Month_9+Month_10+Month_11+Month_12',data=train1).fit()
lonth_2','Month_3','Month_4','Month_5','Month_6','Month_7','Month_8','Month_9','Month_10','M
:dadd))**2))
```

Out[48]:

118.60439843615572

In [49]:

```
addlinear= smf.ols('Passengers~ Month_1+Month_2+Month_3+Month_4+Month_5+Month_6+Month_7+Mor
predaddlinear=pd.Series(addlinear.predict(pd.DataFrame(test1[['t','Month_1','Month_2','Mont
rmseaddlinear=np.sqrt(np.mean((np.array(test1['Passengers'])-np.array(predaddlinear))**2))
rmseaddlinear
```

Out[49]:

118.60439843615572

In [50]:

```
addquad=smf.ols('Passengers~t+t_sq+Month_1+Month_2+Month_3+Month_4+Month_5+Month_6+Month_7+
predaddquad=pd.Series(addquad.predict(pd.DataFrame(test1[['t','t_sq','Month_1','Month_2','M
rmseaddquad=np.sqrt(np.mean((np.array(test1['Passengers'])-np.array(predaddquad))**2))
rmseaddquad
```

Out[50]:

83.97481125259831

In [51]:

```
mulsea=smf.ols('log_Passengers~Month_1+Month_2+Month_3+Month_4+Month_5+Month_6+Month_7+Mont
predmul= pd.Series(mulsea.predict(pd.DataFrame(test1[['Month_1','Month_2','Month_3','Month_
rmsemul= np.sqrt(np.mean((np.array(test1['Passengers'])-np.array(np.exp(predmul)))**2))
rmsemul
```

Out[51]:

122.20973607172188

In [52]:

```
mullin= smf.ols('log_Passengers~t+Month_1+Month_2+Month_3+Month_4+Month_5+Month_6+Month_7+M
predmullin= pd.Series(mullin.predict(pd.DataFrame(test1[['t','Month_1','Month_2','Month_3',
rmsemulin=np.sqrt(np.mean((np.array(test1['Passengers'])-np.array(np.exp(predmullin)))**2))
rmsemulin
```

Out[52]:

40.90194555071152

In [53]:

```
mul_quad= smf.ols('log_Passengers~t+t_sq+Month_1+Month_2+Month_3+Month_4+Month_5+Month_6+Mc
pred_mul_quad= pd.Series(mul_quad.predict(test1[['t','t_sq','Month_1','Month_2','Month_3','
rmse_mul_quad=np.sqrt(np.mean((np.array(test1['Passengers'])-np.array(np.exp(pred_mul_quad)
rmse_mul_quad
```

Out[53]:

```
100.05904451760759
```

lowest RMSE value is best.

In [54]:

```
output = {'Model':pd.Series(['rmse_mul_quad','rmseadd','rmseaddlinear','rmseaddquad','rmsee
          'Values':pd.Series([rmse_mul_quad,rmseadd,rmseaddlinear,rmseaddquad,rmseexpo,rmse
```

In [55]:

```
rmse=pd.DataFrame(output)
```

In [56]:

```
print(output)
```

```
{'Model': 0    rmse_mul_quad
1          rmseadd
2    rmseaddlinear
3      rmseaddquad
4         rmseexpo
5          rmselin
6          rmsemul
7        rmsemulin
8         rmsequad
dtype: object, 'Values': 0     1.000590e+02
1     1.186044e+02
2     1.186044e+02
3     8.397481e+01
4    1.603095e+128
5     2.550398e+01
6     1.222097e+02
7     4.090195e+01
8     5.318956e+01
dtype: float64}
```

Solution on Business Problem: we have created dummy variables of Month, drop other years of month to remove Multicollinearity issues.

The best RMSE value is 25.50 which is Linearity of trend.