0.1 Prediction Accuracy of Linear Processes :

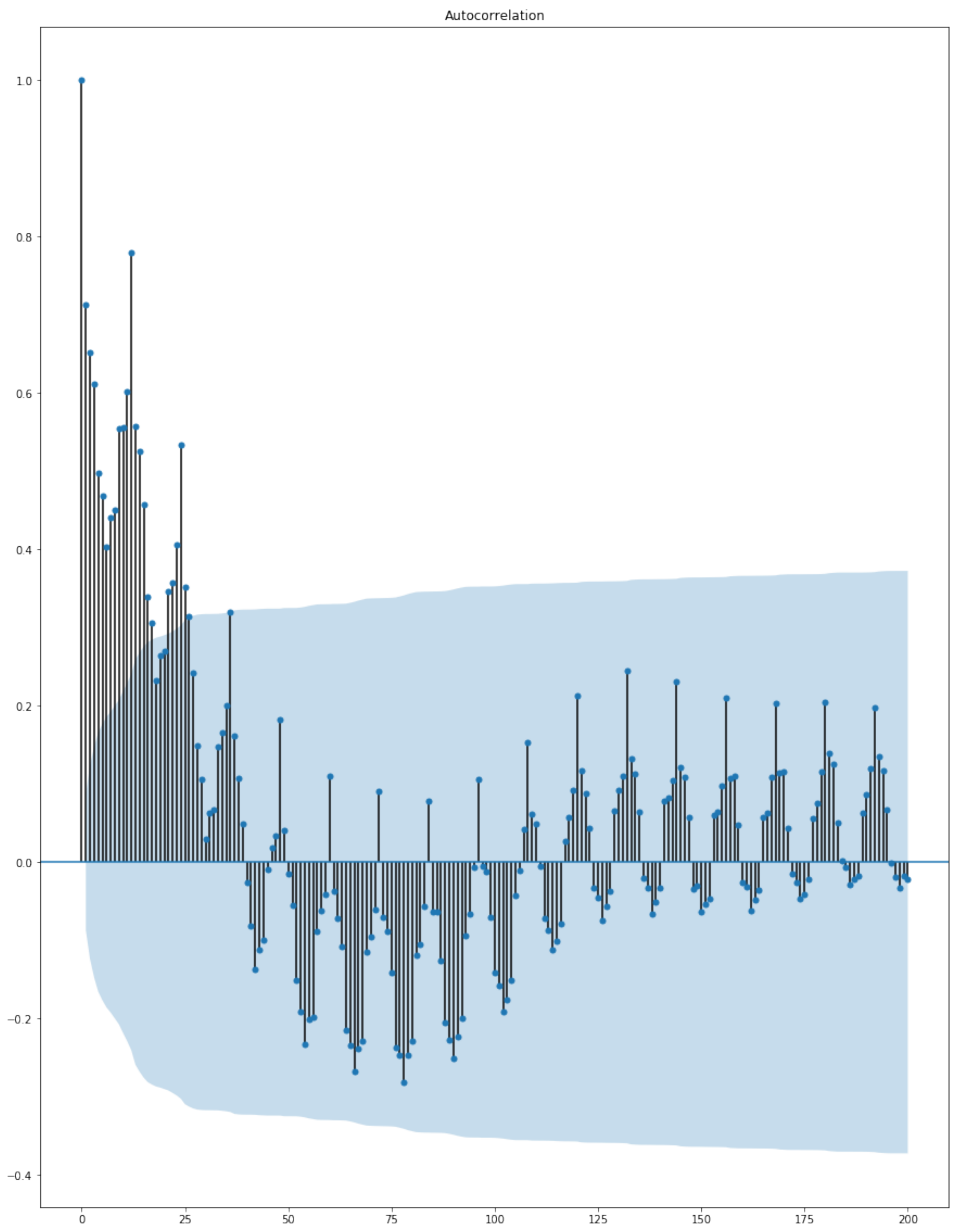
```
In [44]: # test of stationarity: Dickey fuller test:

         from statsmodels.tsa.stattools import adfuller

In [45]: # Dickey fuller test:

         x = series.values
         result = adfuller(x)
         print('ADF Statistic: %f' % result[0])
         print('p-value: %f' % result[1])
         print('Critical Values:')
         for key, value in result[4].items():
                 print('\t%s: %.3f' % (key, value))
```

```
ADF Statistic: -3.056715
p-value: 0.029927
Critical Values:
        1%: -3.444
        5%: -2.867
        10%: -2.570
```

0.1.1 Since the P value of above data is less than 0.05, we can say that Light weight vehicle sales data is stationary

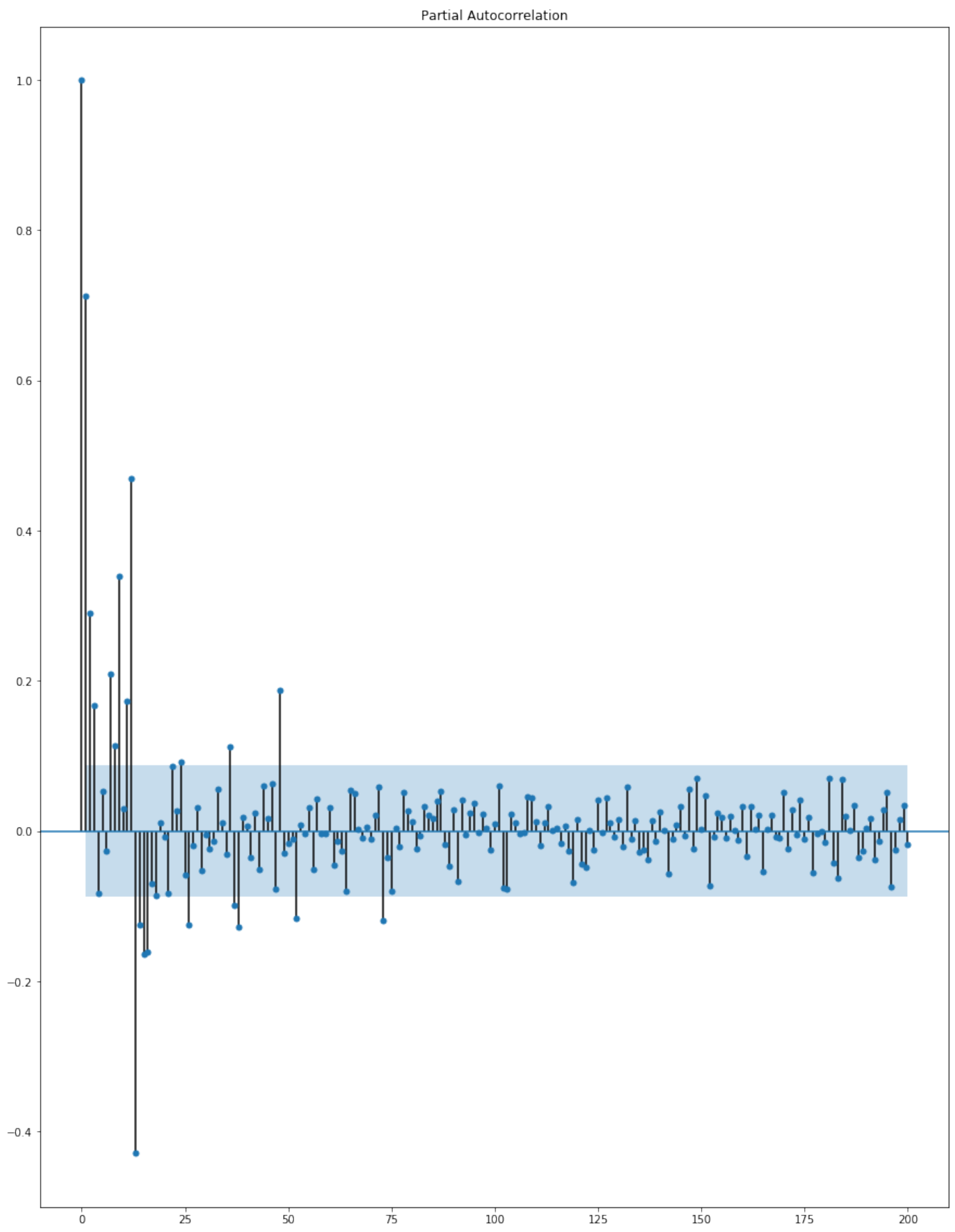
0.1.2 Hence we need not do any kind of differencing to stationarize the data

```
In [46]: import statsmodels.graphics.tsaplots as tsaplots

In [48]: tsaplots.plot_acf(series.values, lags = 200)

   Out[48]:
```



```
In [49]: tsaplots.plot_pacf(series.values, lags = 200)
```

```
Out[49]:
```


Partial Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
0
25
50
75
100
125
150
175
200

0.1.3 As both ACF and PACF plots decrease geometrically, we can say that there are AR and MA terms in the data

0.1.4 In the below code, we are performing a grided search to determine the order of SARIMA MODEL that can explain underlying data generation process of Light vehicle sales.

```
In [50]: import itertools

In [51]: p = d = q = range(0,3)

         pdq = list(itertools.product(p,d,q))

In [52]: seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p,d,q))]

In [53]: for param in pdq:
             for param_seasonal in seasonal_pdq:
                 try:
                     mod = sm.tsa.statespace.SARIMAX(series,
                                                     order=param,
                                                     seasonal_order=param_seasonal,
                                                     enforce_stationarity=False,
                                                     enforce_invertibility=False)
                     results = mod.fit()
                     print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, results.aic))
                 except:
                     continue
```

```
ARIMA(0, 0, 0)x(0, 0, 1, 12)12 - AIC:7842.552975756198
ARIMA(0, 0, 0)x(0, 0, 2, 12)12 - AIC:7219.919837636688
ARIMA(0, 0, 0)x(0, 1, 1, 12)12 - AIC:6115.9781667933585
ARIMA(0, 0, 0)x(0, 1, 2, 12)12 - AIC:5972.542254225153
ARIMA(0, 0, 0)x(0, 2, 1, 12)12 - AIC:6022.45688339753
ARIMA(0, 0, 0)x(0, 2, 2, 12)12 - AIC:5848.901186554778
ARIMA(0, 0, 0)x(1, 0, 0, 12)12 - AIC:6283.7763289096765
ARIMA(0, 0, 0)x(1, 0, 1, 12)12 - AIC:6270.5523206118805
ARIMA(0, 0, 0)x(1, 0, 2, 12)12 - AIC:6119.24276527093
ARIMA(0, 0, 0)x(1, 1, 0, 12)12 - AIC:6128.167940702808
ARIMA(0, 0, 0)x(1, 1, 1, 12)12 - AIC:6117.574141282513
ARIMA(0, 0, 0)x(1, 1, 2, 12)12 - AIC:5939.714330523686
ARIMA(0, 0, 0)x(1, 2, 0, 12)12 - AIC:6192.260406736433
ARIMA(0, 0, 0)x(1, 2, 1, 12)12 - AIC:6020.591917758886
ARIMA(0, 0, 0)x(1, 2, 2, 12)12 - AIC:5850.90118351551


/Users/revanthkota/anaconda3/lib/python3.6/site-packages/statsmodels/base/model.py:496: Conver
  "Check mle_retvals", ConvergenceWarning)


ARIMA(0, 0, 0)x(2, 0, 0, 12)12 - AIC:6129.8833713325
ARIMA(0, 0, 0)x(2, 0, 1, 12)12 - AIC:6131.867263598623
```

```
/Users/revanthkota/anaconda3/lib/python3.6/site-packages/statsmodels/base/model.py:496: Converg
  "Check mle_retvals", ConvergenceWarning)


ARIMA(1, 1, 2)x(2, 1, 2, 12)12 - AIC:5548.642797139928
ARIMA(1, 1, 2)x(2, 2, 0, 12)12 - AIC:5746.328233235452


/Users/revanthkota/anaconda3/lib/python3.6/site-packages/statsmodels/base/model.py:496: Converg
  "Check mle_retvals", ConvergenceWarning)


ARIMA(1, 1, 2)x(2, 2, 1, 12)12 - AIC:5542.646118544128


/Users/revanthkota/anaconda3/lib/python3.6/site-packages/statsmodels/base/model.py:496: Converg
  "Check mle_retvals", ConvergenceWarning)


ARIMA(1, 1, 2)x(2, 2, 2, 12)12 - AIC:5459.169188474549
ARIMA(1, 2, 0)x(0, 0, 0, 12)12 - AIC:6826.33962923996
ARIMA(1, 2, 0)x(0, 0, 1, 12)12 - AIC:6469.893582195147
ARIMA(1, 2, 0)x(0, 0, 2, 12)12 - AIC:6261.326495826825
ARIMA(1, 2, 0)x(0, 1, 0, 12)12 - AIC:6416.822699625098
ARIMA(1, 2, 0)x(0, 1, 1, 12)12 - AIC:6145.071257714915
ARIMA(1, 2, 0)x(0, 1, 2, 12)12 - AIC:5971.952641848971
ARIMA(1, 2, 0)x(0, 2, 0, 12)12 - AIC:6714.313599600071
ARIMA(1, 2, 0)x(0, 2, 1, 12)12 - AIC:6159.5270654492415
ARIMA(1, 2, 0)x(0, 2, 2, 12)12 - AIC:5892.602560681295
ARIMA(1, 2, 0)x(1, 0, 0, 12)12 - AIC:6344.092997613827
ARIMA(1, 2, 0)x(1, 0, 1, 12)12 - AIC:6298.000975580917
ARIMA(1, 2, 0)x(1, 0, 2, 12)12 - AIC:6123.146674300239
ARIMA(1, 2, 0)x(1, 1, 0, 12)12 - AIC:6231.831545593295
ARIMA(1, 2, 0)x(1, 1, 1, 12)12 - AIC:6127.525270690478
ARIMA(1, 2, 0)x(1, 1, 2, 12)12 - AIC:5972.731329506781
ARIMA(1, 2, 0)x(1, 2, 0, 12)12 - AIC:6402.590140302624
ARIMA(1, 2, 0)x(1, 2, 1, 12)12 - AIC:6128.803323594103


/Users/revanthkota/anaconda3/lib/python3.6/site-packages/statsmodels/base/model.py:496: Converg
  "Check mle_retvals", ConvergenceWarning)


ARIMA(1, 2, 0)x(1, 2, 2, 12)12 - AIC:5876.3683584842875
ARIMA(1, 2, 0)x(2, 0, 0, 12)12 - AIC:6184.938153415418
ARIMA(1, 2, 0)x(2, 0, 1, 12)12 - AIC:6134.828018063551
ARIMA(1, 2, 0)x(2, 0, 2, 12)12 - AIC:6124.051560396623
ARIMA(1, 2, 0)x(2, 1, 0, 12)12 - AIC:6041.283818446308
ARIMA(1, 2, 0)x(2, 1, 1, 12)12 - AIC:5967.3083422391
```

```
/Users/revanthkota/anaconda3/lib/python3.6/site-packages/statsmodels/base/model.py:496: Converg
  "Check mle_retvals", ConvergenceWarning)


ARIMA(2, 2, 2)x(2, 0, 2, 12)12 - AIC:5822.122977811019


/Users/revanthkota/anaconda3/lib/python3.6/site-packages/statsmodels/base/model.py:496: Converg
  "Check mle_retvals", ConvergenceWarning)


ARIMA(2, 2, 2)x(2, 1, 0, 12)12 - AIC:5631.368499696546


/Users/revanthkota/anaconda3/lib/python3.6/site-packages/statsmodels/base/model.py:496: Converg
  "Check mle_retvals", ConvergenceWarning)


ARIMA(2, 2, 2)x(2, 1, 1, 12)12 - AIC:5563.2155851226735


/Users/revanthkota/anaconda3/lib/python3.6/site-packages/statsmodels/base/model.py:496: Converg
  "Check mle_retvals", ConvergenceWarning)


ARIMA(2, 2, 2)x(2, 1, 2, 12)12 - AIC:5609.401653570437


/Users/revanthkota/anaconda3/lib/python3.6/site-packages/statsmodels/base/model.py:496: Converg
  "Check mle_retvals", ConvergenceWarning)


ARIMA(2, 2, 2)x(2, 2, 0, 12)12 - AIC:5749.71167988074


/Users/revanthkota/anaconda3/lib/python3.6/site-packages/statsmodels/base/model.py:496: Converg
  "Check mle_retvals", ConvergenceWarning)


ARIMA(2, 2, 2)x(2, 2, 1, 12)12 - AIC:5543.595338262419


/Users/revanthkota/anaconda3/lib/python3.6/site-packages/statsmodels/base/model.py:496: Converg
  "Check mle_retvals", ConvergenceWarning)


ARIMA(2, 2, 2)x(2, 2, 2, 12)12 - AIC:5775.708279549721
```

0.1.5 Here, we are using SARIMA process of order (1,1,2)x(2,2,2,12)12 as it's AIC score is least of all models generated from grided search.

```
In [54]: mod = sm.tsa.statespace.SARIMAX(series, order=(1,1,2),
                                         seasonal_order = (2,2,2,12),
                                         enforce_stationarity = False,
                                         enforce_invertibility = False) # Light weight vehicle 
```

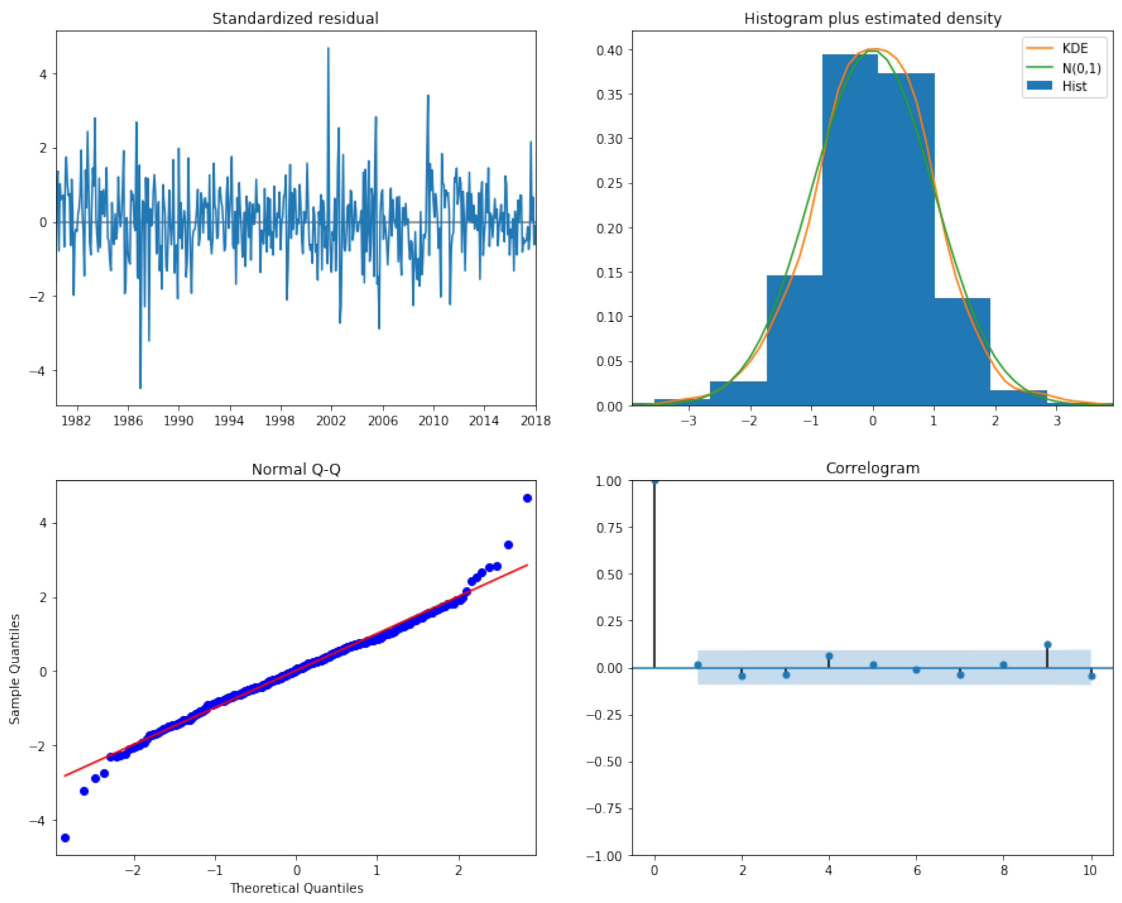
```
In [56]: results = mod.fit()
```

```
/Users/revanthkota/anaconda3/lib/python3.6/site-packages/statsmodels/base/model.py:496: Conver
  "Check mle_retvals", ConvergenceWarning)
```

```
In [57]: print(results.summary().tables[1])
```

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.8895      0.072    -12.310      0.000      -1.031      -0.748
ma.L1          0.2852      0.078      3.673      0.000       0.133       0.437
ma.L2         -0.6128      0.049    -12.435      0.000      -0.709      -0.516
ar.S.L12       0.2181      0.052      4.189      0.000       0.116       0.320
ar.S.L24      -0.1027      0.041     -2.486      0.013      -0.184      -0.022
ma.S.L12      -1.8473      5.366     -0.344      0.731     -12.364       8.669
ma.S.L24       0.8475      4.535      0.187      0.852      -8.041       9.736
sigma2      8107.7563   4.37e+04      0.185      0.853   -7.76e+04    9.38e+04
==============================================================================
```

```
In [59]: results.plot_diagnostics(figsize=(15, 12))
         plt.show() # Residual plot of Light vehicle sales
```



0.1.6 Based on above diagnostics we can see that residuals for light weight vehicles are highly uncorrelated and are random

0.1.7 As KDE Line follows closely with the N(0,1) line, this is a good indication that residuals follow Normal Distribution with mean 0 and standard deviation 1

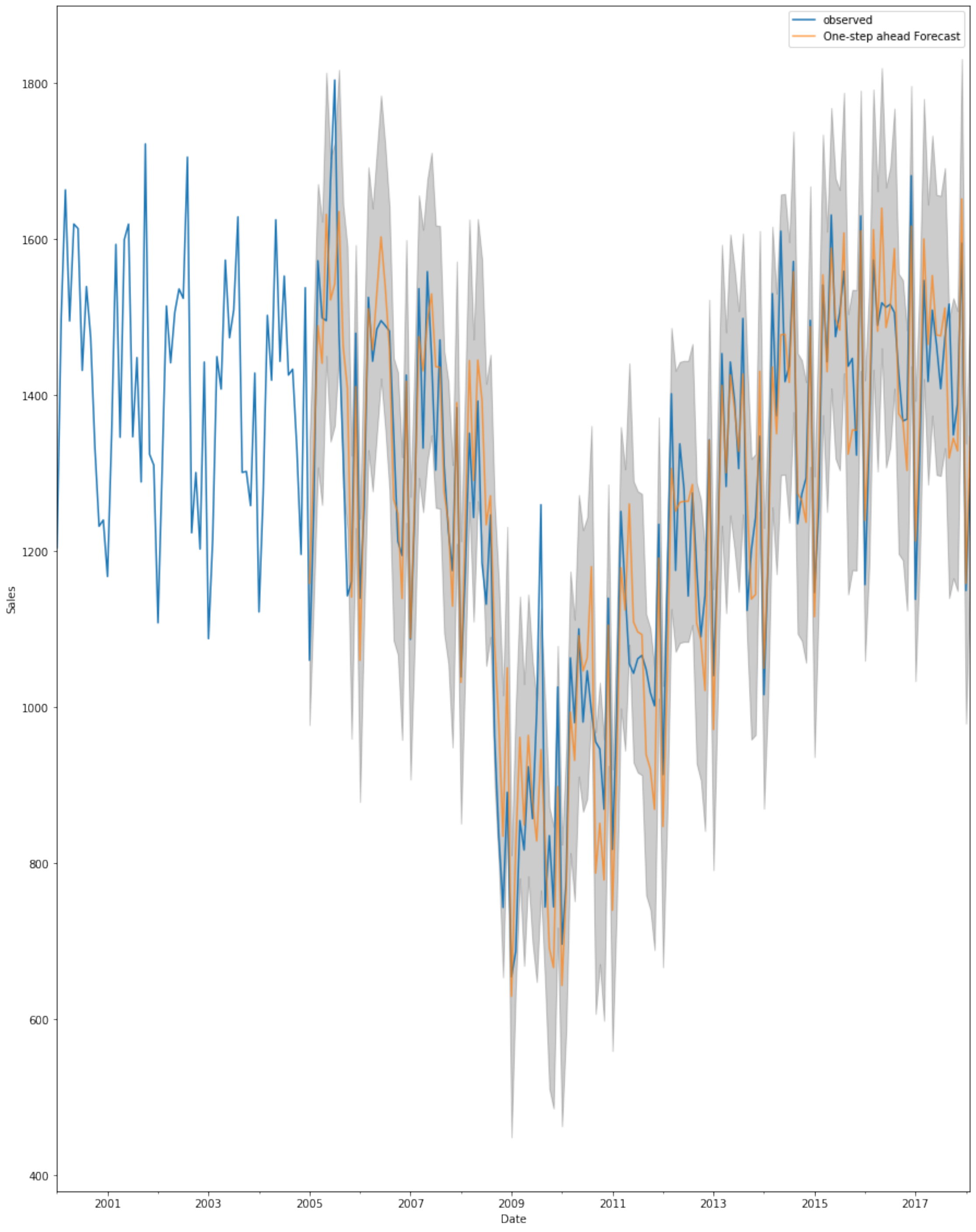
0.1.8 The qq-plot on the bottom shows that the ordered distribution of residuals (blue dots) follows the linear trend of the samples taken from a standard normal distribution with N(0,1).

0.1.9 Again, this is a strong indication that the residuals are normally distributed.

0.1.10 The residuals over time (top left plot) don't display any obvious seasonality and appear to be white noise. this is confirmed by the autocorrelation (i.e. correlogram) plot on the bottom right, which shows that the time series residuals have low correlation with lagged versions of itself.

0.1.11 Those observations lead us to conclude that our model produces a satisfactory fit that could help us understand our time series data and forecast future values.

```
In [60]: pred = results.get_prediction(start = pd.to_datetime('2005-01-01'), dynamic=False)
In [61]: pred_ci = pred.conf_int()

In [62]: ax = series['2000':].plot(label='observed')
         pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=.7)
         ax.fill_between(pred_ci.index,
         pred_ci.iloc[:, 0],
         pred_ci.iloc[:, 1], color='k', alpha=.2)
         ax.set_xlabel('Date')
         ax.set_ylabel('Sales')
         plt.legend()
         plt.show() # Confidence interval for Light weight vehicle sales data which is a stati
```



```
In [63]: y_forecasted = pred.predicted_mean
         y_truth = series['2005-01-01':]
         # Compute the mean square error
         mse = ((y_forecasted - y_truth) ** 2).mean()
         print('The Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))
```

```
The Mean Squared Error of our forecasts is 7554.61
```

0.1.12 Here we can see that the one step forecasting error of LINEAR SARIMA process is 7554 which is quite high hence used neural netroworks to see it they can reduce Forecasting error

0.2 LSTM Neural Networks predicting Light weight vehicles Sales:

0.2.1 The below is an LSTM Model containing 4 Neurons trained on a test data for 100 epochs to predict Light weight vehicle sales.

```
In [18]: from pandas import DataFrame
         from pandas import Series
         from pandas import concat
         from pandas import read_csv
         from pandas import datetime
         from sklearn.metrics import mean_squared_error
         from sklearn.preprocessing import MinMaxScaler
         from keras.models import Sequential
         from keras.layers import Dense
         from keras.layers import LSTM
         from math import sqrt
         from matplotlib import pyplot
         import numpy

         # date-time parsing function for loading the dataset
         def parser(x):
                 return datetime.strptime(x, '%Y-%m')

         # frame a sequence as a supervised learning problem
         def timeseries_to_supervised(data, lag=1):
                 df = DataFrame(data)
                 columns = [df.shift(i) for i in range(1, lag+1)]
                 columns.append(df)
                 df = concat(columns, axis=1)
                 df.fillna(0, inplace=True)
                 return df

         # create a differenced series
         def difference(dataset, interval=1):
                 diff = list()
                 for i in range(interval, len(dataset)):
                         value = dataset[i] - dataset[i - interval]
                         diff.append(value)
                 return Series(diff)

         # invert differenced value
         def inverse_difference(history, yhat, interval=1):
```

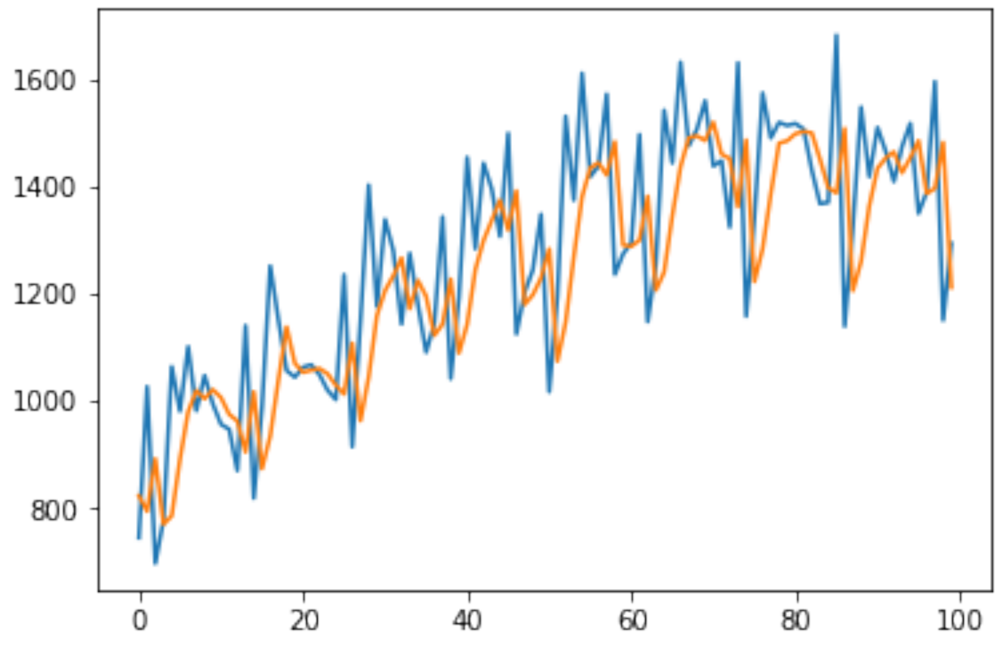
```
        return yhat + history[-interval]

# scale train and test data to [-1, 1]
def scale(train, test):
        # fit scaler
        scaler = MinMaxScaler(feature_range=(-1, 1))
        scaler = scaler.fit(train)
        # transform train
        train = train.reshape(train.shape[0], train.shape[1])
        train_scaled = scaler.transform(train)
        # transform test
        test = test.reshape(test.shape[0], test.shape[1])
        test_scaled = scaler.transform(test)
        return scaler, train_scaled, test_scaled

# inverse scaling for a forecasted value
def invert_scale(scaler, X, value):
        new_row = [x for x in X] + [value]
        array = numpy.array(new_row)
        array = array.reshape(1, len(array))
        inverted = scaler.inverse_transform(array)
        return inverted[0, -1]

# fit an LSTM network to training data
def fit_lstm(train, batch_size, nb_epoch, neurons):
        X, y = train[:, 0:-1], train[:, -1]
        X = X.reshape(X.shape[0], 1, X.shape[1])
        model = Sequential()
        model.add(LSTM(neurons, batch_input_shape=(batch_size, X.shape[1], X.shape[2]
        model.add(Dense(1))
        model.compile(loss='mean_squared_error', optimizer='adam')
        for i in range(nb_epoch):
                model.fit(X, y, epochs=1, batch_size=batch_size, verbose=0, shuffle=F
                model.reset_states()
        return model

# make a one-step forecast
def forecast_lstm(model, batch_size, X):
        X = X.reshape(1, 1, len(X))
        yhat = model.predict(X, batch_size=batch_size)
        return yhat[0,0]

# load dataset
series = read_csv('LTOTALNSA.csv', header=0, parse_dates=[0], index_col=0, squeeze=Tr

# transform data to be stationary
raw_values = series.values
diff_values = difference(raw_values, 1)
```

```
        # transform data to be supervised learning
        supervised = timeseries_to_supervised(diff_values, 1)
        supervised_values = supervised.values

        # split data into train and test-sets
        train, test = supervised_values[0:-100], supervised_values[-100:]

        # transform the scale of the data
        scaler, train_scaled, test_scaled = scale(train, test)

        # fit the model
        lstm_model = fit_lstm(train_scaled, 1, 100, 4)
        # forecast the entire training dataset to build up state for forecasting
        train_reshaped = train_scaled[:, 0].reshape(len(train_scaled), 1, 1)
        lstm_model.predict(train_reshaped, batch_size=1)

        # walk-forward validation on the test data
        predictions = list()
        for i in range(len(test_scaled)):
                # make one-step forecast
                X, y = test_scaled[i, 0:-1], test_scaled[i, -1]
                yhat = forecast_lstm(lstm_model, 1, X)
                # invert scaling
                yhat = invert_scale(scaler, X, yhat)
                # invert differencing
                yhat = inverse_difference(raw_values, yhat, len(test_scaled)+1-i)
                # store forecast
                predictions.append(yhat)
                expected = raw_values[len(train) + i + 1]
                print('Month=%d, Predicted=%f, Expected=%f' % (i+1, yhat, expected))

        # report performance
        rmse = sqrt(mean_squared_error(raw_values[-100:], predictions))
        print('Test RMSE: %.3f' % rmse)
        # line plot of observed vs predicted
        pyplot.plot(raw_values[-100:])
        pyplot.plot(predictions)
        pyplot.show()
```

```
Month=1, Predicted=822.719251, Expected=744.400000
Month=2, Predicted=793.663385, Expected=1026.300000
Month=3, Predicted=891.597665, Expected=696.600000
Month=4, Predicted=769.663400, Expected=778.400000
Month=5, Predicted=785.328216, Expected=1063.600000
Month=6, Predicted=890.779998, Expected=980.400000
Month=7, Predicted=979.150968, Expected=1100.800000
Month=8, Predicted=1018.932047, Expected=981.300000
```

```
Month=9, Predicted=1003.526829, Expected=1047.000000
Month=10, Predicted=1020.618248, Expected=994.300000
Month=11, Predicted=1005.856157, Expected=956.000000
Month=12, Predicted=974.658995, Expected=946.700000
Month=13, Predicted=962.019754, Expected=869.600000
Month=14, Predicted=904.228174, Expected=1140.100000
Month=15, Predicted=1016.571042, Expected=818.100000
Month=16, Predicted=873.128526, Expected=987.900000
Month=17, Predicted=933.879697, Expected=1251.300000
Month=18, Predicted=1038.902183, Expected=1150.500000
Month=19, Predicted=1137.531011, Expected=1056.100000
Month=20, Predicted=1069.671717, Expected=1043.800000
Month=21, Predicted=1052.825174, Expected=1062.400000
Month=22, Predicted=1056.796143, Expected=1066.800000
Month=23, Predicted=1059.973203, Expected=1048.700000
Month=24, Predicted=1049.957173, Expected=1019.200000
Month=25, Predicted=1028.754280, Expected=1002.100000
Month=26, Predicted=1012.306647, Expected=1234.900000
Month=27, Predicted=1107.182047, Expected=913.900000
Month=28, Predicted=962.516159, Expected=1148.900000
Month=29, Predicted=1044.185744, Expected=1402.200000
Month=30, Predicted=1158.779948, Expected=1175.700000
Month=31, Predicted=1205.431224, Expected=1338.000000
Month=32, Predicted=1232.688244, Expected=1279.700000
Month=33, Predicted=1266.053399, Expected=1142.800000
Month=34, Predicted=1171.241821, Expected=1275.000000
Month=35, Predicted=1224.162531, Expected=1179.600000
Month=36, Predicted=1194.596141, Expected=1090.300000
Month=37, Predicted=1121.695211, Expected=1144.000000
Month=38, Predicted=1142.731224, Expected=1343.100000
Month=39, Predicted=1226.706780, Expected=1040.800000
Month=40, Predicted=1088.836266, Expected=1193.000000
Month=41, Predicted=1142.479660, Expected=1453.700000
Month=42, Predicted=1244.591689, Expected=1283.200000
Month=43, Predicted=1298.878342, Expected=1442.900000
Month=44, Predicted=1336.182906, Expected=1394.300000
Month=45, Predicted=1373.608538, Expected=1306.200000
Month=46, Predicted=1318.128180, Expected=1498.700000
Month=47, Predicted=1391.370506, Expected=1124.300000
Month=48, Predicted=1179.223517, Expected=1201.700000
Month=49, Predicted=1197.178547, Expected=1243.700000
Month=50, Predicted=1228.506549, Expected=1347.700000
Month=51, Predicted=1283.009308, Expected=1016.400000
Month=52, Predicted=1073.680311, Expected=1194.300000
Month=53, Predicted=1146.302852, Expected=1530.500000
Month=54, Predicted=1268.501231, Expected=1373.800000
Month=55, Predicted=1379.800172, Expected=1610.600000
Month=56, Predicted=1433.902086, Expected=1417.600000
```

```
Month=57, Predicted=1443.414872, Expected=1437.600000
Month=58, Predicted=1420.640042, Expected=1571.700000
Month=59, Predicted=1482.836827, Expected=1235.500000
Month=60, Predicted=1289.795355, Expected=1273.700000
Month=61, Predicted=1287.726772, Expected=1294.100000
Month=62, Predicted=1299.268016, Expected=1496.300000
Month=63, Predicted=1381.425464, Expected=1147.200000
Month=64, Predicted=1206.026815, Expected=1253.900000
Month=65, Predicted=1239.548407, Expected=1541.600000
Month=66, Predicted=1344.760955, Expected=1443.100000
Month=67, Predicted=1435.952510, Expected=1631.300000
Month=68, Predicted=1490.074206, Expected=1475.200000
Month=69, Predicted=1493.617693, Expected=1506.200000
Month=70, Predicted=1484.899841, Expected=1559.300000
Month=71, Predicted=1519.114807, Expected=1437.400000
Month=72, Predicted=1459.611937, Expected=1447.500000
Month=73, Predicted=1451.391497, Expected=1323.500000
Month=74, Predicted=1361.524456, Expected=1630.100000
Month=75, Predicted=1486.159729, Expected=1157.400000
Month=76, Predicted=1221.900439, Expected=1341.100000
Month=77, Predicted=1282.856155, Expected=1573.700000
Month=78, Predicted=1385.801583, Expected=1489.700000
Month=79, Predicted=1479.922564, Expected=1518.700000
Month=80, Predicted=1484.436133, Expected=1513.100000
Month=81, Predicted=1497.641076, Expected=1516.700000
Month=82, Predicted=1501.733193, Expected=1506.200000
Month=83, Predicted=1499.192134, Expected=1429.700000
Month=84, Predicted=1449.898116, Expected=1367.200000
Month=85, Predicted=1397.000141, Expected=1369.600000
Month=86, Predicted=1387.262420, Expected=1681.600000
Month=87, Predicted=1507.630211, Expected=1138.600000
Month=88, Predicted=1204.969848, Expected=1324.100000
Month=89, Predicted=1259.512292, Expected=1547.500000
Month=90, Predicted=1362.969920, Expected=1417.700000
Month=91, Predicted=1433.323553, Expected=1509.300000
Month=92, Predicted=1452.194812, Expected=1465.100000
Month=93, Predicted=1463.952632, Expected=1408.600000
Month=94, Predicted=1425.015742, Expected=1473.900000
Month=95, Predicted=1452.858464, Expected=1517.100000
Month=96, Predicted=1484.652571, Expected=1349.200000
Month=97, Predicted=1386.084499, Expected=1389.200000
Month=98, Predicted=1395.855028, Expected=1595.100000
Month=99, Predicted=1481.541371, Expected=1149.900000
Month=100, Predicted=1211.152909, Expected=1293.400000
Test RMSE: 159.757
```



0.3 Conclusions:

0.3.1 By comparing above prediction results of both Linear SARIMA and LSTM Models, we can say that Though Linear SARIMA(1,1,2)x(2,2,2,12)12 does a good job in explaining the underlying data generation process, LSTM models do a better job in forecasting Light weight vehicle sales as the error of LSTM forecasting is 7000 times less than SARIMA Forecasting