In [45]:
```
#Homework1
#AMS559: Smart Enegry in the Information Age
#Author: Shivasagar Boraiah, Nikhil Siddhartha

#Prediction of Absolute Mean Error for Home Data by training the model

#I concentrated on one Home data and generated all posiible plots such a
s (Weekly variation, Monthly Variation,
# Daily Variation and other intresting plots according to dataset give
n.)

#Trained with Linear Regression, RandomForestRegressor, SARIMA Model.
#Acheived an RMSE of 1.37, 1.36 and 1.74 respectively

#I have discussed this problem with Sravani Panakanti and reffered GitHu
b

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from datetime import datetime
%matplotlib inline

from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.pipeline import make_pipeline
from sklearn import linear_model
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor
```

In [46]:
```
#Adding the calender to the dataset to get more features to train
time = datetime.strptime('2014-12-01 00:00:00', '%Y-%m-%d %H:%M:%S')
time
```

Out[46]:
```
datetime.datetime(2014, 12, 1, 0, 0)
```

In [48]:
```
#Interval the data with the frequency of 15Mins
range_ = pd.date_range('2014-12-01',periods=35039 ,freq="15min")
print (len(range_))
```

```
35039
```

In [49]:
```
data1 = pd.read_csv('Home1_yr1.csv')
```

In [50]:
```
data1['userId'] = 1
```

In [51]:
```
date_range=range(0, 35040)
```

In [52]:
```
len(data1)
data1['dateval']=range(0, 35039)
```

In [53]:
```
data1.head()
```

Out[53]:

	0.65018	userId	dateval
0	1.45400	1	0
1	0.72971	1	1
2	3.10750	1	2
3	0.63572	1	3
4	0.69720	1	4

In [54]:
```
data1['month'] = range_.month
```

In [55]:
```
data1['dayofweek'] = range_.dayofweek
```

In [56]:
```
data1['weekday'] = range_.weekday
```

In [57]:
```
data1['hour'] = range_.hour
```

In [58]:
```
data1.head()
```

Out[58]:

	0.65018	userId	dateval	month	dayofweek	weekday	hour
0	1.45400	1	0	12	0	0	0
1	0.72971	1	1	12	0	0	0
2	3.10750	1	2	12	0	0	0
3	0.63572	1	3	12	0	0	0
4	0.69720	1	4	12	0	0	1

In [59]:
```
data1 = data1.rename(columns={'0.65018': 'demand'})
data1.describe()
```

Out[59]:

	demand	userId	dateval	month	dayofweek	weekday
count	35039.000000	35039.0	35039.000000	35039.000000	35039.000000	35039.000000
mean	1.327227	1.0	17519.000000	6.525900	2.991866	2.991866
std	1.399034	0.0	10115.032378	3.447867	2.003398	2.003398
min	0.015124	1.0	0.000000	1.000000	0.000000	0.000000
25%	0.315675	1.0	8759.500000	4.000000	1.000000	1.000000
50%	0.722890	1.0	17519.000000	7.000000	3.000000	3.000000
75%	1.988550	1.0	26278.500000	10.000000	5.000000	5.000000
max	15.500000	1.0	35038.000000	12.000000	6.000000	6.000000

In [60]:
```
data1.head()
```

Out[60]:

	demand	userId	dateval	month	dayofweek	weekday	hour
0	1.45400	1	0	12	0	0	0
1	0.72971	1	1	12	0	0	0
2	3.10750	1	2	12	0	0	0
3	0.63572	1	3	12	0	0	0
4	0.69720	1	4	12	0	0	1

In [61]:
```
rf = RandomForestRegressor()
len(data1)
```

Out[61]: 35039

In [62]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import sklearn as sk
import scipy.stats as scp
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt
from statsmodels.tsa.stattools import adfuller
import statsmodels.api as sm
from statsmodels.tsa.stattools import acf
from statsmodels.tsa.stattools import pacf
from statsmodels.tsa.seasonal import seasonal_decompose
import itertools as it
import warnings
```

In [63]:
```
X = data1.drop('demand',axis=1)
y = y = data1[['demand']]
```

In [64]:
```
from sklearn.model_selection import train_test_split
```

In [65]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
 random_state=42)
```

In [66]:
```
#Thanks to Abhishek Reddy Y N for helping me with this API
from sklearn.metrics import mean_squared_error
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error

standard_scaler = StandardScaler().fit(X_train)
rescaled_X_train = standard_scaler.transform(X_train)
lin_model = LinearRegression()
lin_model.fit(rescaled_X_train, y_train)
pred = lin_model.predict(X_test)
error = np.sqrt(mean_squared_error(y_test,pred))
print(error)
```

```
9647.228373844928
```

In [67]:
```
print(pred)
```

```
[[-12142.88964776]
 [  -424.00229051]
 [-13970.30428738]
 ...
 [-13321.28148451]
 [ -1600.57611836]
 [-15315.29040964]]
```

In [68]:
```
def getUserData(id, frame):
    currentdata = data1[data1.userId==id]
    currentdata = currentdata.sample(frac=1).reset_index(drop=True)
    return   np.split(currentdata, [int(.6*len(currentdata)), int(.8*len
(currentdata))])
```

In [69]:
```
train, validate, test = getUserData(1,data1)
```

In [70]:
```
y_train = train[['demand']]
```

In [71]:
```
x_train = train.drop(['demand','userId','month','weekday','dateval'],axi
s=1)
```

In [72]:
```
x_test = test.drop(['demand','userId','month','weekday','dateval'],axis=
1)
```

In [73]:
```
y_test = test[['demand']]
```

In [74]:
```
rf.fit(x_train, y_train)
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: DataCon
versionWarning: A column-vector y was passed when a 1d array was expect
ed. Please change the shape of y to (n_samples,), for example using rav
el().
  """Entry point for launching an IPython kernel.
```

Out[74]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=Fa
lse)
```

In [78]:
```
ypredRf = rf.predict(x_test)
error = np.sqrt(mean_squared_error(y_test,ypredRf))
print(error)
```

```
1.3868311874833565
```

In [79]:
```
print(ypredRf)
```

```
[1.06838918 1.30402286 1.39387371 ... 1.25520045 1.26780932 1.26780932]
```

In [80]:
```
ypredRf
```

Out[80]:
```
array([1.06838918, 1.30402286, 1.39387371, ..., 1.25520045, 1.26780932,
       1.26780932])
```

In [81]:
```
def getUserData(id, frame):
    currentdata = data1[data1.userId==id]
    currentdata = currentdata.sample(frac=1).reset_index(drop=True)
    return   np.split(currentdata, [int(.6*len(currentdata)), int(.3*len
(currentdata))])
```

In [82]:
```
y_train = train[['demand']]
x_train = train.drop(['demand','userId','month','weekday','dateval'],axi
s=1)
x_test = test.drop(['demand','userId','month','weekday','dateval'],axis=
1)
y_test = test[['demand']]

lin_model = LinearRegression()
lin_model.fit(x_train, y_train)
pred = lin_model.predict(x_test)
error = np.sqrt(mean_squared_error(y_test,pred))
print(error)
```

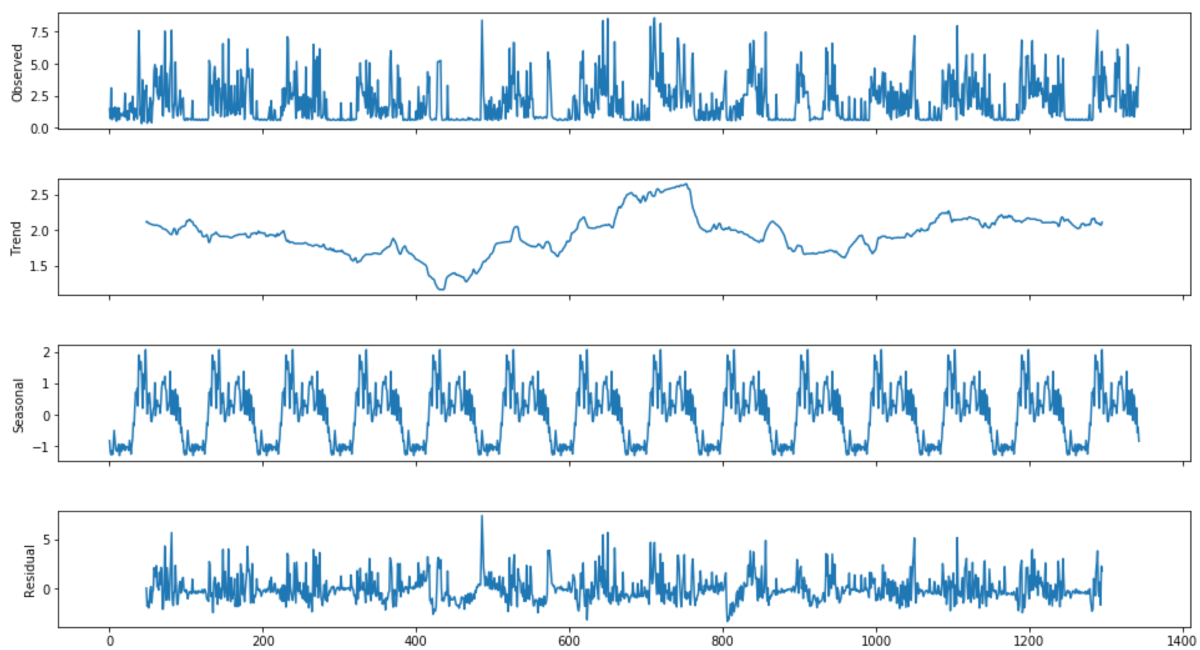
```
1.4074206220659675
```

In [83]:
```
print(pred)
```

```
[[1.14837053]
 [1.78300474]
 [1.40545103]
 ...
 [1.41310572]
 [1.49247904]
 [1.49247904]]
```

In [84]:
```
week_split = seasonal_decompose(data1.iloc[:672*2].demand, freq=96)
fig = plt.figure()
fig = week_split.plot()
fig.set_size_inches(15, 8)
```

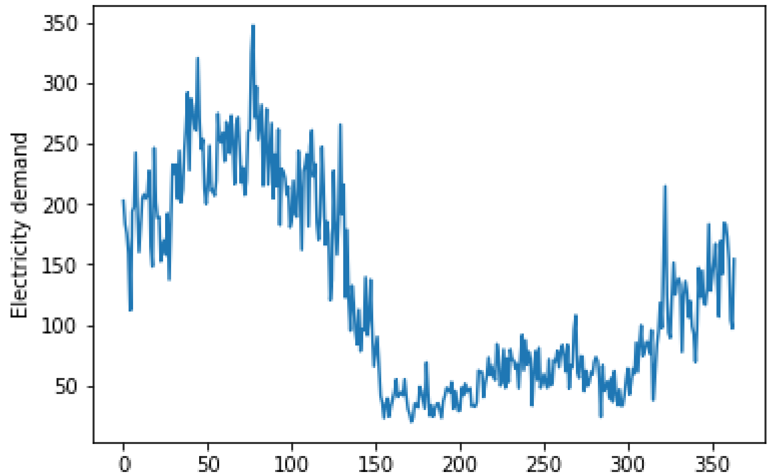
```
<Figure size 432x288 with 0 Axes>
```



In [85]:
```
#Electricity Consumption according to the daily time(96 rows)
df=list(data1.demand)
daily=[]
for i in range(0,34943,96):
    sum=0
    for j in range(i,i+96):
        sum = sum + df[j]
    daily.append(sum)
```

In [43]:
```
plt.plot(daily)
plt.ylabel('Electricity demand')
```

Out[43]: Text(0,0.5,'Electricity demand')



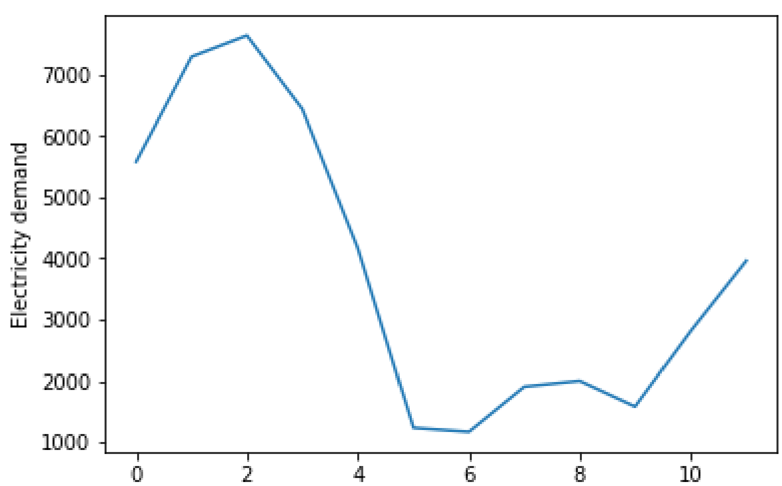
In [ ]:
```
np.corrcoef(data1[0:-1], data1[1:])
```

In [86]:
```
#sum of consumptions of each month
monthly=[]
for i in range(0,335,30):
    sum=0
    for j in range(i,i+30):
        sum = sum + daily[j]
    monthly.append(sum)
print(len(monthly))
```

```
12
```

In [87]:
```
plt.plot(monthly)
plt.ylabel('Electricity demand')
```

Out[87]: Text(0,0.5,'Electricity demand')



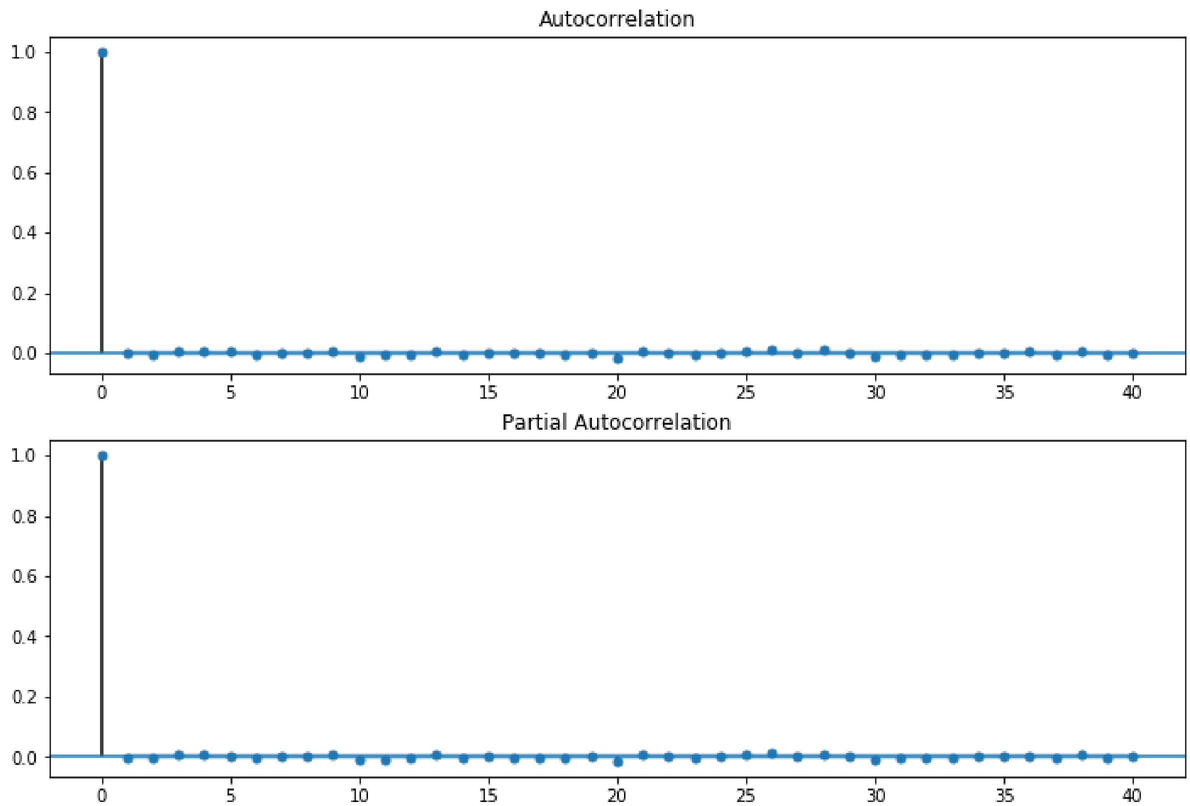
In [92]:
```
# Naive approach
meanSquaredError=mean_squared_error(data1[1344:1440], data1[1248:1344])
rootMeanSquaredError = sqrt(meanSquaredError)
print("RMSE:", rootMeanSquaredError)
```

```
RMSE: 36.43542520500079
```

In [95]:
```
# create training and testing variables
X_train, X_test = train_test_split(data1, test_size=0.0027398)
print (X_train.shape)
print (X_test.shape)
```

```
(34943, 7)
(96, 7)
```

In [96]:
```
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(X_train.demand[97:], lags=40, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(X_train.demand[97:], lags=40, ax=ax2)
```



In [98]:
```
seasonal_decompose(data1, freq=96)
```

Out[98]: <statsmodels.tsa.seasonal.DecomposeResult at 0x1c21c51908>

In [100]:
```
X_train=data1[:1344]
X_train = X_train.rename(columns={'0.65018': 'demand'})
X_train.describe()
```

Out[100]:

	demand	userId	dateval	month	dayofweek	weekday	hour
count	1344.000000	1344.0	1344.000000	1344.0	1344.000000	1344.000000	1344.000000
mean	1.948154	1.0	671.500000	12.0	3.000000	3.000000	11.500000
std	1.606076	0.0	388.123692	0.0	2.000744	2.000744	6.924763
min	0.318900	1.0	0.000000	12.0	0.000000	0.000000	0.000000
25%	0.670080	1.0	335.750000	12.0	1.000000	1.000000	5.750000
50%	1.437700	1.0	671.500000	12.0	3.000000	3.000000	11.500000
75%	2.632600	1.0	1007.250000	12.0	5.000000	5.000000	17.250000
max	8.570700	1.0	1343.000000	12.0	6.000000	6.000000	23.000000

In [101]:
```
p = range(0, 2)
d = range(0, 2)
q = range(0, 2)
params = list(it.product(p, d, q))
seasonal_params = [(x[0], x[1], x[2], 96) for x in list(it.product(p, d,
 q))]
```

In [103]:
```
 print(data1.corrwith(data1['demand']))
```

```
demand       1.000000
userId            NaN
dateval     -0.372315
month       -0.244674
dayofweek   -0.008742
weekday     -0.008742
hour         0.203713
dtype: float64
```

In [105]:
```
def test_stationarity(timeseries):
    print ('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value',
'#Lags Used','Number of Observations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput)
```

In [107]:
```
test_stationarity(X_train.demand)
```

```
Results of Dickey-Fuller Test:
Test Statistic                -8.598752e+00
p-value                        6.961221e-14
#Lags Used                     5.000000e+00
Number of Observations Used    1.338000e+03
Critical Value (1%)           -3.435247e+00
Critical Value (5%)           -2.863703e+00
Critical Value (10%)          -2.567921e+00
dtype: float64
```

In [108]:
```
p = range(0, 2)
d = range(0, 2)
q = range(0, 2)
params = list(it.product(p, d, q))
seasonal_params = [(x[0], x[1], x[2], 96) for x in list(it.product(p, d,
 q))]
```

In [109]:
```
warnings.filterwarnings("ignore")

for param in params:
    for param_seasonal in seasonal_params:
        try:
            model = sm.tsa.statespace.SARIMAX(X_train.demand,
                                              order=param,
                                              seasonal_order=param_seasona
l,
                                              enforce_stationarity=False,
                                              enforce_invertibility=False)
            results = model.fit()
            print('SARIMAX{}x{} - AIC:{}'.format(param, param_seasonal,
round(results.aic,2)))
        except:
            continue
```

```
SARIMAX(0, 0, 0)x(0, 0, 0, 96) - AIC:6301.24
SARIMAX(0, 0, 0)x(0, 0, 1, 96) - AIC:5443.55
SARIMAX(0, 0, 0)x(0, 1, 0, 96) - AIC:5207.97
SARIMAX(0, 0, 0)x(0, 1, 1, 96) - AIC:4241.75
SARIMAX(0, 0, 0)x(1, 0, 0, 96) - AIC:5012.58
SARIMAX(0, 0, 0)x(1, 0, 1, 96) - AIC:4657.78
SARIMAX(0, 0, 0)x(1, 1, 0, 96) - AIC:4509.88
SARIMAX(0, 0, 0)x(1, 1, 1, 96) - AIC:4243.2
SARIMAX(0, 0, 1)x(0, 0, 0, 96) - AIC:5532.7
SARIMAX(0, 0, 1)x(0, 0, 1, 96) - AIC:4956.34
SARIMAX(0, 0, 1)x(0, 1, 0, 96) - AIC:5075.23
SARIMAX(0, 0, 1)x(0, 1, 1, 96) - AIC:4063.45
SARIMAX(0, 0, 1)x(1, 0, 0, 96) - AIC:4783.16
SARIMAX(0, 0, 1)x(1, 0, 1, 96) - AIC:4474.6
SARIMAX(0, 0, 1)x(1, 1, 0, 96) - AIC:4347.16
SARIMAX(0, 0, 1)x(1, 1, 1, 96) - AIC:4117.11
SARIMAX(0, 1, 0)x(0, 0, 0, 96) - AIC:5038.38
SARIMAX(0, 1, 0)x(0, 0, 1, 96) - AIC:4649.74
SARIMAX(0, 1, 0)x(0, 1, 0, 96) - AIC:5573.91
SARIMAX(0, 1, 0)x(0, 1, 1, 96) - AIC:4465.14
SARIMAX(0, 1, 0)x(1, 0, 0, 96) - AIC:4654.01
SARIMAX(0, 1, 0)x(1, 0, 1, 96) - AIC:4651.73
SARIMAX(0, 1, 0)x(1, 1, 0, 96) - AIC:4769.49
SARIMAX(0, 1, 0)x(1, 1, 1, 96) - AIC:4505.45
SARIMAX(0, 1, 1)x(0, 0, 0, 96) - AIC:4693.65
SARIMAX(0, 1, 1)x(0, 0, 1, 96) - AIC:4330.99
SARIMAX(0, 1, 1)x(0, 1, 0, 96) - AIC:5158.21
SARIMAX(0, 1, 1)x(0, 1, 1, 96) - AIC:4130.53
SARIMAX(0, 1, 1)x(1, 0, 0, 96) - AIC:4338.09
SARIMAX(0, 1, 1)x(1, 0, 1, 96) - AIC:4332.89
SARIMAX(0, 1, 1)x(1, 1, 0, 96) - AIC:4427.55
SARIMAX(0, 1, 1)x(1, 1, 1, 96) - AIC:4181.39
SARIMAX(1, 0, 0)x(0, 0, 0, 96) - AIC:4906.44
SARIMAX(1, 0, 0)x(0, 0, 1, 96) - AIC:4530.67
SARIMAX(1, 0, 0)x(0, 1, 0, 96) - AIC:5069.47
SARIMAX(1, 0, 0)x(0, 1, 1, 96) - AIC:4054.26
SARIMAX(1, 0, 0)x(1, 0, 0, 96) - AIC:4529.62
SARIMAX(1, 0, 0)x(1, 0, 1, 96) - AIC:4462.69
SARIMAX(1, 0, 0)x(1, 1, 0, 96) - AIC:4339.79
SARIMAX(1, 0, 0)x(1, 1, 1, 96) - AIC:4103.68
SARIMAX(1, 0, 1)x(0, 0, 0, 96) - AIC:4687.89
SARIMAX(1, 0, 1)x(0, 0, 1, 96) - AIC:4323.2
SARIMAX(1, 0, 1)x(0, 1, 0, 96) - AIC:5066.75
SARIMAX(1, 0, 1)x(0, 1, 1, 96) - AIC:4051.88
SARIMAX(1, 0, 1)x(1, 0, 0, 96) - AIC:4327.32
SARIMAX(1, 0, 1)x(1, 0, 1, 96) - AIC:4325.15
SARIMAX(1, 0, 1)x(1, 1, 0, 96) - AIC:4341.66
SARIMAX(1, 0, 1)x(1, 1, 1, 96) - AIC:4100.01
SARIMAX(1, 1, 0)x(0, 0, 0, 96) - AIC:4899.99
SARIMAX(1, 1, 0)x(0, 0, 1, 96) - AIC:4516.33
SARIMAX(1, 1, 0)x(0, 1, 0, 96) - AIC:5413.61
SARIMAX(1, 1, 0)x(0, 1, 1, 96) - AIC:4346.17
SARIMAX(1, 1, 0)x(1, 0, 0, 96) - AIC:4516.28
SARIMAX(1, 1, 0)x(1, 0, 1, 96) - AIC:4518.27
SARIMAX(1, 1, 0)x(1, 1, 0, 96) - AIC:4647.38
SARIMAX(1, 1, 0)x(1, 1, 1, 96) - AIC:4388.42
SARIMAX(1, 1, 1)x(0, 0, 0, 96) - AIC:4642.88
```

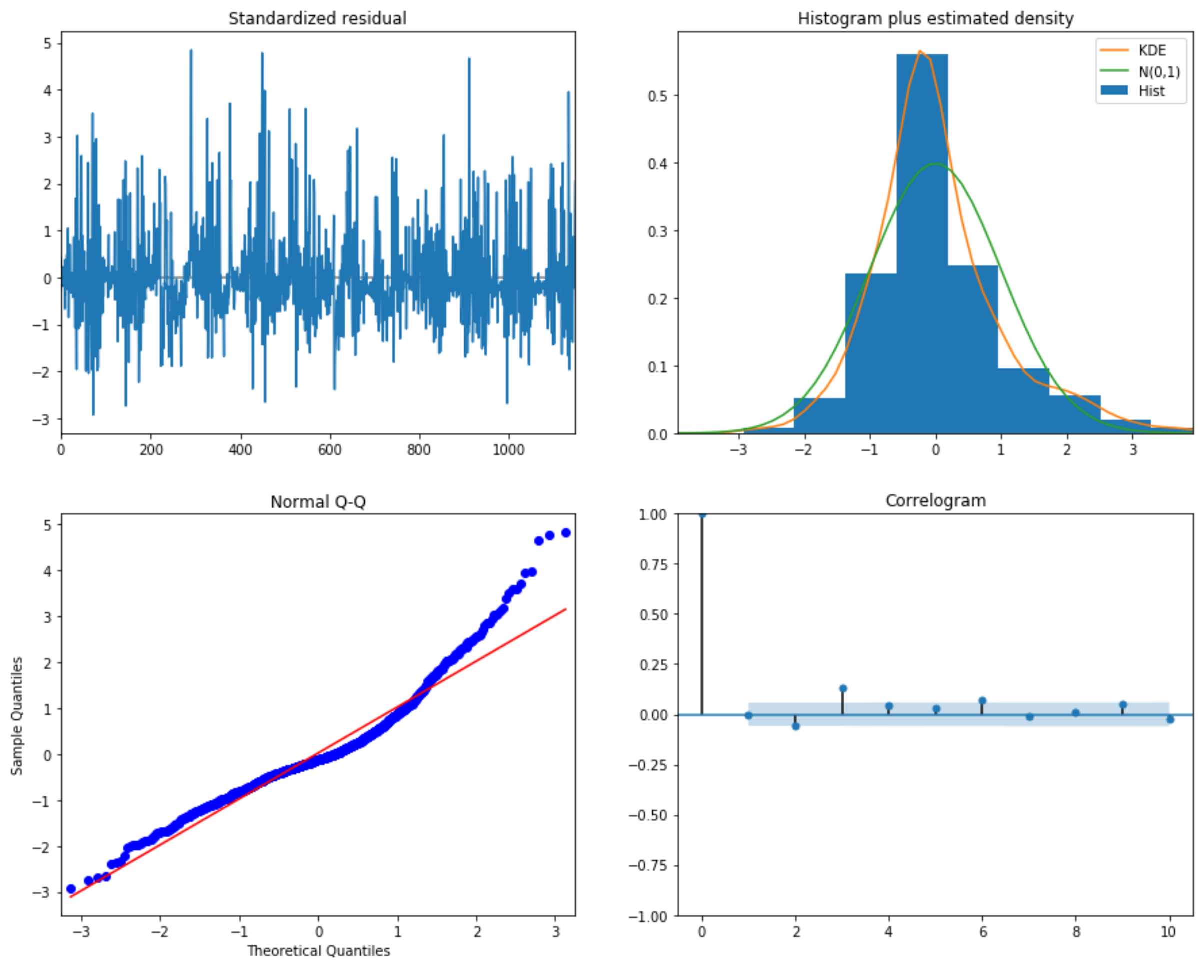
```
SARIMAX(1, 1, 1)x(0, 0, 1, 96) - AIC:4290.45
SARIMAX(1, 1, 1)x(0, 1, 0, 96) - AIC:5069.43
SARIMAX(1, 1, 1)x(0, 1, 1, 96) - AIC:4050.39
SARIMAX(1, 1, 1)x(1, 0, 0, 96) - AIC:4293.35
SARIMAX(1, 1, 1)x(1, 0, 1, 96) - AIC:4292.43
SARIMAX(1, 1, 1)x(1, 1, 0, 96) - AIC:4341.71
SARIMAX(1, 1, 1)x(1, 1, 1, 96) - AIC:4099.75
```

In [110]:

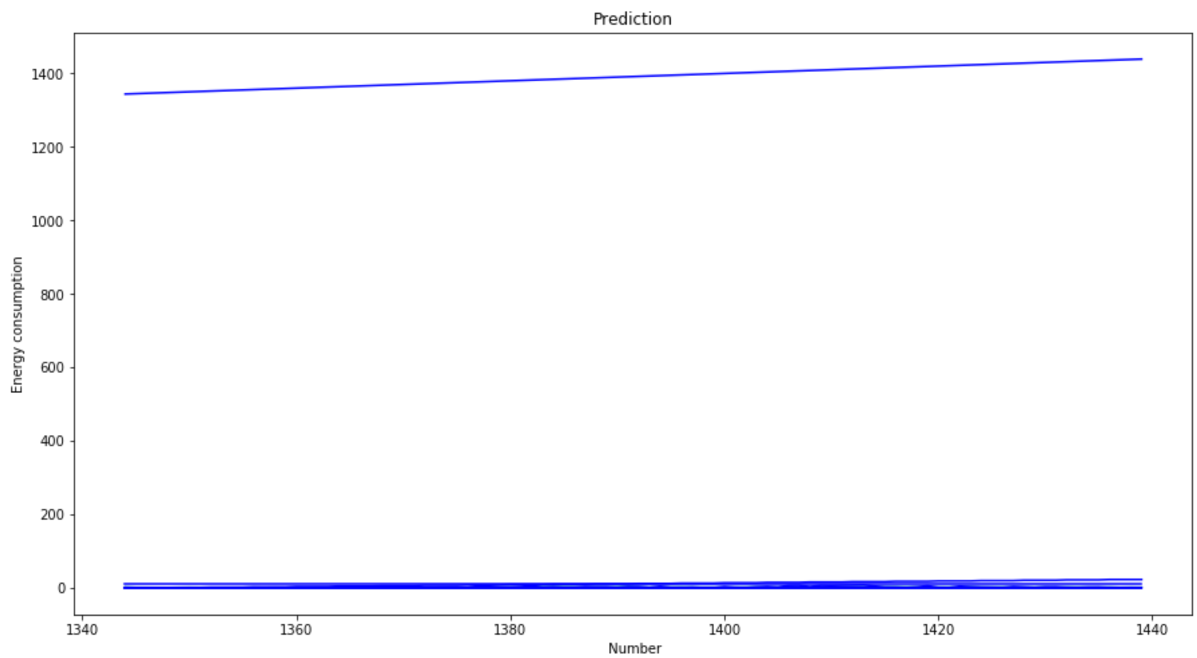
```
SARIMA_model = sm.tsa.statespace.SARIMAX(X_train.demand,
                                order=(1, 1, 1),
                                seasonal_order=(0, 1, 1, 96),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results = SARIMA_model.fit()

print(results.summary().tables[1])
```

```
==============================================================================
=======
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-------
ar.L1          0.3702      0.022     17.106      0.000       0.328
  0.413
ma.L1         -0.9859      0.005   -187.622      0.000      -0.996
-0.976
ma.S.L96      -1.0000    231.002     -0.004      0.997    -453.756
451.756
sigma2         1.6777    387.579      0.004      0.997    -757.964
761.319
==============================================================================
=======
```

In [111]:
```
results.plot_diagnostics(figsize=(15, 12))
plt.show()
```



In [118]:
```
pred_uc = results.get_forecast(steps=96)
plt.rcParams['figure.figsize'] = [15, 8]
plt.plot(pred_uc.predicted_mean)
plt.plot(data1[1344:1440], color='blue')
plt.xlabel(' Number')
plt.ylabel('Energy consumption')
plt.title('Prediction')
plt.show()
```



In [114]:
```
pred_uc.predicted_mean = pred_uc.predicted_mean.astype(int)
```

In [116]:
```
meanSquaredError=mean_squared_error(data1[1344:1440].demand, pred_uc.pre
dicted_mean)
rootMeanSquaredError = sqrt(meanSquaredError)
print("RMSE:", rootMeanSquaredError)
```

```
RMSE: 1.7452254974474286
```

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt
spearman_correlation = data1.corr(method='spearman')
pick_columns=spearman_correlation.nlargest(10, 'demand').index
correlationmap = np.corrcoef(data1[pick_columns].values.T)
sns.set(font_scale=1.0)
heatmap = sns.heatmap(correlationmap, cbar=True, annot=True, square=True
 , fmt='.2f', yticklabels=data1.values, xticklabels=data1.values)
plt.show()
```