Energy Consumption

Approach :

1. **Combine all blocks into a single dataframe- keeping on relevant columns.**
2. **Use day-level energy consumption data per household to normalize data for inconsistent household count**
3. **Explore relationships between weather conditions and energy consumptions. Create clusters for the weather data- using which we can add weather identifiers to day-level data**
4. **Add UK holidays data to the day level data as an indicator.**
5. **Fit an ARIMA model**

```
    i) ACF, PACF
    ii) Explore Seasonal Decomposition
    iii) Modelling
```

6. **Fit an LSTM model**

Daily Energy Data Preparation

Importing Libraries

In [1]:

```
#!pip install pmdarima
```

In [2]:

```
import pandas as pd
import numpy as np
from pandas import datetime
from matplotlib import pyplot as plt
import os

from statsmodels.tsa.arima_model import ARIMA
from matplotlib import pyplot
from pandas.tools.plotting import autocorrelation_plot

#from pyramid.arima import auto_arima
#from pmdarima.arima import auto_arima
import pyflux as pf
from sklearn.cluster import KMeans
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import statsmodels.api as sm
from statsmodels.tsa.statespace.sarimax import SARIMAX

import math

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
```

```
Using TensorFlow backend.
```

Energy Data

> **We are predicting for energy demand in the future- therefore we are taking only energy sum i.e.**

> **total energy use per day for a given household.**

In [3]:

```
# Combining all blocks
for num in range(0,112):
    df = pd.read_csv("../input/daily_dataset/daily_dataset/block_"+str(num)+".csv")
    df = df[['day','LCLid','energy_sum']]
    df.reset_index()
    df.to_csv("hc_"+str(num)+".csv")

fout= open("energy.csv","a")
# first file:
for line in open("hc_0.csv"):
    fout.write(line)
# now the rest:
for num in range(0,112):
    f = open("hc_"+str(num)+".csv")
    f.readline() # skip the header
    for line in f:
         fout.write(line)
    f.close()
fout.close()
```

Energy at Day Level

In [4]:

```
energy = pd.read_csv('energy.csv')
len(energy)
```

Out[4]:

```
3536007
```

House Count

> **In the dataset we see that the number of households for which energy data was collected across different days are different. This is probably due to the gradually increasing adoption of smart meters in London. This could lead to false interpretation that the energy for a particular day might be high when it could be that the data was only collected for more number of houses. We will look at the house count for each day.**

In [5]:

```
housecount = energy.groupby('day')[['LCLid']].nunique()
housecount.head(4)
```

Out[5]:

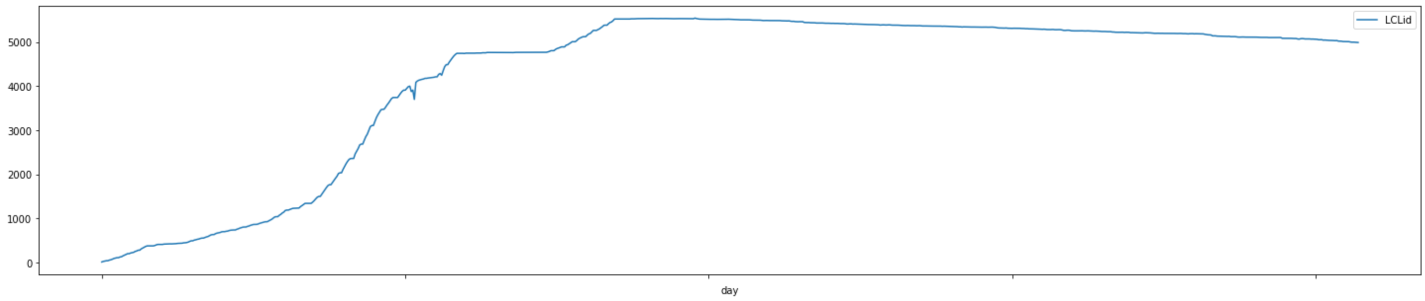
	LCLid
day	
2011-11-23	13
2011-11-24	25
2011-11-25	32
2011-11-26	41

In [6]:

```
housecount.plot(figsize=(25,5))
```

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f011f84d940>
```



Normalization across households

> **The data collection across households are inconsistent- therefore we will be using *energy per household* as the target to predict rather than energy alone. This is an optional step as we can also predict for energy sum as whole for each household. However there are quite a lot of unique households for which we have to repeat the exercise and our ultimate goal is to predict overall consumption forecast and not at household level.**
> **This also means that since household level is removed, we are not looking into the ACORN details which is available at household level**

In [7]:

```
energy = energy.groupby('day')[['energy_sum']].sum()
energy = energy.merge(housecount, on = ['day'])
energy = energy.reset_index()
```

In [8]:

```
energy.count()
```

Out[8]:

```
day           829
energy_sum    829
LCLid         829
dtype: int64
```

In [9]:

```
energy.day = pd.to_datetime(energy.day,format='%Y-%m-%d').dt.date
```

In [10]:

```
energy['avg_energy'] =  energy['energy_sum']/energy['LCLid']
print("Starting Point of Data at Day Level",min(energy.day))
print("Ending Point of Data at Day Level",max(energy.day))
```

```
Starting Point of Data at Day Level 2011-11-23
Ending Point of Data at Day Level 2014-02-28
```

In [11]:

```
energy.describe()
```

Out[11]:

	energy_sum	LCLid	avg_energy
count	829.000000	829.000000	829.000000
mean	43535.325676	4234.539204	10.491862
std	20550.594031	1789.994799	1.902513
min	90.385000	13.000000	0.211766
25%	34665.436003	4084.000000	8.676955

	energy_sum	LCLid	avg_energy
50%	46641.760997	5138.000000	10.516985
75%	59755.616996	5369.000000	12.000690
max	84156.135002	5541.000000	15.964434

Weather Information

Daily level weather information is taken using darksky api in the dataset

In [12]:

```
weather = pd.read_csv('../input/weather_daily_darksky.csv')
weather.head(4)
```

Out[12]:

	temperatureMax	temperatureMaxTime	windBearing	icon	dewPoint	temperatureMinTime	cloudCover	windSpeed	pres
0	11.96	2011-11-11 23:00:00	123	fog	9.40	2011-11-11 07:00:00	0.79	3.88	10
1	8.59	2011-12-11 14:00:00	198	partly-cloudy-day	4.49	2011-12-11 01:00:00	0.56	3.94	10
2	10.33	2011-12-27 02:00:00	225	partly-cloudy-day	5.47	2011-12-27 23:00:00	0.85	3.54	10
3	8.07	2011-12-02 23:00:00	232	wind	3.69	2011-12-02 07:00:00	0.32	3.00	10

In [13]:

```
weather.describe()
```

Out[13]:

	temperatureMax	windBearing	dewPoint	cloudCover	windSpeed	pressure	apparentTemperatureHigh	visibility
count	882.000000	882.000000	882.000000	881.000000	882.000000	882.000000	882.000000	882.00000
mean	13.660113	195.702948	6.530034	0.477605	3.581803	1014.127540	12.723866	11.16714
std	6.182744	89.340783	4.830875	0.193514	1.694007	11.073038	7.279168	2.46610
min	-0.060000	0.000000	-7.840000	0.000000	0.200000	979.250000	-6.460000	1.48000
25%	9.502500	120.500000	3.180000	0.350000	2.370000	1007.435000	7.032500	10.32750
50%	12.625000	219.000000	6.380000	0.470000	3.440000	1014.615000	12.470000	11.97000
75%	17.920000	255.000000	10.057500	0.600000	4.577500	1021.755000	17.910000	12.83000
max	32.400000	359.000000	17.770000	1.000000	9.960000	1040.920000	32.420000	15.34000

In [14]:

```
weather['day']=  pd.to_datetime(weather['time']) # day is given as timestamp
weather['day']=  pd.to_datetime(weather['day'],format='%Y%m%d').dt.date
# selecting numeric variables
weather = weather[['temperatureMax', 'windBearing', 'dewPoint', 'cloudCover', 'windSpeed
',
```

```
        'pressure', 'apparentTemperatureHigh', 'visibility', 'humidity',
        'apparentTemperatureLow', 'apparentTemperatureMax', 'uvIndex',
        'temperatureLow', 'temperatureMin', 'temperatureHigh',
        'apparentTemperatureMin', 'moonPhase','day']]
weather = weather.dropna()
```

Relationship of weather conditions with electricity consumption

In [15]:

```
weather_energy =  energy.merge(weather,on='day')
weather_energy.head(2)
```

Out[15]:

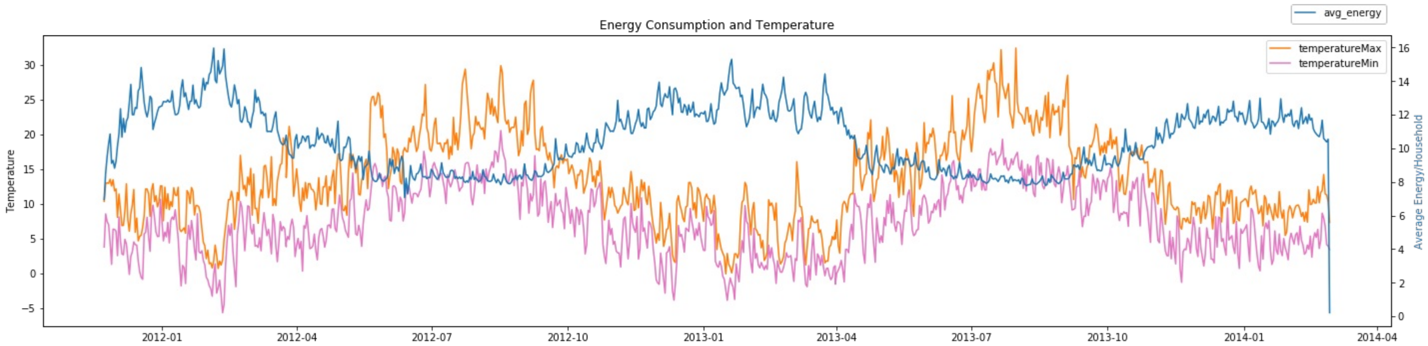
	day	energy_sum	LCLid	avg_energy	temperatureMax	windBearing	dewPoint	cloudCover	windSpeed	pressure	appar
0	2011-11-23	90.385	13	6.952692	10.36	229	6.29	0.36	2.04	1027.12	
1	2011-11-24	213.412	25	8.536480	12.93	204	8.56	0.41	4.04	1027.22	

1. Temperature

> **We can see that energy and temperature have an inverse relationship-we can see the peaks in one appearing with troughs in the other. This confirms the business intuition that during low temperature, it is likely that the energy consumption through heaters etc. increases.**

In [16]:

```
fig, ax1 = plt.subplots(figsize = (20,5))
ax1.plot(weather_energy.day, weather_energy.temperatureMax, color = 'tab:orange')
ax1.plot(weather_energy.day, weather_energy.temperatureMin, color = 'tab:pink')
ax1.set_ylabel('Temperature')
ax1.legend()
ax2 = ax1.twinx()
ax2.plot(weather_energy.day,weather_energy.avg_energy,color = 'tab:blue')
ax2.set_ylabel('Average Energy/Household',color = 'tab:blue')
ax2.legend(bbox_to_anchor=(0.0, 1.02, 1.0, 0.102))
plt.title('Energy Consumption and Temperature')
fig.tight_layout()
plt.show()
```



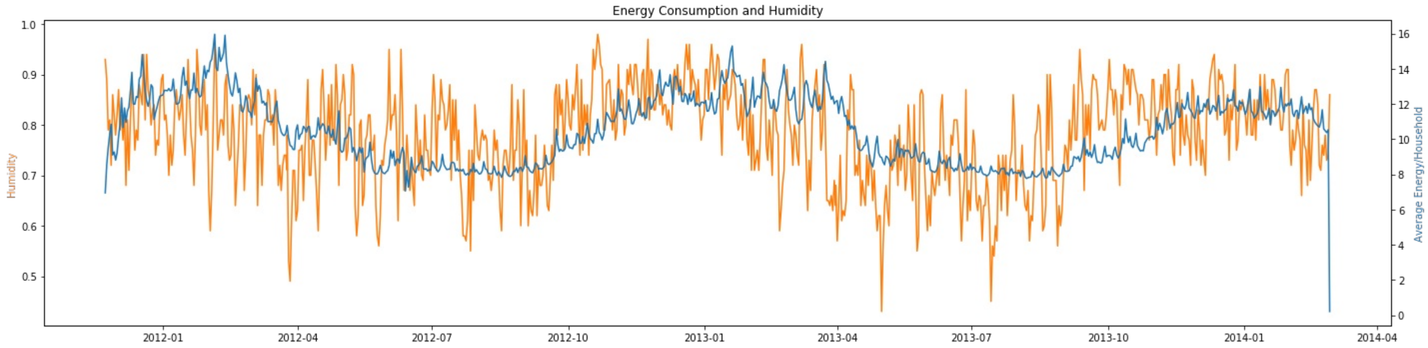
2. Humidity

> **Humidity and the average consumption of energy seems to have the same trend.**

In [17]:

```
fig, ax1 = plt.subplots(figsize = (20,5))
```

```
ax1.plot(weather_energy.day, weather_energy.humidity, color = 'tab:orange')
ax1.set_ylabel('Humidity',color = 'tab:orange')
ax2 = ax1.twinx()
ax2.plot(weather_energy.day,weather_energy.avg_energy,color = 'tab:blue')
ax2.set_ylabel('Average Energy/Household',color = 'tab:blue')
plt.title('Energy Consumption and Humidity')
fig.tight_layout()
plt.show()
```

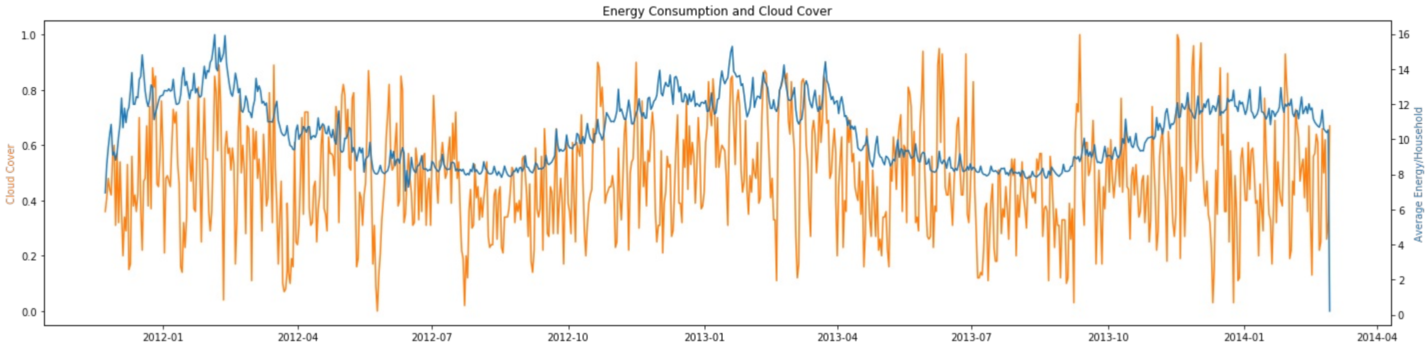


3. Cloud Cover

> **The cloud cover value seems to be following the same pattern as the energy consumption.**

In [18]:

```
fig, ax1 = plt.subplots(figsize = (20,5))
ax1.plot(weather_energy.day, weather_energy.cloudCover, color = 'tab:orange')
ax1.set_ylabel('Cloud Cover',color = 'tab:orange')
ax2 = ax1.twinx()
ax2.plot(weather_energy.day,weather_energy.avg_energy,color = 'tab:blue')
ax2.set_ylabel('Average Energy/Household',color = 'tab:blue')
plt.title('Energy Consumption and Cloud Cover')
fig.tight_layout()
plt.show()
```

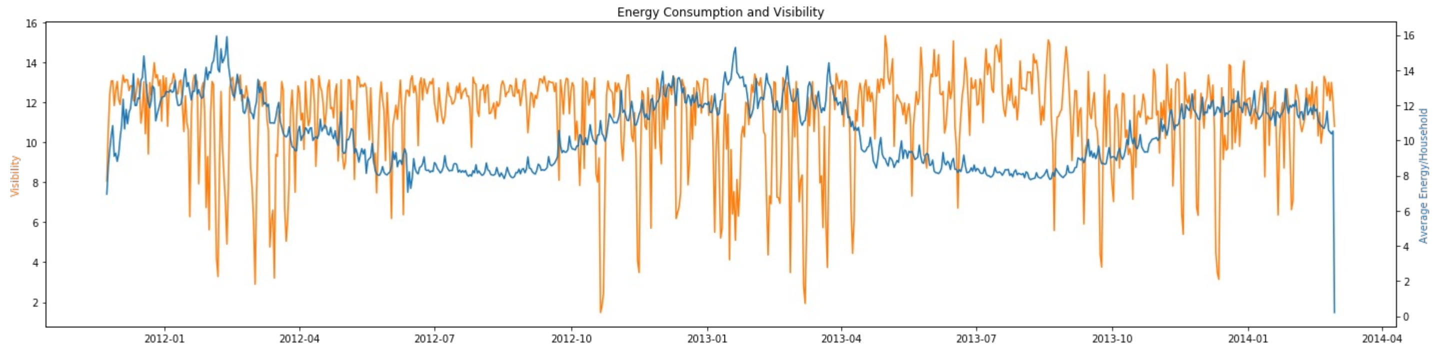


4. Visibility

> **The visibility factor does not seem to affect energy consumption at all- since visibility is most likely an outdoors factor, it is unlikely that it's increase or decrease affects energy consumption within a household.**

In [19]:

```
fig, ax1 = plt.subplots(figsize = (20,5))
ax1.plot(weather_energy.day, weather_energy.visibility, color = 'tab:orange')
ax1.set_ylabel('Visibility',color = 'tab:orange')
ax2 = ax1.twinx()
ax2.plot(weather_energy.day,weather_energy.avg_energy,color = 'tab:blue')
ax2.set_ylabel('Average Energy/Household',color = 'tab:blue')
plt.title('Energy Consumption and Visibility')
fig.tight_layout()
plt.show()
```

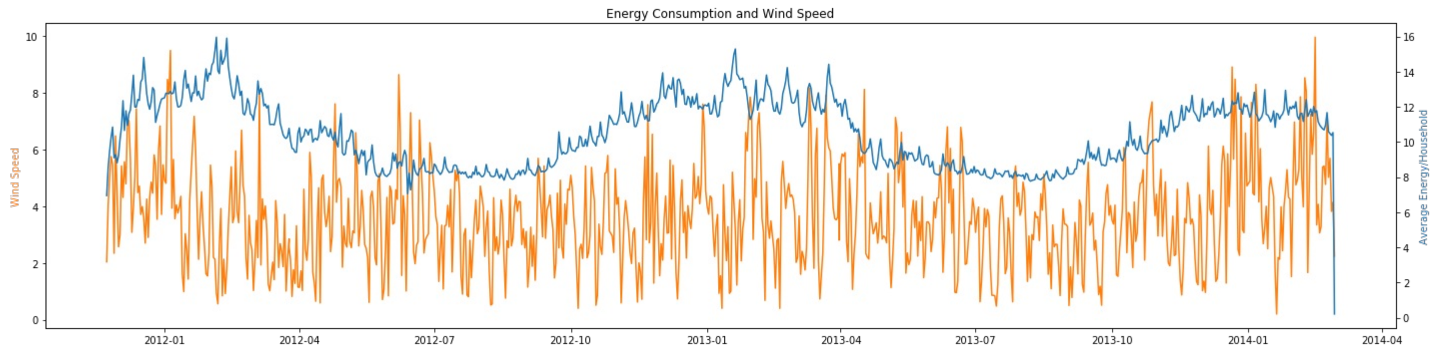


5. Wind Speed

> **Like visibility, wind speed seems to be an outdoors factor which does not affect in the energy consumption as such.**

In [20]:

```
fig, ax1 = plt.subplots(figsize = (20,5))
ax1.plot(weather_energy.day, weather_energy.windSpeed, color = 'tab:orange')
ax1.set_ylabel('Wind Speed',color = 'tab:orange')
ax2 = ax1.twinx()
ax2.plot(weather_energy.day,weather_energy.avg_energy,color = 'tab:blue')
ax2.set_ylabel('Average Energy/Household',color = 'tab:blue')
plt.title('Energy Consumption and Wind Speed')
fig.tight_layout()
plt.show()
```

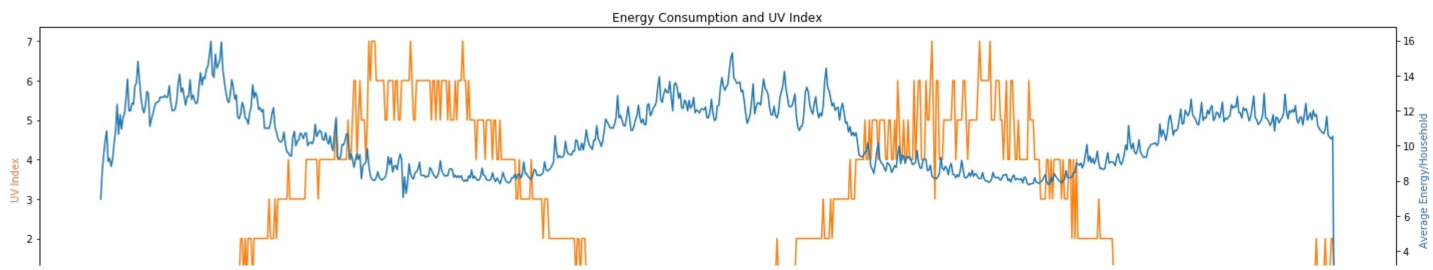


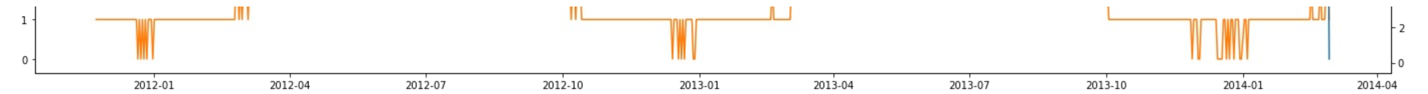
6. UV Index

> **The UV index has an inverse relationship with energy consumption- why?**

In [21]:

```
fig, ax1 = plt.subplots(figsize = (20,5))
ax1.plot(weather_energy.day, weather_energy.uvIndex, color = 'tab:orange')
ax1.set_ylabel('UV Index',color = 'tab:orange')
ax2 = ax1.twinx()
ax2.plot(weather_energy.day,weather_energy.avg_energy,color = 'tab:blue')
ax2.set_ylabel('Average Energy/Household',color = 'tab:blue')
plt.title('Energy Consumption and UV Index')
fig.tight_layout()
plt.show()
```



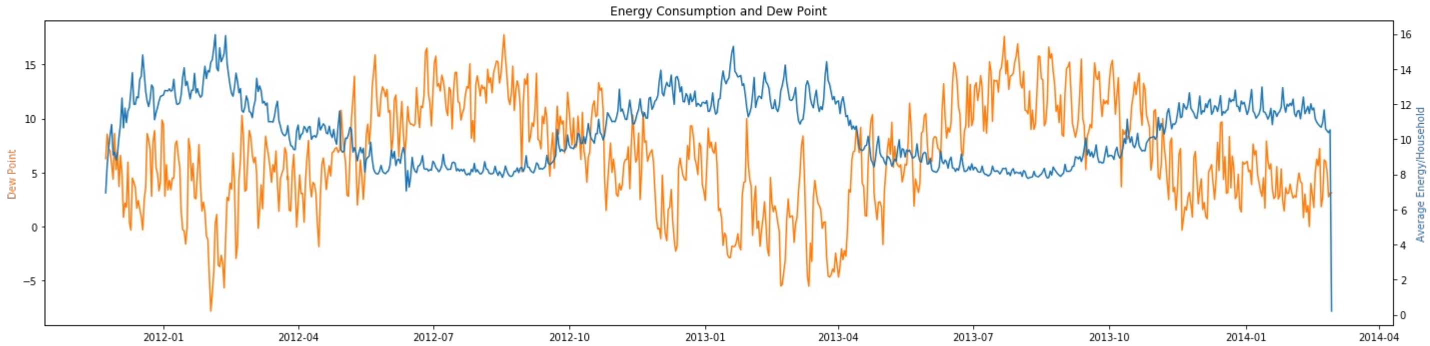


7. dewPoint

> **Dew Point- is a function of humidity and temperature therefore it displays similar relation to energy consumption.**

In [22]:

```
fig, ax1 = plt.subplots(figsize = (20,5))
ax1.plot(weather_energy.day, weather_energy.dewPoint, color = 'tab:orange')
ax1.set_ylabel('Dew Point',color = 'tab:orange')
ax2 = ax1.twinx()
ax2.plot(weather_energy.day,weather_energy.avg_energy,color = 'tab:blue')
ax2.set_ylabel('Average Energy/Household',color = 'tab:blue')
plt.title('Energy Consumption and Dew Point')
fig.tight_layout()
plt.show()
```



Correlation between Weather Variables and Energy Consumption

- **Energy has high positive correlation with humidity and high negative correlation with temperature.**
- **Dew Point, UV Index display multicollinearity with Temperature, hence discarded**
- **Cloud Cover and Visibility display multicollinearity with Humidity, hence discarded**
- **Pressure and Moon Phase have minimal correlation with Energy, hence discarded**
- **Wind Speed has low correlation with energy but does not show multicollinearity**

In [23]:

```
cor_matrix = weather_energy[['avg_energy','temperatureMax','dewPoint', 'cloudCover', 'wi
ndSpeed','pressure', 'visibility', 'humidity','uvIndex', 'moonPhase']].corr()
cor_matrix
```

Out[23]:

	avg_energy	temperatureMax	dewPoint	cloudCover	windSpeed	pressure	visibility	humidity	uvIndex	mo
avg_energy	1.000000	-0.846965	-0.755901	0.241779	0.149624	-0.028851	-0.246404	0.361237	-0.733171	
temperatureMax	-0.846965	1.000000	0.865038	-0.333409	-0.153602	0.118933	0.259108	-0.404899	0.696497	
dewPoint	-0.755901	0.865038	1.000000	-0.025207	-0.092212	-0.028121	0.042633	0.055514	0.486692	
cloudCover	0.241779	-0.333409	-0.025207	1.000000	0.170235	-0.101079	-0.330177	0.480056	-0.248695	
windSpeed	0.149624	-0.153602	-0.092212	0.170235	1.000000	-0.344354	0.281088	-0.042391	-0.152634	
pressure	-0.028851	0.118933	-0.028121	-0.101079	-0.344354	1.000000	-0.012508	-0.250941	0.100774	

	avg_energy	temperatureMax	dewPoint	cloudCover	windSpeed	pressure	visibility	humidity	uvIndex	mo
visibility	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	-0.012508	1.000000	-0.578130	0.240485	
humidity	0.361237	-0.404899	0.055514	0.480056	-0.042391	-0.250941	-0.578130	1.000000	-0.533919	
uvIndex	-0.733171	0.696497	0.486692	-0.248695	-0.152634	0.100774	0.240485	-0.533919	1.000000	
moonPhase	-0.031716	0.003636	-0.008239	-0.062126	-0.023273	0.038462	0.062813	-0.013997	0.012833	

Creating Weather Clusters

> **The weather information has a lot of variables- which might not all be useful. We will attempt to create weather clusters to see if we can define a weather of the day based on the granular weather data like temperature, precipitation etc.**

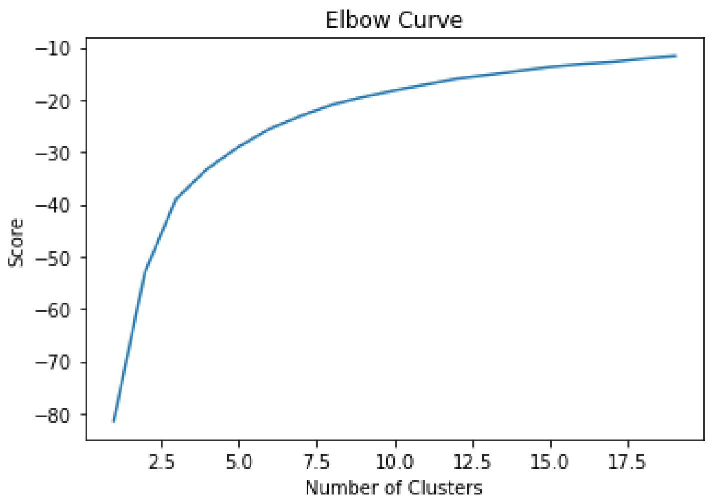
In [24]:

```
#scaling
scaler = MinMaxScaler()
weather_scaled = scaler.fit_transform(weather_energy[['temperatureMax','humidity','windSp
eed']])
```

In [25]:

```
# optimum K
Nc = range(1, 20)
kmeans = [KMeans(n_clusters=i) for i in Nc]
kmeans

score = [kmeans[i].fit(weather_scaled).score(weather_scaled) for i in range(len(kmeans))
]
score
plt.plot(Nc,score)
plt.xlabel('Number of Clusters')
plt.ylabel('Score')
plt.title('Elbow Curve')
plt.show()
```



In [26]:

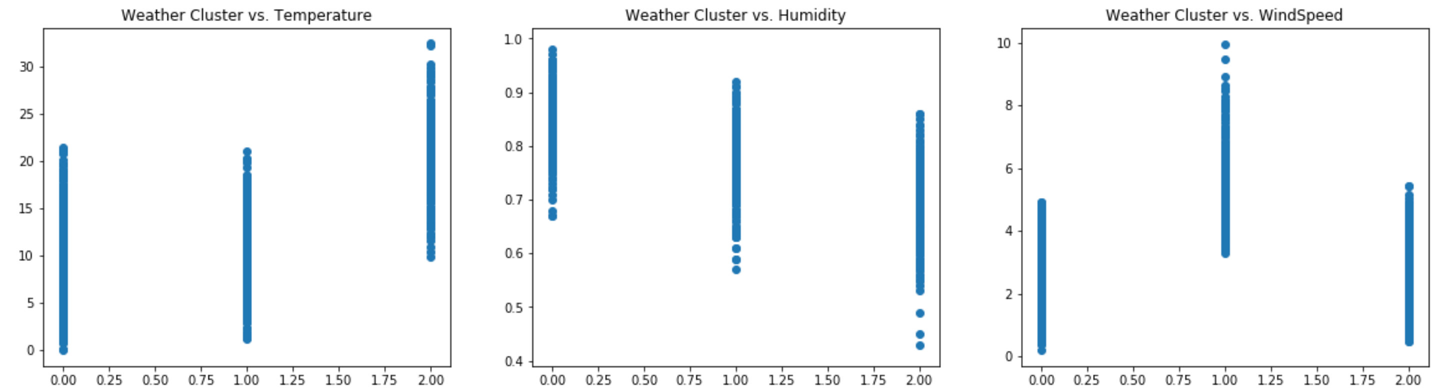
```
kmeans = KMeans(n_clusters=3, max_iter=600, algorithm = 'auto')
kmeans.fit(weather_scaled)
weather_energy['weather_cluster'] = kmeans.labels_
```

In [27]:

```
# Cluster Relationships with weather variables
```

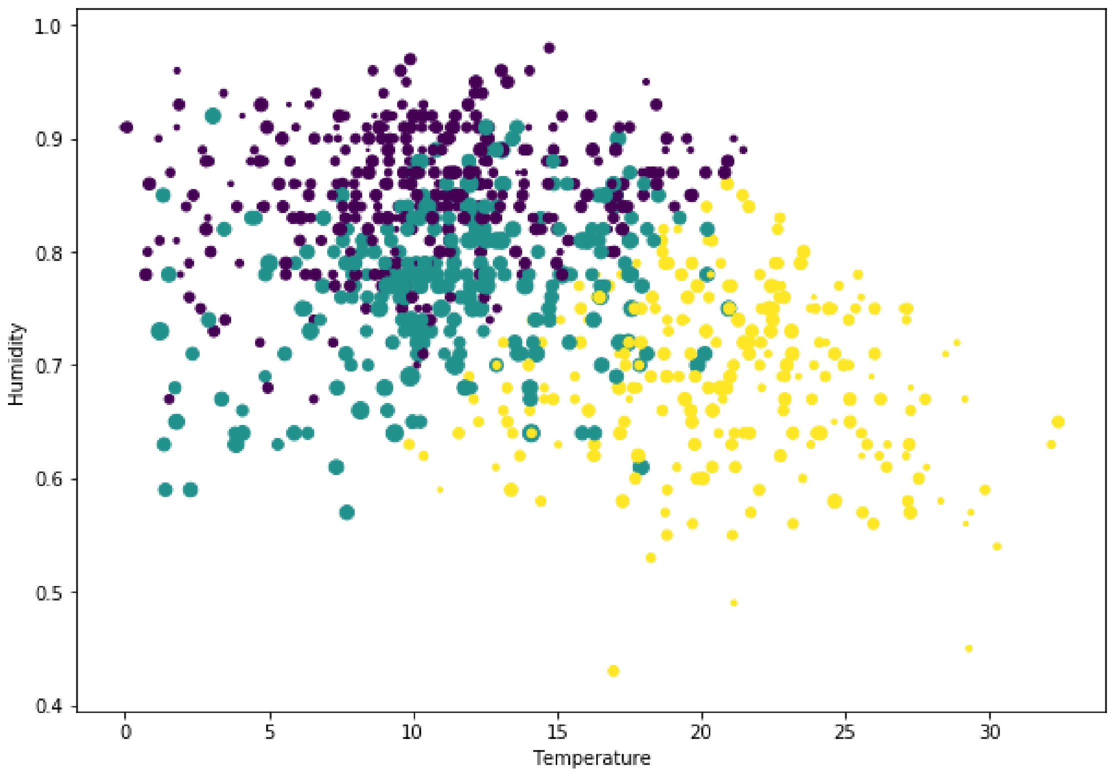
```
plt.figure(figsize=(20,5))
plt.subplot(1, 3, 1)
plt.scatter(weather_energy.weather_cluster,weather_energy.temperatureMax)
plt.title('Weather Cluster vs. Temperature')
plt.subplot(1, 3, 2)
plt.scatter(weather_energy.weather_cluster,weather_energy.humidity)
plt.title('Weather Cluster vs. Humidity')
plt.subplot(1, 3, 3)
plt.scatter(weather_energy.weather_cluster,weather_energy.windSpeed)
plt.title('Weather Cluster vs. WindSpeed')

plt.show()
# put this in a loop
```



In [28]:

```
fig, ax1 = plt.subplots(figsize = (10,7))
ax1.scatter(weather_energy.temperatureMax,
            weather_energy.humidity,
            s = weather_energy.windSpeed*10,
            c = weather_energy.weather_cluster)
ax1.set_xlabel('Temperature')
ax1.set_ylabel('Humidity')
plt.show()
```



UK Bank Holidays

In [29]:

```
holiday = pd.read_csv('../input/uk_bank_holidays.csv')
```

```
holiday['Bank holidays'] = pd.to_datetime(holiday['Bank holidays'],format='%Y-%m-%d').dt
.date
holiday.head(4)
```

Out[29]:

	Bank holidays	Type
0	2012-12-26	Boxing Day
1	2012-12-25	Christmas Day
2	2012-08-27	Summer bank holiday
3	2012-05-06	Queen?s Diamond Jubilee (extra bank holiday)

Creating a holiday indicator on weather data

In [30]:

```
weather_energy = weather_energy.merge(holiday, left_on = 'day',right_on = 'Bank holidays
',how = 'left')
weather_energy['holiday_ind'] = np.where(weather_energy['Bank holidays'].isna(),0,1)
```

ARIMAX

In [31]:

```
weather_energy['Year'] = pd.DatetimeIndex(weather_energy['day']).year
weather_energy['Month'] = pd.DatetimeIndex(weather_energy['day']).month
weather_energy.set_index(['day'],inplace=True)
```

Subset for required columns and 70-30 train-test split

In [32]:

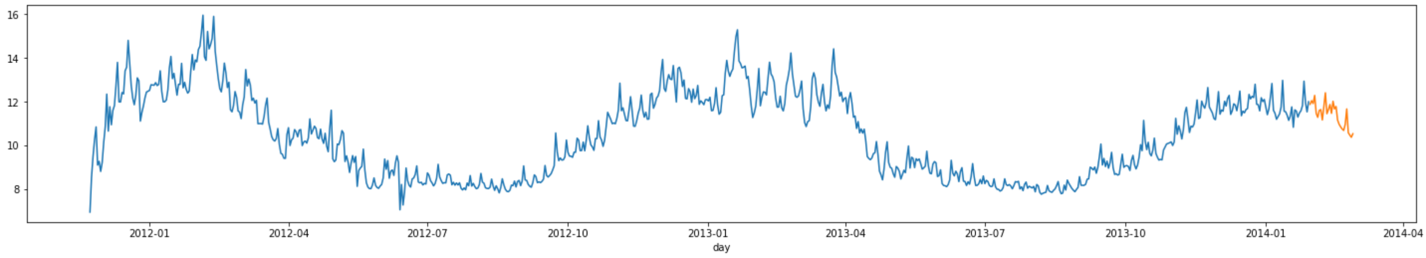
```
model_data = weather_energy[['avg_energy','weather_cluster','holiday_ind']]
# train = model_data.iloc[0:round(len(model_data)*0.90)]
# test = model_data.iloc[len(train)-1:]
train = model_data.iloc[0:(len(model_data)-30)]
test = model_data.iloc[len(train):(len(model_data)-1)]
```

In [33]:

```
train['avg_energy'].plot(figsize=(25,4))
test['avg_energy'].plot(figsize=(25,4))
```

Out[33]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f0111ad79e8>



In [34]:

```
test.head(1)
```

Out[34]:

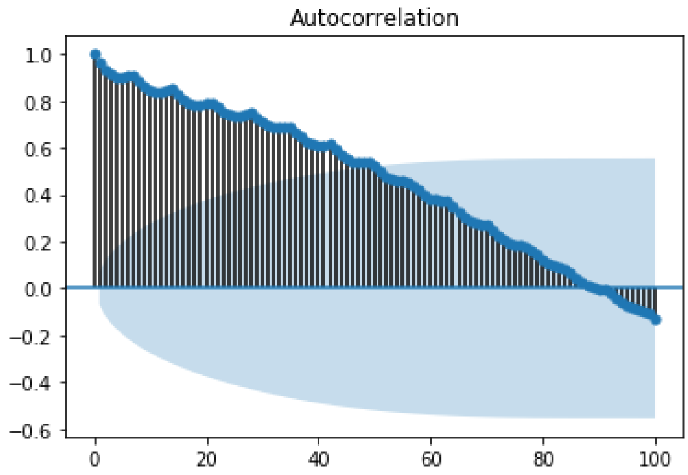
	avg_energy	weather_cluster	holiday_ind
day			

	avg_energy	weather_cluster	holiday_ind
2014-01-30	11.886982	0	0

ACF PACF

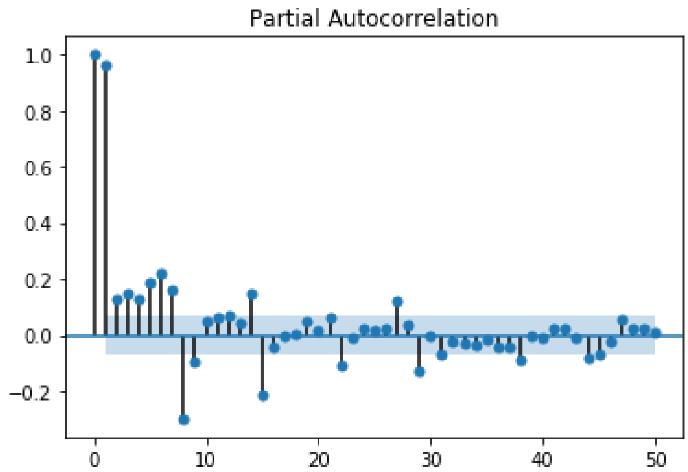
In [35]:

```
plot_acf(train.avg_energy,lags=100)
plt.show()
```



In [36]:

```
plot_pacf(train.avg_energy,lags=50)
plt.show()
```



Autocorrelation plot shows gradual decay while Partial AutoCorrelation shows that there is a sharp drop after 1st lag. This means that most of the higher-order autocorrelations are effectively explained by the k = 1 lag. Therefore, the series displays AR 'signature'

Dickey Fuller's Test

> **p is greater than 0.05 therefore the data is not stationary. After differencing, p < 0.05.**

In [37]:

```
t = sm.tsa.adfuller(train.avg_energy, autolag='AIC')
pd.Series(t[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
```

Out[37]:

```
Test Statistic                  -1.872794
p-value                          0.344966
#Lags Used                      21.000000
Number of Observations Used    776.000000
dtype: float64
```

```
dtype: float64
```

In [38]:

```python
# function for differencing
def difference(dataset, interval):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset.iloc[i] - dataset.iloc[i - interval]
        diff.append(value)
    return diff
```

In [39]:

```python
t  = sm.tsa.adfuller(difference(train.avg_energy,1), autolag='AIC')
pd.Series(t[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations
Used'])
```

Out[39]:

```
Test Statistic                -6.715004e+00
p-value                        3.600554e-09
#Lags Used                     2.000000e+01
Number of Observations Used    7.760000e+02
dtype: float64
```

Seasonal Decomposition

> **The seasonal component is quite low while the trend is quite strong with obvious dips in electricity consumption during summers i.e. April to September. This may be attributed to longer days during summer.**

In [40]:

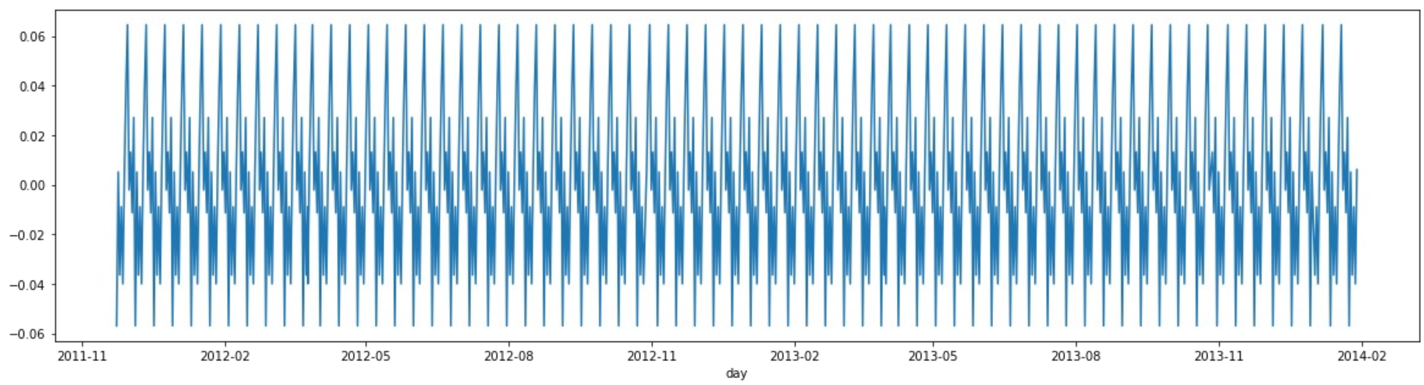
```python
s = sm.tsa.seasonal_decompose(train.avg_energy,freq=12)
```

In [41]:

```python
s.seasonal.plot(figsize=(20,5))
```

Out[41]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0111a03a20>
```

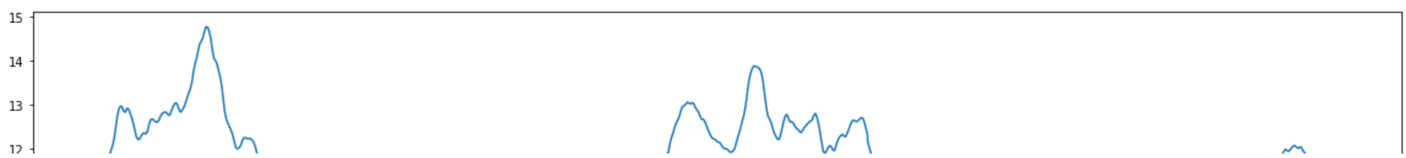


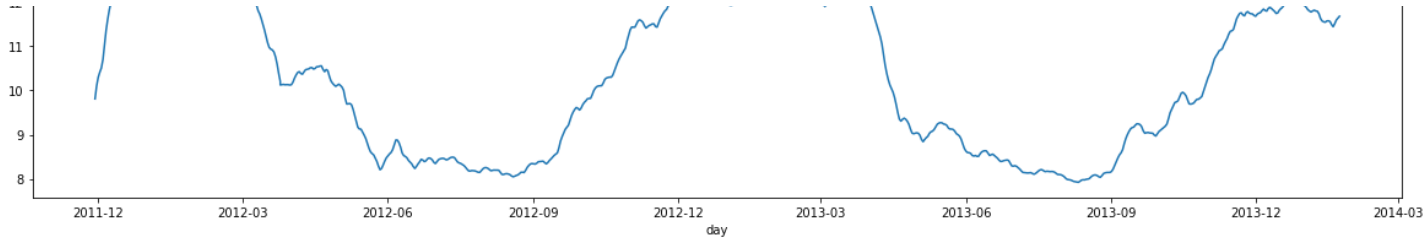
In [42]:

```python
s.trend.plot(figsize=(20,5))
```

Out[42]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f011198eac8>
```



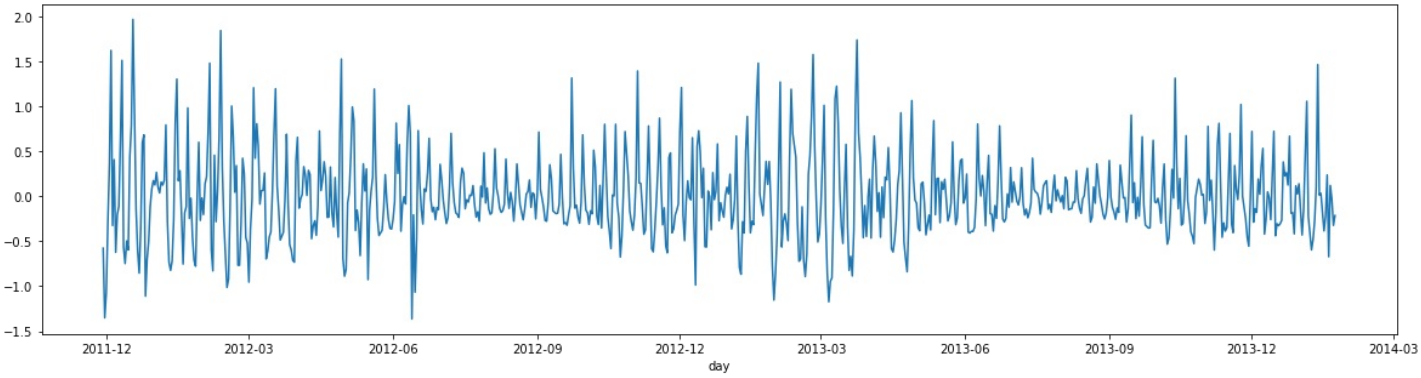


In [43]:

```
s.resid.plot(figsize=(20,5))
```

Out[43]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f011199de10>
```



In [44]:

```
endog = train['avg_energy']
exog = sm.add_constant(train[['weather_cluster','holiday_ind']])

mod = sm.tsa.statespace.SARIMAX(endog=endog, exog=exog, order=(7,1,1),seasonal_order=(1,
1, 0, 12),trend='c')
model_fit = mod.fit()
model_fit.summary()
```

Out[44]:

Statespace Model Results

Dep. Variable:	avg_energy	**No. Observations:**	798
Model:	SARIMAX(7, 1, 1)x(1, 1, 0, 12)	**Log Likelihood**	-649.430
Date:	Mon, 28 Mar 2022	**AIC**	1326.860
Time:	18:00:54	**BIC**	1392.179
Sample:	0	**HQIC**	1351.975
	- 798		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	-0.0064	0.017	-0.378	0.706	-0.039	0.027
const	-3.093e-08	3.64e-10	-84.858	0.000	-3.16e-08	-3.02e-08
weather_cluster	0.0008	0.024	0.035	0.972	-0.046	0.048
holiday_ind	-0.0343	0.088	-0.390	0.696	-0.207	0.138
ar.L1	-0.0007	0.086	-0.008	0.994	-0.170	0.168
ar.L2	-0.1548	0.032	-4.841	0.000	-0.217	-0.092
ar.L3	-0.1430	0.038	-3.747	0.000	-0.218	-0.068
ar.L4	-0.1512	0.038	-3.979	0.000	-0.226	-0.077
ar.L5	-0.1630	0.040	-4.076	0.000	-0.241	-0.085

ar.L6	0.0087	0.036	0.241	0.810	-0.062	0.080
ar.L7	0.3527	0.029	12.278	0.000	0.296	0.409
ma.L1	-0.1857	0.091	-2.038	0.042	-0.364	-0.007
ar.S.L12	-0.4830	0.032	-14.907	0.000	-0.547	-0.420
sigma2	0.3041	0.013	24.103	0.000	0.279	0.329

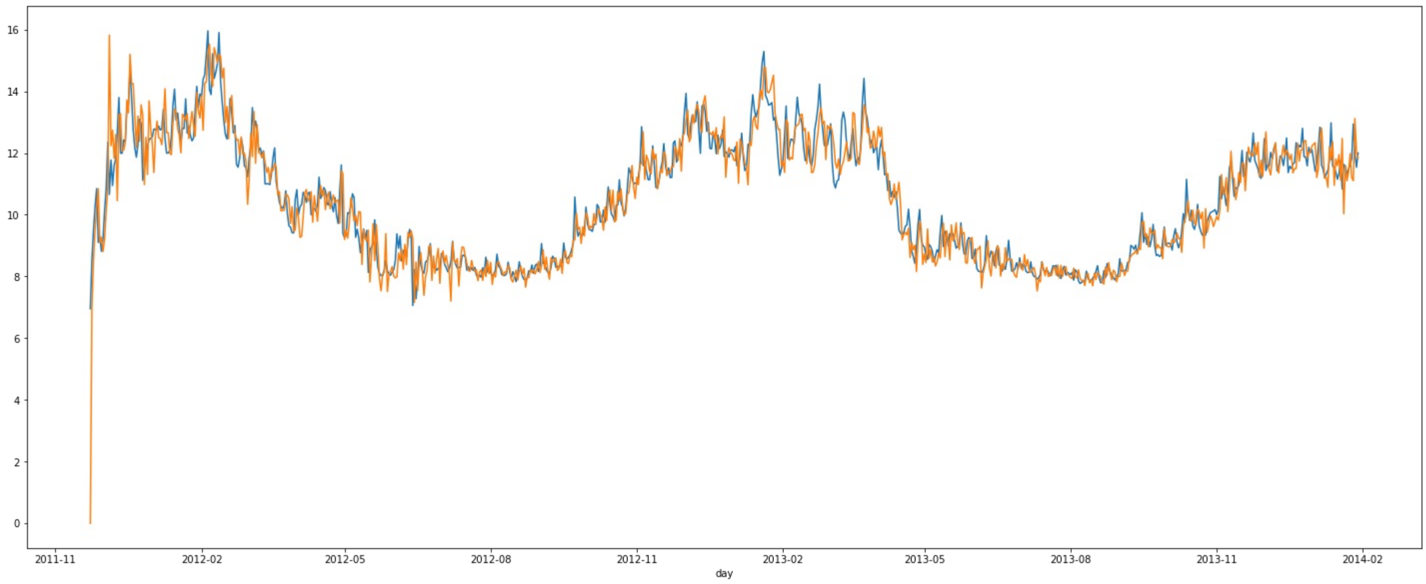
Ljung-Box (Q):	221.18	**Jarque-Bera (JB):**	45.20
Prob(Q):	0.00	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.53	**Skew:**	-0.16
Prob(H) (two-sided):	0.00	**Kurtosis:**	4.13

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 4.9e+16. Standard errors may be unstable.

Model Fit

In [45]:

```
train['avg_energy'].plot(figsize=(25,10))
model_fit.fittedvalues.plot()
plt.show()
```



Prediction

In [46]:

```
predict = model_fit.predict(start = len(train),end = len(train)+len(test)-1,exog = sm.ad
d_constant(test[['weather_cluster','holiday_ind']]))
test['predicted'] = predict.values
test.tail(5)
```

Out[46]:

	avg_energy	weather_cluster	holiday_ind	predicted
day				
2014-02-23	11.673756	1	0	11.558856
2014-02-24	10.586235	1	0	10.710839
2014-02-25	10.476498	1	0	11.449408

	avg_energy	weather_cluster	holiday_ind	predicted
day				
2014-02-26	[illegible]	1	0	[illegible]
2014-02-27	10.537250	1	0	11.487253

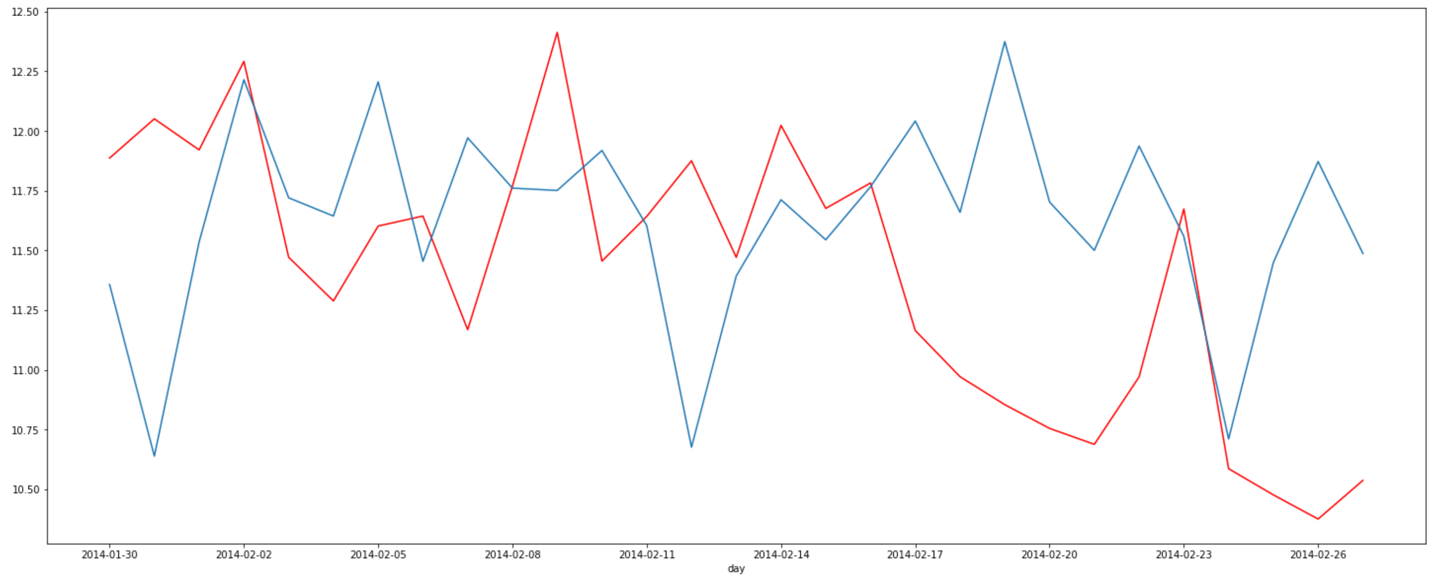
In [47]:

```
test['residual'] = abs(test['avg_energy']-test['predicted'])
MAE = test['residual'].sum()/len(test)
MAPE = (abs(test['residual'])/test['avg_energy']).sum()*100/len(test)
print("MAE:", MAE)
print("MAPE:", MAPE)
```

```
MAE: 0.5857320753255805
MAPE: 5.242918458200601
```

In [48]:

```
test['avg_energy'].plot(figsize=(25,10),color = 'red')
test['predicted'].plot()
plt.show()
```

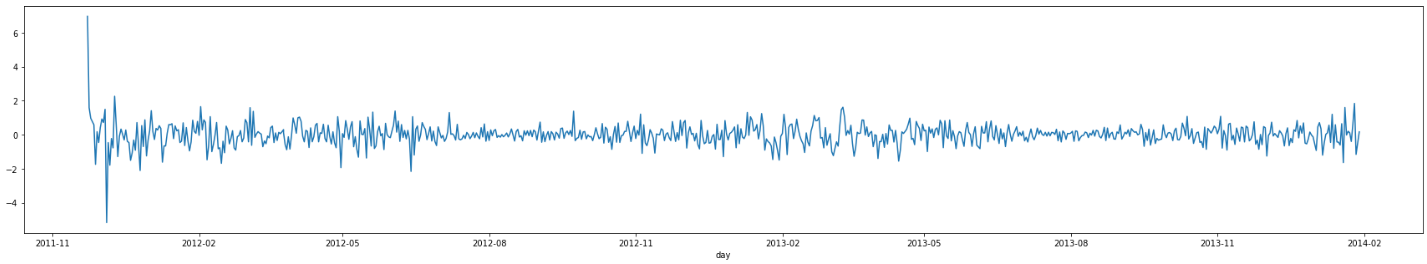


In [49]:

```
model_fit.resid.plot(figsize= (30,5))
```

Out[49]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0111971160>
```

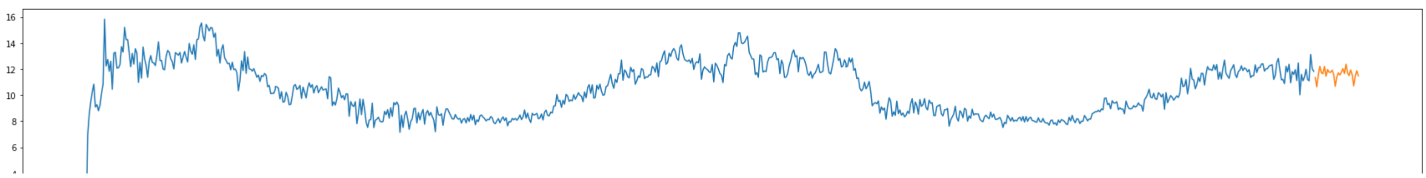


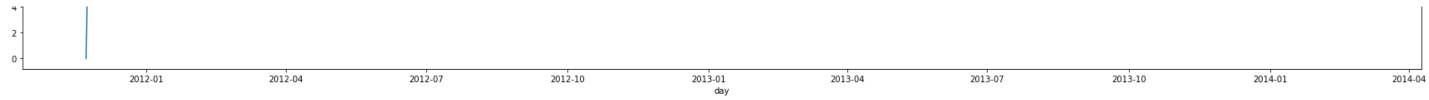
In [50]:

```
model_fit.fittedvalues.plot(figsize = (30,5))
test.predicted.plot()
```

Out[50]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0111b650f0>
```





In [51]:

```
test['predicted'].tail(5)
```

Out[51]:

```
day
2014-02-23    11.558856
2014-02-24    10.710839
2014-02-25    11.449408
2014-02-26    11.872326
2014-02-27    11.487253
Name: predicted, dtype: float64
```

LSTM

Using lags of upto 7 days we are going to convert this into a supervised problem. I have taken the function to create lags from this tutorial by Jason Brownlee. He has also applied the same to convert multivariate data to a supervised dataframe which he has in turn applied LSTM on.

In [52]:

```
np.random.seed(11)
dataframe = weather_energy.loc[:,'avg_energy']
dataset = dataframe.values
dataset = dataset.astype('float32')
```

In [53]:

```
# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1
    df = pd.DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = pd.concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
```

In [54]:

```
reframed = series_to_supervised(dataset, 7,1)
reframed.head(3)
```

Out[54]:

	var1(t-7)	var1(t-6)	var1(t-5)	var1(t-4)	var1(t-3)	var1(t-2)	var1(t-1)	var1(t)
7	6.952693	8.536480	9.499782	10.267707	10.850805	9.103382	9.274873	8.813513
8	8.536480	9.499782	10.267707	10.850805	9.103382	9.274873	8.813513	9.227707

	var1(t-7)	var1(t-6)	var1(t-5)	var1(t-4)	var1(t-3)	var1(t-2)	var1(t-1)	var1(t)
9	9.499782	10.267707	10.850805	9.103382	9.274873	8.813513	9.227707	10.145910

In [55]:

```
reframed['weather_cluster'] = weather_energy.weather_cluster.values[7:]
reframed['holiday_ind']= weather_energy.holiday_ind.values[7:]
```

In [56]:

```
reframed = reframed.reindex(['weather_cluster', 'holiday_ind','var1(t-7)', 'var1(t-6)', '
var1(t-5)', 'var1(t-4)', 'var1(t-3)','var1(t-2)', 'var1(t-1)', 'var1(t)'], axis=1)
reframed = reframed.values
```

Normalization

In [57]:

```
scaler = MinMaxScaler(feature_range=(0, 1))
reframed = scaler.fit_transform(reframed)
```

In [58]:

```
# split into train and test sets
train = reframed[:(len(reframed)-30), :]
test = reframed[(len(reframed)-30):len(reframed), :]
```

In [59]:

```
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
```

In [60]:

```
# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

```
(791, 1, 9) (791,) (30, 1, 9) (30,)
```

Modelling

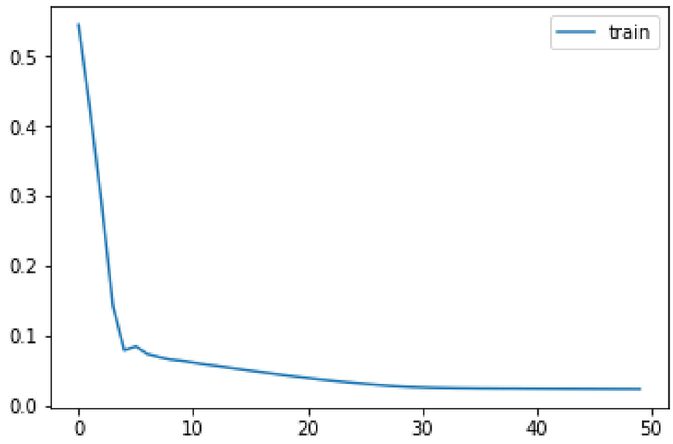
In [61]:

```
# design network
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
# fit network
history = model.fit(train_X, train_y, epochs=50, batch_size=72, verbose=2, shuffle=False
)
# plot history
pyplot.plot(history.history['loss'], label='train')
pyplot.legend()
pyplot.show()
```

```
Epoch 1/50
 - 1s - loss: 0.5452
Epoch 2/50
 - 0s - loss: 0.4243
Epoch 3/50
 - 0s - loss: 0.2912
Epoch 4/50
 - 0s - loss: 0.1434
Epoch 5/50
 - 0s - loss: 0.0790
Epoch 6/50
 - 0s - loss: 0.0846
Epoch 7/50
```

```
Epoch 7/50
 - 0s - loss: 0.0740
Epoch 8/50
 - 0s - loss: 0.0696
Epoch 9/50
 - 0s - loss: 0.0661
Epoch 10/50
 - 0s - loss: 0.0642
Epoch 11/50
 - 0s - loss: 0.0616
Epoch 12/50
 - 0s - loss: 0.0591
Epoch 13/50
 - 0s - loss: 0.0569
Epoch 14/50
 - 0s - loss: 0.0545
Epoch 15/50
 - 0s - loss: 0.0523
Epoch 16/50
 - 0s - loss: 0.0499
Epoch 17/50
 - 0s - loss: 0.0477
Epoch 18/50
 - 0s - loss: 0.0455
Epoch 19/50
 - 0s - loss: 0.0434
Epoch 20/50
 - 0s - loss: 0.0413
Epoch 21/50
 - 0s - loss: 0.0394
Epoch 22/50
 - 0s - loss: 0.0375
Epoch 23/50
 - 0s - loss: 0.0357
Epoch 24/50
 - 0s - loss: 0.0340
Epoch 25/50
 - 0s - loss: 0.0325
Epoch 26/50
 - 0s - loss: 0.0310
Epoch 27/50
 - 0s - loss: 0.0296
Epoch 28/50
 - 0s - loss: 0.0284
Epoch 29/50
 - 0s - loss: 0.0273
Epoch 30/50
 - 0s - loss: 0.0264
Epoch 31/50
 - 0s - loss: 0.0259
Epoch 32/50
 - 0s - loss: 0.0254
Epoch 33/50
 - 0s - loss: 0.0251
Epoch 34/50
 - 0s - loss: 0.0248
Epoch 35/50
 - 0s - loss: 0.0247
Epoch 36/50
 - 0s - loss: 0.0245
Epoch 37/50
 - 0s - loss: 0.0244
Epoch 38/50
 - 0s - loss: 0.0243
Epoch 39/50
 - 0s - loss: 0.0242
Epoch 40/50
 - 0s - loss: 0.0241
Epoch 41/50
 - 0s - loss: 0.0241
Epoch 42/50
 - 0s - loss: 0.0240
Epoch 43/50
```

```
Epoch 43/50
 - 0s - loss: 0.0239
Epoch 44/50
 - 0s - loss: 0.0238
Epoch 45/50
 - 0s - loss: 0.0237
Epoch 46/50
 - 0s - loss: 0.0237
Epoch 47/50
 - 0s - loss: 0.0236
Epoch 48/50
 - 0s - loss: 0.0235
Epoch 49/50
 - 0s - loss: 0.0235
Epoch 50/50
 - 0s - loss: 0.0234
```



Prediction

In [62]:

```
# make a prediction
yhat = model.predict(test_X)
```

In [63]:

```
test_X = test_X.reshape(test_X.shape[0], test_X.shape[2])
```

In [64]:

```
# invert scaling for forecast
inv_yhat = np.concatenate((yhat, test_X), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
```

In [65]:

```
# invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))
inv_y = np.concatenate((test_y, test_X), axis=1)
inv_y = scaler.inverse_transform(inv_y)
```

Performance

In [66]:

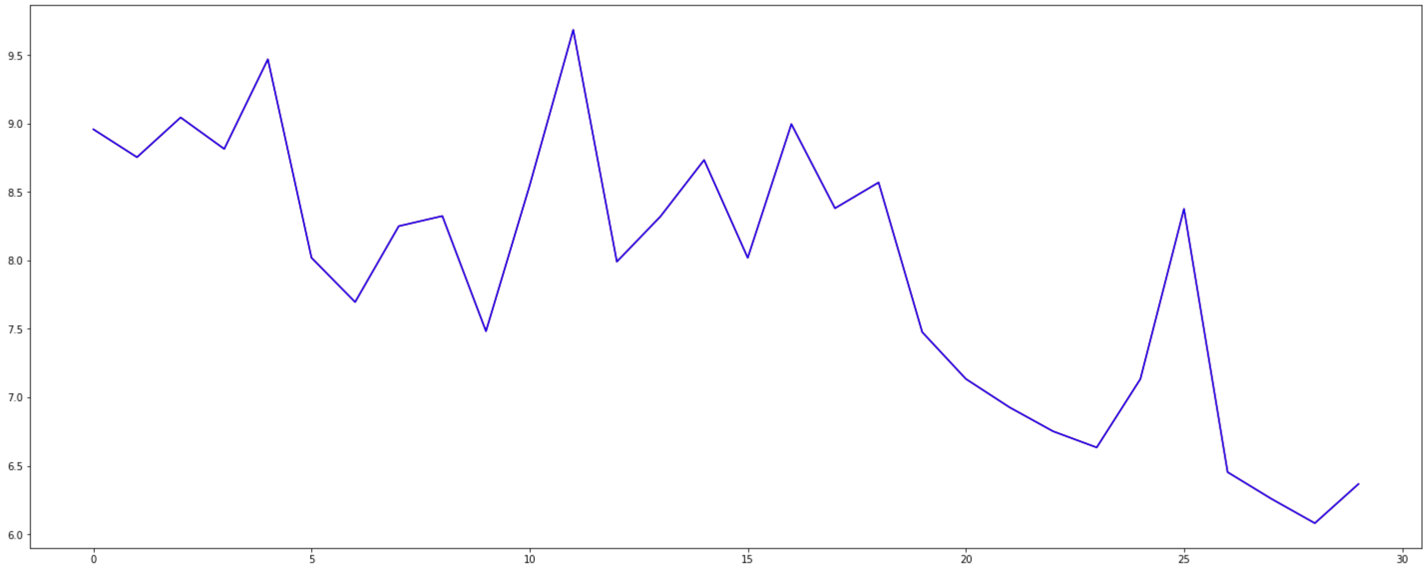
```
act = [i[9] for i in inv_y] # last element is the predicted average energy
pred = [i[9] for i in inv_yhat] # last element is the actual average energy

# calculate RMSE
import math
rmse = math.sqrt(mean_squared_error(act, pred))
print('Test RMSE: %.3f' % rmse)
```

```
Test RMSE: 0.000
```

In [67]:

```
predicted_lstm = pd.DataFrame({'predicted':pred,'avg_energy':act})
predicted_lstm['avg_energy'].plot(figsize=(25,10),color = 'red')
predicted_lstm['predicted'].plot(color = 'blue')
plt.show()
```



In [68]:

```

```