Code Implementation

Proposing Energy-Efficient Urban Spaces using IoT and Data Visualisation

Anubhav Rakeshwar Singh Guleria

18BCE0186

7758984636

anubhavsingh.guleria2018@vitstudent.ac.in

Ranjan Nitin Pramod

18BCE0272

8529589264

nitin.ranjan2018@vitstudent.ac.in

Srinivasan Sharma

18BCE0931

6385664521

srinivasan.sharma2018@vitstudent.ac.in

Dr. Gladys Gnana Kiruba B

Assistant Professor (Senior)

9597397290

gladys@vit.ac.in

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]:

```python
#!pip install pmdarima
```

In [2]:

```python
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.**

```python
# 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]:

```python
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]:

```python
housecount = energy.groupby('day')[['LCLid']].nunique()
housecount.head(4)
```
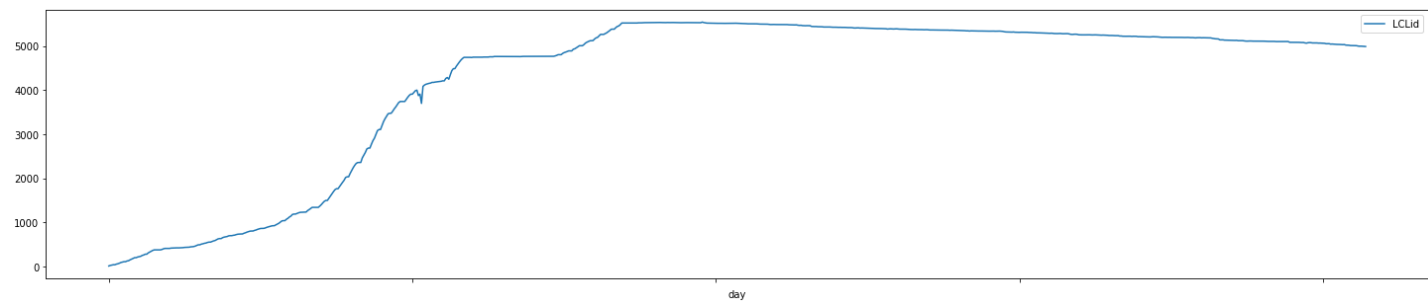
Out[5]:

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

In [6]:

```python
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]:

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

In [8]:

```python
energy.count()
```

Out[8]:

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

In [9]:

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

In [10]:

```python
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]:

```python
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%	46841.760997	5138.000000	10.516988
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	100
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	882.000000
mean	13.660113	195.702948	6.530034	0.477605	3.581803	1014.127540	12.723866	11.167143
std	6.182744	89.340783	4.830875	0.193514	1.694007	11.073038	7.279168	2.466109
min	-0.060000	0.000000	-7.840000	0.000000	0.200000	979.250000	-6.460000	1.480000
25%	9.502500	120.500000	3.180000	0.350000	2.370000	1007.435000	7.032500	10.327500
50%	12.625000	219.000000	6.380000	0.470000	3.440000	1014.615000	12.470000	11.970000
75%	17.920000	255.000000	10.057500	0.600000	4.577500	1021.755000	17.910000	12.830000
max	32.400000	359.000000	17.770000	1.000000	9.960000	1040.920000	32.420000	15.340000

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()
```
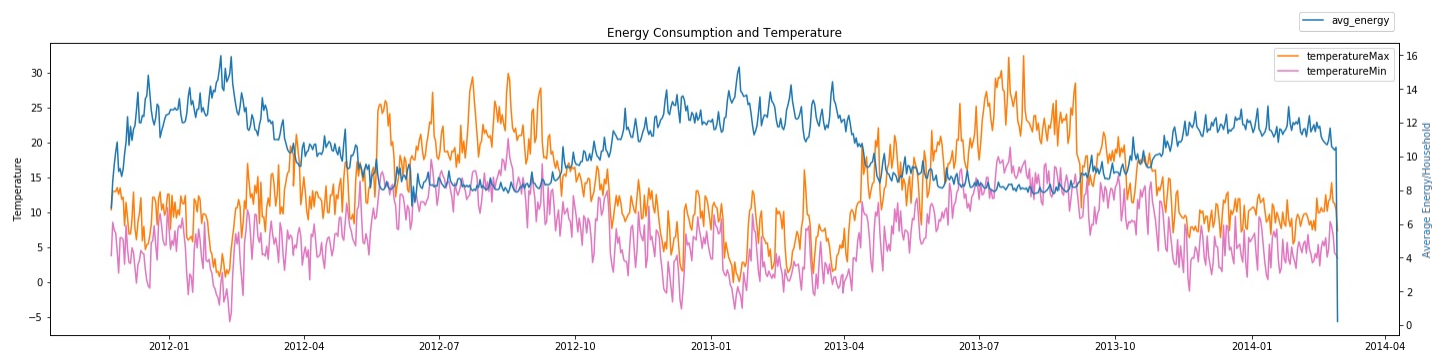

2. Humidity

> **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()
```
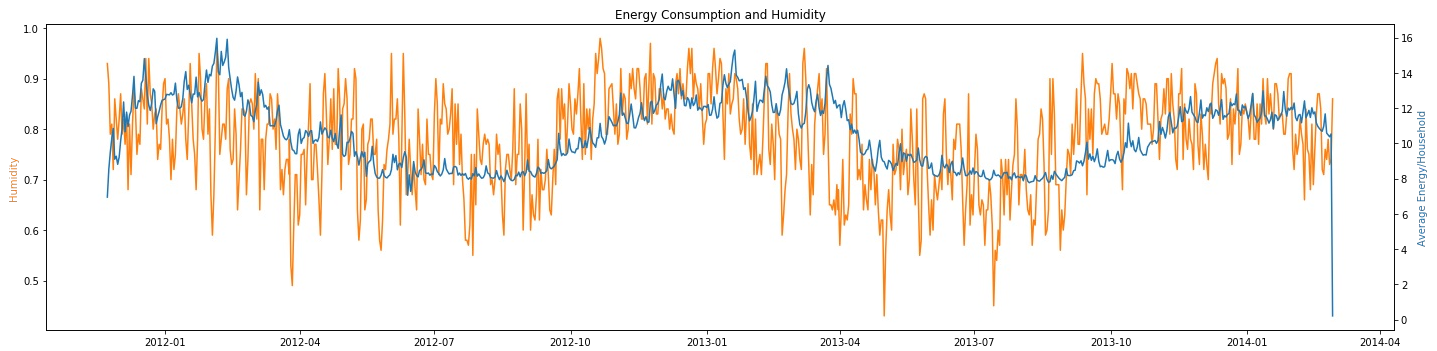

3. Cloud Cover

> 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()
```
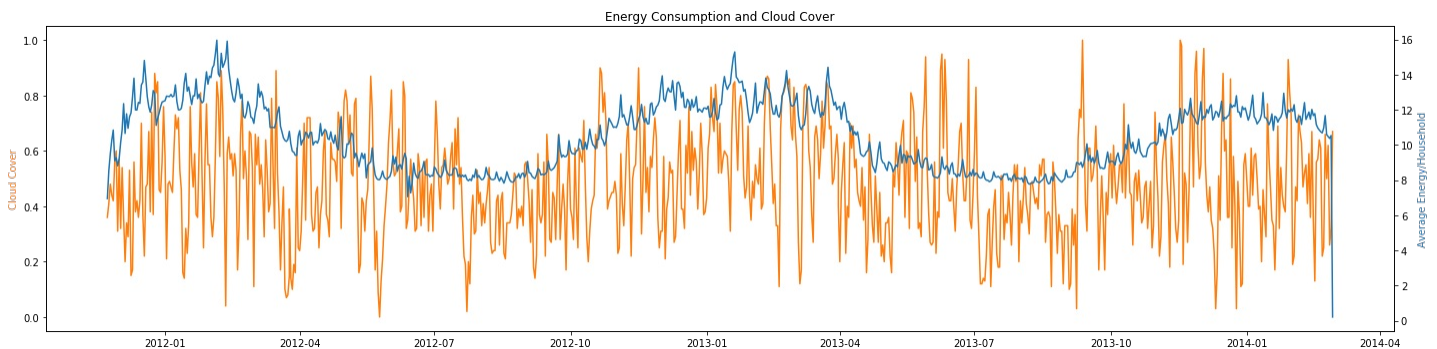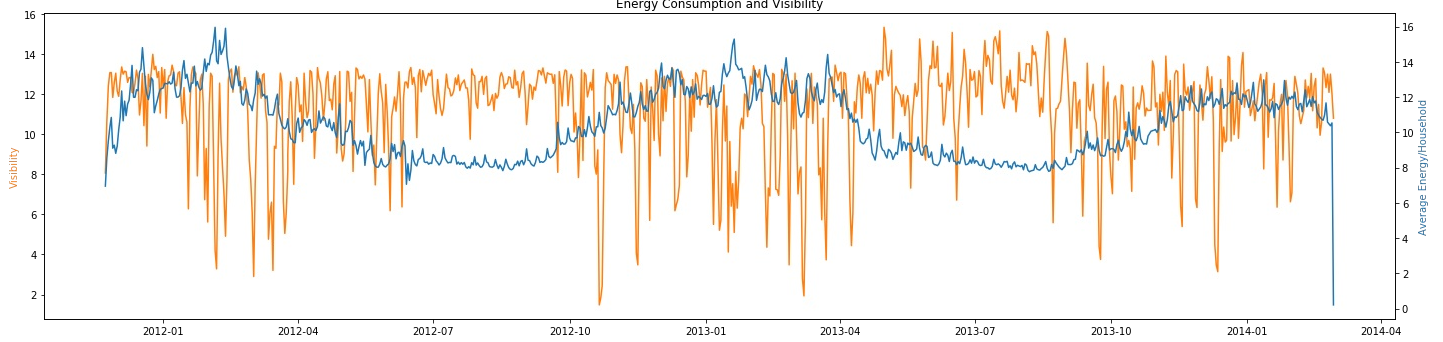

4. Visibility

> 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()
```
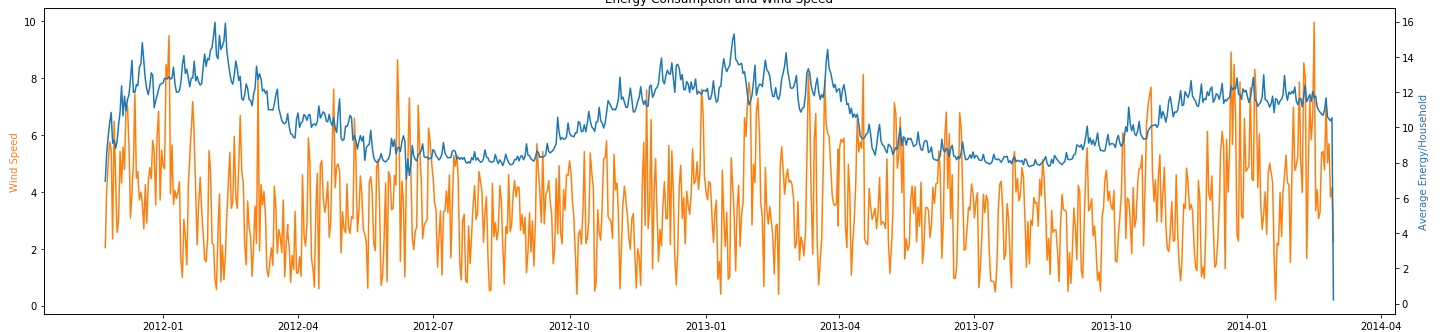

6. UV Index

> 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()
```
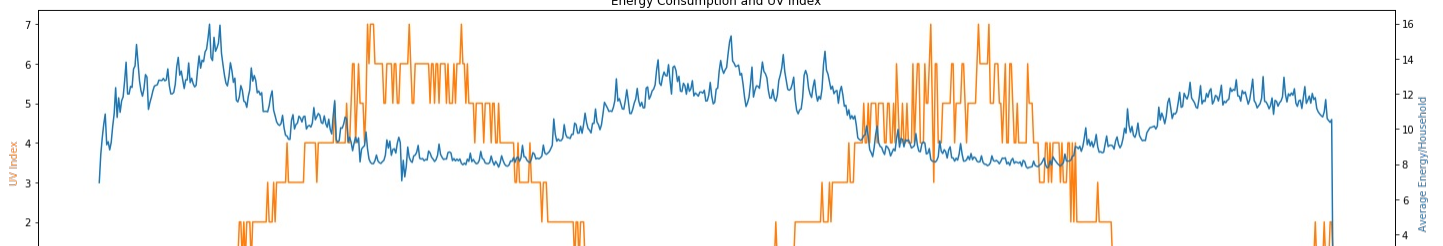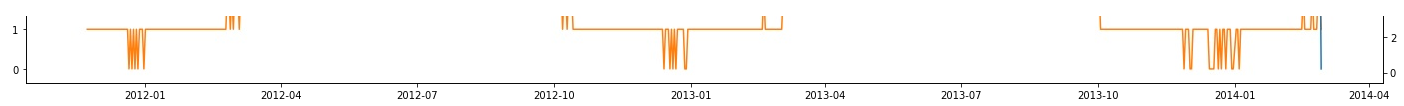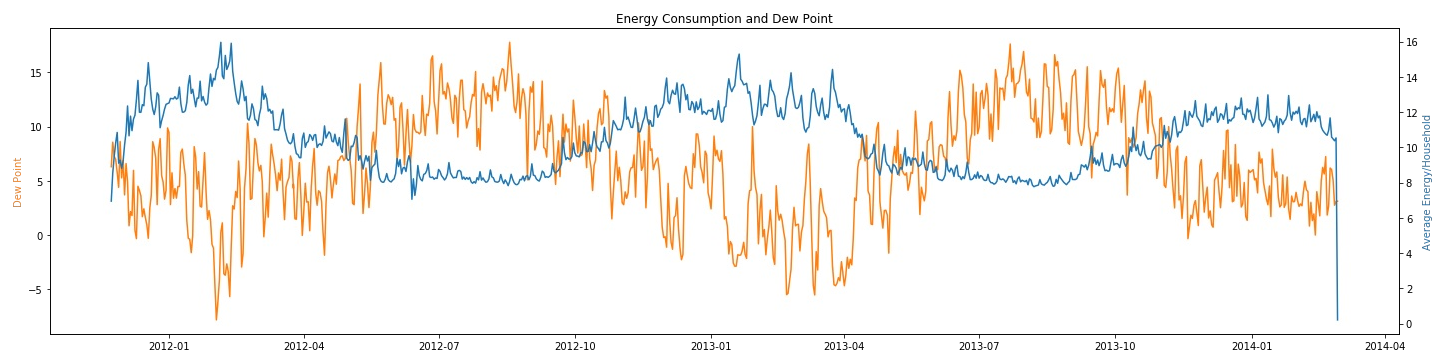
7. dewPoint

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

In [22]:

```python
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]:

```python
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	

	visibility	avg_energy	temperatureMax	dewPoint	cloudCover	windSpeed	pressure	visibility	humidity	uvIndex	mo
							0.012508		0.578130		
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]:

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

In [25]:

```python
# 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()
```
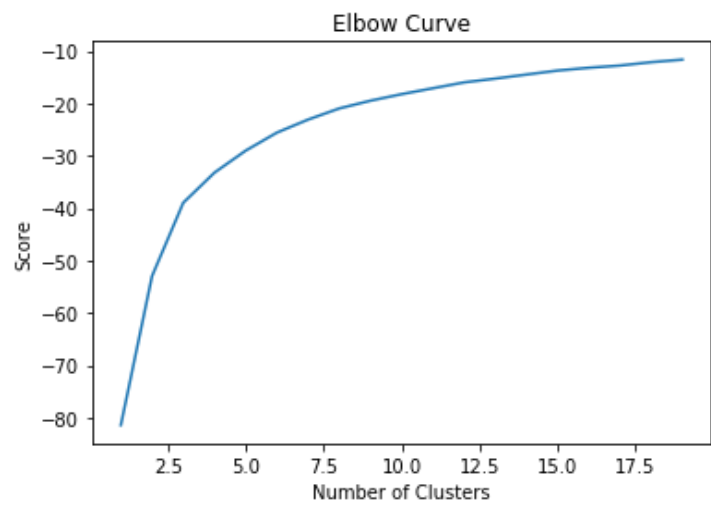

In [26]:

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

In [27]:

```python
# 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
```
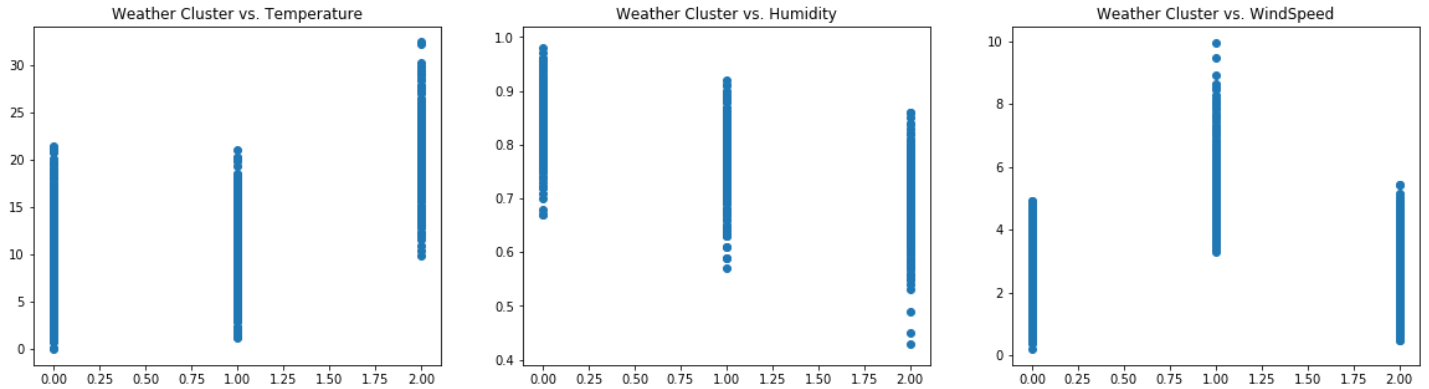

In [28]:

```
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()
```
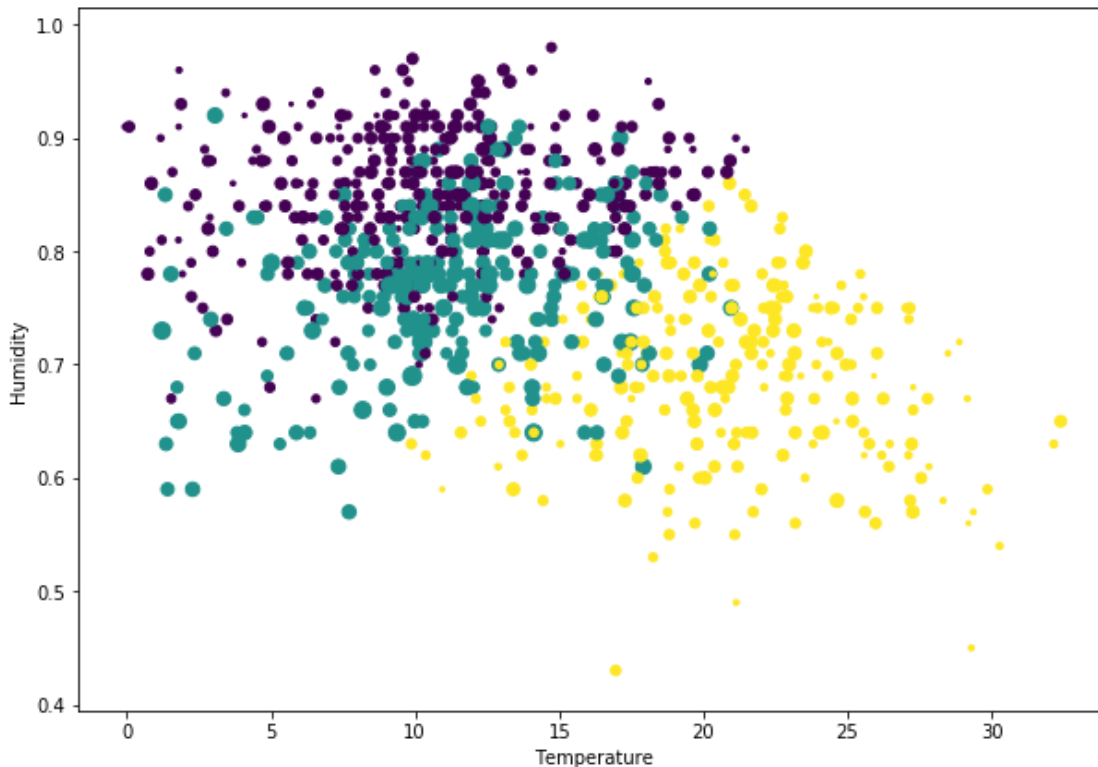

UK Bank Holidays

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
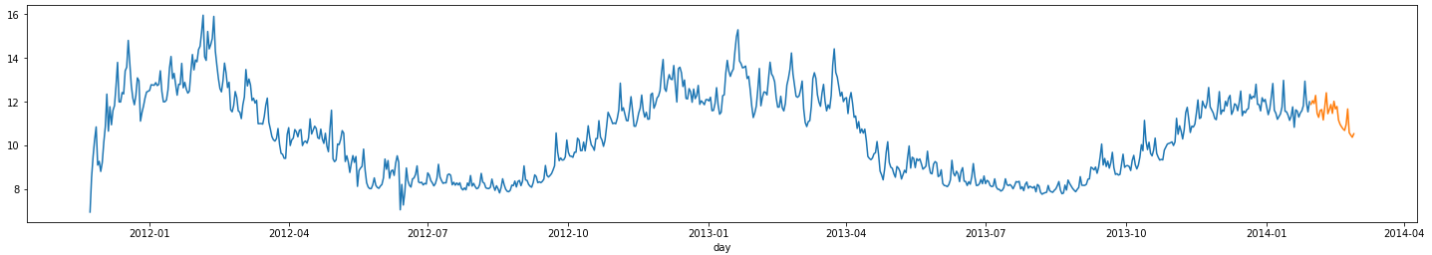
In [32]:

```
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>
```
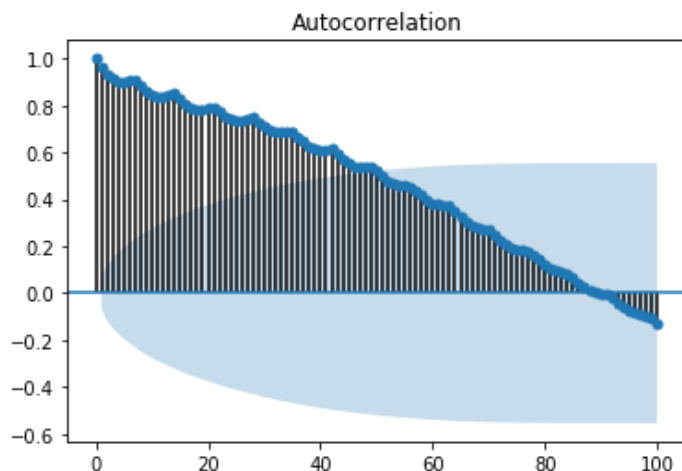

In [34]:

```
test.head(1)
```
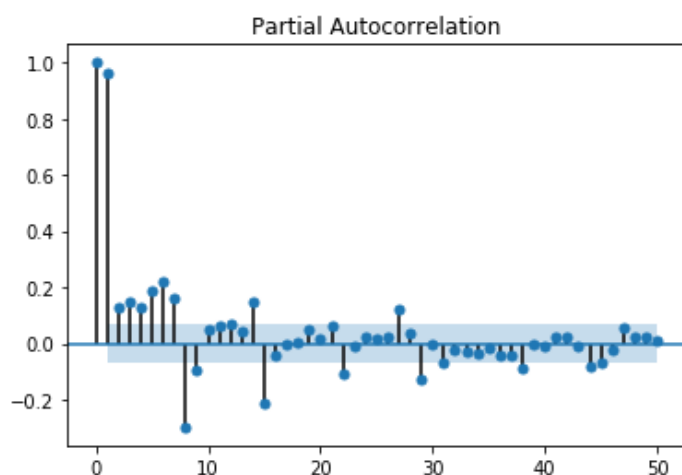
Out[34]:

	avg_energy	weather_cluster	holiday_ind
day			

ACF PACF

In [35]:

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

Autocorrelation

In [36]:

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

Partial Autocorrelation

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
dtupe. float64
```

```
dtype: float64
```

```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.
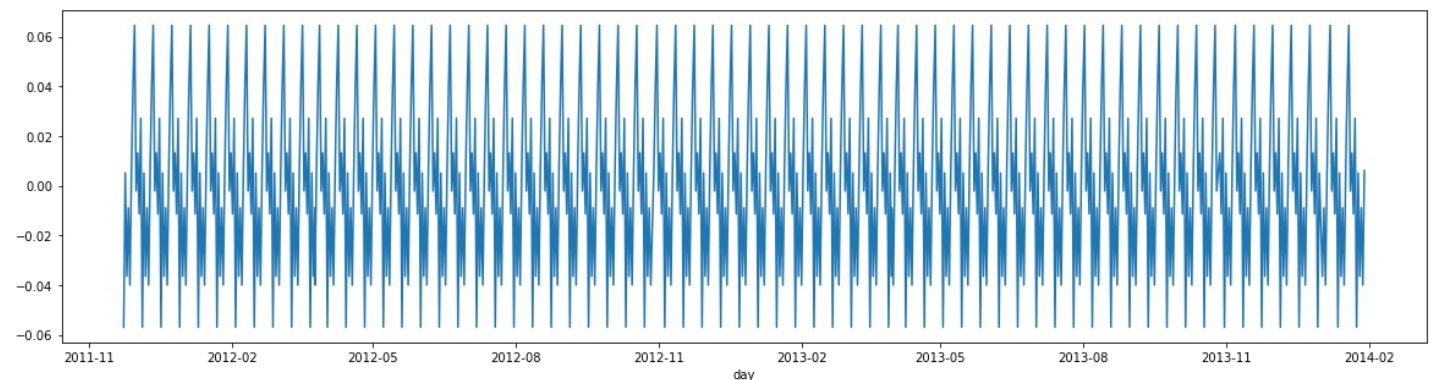
In [40]:

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

In [41]:

```python
s.seasonal.plot(figsize=(20,5))
```
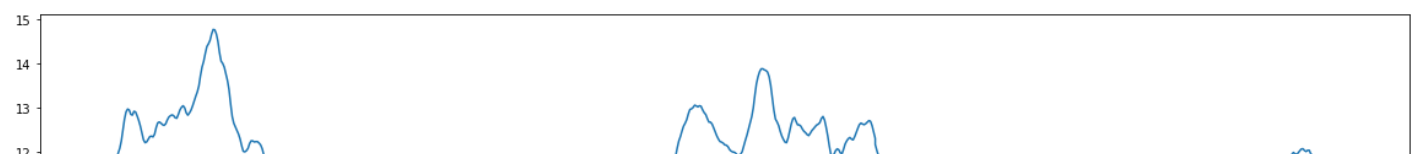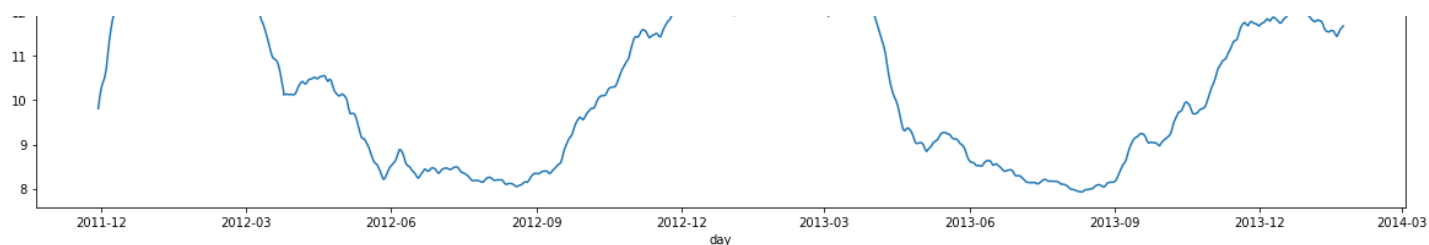
Out[41]:

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


In [42]:

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

Out[42]:

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

```
s.resid.plot(figsize=(20,5))
```
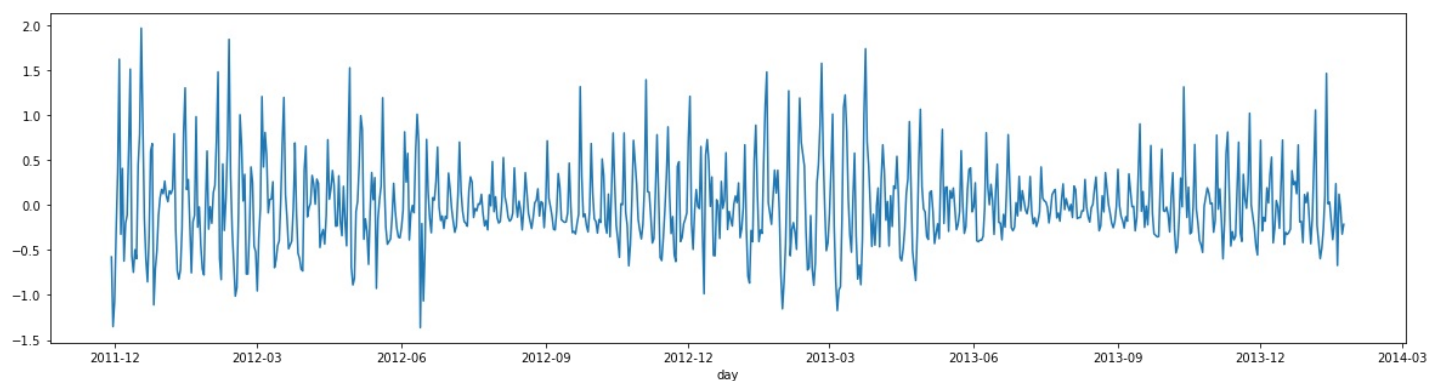
Out[43]:

```
<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()
```
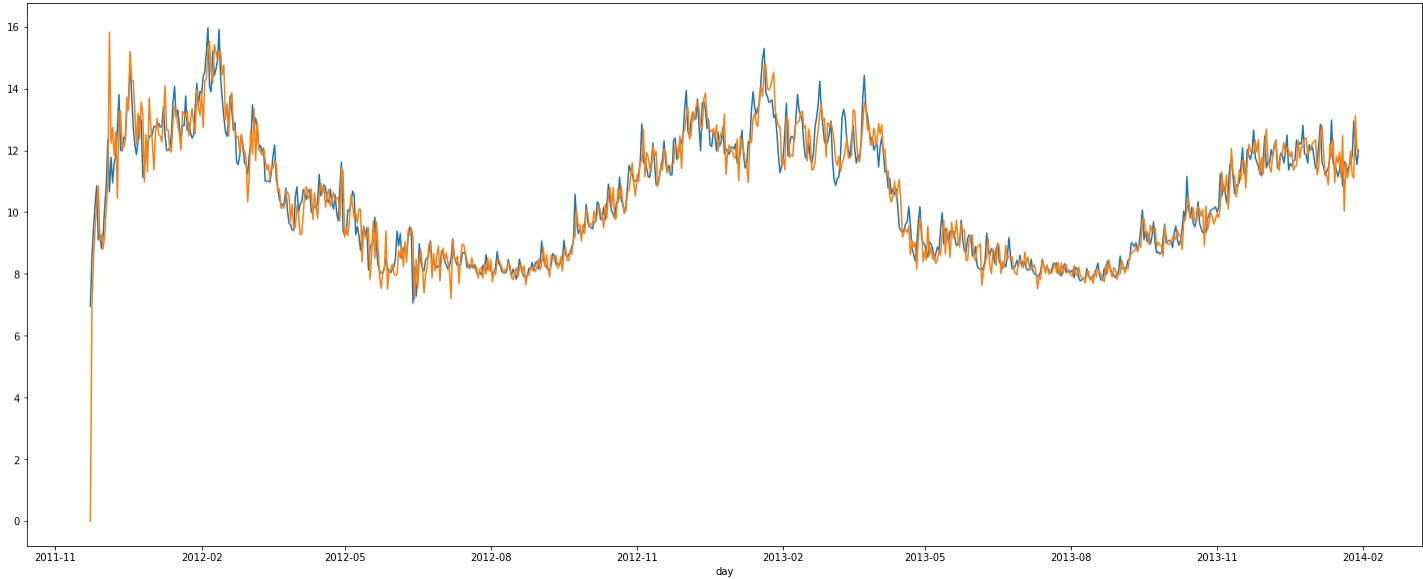

Prediction

In [46]:

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

Out[46]:

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

In [47]:

```python
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]:

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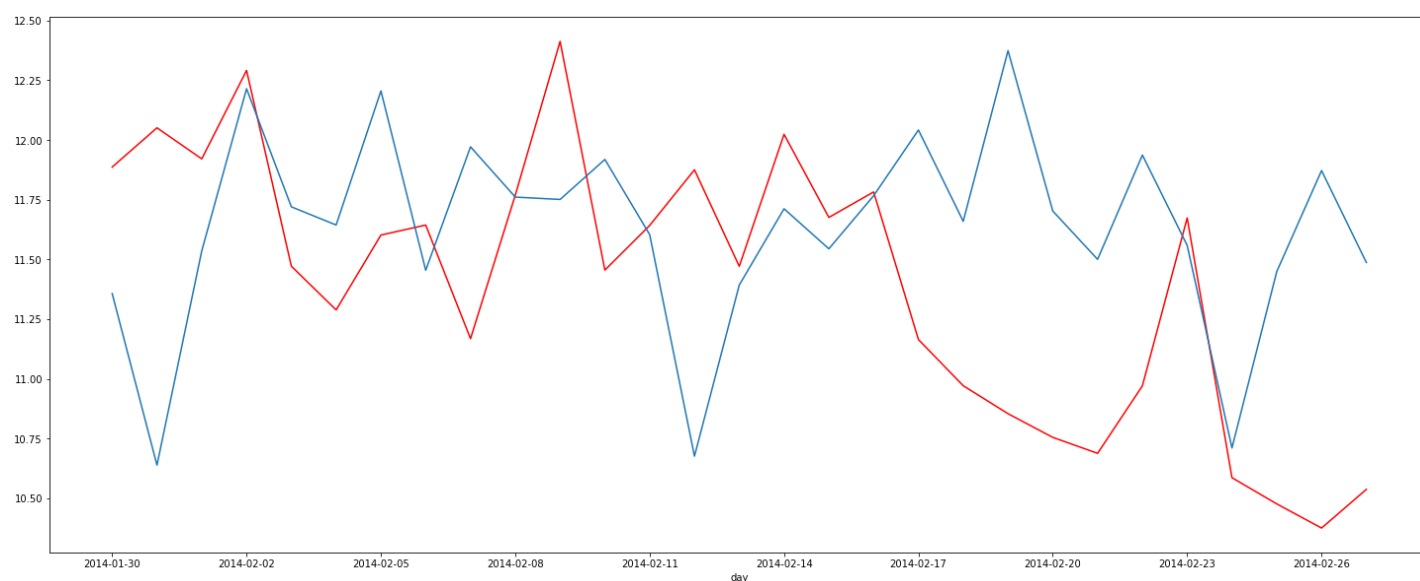

In [49]:

```python
model_fit.resid.plot(figsize= (30,5))
```
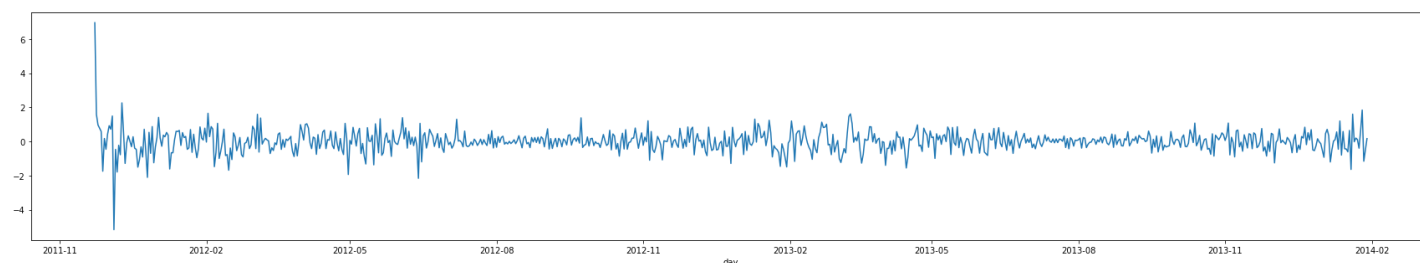
Out[49]:

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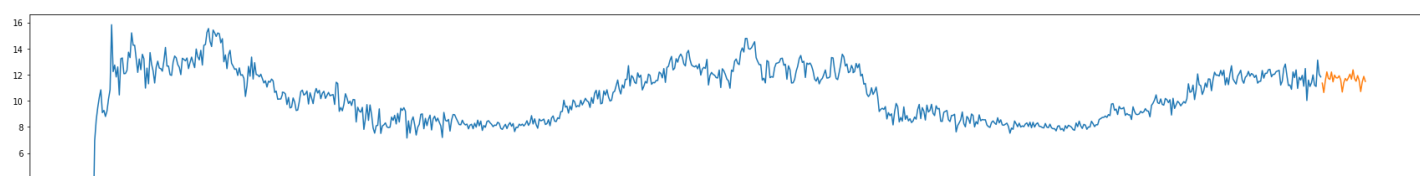

In [50]:

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

Out[50]:

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

```
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

In [55]:

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

In [56]:

```python
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]:

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

In [58]:

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

In [59]:

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

In [60]:

```python
# 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]:

```python
# 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 45/50
 - 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
```
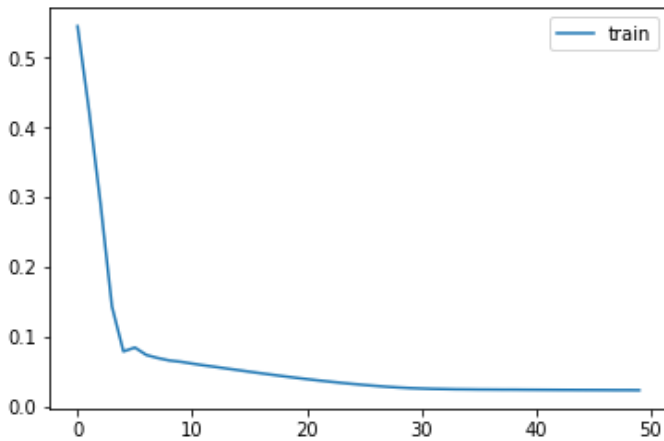

Prediction

In [62]:

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

In [63]:

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

In [64]:

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

In [65]:

```python
# 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]:

```python
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
```

```
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()
```
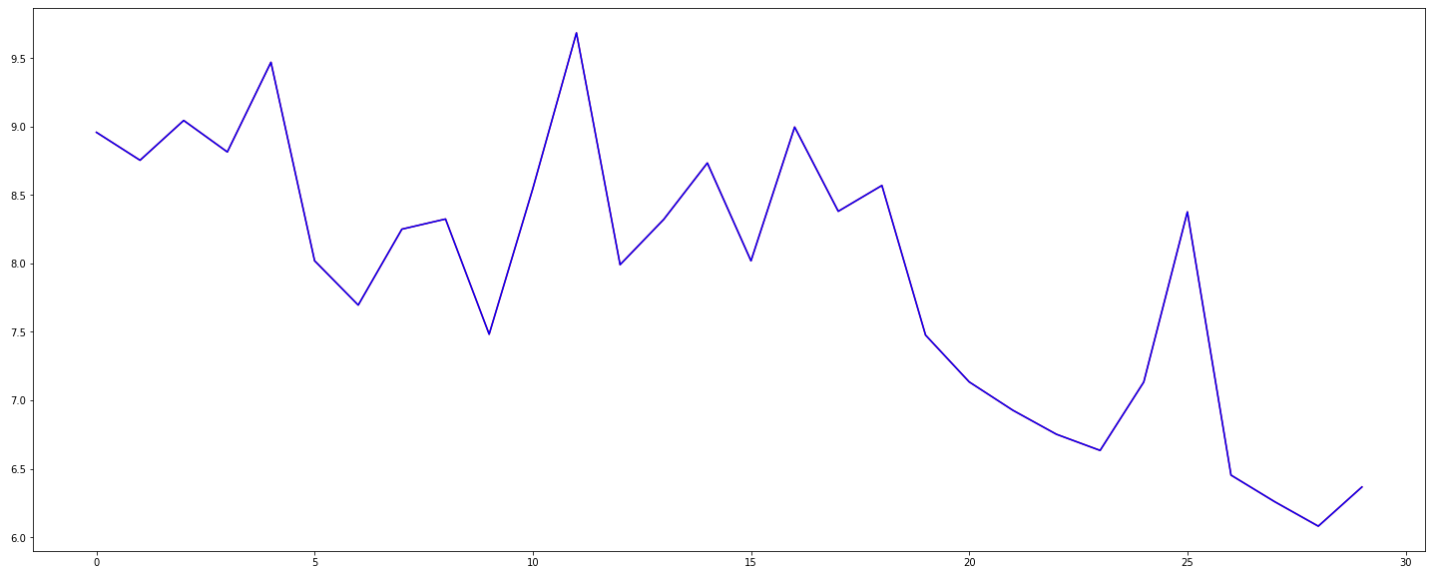
```python
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model, metrics# linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files
under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserve
d as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of
the current session
```

```
/kaggle/input/energyconsumedinindia/Powerdata.csv
/kaggle/input/india-power/Powerdata.csv
/kaggle/input/india-power/long_data.csv
```

In [2]:

```python
df = pd.read_csv('../input/energyconsumedinindia/Powerdata.csv')
df2 = pd.read_csv('../input/india-power/Powerdata.csv')
```

In [3]:

```python
df.shape
```

Out[3]:

```
(210, 5)
```

In [4]:

```python
df2.head()
```

Out[4]:

	Date	Punjab	Haryana	Rajasthan	Delhi	UP	Uttarakhand	HP	J&K	Chandigarh	...	West Bengal	Sikkim	Arunachal Pradesh
0	28/10/2019	95.3	105.7	205.6	59.2	232.8		33.2	24.6	42.6	3.3	...	129.0	1.4	2.1
1	29/10/2019	96.5	111.5	213.6	60.2	240.9		33.8	24.1	43.1	3.2	...	142.0	1.4	2.0
2	30/10/2019	102.4	115.1	215.2	60.7	249.9		34.0	24.7	46.0	3.2	...	142.2	1.4	2.2
3	31/10/2019	103.9	116.2	216.8	60.6	256.7		33.7	24.7	44.8	3.1	...	141.3	1.4	2.2
4	01/11/2019	105.6	117.4	210.3	63.4	263.0		34.4	21.5	47.3	3.1	...	140.2	1.3	2.3

5 rows × 35 columns

In [5]:

```python
df2.describe()
```

Out[5]:

|	Punjab	Haryana	Rajasthan	Delhi	UP	Uttarakhand	HP	J&K	Chandigarh	Chhatti
count	210.000000	210.000000	210.000000	210.000000	210.000000	210.000000	210.000000	210.000000	210.000000	210.
mean	104.077619	111.281905	210.673333	64.659524	269.191905	31.842381	24.453333	45.226190	3.480952	76.
std	19.233014	19.320635	31.857113	9.845594	33.564225	7.043567	6.388590	5.326384	0.704408	5.
min	56.100000	64.800000	105.800000	41.800000	186.800000	16.800000	11.800000	17.800000	2.200000	64.
25%	94.325000	98.050000	194.000000	60.300000	248.475000	26.575000	20.200000	42.425000	3.100000	72.
50%	105.650000	115.100000	215.750000	63.800000	268.650000	34.000000	27.200000	44.850000	3.400000	75.
75%	114.000000	125.175000	236.450000	71.325000	284.550000	37.200000	29.275000	49.175000	4.000000	79.
max	171.000000	158.000000	251.700000	95.200000	417.500000	42.200000	32.800000	54.100000	5.000000	91.

8 rows × 34 columns

Linear Regression

In [6]:

```
df2.plot(x='Date', y='ALL INDIA TOTAL', style='o')
plt.title('Energy vs Time')
plt.xlabel('energy consumed')
plt.ylabel('Time')
plt.show()
```
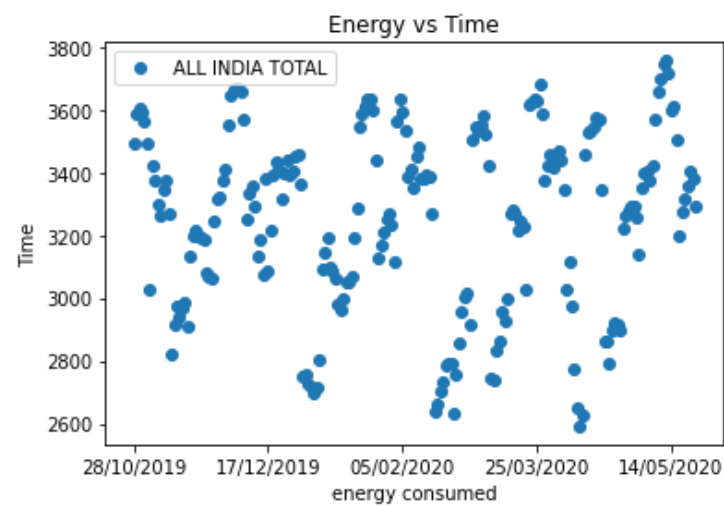

In [7]:

```
X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values
```

In [8]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

In [9]:

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[9]:

```
LinearRegression()
```

In [10]:

```
print(regressor.intercept_)
```

7.448092098690722

In [11]:

```
print(regressor.coef_)
```

[0.33343107 0.66656893 -0.33343107 -0.00226358]

In [12]:

```
df.head()
```

Out[12]:

	ALL INDIA TOTAL	MEAN	ABSOLUTE DIFFERENCE WITH MEAN	MEDIAN	ABSOLUTE DIFFERENCE MEDIAN
0	3494.4	3249.592381	244.807619	3290.4	204.0
1	3586.4	3249.592000	336.808000	3290.4	296.0
2	3604.8	3249.592000	355.208000	3290.4	314.4
3	3593.5	3249.592000	343.908000	3290.4	303.1
4	3565.2	3249.592000	315.608000	3290.4	274.8

In [13]:

```
y_pred = regressor.predict(X_test)
```

In [14]:

```
#df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
#df
```

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files und

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the
```

```
    /kaggle/input/india-power/Powerdata.csv
    /kaggle/input/india-power/long_data.csv
```

```
!pip install bar_chart_race
```

    Collecting bar_chart_race
      Downloading bar_chart_race-0.1.0-py3-none-any.whl (156 kB)
         |████████████████████████████████| 156 kB 926 kB/s
    Requirement already satisfied: matplotlib>=3.1 in /opt/conda/lib/python3.7/site-packages
    Requirement already satisfied: pandas>=0.24 in /opt/conda/lib/python3.7/site-packages (f
    Requirement already satisfied: python-dateutil>=2.7 in /opt/conda/lib/python3.7/site-pac
    Requirement already satisfied: pyparsing>=2.2.1 in /opt/conda/lib/python3.7/site-package
    Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.7/site-packages (
    Requirement already satisfied: fonttools>=4.22.0 in /opt/conda/lib/python3.7/site-packag
    Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.7/site-packages
    Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.7/site-packages (f
    Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/python3.7/site-packag
    Requirement already satisfied: numpy>=1.17 in /opt/conda/lib/python3.7/site-packages (fr
    Requirement already satisfied: pytz>=2017.3 in /opt/conda/lib/python3.7/site-packages (f
    Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.7/site-packages (from
    Installing collected packages: bar-chart-race
   Successfully installed bar-chart-race-0.1.0
    WARNING: Running pip as the 'root' user can result in broken permissions and conflicting
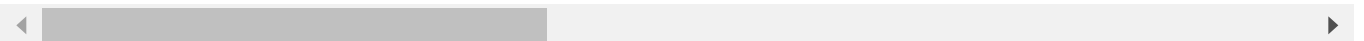
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib notebook
import plotly.express as px
import plotly.graph_objects as go
import plotly.figure_factory as ff
```

```
from IPython.display import HTML
import calendar
from plotly.subplots import make_subplots
import bar_chart_race as bcr
```

```
/opt/conda/lib/python3.7/site-packages/geopandas/_compat.py:115: UserWarning: The Shapel
  shapely_geos_version, geos_capi_version_string
```

◄ ▐▬▬ ► ▶
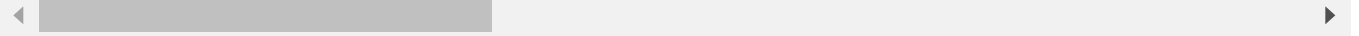
```
df = pd.read_csv('../input/india-power/Powerdata.csv')
df_long = pd.read_csv('../input/india-power/long_data.csv')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 210 entries, 0 to 209
Data columns (total 35 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Date               210 non-null    object
 1   Punjab             210 non-null    float64
 2   Haryana            210 non-null    float64
 3   Rajasthan          210 non-null    float64
 4   Delhi              210 non-null    float64
 5   UP                 210 non-null    float64
 6   Uttarakhand        210 non-null    float64
 7   HP                 210 non-null    float64
 8   J&K                210 non-null    float64
 9   Chandigarh         210 non-null    float64
 10  Chhattisgarh       210 non-null    float64
 11  Gujarat            210 non-null    float64
 12  MP                 210 non-null    float64
 13  Maharashtra        210 non-null    float64
 14  Goa                210 non-null    float64
 15  DNH                210 non-null    float64
 16  Andhra Pradesh     210 non-null    float64
 17  Telangana          210 non-null    float64
 18  Karnataka          210 non-null    float64
 19  Kerala             210 non-null    float64
 20  Tamil Nadu         210 non-null    float64
 21  Pondy              210 non-null    float64
 22  Bihar              210 non-null    float64
 23  Jharkhand          210 non-null    float64
 24  Odisha             210 non-null    float64
 25  West Bengal        210 non-null    float64
 26  Sikkim             210 non-null    float64
 27  Arunachal Pradesh  210 non-null    float64
 28  Assam              210 non-null    float64
 29  Manipur            210 non-null    float64
 30  Meghalaya          210 non-null    float64
 31  Mizoram            210 non-null    float64
 32  Nagaland           210 non-null    float64
 33  Tripura            210 non-null    float64
```

```
    34  ALL INDIA TOTAL     210 non-null     float64
dtypes: float64(34), object(1)
memory usage: 57.5+ KB
```

```python
#df['Date'] = pd.to_datetime(df.Date, dayfirst=True)
df_long['Dates'] = pd.to_datetime(df_long.Dates, dayfirst=True)
```

RegionWise Analysis

```python
df['NR'] = df['Punjab']+ df['Haryana']+ df['Rajasthan']+ df['Delhi']+df['UP']+df['Uttarakhand

df['WR'] = df['Chhattisgarh']+df['Gujarat']+df['MP']+df['Maharashtra']+df['Goa']+df['DNH']

df['SR'] = df['Andhra Pradesh']+df['Telangana']+df['Karnataka']+df['Kerala']+df['Tamil Nadu']

df['ER'] = df['Bihar']+df['Jharkhand']+ df['Odisha']+df['West Bengal']+df['Sikkim']

df['NER'] =df['Arunachal Pradesh']+df['Assam']+df['Manipur']+df['Meghalaya']+df['Mizoram']+df
```

```python
fig = go.Figure()

fig.add_trace(go.Scatter(
    x=df.Date, y=df.NR,
    mode='lines+markers',
    name='Northern region',
    marker=dict(
            color='rgba(300, 50, 50, 0.8)',
            size=5,
            line=dict(
                color='DarkSlateGrey',
                width = 1
                    )
                )
))

fig.add_trace(go.Scatter(
    x=df.Date, y=df.SR,
    mode='lines+markers',
    name='Southern Region',
    marker=dict(
            color='rgba(50, 300, 50, 0.8)',
            size=5,
            line=dict(
                color='DarkSlateGrey',
                width = 1
                    )
                )
))
```

```python
fig.add_trace(go.Scatter(
    x=df.Date, y=df.ER,
    mode='lines+markers',
    name='Eastern Region',
    marker=dict(
        color='rgba(50, 50, 300, 0.8)',
        size=5,
        line=dict(
            color='DarkSlateGrey',
            width = 1
                )
            )
))

fig.add_trace(go.Scatter(
    x=df.Date, y=df.WR,
    mode='lines+markers',
    name='Western Region',
    marker=dict(
        color='rgba(300, 100, 200, 0.8)',
        size=5,
        line=dict(
            color='DarkSlateGrey',
            width = 1
                )
            )
))

fig.add_trace(go.Scatter(
    x=df.Date, y=df.NER,
    mode='lines+markers',
    name='North-Eastern',
    marker=dict(
        color='rgba(100, 200, 300, 0.8)',
        size=5,
        line=dict(
            color='DarkSlateGrey',
            width = 1
                )
            )
))


fig.update_xaxes(
    rangeslider_visible=True,
    rangeselector=dict(
        buttons=list([
            dict(count=1, label="1m", step="month", stepmode="backward"),
            dict(count=3, label="3m", step="month", stepmode="backward"),
            dict(count=6, label="6m", step="month", stepmode="backward"),
            dict(step="all")
```

```
        ])
    )
)

fig.update_layout(title='Power Consumption in Various Region')
fig.update_layout(width=800,height=500)
fig.show()
```

```
df1= df[['Date', 'Punjab', 'Haryana', 'Rajasthan', 'Delhi', 'UP',
        'Uttarakhand', 'HP', 'J&K', 'Chandigarh', 'Chhattisgarh', 'Gujarat',
        'MP', 'Maharashtra', 'Goa', 'DNH',
        'Andhra Pradesh', 'Telangana', 'Karnataka', 'Kerala', 'Tamil Nadu',
        'Pondy', 'Bihar', 'Jharkhand', 'Odisha', 'West Bengal', 'Sikkim',
        'Arunachal Pradesh', 'Assam', 'Manipur', 'Meghalaya', 'Mizoram',
        'Nagaland', 'Tripura']]

df1 = df1.set_index('Date')
bcr.bar_chart_race(df1, figsize=(4, 3.5),period_length =500,filename = None, title='power usa
```

```
/opt/conda/lib/python3.7/site-packages/bar_chart_race/_make_chart.py:286: UserWarning:

FixedFormatter should only be used together with FixedLocator

/opt/conda/lib/python3.7/site-packages/bar_chart_race/_make_chart.py:287: UserWarning:

FixedFormatter should only be used together with FixedLocator
```

Monthly

```
monthly_df = df_long.groupby([df_long.Dates.dt.year, df_long.Dates.dt.month,df_long.States,df
monthly_df.index = monthly_df.index.set_names(['year', 'month','State','Region','latitude','l
monthly_df = monthly_df.reset_index()
monthly_df['month'] = monthly_df['month'].apply(lambda x: calendar.month_abbr[x])


monthly_df.head()
```

year	month		State	Region	latitude	longitude	Usage

```
fig = px.sunburst(monthly_df, path=['Region', 'State','month'], values='Usage',
                  color='Usage',
                  color_continuous_scale='RdBu')
fig.update_layout(title='Click various Regions/States to view power distribution')
fig.update_layout( width=800,height=600)
fig.show()
```

```
fig = px.bar(monthly_df, x="Region", y="Usage",color='State',animation_frame = 'month')
fig.update_layout(xaxis={'categoryorder':'total descending'})
fig.update_layout(title='Region-wise Bar plots')
fig.show()
```

Lockdown

```
df_before = df.iloc[0:150,:]
df_after = df.iloc[151:,]


fig = go.Figure()
fig.add_trace(go.Scatter( x=df_before['Date'], y=df_before['Gujarat'], name='Gujarat before l
    line=dict(width=2,dash='dot',color='firebrick')
))
fig.add_trace(go.Scatter( x=df_before['Date'], y=df_before['Maharashtra'], name='Maharashtra
    line=dict(width=2,dash='dot',color='coral')
))

fig.add_trace(go.Scatter( x=df_before['Date'], y=df_before['MP'], name='MP before lockdown',f
    line=dict(width=2,dash='dot',color='darkred')
))

fig.add_trace(go.Scatter(x=df_after['Date'], y=df_after['Gujarat'],name='Gujarat after lockdo
    line=dict(color='firebrick', width=2)
))

fig.add_trace(go.Scatter(x=df_after['Date'], y=df_after['Maharashtra'],name='Maharashtra afte
```

```
        line=dict(color='coral', width=2)
))

fig.add_trace(go.Scatter(x=df_after['Date'], y=df_after['MP'],name='MP after lockdown',fill='
        line=dict(color='darkred', width=2)
))

fig.update_layout(title='Power Consumption in top 3 WR states')
fig.update_layout( width=800,height=500)
fig.show()
```

```
fig = go.Figure()
fig.add_trace(go.Scatter( x=df_before['Date'], y=df_before['Karnataka'], name='Karnataka befo
        line=dict(width=2,dash='dot',color='skyblue')
))
fig.add_trace(go.Scatter( x=df_before['Date'], y=df_before['Tamil Nadu'], name='Tamil Nadu be
        line=dict(width=2,dash='dot',color='lightblue')
))

fig.add_trace(go.Scatter( x=df_before['Date'], y=df_before['Telangana'], name='Telangana befo
        line=dict(width=2,dash='dot',color='midnightblue')
))
```

```
fig.add_trace(go.Scatter(x=df_after['Date'], y=df_after['Karnataka'],name='Karnataka after lo
    line=dict(color='skyblue', width=2)
))

fig.add_trace(go.Scatter(x=df_after['Date'], y=df_after['Tamil Nadu'],name='Tamil Nadu after
    line=dict(color='lightblue', width=2)
))

fig.add_trace(go.Scatter(x=df_after['Date'], y=df_after['Telangana'],name='Telangana after lo
    line=dict(color='midnightblue', width=2)
))

fig.update_layout(title='Power Consumption in top 3 WR states')
fig.update_layout( width=800,height=500)
fig.show()
```

```
fig = go.Figure()
fig.add_trace(go.Scatter( x=df_before['Date'], y=df_before['Rajasthan'], name='Rajasthan befo
    line=dict(width=2,dash='dot',color='darkviolet')
))

fig.add_trace(go.Scatter( x=df_before['Date'], y=df_before['UP'], name='UP before lockdown',f
    line=dict(width=2,dash='dot',color='deeppink')
```

```
))


fig.add_trace(go.Scatter( x=df_before['Date'], y=df_before['Haryana'], name='Haryana before l
    line=dict(width=2,dash='dot',color='indigo')
))

fig.add_trace(go.Scatter(x=df_after['Date'], y=df_after['Rajasthan'],name='Rajasthan after lo
    line=dict(color='darkviolet', width=2)
))

fig.add_trace(go.Scatter(x=df_after['Date'], y=df_after['UP'],name='UP after lockdown',fill='
    line=dict(color='deeppink', width=2)
))


fig.add_trace(go.Scatter(x=df_after['Date'], y=df_after['Haryana'],name='Haryana after lockdo
    line=dict(color='indigo', width=2)
))

fig.update_layout(title='Power Consumption in top 3 NR states')
fig.update_layout( width=800,height=500)
fig.show()
```

MAx

```python
WR_df = df_long[df_long['Regions']=='WR']
NR_df = df_long[df_long['Regions']=='NR']
SR_df = df_long[df_long['Regions']=='SR']
ER_df = df_long[df_long['Regions']=='ER']
NER_df = df_long[df_long['Regions']=='NER']



fig= go.Figure(go.Indicator(
    mode = "gauge+number",
    value = WR_df['Usage'].max(),
    title = {'text': "Max Power Usage In WR:Maharashtra 13/05/2020"},
    gauge = {
        'axis': {'range': [None, 500], 'tickwidth': 1},
        'threshold': {
            'line': {'color': "red", 'width': 4},
            'thickness': 0.75,
            'value': 490}}
))

fig.show()
```

```python
fig = go.Figure(go.Indicator(
    mode = "gauge+number",
    value = NR_df['Usage'].max(),
    title = {'text': "Max Power Usage In NR :UP 09/05/2020"},
    gauge = {
        'axis': {'range': [None, 500], 'tickwidth': 1},
        'threshold': {
            'line': {'color': "red", 'width': 4},
            'thickness': 0.75,
            'value': 490}}
))
fig.update_layout(legend_title_text='State    Date::UP')
fig.show()
```

```python
fig = go.Figure(go.Indicator(
    mode = "gauge+number",
    value = SR_df['Usage'].max(),
    title = {'text': "Max Power Usage In SR : Tamil Nadu  01/11/2019"},
```

```
    gauge = {
        'axis': {'range': [None, 500], 'tickwidth': 1},
        'threshold': {
            'line': {'color': "red", 'width': 4},
            'thickness': 0.75,
            'value': 490}}
))

fig.show()
```

```
fig = go.Figure(go.Indicator(
    mode = "gauge+number",
    value = ER_df['Usage'].max(),
    title = {'text': "Max Power Usage In ER: West Bangal 04/05/2020"},
    gauge = {
        'axis': {'range': [None, 500], 'tickwidth': 1},
        'threshold': {
            'line': {'color': "red", 'width': 4},
            'thickness': 0.75,
            'value': 490}}
))
```

```
        fig.show()
```

```
    fig = go.Figure(go.Indicator(
        mode = "gauge+number",
        value = NER_df['Usage'].max(),
        title = {'text': "Max Power Usage In NER: Assam 05/05/2020"},
        gauge = {
            'axis': {'range': [None, 500], 'tickwidth': 1},
            'threshold': {
                'line': {'color': "red", 'width': 4},
                'thickness': 0.75,
                'value': 490}}
    ))

    fig.show()
```

Maps

```
df_long = pd.read_csv('../input/india-power/long_data.csv')
df_long.dropna(inplace = True)


fig = px.scatter_geo(df_long,'latitude','longitude', color="Regions",
                     hover_name="States", size="Usage",
                     animation_frame="Dates", scope='asia')
fig.update_geos(lataxis_range=[5,35], lonaxis_range=[65, 100])
fig.show()
```