FLOOD MONITORING SYSTEM

622621121018: HARIGOPINATH.V

INTRODUCTION :

A flood monitoring system is used to monitor a rise in water levels. The system comprises sensors that are deployed in cities or any area of interest. The sensors can be connected to either the main electricity or can be solar-powered. These sensors are deployed on bridges, wells, lakes, or beaches to measure water levels in real-time and continuously send data remotely to the centralized data system management via different networks such as GSM, mobile cell networks, or Wi-Fi

BUILD A USE CASE SMART HOME AUTOMATION:

Home automation is a network of hardware, communication, and electronic interfaces that work to integrate everyday devices with one another via the Internet. Each device has sensors and is connected through WiFi, so you can manage them from your smartphone or tablet whether you're at home, or miles away.

INTRODUCTION TO COMPUTER VISION WITH 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 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 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 preserved 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/smart-home-dataset-with-weather-information/HomeC.csv
```

Data Exploration

Data reading

```
df = pd.read_csv(os.path.join(dirname, filename), low_memory=False)
```

```
df.head()
time	use [kW]	gen [kW]	House overall [kW]	Dishwasher [kW]	Furnace 1 [kW]	Furnace 2 [kW]	Home office [kW]	Fridge [kW]	Wine cellar [kW]	...	visibility	summary	apparentTemperature	pressure	windSpeed	cloudCover	windBearing	precipIntensity	dewPoint	precipProbability
0	1451624400	0.932833	0.003483	0.932833	0.000033	0.020700	0.061917	0.442633	0.124150	0.006983	...	10.0	Clear	29.26	1016.91	9.18	cloudCover	282.0	0.0	24.4	0.0
1	1451624401	0.934333	0.003467	0.934333	0.000000	0.020717	0.063817	0.444067	0.124000	0.006983	...	10.0	Clear	29.26	1016.91	9.18	cloudCover	282.0	0.0	24.4	0.0
2	1451624402	0.931817	0.003467	0.931817	0.000017	0.020700	0.062317	0.446067	0.123533	0.006983	...	10.0	Clear	29.26	1016.91	9.18	cloudCover	282.0	0.0	24.4	0.0
3	1451624403	1.022050	0.003483	1.022050	0.000017	0.106900	0.068517	0.446583	0.123133	0.006983	...	10.0	Clear	29.26	1016.91	9.18	cloudCover	282.0	0.0	24.4	0.0
4	1451624404	1.139400	0.003467	1.139400	0.000133	0.236933	0.063983	0.446533	0.122850	0.006850	...	10.0	Clear	29.26	1016.91	9.18	cloudCover	282.0	0.0	24.4	0.0
5 rows × 32 columns
```

```
df.columns
Index(['time', 'use [kW]', 'gen [kW]', 'House overall [kW]', 'Dishwasher [kW]',
       'Furnace 1 [kW]', 'Furnace 2 [kW]', 'Home office [kW]', 'Fridge [kW]',
       'Wine cellar [kW]', 'Garage door [kW]', 'Kitchen 12 [kW]',
       'Kitchen 14 [kW]', 'Kitchen 38 [kW]', 'Barn [kW]', 'Well [kW]',
       'Microwave [kW]', 'Living room [kW]', 'Solar [kW]', 'temperature',
       'icon', 'humidity', 'visibility', 'summary', 'apparentTemperature',
       'pressure', 'windSpeed', 'cloudCover', 'windBearing', 'precipIntensity',
       'dewPoint', 'precipProbability'],
      dtype='object')
```

Data Preprocessing

```
# Rename columns to remove spaces and the kW unit
df.columns = [col[:-5].replace(' ','_') if 'kW' in col else col for col in df.columns]

# Drop rows with nan values
df = df.dropna()

# The columns "use" and "house_overall" are the same, so let's remove the 'house_overall' column
df.drop(['House_overall'], axis=1, inplace=True)

# The columns "gen" and "solar" are the same, so let's remove the 'solar' column
df.drop(['Solar'], axis=1, inplace=True)

# drop rows with cloudCover column values that are not numeric (bug in sensors) and convert column to numeric
```

```
df = df[df['cloudCover']!='cloudCover']
df["cloudCover"] = pd.to_numeric(df["cloudCover"])

# Create columns that regroup kitchens and furnaces
df['kitchen'] = df['Kitchen_12'] + df['Kitchen_14'] + df['Kitchen_38']
df['Furnace'] = df['Furnace_1'] + df['Furnace_2']

# Convert "time" column (which is a unix timestamp) to a Y-m-d H-M-S
import time
start_time = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(int(df['time'].iloc[0])))
time_index = pd.date_range(start_time, periods=len(df), freq='min')
time_index = pd.DatetimeIndex(time_index)
df = df.set_index(time_index)
df = df.drop(['time'], axis=1)
```
Data Analysis
```
df.shape
(503852, 31)
df.columns
Index(['use', 'gen', 'Dishwasher', 'Furnace_1', 'Furnace_2', 'Home_office',
       'Fridge', 'Wine_cellar', 'Garage_door', 'Kitchen_12', 'Kitchen_14',
       'Kitchen_38', 'Barn', 'Well', 'Microwave', 'Living_room', 'temperature',
       'icon', 'humidity', 'visibility', 'summary', 'apparentTemperature',
       'pressure', 'windSpeed', 'cloudCover', 'windBearing', 'precipIntensity',
       'dewPoint', 'precipProbability', 'kitchen', 'Furnace'],
      dtype='object')
# lower frist letter of a string
func = lambda s: s[:1].lower() + s[1:] if s else ''
cols = list(df.dtypes.keys())
categ_cols = [col for col in cols if df[col].dtype=='O']
num_cols = [col for col in cols if col not in categ_cols]
print('categ_cols : ', categ_cols)
print('num_cols : ', num_cols)
categ_cols :  ['icon', 'summary']
num_cols :  ['use', 'gen', 'Dishwasher', 'Furnace_1', 'Furnace_2', 'Home_office', 'Fridge',
'Wine_cellar', 'Garage_door', 'Kitchen_12', 'Kitchen_14', 'Kitchen_38', 'Barn', 'Well',
'Microwave', 'Living_room', 'temperature', 'humidity', 'visibility', 'apparentTemperature',
'pressure', 'windSpeed', 'cloudCover', 'windBearing', 'precipIntensity', 'dewPoint',
'precipProbability', 'kitchen', 'Furnace']
# Let's remove rows with values that appear less than a certain percentage %

def remove_less_percent(col, percent):
   keys_to_conserve = [key for key,value in df[col].value_counts(normalize=True).items() if
value>=percent]
   return df[df[col].isin(keys_to_conserve)]

print(len(df))
df = remove_less_percent('summary', 0.05)
print(len(df))
```

```
df = remove_less_percent('icon', 0.05)
print(len(df))
503852
466308
466308
# plot bars of unique values of categorical columns

def plot_bars(col):

    import matplotlib.pyplot as plt
    from matplotlib.pyplot import figure

    figure(figsize=(14, 8), dpi=80)
    plt.xticks(rotation = 90)

    D = df[col].value_counts(normalize=True).to_dict()

    plt.bar(*zip(*D.items()))
    plt.show()
plot_bars('icon')

plot_bars('summary')

df['use'].resample(rule='D').mean().plot(figsize=(25,5))
<AxesSubplot:>

df['temperature'].resample(rule='D').mean().plot(figsize=(25,5))
<AxesSubplot:>

df['cloudCover'].resample(rule='D').mean().plot(figsize=(25,5))
<AxesSubplot:>

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

general_energy_cols = ['gen', 'use']
general_energy_per_month = df[general_energy_cols].resample('M').sum() # for energy we
use sum to calculate overall consumption in period

plt.figure(figsize=(20,10))

sns.lineplot(data=general_energy_per_month, dashes=False)
<AxesSubplot:>

rooms_energy_cols = ['Home_office', 'Wine_cellar','Garage_door',
                     'kitchen', 'Barn', 'Well','Living_room']
```

```
rooms_energy_per_month = df[rooms_energy_cols].resample('M').mean()

plt.figure(figsize=(20,8))

sns.lineplot(data=rooms_energy_per_month, dashes=False)
<AxesSubplot:>
```

The energy consumption of kitchen, garage and the well remained almost the same throughout the year
There's seasonality of energy consumption in other parts of the house :
A clear spike in september in the energy consumed by the wine cellar and the home office
A clear downtrend in the summer for the barn energy consumption

```
equipements_cols = ['Microwave', 'Dishwasher', 'Furnace', 'Fridge']

equipements_energy_per_month = df[equipements_cols].resample('M').mean()

plt.figure(figsize=(25,8))

sns.lineplot(data= equipements_energy_per_month, dashes=False)
<AxesSubplot:>
```

The usage of the furnace decreases in the summer

```
weather_columns = ['temperature','humidity', 'visibility', 'apparentTemperature',
                'windSpeed', 'dewPoint']

weather_per_month = df[weather_columns].resample('M').mean()

plt.figure(figsize=(25,8))

sns.lineplot(data=weather_per_month, dashes=False)
<AxesSubplot:>

fig,ax = plt.subplots(figsize=(20, 18))
corr = df[weather_columns].corr()
sns.heatmap(corr, annot=True, vmin=-1.0, vmax=1.0, center=0)
ax.set_title('Correlation of Weather Information', size=20)
plt.show()
```

Modeling : What are we trying to solve ?
Case 1 Change Detection : Detecting excessive energy consumption in advance and preventing increase in usage fees.
Case 2 Predict Future Consumption : Predicting future energy consumption and generation by utilizing weather information and optimizing energy supply.

Inspired by kohei-mu

Case 1 : Change detection

The change point is the point at which the trends in time series data change over time. Outliers indicate a momentary abnormal condition (rapid decrease or increase), while change points mean that the abnormal condition does not return to its original state and continue.

Let's use ChangeFinder algorithm

ChangeFinder is an algorithm used to detect change points. ChangeFinder uses the log-likelihood based on the SDAR(Sequencially Discounting AR) algorithm to calculate the change score. SDAR algorithm introduces a discounting parameter into the AR algorithm to reduce the influence of past data, so that even non-stationary time series data can be learned robustly.

ChangeFinder has two steps of model training:

Training STEP1 Train a time series model at each data point using the SDAR algorithm Based on the trained time series model, calculate the likelihood that the data points at the next time point will appear Calculate the logarithmic loss and use it as an outlier score

$Score(x_t) = -\log P_{t-1}(x_t|x_1,x_2,\ldots,x_{t-1})$

Smoothing Step Smooth the outlier score within the smoothing window(W). By smoothing, the score due to outliers is attenuated, and it is possible to determine whether the abnormal condition has continued for a long time.

$Score\_smoothed(x_t) = \frac{1}{W}\sum_{t=t-W+1}^{t} Score(x_i)$

Training STEP2 Using the score obtained by smoothing, train the model with the SDAR algorithm Based on the trained time series model, calculate the likelihood that the data points at the next time point will appear Calculate the logarithmic loss and use it as an change score

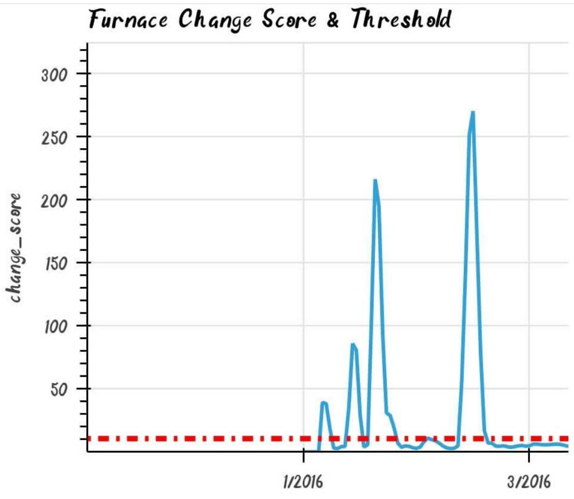
```
# Let's install & import changefinder python library
!pip install changefinder
import changefinder
```

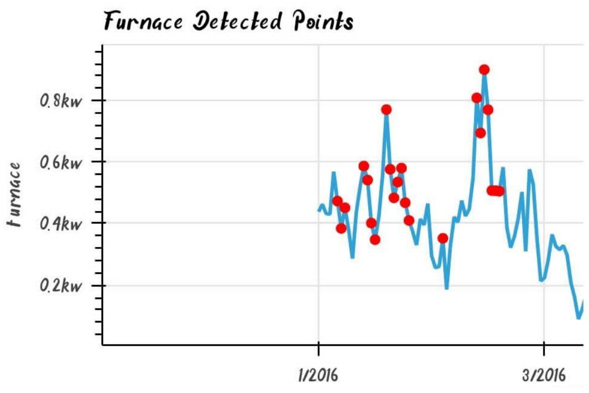
```
Collecting changefinder
  Downloading changefinder-0.03.tar.gz (3.8 kB)
Requirement already satisfied: numpy in /opt/conda/lib/python3.7/site-packages (from changefinder) (1.19.5)
Requirement already satisfied: scipy in /opt/conda/lib/python3.7/site-packages (from changefinder) (1.6.3)
Requirement already satisfied: statsmodels in /opt/conda/lib/python3.7/site-packages (from changefinder) (0.12.2)
Requirement already satisfied: nose in /opt/conda/lib/python3.7/site-packages (from changefinder) (1.3.7)
Requirement already satisfied: patsy>=0.5 in /opt/conda/lib/python3.7/site-packages (from statsmodels->changefinder) (0.5.1)
Requirement already satisfied: pandas>=0.21 in /opt/conda/lib/python3.7/site-packages (from statsmodels->changefinder) (1.2.4)
Requirement already satisfied: python-dateutil>=2.7.3 in
/opt/conda/lib/python3.7/site-packages (from pandas>=0.21->statsmodels->changefinder)
(2.8.1)
Requirement already satisfied: pytz>=2017.3 in /opt/conda/lib/python3.7/site-packages (from
pandas>=0.21->statsmodels->changefinder) (2021.1)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packages (from
patsy>=0.5->statsmodels->changefinder) (1.15.0)
Building wheels for collected packages: changefinder
  Building wheel for changefinder (setup.py) ... - \ done
  Created wheel for changefinder: filename=changefinder-0.3-py3-none-any.whl size=3823
sha256=5eb778d280a414eb76141c271f4656ab223ce0ff2c618fd8cbe3741da7324fbb
  Stored in directory:
/root/.cache/pip/wheels/48/8f/4c/ba040e4cad3dbf0409c5c38ac9d3f92f947ea490412cdf3400
Successfully built changefinder
Installing collected packages: changefinder
Successfully installed changefinder-0.3
WARNING: Running pip as root will break packages and permissions. You should install
packages reliably by using venv: https://pip.pypa.io/warnings/venv
```

```python
from scipy import stats
import holoviews as hv
from holoviews import opts
hv.extension('bokeh')


def chng_detection(col, _r=0.01, _order=1, _smooth=10):
    cf = changefinder.ChangeFinder(r=_r, order=_order, smooth=_smooth)
    ch_df = pd.DataFrame()
    ch_df[col] = df[col].resample('D').mean()

    # calculate the change score
    ch_df['change_score'] = [cf.update(i) for i in ch_df[col]]
    ch_score_q1 = stats.scoreatpercentile(ch_df['change_score'], 25)
    ch_score_q3 = stats.scoreatpercentile(ch_df['change_score'], 75)
    thr_upper = ch_score_q3 + (ch_score_q3 - ch_score_q1) * 3

    anom_score = hv.Curve(ch_df['change_score'])
    anom_score_th = hv.HLine(thr_upper).opts(color='red', line_dash="dotdash")

    anom_points = [[ch_df.index[i],ch_df[col][i]] for i, score in
enumerate(ch_df["change_score"]) if score > thr_upper]
    org = hv.Curve(ch_df[col],label=col).opts(yformatter='%.1fkw')
    detected = hv.Points(anom_points, label=f"{col} detected").opts(color='red',
legend_position='bottom', size=5)

    return ((anom_score * anom_score_th).opts(title=f"{col} Change Score & Threshold") + \
            (org * detected).opts(title=f"{col} Detected Points")).opts(opts.Curve(width=800,
height=300, show_grid=True, tools=['hover'])).cols(1)
```

Discounting parameter $r (0<r<1)$: The smaller this value, the greater the influence of the past data points and the greater the variation in the change score
Order parameter for AR $order$: How far past data points are included in the model
Smoothing window $smooth$: The greater this parameter is, the easier it is to capture the essential changes rather than the outliers, but if it is too large, it will be difficult to capture the changes themselves

```
chng_detection('use', _r=0.001, _order=1, _smooth=3)
df.columns
Index(['use', 'gen', 'Dishwasher', 'Furnace_1', 'Furnace_2', 'Home_office',
       'Fridge', 'Wine_cellar', 'Garage_door', 'Kitchen_12', 'Kitchen_14',
       'Kitchen_38', 'Barn', 'Well', 'Microwave', 'Living_room', 'temperature',
       'icon', 'humidity', 'visibility', 'summary', 'apparentTemperature',
       'pressure', 'windSpeed', 'cloudCover', 'windBearing', 'precipIntensity',
       'dewPoint', 'precipProbability', 'kitchen', 'Furnace'],
      dtype='object')
chng_detection('Furnace', _r=0.001, _order=1, _smooth=3)
```





OUT PUT:

kaggle kernels output offmann/smart-home-dataset -p /path/to/dest