MEASURE ENERGY CONSUMPTION

INTRODUCTION:

"Measure Energy Consumption" aims to develop a system or methodology to accurately quantifyand monitor energy usage in various settings. This project focuses on implementing efficient methods to measure energy consumption in order to promote energy conservation and optimize energy usage. By providing accurate data on energy consumption.

Household Power Consumption Dataset

The household power consumption dataset is a multivariate time series dataset that describes the electricity consumption for a single household over four years.The data was collected between December 2006 and November 2010 and observations of power consumption within the household were collected every minute.

It is a multivariate series comprised of seven variables (besides the date and time); they are:

- **global_active_power**: The total active power consumed by the household (kilowatts).
- **global_reactive_power**: The total reactive power consumed by the household (kilowatts).
- **voltage**: Average voltage (volts).
- **global_intensity**: Average current intensity (amps).
- **sub_metering_1**: Active energy for kitchen (watt-hours of active energy).
- **sub_metering_2**: Active energy for laundry (watt-hours of active energy).
- **sub_metering_3**: Active energy for climate control systems (watt-hours of active energy)

Load Dataset

The dataset can be downloaded from the UCI Machine Learning repository as a single 20 megabyte .zip file:

household_power_consumption.zip

The dataset is downloaded and unzip into current working directory. Now we have the file "*household_power_consumption.txt*" that is about 127 megabytes in size and contains all of the observations

Inspect the data file.

Below are the first five rows of data (and the header) from the raw data file.

```
1 Date;Time;Global_active_power;Global_reactive_power;Voltage;Global_intensity;Sub_metering_1;Sub_metering_2;Sub_metering_3
2 16/12/2006;17:24:00;4.216;0.418;234.840;18.400;0.000;1.000;17.000
3 16/12/2006;17:25:00;5.360;0.436;233.630;23.000;0.000;1.000;16.000
4 16/12/2006;17:26:00;5.374;0.498;233.290;23.000;0.000;2.000;17.000
5 16/12/2006;17:27:00;5.388;0.502;233.740;23.000;0.000;1.000;17.000
6 16/12/2006;17:28:00;3.666;0.528;235.680;15.800;0.000;1.000;17.000
7 ...
```

The data does have missing values; for example, we can see 2-3 days worth of missing data around 28/04/2007.

...

28/4/2007;00:20:00;0.492;0.208;236.240;2.200;0.000;0.000;0.000

28/4/2007;00:21:00;?;?;?;?;?;?;?;

28/4/2007;00:22:00;?;?;?;?;?;?;?;

28/4/2007;00:23:00;?;?;?;?;?;?;?;

28/4/2007;00:24:00;?;?;?;?;?;?;?;

...

We can start-off by loading the data file as a Pandas DataFrame and summarize the loaded data.

We can use the read_csv() function to load the data.
Specifically, we need to do a few custom things:

- Specify the separate between columns as a semicolon (sep=';')
- Specify that line 0 has the names for the columns (header=0)
- Specify that we have lots of RAM to avoid a warning that we are loading the data as an array of objects instead of an array of numbers, because of the '?' values for missing data (low_memory=False).
- Specify that it is okay for Pandas to try to infer the date-time format when parsing dates, which is way faster (infer_datetime_format=True)
- Specify that we would like to parse the date and time columns together as a new column called 'datetime' (parse_dates={'datetime':[0,1]})
- Specify that we would like our new 'datetime' column to be the index for the DataFrame (index_col=['datetime']).
- Putting all of this together, we can now load the data and summarize the loaded shape and first few rows.

```
# load all data
dataset = read_csv('household_power_consumption.txt', sep=';', header=0, low_memory=False, infer_datetime_format=True,
parse_dates={'datetime':[0,1]}, index_col=['datetime'])
# summarize
print(dataset.shape)
print(dataset.head())
```

- Next, we can mark all missing values indicated with a '?' character with a NaN value, which is a float.
- This will allow us to work with the data as one array of floating point values rather than mixed types, which is less efficient.

```
# mark all missing values
dataset.replace('?', nan, inplace=True)
```

- Now we can create a new column that contains the remainder of the sub-metering, using the calculation from the previous section.

```
# add a column for for the remainder of sub metering
values = dataset.values.astype('float32')
dataset['sub_metering_4'] = (values[:,0] * 1000 / 60) - (values[:,4] + values[:,5] + values[:,6])
```

- We can now save the cleaned-up version of the dataset to a new file; in this case we will just change the file extension to .csv and save the dataset as '*household_power_consumption.csv*'.

```
# save updated dataset
dataset.to_csv('household_power_consumption.csv')
```

- To confirm that we have not messed-up, we can re-load the dataset and summarize the first five rows.

```
# load the new file
dataset = read_csv('household_power_consumption.csv',
header=None)
print(dataset.head())
```

Tying all of this together, the complete example of loading, cleaning-up, and saving the dataset is listed below.

```
# load and clean-up data
from numpy import nan
from pandas import read_csv
# load all data
dataset = read_csv('household_power_consumption.txt', sep=';',
header=0, low_memory=False, infer_datetime_format=True,
parse_dates={'datetime':[0,1]}, index_col=['datetime'])
# summarize
```

```
print(dataset.shape)
print(dataset.head())
# mark all missing values
dataset.replace('?', nan, inplace=True)
# add a column for for the remainder of sub metering
values = dataset.values.astype('float32')
dataset['sub_metering_4'] = (values[:,0] * 1000 / 60) - (values[:,4]
+ values[:,5] + values[:,6])
# save updated dataset
dataset.to_csv('household_power_consumption.csv')
```

```
# load the new dataset and summarize
dataset = read_csv('household_power_consumption.csv',
header=0, infer_datetime_format=True, parse_dates=['datetime'],
index_col=['datetime'])
print(dataset.head())
```

Running the example first loads the raw data and summarizes the shape and first five rows of the loaded data.

```
(2075259, 7)

              Global_active_power  ...  Sub_metering_3
datetime                           ...
2006-12-16 17:24:00            4.216  ...            17.0
2006-12-16 17:25:00            5.360  ...            16.0
2006-12-16 17:26:00            5.374  ...            17.0
2006-12-16 17:27:00            5.388  ...            17.0
2006-12-16 17:28:00            3.666  ...            17.0
```

The dataset is then cleaned up and saved to a new file.

We load this new file and again print the first five rows, showing the removal of the date and time columns and addition of the new sub-metered column.

```
              Global_active_power  ...  sub_metering_4
datetime                           ...
2006-12-16 17:24:00            4.216  ...       52.266670
2006-12-16 17:25:00            5.360  ...       72.333336
2006-12-16 17:26:00            5.374  ...       70.566666
2006-12-16 17:27:00            5.388  ...       71.800000
2006-12-16 17:28:00            3.666  ...       43.100000
```

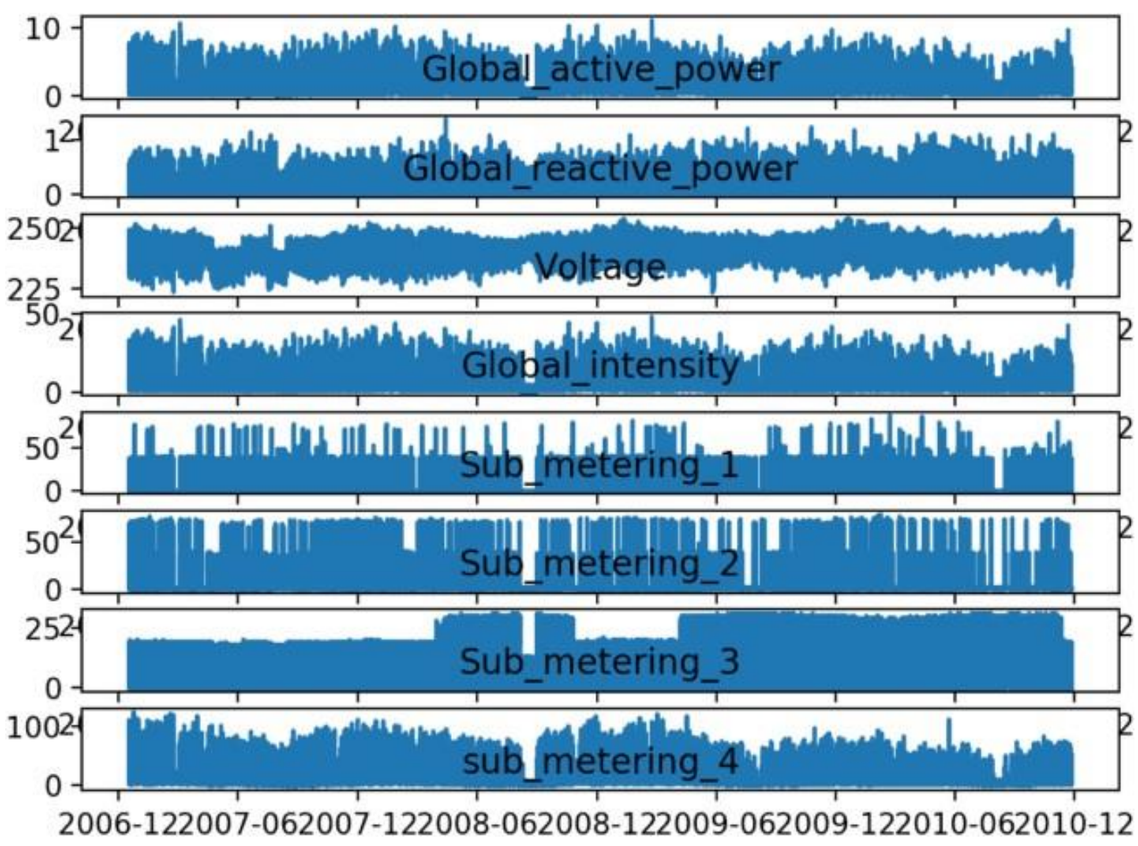
We can peek inside the new '*household_power_consumption.csv*' file and check that the missing observations are marked with an empty column, that pandas will correctly read as NaN, for example around row 190,499:

```
...
2007-04-28 00:20:00,0.492,0.208,236.240,2.200,0.000,0.000,0.0,8.2
2007-04-28 00:21:00,,,,,,,
2007-04-28 00:22:00,,,,,,,
2007-04-28 00:23:00,,,,,,,
2007-04-28 00:24:00,,,,,,,
2007-04-28 00:25:00,,,,,,,
...
```

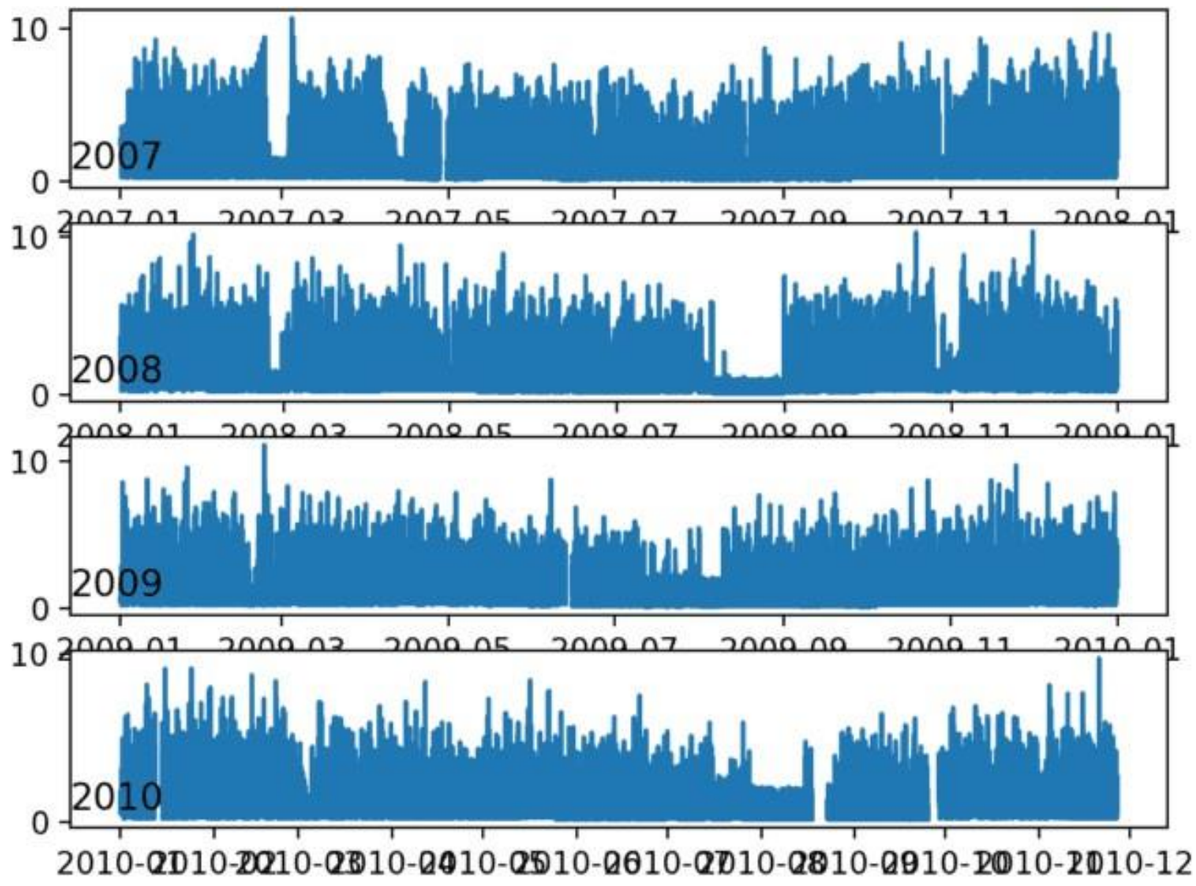
Patterns in Observations Over Time

The data is a multivariate time series and the best way to understand a time series is to create lineplots.The complete example is listed below.

```
# line plots

from pandas import read_csvfrom
matplotlib import pyplot# load the new
file
dataset = read_csv('household_power_consumption.csv', header=0, infer_datetime_format=True,parse_dates=['datetime'],
 index_col=['datetime'])

# line plot for each variable
pyplot.figure()
for i in range(len(dataset.columns)):
 pyplot.subplot(len(dataset.columns), 1, i+1)name =
 dataset.columns[i] pyplot.plot(dataset[name])
 pyplot.title(name, y=0)
pyplot.show()
```



Running the example creates one single image with four line plots, one for each full year (or mostly full years) of data in the dataset.We also seem to see a downward trend over the summermonthsThese may show an annual seasonal pattern in consumption.

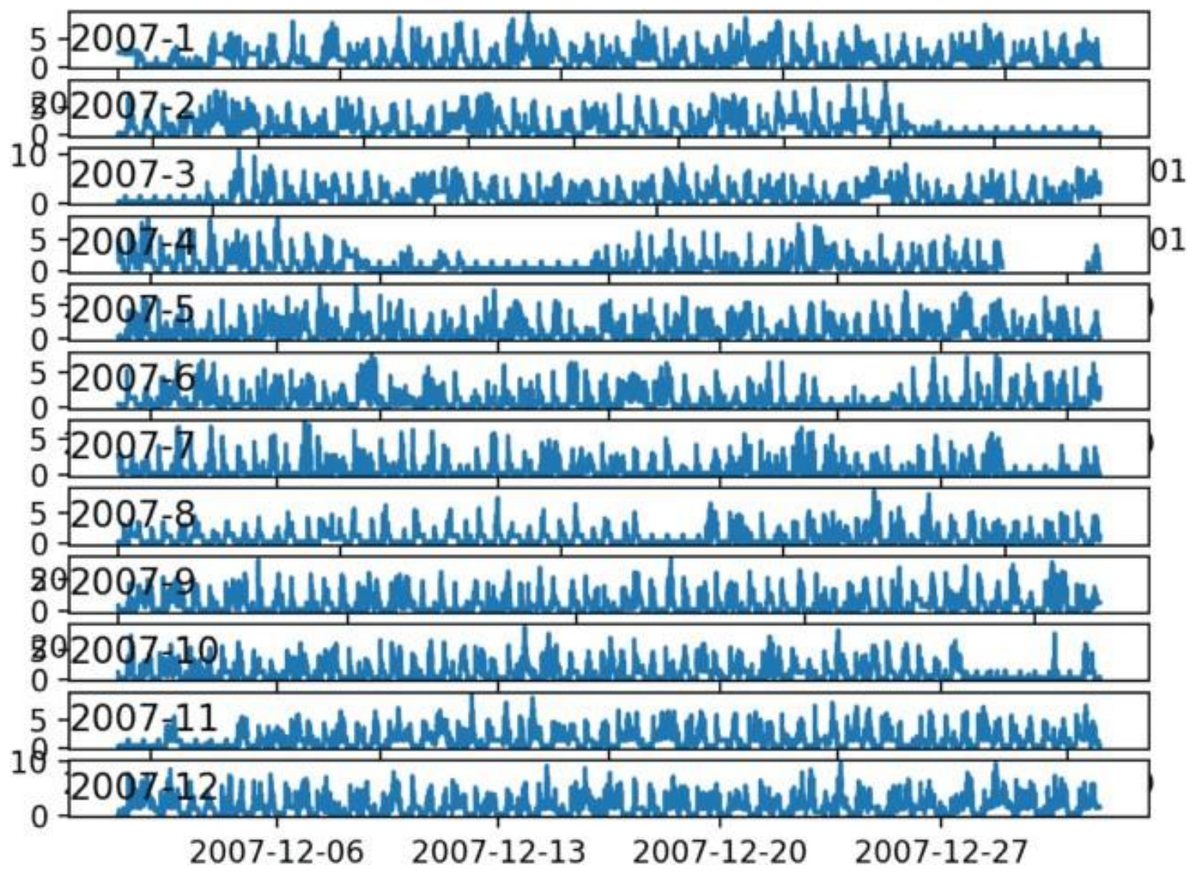


The complete example is listed below.

```
# monthly line plots

from pandas import read_csv
from matplotlib import pyplot#
load the new file
dataset = read_csv('household_power_consumption.csv', header=0, infer_datetime_format=True,
 parse_dates=['datetime'], index_col=['datetime'])

# plot active power for each year
months = [x for x in range(1, 13)]
pyplot.figure()
for i in range(len(months)):

 # prepare subplot
 ax = pyplot.subplot(len(months), 1, i+1)#
 determine the month to plot
 month = '20 7-' + str(months[i])

 # get all observations for the monthresult =
 dataset[month]
 # plot the active power for the month
pyplot.plot(result['Global_active_power'])#
add a title to the subplot
pyplot.title(month, y=0, loc='left')
pyplot.show()
```

Running the example creates a single image with 12 line plots, one for each month in 2007.Wecan see the sign-wave of power consumption of the days within each month

These may represent vacation periods where the home was unoccupied and where powerconsumption was minimal.



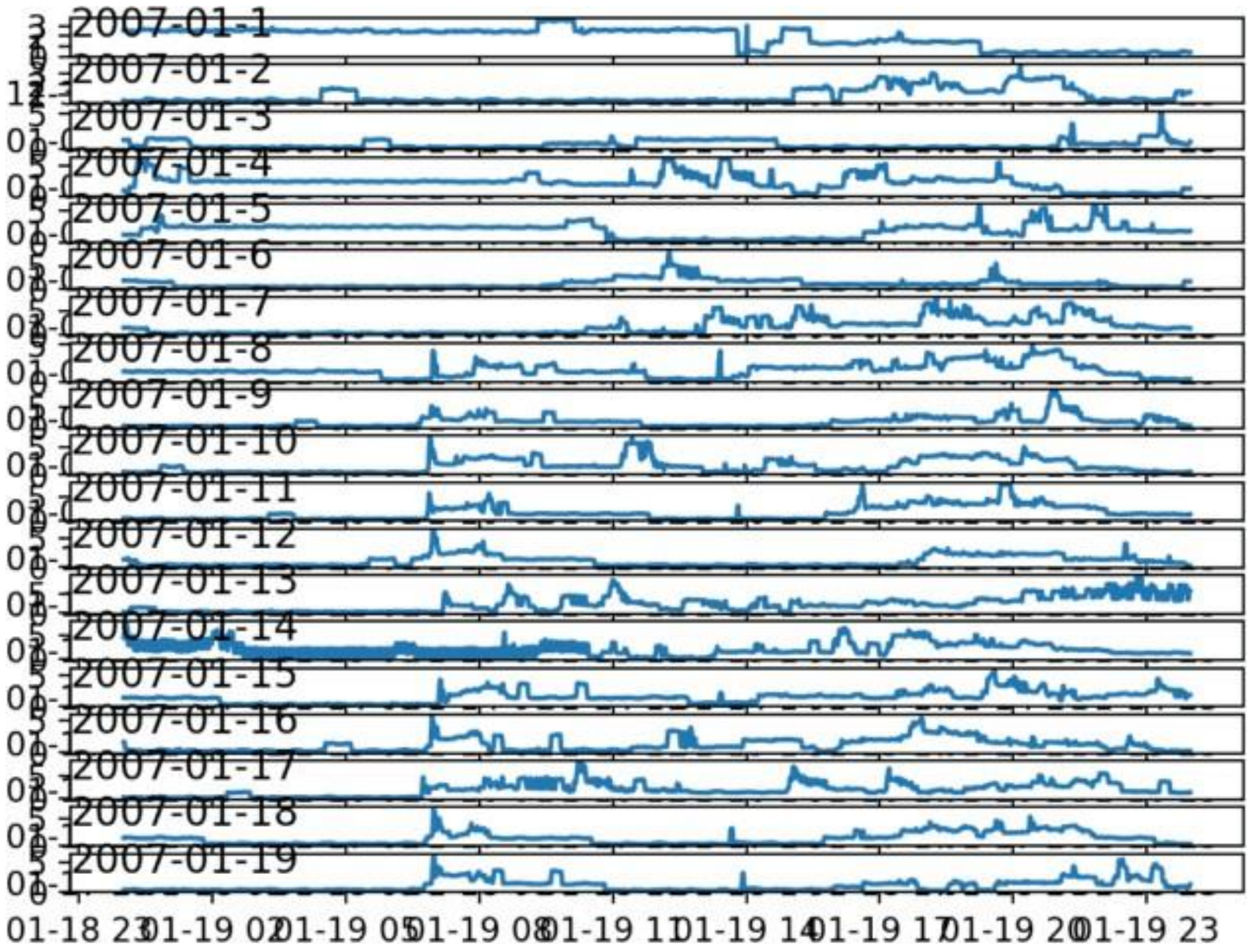
Finally, we can zoom in one more level and take a closer look at power consumption at the dailylevel.We would expect there to be some pattern to consumption each day, and perhaps differences in days over a week.

```
# daily line plots

from pandas import read_csvfrom
matplotlib import pyplot# load the new
file
dataset = read_csv('household_power_consumption.csv', header=0, infer_date time_format=True,parse_dates=['datetime'],
 index_col=['datetime'])

# plot active power for each yeardays = [x
for x in range(1, 20)] pyplot.figure()
for i in range(len(days)):

 # prepare subplot

 ax = pyplot.subplot(len(days), 1, i+1)
 # determine the day to plot day = '2 0
 7-01-' + str(days[i])
 # get all observations for the dayresult =
 dataset[day]
 # plot the active power for the day
pyplot.plot(result['Global_active_power'])# a d     a title
to the subplot pyplot.title(day, y=0, loc='left')
pyplot.show()
```

Running the example creates a single image with 20 line plots, one for the first 20 days in January2007.

There is commonality acro ss the days; for example, many days consumption starts early morning,around 6-7AM.

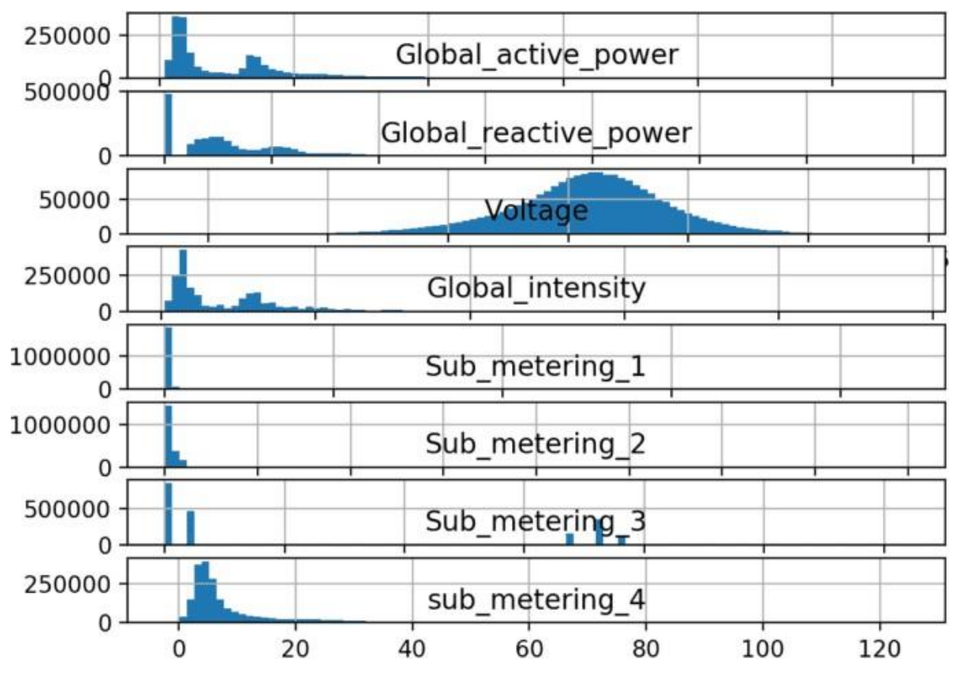
Time of year, specifically the season and the weather that it brings, will be an important factor inmodeling this data, as would be expected.



Time Series Data Distributions

We can investigate the distributions of the data by reviewing histograms.# histogram
plots
from pandas import read_csvfrom
matplotlib import pyplot# load the
new file
dataset = read_csv('household_power_consumption.csv', header=0, infer_datetime_format=True,
parse_dates=['datetime'], index_col=['datetime'])

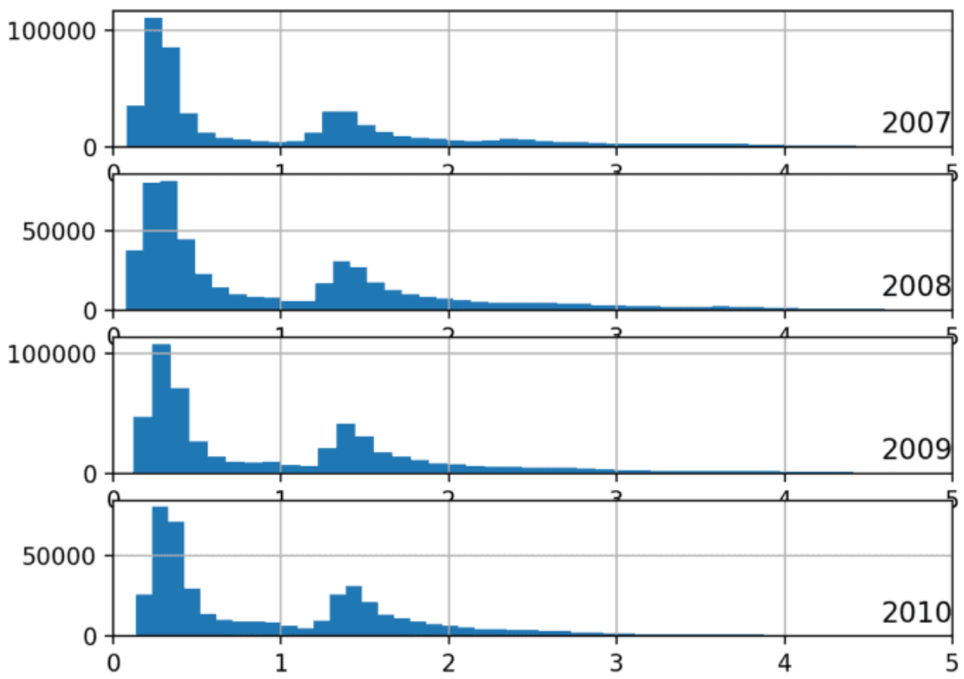
# histogram plot for each variablepyplot.figure()
for i in range(len(dataset.columns)):
pyplot.subplot(len(dataset.columns), 1, i+1)name =
dataset.columns[i]
dataset[name].hist(bins=100)
pyplot.title(name, y=0) pyplot.show()
We can also see that distribution of voltage data is strongly Gaussian.



The distribution of active power appears to be bi-modal, meaning it l**o** ks like it has two meangroups of observations.We can investigate this further by looking at the distribution of active power consumption for the four full years of data.
The complete example is listed below.

```
# yearly histogram plots

from pandas import read_csvfrom
matplotlib import pyplot# load the new
file
dataset = read_csv('household_power_consumption.csv', header=0, infer_datetime_format=True,parse_dates=['datetime'],
 index_col=['datetime'])

# plot active power for each year years = ['2 0  7',
'2008', '2 0  9', '2010']
pyplot.figure()
for i in range(len(years)):
# prepare subplot
ax = pyplot.subplot(len(years), 1, i+1)# determine the year to plot
 year = years[i]
 # get all observations for the yearresult = dataset[str(year)]
 # plot the active power for the year
 result['Global_active_power'].hist(bins=100)# zoom in on
 the distribution
 ax.set_xlim(0, 5)
 # ad a title to the subplot
  pyplot.title(str(year), y=0, loc='right')
 pyplot.show()
```

There is a long tail on the distribution to higher kilowatt values. It might open the door to notions of discretizing the data and separating it into peak 1, peak 2 or long tail. These groups or clusters for usage on a day or hour may be helpful in developing a predictive model.



It is possible that the identified groups may vary over the seasons of the year.# monthly histogram plots
from pandas import read_csvfrom
matplotlib import pyplot# load the new
file
dataset = read_csv('household_power_consumption.csv', header=0, infer_datetime_format=True,parse_dates=['datetime'],
index_col=['datetime'])

plot active power for each year months =
[x for x in range(1, 13)]pyplot.figure()
for i in range(len(months)):

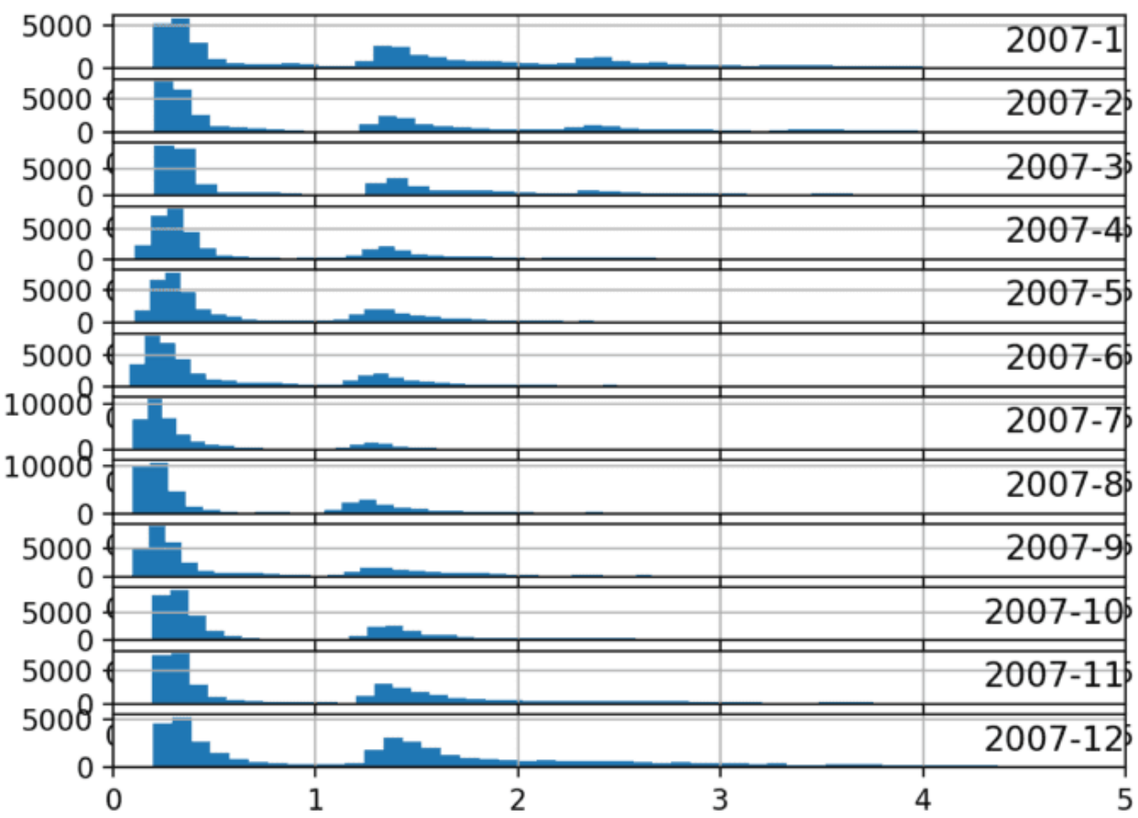
prepare subplot

ax = pyplot.subplot(len(months), 1, i+1)

determine the month to plot month = '2**0**
7-' + str(months[i])
get all observations for the monthresult =
dataset[month]
plot the active power for the month
result['Global_active_power'].hist(bins=100)# zoom in on
the distribution
ax.set_xlim(0, 5)

# a**d** a title to the subplot
pyplot.title(month, y=0, loc='right')
pyplot.show()

We can also see a thicker more prominent tail toward larger kilowatt values for the coolermonths of December through to March.



Ideas on Modeling:

In this section, we will take a closer look at three main areas when working with the data; they are:

- Problem Framing
- Data Preparation
- Modeling Methods

Problem Framing

There does not appear to be a seminal publication for the dataset to demonstrate the intendedway to frame the data in a predictive modeling problem.
Four examples include:

- Forecast hourly consumption for the next day.
- Forecast daily consumption for the next week.
- Forecast daily consumption for the next month.
- Forecast monthly consumption for the next year.

Generally, these types of forecasting problems are referred to as multi-step forecasting. Modelsthat make use of all of the variables might be referred to as a multivariate multi-step forecastingmodels.

Data Preparation:

Daily differencing may be useful to adjust for the daily cycle in the data. Annual differencing may be useful to adjust for any yearly cycle in the data.
Normalization may aid in reducing the variables with differing units to the same scale.

Indicating the time of day, to account for the likelihood of people being home or not.Indicating whether a day is a weekday or weekend.

Indicating the season, which may lead to the type or amount environmental control systems beingused.

Modeling Methods

- Naive Methods.
- Classical Linear Methods..
- Machine Learning Methods.
- Deep Learning Methods.

- Clasical linear methods include techniques are very effective for univariate time seriesforecasting.
- SARIMA
- ETS (triple exponential smoothing)Machine Learning Methods
- k-nearest neighbors.
- Support vector machines
- Decision trees
- Random forest

Gradient boosting machines

Deep Learning Methods:

Generally, neural networks have not proven very effective at autoregression type problems.

Nevertheless, techniques such as convolutional neural networks are able to automatically learncomplex features from raw data, including one-dimensional signal data. And recurrent neural networks, such as the long short-term memory network, are capable of directly learning across multiple parallel sequences of input data.
Further, combinations of these methods, such as CNN LSTM and ConvLSTM, have proveneffective on time series tasks.