TUTORIAL 01 : CAISO OASIS Renewables

Goal

Your mission, should you choose to accept it, is to replicate the following two graphs from the CAISO Renewables Reporting page (http://www.caiso.com/market/Pages/ReportsBulletins/RenewablesReporting.aspx).

Example

Renewables Reporting First Page

Setup

In [1]:
```
import sqlite3
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
from datetime import datetime
from dateutil import parser
from pandasql import PandaSQL
pdf = PandaSQL('sqlite:///:memory:', persist=True)

# make graphs look modern and pretty
import seaborn as sns
sns.set()

# make tables look pretty
# (cribbed from Brandon Rhodes' tutorials)
from IPython.core.display import HTML
css = open('style-table.css').read() + open('style-notebook.css').read()
HTML('<style>{}</style>'.format(css))
```

Out[1]:

Problem 01: Replicate the 24-Hour Renewables Production Report

Monday, October 28, 2019

24 Hour Production

- [ ] Find the data for this report
- [ ] Create dataframe with this report data
- [ ] Query for this particular day
- [ ] Query for the subtotals
- [ ] Display a plot that looks similar to the graph above

Answer : Find the data for this report

Download the dataset

```
curl -O https://s3.us-west-1.wasabisys.com/eap/energy-dashboard/data/data-oasis-daily
-renewables-output/db/data-oasis-daily-renewables-output_00.db.gz
```

Unzip

```
gunzip data-oasis-daily-renewables-output_00.db.gz
```

Verify the database

```
sqlite3 data-oasis-daily-renewables-output_00.db
> .tables
> select count(*) from renewable;
> select count(*) from total;
```

Answer : Create dataframe with this report data

In [2]:
```
# create the connection to the unzipped database in this directory
cnx = sqlite3.connect(r'./data-oasis-daily-renewables-output_00.db')

# df1 : renewable(s) table
df1 = pd.read_sql("select * from renewable", cnx)
# df1a : total(s) table
df1a = pd.read_sql("select * from total", cnx)
```

In [3]:
```
# Examine the table and verify the dtypes
# ...especially the 'date' column
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 83442 entries, 0 to 83441
Data columns (total 11 columns):
id               0 non-null object
date             83442 non-null object
hour             83442 non-null int64
geothermal       83402 non-null float64
biomass          83402 non-null float64
biogas           83401 non-null float64
small_hydro      83399 non-null float64
wind_total       83374 non-null float64
solar_pv         60631 non-null float64
solar_thermal    60631 non-null float64
solar            22744 non-null float64
dtypes: float64(8), int64(1), object(2)
memory usage: 7.0+ MB
```

In [4]:
```
# Examine the table and verify the dtypes
# ...especially the 'date' column
df1a.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 83447 entries, 0 to 83446
Data columns (total 8 columns):
id            0 non-null object
date          83447 non-null object
hour          83447 non-null int64
renewables    83397 non-null float64
nuclear       83422 non-null float64
thermal       83380 non-null float64
imports       83419 non-null float64
hydro         83407 non-null float64
dtypes: float64(5), int64(1), object(2)
memory usage: 5.1+ MB
```

In [5]:
```
# The magic of merge. It does the join intelligently for us. Kinda scary, reall
y.
df2 = df1.merge(df1a)
df2.head()
```

Out[5]:

	id	date	hour	geothermal	biomass	biogas	small_hydro	wind_total	solar_pv	solar_thermal	solar	renewables
0	None	2011-05-30 00:00:00	1	960.0	367.0	155.0	510.0	1873.0	NaN	NaN	0.0	3864.0
1	None	2011-05-30 00:00:00	2	792.0	370.0	154.0	510.0	1965.0	NaN	NaN	0.0	3792.0
2	None	2011-05-30 00:00:00	3	771.0	368.0	154.0	509.0	1784.0	NaN	NaN	0.0	3586.0
3	None	2011-05-30 00:00:00	4	890.0	369.0	154.0	509.0	1627.0	NaN	NaN	0.0	3550.0
4	None	2011-05-30 00:00:00	5	996.0	373.0	154.0	510.0	1354.0	NaN	NaN	0.0	3387.0

In [6]:
```
# Looks like we have merged correctly and basically appended the 'total' datafr
ame to the 'renewable' dataframe
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 83441 entries, 0 to 83440
Data columns (total 16 columns):
id                0 non-null object
date              83441 non-null object
hour              83441 non-null int64
geothermal        83401 non-null float64
biomass           83401 non-null float64
biogas            83401 non-null float64
small_hydro       83398 non-null float64
wind_total        83374 non-null float64
solar_pv          60631 non-null float64
solar_thermal     60631 non-null float64
solar             22743 non-null float64
renewables        83391 non-null float64
nuclear           83416 non-null float64
thermal           83374 non-null float64
imports           83413 non-null float64
hydro             83401 non-null float64
dtypes: float64(13), int64(1), object(2)
memory usage: 10.8+ MB
```

In [7]:
```
# Sanitize the dataframe by converting the 'date' column to a datetime,
# drop the useless 'id' column, and replace NaN|None values with zeros.
# Q: Why do we have NULL values? B/C the original downloaded data feeds
# had data errors splattered within the data files.
df2['date'] = pd.to_datetime(df2['date'], infer_datetime_format=True)
df2.pop('id')
df2.fillna(0, inplace=True)
```

In [8]:
```
# Create a multi-index and experience the glory of time-series data!
df3 = df2.set_index(['date', 'hour'])
df3.head()
```

Out[8]:

		geothermal	biomass	biogas	small_hydro	wind_total	solar_pv	solar_thermal	solar	renewables	nuclear
date	hour										
2011-05-30	1	960.0	367.0	155.0	510.0	1873.0	0.0	0.0	0.0	3864.0	3432.0
	2	792.0	370.0	154.0	510.0	1965.0	0.0	0.0	0.0	3792.0	3433.0
	3	771.0	368.0	154.0	509.0	1784.0	0.0	0.0	0.0	3586.0	3433.0
	4	890.0	369.0	154.0	509.0	1627.0	0.0	0.0	0.0	3550.0	3433.0
	5	996.0	373.0	154.0	510.0	1354.0	0.0	0.0	0.0	3387.0	3436.0

In [9]:
```
# Notice that the index is a DatetimeIndex with ~ 83K entries
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 83441 entries, (2011-05-30 00:00:00, 1) to (2012-06-22 00:00:00, 2
4)
Data columns (total 13 columns):
geothermal       83441 non-null float64
biomass          83441 non-null float64
biogas           83441 non-null float64
small_hydro      83441 non-null float64
wind_total       83441 non-null float64
solar_pv         83441 non-null float64
solar_thermal    83441 non-null float64
solar            83441 non-null float64
renewables       83441 non-null float64
nuclear          83441 non-null float64
thermal          83441 non-null float64
imports          83441 non-null float64
hydro            83441 non-null float64
dtypes: float64(13)
memory usage: 8.5 MB
```

In [10]:
```
# Notice that the date range in the DatetimeIndex goes from 2011 to 2012??? We
*should*
# have data up to today (2019)...
df3.tail()
```

Out[10]:

		geothermal	biomass	biogas	small_hydro	wind_total	solar_pv	solar_thermal	solar	renewables	nuclear
date	hour										
2012-06-22	20	919.0	320.0	198.0	424.0	2564.0	0.0	0.0	195.0	4620.0	2263.0
	21	920.0	320.0	198.0	430.0	2584.0	0.0	0.0	173.0	4626.0	2265.0
	22	921.0	310.0	199.0	421.0	2703.0	0.0	0.0	158.0	4713.0	2265.0
	23	922.0	287.0	200.0	408.0	2428.0	0.0	0.0	58.0	4302.0	2265.0
	24	925.0	280.0	198.0	395.0	2257.0	0.0	0.0	0.0	4055.0	2265.0

In [11]:
```
# Could this be a sorting issue?
```

In [12]:
```
df4 = df3.sort_index()
df4.tail()
```

Out[12]:

		geothermal	biomass	biogas	small_hydro	wind_total	solar_pv	solar_thermal	solar	renewables	nuclear
date	hour										
2019-10-30	20	268.0	314.0	218.0	202.0	586.0	0.0	0.0	0.0	1588.0	1120.0
	21	268.0	285.0	211.0	192.0	538.0	0.0	0.0	0.0	1494.0	1121.0
	22	268.0	284.0	215.0	193.0	485.0	0.0	0.0	0.0	1445.0	1121.0
	23	268.0	276.0	227.0	194.0	486.0	0.0	0.0	0.0	1451.0	1122.0
	24	267.0	270.0	233.0	174.0	427.0	0.0	0.0	0.0	1371.0	1121.0

In [13]:
```
# Ahhhh, so it was a sorting issue.
# Note to self: remember to sort your indexes.
```

In [14]:
```
# I've currently downloaded through Oct 31, 2019 (or therabouts), so this seems close enough
len(df4.loc['2019'].index.unique())/24
```

Out[14]: 303.0

Answer : Query for this particular day

In [15]:
```
# We are graphing a particular date, so filter a DF accordingly
df5 = df4.loc['2019-10-28']
df5.head()
```

Out[15]:

		geothermal	biomass	biogas	small_hydro	wind_total	solar_pv	solar_thermal	solar	renewables	nuclear
date	hour										
2019-10-28	1	267.0	280.0	207.0	172.0	2564.0	0.0	0.0	0.0	3490.0	1127.0
	2	267.0	276.0	206.0	172.0	2251.0	0.0	0.0	0.0	3172.0	1126.0
	3	266.0	272.0	206.0	171.0	1824.0	0.0	0.0	0.0	2473.0	1127.0
	4	267.0	274.0	206.0	172.0	1428.0	0.0	0.0	0.0	2347.0	1127.0
	5	267.0	280.0	205.0	171.0	1140.0	0.0	0.0	0.0	2063.0	1127.0

Answer : Query for subtotals

In [16]:
```
# Get the max index for each column
peak_prod_idx = df5.idxmax()
peak_prod_idx
```

Out[16]:
```
geothermal       (2019-10-28 00:00:00, 18)
biomass          (2019-10-28 00:00:00, 13)
biogas           (2019-10-28 00:00:00, 19)
small_hydro       (2019-10-28 00:00:00, 8)
wind_total        (2019-10-28 00:00:00, 1)
solar_pv         (2019-10-28 00:00:00, 12)
solar_thermal    (2019-10-28 00:00:00, 11)
solar             (2019-10-28 00:00:00, 1)
renewables       (2019-10-28 00:00:00, 11)
nuclear           (2019-10-28 00:00:00, 1)
thermal          (2019-10-28 00:00:00, 19)
imports          (2019-10-28 00:00:00, 20)
hydro            (2019-10-28 00:00:00, 18)
dtype: object
```

In [17]:
```
# Extract the peak hour from the max index
peak_hour_dict = {}
for idx in peak_prod_idx.index:
    (_, hour) = peak_prod_idx[idx]
    peak_hour_dict[idx] = hour
peak_hour = pd.Series(peak_hour_dict)
peak_hour
```

Out[17]:
```
geothermal       18
biomass          13
biogas           19
small_hydro       8
wind_total        1
solar_pv         12
solar_thermal    11
solar             1
renewables       11
nuclear           1
thermal          19
imports          20
hydro            18
dtype: int64
```

In [18]:
```
daily_peak = df5.max()
daily_peak
```

Out[18]:
```
geothermal         270.0
biomass            324.0
biogas             234.0
small_hydro        192.0
wind_total        2564.0
solar_pv          8553.0
solar_thermal      424.0
solar                0.0
renewables       10264.0
nuclear           1127.0
thermal          15349.0
imports           7849.0
hydro             2193.0
dtype: float64
```

In [19]:
```
daily_total = df5.sum(axis='rows')
daily_total
```

Out[19]:
```
geothermal         6429.0
biomass            7019.0
biogas             5195.0
small_hydro        4158.0
wind_total        20258.0
solar_pv          69583.0
solar_thermal      2751.0
solar                 0.0
renewables       115127.0
nuclear           27021.0
thermal          213202.0
imports          130822.0
hydro             50596.0
dtype: float64
```

Answer : Display a plot that looks similar to the graph above

In [20]:
```
series = [peak_hour, daily_peak, daily_total]
dfdaily = pd.DataFrame(series, index=['peak_hour', 'daily_peak', 'daily_total
'])
dfdaily.transpose()
```

Out[20]:

	peak_hour	daily_peak	daily_total
geothermal	18.0	270.0	6429.0
biomass	13.0	324.0	7019.0
biogas	19.0	234.0	5195.0
small_hydro	8.0	192.0	4158.0
wind_total	1.0	2564.0	20258.0
solar_pv	12.0	8553.0	69583.0
solar_thermal	11.0	424.0	2751.0
solar	1.0	0.0	0.0
renewables	11.0	10264.0	115127.0
nuclear	1.0	1127.0	27021.0
thermal	19.0	15349.0	213202.0
imports	20.0	7849.0	130822.0
hydro	18.0	2193.0	50596.0

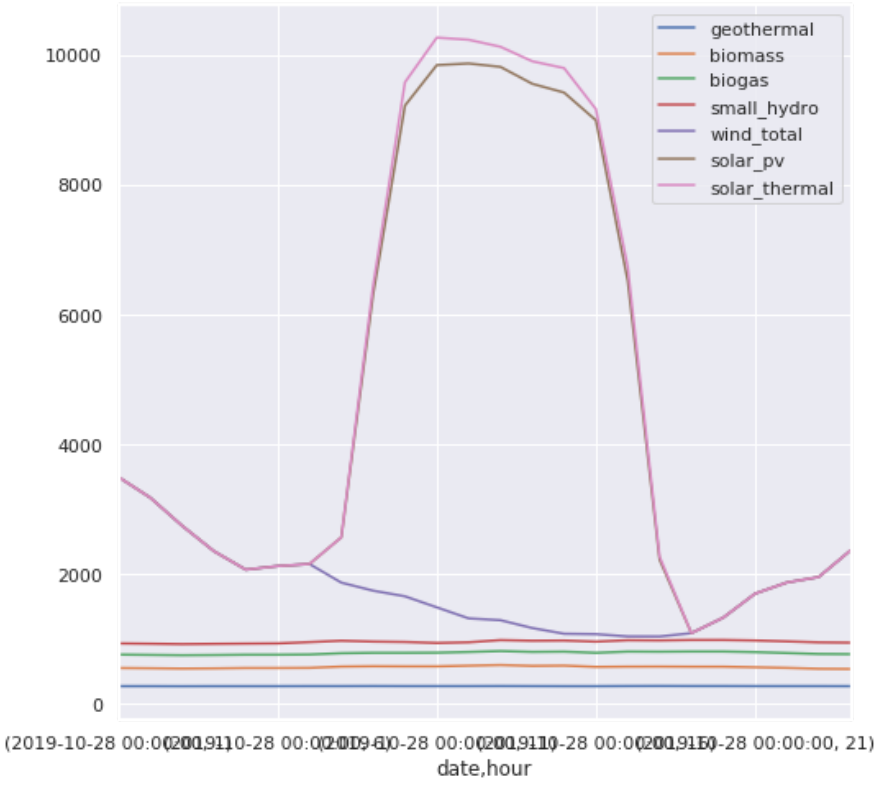
Problem 02: Replicate the Hourly Average Breakdown

Hourly Average Breakdown

Answer : Display a plot that looks similar to the graph above

In [21]:
```
renewables = df5.copy()
renewables.pop('renewables')
renewables.pop('nuclear')
renewables.pop('thermal')
renewables.pop('imports')
renewables.pop('hydro')
renewables.pop('solar')
renewables.plot(figsize=(8,8), stacked=True)
```

Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x7f4c47c7b5f8>



Problem 03 : How ALL the energy sources changing over time?

Answer : Construct a DF with peak_hour, daily_peak, and daily_total for entire timeframe

In [22]:
```
# Convert date into an actual timestamp, need to combine the date with the hour
import statsmodels.api as sm
dfx = df2.copy()
dfx['tdelta'] = pd.to_timedelta(dfx['hour'] -1, 'H')
dfx['ts'] = dfx['date'] + dfx['tdelta']
dfx = dfx.set_index('ts')
dfx.pop('date')
dfx.pop('tdelta')
dfx.pop('hour')
dfx.info()
dfx.head()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 83441 entries, 2011-05-30 00:00:00 to 2012-06-22 23:00:00
Data columns (total 13 columns):
geothermal       83441 non-null float64
biomass          83441 non-null float64
biogas           83441 non-null float64
small_hydro      83441 non-null float64
wind_total       83441 non-null float64
solar_pv         83441 non-null float64
solar_thermal    83441 non-null float64
solar            83441 non-null float64
renewables       83441 non-null float64
nuclear          83441 non-null float64
thermal          83441 non-null float64
imports          83441 non-null float64
hydro            83441 non-null float64
dtypes: float64(13)
memory usage: 8.9 MB
```

Out[22]:

	geothermal	biomass	biogas	small_hydro	wind_total	solar_pv	solar_thermal	solar	renewables	nuclear	therm
ts											
2011-05-30 00:00:00	960.0	367.0	155.0	510.0	1873.0	0.0	0.0	0.0	3864.0	3432.0	3225
2011-05-30 01:00:00	792.0	370.0	154.0	510.0	1965.0	0.0	0.0	0.0	3792.0	3433.0	3207
2011-05-30 02:00:00	771.0	368.0	154.0	509.0	1784.0	0.0	0.0	0.0	3586.0	3433.0	3244
2011-05-30 03:00:00	890.0	369.0	154.0	509.0	1627.0	0.0	0.0	0.0	3550.0	3433.0	3233
2011-05-30 04:00:00	996.0	373.0	154.0	510.0	1354.0	0.0	0.0	0.0	3387.0	3436.0	2886

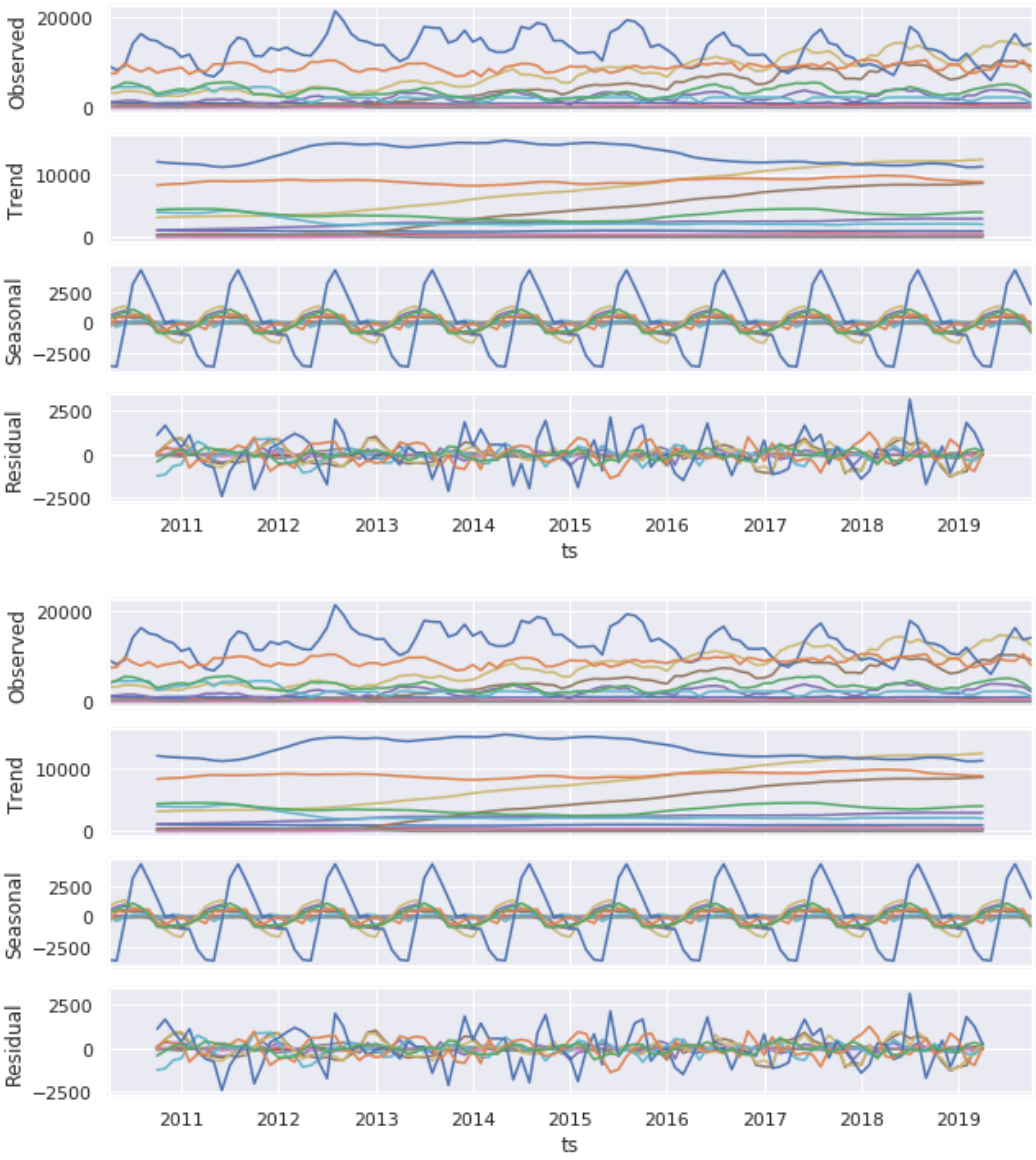
In [31]:
```
all_daily_peak = dfx.resample('D').max().resample('M').mean()
all_daily_peak.fillna(0, inplace=True)
all_daily_peak.head()
```

Out[31]:

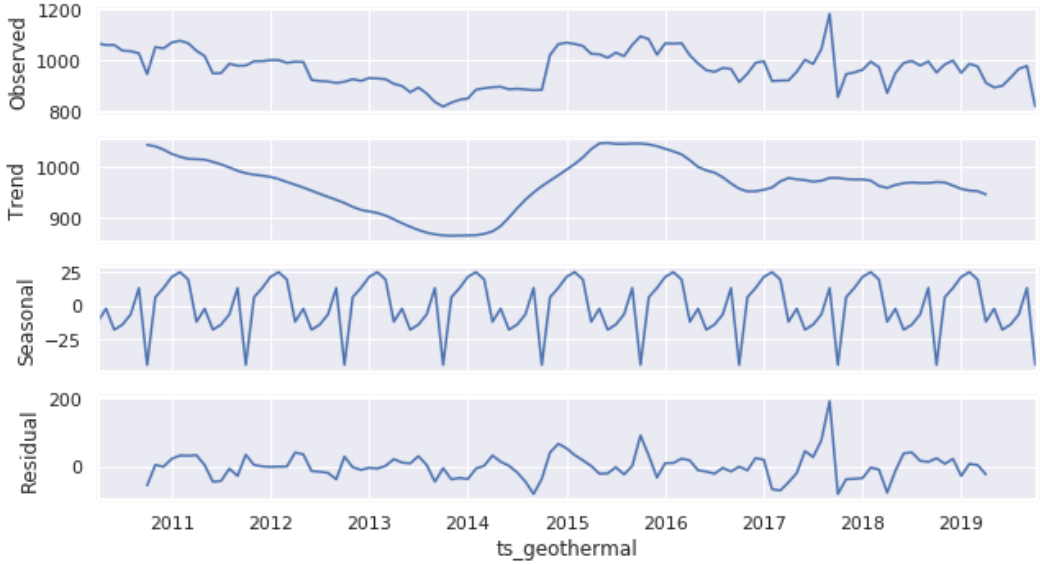
	geothermal	biomass	biogas	small_hydro	wind_total	solar_pv	solar_thermal	solar	renewa
ts									
2010-04-30	1066.636364	301.000000	175.090909	312.272727	1289.090909	0.0	0.0	292.909091	3200.1
2010-05-31	1060.000000	245.258065	162.193548	377.870968	1351.032258	0.0	0.0	348.419355	3257.1
2010-06-30	1060.366667	368.866667	171.733333	431.000000	1549.900000	0.0	0.0	387.966667	3702.8
2010-07-31	1037.935484	429.032258	169.258065	401.387097	1591.000000	0.0	0.0	382.903226	3652.8
2010-08-31	1035.806452	405.677419	170.322581	442.064516	1398.967742	0.0	0.0	376.387097	3491.6

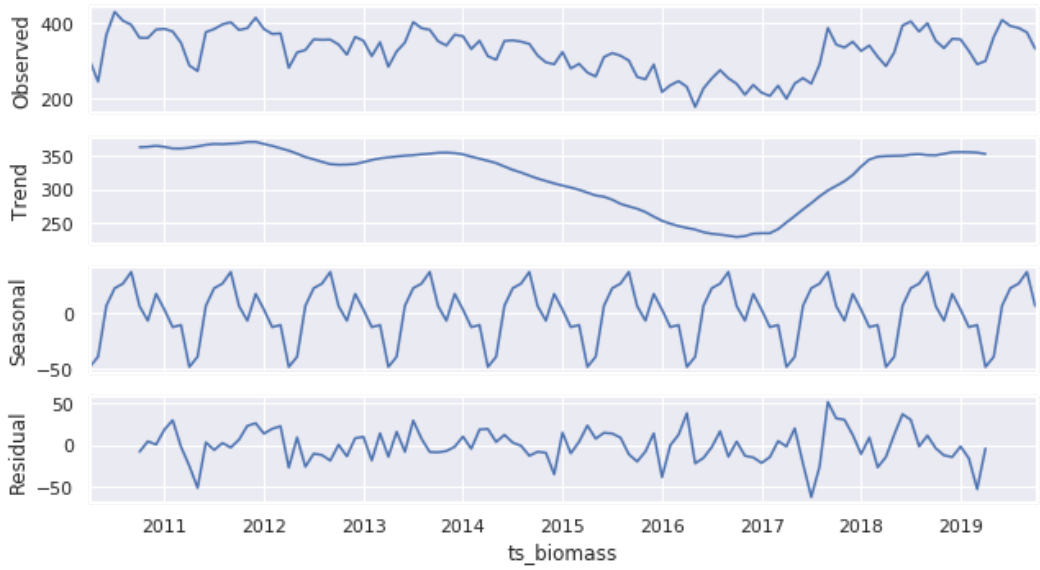
In [32]:
```
decomp_daily_peak = sm.tsa.seasonal_decompose(all_daily_peak, model='additive')
matplotlib.rcParams['figure.figsize'] = [9.0, 5.0]
decomp_daily_peak.plot()
```

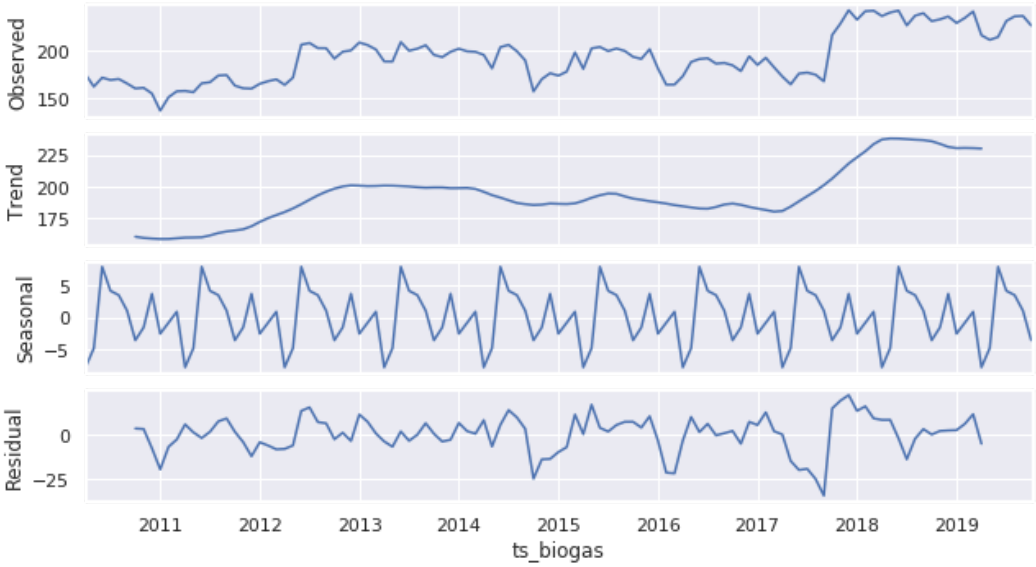
Out[32]:

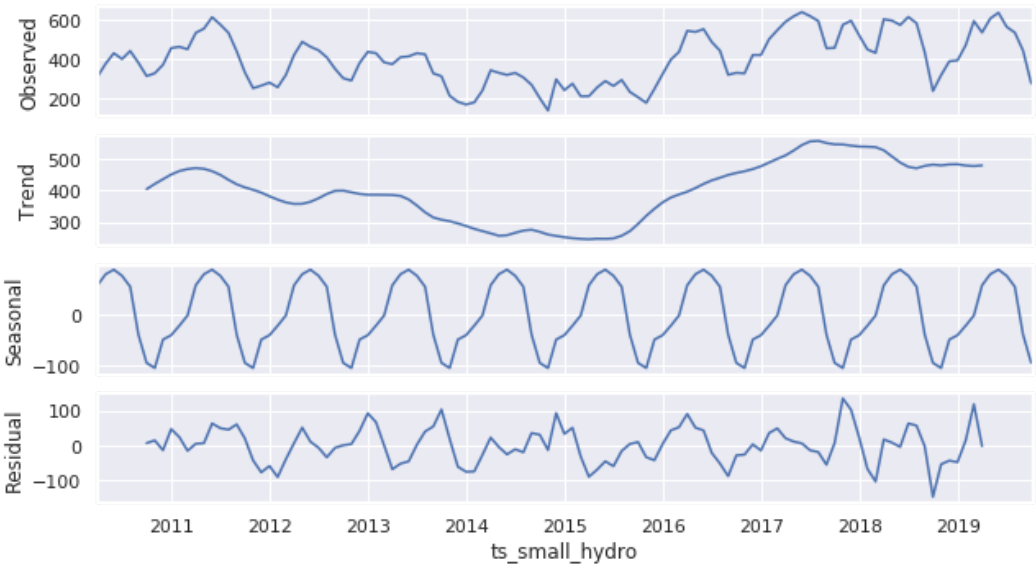


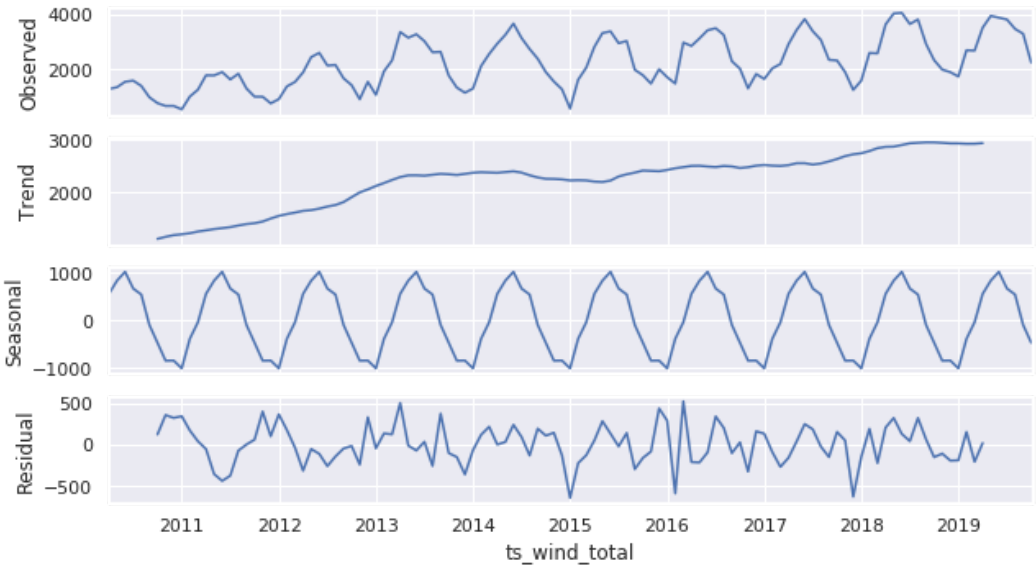
In [41]:
```
matplotlib.rcParams['figure.figsize'] = [9.0, 5.0]
for col in list(all_daily_peak):
    v = all_daily_peak[col]
    # hack
    v.index.names = ['ts_%s' % col]
    # hack
    decomp_daily_peak = sm.tsa.seasonal_decompose(v, model='additive')
    decomp_daily_peak.plot()
```

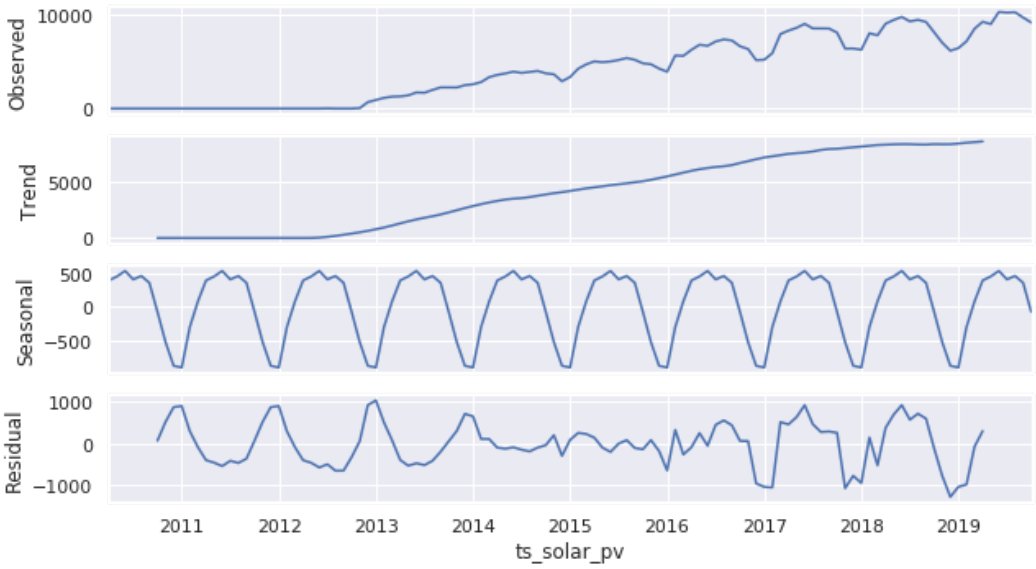

Observed
1200
1000
800
Trend
1000
900
Seasonal
25
0
−25
Residual
200
0
2011
2012
2013
2014
2015
2016
2017
2018
2019
ts_geothermal

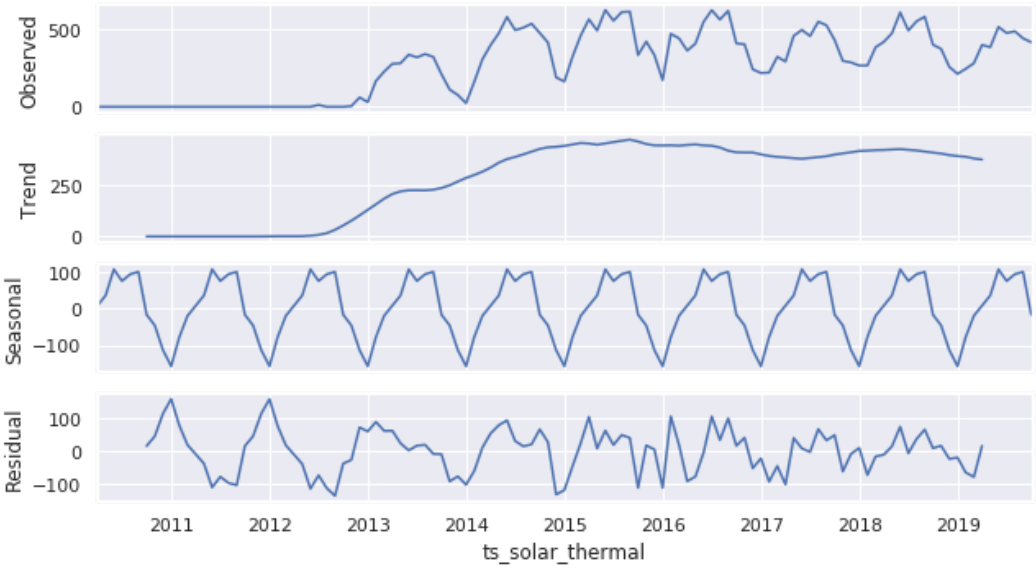

Observed
400
200
Trend
350
300
250
Seasonal
0
−50
Residual
50
0
−50
2011
2012
2013
2014
2015
2016
2017
2018
2019
ts_biomass

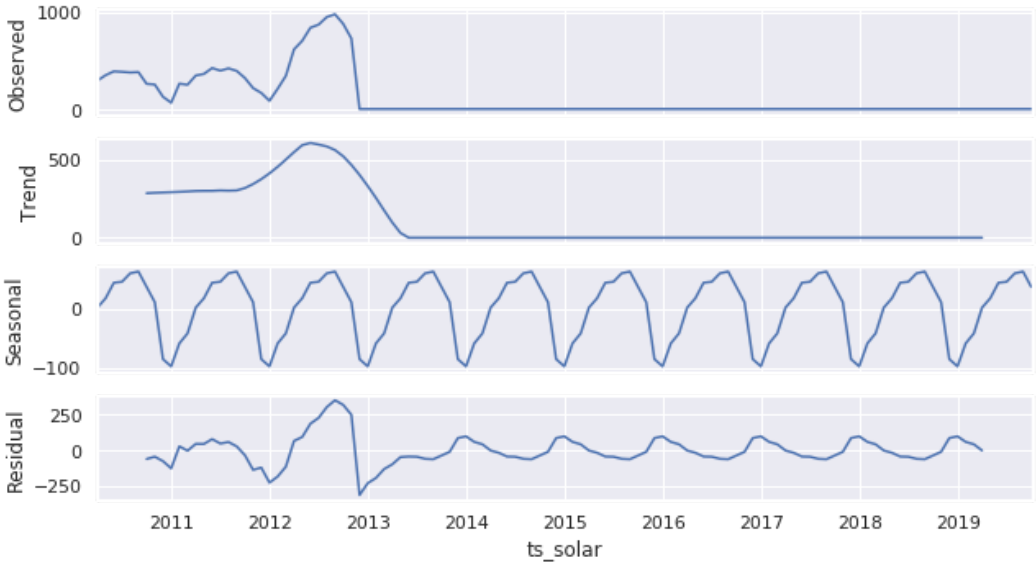

Observed
200
150
Trend
225
200
175
Seasonal
5
0
−5
Residual
0
−25
2011
2012
2013
2014
2015
2016
2017
2018
2019
ts_biogas

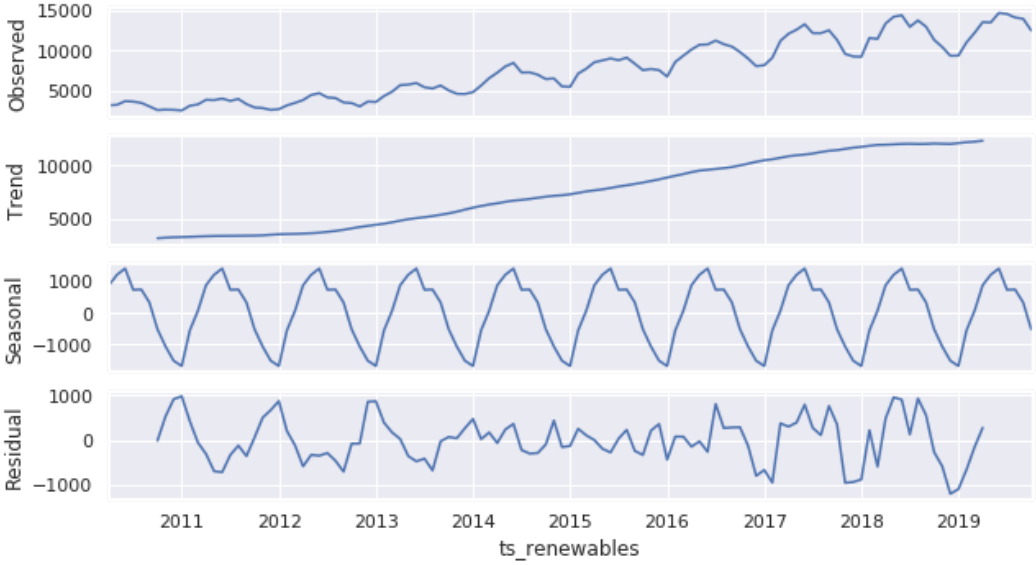

Observed
600
400
200
Trend
500
400
300
Seasonal
0
−100
Residual
100
0
−100
2011
2012
2013
2014
2015
2016
2017
2018
2019
ts_small_hydro

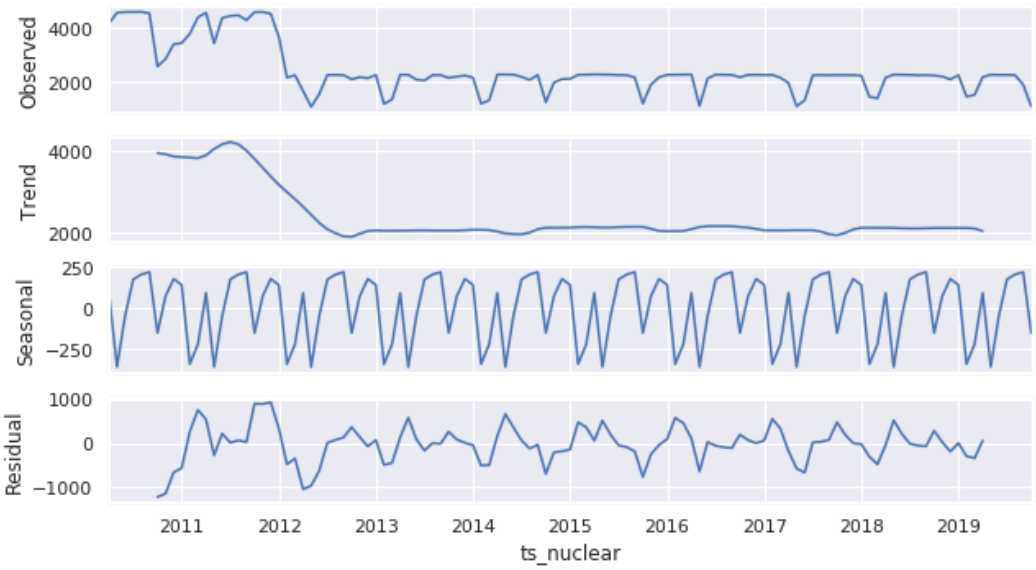

Observed
4000
2000
Trend
3000
2000
Seasonal
1000
0
−1000
Residual
500
0
−500
2011
2012
2013
2014
2015
2016
2017
2018
2019
ts_wind_total

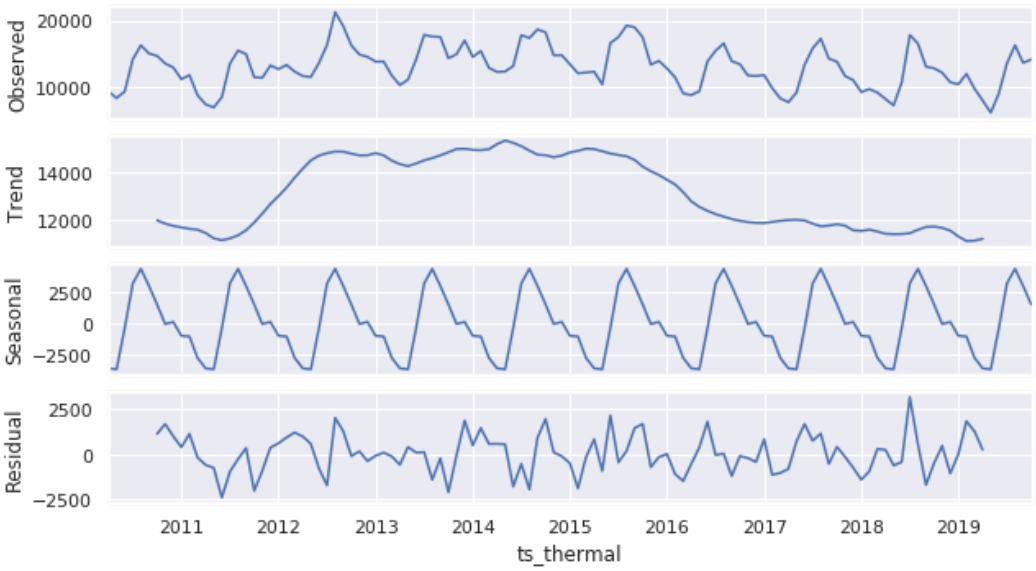

Observed
10000
0
Trend
5000
0
Seasonal
500
0
−500
Residual
1000
0
−1000
2011
2012
2013
2014
2015
2016
2017
2018
2019
ts_solar_pv

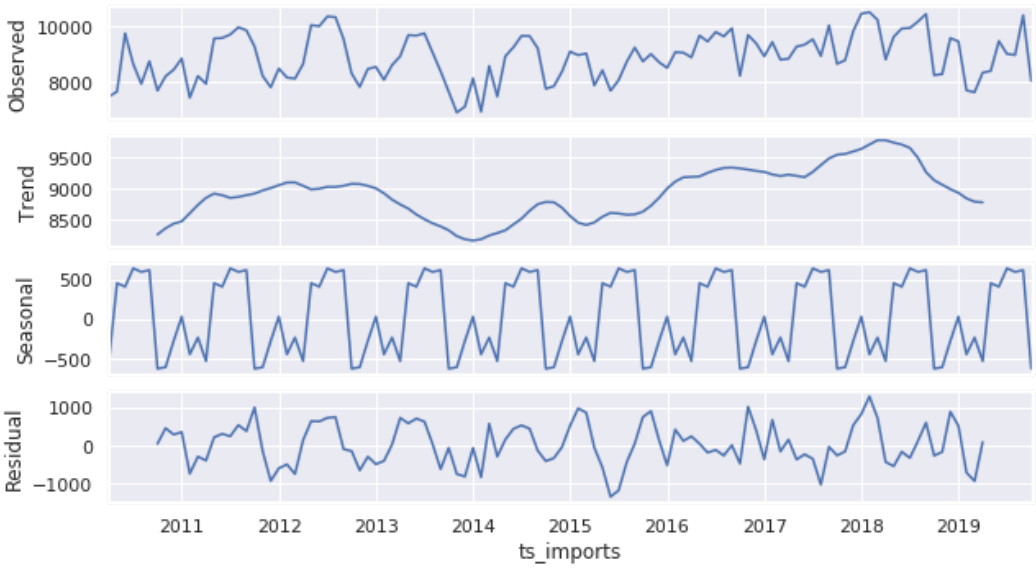

Observed
500
0
Trend
250
0
Seasonal
100
0
−100
Residual
100
0
−100
2011
2012
2013
2014
2015
2016
2017
2018
2019
ts_solar_thermal

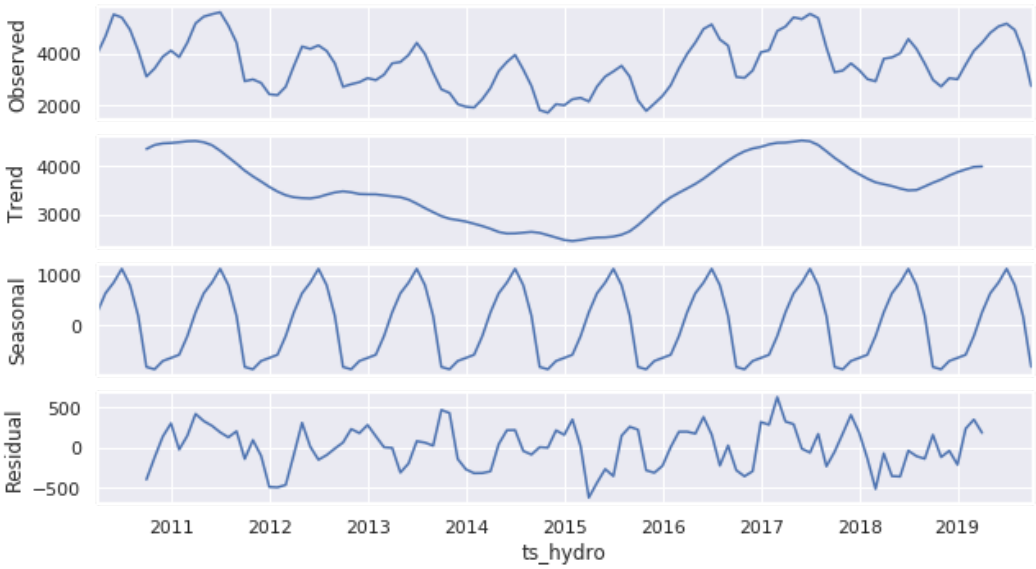

Observed
1000
0
Trend
500
0
Seasonal
0
−100
Residual
250
0
−250
2011
2012
2013
2014
2015
2016
2017
2018
2019
ts_solar


Observed
15000
10000
5000
Trend
10000
5000
Seasonal
1000
0
−1000
Residual
1000
0
−1000
2011
2012
2013
2014
2015
2016
2017
2018
2019
ts_renewables


Observed
4000
2000
Trend
4000
2000
Seasonal
250
0
−250
Residual
1000
0
−1000
2011
2012
2013
2014
2015
2016
2017
2018
2019
ts_nuclear


Observed
20000
10000
Trend
14000
12000
Seasonal
2500
0
−2500
Residual
2500
0
−2500
2011
2012
2013
2014
2015
2016
2017
2018
2019
ts_thermal


Observed
10000
8000
Trend
9500
9000
8500
Seasonal
500
0
−500
Residual
1000
0
−1000
2011
2012
2013
2014
2015
2016
2017
2018
2019
ts_imports



In [26]:
```
# peak hour
```

In [27]:
```
# daily peak
```

In [28]:
```
# daily total
```

Links

- Daily Renewables Watch (http://content.caiso.com/green/renewrpt/DailyRenewablesWatch.pdf)
- CAISO Interface Specification (http://www.caiso.com/Documents/OASIS-InterfaceSpecification_v5_1_8Clean_Independent2019Release.pdf#search=Interface%20Specification)
- Wind Solar RTD & Curtailment (http://www.caiso.com/Documents/Wind_SolarReal-TimeDispatchCurtailmentReportOct21_2019.pdf#search=Real%20Time%20Dispatch)
- Daily Renewables Watch (local) (./resources/docs/DailyRenewablesWatch.pdf)
- CAISO Interface Specification (local) (./resources/docs/OASIS-InterfaceSpecification_v5_1_8Clean_Independent2019Release.pdf)
- Wind Solar RTD & Curtailment (local) (./resources/docs/Wind_SolarReal-TimeDispatchCurtailmentReportOct21_2019.pdf)

In [ ]:
```

```

normal