# **Importing packages**

In [1]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import matplotlib as mat

import statsmodels.api as sm

from fbprophet import Prophet

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

from statsmodels.tsa.stattools import adfuller

In [2]:

pd.plotting.register\_matplotlib\_converters()

In [3]:

mat.rcParams.update({'figure.figsize':(20,15), 'font.size': 14})

# **Reading the data**

In [4]:

energy\_consumption = pd.read\_csv('../input/hourly-energy-consumption/PJME\_hourly.csv')

# **Preprocessing**

In [5]:

energy\_consumption.head()

Out[5]:

|  | Datetime | PJME\_MW |
| --- | --- | --- |
| 0 | 2002-12-31 01:00:00 | 26498.0 |
| 1 | 2002-12-31 02:00:00 | 25147.0 |
| 2 | 2002-12-31 03:00:00 | 24574.0 |
| 3 | 2002-12-31 04:00:00 | 24393.0 |
| 4 | 2002-12-31 05:00:00 | 24860.0 |

In [6]:

energy\_consumption.dtypes

Out[6]:

Datetime object

PJME\_MW float64

dtype: object

In [7]:

energy\_consumption.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 145366 entries, 0 to 145365

Data columns (total 2 columns):

Datetime 145366 non-null object

PJME\_MW 145366 non-null float64

dtypes: float64(1), object(1)

memory usage: 2.2+ MB

In [8]:

energy\_consumption['Datetime'] = pd.to\_datetime(energy\_consumption['Datetime'])

In [9]:

energy\_consumption = energy\_consumption.set\_index('Datetime').resample('H').sum()

In [10]:

energy\_consumption.info()

<class 'pandas.core.frame.DataFrame'>

DatetimeIndex: 145392 entries, 2002-01-01 01:00:00 to 2018-08-03 00:00:00

Freq: H

Data columns (total 1 columns):

PJME\_MW 145392 non-null float64

dtypes: float64(1)

memory usage: 2.2 MB

In [11]:

hours\_no\_consumption = energy\_consumption.loc[energy\_consumption['PJME\_MW'] == 0].copy()

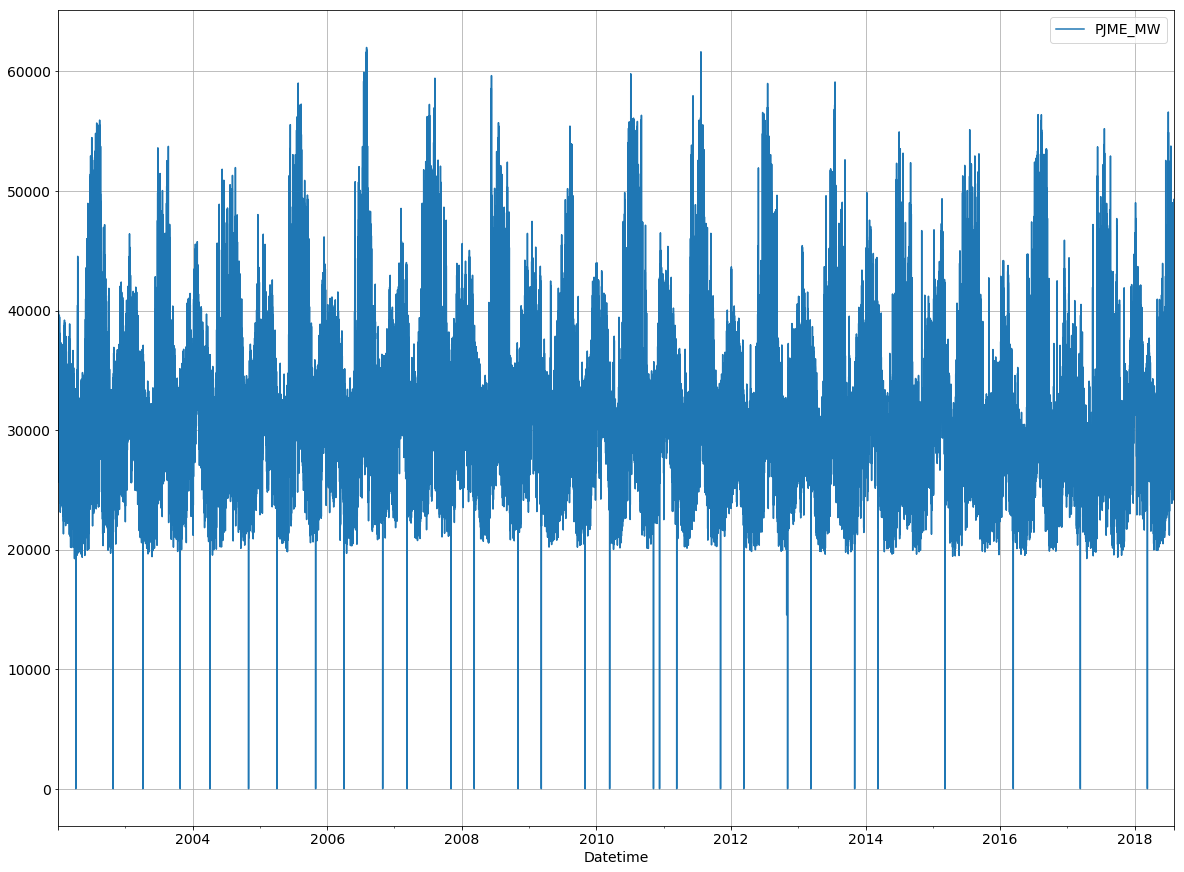
# **Exploratory Analysis**

In [12]:

energy\_consumption.plot(grid=True)

*# plt.yscale('log')*

plt.show()



In [13]:

energy\_consumption.describe()

Out[13]:

|  | PJME\_MW |
| --- | --- |
| count | 145392.000000 |
| mean | 32074.486024 |
| std | 6479.660890 |
| min | 0.000000 |
| 25% | 27571.000000 |
| 50% | 31420.000000 |
| 75% | 35648.250000 |
| max | 62009.000000 |

* Total energy consumption falls between (0 - 62009) MW
* Averge energy consumption per hour is 32074.5 MW with std of 6479.7

In [14]:

energy\_consumption.loc[~energy\_consumption.index.isin(hours\_no\_consumption.index)].describe()

Out[14]:

|  | PJME\_MW |
| --- | --- |
| count | 145362.000000 |
| mean | 32081.105598 |
| std | 6463.923399 |
| min | 14544.000000 |
| 25% | 27574.000000 |
| 50% | 31421.000000 |
| 75% | 35650.750000 |
| max | 62009.000000 |

Removing the hours with no energy consumption data:

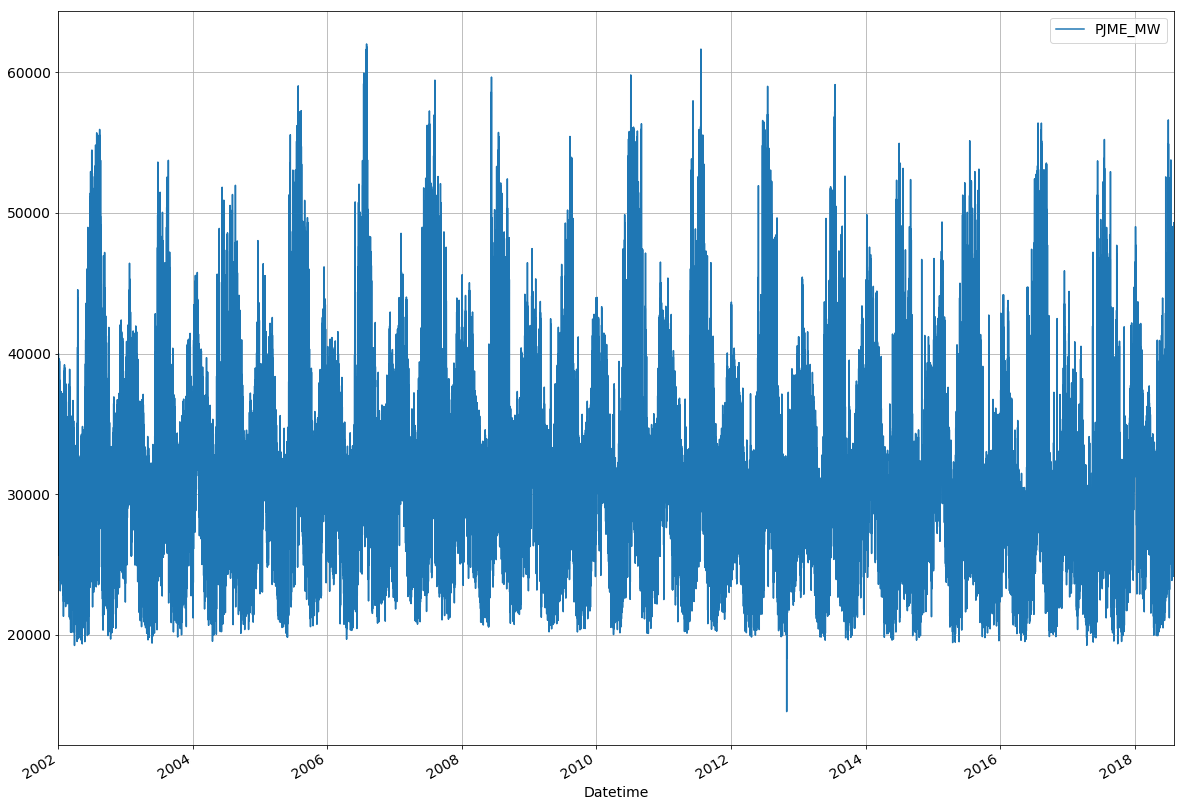
* The average energy consumption didn't change alot.
* Minimum recorded energy consumtion is 14544 MW.

In [15]:

energy\_consumption.loc[~energy\_consumption.index.isin(hours\_no\_consumption.index)].plot(grid=True)

*# plt.yscale('log')*

plt.show()



95% of the energy consumption falls between:

In [16]:

energy\_consumption['PJME\_MW'].quantile([0.025]).values[0]

Out[16]:

21678.0

In [17]:

energy\_consumption['PJME\_MW'].quantile([0.975]).values[0]

Out[17]:

47523.225000000006

In [18]:

energy\_consumption.loc[(energy\_consumption['PJME\_MW'] >= energy\_consumption['PJME\_MW'].quantile([0.025]).values[0])

&

(energy\_consumption['PJME\_MW'] <= energy\_consumption['PJME\_MW'].quantile([0.975]).values[0])].describe()

Out[18]:

|  | PJME\_MW |
| --- | --- |
| count | 138123.000000 |
| mean | 31872.380813 |
| std | 5575.156271 |
| min | 21678.000000 |
| 25% | 27801.500000 |
| 50% | 31420.000000 |
| 75% | 35363.000000 |
| max | 47523.000000 |

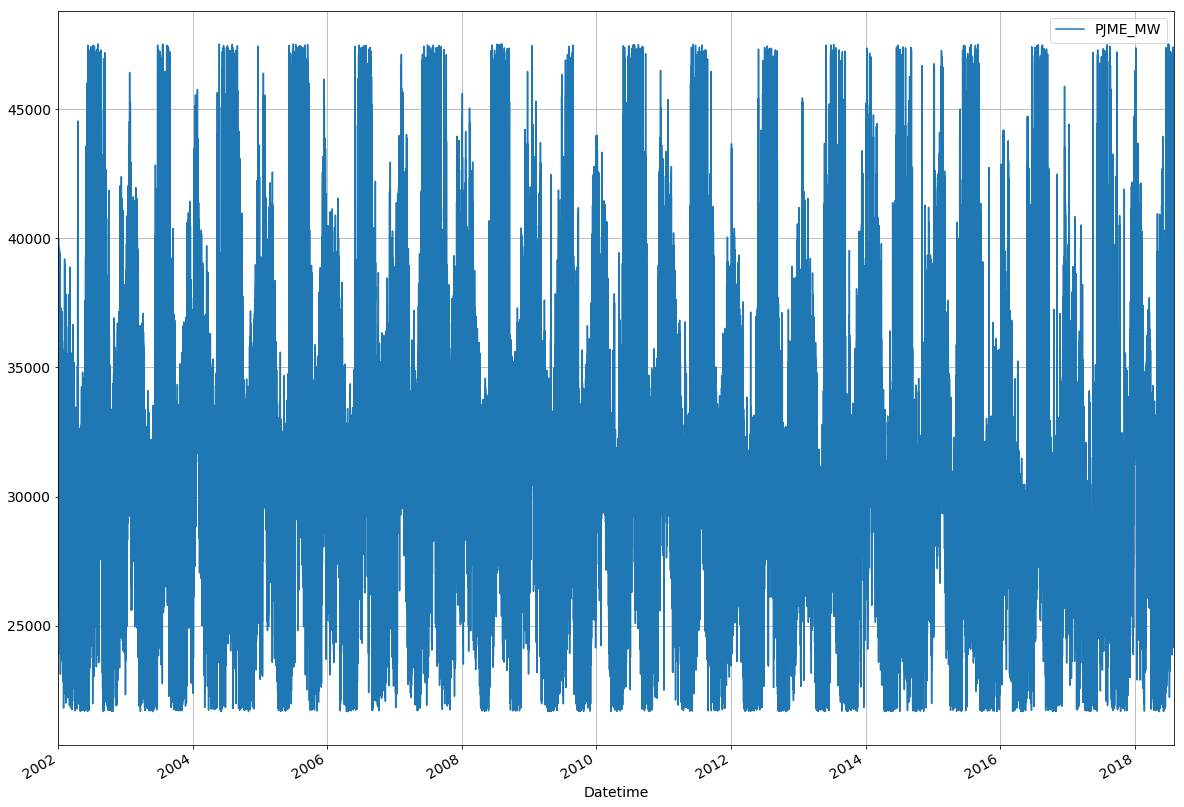
In [19]:

energy\_consumption.loc[(energy\_consumption['PJME\_MW'] >= energy\_consumption['PJME\_MW'].quantile([0.025]).values[0])

&

(energy\_consumption['PJME\_MW'] <= energy\_consumption['PJME\_MW'].quantile([0.975]).values[0])].plot(grid=True)

plt.show()



In [20]:

energy\_consumption.resample('YS').mean().sort\_values('PJME\_MW',ascending=False).head(1)

Out[20]:

|  | PJME\_MW |
| --- | --- |
| Datetime |  |
| 2007-01-01 | 33605.794292 |

In [21]:

energy\_consumption.resample('YS').mean().sort\_values('PJME\_MW',ascending=False).tail(1)

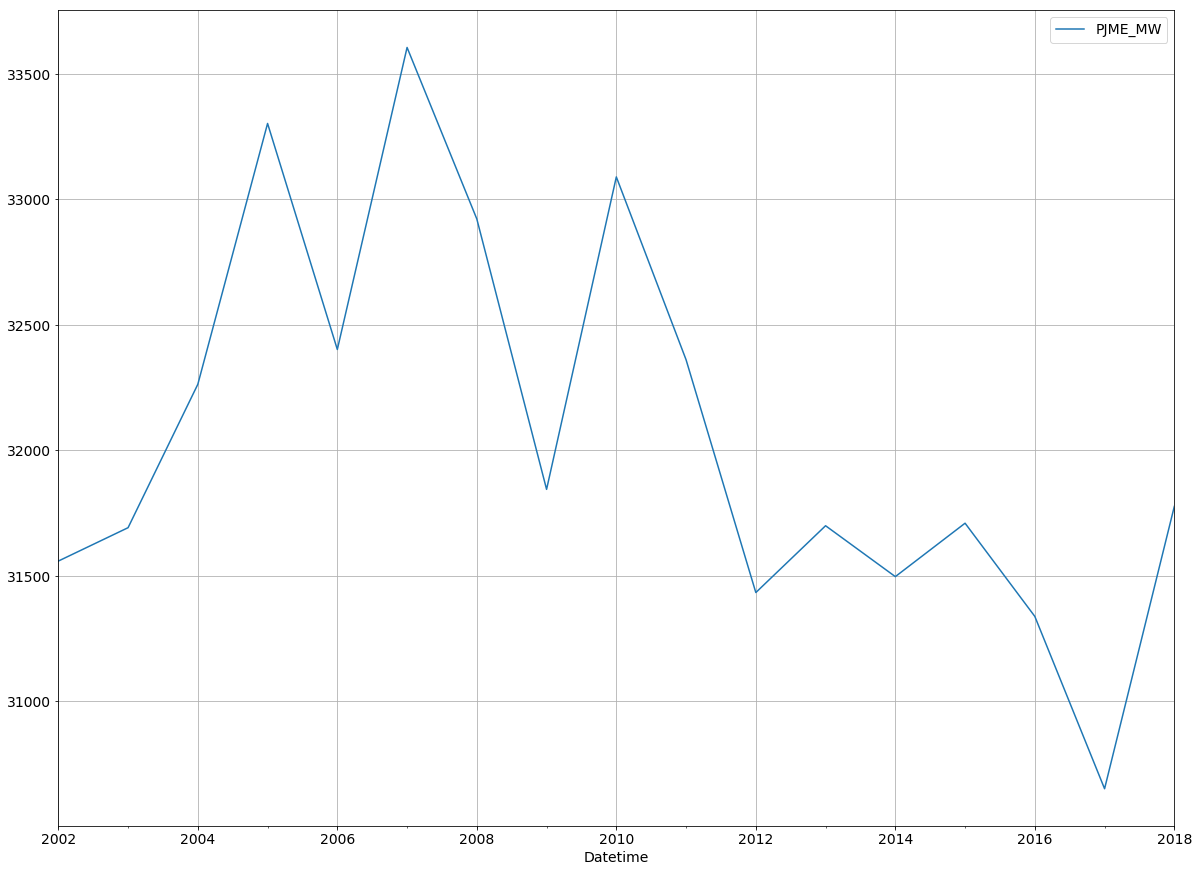
Out[21]:

|  | PJME\_MW |
| --- | --- |
| Datetime |  |
| 2017-01-01 | 30650.911644 |

In [22]:

energy\_consumption.resample('YS')[['PJME\_MW']].mean().plot(grid=True)

plt.show()

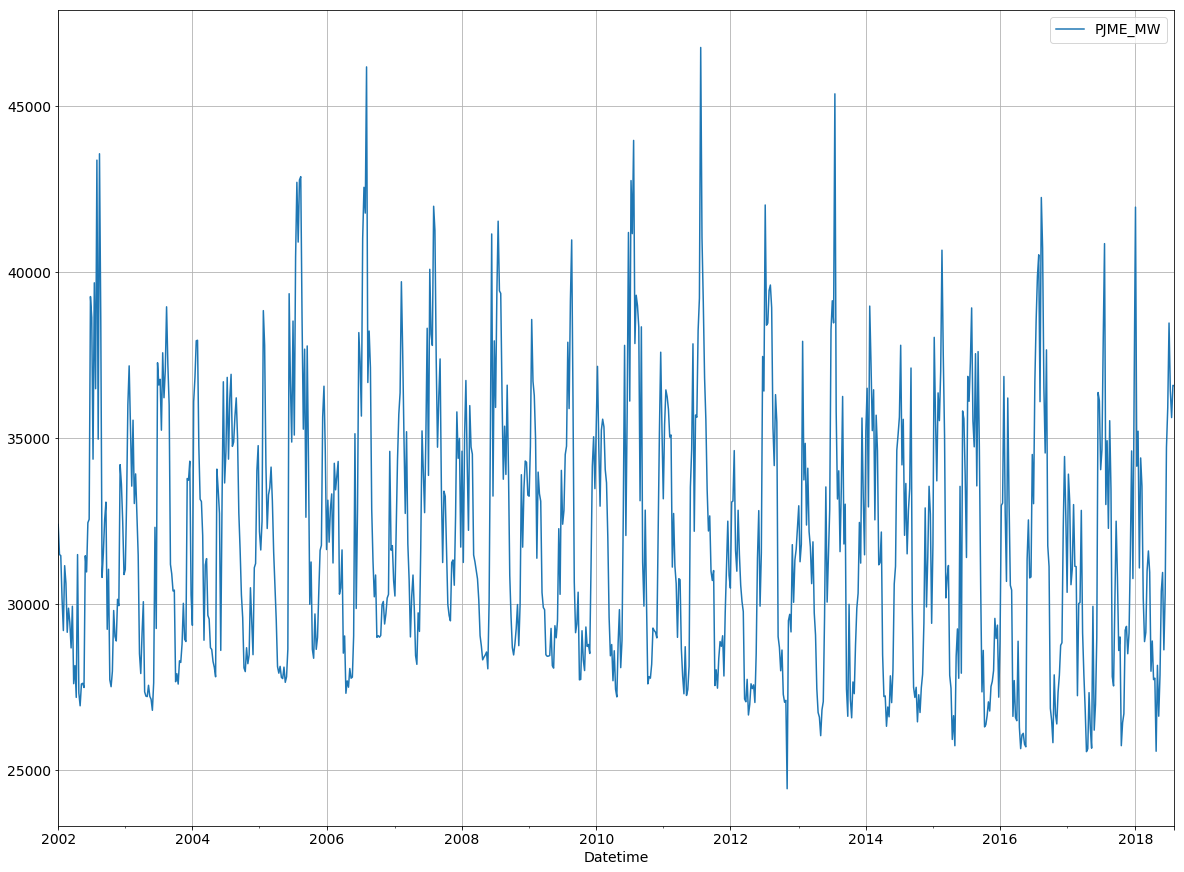


* 2007 Had the highest average energy consumption per hour, 2005 follows
* 2017 has the lowest, interesting!

In [23]:

energy\_consumption.resample('W').mean().plot(grid=True)

plt.show()



In [24]:

energy\_consumption.loc['01-01-2018':].resample('W').mean().sort\_values('PJME\_MW',ascending=False).head(1)

Out[24]:

|  | PJME\_MW |
| --- | --- |
| Datetime |  |
| 2018-01-07 | 41951.339286 |

* first week in Jan 2018 had the highest energy consumption

In [25]:

energy\_consumption['Hour'] = energy\_consumption.index.hour

In [26]:

max\_hour = energy\_consumption.loc[energy\_consumption.loc['06-01-2018':'07-31-2018'].resample('D')['PJME\_MW'].idxmax()].copy()

* The highest energy consumption in 2018 was in 2018-07-03 "Tuesday"

In [27]:

df = energy\_consumption.loc['06-01-2018':'07-31-2018'].resample('D')[['PJME\_MW']].sum()

\_ = plt.plot(df.index, df.PJME\_MW)

for H **in** max\_hour['Hour'].unique():

df = energy\_consumption.loc['06-01-2018':'07-31-2018'].resample('D')[['PJME\_MW']].sum()

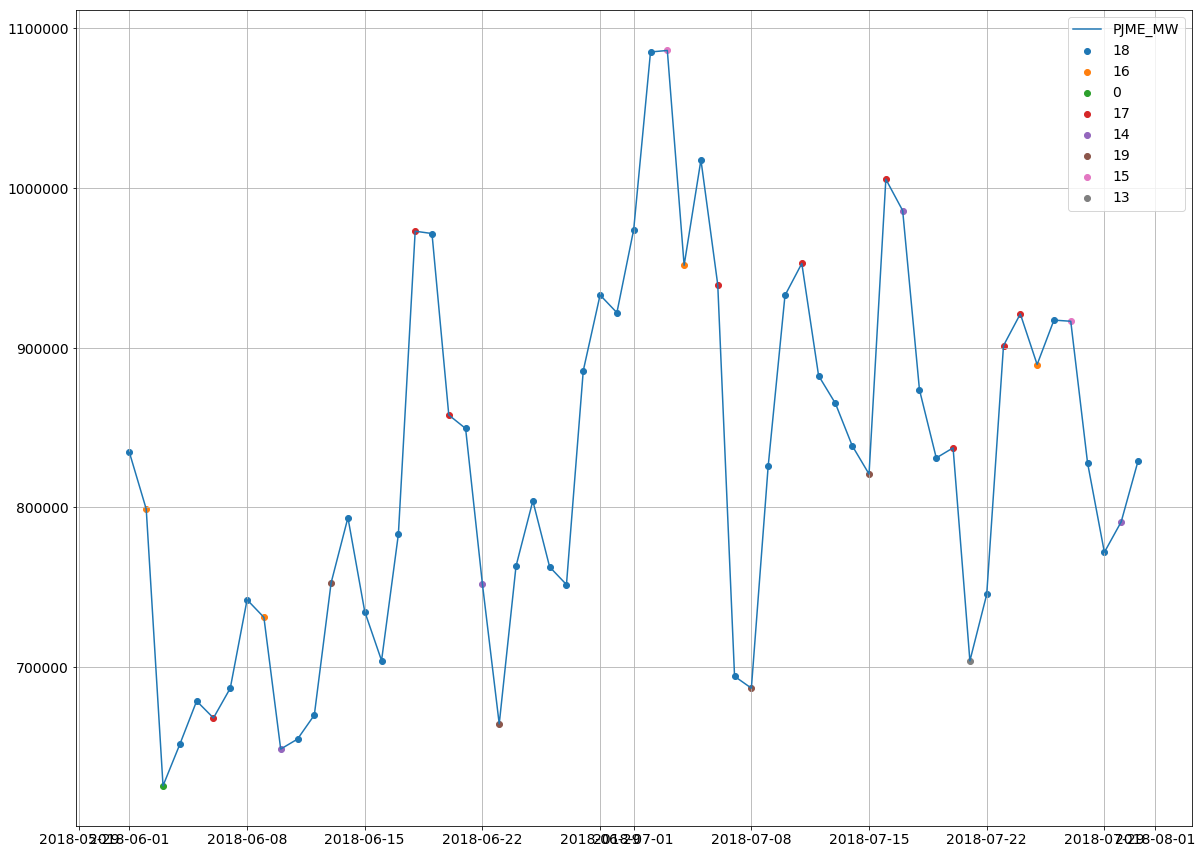
df = df.loc[max\_hour.loc[max\_hour['Hour'] == H].index.date].copy()

\_ = plt.scatter( x = df.index, y = df.PJME\_MW, label = H)

plt.grid()

plt.legend()

plt.show()



* 6 PM is the hour with the highest consumtion in most of the days for the selected period

In [28]:

dayofweek = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']

energy\_consumption['Day of Week'] = energy\_consumption.index.dayofweek

energy\_consumption['Day of Week'] = energy\_consumption['Day of Week'].apply(lambda x: dayofweek[x])

In [29]:

energy\_consumption.loc['01-01-2018':].groupby([energy\_consumption.loc['01-01-2018':].index.date,

'Day of Week'])[['PJME\_MW']].sum().sort\_values('PJME\_MW',

ascending=False).head()

Out[29]:

|  |  | PJME\_MW |
| --- | --- | --- |
|  | Day of Week |  |
| 2018-07-03 | Tuesday | 1086193.0 |
| 2018-07-02 | Monday | 1085235.0 |
| 2018-01-05 | Friday | 1060747.0 |
| 2018-01-06 | Saturday | 1045578.0 |
| 2018-07-05 | Thursday | 1017657.0 |

* The highest energy consumption in 2018 was in 2018-07-03 "Tuesday"

In [30]:

df = energy\_consumption.loc['06-01-2018':'07-31-2018'].resample('D')[['PJME\_MW']].sum()

\_ = plt.plot(df.index, df.PJME\_MW)

for D **in** energy\_consumption['Day of Week'].unique():

df = energy\_consumption.loc['06-01-2018':'07-31-2018'].loc[energy\_consumption['Day of Week'] == D].copy()

df = df.resample('D')[['PJME\_MW']].sum().copy()

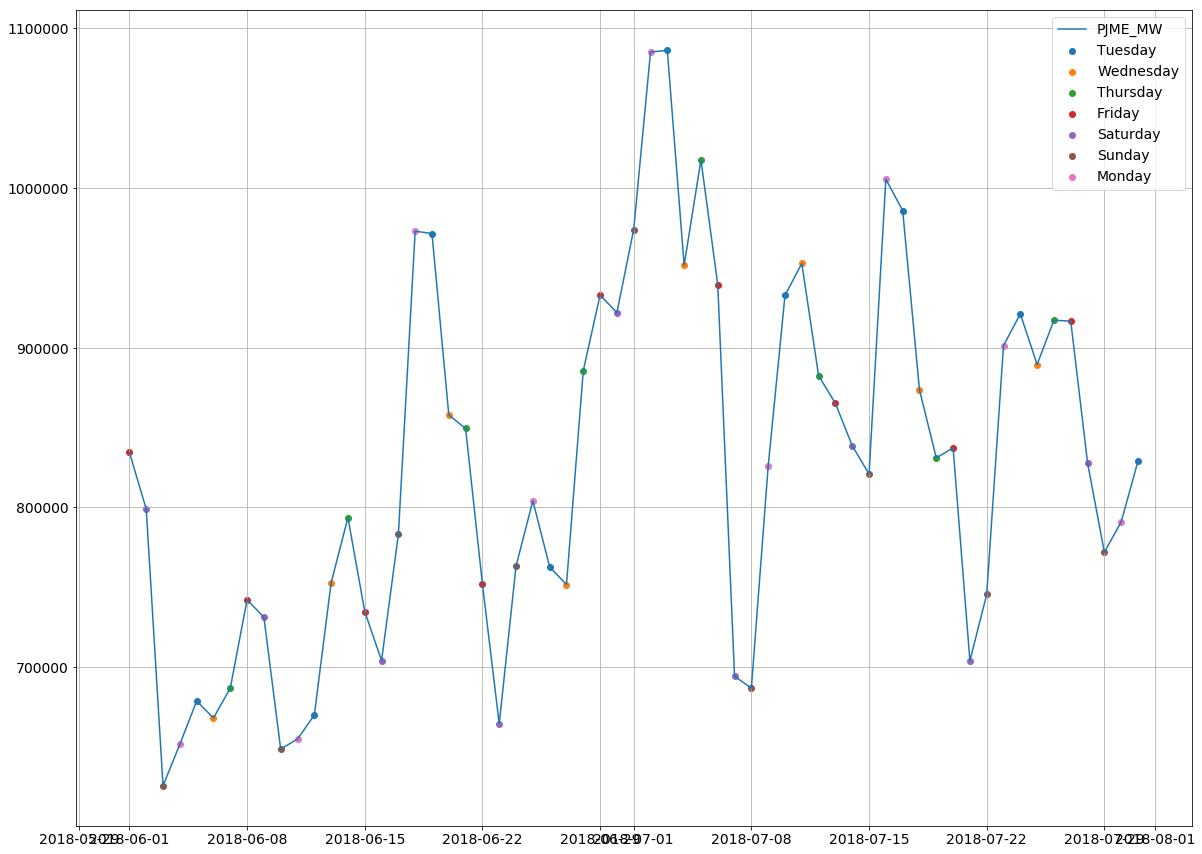
df = df.loc[df['PJME\_MW'] != 0].copy()

\_ = plt.scatter( x = df.index, y = df.PJME\_MW, label = D)

plt.grid()

plt.legend()

plt.show()



* peaks dominated by Mondays and Tuesdays
* whereas, troughs are dominated by Saturdays and Sundays

In [31]:

energy\_consumption.loc['01-01-2017':'31-12-2017'].resample('Q').sum().sort\_values('PJME\_MW',ascending=False)

Out[31]:

|  | PJME\_MW | Hour |
| --- | --- | --- |
| Datetime |  |  |
| 2017-09-30 | 73627696.0 | 25392 |
| 2017-03-31 | 66768754.0 | 24840 |
| 2017-12-31 | 65425156.0 | 25392 |
| 2017-06-30 | 62680380.0 | 25116 |

* The highest energy consumption Qurter in 2017 was in third Qurter (Summer).

In [32]:

year\_quarter\_con = energy\_consumption.copy()

year\_quarter\_con['Year'] = year\_quarter\_con.index.year

year\_quarter\_con['Quarter'] = year\_quarter\_con.index.quarter

In [33]:

year\_quarter\_con = year\_quarter\_con.groupby(['Year','Quarter'])[['PJME\_MW']].sum()

year\_quarter\_con.iloc[year\_quarter\_con.reset\_index().groupby(['Year'])['PJME\_MW'].idxmax()]

Out[33]:

|  |  | PJME\_MW |
| --- | --- | --- |
| Year | Quarter |  |
| 2002 | 3 | 79108260.0 |
| 2003 | 3 | 76773824.0 |
| 2004 | 3 | 76182651.0 |
| 2005 | 3 | 83771227.0 |
| 2006 | 3 | 80581425.0 |
| 2007 | 3 | 80811919.0 |
| 2008 | 3 | 78821422.0 |
| 2009 | 3 | 74989501.0 |
| 2010 | 3 | 81481624.0 |
| 2011 | 3 | 79785091.0 |
| 2012 | 3 | 78837294.0 |
| 2013 | 3 | 75607834.0 |
| 2014 | 1 | 74726381.0 |
| 2015 | 3 | 76681092.0 |
| 2016 | 3 | 80488648.0 |
| 2017 | 3 | 73627696.0 |
| 2018 | 1 | 69982662.0 |

In [34]:

energy\_consumption['Quarter'] = energy\_consumption.index.quarter

energy\_consumption['Month'] = energy\_consumption.index.month

In [35]:

df = energy\_consumption.loc[:'31-07-2018'].resample('MS')[['PJME\_MW']].sum()

\_ = plt.plot( df.index, df.PJME\_MW)

for Q **in** energy\_consumption['Quarter'].unique():

df = energy\_consumption.loc[:'31-07-2018'].loc[energy\_consumption['Quarter'] == Q].resample('MS')[['PJME\_MW']].sum().copy()

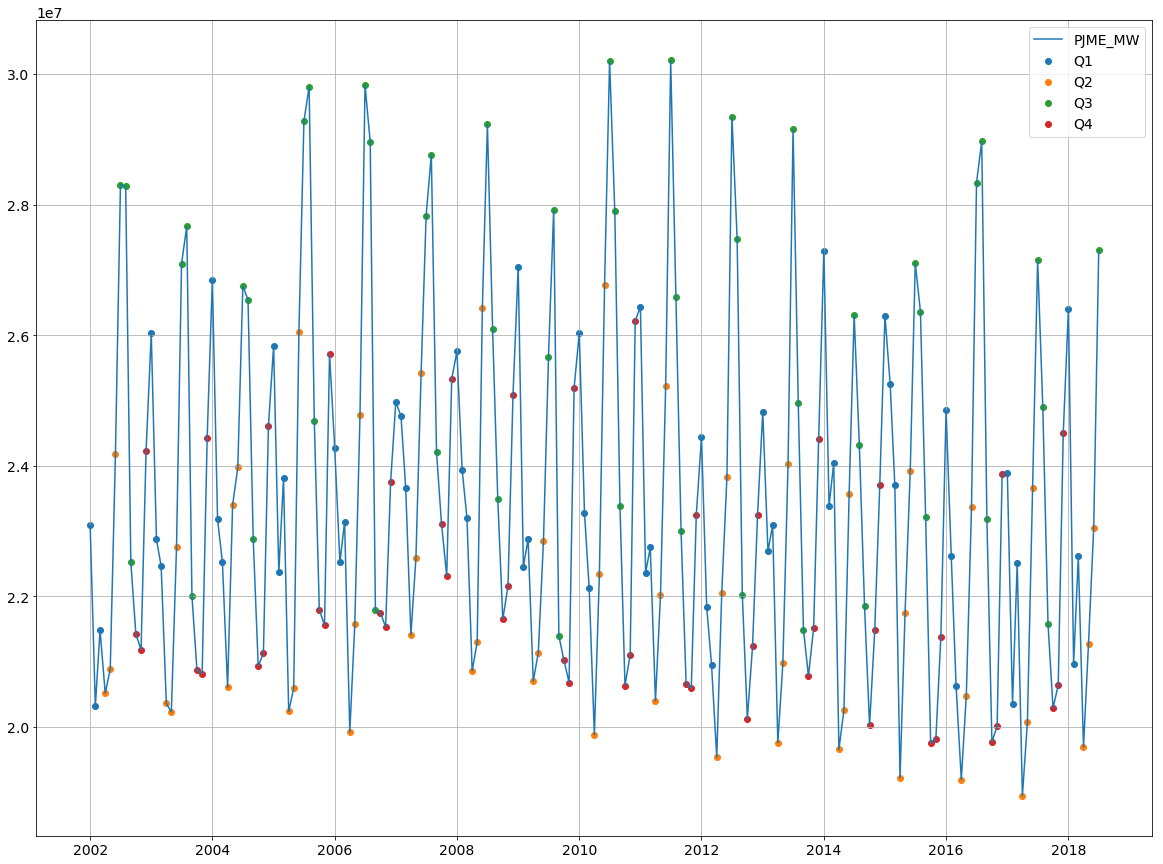
df = df.loc[df['PJME\_MW'] != 0].copy()

\_ = plt.scatter( x = df.index, y = df.PJME\_MW, label = 'Q' +str(Q))

plt.grid()

plt.legend()

plt.show()



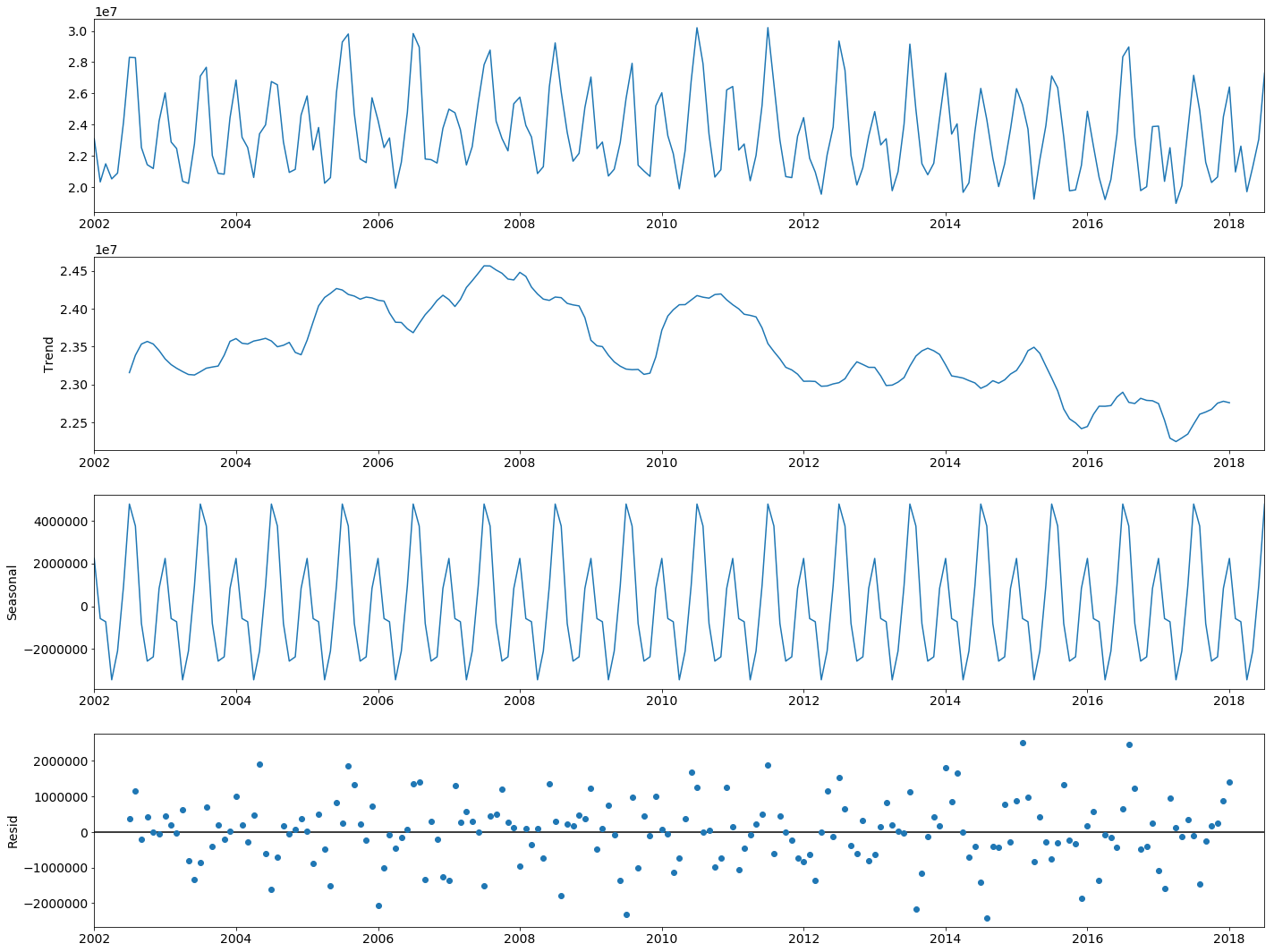
* okay so summer had always been the season with the highest consumption through the years.
* 2nd and 4th quarters had the lowest consumption

In [36]:

decompose = sm.tsa.seasonal\_decompose(energy\_consumption.loc[:'31-07-2018'].resample('MS')[['PJME\_MW']].sum())

decompose.plot()

plt.show()



In [37]:

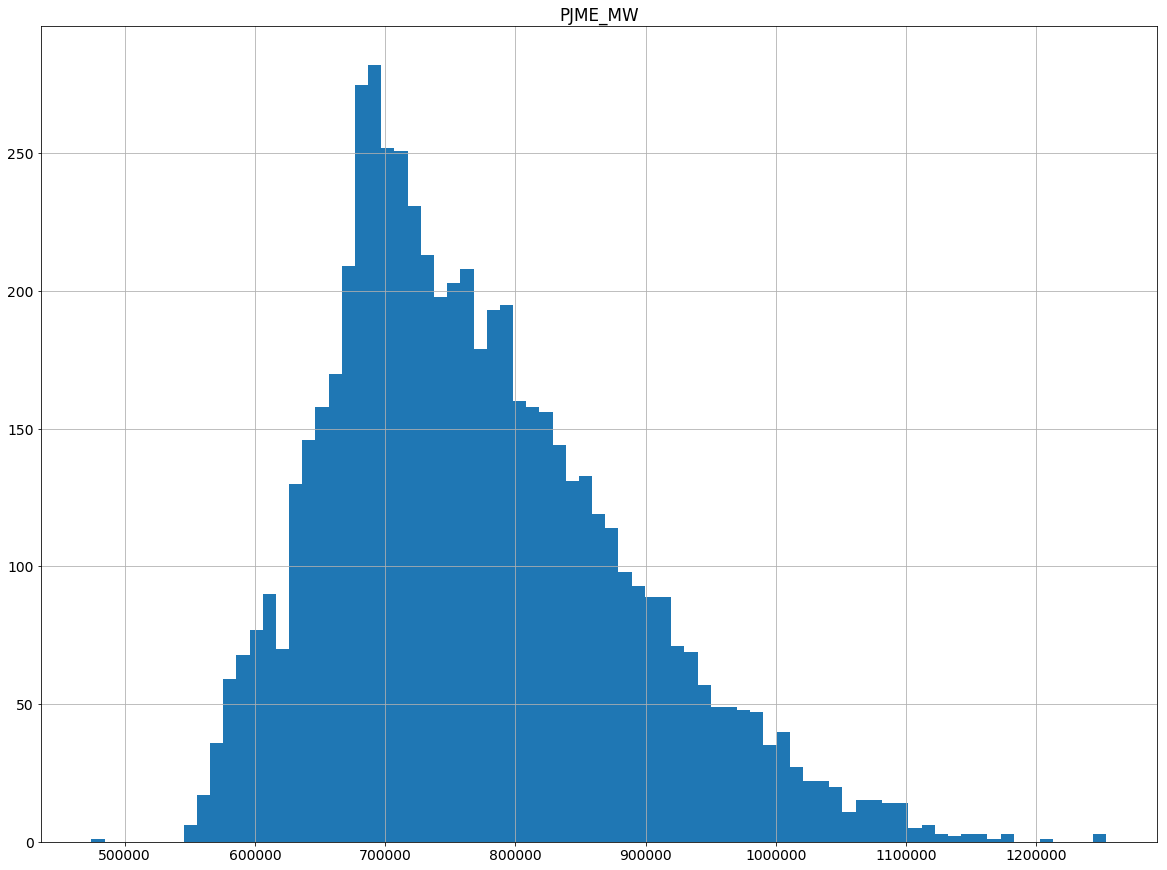
day\_consum = energy\_consumption.loc[:'31-07-2018'].resample('D')[['PJME\_MW']].sum()

day\_consum.hist(bins=int(np.sqrt(len(day\_consum))))

Out[37]:

array([[<matplotlib.axes.\_subplots.AxesSubplot object at 0x7f9d724e3a90>]],

dtype=object)



In [38]:

day\_consum.reset\_index(inplace=True)

In [39]:

day\_consum['Datetime'] = pd.to\_datetime(day\_consum['Datetime'])

In [40]:

day\_consum.index = pd.DatetimeIndex(day\_consum['Datetime'],freq='D')

In [41]:

day\_consum.drop(['Datetime'],1,inplace=True)

# **Stationarity Test**

In [42]:

result = adfuller(day\_consum['PJME\_MW'].values)

print('ADF Statistic: **%f**' % result[0])

print('p-value: **%f**' % result[1])

print('Critical Values:')

for key, value **in** result[4].items():

print('**\t%s**: **%.3f**' % (key, value))

if result[0] <= value:

print('Stationary at ' + key)

else:

print('Non-Stationary at ' + key)

ADF Statistic: -8.276309

p-value: 0.000000

Critical Values:

1%: -3.431

Stationary at 1%

5%: -2.862

Stationary at 5%

10%: -2.567

Stationary at 10%

In [43]:

day\_consum['ds'] = day\_consum.index

day\_consum.rename(columns={'PJME\_MW':'y'},inplace=True)

# **Train Test Data**

In [44]:

split\_date = '06-30-2018'

train = day\_consum.loc[:split\_date].copy()

test = day\_consum.loc[split\_date:].copy()

# **Time Series Prediction**

In [45]:

model = Prophet()

In [46]:

model.fit(train)

Out[46]:

<fbprophet.forecaster.Prophet at 0x7f9d7248a748>

In [47]:

future = model.make\_future\_dataframe(periods=len(test))

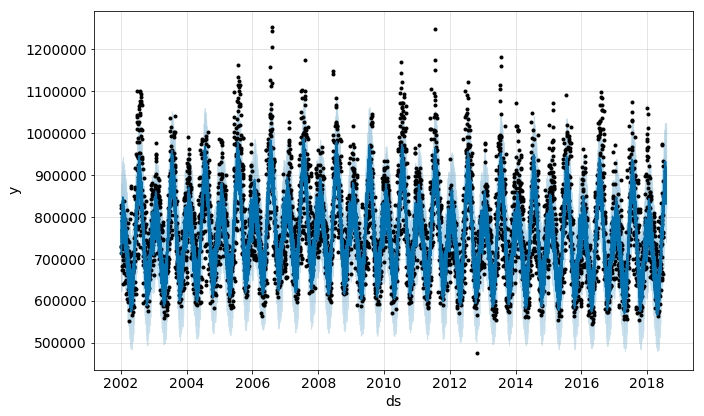
In [48]:

forecast = model.predict(future)

In [49]:

model.plot(forecast)

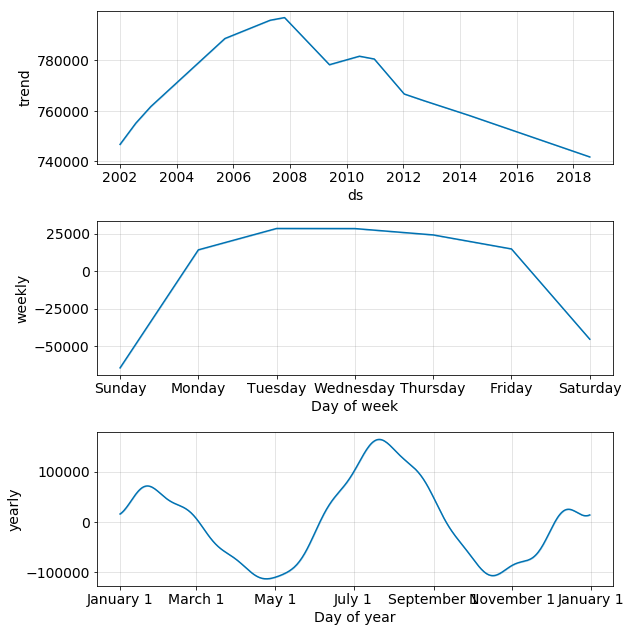
plt.show()



In [50]:

model.plot\_components(forecast)

plt.show()



In [51]:

prediction\_vs\_real = forecast.set\_index('ds')[['yhat', 'yhat\_lower', 'yhat\_upper']].join(day\_consum.set\_index('ds'))

In [52]:

def calculate\_forecast\_errors(df, prediction\_size):

*"""Calculate MAPE and MAE of the forecast.*

*Args:*

*df: joined dataset with 'y' and 'yhat' columns.*

*prediction\_size: number of days at the end to predict.*

*"""*

*# Make a copy*

df = df.copy()

*# Now we calculate the values of e\_i and p\_i according to the formulas given in the article above.*

df['e'] = df['y'] - df['yhat']

df['p'] = 100 \* df['e'] / df['y']

*# Recall that we held out the values of the last `prediction\_size` days*

*# in order to predict them and measure the quality of the model.*

*# Now cut out the part of the data which we made our prediction for.*

predicted\_part = df[-prediction\_size:]

*# Define the function that averages absolute error values over the predicted part.*

error\_mean = lambda error\_name: np.mean(np.abs(predicted\_part[error\_name]))

*# Now we can calculate MAPE and MAE and return the resulting dictionary of errors.*

return {'MAPE': error\_mean('p'), 'MAE': error\_mean('e')}

In [53]:

calculate\_forecast\_errors(prediction\_vs\_real, len(test))

Out[53]:

{'MAPE': 8.783032537399167, 'MAE': 76745.33751399578}