HI3_handl_and_expl

November 7, 2017

1 Hand-in 3, Part 1: Data handling and exploration

In this first notebook you will show us how you handle data being separated over several files, as well as exploring the quality and properties of your data.

Section 1: bash scripting You have downloaded a zip file containing 5 CSV files, each containing part of the data you need. First, use your bash tools to look at the headers and size of the file. What do the different files contain?

Write a bash script that concatenates the 4 data files (except the flow_criticality_data.csv file). Exlain in the markdown cell below, what each part of your script does.

Q#1 *Explain your script here (by double clicking on this text).*

- Define that we are using bash

```
#! /bin/bash
```

- Quick explanation of what the bashscript does

```
# Joining the files
#
# Usage "./collect_data.sh
```

- Make temp1.csv containing half the data -t, means the file is comma seperated, -a1 and -a2 means that it will also print any unpairable line -oauto means that it will automatically format the data

```
join -t, -a1 -a2 -oauto energy_demand_data.csv exchange_data.csv > temp1.csv
```

- Make temp2.csv containing the other half of data

```
join -t, -a1 -a2 -oauto renewable_production_data.csv generator_production_data.csv > temp
```

- Combine the temp files, to one whole file

```
join -t, -a1 -a2 -oauto temp1.csv temp2.csv > joined_data.csv
```

- Clean up by removing the temp files

```
rm temp1.csv
rm temp2.csv
```

1.1 Section 2: Visualizing the data

Here you will plot the resulting data file from the previous section, and plot it in order to identify missing data and see if you can already draw some conclusions on the data.

- *Hint: remember the hint given in Exc.13.3, on how to find out if your data contains NaN values*

```
In [23]: # Importing the packages we need
         import pandas as pd
         import matplotlib.pyplot as plt
         import seaborn as sns
         %matplotlib notebook
         import numpy as np
         import scipy

In [24]: # Import of the data created with the bash script
         data  = pd.read_csv('joined_data.csv')

In [25]: # Simple plots showing a small pick of our data

         data.plot(x='time',y='prod_gen_1',title='prod_gen_1')
         data.plot(x='time',y='load_node_130',title='load_node_130')
         data.plot(x='time',y='renew_node_24',title='renew_node_24')
```

```
<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


/home/student/.local/lib/python3.5/site-packages/matplotlib/__init__.py:830: MatplotlibDeprecat
  mplDeprecation)


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x7f314fe04908>
```

would it be practical to plot all possible scatter plots (scatter matrix)?

Not really, unless you use a cluster to compute the data, as there are way too many datapoints and columns

Q#2 For this data, what is the reasonable approach to dealing with the NaN values? Why?

As there is no directly observable pattern, interpolation does not seem like a good aproach. Therefor we will be dropping the NaN's.

In [26]:
```
# Removal of NaN's


print("Total number of NaN's before removal: " +str(data.isnull().sum().sum()))
# Drop any row containing any NaN's
data_nona = data.dropna(how='any')
print("Total number of NaN's after removal: " +str(data_nona.isnull().sum().sum()))
```

```
Total number of NaN's before removal: 1711
Total number of NaN's after removal: 0
```

1.1.1 Feature reduction

Since you must reduce the amount of sensors, you need to find out which ones you can get rid of.

Q#3 Why would PCA be useful for this?

It shows you which components are principal for your data. Thereby showing which sensors are impacting the overall data more

With that in mind, you can narrow down the amount of sensors, used to generate data. Obviously you will loose some data, but choosing the correct amount of components, you can get pretty close to the full data

In [27]:
```
from sklearn.decomposition import PCA

# Removal of the time column, as this is not usable in math
datanotime = data_nona.drop('time',1)


# Removing any column where more than 90% of the data is zero
data_notime2 = datanotime.loc[:, (datanotime != 0).sum()>len(datanotime)*.1]

# First we standardize the data
data_stand = (data_notime2 - data_notime2.mean()) / data_notime2.std()

# The standardization created a lot of NaN's
# These came from dividing by zero, as a lot of the prod_gen sensors had only zero's
data_stand = data_stand.dropna(axis='columns',how='all')



print("NaN's in data set: " + str(data_stand.isnull().sum().sum()))
```

```
        # Then instantiate the data, and make the fit
        comp=data_stand.shape[1]
        pca = PCA(n_components=comp)
        pca_data=pca.fit(data_stand)

        # Plotting the explained variance
        fig = plt.figure(figsize=(8,8))
        sing_vals = np.arange(comp) + 1
        plt.plot(sing_vals, pca.explained_variance_ratio_, 'o-', linewidth=2)
        plt.title('Scree Plot')
        plt.xlabel('Principal Component')
        plt.ylabel('Cumulative explained variance')
        plt.plot(sing_vals, pca.explained_variance_ratio_.cumsum(), 'o-', linewidth=2)

        plt.legend(['Individual','Cumulative'], loc='best', borderpad=0.3,
                   shadow=False,
                   markerscale=0.4)

        # Saving the pca components
        data_components = pca.components_

        #with pd.option_context('display.max_rows', None, 'display.max_columns', 3):
        print((datanotime != 0).sum())
```

```
NaN's in data set: 0
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
load_node_130    8311
load_node_133    8311
load_node_134    8311
load_node_143    8311
load_node_16     8311
load_node_145    8311
load_node_21     8311
load_node_153    8311
load_node_154    8311
load_node_156    8311
load_node_159    8311
load_node_160    8311
load_node_162    8311
load_node_167    8311
load_node_43     8311
```

```
load_node_52     8311
load_node_181    8311
load_node_182    8311
load_node_184    8311
load_node_186    8311
load_node_60     8311
load_node_63     8311
load_node_64     8311
load_node_66     8311
load_node_68     8311
load_node_69     8311
load_node_197    8311
load_node_73     8311
load_node_77     8311
load_node_208    8311
                 ...
prod_gen_22      3227
prod_gen_23      3304
prod_gen_24         0
prod_gen_25      3328
prod_gen_26         0
prod_gen_27      3103
prod_gen_28         0
prod_gen_29      3035
prod_gen_30      2238
prod_gen_31      3091
prod_gen_32      3210
prod_gen_33      2344
prod_gen_34      3083
prod_gen_35       750
prod_gen_36         0
prod_gen_37      3022
prod_gen_38      3086
prod_gen_39      3018
prod_gen_40      2963
prod_gen_41      2776
prod_gen_42        75
prod_gen_43         0
prod_gen_44       151
prod_gen_45         0
prod_gen_46         0
prod_gen_47         0
prod_gen_48         0
prod_gen_49         0
prod_gen_50         2
prod_gen_51      4829
dtype: int64
```

```
In [7]: # Find the number of components needed to explain 90% of the variance
        for i in range(0,comp):
            if pca.explained_variance_ratio_.cumsum()[i] >= 0.9:
                print("with " +str(i+1) +" components you explain " + str(pca.explained_varianc
                new_comp=(i+1)
                break;
```

```
with 21 components you explain 90.2140550348%
```

1.1.2 Scree plot

Q#4 How many principal components do you need to explain 90 % of the variance
26 pricipal components would be needed to explain more than 90% of the variance
26 components will explain 90.35% of the variance

1.1.3 Clustering

You want to reduce the amount of field sensors to 20. You should now have from the previous question, an array with all your loading vectors (pca.components_), one vector per principal component, with 137 elements (one per each sensor). Use clustering to group sensors that behave the same.

Q#5 How would you choose which sensors in each cluster you should keep?

```
In [28]: from sklearn.cluster import KMeans

         # Slice the components we need. We take 26 as that gives us about 90% explained varia
         slice=data_components[0:new_comp]

         # Instanciate and fit the clusters
         kmeans = KMeans(n_clusters=20, n_init=200, max_iter=300, tol=0.0001, verbose=0, rando
         kmeans.fit(slice.T)
         cluster_pred = kmeans.predict(slice.T)

         # Prints the predicted clusters
         print(cluster_pred)
```

```
[ 0  0  0  0 19  0 19  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0 14 13 17  0  4  0 14 18 14 10  0  3 11  0
  3 10  0  5 14 15 13  7  0  8  6 18  8  7  6  7 16 18  6  5  6 18 18  2  2
  2  9 11  9 11 10 11 13 10  0  1 15 12 14 14  3 15  1  3 15  4  1  1  1  3
  4  4  3 12  1 15 15 15  3 14]
```

```
In [29]: import scipy

         # Make a pandas dataframe of our sensors, and the clusters they belong to
         clust = pd.DataFrame(data_stand.columns, columns=['sensors'])
         clust['cluster'] = cluster_pred
```

```
        # calculate the distance from a sensor to the cluster center
        temp = []
        for i in range(slice.T.shape[0]):
            A=slice.T[i]
            B=kmeans.cluster_centers_[clust.cluster[i]]
            temp.append(scipy.spatial.distance.euclidean(A, B))
        clust['dist_to_center'] = temp

        # Find the sensor in a cluster that is closest to the cluster centers
        x=0
        data_reduced = []
        sensor_cluster = []
        while x <= max(clust.cluster):
            temp_min = 100000
            temp_ind = 0
            for i in range(slice.shape[1]):
                if clust.cluster[i] == x:
                    if clust.dist_to_center[i] < temp_min:
                        temp_min = clust.dist_to_center[i]
                        temp_ind = i
            data_reduced.append(clust.sensors[temp_ind])
            x+=1

        # Make a new pandas dataframe witht the reduced sensor list
        data_reduced = pd.DataFrame(data_reduced, columns=['sensors'])

        # Sort the list of clusters by their distance to the center, for manual check up
        sorted_clust = clust.sort_values(by=['dist_to_center'])

        # For manual check
        #print(data_reduced.to_string(index=False))
        #with pd.option_context('display.max_rows', None, 'display.max_columns', 3):
        #    print(sorted_clust)
```

In [30]:
```
        x=0
        index = []
        for i in range(data_nona.shape[1]):
               for x in range(data_reduced.sensors.shape[0]):
                    if data_reduced.sensors.iloc[x] == data_nona.columns[i]:
                        #print(i)
                        #print(data_stand.columns[i])
                        index.append(i)
                        x+=1

        data_reduced = data_nona[index]
        #print(data_stand.columns[0])
        #data_reduced.sensors.iloc[2]
```

```
print(data_reduced)
```

```
      load_node_21  load_node_98  export_node_130  export_node_68  \
0          215.2788       28.1763       -13.551973      -42.135706
1          206.9927       26.2146       -11.235127      -39.463754
2          197.3522       24.4661       -11.064903      -36.113954
3          182.1743       23.3269        -8.057087      -33.309821
4          170.3825       22.8990        -6.710398      -33.277878
5          163.4111       22.8241       -11.968168      -33.750251
6          159.7063       22.9061       -14.185416      -33.848466
7          157.6746       22.5140        -7.571923      -32.970572
8          147.4763       22.7774        -6.455184      -32.062082
9          149.3885       24.1631        -4.704244      -28.566337
10         163.1721       25.8698        -8.997888      -24.867395
11         181.8954       26.7083        -9.447832      -12.307933
12         198.4277       27.2846        -9.721543       -5.917242
13         201.8139       27.2777       -11.967444       -0.099741
15         186.0385       25.5938       -22.109946       13.994299
16         187.3132       26.7856       -19.225204       18.059623
17         194.3245       30.7734       -21.608521       25.246866
18         227.8672       32.4311       -22.851104        1.513755
19         239.9777       33.1325       -21.395720      -10.393060
20         244.2801       33.7336       -21.962749      -12.033530
21         246.4312       33.3345       -21.024712       24.105576
22         241.3321       30.7683       -16.280193       47.409944
23         231.7314       27.5392        -9.691951       55.242138
24         211.1756       24.5830        -3.743955      101.539391
25         193.1294       22.7913         2.064144       95.347934
26         179.8637       22.0267         9.537699       98.124698
27         171.8167       21.8283        13.120048       86.721774
28         167.9127       22.6468        37.358496       47.567350
29         166.3192       25.3687        54.988523       22.415660
30         171.7768       29.1893        30.147381       23.104025
...             ...           ...              ...             ...
8752       201.1765       29.3094       -23.016304      -35.480752
8754       239.3403       34.0242       -27.037347      -49.252993
8755       253.2434       34.9559       -26.809996      -52.041206
8756       257.5856       35.4617       -24.635875      -53.379236
8757       256.5100       34.5375       -25.268615      -52.431439
8758       250.4548       31.8630       -22.778341      -48.235997
8759       238.3444       28.7036       -20.190085      -41.896757
8760       215.9162       25.6054       -17.694564      -36.326660
8761       196.0375       23.7367       -17.035022      -32.447554
8762       178.5093       22.7860       -16.782579      -31.685681
8763       168.8687       22.3913       -17.312934      -30.590878
8764       165.9606       22.7150       -16.828392      -30.644860
8765       165.2037       23.9888       -18.996376      -31.397702
8766       166.9964       26.5631       -21.957257      -34.737374
```

8767	173.7687	29.0247	-23.908185	-37.066422
8768	179.5052	32.0381	-23.348965	-40.329824
8769	198.8660	34.1407	-20.430098	-39.661120
8770	219.4218	34.2206	-17.067493	-21.862124
8771	228.7437	33.5561	-9.877864	6.029681
8772	236.8704	33.5756	-0.209968	1.975116
8773	234.9184	32.9186	0.502809	-0.810884
8774	231.1339	32.0022	-0.078103	-14.927743
8775	234.4802	31.6617	-11.544949	-4.827103
8776	233.3249	32.5491	-23.345121	-6.086408
8777	238.3045	36.0762	-29.924388	9.704716
8778	263.5611	37.0401	-21.911372	-10.011761
8780	251.9287	33.5002	-18.139910	7.433872
8781	226.7120	30.6642	-8.244008	1.656642
8782	214.1235	29.3810	5.043746	11.181359
8783	207.5106	28.1166	31.612792	-3.659594

	export_node_170	export_node_108	export_node_59	renew_node_24	\
0	-1483.942990	30.327760	-13.871218	153.470195	
1	-1426.811167	28.628956	-11.674057	146.519552	
2	-1374.632122	29.129438	-8.783290	138.456806	
3	-1311.463964	32.912220	-8.360663	102.313463	
4	-1285.376107	42.026312	-7.590000	77.569175	
5	-1256.864499	37.536154	-6.717537	82.851663	
6	-1267.533880	28.812487	-5.332699	64.223940	
7	-1245.461478	6.303337	2.196032	50.600680	
8	-1236.965450	2.706778	21.206081	66.453654	
9	-1265.019227	13.897688	41.972870	110.786784	
10	-1333.540199	25.724247	69.281997	161.459246	
11	-1370.919142	31.812092	78.626633	219.761975	
12	-1402.586240	32.023168	86.647138	258.969846	
13	-1416.671112	28.639073	104.007273	278.758641	
15	-1390.179914	20.114482	98.419034	274.565962	
16	-1525.414121	-3.407993	78.190772	241.326321	
17	-1756.506130	-12.513667	81.893064	274.133355	
18	-1860.564570	-9.606389	88.921471	264.958507	
19	-1897.724337	-12.220960	91.272087	260.232069	
20	-1930.662726	-9.507532	101.874609	429.271704	
21	-1898.906013	-4.281519	100.484876	260.232069	
22	-1734.363439	-11.985458	78.863836	197.120232	
23	-1487.705929	-9.456186	58.363388	309.442621	
24	-1276.601701	-14.704700	89.089751	330.294550	
25	-1133.655934	-13.635200	79.357536	263.846404	
26	-1269.043500	-13.183500	19.291446	266.626661	
27	-1258.139100	-13.070200	18.164159	312.222878	
28	-906.310744	-13.569000	17.763272	308.330518	
29	-1464.916600	-15.218300	24.181379	303.882107	
30	-1216.236898	-17.526800	28.037562	313.891032	

...	...	...	...	...
8752	-1664.042029	33.771070	-13.539574	316.393264
8754	-1947.256135	19.528014	-15.739484	164.591223
8755	-2001.417356	25.996428	-14.629350	160.976889
8756	-2027.311363	18.996654	-13.242404	118.716980
8757	-1973.037502	2.957761	-10.353199	117.604878
8758	-1813.443189	1.401071	-6.064657	82.573637
8759	-1617.384817	-0.805285	3.023686	75.622995
8760	-1426.101490	-6.292839	9.674547	68.116300
8761	-1330.370396	-8.963755	9.745359	67.004197
8762	-1283.221927	-10.635380	14.677757	81.461535
8763	-1260.634082	-10.760473	31.279590	98.421103
8764	-1285.323462	-10.053062	50.168027	110.376209
8765	-1362.106764	-11.374880	58.559508	120.941186
8766	-1508.886286	-13.126104	49.162631	100.923335
8767	-1638.146954	-10.631164	51.292363	75.901020
8768	-1769.356791	-2.612952	46.376855	115.015204
8769	-1836.639625	4.850390	61.966193	197.685432
8770	-1806.669920	13.832932	77.056192	339.124788
8771	-1728.653672	25.868577	103.833084	429.592055
8772	-1680.653125	31.420050	116.753103	478.515254
8773	-1638.752509	-19.731200	-17.862700	500.177665
8774	-1470.699471	-19.179600	-17.363300	499.261266
8775	-1452.690305	29.700201	-17.174200	442.218893
8776	-1491.993437	8.444941	98.356955	221.864521
8777	-1700.871891	4.081316	95.020273	259.397992
8778	-1872.246659	10.507576	90.086575	247.442887
8780	-1792.845586	39.009472	85.463238	331.962704
8781	-1627.009469	5.083337	80.633873	477.092127
8782	-1538.843315	-1.106469	93.283131	580.239668
8783	-1481.595250	-0.120091	86.681057	735.934068

	renew_node_154	renew_node_155	renew_node_25	renew_node_202	\
0	0.370617	29.337907	214.272551	72.803164	
1	0.648579	23.361666	219.077178	61.440821	
2	0.440107	20.916841	192.427051	48.605581	
3	0.416944	16.706308	210.919682	52.813856	
4	0.509598	13.446541	226.059443	48.605581	
5	0.625416	11.952480	218.869784	44.818133	
6	1.482467	10.458420	238.295688	51.340960	
7	2.895443	9.371831	244.448376	55.128408	
8	8.661745	42.433941	244.469218	70.159737	
9	20.134604	105.891829	252.621585	133.034930	
10	30.162869	154.510438	281.913241	206.621388	
11	35.869314	200.357980	320.059947	254.185544	
12	37.420754	218.053820	338.237217	301.456276	
13	35.968062	203.690635	338.832409	363.278801	
15	27.622524	140.441553	333.683362	387.746552	

```
16          12.184026       48.217393     336.254779      236.715491
17          11.720755       75.382121     331.692112      324.668446
18          11.581774       86.791307     320.008917      224.301078
19          11.836573      102.411026     309.328127      155.706190
20          13.458021      124.686103     244.033588      189.161979
21          18.970945      148.862711     213.754066      227.457285
22          28.884943      219.491005     203.730024      175.905911
23          87.187592      253.718563     198.026689      143.712605
24         122.581492      262.275452     169.959370      119.094194
25          73.405281      244.210908     159.797065       78.694750
26          61.869835      303.565839     184.718908       65.017855
27          30.460065      422.411525     140.129202       51.340960
28          34.050414      434.907300     128.964492       48.605581
29          56.495892      416.163638     122.327885       41.872340
30          88.762713      545.060274     121.567440       49.236822
...               ...             ...            ...             ...
8752         0.579089       47.402451     138.539181      372.642786
8754         0.972869       32.597674     104.941356      289.529347
8755         1.111850       37.487325     107.430084      269.540039
8756         0.579089       37.623149     111.059478      207.888804
8757         0.416944       33.412616     119.113278      153.391638
8758         0.602252       36.536560     148.459527      119.515021
8759         0.486434       34.906676     205.320044       97.842403
8760         0.741234       38.166443     222.568310       95.738265
8761         0.972869       29.337907     228.962958      114.044263
8762         0.857051       19.558604     228.720999      128.562813
8763         1.436140       19.422781     232.177565      143.923018
8764         2.270028       14.261482     244.482941      139.083502
8765         2.038392       14.397306     276.214221      119.304607
8766         1.227668       10.322597     293.946407       95.107024
8767         2.038392        9.236008     302.622388       79.115577
8768         4.976219       25.274098     323.525866      102.943769
8769         6.186009       61.953969     337.450836      153.536165
8770         8.702043      113.131238     348.035808      214.968369
8771        11.755251      157.055922     347.885755      258.790909
8772        13.836786      213.047774     345.972212      284.008998
8773        25.800815      246.548254     345.180380      282.104378
8774        24.070193      203.415100     348.674041      251.512486
8775        13.016961      153.333330     342.565400      201.036896
8776         5.605578       68.590939     291.353982       53.865925
8777        11.164830       63.429641     227.303806       72.803164
8778        10.052979       58.404166     254.126762       52.393029
8780         4.447401       54.872751     175.420745      117.410883
8781         5.281289       76.468710     146.523850      143.081363
8782         7.667134       80.815067     126.060977      160.545706
8783         8.825311       77.012005     122.051360      143.502191

      renew_node_86  renew_node_99  prod_gen_5  prod_gen_7   prod_gen_9  \
```

0	24.167230	11.497852	630.000000	216.0	2087.000000
1	22.190728	14.947207	630.000000	216.0	2087.000000
2	27.805791	16.096993	630.000000	216.0	2087.000000
3	26.795080	23.871730	630.000000	216.0	1285.816619
4	13.723213	28.525623	630.000000	216.0	1161.453333
5	12.083615	20.860388	630.000000	216.0	1187.137617
6	10.399096	13.304657	630.000000	216.0	1735.962751
7	6.962678	14.180684	630.000000	216.0	1610.288014
8	10.405837	16.849188	589.355080	216.0	0.000000
9	24.563587	21.137547	0.000000	216.0	0.000000
10	35.466481	34.093673	0.000000	0.0	0.000000
11	42.296404	58.418471	0.000000	0.0	0.000000
12	44.415270	73.544911	89.888871	0.0	0.000000
13	45.320811	91.973110	0.000000	0.0	0.000000
15	14.846226	96.301855	0.000000	0.0	0.000000
16	14.060117	123.848289	0.000000	0.0	0.000000
17	9.882511	119.741914	76.756049	0.0	0.000000
18	9.298544	153.523697	630.000000	43.2	0.000000
19	9.949891	199.077091	630.000000	0.0	0.000000
20	13.318929	173.289052	630.000000	0.0	0.000000
21	9.972352	179.147481	0.000000	216.0	0.000000
22	12.937105	183.801374	0.000000	0.0	0.000000
23	19.585339	227.493210	0.000000	0.0	0.000000
24	0.000000	285.365731	0.000000	0.0	0.000000
25	0.000000	319.476025	0.000000	0.0	0.000000
26	0.000000	385.123283	0.000000	0.0	0.000000
27	0.000000	388.736894	0.000000	0.0	0.000000
28	0.000000	344.388037	0.000000	0.0	0.000000
29	0.000000	346.194842	0.000000	0.0	0.000000
30	0.000000	287.720053	0.000000	0.0	0.000000
...	...	...	...	...	...
8752	17.900820	10.293315	630.000000	216.0	2087.000000
8754	9.073941	16.480254	630.000000	216.0	2087.000000
8755	6.131649	10.786080	630.000000	216.0	2087.000000
8756	1.976502	9.307785	630.000000	216.0	2087.000000
8757	1.527297	14.290187	630.000000	216.0	2087.000000
8758	1.123013	14.454442	630.000000	216.0	2087.000000
8759	1.078092	18.944080	630.000000	216.0	2087.000000
8760	1.639598	22.393435	630.000000	216.0	2087.000000
8761	1.909121	19.272590	630.000000	0.0	2087.000000
8762	1.392536	17.301529	630.000000	0.0	2087.000000
8763	1.325155	26.335556	630.000000	0.0	2087.000000
8764	1.257774	28.087609	630.000000	0.0	2087.000000
8765	1.392536	26.992576	630.000000	0.0	2087.000000
8766	1.370075	33.234267	630.000000	0.0	2087.000000
8767	3.189356	41.611273	630.000000	0.0	2087.000000
8768	13.871550	75.015603	515.810531	0.0	2087.000000
8769	25.773008	118.563817	521.793750	0.0	2087.000000

8770	35.709014	132.009118	232.585510	0.0	0.000000
8771	37.095782	185.078698	0.000000	0.0	0.000000
8772	38.916159	287.753869	0.000000	0.0	0.000000
8773	30.477718	403.776184	0.000000	0.0	0.000000
8774	27.523778	384.457439	0.000000	0.0	0.000000
8775	15.317891	356.972976	0.000000	0.0	0.000000
8776	10.264335	270.199517	0.000000	0.0	0.000000
8777	9.029021	254.266780	0.000000	0.0	0.000000
8778	12.442979	211.177211	0.000000	0.0	2087.000000
8780	13.588452	160.422408	0.000000	0.0	2087.000000
8781	12.600201	202.088433	0.000000	0.0	40.513982
8782	18.619548	262.862793	0.000000	0.0	0.000000
8783	14.329640	253.171746	0.000000	216.0	690.879932

	prod_gen_25	prod_gen_31	prod_gen_41
0	957.000000	146.000000	0.000000
1	957.000000	146.000000	0.000000
2	957.000000	146.000000	0.000000
3	957.000000	146.000000	0.000000
4	957.000000	146.000000	0.000000
5	957.000000	146.000000	0.000000
6	957.000000	146.000000	0.000000
7	957.000000	146.000000	0.000000
8	191.400000	146.000000	0.000000
9	0.000000	10.990596	0.000000
10	0.000000	0.000000	0.000000
11	0.000000	0.000000	0.000000
12	0.000000	0.000000	0.000000
13	0.000000	0.000000	0.000000
15	0.000000	0.000000	0.000000
16	0.000000	0.000000	0.000000
17	957.000000	0.000000	0.000000
18	957.000000	146.000000	0.000000
19	957.000000	18.209404	0.000000
20	957.000000	146.000000	0.000000
21	0.000000	0.000000	0.000000
22	0.000000	0.000000	0.000000
23	0.000000	0.000000	0.000000
24	0.000000	0.000000	0.000000
25	0.000000	0.000000	0.000000
26	0.000000	0.000000	0.000000
27	0.000000	0.000000	0.000000
28	0.000000	0.000000	0.000000
29	0.000000	0.000000	0.000000
30	0.000000	0.000000	0.000000
...	...	...	...
8752	957.000000	146.000000	1400.000000
8754	957.000000	146.000000	1400.000000

```
8755   957.000000   146.000000  1400.000000
8756   957.000000   146.000000  1400.000000
8757   957.000000   146.000000  1400.000000
8758   957.000000   146.000000  1400.000000
8759   957.000000   146.000000  1400.000000
8760   957.000000   146.000000  1400.000000
8761   957.000000   146.000000  1400.000000
8762   957.000000    28.429972  1323.121084
8763   957.000000     0.000000   582.484027
8764   957.000000     0.000000   188.162833
8765   957.000000     0.000000   871.768327
8766   957.000000     0.000000  1400.000000
8767   957.000000     0.000000  1400.000000
8768   957.000000     0.000000   881.359767
8769     0.000000     0.000000     0.000000
8770     0.000000     0.000000     0.000000
8771     0.000000     0.000000     0.000000
8772     0.000000     0.000000     0.000000
8773     0.000000     0.000000     0.000000
8774     0.000000     0.000000     0.000000
8775     0.000000     0.000000     0.000000
8776     0.000000     0.000000     0.000000
8777     0.000000     0.000000     0.000000
8778     0.000000     0.000000     0.000000
8780   957.000000     0.000000     0.000000
8781   389.722301     0.000000     0.000000
8782    41.189710     0.000000     0.000000
8783     0.000000     0.000000     0.000000

[8311 rows x 20 columns]
```

1.1.4 Save your chosen sensors

Now that you have chosen 20 sensors which are representative of your data, create a DataFrame that contains these sensors. You can save them to csv file using the code in the following cell.

```
In [32]: # Assuming of course that your reduced data set is called data_reduced

         data_reduced['time']=data_nona['time']
         data_reduced.to_csv('reduced_field_data.csv')
```

```
/home/student/.local/lib/python3.5/site-packages/ipykernel_launcher.py:3: SettingWithCopyWarnin
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  This is separate from the ipykernel package so we can avoid doing imports until
```