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).*

1: #!/bin/bash

2: cut -d ',' -f1 --complement energy_demand_data.csv | paste -d ',' - exchange_data.csv > all.csv

3: cut -d ',' -f1 --complement generator_production_data.csv | paste -d ',' - all.csv > helper.csv

4: cut -d ',' -f1 --complement renewable_production_data.csv | paste -d ',' - helper.csv > all.csv

5: rm -rf helper.csv

line 1: makes sure the script runs in bash

line 2: cuts the time column out of energy_demand_data.csv and pipes the output to paste. Paste writes linewise from the piped input and exchange_data into helper.csv . Notice that exchange_data.csv still has the time coloumn.

line 3-4: does the same as line 2, but with the rest of the files, changing to update/read either helper.csv or all.csv

line 5: removes helper.csv

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.

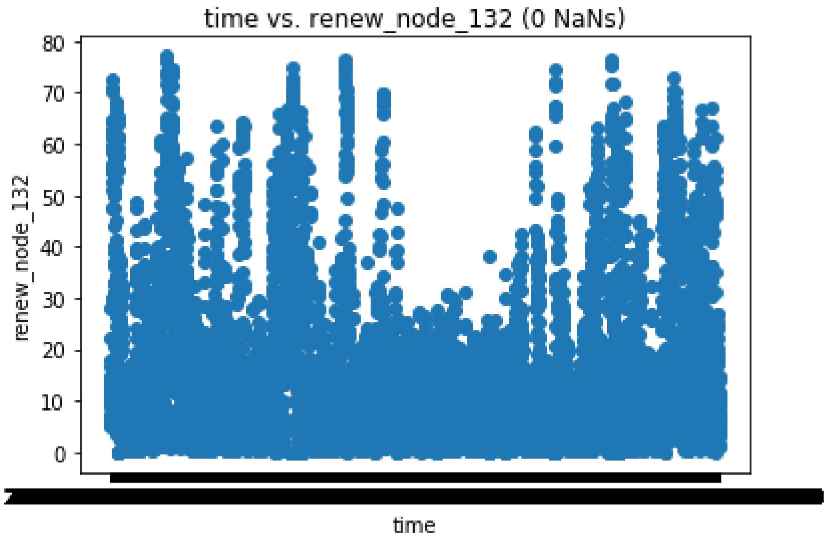
- *Hint: remember the hint given in Exc.13.3, on how to find out if your data contains NaN values*

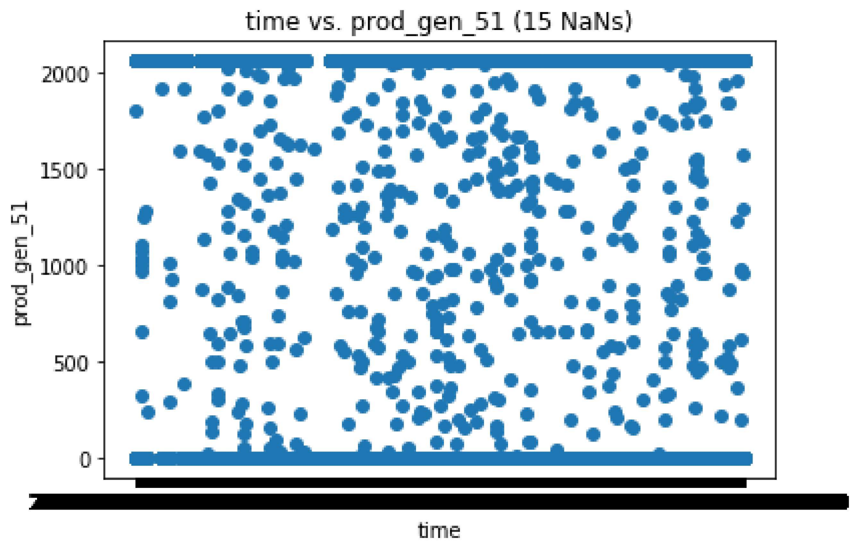
In [1]:
```
# You will probably be needing the following libraries:
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import numpy as np
```

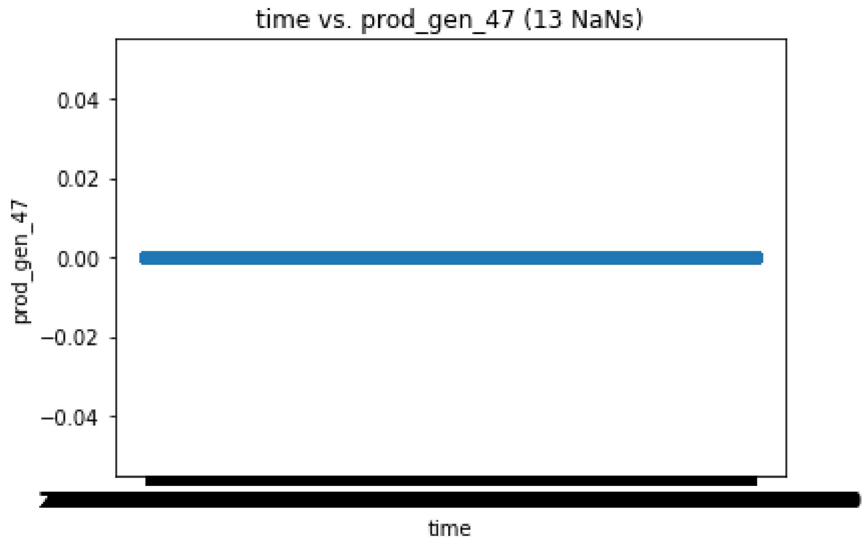
In [2]:
```
# Import your data here
data = {}
data['Raw'] = pd.read_csv('all.csv')
print("NaN's:",data['Raw'].isnull().sum(0).sum())
print("Data shape: ", data['Raw'].shape)
```

```
NaN's: 1711
Data shape:  (8784, 137)
```

In [3]:
```
# First do some time series plots on the data, would it be practical to plot all 

# If you have a cluster:
# Sure!  you can see all the data with only 1 command


# If you only have a laptop:
# NO!   There's way too many colomns for a scatter matrix! it would take forever

plt.figure(1)
plt.plot(data['Raw']['time'],data['Raw']['renew_node_132'],'o')
plt.xlabel("time")
plt.ylabel("renew_node_132")
plt.title("time vs. renew_node_132 (0 NaNs)")
plt.show()

plt.figure(2)
plt.plot(data['Raw']['time'],data['Raw']['prod_gen_51'],'o')
plt.xlabel("time")
plt.ylabel("prod_gen_51")
plt.title("time vs. prod_gen_51 (15 NaNs)")
plt.show()

plt.figure(3)
plt.plot(data['Raw']['time'],data['Raw']['prod_gen_47'],'o')
plt.xlabel("time")
plt.ylabel("prod_gen_47")
plt.title("time vs. prod_gen_47 (13 NaNs)")
plt.show()
```







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

Since there is no direct pattern in figure 2 (for the dots inbetween), interpolation is NOT an option. Dropping rows containing a NaN seems resonable since the columns are to be compared.

In [41]:
```
# Get rid of your NaNs here
data['noNaN'] = data['Raw'].dropna(axis=0, how='any')
```

Feature reduction

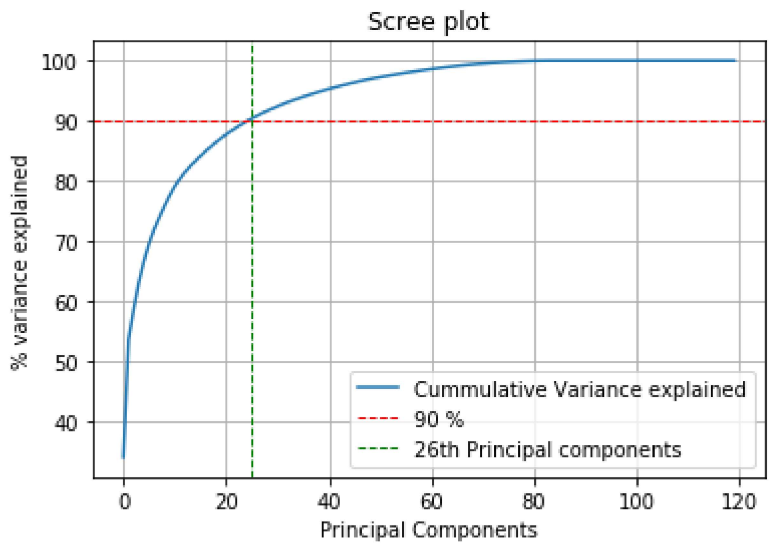
Since you must reduce the amount of sensors, you need to find out which ones you can get rid of.

Q#3 Why would PCA be useful for this?

PCA lets you inspect the data on a 2D graph, which is very useful considering the size of this dataset.

When using PCA it's possible to narrow down the number of parameters based upon the principal components. You will lose some data, but if you eg. can reduce the ammount of sensors from 137 to 30 while only losing 5% accuracy, it's propably worth it.

In [5]:
```
from sklearn.decomposition import PCA

data['noTime'] = data['noNaN'].drop('time',1)             # remove timestamps
data['norm'] = (data['noTime'] - data['noTime'].mean()) / (data['noTime'].std())
data['normNoNan'] = data['norm'].dropna(axis=1, how='all') #This removes columns 

pca = PCA(n_components=data['normNoNan'].shape[1])
pca.fit_transform(data['normNoNan'])

cummulative_var = np.cumsum(pca.explained_variance_ratio_)

plt.figure(5)
plt.plot(cummulative_var*100, label='Cummulative Variance explained')
plt.axhline(y=90, linewidth=1, color='r', linestyle='dashed', label='90 %')
plt.axvline(x=25, linewidth=1, color='g', linestyle='dashed', label='26th Princip
plt.title("Scree plot")
plt.ylabel("% variance explained")
plt.xlabel("Principal Components")
plt.legend()
plt.grid()
plt.show()
```



Scree plot

Q#4 How many principal components do you need to explain 90 % of the variance?

As seen on the figure above, 26 principal component describes just above 90 % of the variance.

In [14]:
```
print("Variance explained with 25 components: {0:.2f} %".format(cummulative_var[2
```

```
Variance explained with 25 components: 90.35 %
```

Clustering

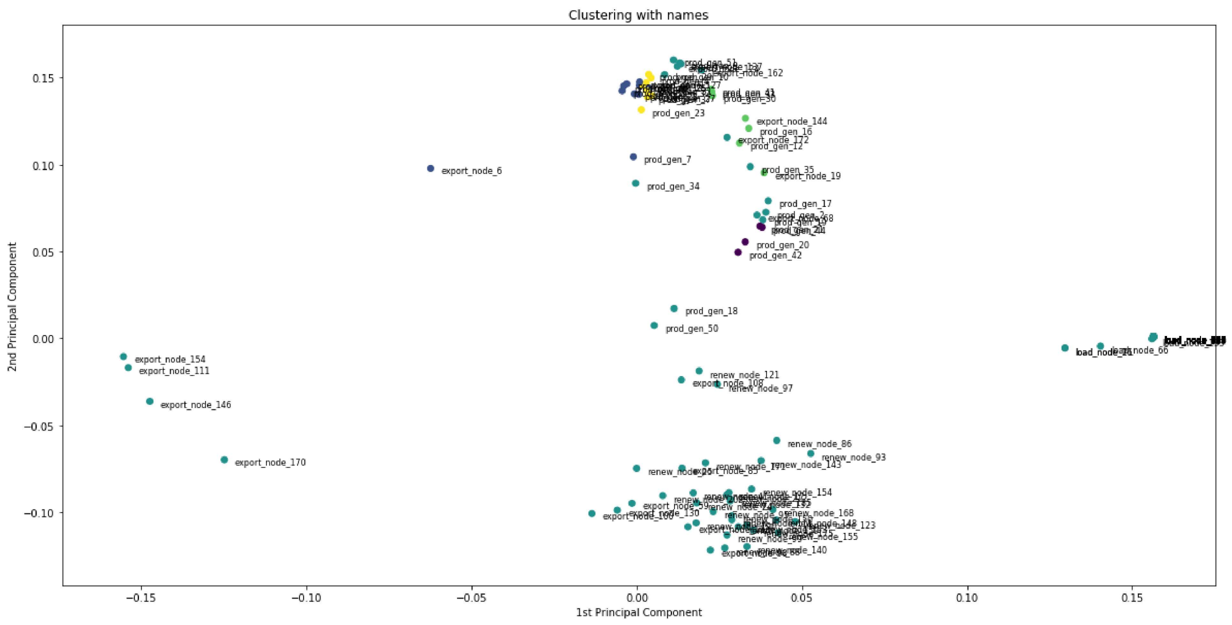
You want to reduce the amount of field sensors to 20. You should now have from the previous question, an array with all your loading vectors (pca.components_), one vector per principal component, with 137 elements (one per each sensor). Use clustering to group sensors that behave the same.

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

The sensors are divided into 5 clusters calculated by K-Means clustering. From each cluster we select the closest point, save and remove it from the dataset and repeat the process. For a total of 20 sensors, we need 4 rounds. Choosing the point closest to the center is done in hope of the best representation of the cluster. This is done to show the diversity of the dataset.

In [15]:
```
# Apply your clustering here
from sklearn.cluster import KMeans

pcaproj = pca.components_[[np.arange(0,26)]]
kmeans = KMeans(n_clusters=5, random_state=14).fit(pcaproj.T)
cluster_pred = kmeans.predict(pcaproj.T)
```

In [23]:
```
# Plot with names
fig, ax = plt.subplots(figsize=(20,10))
plt.scatter(pcaproj[0],pcaproj[1], c=cluster_pred)
for i, txt in enumerate(data["normNoNan"].columns):
    ax.annotate(txt, (pca.components_[0][i],pca.components_[1][i]),xytext=(10,-5)
plt.title("Clustering with names")
plt.xlabel("1st Principal Component")
plt.ylabel("2nd Principal Component")
plt.show()
```



```
It is impossible to interpret this figure as we are using 26 principle
components, which means we would need 26 dimensions to visualize every node. An
alternative is to generate a table showing which column/sensor belongs in which
cluster.
```

In [29]:
```
{i: np.where(kmeans.labels_ == i)[0] for i in range(kmeans.n_clusters)}
```

Out[29]:
```
{0: array([42, 43, 60, 61], dtype=int64),
 1: array([ 31,  32,  36,  44,  50,  56,  57,  58, 104, 119], dtype=int64),
 2: array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
         13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
         26,  27,  28,  29,  33,  34,  39,  40,  41,  53,  54,  62,  63,
         64,  65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,
         77,  78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,
         90,  91,  92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102,
        103, 105, 106, 107, 108, 109, 110, 112, 113, 115, 116, 117, 118], dtype
=int64),
 3: array([ 35,  38,  49,  52,  59, 111, 114], dtype=int64),
 4: array([30, 37, 45, 46, 47, 48, 51, 55], dtype=int64)}
```

In [40]:
```
from sklearn.metrics import pairwise_distances_argmin_min

temp = data['normNoNan']
current = pd.DataFrame(pcaproj.T).copy()
highs = {}

for _ in range(4):                              # Loop for finding sensor nearest
    closest, _ = pairwise_distances_argmin_min(kmeans.cluster_centers_, current.v
    for item in closest:
        highs[temp.columns.values[item]] = item
        current.iloc[item] = 9000               # Set distance for "used sensor"

relevant_values = ['time'] + list(highs.keys())      # Define relevant columns: t
data['reduced'] = data['noNaN'][relevant_values] # Copy values to new dataframe

print("Chosen sensor\t column id\n-------------------------\n{}".format(pd.Series
```

```
Chosen sensor    column id
-------------------------
export_node_127    119
load_node_134       66
load_node_153       71
load_node_63        85
load_node_98        97
prod_gen_13         36
prod_gen_15         37
prod_gen_16         38
prod_gen_20         42
prod_gen_21         43
prod_gen_22         44
prod_gen_25         46
prod_gen_29         48
prod_gen_30         49
prod_gen_33         52
prod_gen_4          30
prod_gen_40         58
prod_gen_41         59
prod_gen_42         60
prod_gen_44         61
dtype: int64
```

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

data['reduced'].to_csv('reduced_field_data.csv') # Save reduced data as .csv file
```