This notebook is designed to run in a IBM Watson Studio default runtime (NOT the Watson Studio Apache Spark Runtime as the default runtime with 1 vCPU is free of charge). Therefore, we install Apache Spark in local mode for test purposes only. Please don't use it in production.

In case you are facing issues, please read the following two documents first:

https://github.com/IBM/skillsnetwork/wiki/Environment-Setup (https://github.com/IBM/skillsnetwork/wiki/Environment-Setup)

https://github.com/IBM/skillsnetwork/wiki/FAQ (https://github.com/IBM/skillsnetwork/wiki/FAQ)

Then, please feel free to ask:

https://coursera.org/learn/machine-learning-big-data-apache-spark/discussions/all (https://coursera.org/learn/machine-learning-big-data-apache-spark/discussions/all)

Please make sure to follow the guidelines before asking a question:

https://github.com/IBM/skillsnetwork/wiki/FAQ#im-feeling-lost-and-confused-please-help-me (https://github.com/IBM/skillsnetwork/wiki/FAQ#im-feeling-lost-and-confused-please-help-me)

If running outside Watson Studio, this should work as well. In case you are running in an Apache Spark context outside Watson Studio, please remove the Apache Spark setup in the first notebook cells.

In [1]:

```
from IPython.display import Markdown, display
def printmd(string):
    display(Markdown('# <span style="color:red">'+string+'</span>'))


if ('sc' in locals() or 'sc' in globals()):
    printmd('<<<<<!!!!! It seems that you are running in a IBM Watson Studio Apache Spa
rk Notebook. Please run it in an IBM Watson Studio Default Runtime (without Apache Spar
k) !!!!!>>>>>')
```

In [2]:

```
!pip install pyspark==2.4.5
```

```
Collecting pyspark==2.4.5
  Downloading pyspark-2.4.5.tar.gz (217.8 MB)
     |████████████████████████████████| 217.8 MB 12 kB/s s eta 0:00:01
Collecting py4j==0.10.7
  Downloading py4j-0.10.7-py2.py3-none-any.whl (197 kB)
     |████████████████████████████████| 197 kB 66.1 MB/s eta 0:00:01
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-2.4.5-py2.py3-none-any.whl s
ize=218257927 sha256=31a2a9fa9e88c8f751010f9590fb5d00e768d3b604fd084348225
9fa13594be8
  Stored in directory: /tmp/wsuser/.cache/pip/wheels/01/c0/03/1c241c9c482b
647d4d99412a98a5c7f87472728ad41ae55e1e
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.7 pyspark-2.4.5
```

In [3]:

```
try:
    from pyspark import SparkContext, SparkConf
    from pyspark.sql import SparkSession
except ImportError as e:
    printmd('<<<<<!!!!! Please restart your kernel after installing Apache Spark !!!!!>
>>>>')
```

In [4]:

```
sc = SparkContext.getOrCreate(SparkConf().setMaster("local[*]"))

spark = SparkSession \
    .builder \
    .getOrCreate()
```

Exercise 3.1

Welcome to Exercise 3.

Now it is time to grab a PARQUET file and create a dataframe out of it. Using SparkSQL you can handle it like a database.

In [5]:

```
!wget https://github.com/IBM/coursera/blob/master/coursera_ds/washing.parquet?raw=true
!mv washing.parquet?raw=true washing.parquet
```

```
--2021-03-25 01:02:25--  https://github.com/IBM/coursera/blob/master/cours
era_ds/washing.parquet?raw=true
Resolving github.com (github.com)... 140.82.113.3
Connecting to github.com (github.com)|140.82.113.3|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://github.com/IBM/skillsnetwork/blob/master/coursera_ds/was
hing.parquet?raw=true [following]
--2021-03-25 01:02:26--  https://github.com/IBM/skillsnetwork/blob/master/
coursera_ds/washing.parquet?raw=true
Reusing existing connection to github.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://github.com/IBM/skillsnetwork/raw/master/coursera_ds/wash
ing.parquet [following]
--2021-03-25 01:02:26--  https://github.com/IBM/skillsnetwork/raw/master/c
oursera_ds/washing.parquet
Reusing existing connection to github.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/IBM/skillsnetwork/master/cours
era_ds/washing.parquet [following]
--2021-03-25 01:02:26--  https://raw.githubusercontent.com/IBM/skillsnetwo
rk/master/coursera_ds/washing.parquet
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.19
9.110.133, 185.199.109.133, 185.199.108.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.19
9.110.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 112048 (109K) [application/octet-stream]
Saving to: ‘washing.parquet?raw=true’

washing.parquet?raw 100%[===================>] 109.42K  --.-KB/s    in 0.0
03s

2021-03-25 01:02:26 (40.2 MB/s) - ‘washing.parquet?raw=true’ saved [11204
8/112048]
```

In [6]:

```
df = spark.read.parquet('washing.parquet')
```

So let's check how may rows we have got

In [7]:

```
df.count()
```

Out[7]:

```
2058
```

Now we register the data frame in the ApacheSparkSQL catalog so that we can query it using SQL

In [8]:

```
df.createOrReplaceTempView("washing")
spark.sql("SELECT * FROM washing").show()
```

```
+--------------------+--------------------+-----+--------+----------+-----
----+--------+-----+-----------+-------------+-------+
|                 _id|                _rev|count|flowrate|fluidlevel|frequ
ency|hardness|speed|temperature|           ts|voltage|
+--------------------+--------------------+-----+--------+----------+-----
----+--------+-----+-----------+-------------+-------+
|0d86485d0f88d1f9d...|1-57940679fb8a713...|    4|      11|acceptable|
null|      77| null|        100|1547808723923|   null|
|0d86485d0f88d1f9d...|1-15ff3a0b304d789...|    2|    null|      null|
null|    null| 1046|       null|1547808729917|   null|
|0d86485d0f88d1f9d...|1-97c2742b68c7b07...|    4|    null|      null|
71|    null| null|       null|1547808731918|    236|
|0d86485d0f88d1f9d...|1-eefb903dbe45746...|   19|      11|acceptable|
null|      75| null|         86|1547808738999|   null|
|0d86485d0f88d1f9d...|1-5f68b4c72813c25...|    7|    null|      null|
75|    null| null|       null|1547808740927|    235|
|0d86485d0f88d1f9d...|1-cd4b6c57ddbe77e...|    5|    null|      null|
null|    null| 1014|       null|1547808744923|   null|
|0d86485d0f88d1f9d...|1-a35b25b5bf43aaf...|   32|      11|acceptable|
null|      73| null|         84|1547808752028|   null|
|0d86485d0f88d1f9d...|1-b717f7289a8476d...|   48|      11|acceptable|
null|      79| null|         84|1547808768065|   null|
|0d86485d0f88d1f9d...|1-c2f1f8fcf178b2f...|   18|    null|      null|
73|    null| null|       null|1547808773944|    228|
|0d86485d0f88d1f9d...|1-15033dd9eebb4a8...|   59|      11|acceptable|
null|      72| null|         96|1547808779093|   null|
|0d86485d0f88d1f9d...|1-753dae825f9a6c2...|   62|      11|acceptable|
null|      73| null|         88|1547808782113|   null|
|0d86485d0f88d1f9d...|1-b168089f44f03f0...|   13|    null|      null|
null|    null| 1097|       null|1547808784940|   null|
|0d86485d0f88d1f9d...|1-403b687c6be0dea...|   23|    null|      null|
80|    null| null|       null|1547808788955|    236|
|0d86485d0f88d1f9d...|1-195551e0455a24b...|   72|      11|acceptable|
null|      77| null|         87|1547808792134|   null|
|0d86485d0f88d1f9d...|1-060a39fc6c2ddee...|   26|    null|      null|
62|    null| null|       null|1547808797959|    233|
|0d86485d0f88d1f9d...|1-2234514bffee465...|   27|    null|      null|
61|    null| null|       null|1547808800960|    226|
|0d86485d0f88d1f9d...|1-4265898bb401db0...|   82|      11|acceptable|
null|      79| null|         96|1547808802154|   null|
|0d86485d0f88d1f9d...|1-2fbf7ca9a0425a0...|   94|      11|acceptable|
null|      73| null|         90|1547808814186|   null|
|0d86485d0f88d1f9d...|1-203c0ee6d7fbd21...|   97|      11|acceptable|
null|      77| null|         88|1547808817190|   null|
|0d86485d0f88d1f9d...|1-47e1965db94fcab...|  104|      11|acceptable|
null|      75| null|         80|1547808824198|   null|
+--------------------+--------------------+-----+--------+----------+-----
----+--------+-----+-----------+-------------+-------+
only showing top 20 rows
```

So let's visualize voltage using a box plot to get an idea on the value distribution of this parameter. First, we have to create a python list. Make sure you use the sample function in order to not blast your spark driver or plotting library.

In [9]:

```
result = spark.sql("select voltage from washing where voltage is not null")
result_array = result.rdd.map(lambda row : row.voltage).sample(False,0.1).collect()

#just print the 1st 15 elements
result_array[:15]
```

Out[9]:

```
[235, 225, 236, 227, 229, 239, 226, 235, 237, 235, 235, 231, 235, 231, 24
8]
```

Now we have to activate the notebook to show the plots directly under the cell

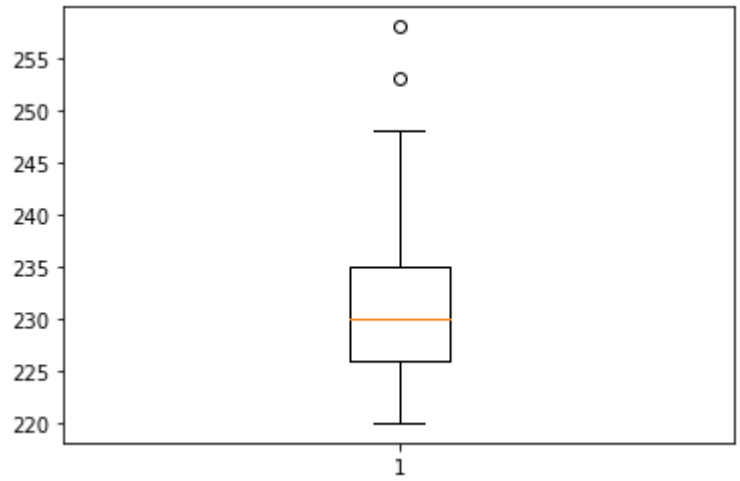
In [10]:

```
%matplotlib inline
```

Now it's time to plot the python list by importing the matplotlib library, calling "boxplot" and "show". Note that you can see mean (red line) around 230. Then you see that 50% of all values are between 225 and 235 (blue box). Per default values up to 250 are not seen as outliers (little, black horizontal line). And the "plus" symbols on top are definitely outliers. Congratulations, you've written your first anomaly detection algorithm. Unfurtunately you still need a brain attached to it, so we'll cover on how we write one without a brain needed in the next course. And in the course after that we'll even tell you how to implement a artificial brain to further improve it, so stay tuned :)

In [11]:

```
import matplotlib.pyplot as plt
plt.boxplot(result_array)
plt.show()
```



Since we are dealing with time series data we want to make use of the time dimension as well. The least complex plots are run charts where the time domain (dimension) is represented at the horizontal x-axis and the y-axis shows the actual sensor value. Let's do this for voltage as well.

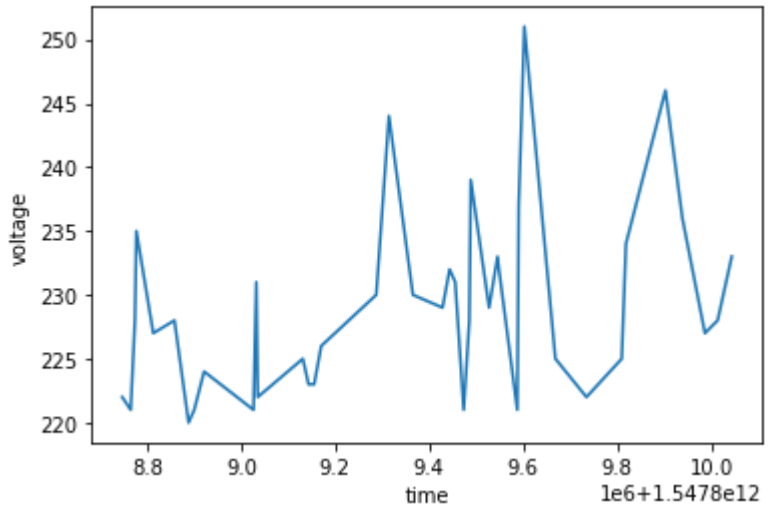
In [12]:

```
result = spark.sql("select voltage,ts from washing where voltage is not null order by t
s asc")
result_rdd = result.rdd.sample(False,0.1).map(lambda row : (row.ts,row.voltage))
result_array_ts = result_rdd.map(lambda ts_voltage: ts_voltage[0]).collect()
result_array_voltage = result_rdd.map(lambda ts_voltage: ts_voltage[1]).collect()
print(result_array_ts[:15])
print(result_array_voltage[:15])
```

```
[1547808746929, 1547808764937, 1547808773944, 1547808776948, 154780881297
0, 1547808858035, 1547808888076, 1547808900085, 1547808921096, 15478090262
03, 1547809032208, 1547809035209, 1547809131265, 1547809143268, 1547809155
272]
[222, 221, 228, 235, 227, 228, 220, 221, 224, 221, 231, 222, 225, 223, 22
3]
```

In [13]:

```
plt.plot(result_array_ts,result_array_voltage)
plt.xlabel("time")
plt.ylabel("voltage")
plt.show()
```



But this time we want to only plot data worth of one hour. Therefore we first have to find out in which date range we have data available:

In [14]:

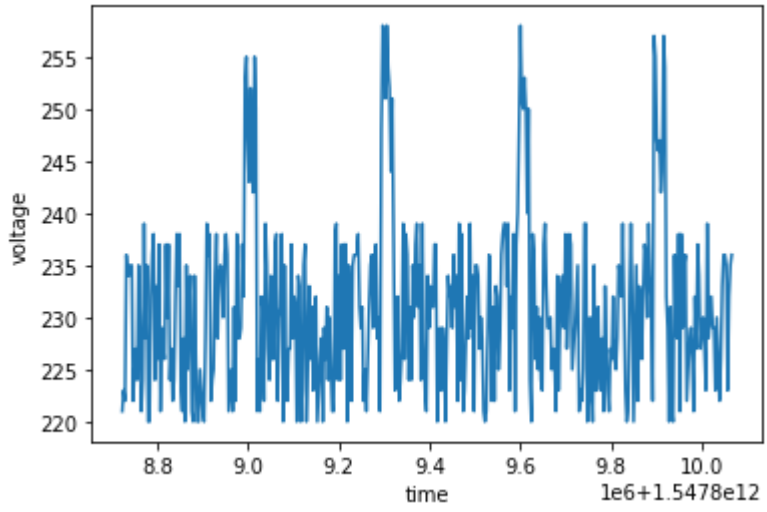
```
spark.sql("select min(ts),max(ts) from washing").show()
```

```
+-------------+-------------+
|      min(ts)|      max(ts)|
+-------------+-------------+
|1547808720911|1547810064867|
+-------------+-------------+
```

Now lets repeat the previous steps but only show data for hour. We've found out the low and high boundary of data available and we know that "ts" stand for "timestamp". Timestamp are the number of millisecons passed since the 1st of Jan. 1970. You can also use an online tool like http://www.epochconverter.com/ (http://www.epochconverter.com/) to convert these. But for now just an interval of 60 minutes (1000*60*60)=3600000 within the range above (note that we have removed the sample function because the data set is already reduced)

In [15]:

```
result = spark.sql(
"""
select voltage,ts from washing
    where voltage is not null and
    ts > 1547808720911 and
    ts <= 1547810064867+3600000
    order by ts asc
""")
result_rdd = result.rdd.map(lambda row : (row.ts,row.voltage))
result_array_ts = result_rdd.map(lambda ts_voltage: ts_voltage[0]).collect()
result_array_voltage = result_rdd.map(lambda ts_voltage: ts_voltage[1]).collect()
plt.plot(result_array_ts,result_array_voltage)
plt.xlabel("time")
plt.ylabel("voltage")
plt.show()
```



As you can see we are not only able to spot the outliers but also see a time pattern of these outliers occuring which can be used for further downstream analysis. Again your brain was already capable of spotting the pattern. In the next two coursera courses we will teach a machine to spot those patterns as well.

Now we've plotted a maximum of two dimensions at a time, let's go for three in a so-called 3D scatter plot. Again we have to create python lists (with applied sampling if necessary) from three properties of our data.

In [16]:

```
result_df = spark.sql("""
select hardness,temperature,flowrate from washing
    where hardness is not null and
    temperature is not null and
    flowrate is not null
""")
result_rdd = result_df.rdd.sample(False,0.1).map(lambda row : (row.hardness,row.tempera
ture,row.flowrate))
result_array_hardness = result_rdd.map(lambda hardness_temperature_flowrate: hardness_t
emperature_flowrate[0]).collect()
result_array_temperature = result_rdd.map(lambda hardness_temperature_flowrate: hardnes
s_temperature_flowrate[1]).collect()
result_array_flowrate = result_rdd.map(lambda hardness_temperature_flowrate: hardness_t
emperature_flowrate[2]).collect()
```

Once done it is very simple to import the necessary library and create a scatter plot

In [17]:

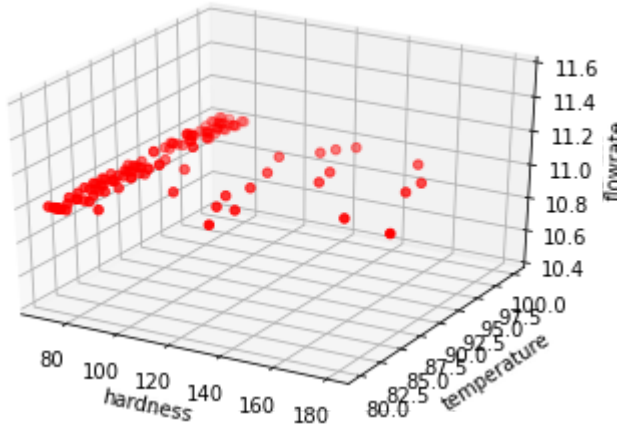
```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.scatter(result_array_hardness,result_array_temperature,result_array_flowrate, c='r',
marker='o')

ax.set_xlabel('hardness')
ax.set_ylabel('temperature')
ax.set_zlabel('flowrate')

plt.show()
```



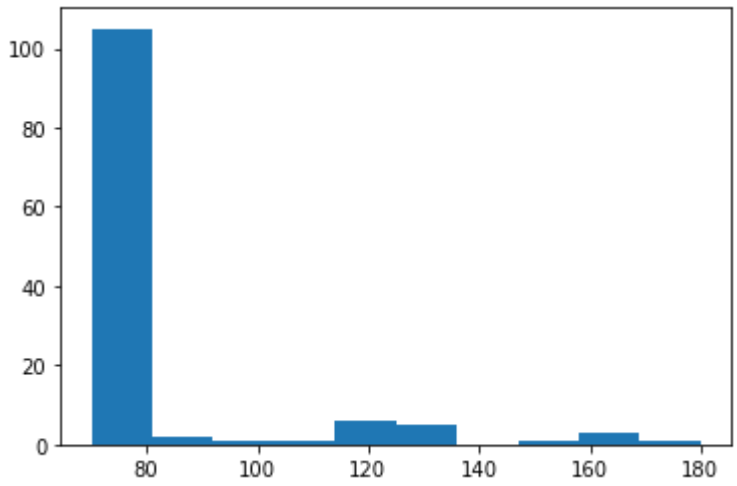
You will notice that the individual points resemble in some sort of plane. But this is not a surprise. Actually we can draw the following conclusions from the plot:

- most of the data points are lying around hardness 60-80, temperature 80-100 and flowrate 80-100
- there are some outliers, especially when it comes to the range of hardness 100-200
- the data follows some narrow boundaries

So lets double-check what's going on with hardness since it seems that it really sticks around 60-80 and very seldom creates values above that. We can use a histogram for that which bins together certain value ranges and counts the frequency of occurences of values within this range. Those frequencies are ordered and shown as a bar diagram, so let's plot it:

In [18]:

```
plt.hist(result_array_hardness)
plt.show()
```



Our assumpion was correct, nearly all values are around 60-80 with very less values about that threshold. This concludes Exercice 3.1.

In [ ]:

```

```