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 10 kB/s s eta 0:00:01
|███                             | 20.5 MB 14.9 MB/s eta 0:00:14
Collecting py4j==0.10.7
  Downloading py4j-0.10.7-py2.py3-none-any.whl (197 kB)
     |████████████████████████████████| 197 kB 38.0 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=3bbb8997c9b105bcb37aa0c193a355aaf19f5d98c38168a66f8bc
c0690eebdaf
  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 3.2

Welcome to the last exercise of this course. This is also the most advanced one because it somehow glues everything together you've learned.

These are the steps you will do:

- load a data frame from cloudant/ApacheCouchDB
- perform feature transformation by calculating minimal and maximal values of different properties on time windows (we'll explain what a time windows is later in here)
- reduce these now twelve dimensions to three using the PCA (Principal Component Analysis) algorithm of SparkML (Spark Machine Learning) => We'll actually make use of SparkML a lot more in the next course
- plot the dimensionality reduced data set

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
```

```
--2021-03-31 00:16:43--  https://github.com/IBM/coursera/blob/master/cours
era_ds/washing.parquet?raw=true
Resolving github.com (github.com)... 140.82.112.4
Connecting to github.com (github.com)|140.82.112.4|: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]
--2021-03-31 00:16:43--  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]
--2021-03-31 00:16:44--  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]
--2021-03-31 00:16:44--  https://raw.githubusercontent.com/IBM/skillsnetwo
rk/master/coursera_ds/washing.parquet
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.19
9.111.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.19
9.111.133|: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

2021-03-31 00:16:44 (33.6 MB/s) - ‘washing.parquet?raw=true’ saved [11204
8/112048]
```

In [6]:

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

This is the feature transformation part of this exercise. Since our table is mixing schemas from different sensor data sources we are creating new features. In other word we use existing columns to calculate new ones. We only use min and max for now, but using more advanced aggregations as we've learned in week three may improve the results. We are calculating those aggregations over a sliding window "w". This window is defined in the SQL statement and basically reads the table by a one by one stride in direction of increasing timestamp. Whenever a row leaves the window a new one is included. Therefore this window is called sliding window (in contrast to tubling, time or count windows). More on this can be found here: https://flink.apache.org/news/2015/12/04/Introducing-windows.html (https://flink.apache.org/news/2015/12/04/Introducing-windows.html)

In [7]:

```
result = spark.sql("""
SELECT * from (
    SELECT
    min(temperature) over w as min_temperature,
    max(temperature) over w as max_temperature,
    min(voltage) over w as min_voltage,
    max(voltage) over w as max_voltage,
    min(flowrate) over w as min_flowrate,
    max(flowrate) over w as max_flowrate,
    min(frequency) over w as min_frequency,
    max(frequency) over w as max_frequency,
    min(hardness) over w as min_hardness,
    max(hardness) over w as max_hardness,
    min(speed) over w as min_speed,
    max(speed) over w as max_speed
    FROM washing
    WINDOW w AS (ORDER BY ts ROWS BETWEEN CURRENT ROW AND 10 FOLLOWING)
)
WHERE min_temperature is not null
AND max_temperature is not null
AND min_voltage is not null
AND max_voltage is not null
AND min_flowrate is not null
AND max_flowrate is not null
AND min_frequency is not null
AND max_frequency is not null
AND min_hardness is not null
AND min_speed is not null
AND max_speed is not null
""")
```

Since this table contains null values also our window might contain them. In case for a certain feature all values in that window are null we obtain also null. As we can see here (in my dataset) this is the case for 9 rows.

In [8]:

```
df.count()-result.count()
```

Out[8]:

7

Now we import some classes from SparkML. PCA for the actual algorithm. Vectors for the data structure expected by PCA and VectorAssembler to transform data into these vector structures.

In [9]:

```
from pyspark.ml.feature import PCA
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
```

Let's define a vector transformation helper class which takes all our input features (result.columns) and created one additional column called "features" which contains all our input features as one single column wrapped in "DenseVector" objects

In [10]:

```
assembler = VectorAssembler(inputCols=result.columns, outputCol="features")
```

Now we actually transform the data, note that this is highly optimized code and runs really fast in contrast if we had implemented it.

In [11]:

```
features = assembler.transform(result)
```

Let's have a look at how this new additional column "features" looks like:

In [12]:

```
features.rdd.map(lambda r : r.features).take(10)
```

Out[12]:

```
[DenseVector([81.0, 100.0, 221.0, 223.0, 11.0, 11.0, 68.0, 76.0, 71.0, 78.
0, 1033.0, 1033.0]),
 DenseVector([81.0, 100.0, 221.0, 223.0, 11.0, 11.0, 68.0, 76.0, 72.0, 78.
0, 1033.0, 1033.0]),
 DenseVector([81.0, 100.0, 221.0, 223.0, 11.0, 11.0, 68.0, 76.0, 72.0, 80.
0, 1033.0, 1033.0]),
 DenseVector([81.0, 100.0, 222.0, 223.0, 11.0, 11.0, 68.0, 74.0, 72.0, 80.
0, 1033.0, 1046.0]),
 DenseVector([81.0, 100.0, 222.0, 223.0, 11.0, 11.0, 68.0, 74.0, 73.0, 80.
0, 1033.0, 1046.0]),
 DenseVector([80.0, 94.0, 222.0, 223.0, 11.0, 11.0, 68.0, 74.0, 73.0, 80.
0, 1033.0, 1046.0]),
 DenseVector([80.0, 94.0, 222.0, 236.0, 11.0, 11.0, 68.0, 74.0, 73.0, 80.
0, 1046.0, 1046.0]),
 DenseVector([80.0, 94.0, 222.0, 236.0, 11.0, 11.0, 68.0, 74.0, 73.0, 80.
0, 1046.0, 1046.0]),
 DenseVector([80.0, 94.0, 222.0, 236.0, 11.0, 11.0, 71.0, 74.0, 73.0, 80.
0, 1046.0, 1046.0]),
 DenseVector([80.0, 94.0, 222.0, 236.0, 11.0, 11.0, 71.0, 74.0, 73.0, 80.
0, 1046.0, 1046.0])]
```

Since the source data set has been prepared as a list of DenseVectors we can now apply PCA. Note that the first line again only prepares the algorithm by finding the transformation matrices (fit method)

In [13]:

```
pca = PCA(k=3, inputCol="features", outputCol="pcaFeatures")
model = pca.fit(features)
```

Now we can actually transform the data. Let's have a look at the first 20 rows

In [14]:

```
result_pca = model.transform(features).select("pcaFeatures")
result_pca.show(truncate=False)
```

```
+-----------------------------------------------------------+
|pcaFeatures                                                |
+-----------------------------------------------------------+
|[1459.9789705814187,-18.745237781780922,70.78430794796873] |
|[1459.995481828676,-19.11343146165273,70.72738871425986]   |
|[1460.0895843561282,-20.969471062922928,70.75630600322052] |
|[1469.6993929419532,-20.403124647615513,62.013569674880955]|
|[1469.7159041892107,-20.771318327487293,61.95665044117209] |
|[1469.7128317338704,-20.790751117222456,61.896106678330966]|
|[1478.3530264572928,-20.294557029728722,71.67550104809607] |
|[1478.3530264572928,-20.294557029728722,71.67550104809607] |
|[1478.3686036138165,-20.260626897636314,71.63355353606426] |
|[1478.3686036138165,-20.260626897636314,71.63355353606426] |
|[1483.5412027684088,-20.006222577501354,66.82710394284209] |
|[1483.5171090223353,-20.867020421583753,66.86707301954084] |
|[1483.4224268542928,-19.87574823665505,66.93027077913985]  |
|[1483.4224268542928,-19.87574823665505,66.93027077913985]  |
|[1488.103073547271,-19.311848573386925,72.1626182636411]   |
|[1488.1076926849646,-19.311945711095063,72.27621605605316] |
|[1488.0135901575127,-17.455906109824838,72.2472987670925]  |
|[1488.026374556614,-17.47632766649086,72.2214703423]       |
|[1465.1644738447062,-17.50333829280811,47.06072898272612]  |
|[1465.1644738447062,-17.50333829280811,47.06072898272612]  |
+-----------------------------------------------------------+
only showing top 20 rows
```

So we obtained three completely new columns which we can plot now. Let run a final check if the number of rows is the same.

In [15]:

```
result_pca.count()
```

Out[15]:

```
2051
```

Cool, this works as expected. Now we obtain a sample and read each of the three columns into a python list

In [16]:

```
rdd = result_pca.rdd.sample(False,0.8)
```

In [17]:

```
x = rdd.map(lambda a : a.pcaFeatures).map(lambda a : a[0]).collect()
```

In [18]:

```
y = rdd.map(lambda a : a.pcaFeatures).map(lambda a : a[1]).collect()
```

In [19]:

```
z = rdd.map(lambda a : a.pcaFeatures).map(lambda a : a[2]).collect()
```

Finally we plot the three lists and name each of them as dimension 1-3 in the plot

In [20]:

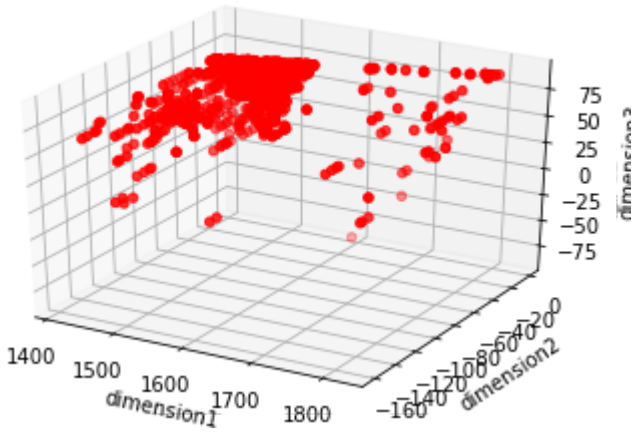
```
%matplotlib inline
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


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

ax.scatter(x,y,z, c='r', marker='o')

ax.set_xlabel('dimension1')
ax.set_ylabel('dimension2')
ax.set_zlabel('dimension3')

plt.show()
```



Congratulations, we are done! We can see two clusters in the data set. We can also see a third cluster which either can be outliers or a real cluster. In the next course we will actually learn how to compute clusters automatically. For now we know that the data indicates that there are two semi-stable states of the machine and sometime we see some anomalies since those data points don't fit into one of the two clusters.