SAE_204_Carte_de_Controle

May 19, 2025

1 Introduction

Dans le cadre de la SAE 2.04, le projet consiste à concevoir et d'exploiter une base de données pour analyser la qualité des mesures issues d'une machine de production de papier, qui est équipée avec de nombreux capteurs réalisant des mesures continue comme la température, l'humidité ou encore la vitesse de rotation des bobines. Afin de garantir la fiabilité de ces capteurs, des mesures manuelles sont également effectuées par des techniciens en laboratoire. L'objectif principal de ce projet est de réaliser des cartes de contrôle permettant de détecter les éventuelles dérives ou défaillances de ces capteurs.

Pour cela, nous avons utilisé des outils tels que SQL et Python dans un environnement Jupyter, afin d'extraire, d'analyser et de visualiser les données.

1.1 Setting global variables.

[3]:
```python
import datetime

import numpy as np
import psycopg
import matplotlib.pyplot as plt
from datetime import date, timedelta
from math import comb

PARAMCONN = {
    "host": "iutinfo-sgbd.uphf.fr",
    "dbname": "capteurs",
    "user": "iutinfo414",
    "password": "BJ6Kv7XT",
    "port": 5432
}

SENSORID = 1  #Constant to select the sensor.
```

2 1. Control Card

Afin de réaliser notre carte de controle nous allons avoir besoin de récuperer la dernière date d'entrée d'un sensor, la différences du sensors et de la prise manuel, récuperer la moyenne et l'écart-type de cette différence et enfin de mettre le tout en valeur en déssinant un graphique.

Nous retrouvons nos quatre fonctions suivante:

getDate()

getDiffSensorLab()

getStdAvg()

plotSensor()

2.0.1 getDate()

Function to get the last sensor measurement date depending on the sensor ID passed as a parameters.

[70]:
```
def getDate(conn, id: int) -> date:
    sql = """
          SELECT controlmeasurement.sensortimestamp
          FROM controlmeasurement
          WHERE controlmeasurement.sensorid = %(id)s
          ORDER BY controlmeasurement.sensortimestamp DESC
          """

    with conn.execute(sql, {"id": id}) as cursor:
        return cursor.fetchone()[0]
```

2.0.2 getDiffSensorLab()

Function to get a tuple of list containing the difference between the manuel and electronic values and the times of the input. The list is from the last 8 days per default and must contain between 20 and 100 values.

[71]:
```
def getDiffSensorLab(conn, id: int, interval: int = 8) -> tuple[list[float],␣
 ↪list[datetime.datetime]]:
    sql = """
          SELECT controlvalue - sensorvalue as error, timestamp
          FROM controlmeasurement
                   JOIN sensormeasurement on sensormeasurement.sensorid =␣
 ↪controlmeasurement.sensorid and
                                             sensormeasurement.timestamp =␣
 ↪controlmeasurement.sensortimestamp
          WHERE controlmeasurement.controltimestamp > %(dateD)s
            AND controlmeasurement.controltimestamp < %(dateF)s
            AND controlmeasurement.sensorid = %(id)s
            AND controlmeasurement.controllerid = %(id)s
          ORDER BY timestamp DESC
          LIMIT 100
          """

    param = {
```

```
        "dateD": getDate(conn, id) - timedelta(days=interval),
        "dateF": getDate(conn, id),
        "id": id
    }

    valuesList: list[float] = []
    dateList: list[datetime.datetime] = []
    with conn.execute(sql, param) as cursor:
        for row in cursor:
            valuesList.append(row[0])
            dateList.append(row[1])

    if len(valuesList) < 20:
        return getDiffSensorLab(conn, id, interval * 1.5)
    else:
        return valuesList, dateList
```

2.0.3 getEcartAvg()

Return the average and the standard deviation of the difference between the sensor values and the control values of a sensor.

[72]:
```
def getStdAvg(conn, id: int) -> (float, float):
    sql = """
          SELECT stddev(sensorvalue - controlvalue) AS ecart, avg(sensorvalue -␣
 ↪controlvalue) AS avg
          FROM controlmeasurement
                   JOIN sensormeasurement on sensormeasurement.sensorid =␣
 ↪controlmeasurement.sensorid and
                                                sensormeasurement.timestamp =␣
 ↪controlmeasurement.sensortimestamp
          WHERE sensormeasurement.sensorid = %(id)s
            and controlmeasurement.sensorid = %(id)s
          """

    with conn.execute(sql, {"id": id}) as cursor:
        return cursor.fetchone()
```

2.0.4 plotSensor()

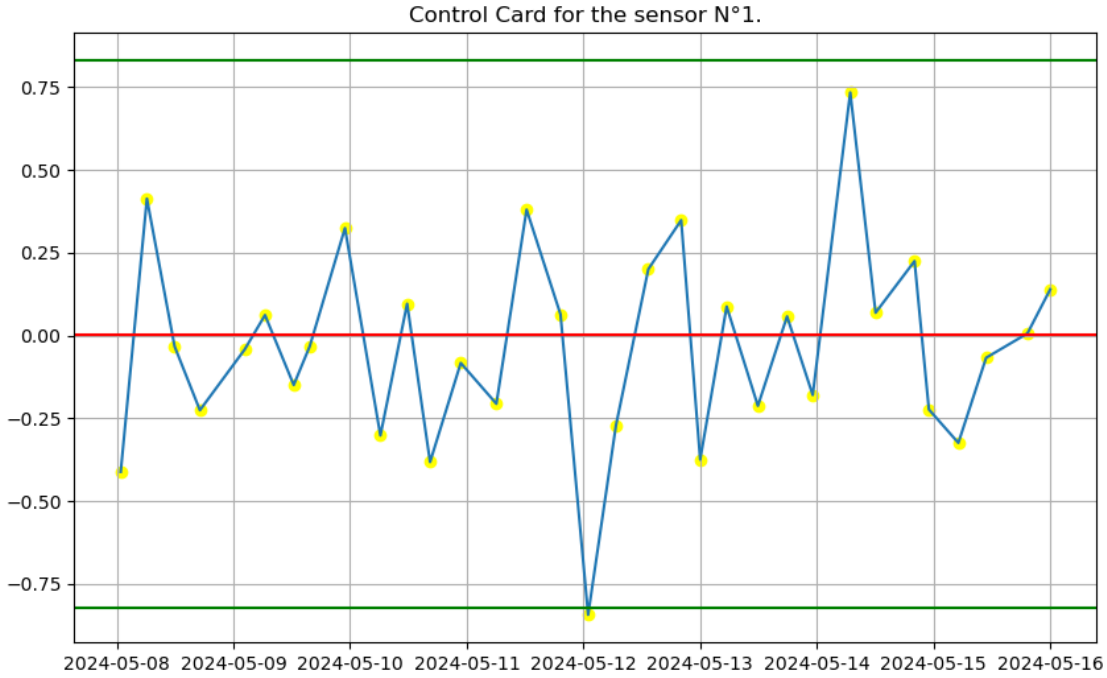
Plot a graph to visualize the values with the average of the sensor and his standard deviation. This is the visualisation of our control card.

[73]:
```
def plotSensor(data: tuple, average: float, stddev: float) -> None:
    #Figures settings.
    plt.figure(figsize=(10, 6))
    plt.grid(True)
    plt.xticks(rotation=0)
```

```
    plt.title("Control Card for the sensor N°" + str(SENSORID) + ".")

    #Lines draws.
    plt.plot(data[1], data[0])
    plt.scatter(data[1], data[0], color='yellow')
    plt.axhline(average, color='r')
    plt.axhline(average - 2 * stddev, color='g')
    plt.axhline(average + 2 * stddev, color='g')

    plt.show()
```

We are now calling our different functions to get our control card.

[74]:
```
with psycopg.connect(**PARAMCONN) as conn:
    std, avg = getStdAvg(conn, SENSORID)
    values = getDiffSensorLab(conn, SENSORID)
    plotSensor(values, avg, std)
```



3 2. Statistics

3.0.1 2.1 Visualisation des marges d'erreur

Nous avons besoin de récuperer toutes les valeurs d'une comparaison de sensor afin de pouvoir calculer nos marges d'erreur. Il nous suffiras ensuite de les afficher dans un graphe. Nous avons donc les deux fonctions suivantes:

getAllDifValues()

plotNormalDistriPython()

3.0.2 getAllDifValues()

Return all the values of the comparaison between control values and sensor values.

[75]:
```
def getAllDifValues(conn, id: int) -> list[float]:
    sql = """
          SELECT (controlvalue - sensorvalue) as error
          FROM controlmeasurement
                   JOIN sensormeasurement on sensormeasurement.sensorid =␣
 ↪controlmeasurement.sensorid and
                                             sensormeasurement.timestamp =
                                             controlmeasurement.sensortimestamp
          WHERE sensormeasurement.sensorid = %(id)s
            and controlmeasurement.sensorid = %(id)s
          ORDER BY controlmeasurement.sensortimestamp
          """

    output = []

    with conn.execute(sql, {"id": id}) as cursor:
        for row in cursor:
            output.append(row[0])

    return output
```

3.0.3 plotNormalDistriPython()

Draw a graph comparing our data distribution with an normal distribution to showcase our margin of errors.

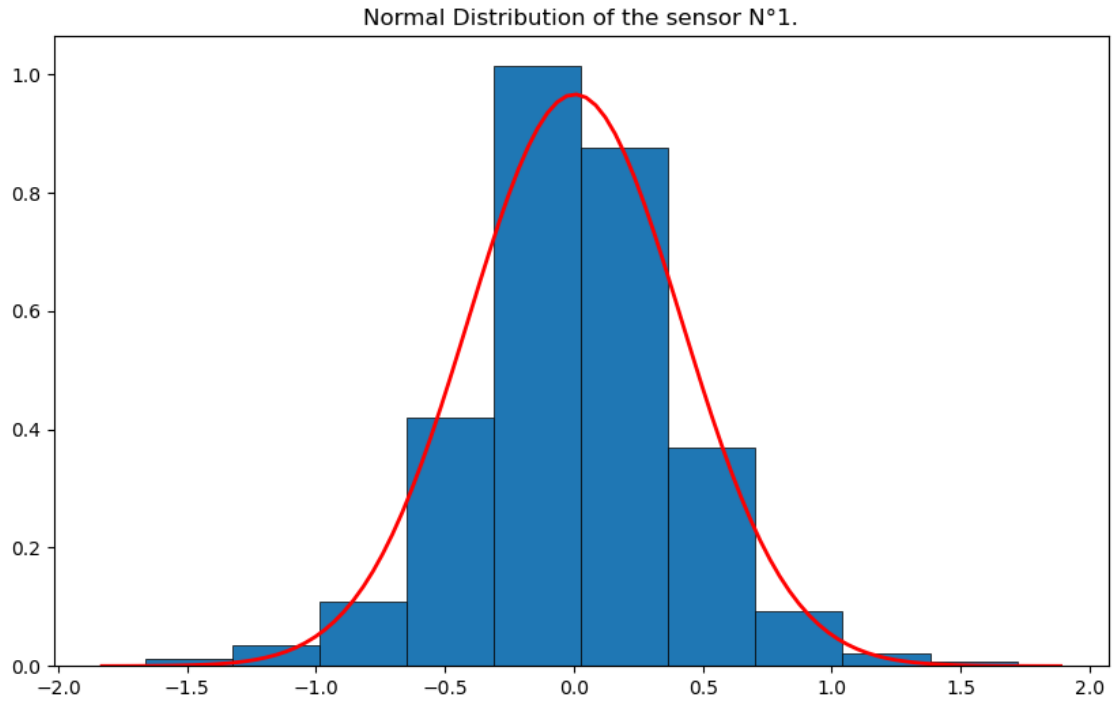
[76]:
```
def plotNormalDistriPython(data: list[float], avg: float, std: float) -> None:
    #Figures settings.
    plt.figure(figsize=(10, 6))
    plt.xticks(rotation=0)
    plt.title("Normal Distribution of the sensor N°" + str(SENSORID) + ".")

    plt.hist(data, density=True, bins=10, linewidth=0.5, edgecolor='black')

    # Sorting each value into our intervals.
    inter1, inter2, inter3 = [], [], []
    for val in data:
        if val < std and val > -std:
            inter1.append(val)
        if val < 2 * std and val > 2 * -std:
            inter2.append(val)
```

```
        if val < 3 * std and val > 3 * -std:
            inter3.append(val)

    #Printing the percentages of each interval.
    print("Percentage of our data data between the each interval of our sensor␣
↪N°" + str(SENSORID) + ".")
    print('[- , +] = ' + str(round(len(inter1) / len(data) * 100, 2)) + "%")
    print('[-2 , +2] = ' + str(round(len(inter2) / len(data) * 100, 2)) + "%")
    print('[-3 , +3] = ' + str(round(len(inter3) / len(data) * 100, 2)) + "%")
    print('[ϖ, -3 [ & ]+3 , ϖ] = ' + str(round(100 - len(inter3) / len(data) *␣
↪100, 2)) + "%")

    #Ploting the normal distribution.
    xmin, xmax = plt.xlim()
    x = np.linspace(xmin, xmax, 100)
    #Calculating the probability density with numpy.
    p = (1 / (std * np.sqrt(2 * np.pi))) * np.exp(-0.5 * ((x - avg) / std) ** 2)
    plt.plot(x, p, linewidth=2, color='r')

    plt.show()
```

We can now combine ours function to draw our graph showcasing the margin of error of our sensor.

[77]:
```
with psycopg.connect(**PARAMCONN) as conn:
    std, avg = getStdAvg(conn, SENSORID)
    data = getAllDifValues(conn, SENSORID)
    plotNormalDistriPython(data, avg, std)
```

```
Percentage of our data data between the each interval of our sensor N°1.
[- , +] = 71.37%
[-2 , +2] = 94.91%
[-3 , +3] = 98.95%
[ϖ, -3 [ & ]+3 , ϖ] = 1.05%
```



Nous pouvons voir ainsi que pour un sensor donnée, les distributions des valeurs suivent la loi binomiale. Même si un écart d'environ 3,3% est présent pour le sensor numéro 1 par rapport à l'interval [- , +] peut laisser penser à une erreur des capteurs dans la prise de nos données, ce n'est pas le cas. Nous avons donc testé nos fonctions avec des données générées aléatoirement grâce à la fonction np.random.normal() de numpy afin de s'assurer le bon fonctionnement de notre carte de contrôle.

3.0.4 2.2 P.Valeur

Soit la probabilité qu 'au moins 12 des 15 derniers points, sachant que la probabilité qu'un point se situe au-dessus ou en dessous de la moyenne est p = 0,5. Soit la loi binomiale:

$$P(X \geq k) = \binom{n}{k} p^k (1-p)^{n-k}$$

Donc:

$$P(X \geq 12) = 1 - P(X \leq 12)$$

$$P(X \geq 12) = P(X = 12) + P(X = 13) + P(X = 14) + P(X = 15)$$

$$P(X = 12) = \binom{15}{12} 0.5^{12} (0.5)^{15-12} = 455 * 0.5^{15}$$

$$P(X = 13) = \binom{15}{13} 0.5^{13} (0.5)^{15-13} = 105 * 0.5^{15}$$

$$P(X = 14) = \binom{15}{14} 0.5^{14} (0.5)^{15-14} = 15 * 0.5^{15}$$

$$P(X = 15) = \binom{15}{15} 0.5^{15} (0.5)^{15-15} = 1 * 0.5^{15}$$

Sachant que:

$$0.5^{15} = \frac{1}{32769}$$

Alors:

$$P(X \geq 12) = \frac{455 + 105 + 15 + 1}{32769} = \frac{576}{32769} \approx 0.0176$$

Enfin:

$$0.0176 * 100 \approx 1.76\%$$

[4]:
```python
#Return the p-value.
def pValue(k: int, n: int, p: float):
    q = 1 - p
    result = []
    for i in range(k, n + 1):
        result.append(comb(n, i) * (p ** i) * (q ** (n - i)))
    final = sum(result)
    print(f"Chance to happen: {(final * 100)}%")
    return round(final * 100, 2)

print(pValue(12, 15, 0.5))
```

```
Chance to happen: 1.7578125%
1.76
```

Pour obtenir la probabilité d'au moins 15 points sur les 20 derniers nous allons procéder de la même manière que la première probabilité : Donc :

$$P(X \geq 15) = P(X = 15) + P(X = 16) + P(X = 17) + P(X = 18) + P(X = 19) + P(X = 20)$$

Ce qui nous donne :

$$P(X = 15) = \binom{20}{15} 0.5^{15} (0.5)^{20-15} = 15504 * 0.5^{20}$$

$$P(X = 16) = \binom{20}{16} 0.5^{16} (0.5)^{20-16} = 4845 * 0.5^{20}$$

$$P(X = 17) = \binom{20}{17} 0.5^{17} (0.5)^{20-17} = 1140 * 0.5^{20}$$

$$P(X = 18) = \binom{20}{18} 0.5^{18} (0.5)^{20-18} = 190 * 0.5^{20}$$

$$P(X = 19) = \binom{20}{19} 0.5^{19} (0.5)^{20-19} = 20 * 0.5^{20}$$

$$P(X = 20) = \binom{20}{20} 0.5^{20} (0.5)^{20-20} = 1 * 0.5^{20}$$

Sachant que :

$$0.5^{20} = \frac{1}{1048576}$$

Alors :

$$P(X \geq 15) = \frac{15504 + 4845 + 1140 + 190 + 20 + 1}{1048576} = \frac{21700}{1048576} \approx 0.0207$$

Enfin :

$$0.0207 * 100 \approx 2.07\%$$

[79]:
```
print(pValue(15, 20, 0.5))
```

```
Chance to happen: 2.0694732666015625%
2.07
```

Pour déterminer la probabilité d'au moins 8 points sur les 10 derniers du même côte de la moyenne, nous allons faire :

$$P(X \geq 8) = P(X = 8) + P(X = 9) + P(X = 10)$$

Donc :

$$P(X = 8) = \binom{10}{8} 0.5^8 (0.5)^{10-8} = 45 * 0.5^{10}$$

$$P(X = 9) = \binom{10}{9} 0.5^9 (0.5)^{10-9} = 10 * 0.5^{10}$$

$$P(X = 10) = \binom{10}{10} 0.5^{10} (0.5)^{10-10} = 1 * 0.5^{10}$$

Sachant que :

$$0.5^{10} = \frac{1}{1024}$$

Alors :

$$P(X \geq 8) = \frac{45 + 10 + 1}{1024} = \frac{56}{1024} \approx 0.0547$$

Enfin :

$$0.0547 * 100 \approx 5.47\%$$

[80]:
```
print(pValue(8, 10, 0.5))
```

```
Chance to happen: 5.46875%
5.47
```

Pour déterminer la p-valeur d'au moins 2 valeurs sur les 7 dernières valeures dans l'intervale [−2 , +2], nous allons procéder de la même manière avec la formule :

$$P(X \geq k) = \binom{n}{k} p^k (1-p)^{n-k}$$

Nous allons venir tout d'abord déterminer notre p. Nous savons que 95% des valeurs se trouve dans l'intervalle [−2 , +2] donc pour calculer le p des valeurs en dehors de cette intervalle nous allons faire :

$$p = 1 - 95\%$$

$$p = 1 - 0.95$$

$$p = 0.05$$

Donc pour calculer les 2 des 7 derniers nous faisons :

$$P(X \geq 2) = P(X = 2) + P(X = 3) + P(X = 4) + P(X = 5) + P(X = 6) + P(X = 7)$$

Cependant, nous allons utiliser le complémentaire pour rendre le calcul plus facile pour nous.

$$P(X \geq 2) = 1 - [P(X = 0) + P(X = 1)]$$

Donc, nous allons venir calculer P(X=0) et P(X=1) :

$$P(X = 0) = \binom{7}{0} 0.05^0 (1 - 0.05)^{7-0}$$

$$P(X = 1) = \binom{7}{1} 0.05^1 (1 - 0.05)^{7-1}$$

Cela nous donne :

$$P(X = 0) \approx 0.6983$$

$$P(X = 1) \approx 0.2573$$

Enfin :

$$P(X \geq 2) = 1 - [P(X = 0) + P(X = 1)]$$

$$P(X \geq 2) = 1 - [0.6983 + 0.2573]$$

$$P(X \geq 2) \approx 0.04438$$

Donc :

$$P(X \geq 2) \approx 4.44\%$$

[81]:
```
print(pValue(2, 7, 0.05))
```

```
Chance to happen: 4.43805421875%
4.44
```

Pour déterminer la p-valeur d'au moins 7 valeurs sur les 10 dernières valeures dans l'intervale [− , +], nous allons procéder de la mẽme manière avec la formule :

$$P(X \geq k) = \binom{n}{k} p^k (1 - p)^{n-k}$$

Nous allons venir tout d'abord déterminer notre p. Nous savons que 95% des valeurs se trouve dans l'intervalle [− , +] donc pour calculer le p des valeurs en dehors de cette intervalle nous allons faire :

$$p = 1 - 68\%$$

$$p = 1 - 0.68$$

$$p = 0.32$$

Donc pour calculer les 7 des 10 derniers nous faisons :

$$P(X \geq 7) = P(X = 7) + P(X = 8) + P(X = 9) + P(X = 10)$$

Donc :

$$P(X = 7) = \binom{10}{7} 0.32^7 (1 - 0.32)^{10-7} \approx 0.01296$$

$$P(X = 8) = \binom{10}{8} 0.32^8 (1 - 0.32)^{10-8} \approx 0.002288$$

$$P(X = 9) = \binom{10}{9} 0.32^9 (1 - 0.32)^{10-9} \approx 0.0002393$$

$$P(X = 10) = \binom{10}{10} 0.32^{10} (1 - 0.32)^{10-10} \approx 1.1259e - 05$$

Cela nous donne :

$$P(X \geq 7) = P(X = 7) + P(X = 8) + P(X = 9) + P(X = 10)$$

$$P(X \geq 7) = 0.01296 + 0.002288 + 0.0002393 + 1.1259e - 05$$

$$P(X \geq 7) \approx 0.0155$$

Donc :

$$P(X \geq 7) \approx 1.55\%$$

[82]:
```
print(pValue(7, 10, 0.32))
```

```
Chance to happen: 1.5502938029781155%
1.55
```

Pour déterminer la règle original, nous avons calculer la p-valeur entre les intervalles $[-\ ,\ +\ ]$, avec 5 succès sur les 10 dernières mesures. Pour cela nous avons effectuer de la manière suivante avec la formule de la loi binomiale :

$$P(X \geq k) = \binom{n}{k} p^k (1 - p)^{n-k}$$

Tout d'abord, nous allons déterminer le p. Pour cela, nous faisons :

$$p = 1 - 68$$

$$p = 0.32$$

Donc pour un seul coté nous allons faire :

$$p = \frac{0,32}{2} = 0,16$$

Et nous calculons les 5 des 10 derniers :

$$P(X \geq 5) = P(X = 7) + P(X = 8) + P(X = 9) + P(X = 10)$$

Donc :

$$P(X = 5) = \binom{10}{5} 0.16^5 (1 - 0.16)^{10-5} \approx 0.0111$$

$$P(X = 6) = \binom{10}{6} 0.16^6 (1 - 0.16)^{10-6} \approx 0.00175$$

$$P(X = 7) = \binom{10}{7} 0.16^7 (1 - 0.16)^{10-7} \approx 0.00019$$

$$P(X = 8) = \binom{10}{8} 0.16^8 (1 - 0.16)^{10-8} \approx 0.000013$$

$$P(X = 9) = \binom{10}{9} 0.16^9 (1 - 0.16)^{10-9} \approx 0.0000005$$

$$P(X = 10) = \binom{10}{10} 0.16^{10} (1 - 0.16)^{10-10} \approx 0.00000001$$

Cela nous donne :

$$P(X \geq 5) = P(X = 5) + P(X = 6) + P(X = 7) + P(X = 8) + P(X = 9) + P(X = 10)$$

$$P(X \geq 5) = 0.0111 + 0.00175 + 0.00019 + 0.000013 + 0.0000005 + 0.00000001$$

$$P(X \geq 5) \approx 0.013$$

Donc : Cette probabilité est la probabilité que 5 points sur 10 soit en dehors de l'intervalle et du même côté comme nous respectons la loi normale, pour la probabilité d'avoir au moins 5 points en dehors de l'intervalle nous additionnons les probabilités pour chaque côté.

Ce qui nous donne :

$$P(X \geq 5) \approx 1.3 + 1.3 \approx 2.6$$

[5]:
```
print(pValue(5, 10, 0.32))
```

```
Chance to happen: 18.66554070288407%
18.67
```

Afin de vérifier que les données prisent par nos capteurs ne sont pas improbable, nous allons implémenter trois fonctions python qui correspondent aux règles suivantes : • au moins 8 points sur les 10 derniers du même côté de la moyenne, -> checkSameSideAvg() • au moins 2 points sur les 7 derniers en dehors de l'intervalle [−2 , +2], -> checkOutsideTwoStd() • au moins 7 points sur les 10 derniers en dehors de [− , +]. -> checkOutsideStd()

3.0.5 checkSameSideAvg()

Return true if the last 10 values of the sensor have 8 or more values from the same side of the average.

[83]:
```
def checkSameSideAvg(avg: float):
    with psycopg.connect(**PARAMCONN) as conn:
        data = getDiffSensorLab(conn, SENSORID)
    last10Value = data[0][:10]
    over = 0
    under = 0
    for i in range(0, len(last10Value)):
        if last10Value[i] > avg:
            over += 1
        else:
```

```
            under += 1

    if under >= 8 or over >= 8:
        return True
    else:
        return False


# Checking each sensor to see if they trigger the rule.
for i in range(1, 8):
    sensor = i
    with psycopg.connect(**PARAMCONN) as conn:
        std, avg = getStdAvg(conn, sensor)
        if checkSameSideAvg(avg):
            print("Sensor N°", sensor, "triggered.")
```

3.0.6 checkOutsiteTwoStd()

Return true if the last 7 values of sensor have 2 or more values outside [-2 , +2]:

[84]:
```
def checkOutsideTwoStd(std: float):
    with psycopg.connect(**PARAMCONN) as conn:
        data = getDiffSensorLab(conn, SENSORID)
    last7Value = data[0][:7]
    outside = 0
    for i in range(0, len(last7Value)):
        if last7Value[i] > 2 * std:
            outside += 1
        if last7Value[i] < -2 * std:
            outside += 1

    if outside >= 2:
        return True
    else:
        return False


# Checking each sensor to see if they trigger the rule.
for i in range(1, 8):
    sensor = i
    with psycopg.connect(**PARAMCONN) as conn:
        std, avg = getStdAvg(conn, sensor)
        if checkOutsideTwoStd(std):
            print("Sensor N°", sensor, "triggered.")
```

3.0.7 checkOutsiteStd()

Return true if the last 10 values of sensor have 7 or more values outside [- , +]:

[85]:
```python
def checkOutsideStd(std: float):
    with psycopg.connect(**PARAMCONN) as conn:
        data = getDiffSensorLab(conn, SENSORID)
    last10Value = data[0][:10]
    outside = 0
    for i in range(0, len(last10Value)):
        if last10Value[i] > std:
            outside += 1
        if last10Value[i] < -std:
            outside += 1

    if outside >= 7:
        return True
    else:
        return False


# Checking each sensor to see if they trigger the rule.
for i in range(1, 8):
    sensor = i
    with psycopg.connect(**PARAMCONN) as conn:
        std, avg = getStdAvg(conn, sensor)
        if checkOutsideStd(std):
            print("Sensor N°", sensor, "triggered.")
```

Durant nos différents tests, la règle numéro 2 implémenté avec la checkOutsideTwoStd() fût déclenché par le sensor numéro 6. Nous avons vérifié auprès d'étudiants de différent groupe et ils ont, eux aussi, ont u cette même activation de la règle. Cependant, en fonction du jour où la base de données est testé cette règle ne se déclenche pas forcément.

3.0.8 2.3 Dérive et régression

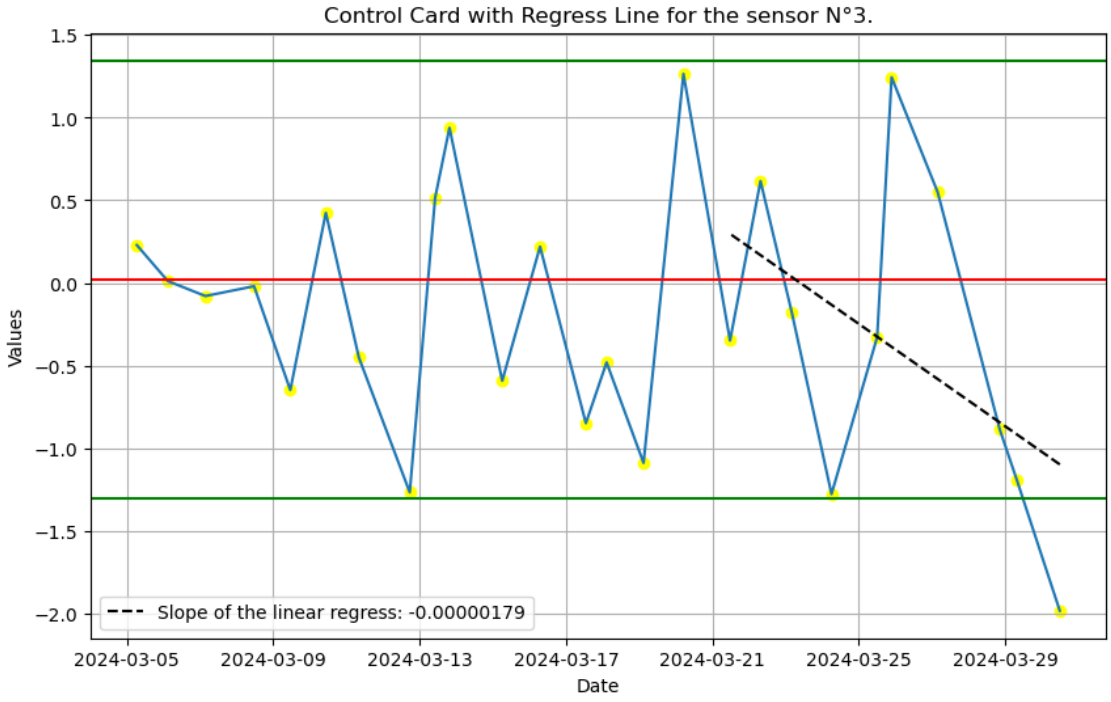
Afin de pouvoir visualiser lorsqu'un capteur commence à être défaillant, nous mettons en place une règle qui affiche la droite de régréssion de ce capteur lorsuq'une anomalie est détecté.

plotLinearRegres() This function shows the linear regression line of our data, then extracts the slope of the directing coefficient of the regression line and analyses whether it exceeds the threshold.

[86]:
```python
def plotLinearRegres(data: tuple, average: float, stddev: float, id) -> None:
    #Figures settings.
    plt.figure(figsize=(10, 6))
    plt.grid(True)
    plt.xticks(rotation=0)
    plt.title("Control Card with Regress Line for the sensor N°" + str(id) + ".
 ↪")

    #Lines draws.
    plt.plot(data[1], data[0])
```

```
    plt.scatter(data[1], data[0], color='yellow')
    plt.axhline(average, color='r')
    plt.axhline(average - 2 * stddev, color='g')
    plt.axhline(average + 2 * stddev, color='g')

    #Getting the last 10 values
    last10Date = data[1][:10]
    last10Value = data[0][:10]

    #Tranforming the timedate to float.
    timeFloat = last10Date.copy()
    for i in range(len(last10Date)):
        timeFloat[i] = last10Date[i].timestamp()

    #Calculating the linear regression
    coef = np.polyfit(timeFloat, last10Value, 1)
    poly1d_fn = np.poly1d(coef)
    slope = coef[0]  # Extract the slope (coefficient directeur)

    #Ploting the regress line
    plt.plot(last10Date, poly1d_fn(timeFloat), '--k', label=f'Slope of the␣
 ↪linear regress: {slope:.8f}')
    plt.legend()
    plt.xlabel("Date") # Customize your labels as needed
    plt.ylabel("Values")

    #Showing the graph only when an error is found.
    if checker(slope):
        print("Anomaly Detected from the sensor N°" + str(id))
        plt.show()
    else:
        plt.close()

# Return true if the slope is between 0.0000015 and -0.0000015.
def checker(slope: float):
        if slope >0.0000015 or slope<-0.0000015:
            return True
        return False

for i in range(1, 8):
    sensor = i
    with psycopg.connect(**PARAMCONN) as conn:
        std, avg = getStdAvg(conn, sensor)
        values = getDiffSensorLab(conn, sensor)
        plotLinearRegres(values, avg, std, sensor)
```

```
Anomaly Detected from the sensor N°3
```



4 3 Base de données

Afin que l'usine puisse avoir des statistiques sur la fiabilité des capteurs. À chaque fois qu'un capteur est mis en service, réétalonné ou mis au rebut, nous allons enregistrer ses informations dans la base de données. Pour ce faire, nous devons réaliser une extension de celle-ci.

Voici le model relation étendu de notre base de données :

Voici le code SQL permettant d'implémenter dans la base de donnée notre extension.

[ ]:
```
"""
DROP TABLE IF EXISTS Calibrate;

CREATE TABLE Calibrate
(
    sensorID serial references sensor not null,
    date timestamp not null,
    primary key (sensorID, date)
);

ALTER TABLE sensor ADD COLUMN dateInstalled timestamp;
ALTER TABLE sensor ADD COLUMN dateRemoved timestamp;
"""
```

Une fois implémenté, nous devons ajouter à tous nos sensor en fonctionnement leur date de mise

en place (leur première timestamp). De plus, nous partons du principe que chaque sensor dans la BDD n'est plus actif. Nous allons ainsi leur mettre une fin de sensor à la date de leur dernière mesure.

[88]:
```python
def setupExtension(conn):
    sql = """
          WITH last AS
                   (SELECT sensorid, timestamp
                    FROM sensormeasurement
                    WHERE sensorid = %(id)s
                    ORDER BY timestamp DESC
                    LIMIT 1)
          SELECT sensormeasurement.timestamp as first, last.timestamp as last
          FROM sensormeasurement
                   JOIN last ON sensormeasurement.sensorid = last.sensorid
          WHERE sensormeasurement.sensorid = %(id)s
          ORDER BY sensormeasurement.timestamp
          LIMIT 1; \
          """

    dates: list[tuple[datetime, datetime]] = []

    for i in range(1, 8):
        with conn.execute(sql, {"id": i}) as cursor:
            for row in cursor:
                dates.append((row[0], row[1]))

    sql = """
          UPDATE sensor
          SET dateInstalled = %(dI)s,
              dateRemoved   = %(dR)s
          WHERE sensorid = %(id)s \
          """

    for i in range(1, 8):
        params = {
            "id": i,
            "dI": dates[i - 1][0],
            "dR": dates[i - 1][1],
        }
        with conn.execute(sql, params) as cursor:
            pass

#with psycopg.connect(**PARAMCONN) as conn:
#    setupExtension(conn)
```

Pour nos different tests, nous allons devoir changer la base de données afin de pouvoir tester notre extension.

[ ]:
```
"""
UPDATE sensor
SET dateRemoved = '2023-10-02 00:00:00'
WHERE sensorid = 5;

INSERT INTO sensor
VALUES (8, 'Wet press whiteness', INTERVAL '10' MINUTE, 14, 564789134,␣
 ↪'2023-01-01 00:00:31.019859',
        '2024-05-15 23:59:59.225586');
INSERT INTO sensor
VALUES (9, 'Headbox inductive linear', INTERVAL '10' MINUTE, 1, 56474,␣
 ↪'2023-01-01 00:00:31.019859',
        '2023-01-18 16:59:59.225586');

INSERT INTO calibrate (sensorID, date)
VALUES (1, '2024-01-13 14:00:00'),
       (1, '2024-03-13 14:00:00'),
       (1, '2023-06-13 14:00:00'),
       (1, '2023-03-24 14:00:00'),
       (2, '2023-06-13 14:00:00'),
       (2, '2023-01-10 14:00:00'),
       (3, '2023-06-13 14:00:00'),
       (4, '2023-06-13 14:00:00'),
       (5, '2023-07-13 14:00:00'),
       (6, '2023-06-13 14:00:00'),
       (7, '2023-06-13 14:00:00'),
       (8, '2023-06-13 14:00:00'),
       (9, '2023-06-13 14:00:00');

"""
```

3.1. Le nombre moyen de réétalonnages par an. (déduisez-en le temps moyen entre deux réétalonages) Pour le modèle 2,3,4,7,8,9,11,13. Pas possible, vu qu'il n'y a pas de capteur. Pour le modèle 1, 10, 12, 15 et 16 -> 163 jours entre le réétalonnages des capteurs. Pour le modèle 14 -> 96 jours Pour le modèle 5 -> 109 jours. Pour le modèle 6 -> 81 jours.

[90]:
```
### Show the average between each sensor calibration.
def avgCalibrateYear(conn):
    sql = """
          SELECT modelId, calibrate.date, sensor.dateInstalled
          FROM model
                   LEFT JOIN Sensor using (modelid)
                   LEFT JOIN calibrate using (sensorId)
            GROUP BY modelId, calibrate.date, sensor.dateInstalled
          ORDER BY modelId, calibrate.date; \
          """
```

```
    output = []
    with conn.execute("""SELECT * FROM model""") as cursor:
        output = [[] for _ in cursor]

    with conn.execute(sql) as cursor:
        for row in cursor:
            if row[1] is not None:
                output[row[0]-1].append((row[1], row[2]))

    for i in range(0,len(output)):
        if len(output[i]) == 0:
            print("Model N°", i+1, "doesn't have enough data to work with.")
        elif len(output[i]) == 1:
            diff = output[i][0][0] - output[i][0][1]
            print("Model N°", i+1, "need to be calibrated in average every",␣
↪diff.days, "days.")
        else:
            diff = [output[i][0][0] - output[i][0][1]]
            for k in range(1, len(output[i])):
                diff.append(output[i][k][0] - output[i][k-1][0])
            for k in range(len(diff)):
                diff[k] = diff[k].days
            print("Model N°", i+1, "need to be calibrated in average every", np.
↪mean(diff), "days.")

with psycopg.connect(**PARAMCONN) as conn:
    avgCalibrateYear(conn)
```

```
Model N° 1 need to be calibrated in average every 163 days.
Model N° 2 doesn't have enough data to work with.
Model N° 3 doesn't have enough data to work with.
Model N° 4 doesn't have enough data to work with.
Model N° 5 need to be calibrated in average every 109.25 days.
Model N° 6 need to be calibrated in average every 81.5 days.
Model N° 7 doesn't have enough data to work with.
Model N° 8 doesn't have enough data to work with.
Model N° 9 doesn't have enough data to work with.
Model N° 10 need to be calibrated in average every 163 days.
Model N° 11 doesn't have enough data to work with.
Model N° 12 need to be calibrated in average every 163 days.
Model N° 13 doesn't have enough data to work with.
Model N° 14 need to be calibrated in average every 96.5 days.
Model N° 15 need to be calibrated in average every 163 days.
Model N° 16 need to be calibrated in average every 163 days.
```

3.2 La proportion de ceux qui n'ont pas été mis au rebut un an après leur mise en service. (taux de survie à un an) Pour le modèle 2,3,4,7,8,9,11,13. Pas possible, vu qu'il

n'y a pas de capteur. Pour le modèle 5,6,10,12,15,16 -> 100% Pour le modèle 14 -> 50% Pour le modèle 1 -> 0%

[91]:
```
#Return the rate of sensor still in service after one year for a model.
def longivityRate(conn) -> list:
    sql = """
          WITH valid as (SELECT sensorid, modelid
                         FROM sensor
                         WHERE dateRemoved > dateInstalled + INTERVAL '1' YEAR),
               total as (SELECT sensor.modelid, count(*) as total
                         FROM sensor
                         GROUP BY sensor.modelid)
          SELECT model.modelid, count(valid.sensorid) as nbUnvalied,␣
 ↪COALESCE(total.total, 0) as total
          FROM model
                   LEFT JOIN total ON model.modelid = total.modelid
                   LEFT JOIN valid ON model.modelid = valid.modelid
          GROUP BY model.modelid, total.total
          ORDER BY model.modelid \
          """
    output = []
    with conn.execute(sql) as cursor:
        for row in cursor:
            output.append((row[0], row[1], row[2]))
    return output


with psycopg.connect(**PARAMCONN) as conn:
    for elem in longivityRate(conn):
        rate = "Rate of sensor still in service after one year for the model␣
 ↪n°" + str(elem[0]) + " is "
        if (elem[2] != 0):
            rate += "of " + str(elem[1] / elem[2] * 100) + "%."
        else:
            rate += "not possible as no sensor has been installed."
        print(rate)
```

```
Rate of sensor still in service after one year for the model n°1 is of 0.0%.
Rate of sensor still in service after one year for the model n°2 is not possible
as no sensor has been installed.
Rate of sensor still in service after one year for the model n°3 is not possible
as no sensor has been installed.
Rate of sensor still in service after one year for the model n°4 is not possible
as no sensor has been installed.
Rate of sensor still in service after one year for the model n°5 is of 100.0%.
Rate of sensor still in service after one year for the model n°6 is of 100.0%.
Rate of sensor still in service after one year for the model n°7 is not possible
as no sensor has been installed.
```

```
Rate of sensor still in service after one year for the model n°8 is not possible
as no sensor has been installed.
Rate of sensor still in service after one year for the model n°9 is not possible
as no sensor has been installed.
Rate of sensor still in service after one year for the model n°10 is of 100.0%.
Rate of sensor still in service after one year for the model n°11 is not
possible as no sensor has been installed.
Rate of sensor still in service after one year for the model n°12 is of 100.0%.
Rate of sensor still in service after one year for the model n°13 is not
possible as no sensor has been installed.
Rate of sensor still in service after one year for the model n°14 is of 50.0%.
Rate of sensor still in service after one year for the model n°15 is of 100.0%.
Rate of sensor still in service after one year for the model n°16 is of 100.0%.
```

3.3 La proportion de ceux qui n'ont ni été mis au rebut, ni réétalonnés un mois après leur mise en service. Pour le modèle 2,3,4,7,8,9,11,13. Pas possible, vu qu'il n'y a pas de capteur. Pour le modèle 5,10,12,14,15,16 -> 100% Pour le modèle 1 et 6 -> 0%

[92]:
```
# Return the rate of sensors for a model that weren't calibrated or removed␣
 ↪after the first month of use.
def manufacturingDefect(conn):
    sql = """
          WITH calibrated as (SELECT calibrate.sensorid, sensor.modelid
                              FROM calibrate
                                       JOIN sensor using (sensorid)
                              WHERE calibrate.date < sensor.dateInstalled +␣
 ↪INTERVAL '1' MONTH
                              GROUP BY calibrate.sensorid, sensor.modelid),
               removed as (SELECT sensorid, modelid
                           FROM sensor
                           WHERE dateRemoved < dateInstalled + INTERVAL '1'␣
 ↪MONTH),
               total as (SELECT sensor.modelid, count(*) as total
                         FROM sensor
                         GROUP BY sensor.modelid)
          SELECT model.modelid,
                 count(removed.sensorid) + count(calibrated.sensorid) as␣
 ↪nbUnvalided,
                 COALESCE(total.total, 0) as total
          FROM model
                   LEFT JOIN total ON model.modelid = total.modelid
                   LEFT JOIN removed ON model.modelid = removed.modelid
                   LEFT JOIN calibrated ON model.modelid = calibrated.modelid
          GROUP BY model.modelid, removed.sensorid, calibrated.sensorid, total.
 ↪total
          ORDER BY model.modelid \
          """
```

```
    output = []
    with conn.execute(sql) as cursor:
        for row in cursor:
            output.append((row[0], row[1], row[2]))
    return output


with psycopg.connect(**PARAMCONN) as conn:
    for elem in manufacturingDefect(conn):
        rate = "Rate of sensor without deflect after one month for the model␣
 ↪n°" + str(elem[0]) + " is "
        if (elem[2] != 0):
            rate += "of " + str((elem[2] - elem[1]) / elem[2] * 100) + "%."
        else:
            rate += "not possible as no sensor has been installed."
        print(rate)
```

```
Rate of sensor without deflect after one month for the model n°1 is of 0.0%.
Rate of sensor without deflect after one month for the model n°2 is not possible
as no sensor has been installed.
Rate of sensor without deflect after one month for the model n°3 is not possible
as no sensor has been installed.
Rate of sensor without deflect after one month for the model n°4 is not possible
as no sensor has been installed.
Rate of sensor without deflect after one month for the model n°5 is of 100.0%.
Rate of sensor without deflect after one month for the model n°6 is of 0.0%.
Rate of sensor without deflect after one month for the model n°7 is not possible
as no sensor has been installed.
Rate of sensor without deflect after one month for the model n°8 is not possible
as no sensor has been installed.
Rate of sensor without deflect after one month for the model n°9 is not possible
as no sensor has been installed.
Rate of sensor without deflect after one month for the model n°10 is of 100.0%.
Rate of sensor without deflect after one month for the model n°11 is not
possible as no sensor has been installed.
Rate of sensor without deflect after one month for the model n°12 is of 100.0%.
Rate of sensor without deflect after one month for the model n°13 is not
possible as no sensor has been installed.
Rate of sensor without deflect after one month for the model n°14 is of 100.0%.
Rate of sensor without deflect after one month for the model n°15 is of 100.0%.
Rate of sensor without deflect after one month for the model n°16 is of 100.0%.
```

5 Conclusion :

Ce projet nous a permis d'acquérir une expérience concrète dans la gestion, l'analyse et la valorisation des données. En utilisant le SQL et le Python, nous avons pu créer une carte de contrôle fiable,

repérer des anomalies statistiques grâce à la p-valeur et établir un modèle solide pour surveiller la fiabilité des capteurs.

Ainsi, un technicien de l'usine de papetrie pourra facilement se rendre compte des capteurs défaillant. L'entreprise pourra, elle aussi, avoir des informations supplémentaires dans la fiabilité des modèles de capteurs qu'elle utilise. La direction pourra ainsi mieux choisir ses futurs capteurs s'ils nécessitent trop de maintenance.

Enfin, ce projet nous à montrer la nécessité de correctement comprendre l'ensemble de projet pour répartir correctement les différentes tâches entre les membres du groupe. Nous vous remercions du temps que vous avez pris à nous lire,

COLLOT Grégoire, MARECAILLE-HENAUT Mattieu, SIMSEK Dilara.