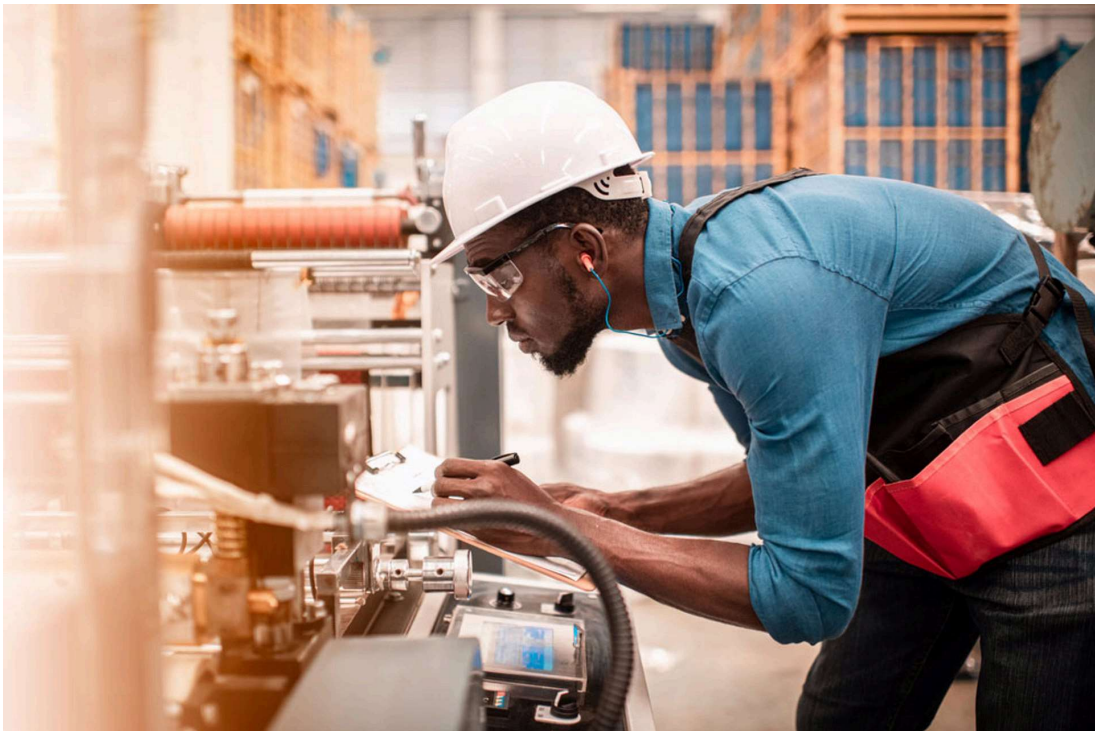


[Afficher le code](#)

Prédiction des situations de danger

on technician Dataset

1. | Introduction 🙌



Data Set Problems 🤔

- 👉 Client is a company which manages the public electricity distribution network over 95% of French territory.
- 👉 The problem concerns the safety of technicians. The technician works outdoors and must follow a set of procedures regarding the wearing of their PPE (Personal Protective Equipment). You are required to retrieve a set of information of the type opening of jackets, gloves not on, etc.
- 👉 From this information you will set up a virtual assistant capable of alerting him of his danger.
- 👉 AI will be the differentiating element to ensure that in any situation the technician will be safe(will wear the necessary and essential protective equipment for the situation).

The objective is to implement a solution to predict the moments when a technician is in danger during an intervention.

- 👉 **Data pre-processing and feature engineering will be performed to prepare the dataset** before it is used by the machine learning model.

👉 Machine learning models are necessary to determine whether a patient has heart disease and speed up the diagnostic process based on the medical information provided about that patient. The variables that most influence a patient to have heart disease will also be explored more deeply in this notebook.

Objectives of Notebook

The objective is to create a self-learning model to alert the technician if he is in danger during an intervention. It is therefore a question of working on unsupervised learning models making it possible to detect periods during which a technician was in danger, i.e. to alert the technician in the event of improper wearing of PPE during the intervention

👉 **This notebook aims to:**

- Perform **initial data exploration**.
- Perform **EDA**.
- Perform **data pre-processing**

Data Set Description

👉 There are **14 variables** in this data set:

- **2 categorical** variables,
- **11 numeric** variables,
- **1 time** ,

👉 The following is the **structure of the data set**.

Column Name	Data Type	Description
Scenario	int	represent type of intervention (1,2,3..)
Correct	Int	indicates wether the technician wear or no the equipments (port d'un équipement ne dit pas forc
Timestamp	Int	Indicates the occurence of scenario over the time in second
Datetime	Object	Indicates the occurence of scenario over the time of the day
t1_casque	Float	indicates the technician 1 wear or no helmet 0= not wear 1= wear
t1_visiere	float	indicates the technician 1 wear or no visior 0= not wear 1= wear
t1_col	Float	indicates the technician 1 wear or no collar 0= not wear 1= wear
t1_gant_gauche	Float	indicates the technician wear or no left glove 0= not wear 1= wear
t1_gant_droit	Float	indicates the technician 1 wear or no right glove 0= not wear 1= wear
danger	Int	it's the target to predict whether the technician is in danger or no 0= no danger 1= in danger

✓ 2. | Importing Libraries

👉 **Importing libraries** that will be used in this notebook.

[Afficher le code](#)

3. | Reading Dataset 🕶

👉 After importing libraries, the dataset that will be used will be imported.

[Afficher le code](#)

Scenario	Correct	Timestamp	Datetime	t1_casque	t1_visiere	t1_col	t1_gant_gauc
1	1	1626949343	7/22/2021 10:22	0.000000	0.000000	0.000000	0.000000
1	1	1626949359	7/22/2021 10:22	0.000000	0.000000	0.000000	0.000000
1	1	1626949374	7/22/2021 10:22	0.000000	0.000000	0.000000	0.000000
1	1	1626949389	7/22/2021 10:23	0.000000	0.000000	0.000000	0.000000
1	1	1626949405	7/22/2021 10:23	0.000000	0.000000	0.000000	0.000000

👉 From dataset report , it can be concluded that:

- There are 2 technicians t1 and t2 detected in the dataset. it also can be seen that there are values of each variable of technician t1 detected by sensor at the first time .
- As can be seen the target is danger, we can concluded that is a binary classification

[Afficher le code](#)

```

➡️ .. Dataset Info ..
*****
Total Rows: 393
Total Columns: 14
*****

.. Dataset Details ..
*****
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 393 entries, 0 to 392
Data columns (total 14 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Scenario        393 non-null    int64
1   Correct          393 non-null    int64
2   Timestamp        393 non-null    int64
3   Datetime         393 non-null    object

```

```

4  t1_casque      225 non-null    float64
5  t1_visiere    225 non-null    float64
6  t1_col        225 non-null    float64
7  t1_gant_gauche 225 non-null    float64
8  t1_gant_droit  225 non-null    float64
9  t2_casque     168 non-null    float64
10 t2_visiere    168 non-null    float64
11 t2_veste     168 non-null    float64
12 t2_col       168 non-null    float64
13 danger       393 non-null    int64
dtypes: float64(9), int64(4), object(1)

```

👉 It can be seen that dataset has successfully imported.

👉 In the dataset, there are **14 columns** and **393 observations** with various data types.

✓ 4. | Initial Data Exploration 🔍

👉 This section will focused on **initial data exploration** before pre-process the data.

✓ 4.2 | Descriptive Statistics 1 2 3 4

👉 This section will show **descriptive statistics** of numerical variables.

[Afficher le code](#)

	Scenario	Correct	Timestamp	t1_casque	t1_visiere	t1_col	t1_
count	393.000000	393.000000	393.000000	225.000000	225.000000	225.000000	
mean	2.167939	0.585242	1626536276.249364	0.822222	0.120000	0.115556	
std	1.424025	0.493308	432199.611987	0.383178	0.325686	0.320404	
min	1.000000	0.000000	1626084505.000000	0.000000	0.000000	0.000000	
25%	1.000000	0.000000	1626087802.000000	1.000000	0.000000	0.000000	
50%	2.000000	1.000000	1626949436.000000	1.000000	0.000000	0.000000	
75%	3.000000	1.000000	1626951220.000000	1.000000	0.000000	0.000000	
max	6.000000	1.000000	1626965782.000000	1.000000	1.000000	1.000000	

✓ 4.3 | Missing Values Exploration ?

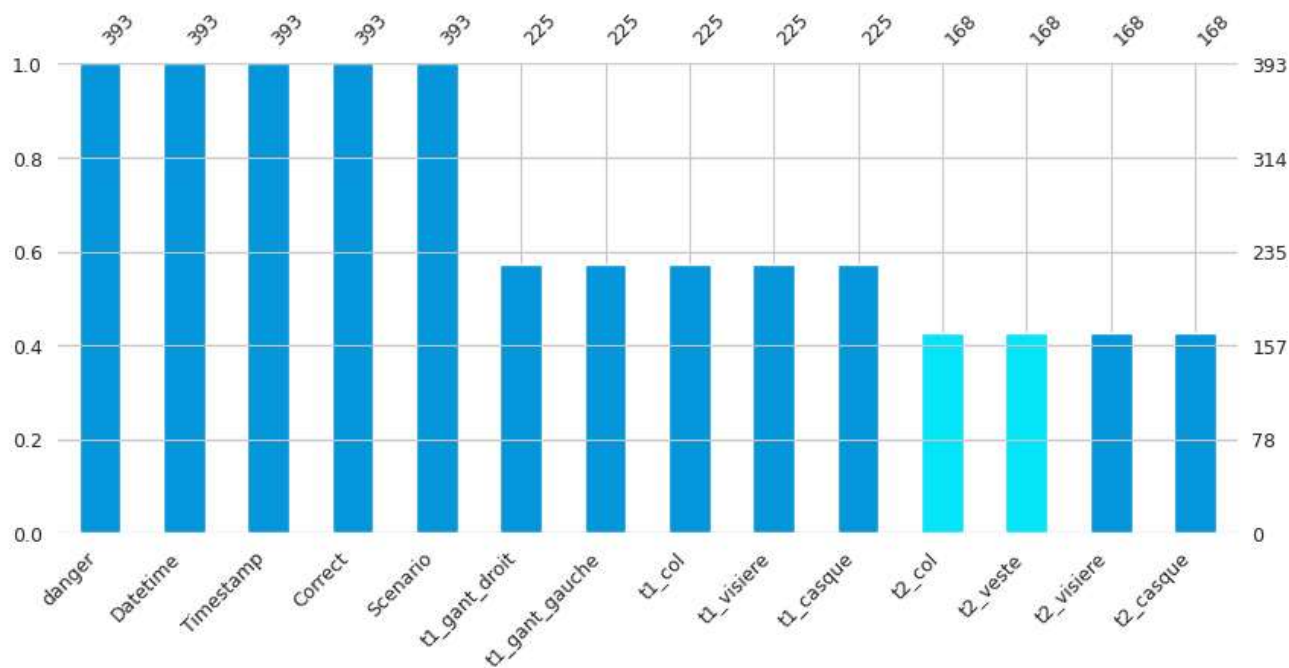
👉 This section will show **missing values exploration** for all columns.

[Afficher le code](#)



```
*****
.: Total Missing Values in each Columns :.
*****
Scenario          0
Correct           0
Timestamp         0
Datetime          0
t1_casque        168
t1_visiere       168
t1_col           168
t1_gant_gauche   168
t1_gant_droit    168
t2_casque        225
t2_visiere       225
t2_veste         225
t2_col           225
danger           0
dtype: int64
```

Missing Values in each Columns



👉 However, there are **null values** detected in "t1_casque" and "t1_visiere" and "t1_col" and "t1_gant_gauche" and "t1_gant_droit" and "t2_casque" and "t2_visiere" and "t2_veste" and "t2_col" columns.

✓ 5. | EDA

👉 This section will **perform some EDA to get more insights about dataset, explore variables relationship** in the dataset using different various plots/charts.

👉 The variables that unnecessary will be deleted. 👉 **we are going to predict the danger for technician t1 in our work and delete technician t2.**

✓ 5.1 | Dropping Unnecessary Variables ▼

[Afficher le code](#)

⇒	Scenario	Correct	Timestamp	Datetime	t1_casque	t1_visiere	\
0	1	1	1626949343	7/22/2021 10:22	0.0	0.0	
1	1	1	1626949359	7/22/2021 10:22	0.0	0.0	
2	1	1	1626949374	7/22/2021 10:22	0.0	0.0	
3	1	1	1626949389	7/22/2021 10:23	0.0	0.0	
4	1	1	1626949405	7/22/2021 10:23	0.0	0.0	

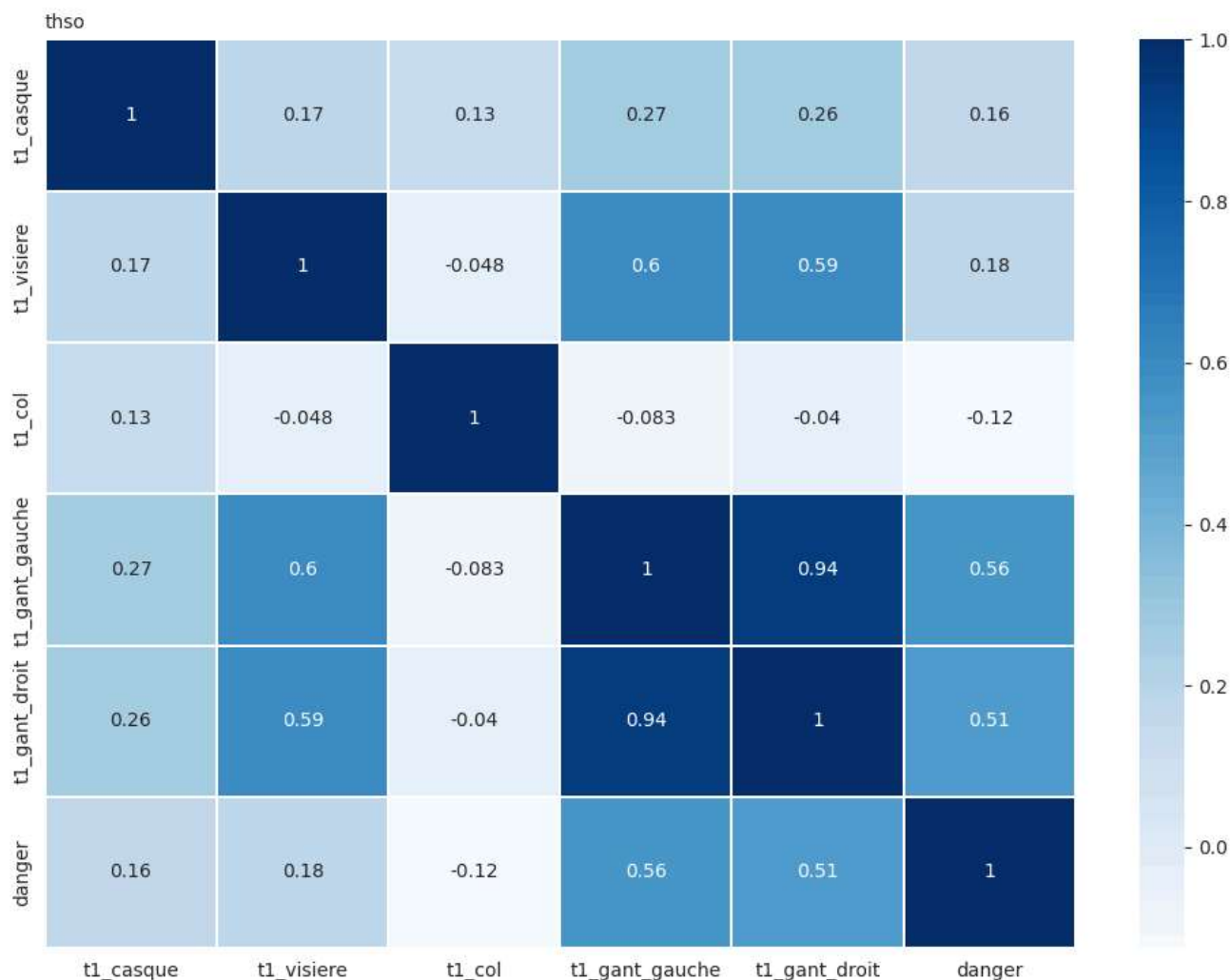
	t1_col	t1_gant_gauche	t1_gant_droit	danger
0	0.0	0.0	0.0	0
1	0.0	0.0	0.0	0
2	0.0	0.0	0.0	0
3	0.0	0.0	0.0	0
4	0.0	0.0	0.0	0

✓ 5.2 | Heatmap

[Afficher le code](#)



Correlation Map of Equipment Item



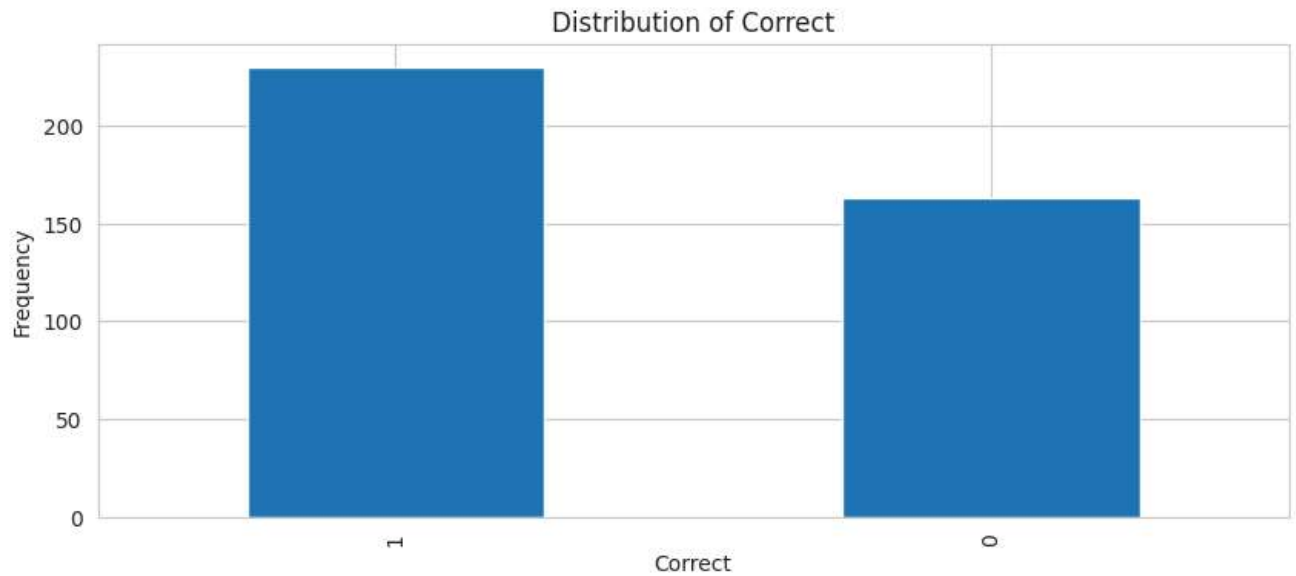
👉 't1_gant_gauche' (left glove) and 't1_gant_droit' (right glove) have the highest positive correlations with 'danger' (0.56 and 0.51, respectively), indicating a potential association between glove checks and danger flags.

👉 Other equipment checks like 't1_casque' (helmet) and 't1_visiere' (visor) show weaker positive correlations with 'danger'. This heatmap provides a visual representation of how each variable in the dataset is correlated with one another, with a particular focus on their relationship with 'danger'

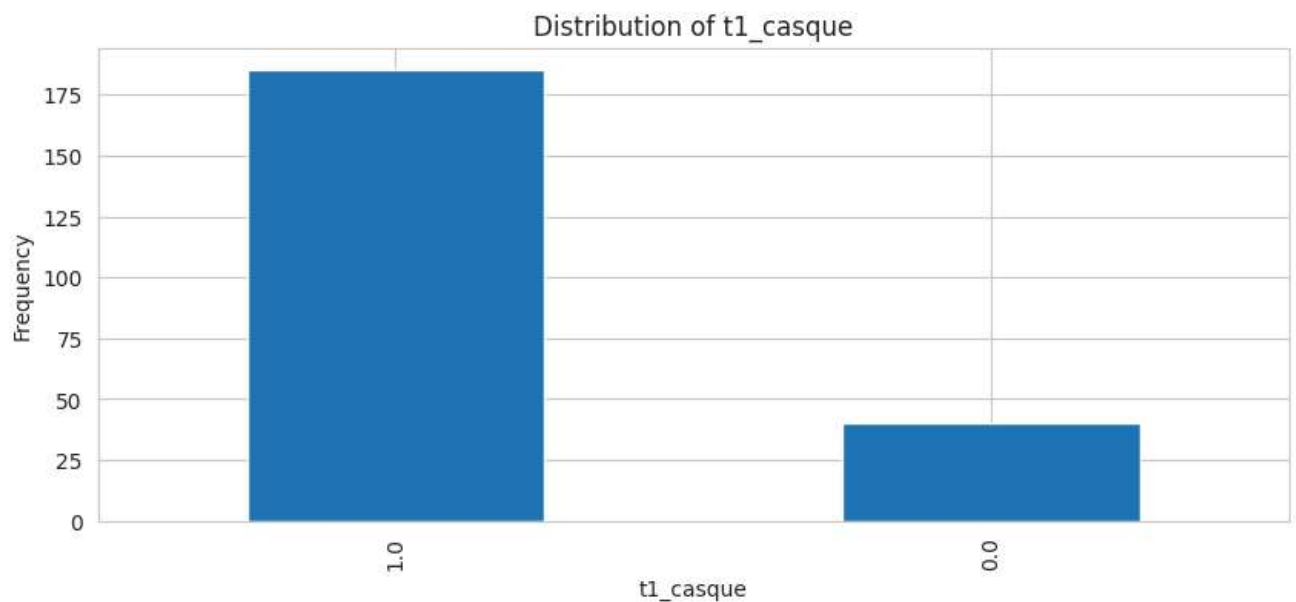
✓ 5.3 | Features distributions

[Afficher le code](#)

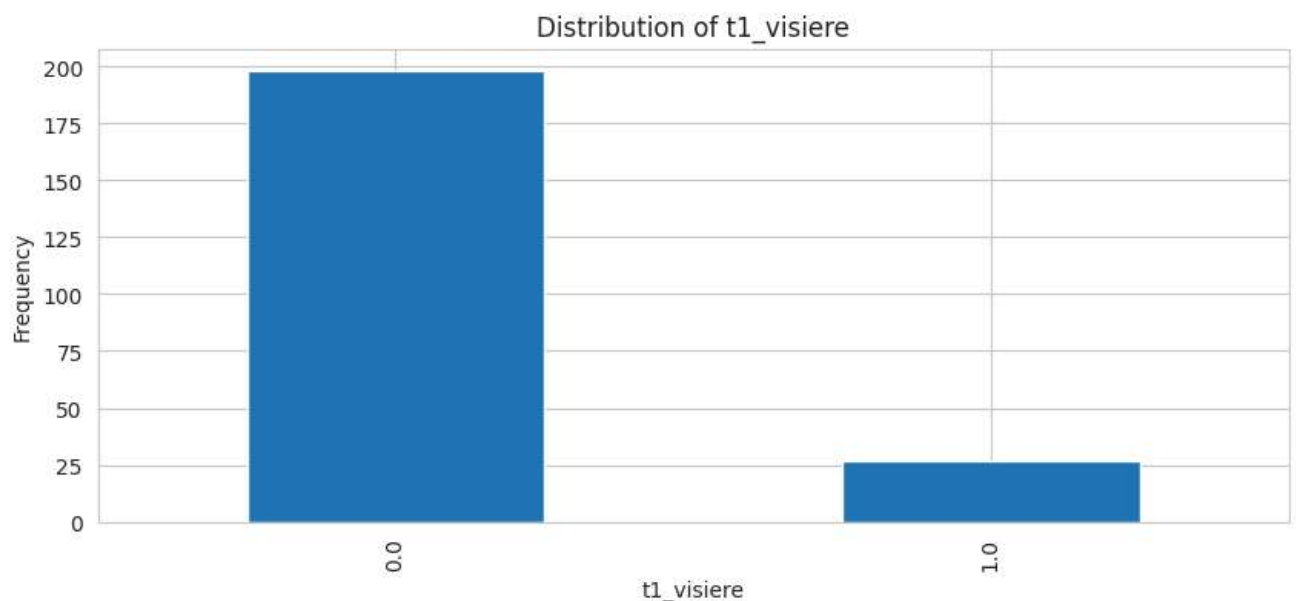
→ /usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning
and should_run_async(code)
Plotting distributions: 0% | 0/7 [00:00<?, ?it/s]



Plotting distributions: 14% | 1/7 [00:00<00:01, 5.11it/s]

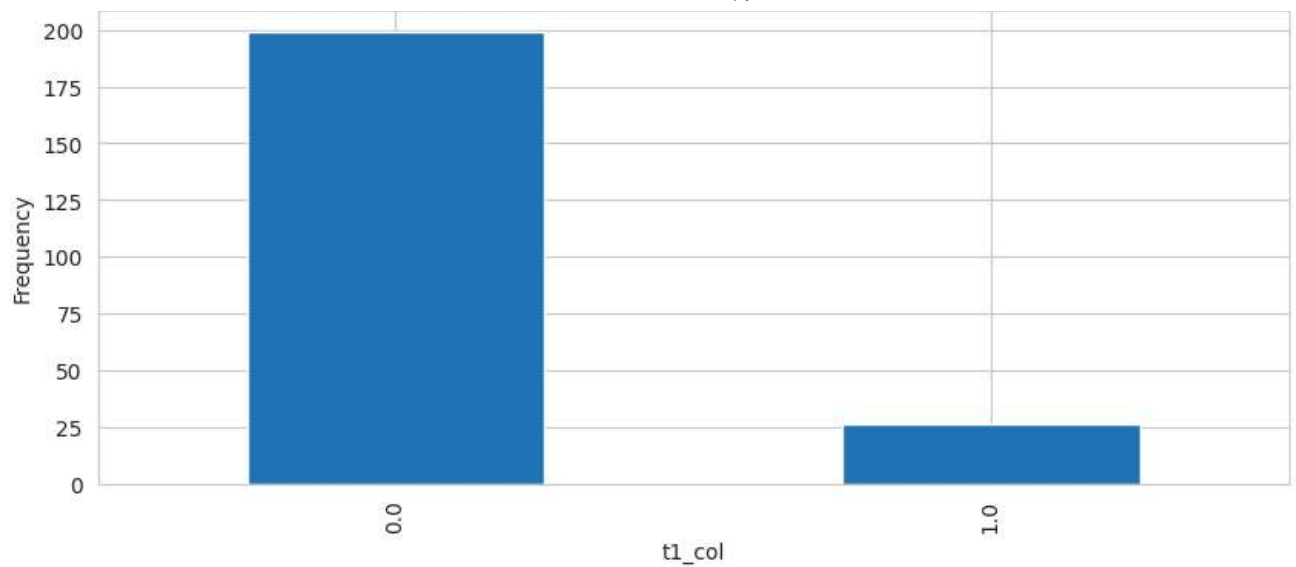


Plotting distributions: 29% | 2/7 [00:00<00:00, 5.37it/s]



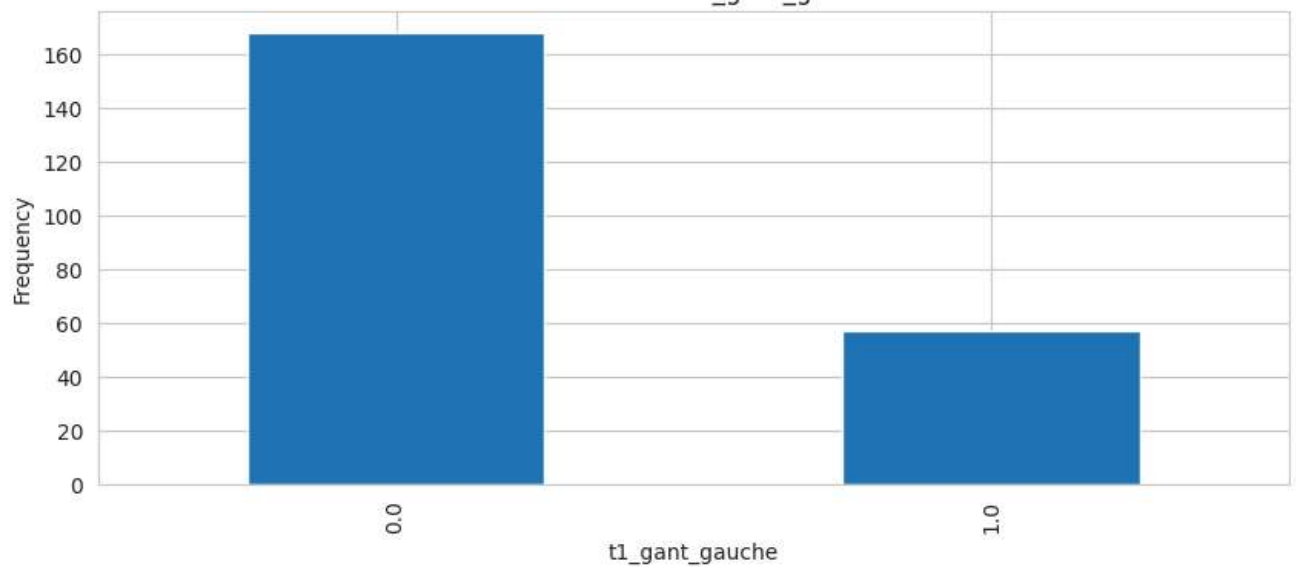
Plotting distributions: 43% | 3/7 [00:00<00:00, 5.41it/s]

Distribution of t1_col



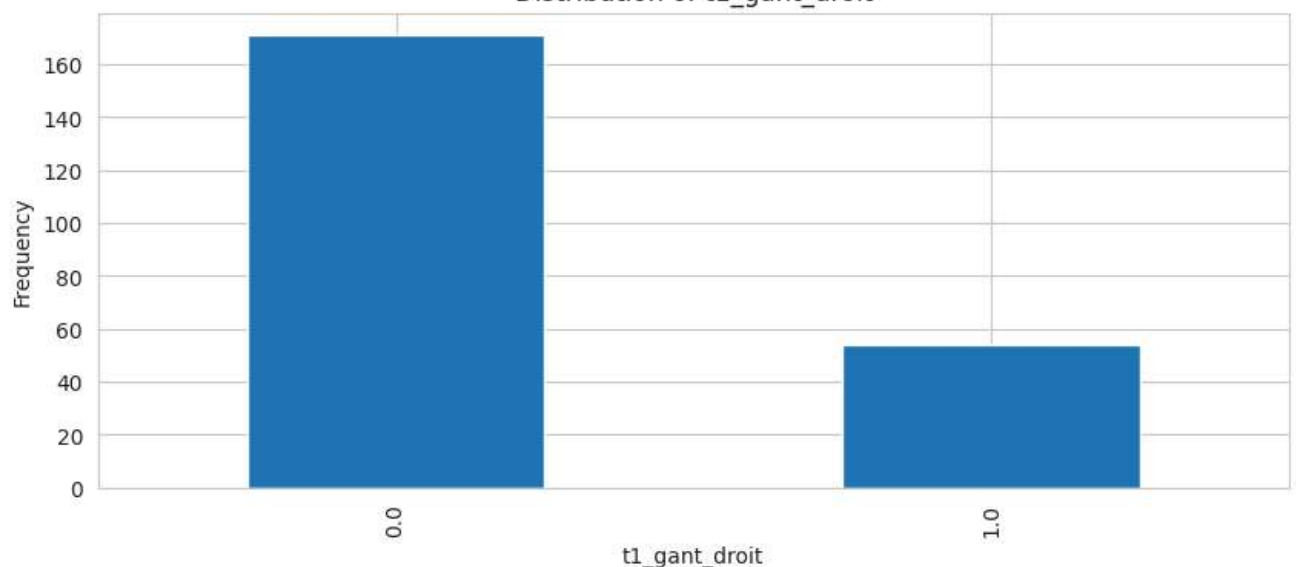
Plotting distributions: 57% | 4/7 [00:01<00:01, 2.83it/s]

Distribution of t1_gant_gauche



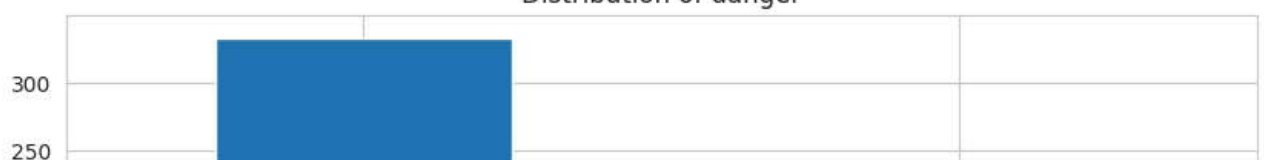
Plotting distributions: 71% | 5/7 [00:01<00:00, 3.35it/s]

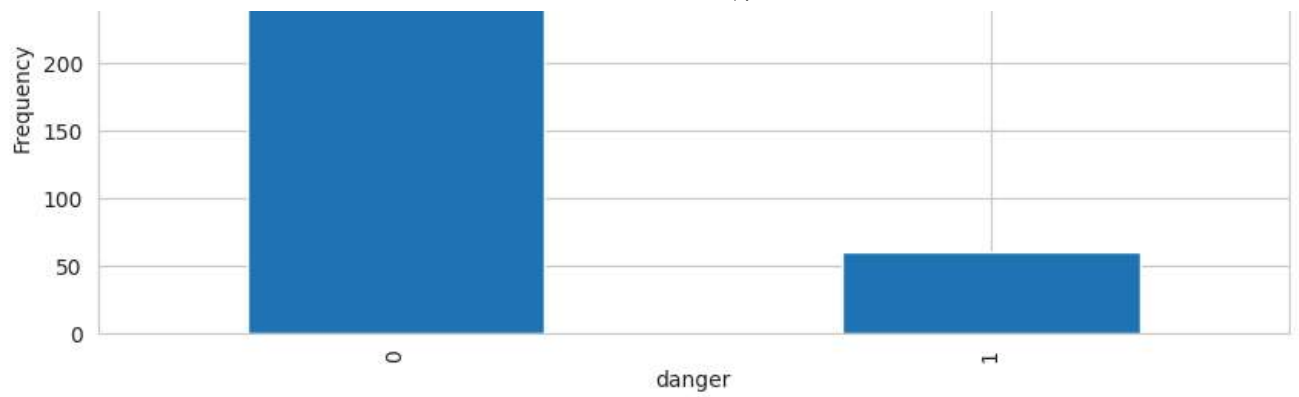
Distribution of t1_gant_droit



Plotting distributions: 86% | 6/7 [00:01<00:00, 3.85it/s]

Distribution of danger





Plotting distributions: 100% | ██████████ | 7/7 [00:01<00:00, 4.02it/s]

[Afficher le code](#)

```
-----  
NameError                                Traceback (most recent call last)  
<ipython-input-2-7039bd96d4f2> in <cell line: 4>()  
      2  
      3 # labels=ds['Outlet_Location_Type'].dropna().unique()  
----> 4 order=df['danger'].value_counts().index  
      5  
      6 # --- Size for Both Figures ---  
  
NameError: name 'df' is not defined
```

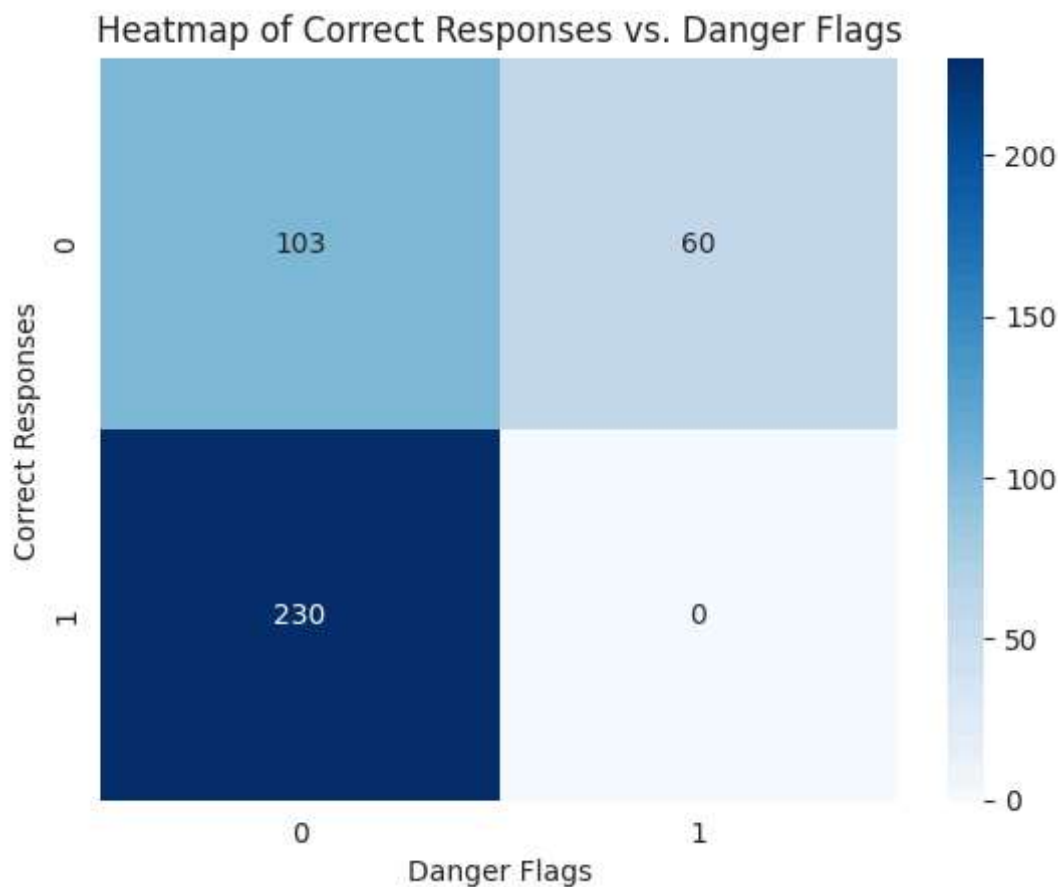
👉 La répartition des classes confirme un déséquilibre important, la classe « 0 » étant bien plus répandue que la classe « 1 ». Pour résoudre ce problème, j'appliquerai une technique de rééchantillonnage pour équilibrer les classes.

✓ 5.4 | Relationship between Correct responses and danger

Next, I will explore the relationship between the **'Correct' responses and the 'danger'** to see if there is a pattern indicating that correct responses correlate with a lower incidence of danger flags. This will involve creating a cross-tabulation and visualizing the relationship.

[Afficher le code](#)

```
➔ /usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning  
and should_run_async(code)  
danger      0    1  
Correct  
0          103   60  
1          230   0
```

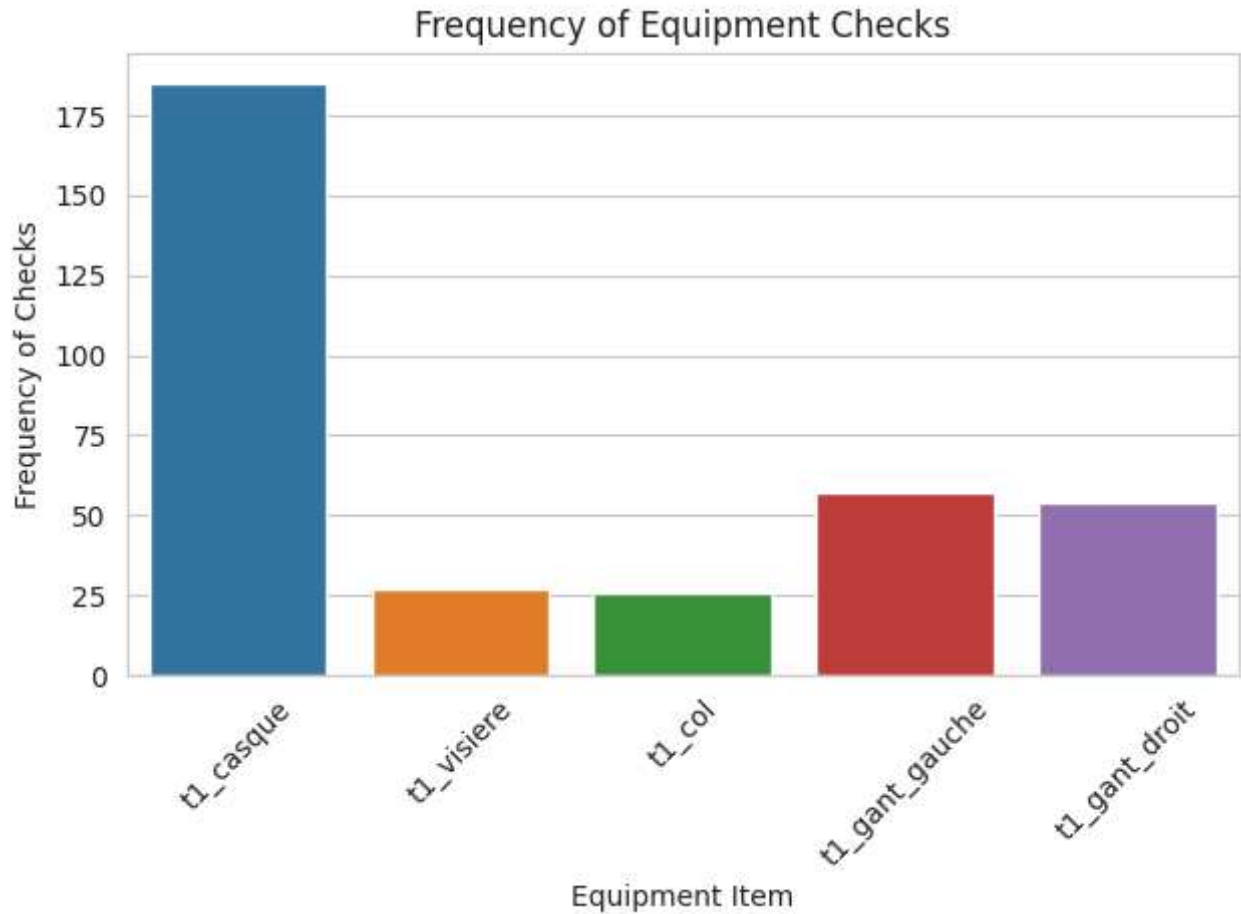


👉 From the heatmap, we can observe that when **responses are correct**, there are **no instances of danger flags** being raised. However, when **responses are incorrect**, there are **both instances with and without danger flags**. This suggests that correct responses may be associated with a lower incidence of danger flags, indicating safer outcomes.

✓ 5.5 | Frequency of checks for each equipment item

[Afficher le code](#)

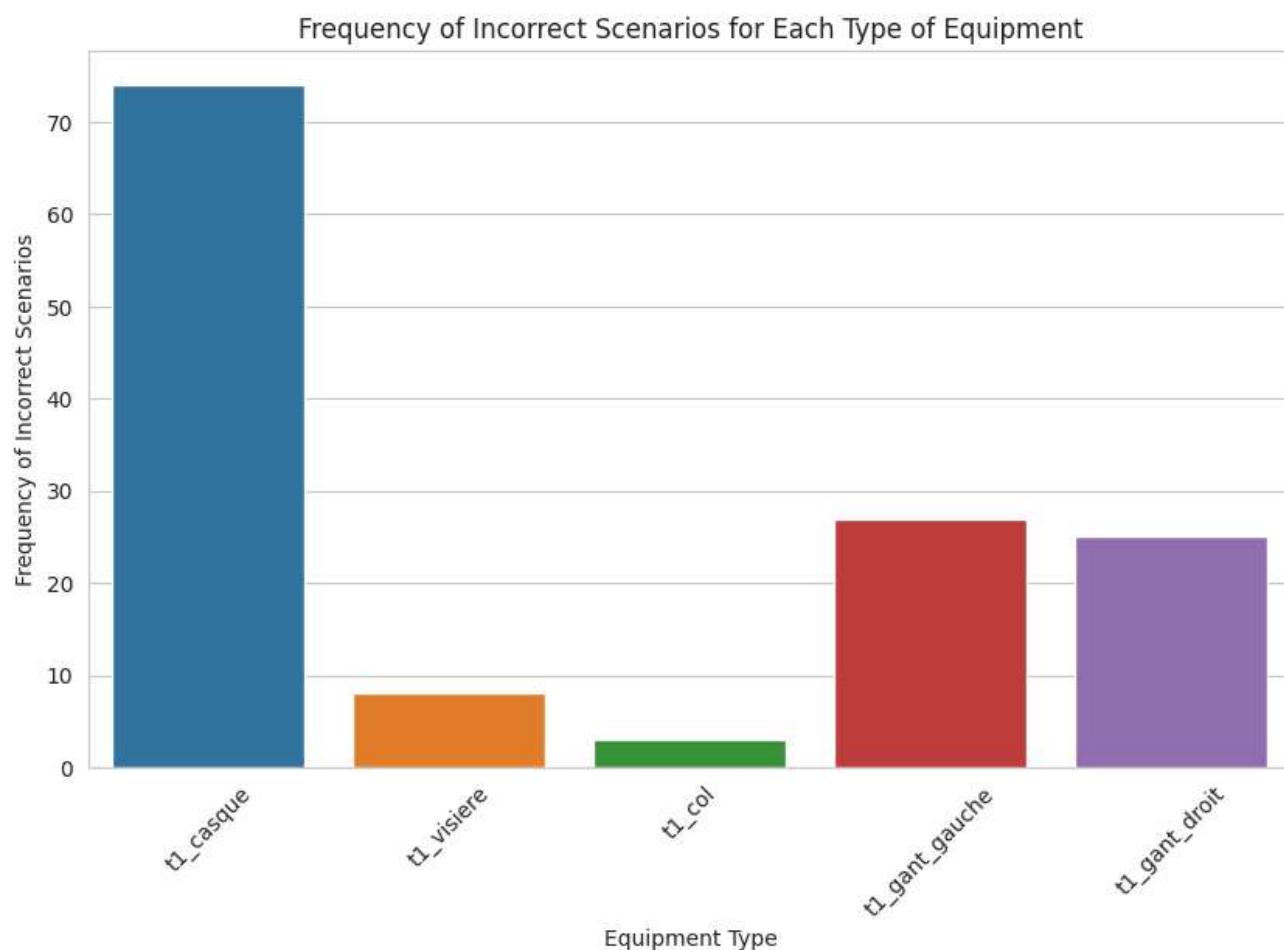
```
→ /usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning  
and should_run_async(code)  
Frequency of checks for each equipment item:  
t1_casque          185.0  
t1_visiere         27.0  
t1_col            26.0  
t1_gant_gauche     57.0  
t1_gant_droit      54.0  
dtype: float64
```



✓ 5.6 | Frequency of incorrect scenarios for each type of equipment

[Afficher le code](#)


→ /usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning and should_run_async(code)

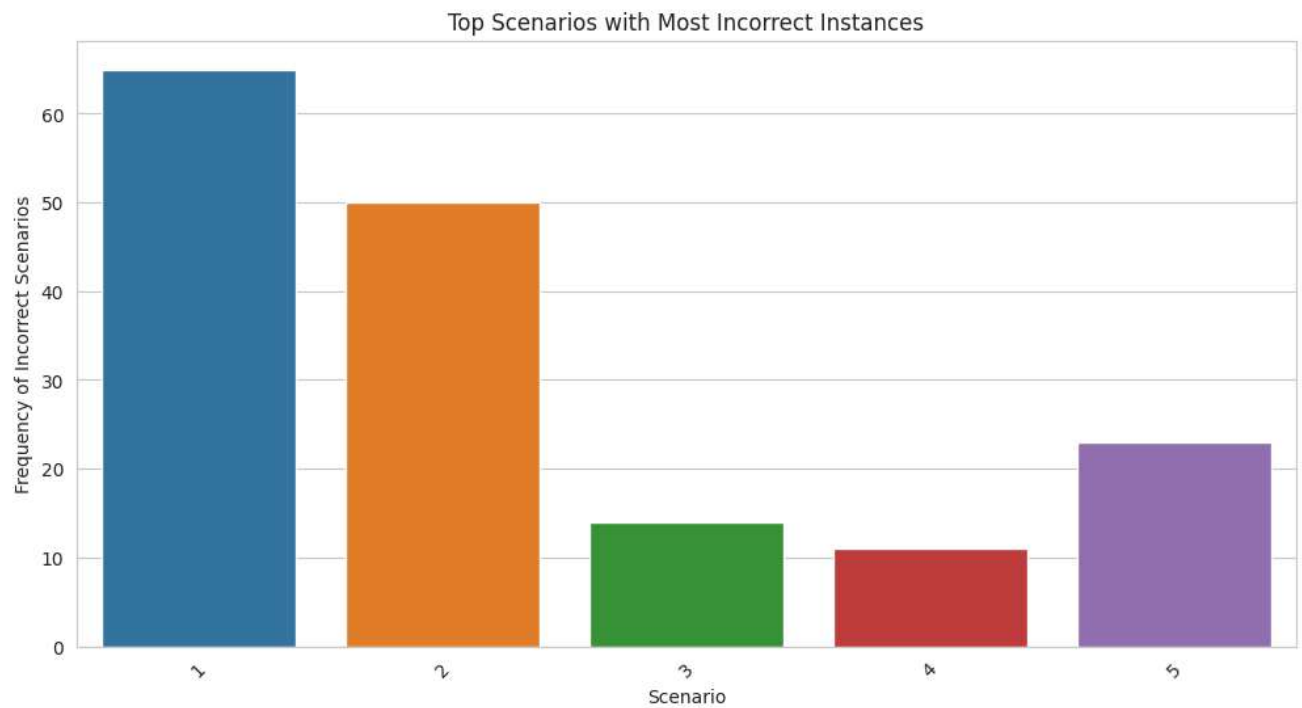


👉 The bar plot above displays the frequency of incorrect scenarios associated with each type of equipment. This visualization can help **identify which equipment items are most often not used correctly**, potentially indicating areas where additional training or safety measures may be needed.

👉 Next, I will conduct a deeper analysis to identify any specific scenarios that have a high frequency of incorrect outcomes. This could help pinpoint particular situations that are problematic and may benefit from targeted interventions or training. I'll calculate the scenario-wise frequency of incorrect outcomes and visualize the top scenarios with the most incorrect instances. Let's proceed with this analysis.


[Afficher le code](#)

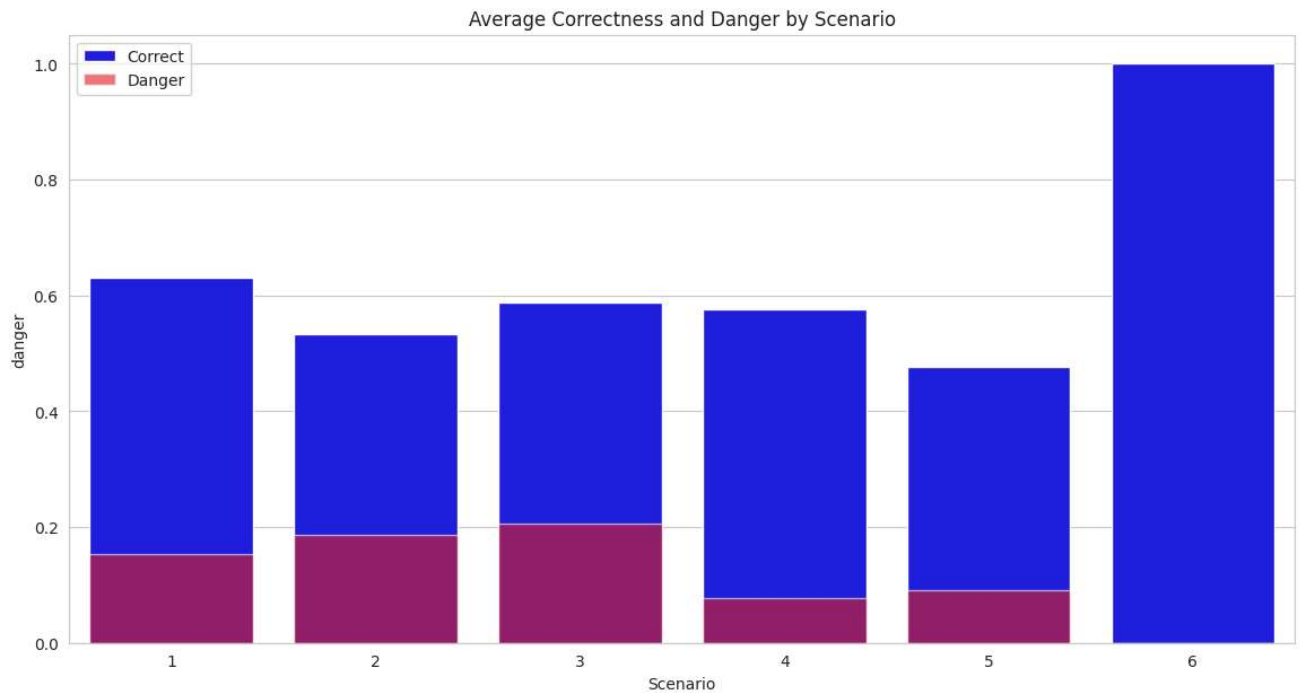
 /usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning and should_run_async(code)



👉 The bar plot above shows the top scenarios with the most incorrect instances. **This information is crucial for identifying which scenarios are most challenging and may require further attention or improvement in protocol adherence.**

[Afficher le code](#)

 /usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning and should_run_async(code)



👉 The bar chart above displays **the average correctness and danger by scenario**. Each scenario is evaluated for its average correctness (in blue) and the average level of danger (in red, overlaid). This visualization can help identify which scenarios are typically performed correctly and which ones are associated with higher levels of danger

👉 I will analyze the correlation between the different equipment checks and the danger flag to see if there's any noticeable pattern that could suggest a relationship between the equipment used and the occurrence of danger. This could be valuable for understanding the importance of each piece of equipment in ensuring safety.

👉 I will perform an analysis to identify the factors that are most associated with the presence of danger flags. This will involve:
Correlation analysis to see how different variables relate to 'danger'.

✓ 5.7 | Scatter matrix for Equipment checks and danger

[Afficher le code](#)

[Afficher la sortie masquée](#)

👉 The scatter matrix provides a visual examination of potential correlations or patterns between the equipment checks and the danger level. Each plot shows the relationship between two variables, with the kernel density estimation (KDE) on the diagonal showing the distribution of a single variable

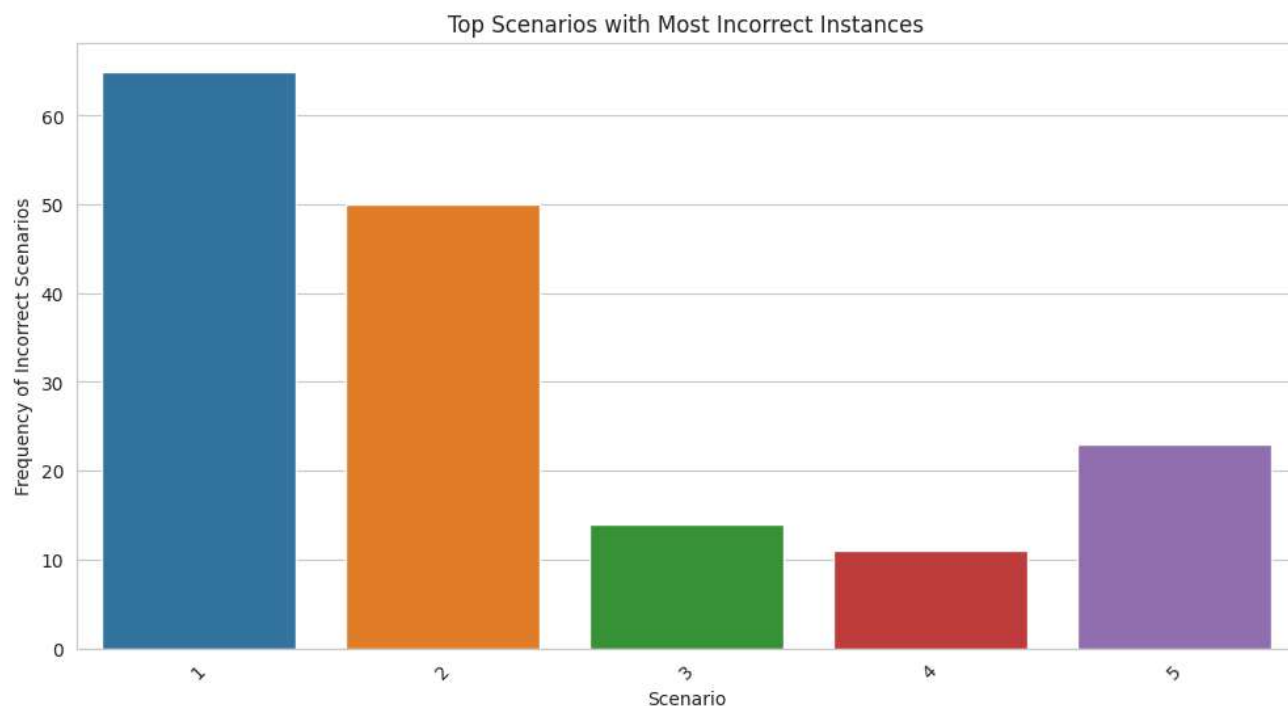
👉 I will explore the scenarios to determine if there are specific types of equipment that are more frequently associated with incorrect scenarios. This could highlight potential areas for safety improvement. I'll calculate the frequency of incorrect scenarios for each type of equipment and visualize the results. Let's proceed with this analysis.

👉 Next, I will conduct a deeper analysis to identify any specific scenarios that have a high frequency of incorrect outcomes. This could help pinpoint particular situations that are problematic and may benefit from targeted interventions or training. I'll calculate the scenario-wise frequency of incorrect outcomes and visualize the top scenarios with the most incorrect instances. Let's proceed with this analysis

✓ 5.8 | Frequency of incorrect scenarios for each scenario type

[Afficher le code](#)

➡ /usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning and should_run_async(code)



👉 The bar plot above shows the top scenarios with the most incorrect instances. This information is crucial for identifying which scenarios are most challenging and may require further attention or improvement in protocol adherence.

✓ 6. | Data Pre-processing ⚙️

👉 **Data pre-processing** will be performed in this section to ensure high-quality data and prepare the dataset before building the machine learning models. by imputing missing values.

✓ 6.1 | Handling Missing Values ❓

👉 This section will **handle missing values** in "Outlet_Size" and "Item_Weight" columns.

✓ 6.2.1 | Imputation 🛠️

👉 Given the presence of **NaN values**, we will fill them with the median of their respective columns. # This is a common strategy for handling missing values.

[Afficher le code](#)

```
Series([], dtype: int64)
```

👉 The values have been successfully filled, and there are no more missing values in the dataset. Now that the data is clean, we can proceed with the logistic regression analysis. Let's reattempt to fit the logistic regression model with the cleaned data.*

✓ 6.2.2 | After Imputation 🛠️

👉 After imputation , **a final check will be done to check if there is still missing values left in the dataset.**

```
filled_data.isna().sum()
```

```
Scenario          0
Correct           0
Timestamp         0
Datetime          0
t1_casque         0
t1_visiere        0
t1_col            0
t1_gant_gauche    0
t1_gant_droit     0
danger            0
dtype: int64
```

✓ 6.2.3 | Features Separating and Splitting 🛠️

👉 After imputation , **the 'target' (dependent) column will be seperated from independent columns. Also, the dataset will be splitted into 80:20 ratio (80% training and 20% testing)..**

[Afficher le code](#)

```
X_train.shape,y_train.shape
```

```
((314, 5), (314,))
```

```
X_test.shape,y_test.shape
```

```
((79, 5), (79,))
```

✓ 7. | Model Implementation 🛠️

👉 This section will implement various machine learning models as mentioned in Introduction section. In addition, explanation for each models also will be discussed.

7.1 | Logistic Regression, Random Forest and SVM

[Afficher le code](#)



100%

3/3 [00:01<00:00, 2.46it/s]

```
Logistic Regression performance:  
Accuracy: 0.8860759493670886  
Precision: 0.0  
Recall: 0.0  
F1 Score: 0.0
```

```
Random Forest performance:  
Accuracy: 0.8987341772151899  
Precision: 0.5714285714285714  
Recall: 0.4444444444444444  
F1 Score: 0.5
```

```
Gradient Boosting performance:  
Accuracy: 0.8987341772151899  
Precision: 0.5714285714285714  
Recall: 0.4444444444444444  
F1 Score: 0.5
```

👉 The accuracy is relatively high, but the precision, recall, and F1 score are all zero for logistic Regression, indicating that the model is not effectively predicting the positive class. This could be due to an imbalance in the dataset or other factors that may require further investigation and model tuning.

👉 As seen i train several classification models on the training set and evaluate their performance on the testing set. The models include Logistic Regression, Random Forest, and SVC.

👉 Given these results, any of the models could be chosen as the best model for detecting periods during which a technician was in danger. However, it would be prudent to further investigate the dataset, possibly perform cross-validation, and consider other metrics or domain-specific considerations before making a final decision.

👉 Given the identical performance metrics of the models, it would be beneficial to investigate the dataset further for potential issues such as class imbalance or data leakage, and to perform cross-validation to ensure the robustness of the model performance. Additionally, exploring feature importance and considering domain-specific knowledge could provide insights into the factors contributing to the prediction of danger periods.

👉 **Check for class imbalance in the target variable. Perform cross-validation on the models to validate their performance. Investigate feature importance to understand which features are most predictive of danger.**

7.2 | Check the imbalance data 🔥

[Afficher le code](#)

```

➡ 0    333
   1     60
   Name: danger, dtype: int64

```

👉 The class distribution in the target variable is as follows:

Class 0 (No Danger): 333 instances Class 1 (Danger): 60 instances The proportion of the minority class (Danger) is approximately 15.27%. This indicates a class imbalance, which could affect the performance of the models and might be the reason why all models have similar performance metrics. To address this, we could consider techniques such as oversampling the minority class, undersampling the majority class, or using algorithms that are less sensitive to class imbalance.

👉 To address this, I will apply a resampling technique to balance the classes

✓ 7.3 | Balance the dataset with SMOTE

👉 We can apply techniques such as **Synthetic Minority Over-sampling Technique (SMOTE)** to create synthetic samples of the minority class. After balancing the classes, we can retrain the models and evaluate their performance again, focusing on precision, recall, and F1 score, which are more informative metrics in the context of imbalanced classes.

[Afficher le code](#)

```

➡ 0    263
   1    263
   Name: danger, dtype: int64

```

👉 The class distribution after applying SMOTE is now balanced: Class 0 (No Danger): 263 instances Class 1 (Danger): 263 instances With a balanced dataset, I will now retrain the models and evaluate their performance using precision, recall, and F1 score

✓ 7.4 | Retrain the models and Evaluate their performance

[Afficher le code](#)

```

➡ 100%|██████████| 3/3 [00:00<00:00, 8.28it/s]Logistic Regression performance:
Accuracy: 0.8607594936708861
Precision: 0.42857142857142855
Recall: 0.6666666666666666
F1 Score: 0.5217391304347826
roc_auc_score: 0.7761904761904761

```

```

Random Forest performance:
Accuracy: 0.27848101265822783
Precision: 0.11290322580645161
Recall: 0.7777777777777778
F1 Score: 0.19718309859154928

```