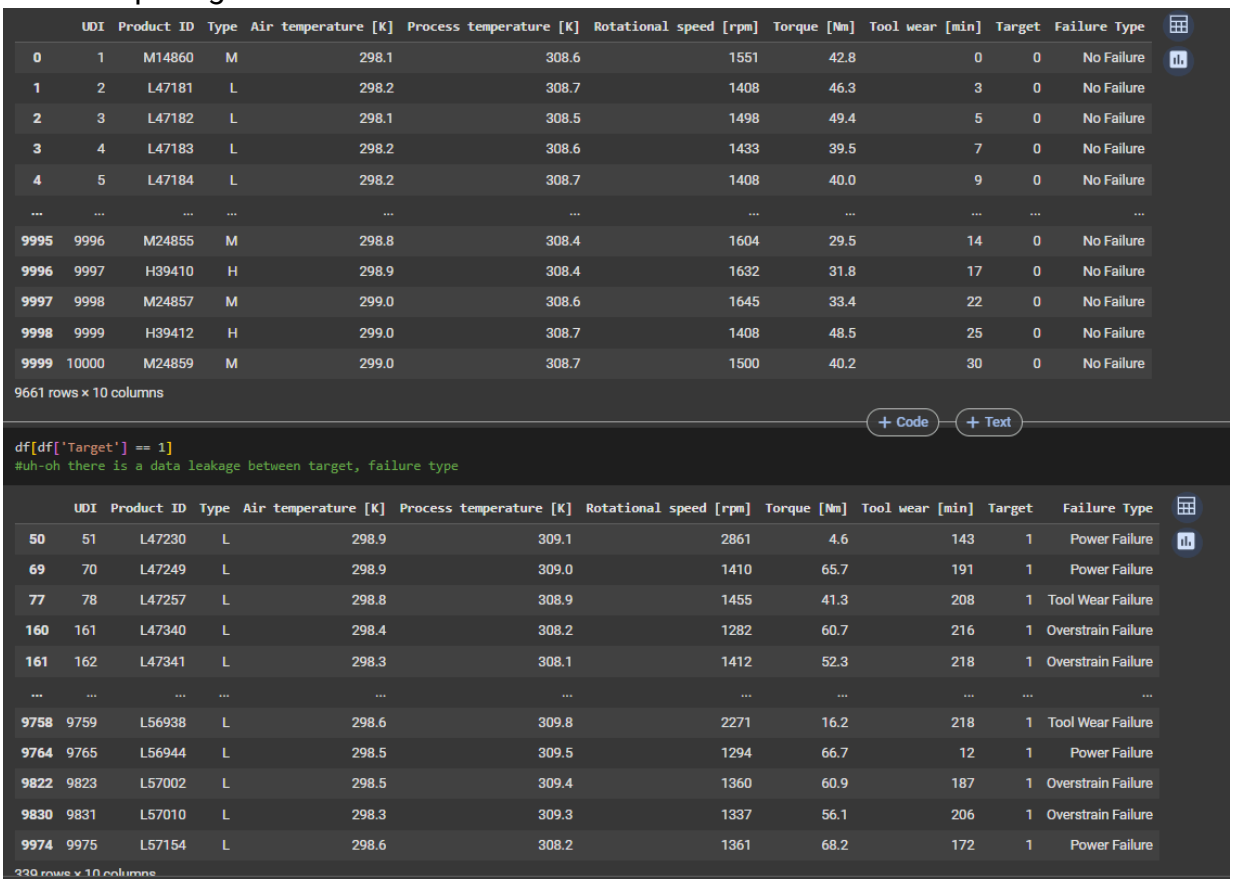
Data leakage assignment

Team names, ids:

Menna Osama Elminshawy	20217011
Yassin Ehab	20216117
Radwa Belal	20217005

Data cleaning steps:

1. First we checked for any duplicates, any missing values in the data. And we found none
2. After that we encoded the categorical column “Type” using the label encoding technique such that: High is 0, low is 1, medium is 2
3. Explored the data to see what may cause the problem of data leakage and found out that :
    - ➔ The **`Failure Type`** column introduces **target leakage** because it only contains values when `Failure = 1`. If used as a feature, the model can trivially predict failures by checking whether `Failure Type` is populated, rather than learning meaningful patterns from sensor data. This cause a poor generalization



	UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target	Failure Type
0	1	M14860	M	298.1	308.6	1551	42.8	0	0	No Failure
1	2	L47181	L	298.2	308.7	1408	46.3	3	0	No Failure
2	3	L47182	L	298.1	308.5	1498	49.4	5	0	No Failure
3	4	L47183	L	298.2	308.6	1433	39.5	7	0	No Failure
4	5	L47184	L	298.2	308.7	1408	40.0	9	0	No Failure
...	...	...	...	...	...	...	...	...	...	...
9995	9996	M24855	M	298.8	308.4	1604	29.5	14	0	No Failure
9996	9997	H39410	H	298.9	308.4	1632	31.8	17	0	No Failure
9997	9998	M24857	M	299.0	308.6	1645	33.4	22	0	No Failure
9998	9999	H39412	H	299.0	308.7	1408	48.5	25	0	No Failure
9999	10000	M24859	M	299.0	308.7	1500	40.2	30	0	No Failure

9661 rows × 10 columns

+ Code + Text

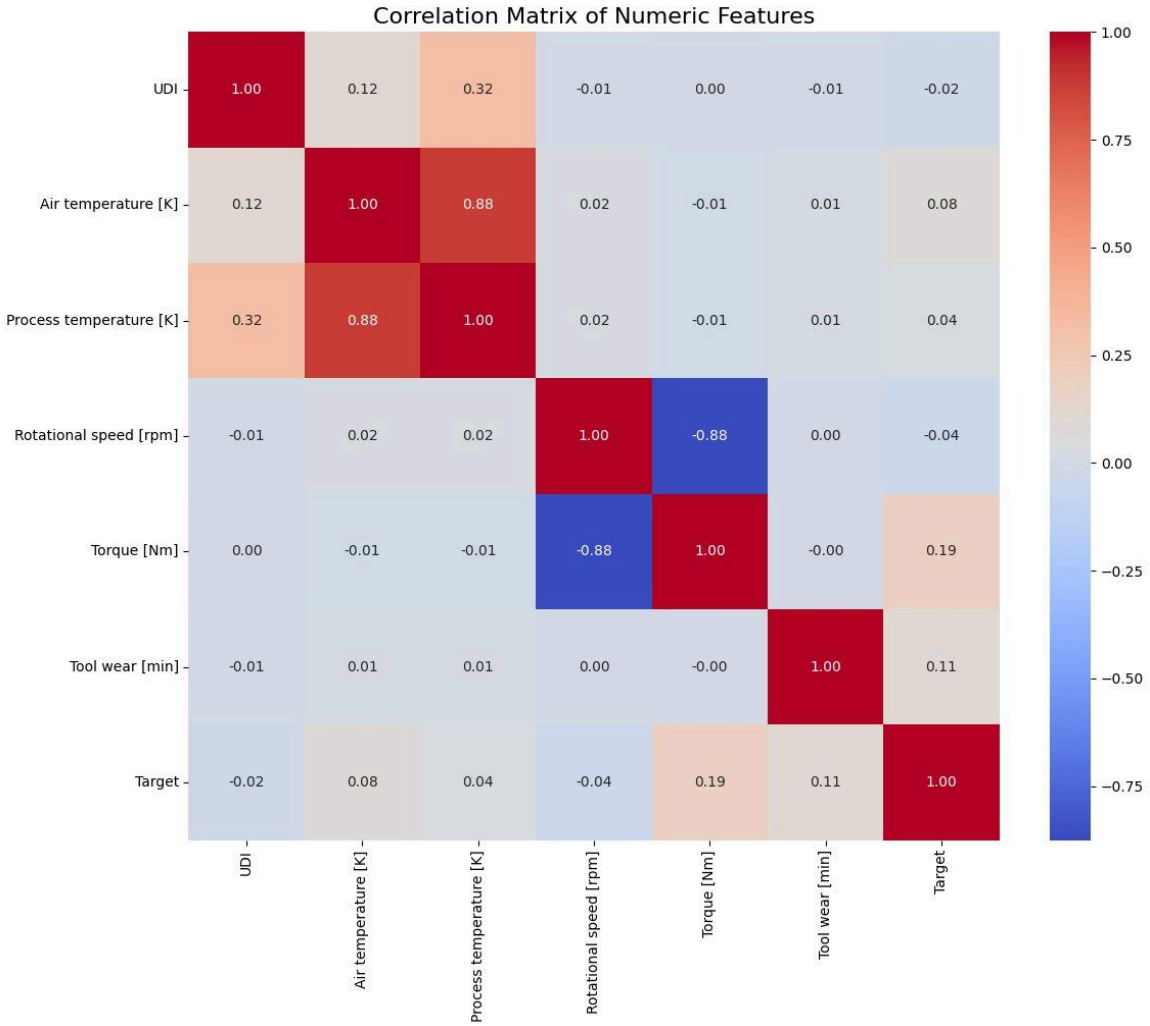
```
df[df['Target'] == 1]
#uh-oh there is a data leakage between target, failure type
```

	UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target	Failure Type
50	51	L47230	L	298.9	309.1	2861	4.6	143	1	Power Failure
69	70	L47249	L	298.9	309.0	1410	65.7	191	1	Power Failure
77	78	L47257	L	298.8	308.9	1455	41.3	208	1	Tool Wear Failure
160	161	L47340	L	298.4	308.2	1282	60.7	216	1	Overstrain Failure
161	162	L47341	L	298.3	308.1	1412	52.3	218	1	Overstrain Failure
...	...	...	...	...	...	...	...	...	...	...
9758	9759	L56938	L	298.6	309.8	2271	16.2	218	1	Tool Wear Failure
9764	9765	L56944	L	298.5	309.5	1294	66.7	12	1	Power Failure
9822	9823	L57002	L	298.5	309.4	1360	60.9	187	1	Overstrain Failure
9830	9831	L57010	L	298.3	309.3	1337	56.1	206	1	Overstrain Failure
9974	9975	L57154	L	298.6	308.2	1361	68.2	172	1	Power Failure

339 rows × 10 columns

    - ➔ The “Product ID” can also be a cause of data leakage if the same machine /product appeared in both training, testing

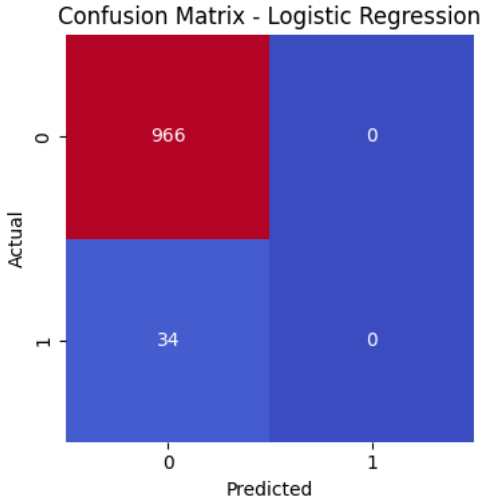
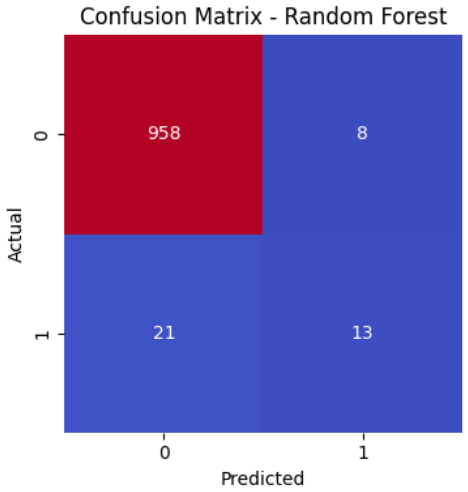
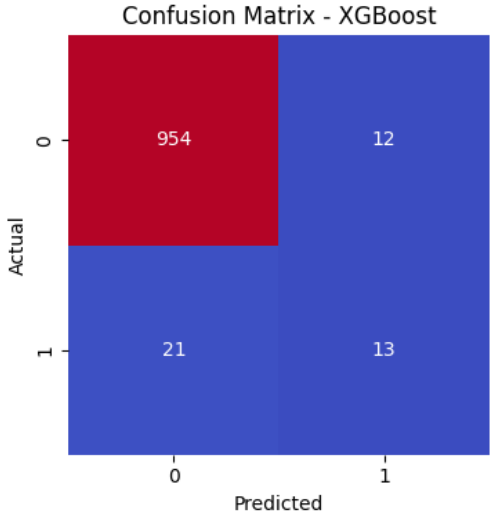
4. Explored the data and found out there are some highly correlated features like “Torque” & “Rotational Speed”, “Process temperature” & “Air temperature” those may affect the performance of the model so we dropped the “Torque” , “Process temperature”



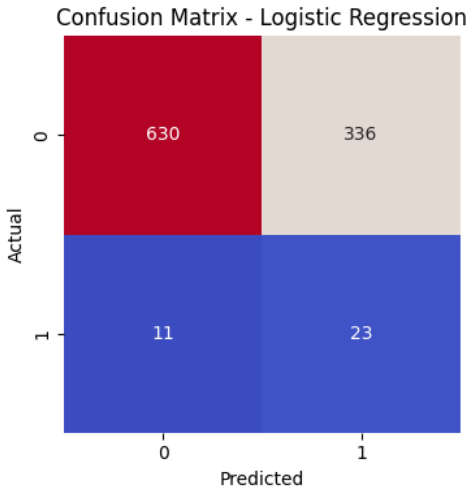
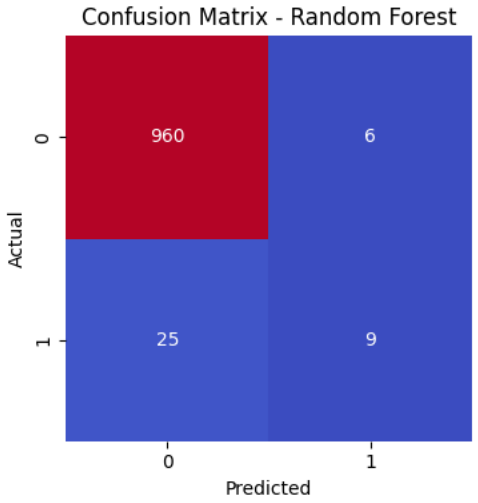
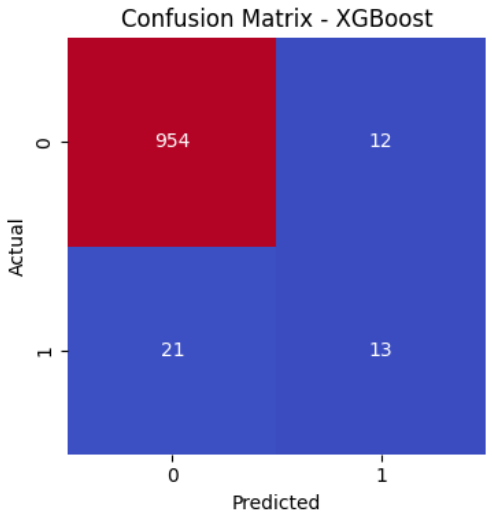
Machine models:

We used many models to classify if it failed or not like DT, Random Forest, Logistic Regression, SVM

Here is the before and after handling data leakage evaluation results:

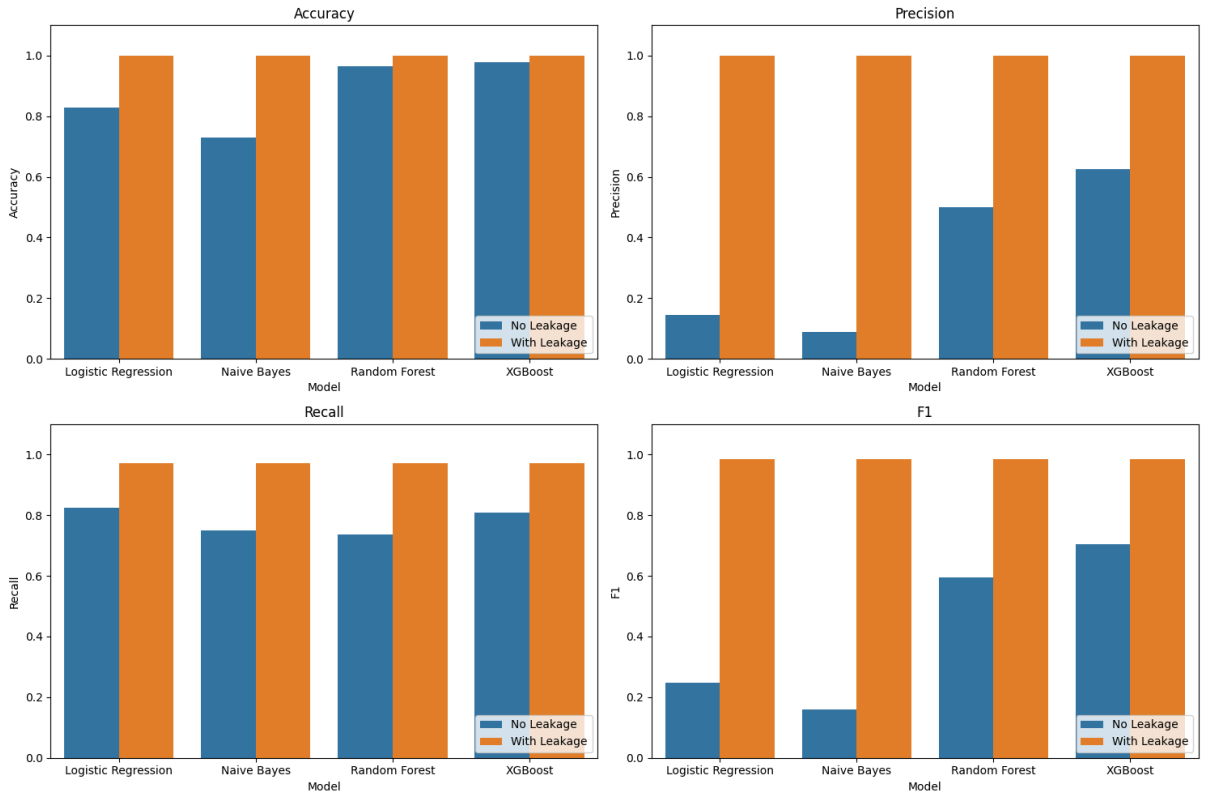
Logistic before	
Random forest	
XGBOOST	

After:

Logistic	
Random forest	
XGBoost	

Results:

```
Performance Comparison:
                 Model      Scenario  Accuracy  Precision    Recall        F1
0  Logistic Regression    No Leakage    0.8295   0.145455  0.823529  0.247241
1  Logistic Regression  With Leakage    0.9990   1.000000  0.970588  0.985075
2          Naive Bayes    No Leakage    0.7305   0.089005  0.750000  0.159126
3          Naive Bayes  With Leakage    0.9990   1.000000  0.970588  0.985075
4        Random Forest    No Leakage    0.9660   0.500000  0.735294  0.595238
5        Random Forest  With Leakage    0.9990   1.000000  0.970588  0.985075
6              XGBoost    No Leakage    0.9770   0.625000  0.808824  0.705128
7              XGBoost  With Leakage    0.9990   1.000000  0.970588  0.985075
```



Data Leakage Different techniques & limitations:

1. Train-Test Performance Comparison

How it works:

- Train your model and check its performance on both the training set and test set
- If the model performs almost perfectly on the training set but poorly on the test set, it might be overfitting (not necessarily leakage).

- If it performs suspiciously well on both, there could be data leakage (e.g., test data influencing training).

Limitation
- Doesn’t tell you where the leakage is coming from.
- Some models (like deep learning) naturally overfit without leakage.

---

2. Check Feature Importance for Illogical Features
How it works:
- Train a model (like Random Forest or XGBoost) and check feature importance.
- If irrelevant features (e.g., IDs, timestamps, future data) have high importance, leakage is likely.

Limitation
- Requires domain knowledge to recognize illogical features.
- Some leakage (e.g., subtle time-based leaks) may not show up in feature importance.

3. Leakage via Time-Based Data Splitting
Technique: Using future data to predict past events (e.g., stock price prediction with test data from before training data).

Limitation: Causes unrealistic model performance since real-world predictions can't rely on future data.