Machine Failure Prediction using Machine Learning

Problem Understanding

The goal of this task is to predict whether a machine operation will fail or not using industrial machine sensor data. Each row in the dataset represents one machine operation and contains information about machine type, temperature, rotational speed, torque, and tool wear.

The target variable is a binary column indicating failure (1) or no failure (0), which makes this a binary classification problem.

```
## Upload & Load Dataset
from google.colab import files
uploaded = files.upload()
```

Choose files Dataset.csv

Dataset.csv(text/csv) - 531016 bytes, last modified: 23/12/2025 - 100% done

```
Saving Dataset.csv to Dataset.csv
```

STEP 3: Dataset Overview

The dataset contains approximately 10,000 records of industrial machine operations.Each row corresponds to a single machine operation and includes machine type, process parameters, and a target column indicating whether the operation failed.

```
import pandas as pd
df = pd.read_csv('Dataset.csv')
df.head()
```

	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

Next steps: Generate code with df New interactive sheet

Dataset Overview

The dataset contains approximately 10,000 records of machine operations. It includes a mix of numerical features, categorical features, and a binary target variable. This overview helps in understanding the structure of the data before applying preprocessing steps.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 10 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   UDI                      10000 non-null  int64
 1   Product ID               10000 non-null  object
 2   Type                     10000 non-null  object
 3   Air temperature [K]      10000 non-null  float64
 4   Process temperature [K]  10000 non-null  float64
 5   Rotational speed [rpm]   10000 non-null  int64
 6   Torque [Nm]              10000 non-null  float64
 7   Tool wear [min]          10000 non-null  int64
 8   Target                   10000 non-null  int64
 9   Failure Type             10000 non-null  object
dtypes: float64(3), int64(4), object(3)
memory usage: 781.4+ KB
```

```
df.columns
```

```
Index(['UDI', 'Product ID', 'Type', 'Air temperature [K]',
       'Process temperature [K]', 'Rotational speed [rpm]', 'Torque [Nm]',
       'Tool wear [min]', 'Target', 'Failure Type'],
      dtype='object')
```

Dataset Summary

The dataset does not contain any missing values, which means no imputation is required. Numerical features such as temperatures, torque, speed, and tool wear are already in suitable format. Some columns are identifiers or descriptive and require further handling during preprocessing.

Data Preprocessing

Before training the model, unnecessary and potentially misleading columns were removed, and categorical features were converted into numerical form. These steps help ensure that the model learns only from meaningful information.

```
# 5.1 : Drop Unnecessary :
df = df.drop(columns=["UDI", "Product ID", "Failure Type"])
df.head()
```

	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target
0	M	298.1	308.6	1551	42.8	0	0
1	L	298.2	308.7	1408	46.3	3	0
2	L	298.1	308.5	1498	49.4	5	0
3	L	298.2	308.6	1433	39.5	7	0

Next steps: Generate code with df New interactive sheet

Removing Unnecessary Columns

The columns `UDI` and `Product ID` were removed because they are unique identifiers and do not contribute to predicting machine failure.

The `Failure Type` column was removed to avoid data leakage, as it directly describes the failure condition and would unfairly influence the model.

```
 ##  5.2 Handle Categorical Feature (Type):

df = pd.get_dummies(df, columns=["Type"], drop_first=True)
df.head()
```

	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target	Type_L	Type_M
0	298.1	308.6	1551	42.8	0	0	False	True
1	298.2	308.7	1408	46.3	3	0	True	False
2	298.1	308.5	1498	49.4	5	0	True	False
3	298.2	308.6	1433	39.5	7	0	True	False

Next steps: Generate code with df New interactive sheet

Encoding Categorical Features

The `Type` column represents machine type and is categorical in nature. One-hot encoding was applied to convert this feature into numerical form, as machine learning models require numerical inputs.

```
## 5.3 Separate Features and Target :

X = df.drop("Target", axis=1)
y = df["Target"]
```

Feature and Target Separation

The dataset was divided into input features (X) and the target variable (y). The target column represents whether a machine operation failed or not.

```
## 5.4 Train-Test Split :
""" The data was split into training and testing sets using an 80:20 ratio."""


from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)
```

Train-Test Split

The data was split into training and testing sets using an 80:20 ratio. Stratified sampling was used to maintain the same proportion of failure and non-failure cases in both sets.

Model Selection

For this task, Logistic Regression was chosen as the machine learning model. This model is suitable for binary classification problems and is easy to interpret. Since the goal is to focus on correct approch and clear explanation rather than complex models

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
```

```
▾    LogisticRegression
LogisticRegression(max_iter=1000)
```

Model Evaluation

The model was evaluated using multiple metrics such as precision, recall, F1-score, and a confusion matrix. These metrics provide better insight than accuracy alone, especially for understanding how well failures are detected.

```
y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:, 1]
```

```
from sklearn.metrics import confusion_matrix, classification_report,
roc_auc_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nROC-AUC Score:")
print(roc_auc_score(y_test, y_prob))
```

```
Confusion Matrix:
[[1927    5]
 [  58   10]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.98      1932
           1       0.67      0.15      0.24        68

    accuracy                           0.97      2000
   macro avg       0.82      0.57      0.61      2000
weighted avg       0.96      0.97      0.96      2000


ROC-AUC Score:
0.8994489099987821
```

```
import pandas as pd

coefficients = pd.Series(model.coef_[0], index=X.columns)
coefficients.sort_values(ascending=False)
```

	0
Air temperature [K]	0.821266
Type_L	0.627183
Torque [Nm]	0.275430
Type_M	0.135478
Tool wear [min]	0.012569
Rotational speed [rpm]	0.011330
Process temperature [K]	-0.907908

dtype: float64

```
### Target Variable Distribution : to Understand balance between failure and non-failure cases.
y.value_counts()
```

Target	count
0	9661
1	339

dtype: int64

```
### Baseline Comparison : which shows model performance was compared against this baseline to understand

baseline_accuracy = y_test.value_counts(normalize=True).max()
baseline_accuracy
```

```
0.966
```

Evaluation Interpretation

Recall is particularly important in this problem because missing a machine failure can lead to high operational costs. Precision is also important to avoid unnecessary maintenance actions caused by false alarms.

Future Improvements

If more time were available, more advanced models such as Random Forests could be explored to capture non-linear relationships. Handling class imbalance and incorporating time-based data could further improve prediction performance.