first , lets import some library for EDA and visualisation of our data

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

let's load the csv file using read_csv

```
df=pd.read_csv('/content/predictive_maintenance.csv')
```

```
df.head()
```

Show hidden output

from DataFrame we can clearly see that 2 colunms UDI and PRODUCT ID has no relation with machine failure so we can exclude this two columns. and we can also rename some columns also

```
df=df.drop(['UDI','Product ID'],axis=1)
```

```
import pandas as pd

df=df.rename(columns={
    'Air temperature [K]':'air_temperature[k]',
    'Process temperature [K]':'process_temperature[k]',
    'Rotational speed [rpm]':'rotational_speed[rpm]',
    'Tool wear [min]':'tool_wear[min]'
})
```

```
df.head()
```

Show hidden output

Double-click (or enter) to edit

exploratory data analysis(EDA)

```
df.info()
```

Show hidden output

```
# Condition 1: Target is 0, but a failure type is present
condition_1 = (df['Target'] == 0) & (df['Failure Type'] != 'No Failure')

# Condition 2: Target is 1, but 'No Failure' is present
condition_2 = (df['Target'] == 1) & (df['Failure Type'] == 'No Failure')

# Combine the conditions using OR
inconsistent_data = df[condition_1 | condition_2]

display(inconsistent_data.shape[0])
```

```
27
```

```
# Drop the inconsistent rows from the original DataFrame df
df = df.drop(inconsistent_data.index)

display(f"Number of rows after dropping inconsistent data: {df.shape[0]}")
```

```
'Number of rows after dropping inconsistent data: 9973'
```

```
df.describe()
```

Show hidden output

by seeing description of data we see that in 3 columns rotational_speed , torque,tool wear there is presence of outliar so lets see how the outliar affecting the output

```
df.loc[(df['rotational_speed[rpm]'] < df['rotational_speed[rpm]'].max()) & (df['rotational_speed[rpm]'] > df['rotational_sp
```

Show hidden output

```
df.loc[(df['Torque [Nm]']<df['Torque [Nm]'].max()) & (df['Torque [Nm]']>df['Torque [Nm]'].max()*0.90)]
```

Show hidden output

```
df.loc[(df['tool_wear[min]']<df['tool_wear[min]'].max()) & (df['tool_wear[min]']>df['tool_wear[min]'].max()*0.90)]
```

Show hidden output

```
display(df['Failure Type'].unique())
```

```
array(['No Failure', 'Power Failure', 'Tool Wear Failure',
       'Overstrain Failure', 'Heat Dissipation Failure'], dtype=object)
```

```
plt.figure(figsize=(10, 6))
ax = sns.histplot(data=df, y='Failure Type')
plt.title('Distribution of Failure Type after cleaning')

# Add annotations to the bars
for container in ax.containers:
    ax.bar_label(container, fmt='%d', label_type='edge')

plt.show()
```

Show hidden output

```
# Get the counts of all failure types
failure_counts = df['Failure Type'].value_counts()

# Filter out 'No Failure' to show only actual failure types
failure_types_only = failure_counts[failure_counts.index != 'No Failure']

# Check if there are any actual failure types to plot
if not failure_types_only.empty:
    # Create the pie chart for actual failure types
    plt.figure(figsize=(10, 8))
    plt.pie(failure_types_only, labels=failure_types_only.index, autopct='%1.1f%%', startangle=90, wedgeprops={'edgecolor'
    plt.title('Percentage of Each Failure Type (Excluding No Failure)')
    plt.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.
    plt.show()
else:
    print("No specific failure types found to plot (all were 'No Failure').")
```

Show hidden output

```
# Select numerical columns for correlation calculation, including the 'Target' column.
numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns

# Calculate the correlation matrix
correlation_matrix = df[numerical_cols].corr()

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Numerical Features and Target')
plt.show()
```

Show hidden output

```
# Select only numerical columns
numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns

# Plot histograms for each numerical column
plt.figure(figsize=(15, 10))
```

```
for i, col in enumerate(numerical_cols):
    plt.subplot(len(numerical_cols) // 2 + 1, 2, i + 1)
    sns.histplot(df[col], kde=True)
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```

Show hidden output

```
# Convert 'Type' column data to specified numerical labels
type_mapping = {'L': 0, 'M': 1, 'H': 2}
df['Type'] = df['Type'].map(type_mapping)

# Convert 'Failure Type' column data to specified numerical labels
failure_type_mapping = {
    'No Failure': 0,
    'Power Failure': 1,
    'Overstrain Failure': 2,
    'Heat Dissipation Failure': 3,
    'Tool Wear Failure': 4
}
df['Failure Type'] = df['Failure Type'].map(failure_type_mapping)

display("DataFrame after label encoding 'Type' and 'Failure Type' columns:")
display(df.head())
display(df.dtypes)
```

Show hidden output

```
from sklearn.model_selection import train_test_split

# Define features (X) and target (y)
# Exclude 'Target' and all 'Failure Type_' one-hot encoded columns to prevent data leakage
# The model should predict 'Target' based on other features, not based on encoded failure types
X = df.drop(columns=['Target','Failure Type'])
y = df[['Target','Failure Type']]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

display(f"X_train shape: {X_train.shape}")
display(f"X_test shape: {X_test.shape}")
display(f"y_train shape: {y_train.shape}")
display(f"y_test shape: {y_test.shape}")
```

```
'X_train shape: (7978, 6)'
'X_test shape: (1995, 6)'
'y_train shape: (7978, 2)'
'y_test shape: (1995, 2)'
```

```
from sklearn.preprocessing import StandardScaler

# Identify numerical columns to scale
# Exclude 'Target' and one-hot encoded 'Failure Type' columns as they are either target or binary labels
columns_to_scale = [
    'air_temperature[k]',
    'process_temperature[k]',
    'rotational_speed[rpm]',
    'Torque [Nm]',
    'tool_wear[min]'
]

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit the scaler only on X_train and transform X_train
X_train[columns_to_scale] = scaler.fit_transform(X_train[columns_to_scale])

# Transform X_test using the same fitted scaler
X_test[columns_to_scale] = scaler.transform(X_test[columns_to_scale])

display("X_train after Standard Scaling:")
display(X_train.head())
display("X_test after Standard Scaling:")
display(X_test.head())
```

Show hidden output

model training and evaluation

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

# Define a dictionary of models
models = {
    'Logistic Regression': LogisticRegression(random_state=42, solver='liblinear'),
    'K-Nearest Neighbors': KNeighborsClassifier(),
    'Support Vector Machine': SVC(random_state=42,probability=True),
    'Random Forest Classifier': RandomForestClassifier(random_state=42),


}

print("Models dictionary created successfully.")
```

```
Models dictionary created successfully.
```

accuracy of binary classification model

```
from sklearn.metrics import accuracy_score

accuracy_results_bn = []

for name, model in models.items():
    # Train the model
    model.fit(X_train, y_train['Target'])

    # Make predictions on the scaled test set
    y_pred = model.predict(X_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test['Target'], y_pred)
    accuracy_results_bn.append({'Model': name, 'Accuracy': accuracy})

print("Models trained and evaluated successfully.")

# Create a DataFrame from the accuracy results
accuracy_df_bn = pd.DataFrame(accuracy_results_bn)

# Sort the DataFrame by accuracy in descending order
accuracy_df_bn = accuracy_df_bn.sort_values(by='Accuracy', ascending=False)

print("accuracy DataFrame of binaray clf :")
display(accuracy_df_bn)
```

Show hidden output

```
accuracy_results_cat = []

for name, model in models.items():
    # Train the model
    model.fit(X_train, y_train['Failure Type'])

    # Make predictions on the scaled test set
    y_pred = model.predict(X_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test['Failure Type'], y_pred)
    accuracy_results_cat.append({'Model': name, 'Accuracy': accuracy})

print("Models trained and evaluated successfully.")

# Create a DataFrame from the accuracy results
accuracy_df_cat = pd.DataFrame(accuracy_results_cat)

# Sort the DataFrame by accuracy in descending order
accuracy_df_cat = accuracy_df_cat.sort_values(by='Accuracy', ascending=False)

print("accuracy DataFrame of categorical clf :")
display(accuracy_df_bn)
```

```
Models trained and evaluated successfully.
accuracy DataFrame of categorical clf :
```

	Model	Accuracy
3	Random Forest Classifier	0.986466
1	K-Nearest Neighbors	0.978446
2	Support Vector Machine	0.977444
0	Logistic Regression	0.969424

Evaluating Binary Classification (Target) with Confusion Matrix, F1, and F2 Scores

```
from sklearn.metrics import confusion_matrix, f1_score, fbeta_score

results_binary_advanced = []

for name, model in models.items():
    print(f"\n--- Model: {name} (Binary Classification) ---")

    # Re-train the model to ensure it's predicting 'Target'
    # (Models were already trained in the accuracy evaluation cell, but good practice to ensure context)
    model.fit(X_train, y_train['Target'])
    y_pred = model.predict(X_test)

    # Confusion Matrix
    cm = confusion_matrix(y_test['Target'], y_pred)
    print("Confusion Matrix:\n", cm)

    # F1 Score (harmonic mean of precision and recall)
    f1 = f1_score(y_test['Target'], y_pred, average='binary')
    print(f"F1 Score: {f1:.4f}")

    # F2 Score (emphasizes recall more than precision, beta=2)
    f2 = fbeta_score(y_test['Target'], y_pred, beta=2, average='binary')
    print(f"F2 Score: {f2:.4f}")

    results_binary_advanced.append({
        'Model': name,
        'F1 Score': f1,
        'F2 Score': f2
    })

df_binary_advanced = pd.DataFrame(results_binary_advanced)
display(df_binary_advanced.sort_values(by='F1 Score', ascending=False))
```

```
--- Model: Logistic Regression (Binary Classification) ---
Confusion Matrix:
 [[1925    4]
 [  57    9]]
F1 Score: 0.2278
F2 Score: 0.1625

--- Model: K-Nearest Neighbors (Binary Classification) ---
Confusion Matrix:
 [[1927    2]
 [  41   25]]
F1 Score: 0.5376
F2 Score: 0.4296

--- Model: Support Vector Machine (Binary Classification) ---
Confusion Matrix:
 [[1928    1]
 [  44   22]]
F1 Score: 0.4944
F2 Score: 0.3833

--- Model: Random Forest Classifier (Binary Classification) ---
Confusion Matrix:
 [[1924    5]
 [  22   44]]
F1 Score: 0.7652
F2 Score: 0.7029
```

	Model	F1 Score	F2 Score
3	Random Forest Classifier	0.765217	0.702875
1	K-Nearest Neighbors	0.537634	0.429553
2	Support Vector Machine	0.494382	0.383275
0	Logistic Regression	0.227848	0.162455

Evaluating Categorical Classification (Failure Type) with Confusion Matrix, F1, and F2 Scores

```python
from sklearn.metrics import confusion_matrix, f1_score, fbeta_score

results_categorical_advanced = []

for name, model in models.items():
    print(f"\n--- Model: {name} (Categorical Classification) ---")

    # Re-train the model to ensure it's predicting 'Failure Type'
    model.fit(X_train, y_train['Failure Type'])
    y_pred = model.predict(X_test)

    # Confusion Matrix
    cm = confusion_matrix(y_test['Failure Type'], y_pred)
    print("Confusion Matrix:\n", cm)

    # F1 Score (weighted average due to potential class imbalance)
    f1 = f1_score(y_test['Failure Type'], y_pred, average='weighted')
    print(f"F1 Score (weighted): {f1:.4f}")

    # F2 Score (emphasizes recall more, weighted average)
    f2 = fbeta_score(y_test['Failure Type'], y_pred, beta=2, average='weighted')
    print(f"F2 Score (weighted): {f2:.4f}")

    results_categorical_advanced.append({
        'Model': name,
        'F1 Score (weighted)': f1,
        'F2 Score (weighted)': f2
    })

df_categorical_advanced = pd.DataFrame(results_categorical_advanced)
display(df_categorical_advanced.sort_values(by='F1 Score (weighted)', ascending=False))
```

```
--- Model: Logistic Regression (Categorical Classification) ---
Confusion Matrix:
 [[1929    0    0    0    0]
 [  10    9    0    0    0]
 [  14    0    1    1    0]
 [  21    0    0    1    0]
 [   9    0    0    0    0]]
F1 Score (weighted): 0.9616
F2 Score (weighted): 0.9678

--- Model: K-Nearest Neighbors (Categorical Classification) ---
Confusion Matrix:
 [[1928    0    0    1    0]
 [  10    9    0    0    0]
 [   7    0    9    0    0]
 [  18    1    0    3    0]
 [   9    0    0    0    0]]
F1 Score (weighted): 0.9700
F2 Score (weighted): 0.9739

--- Model: Support Vector Machine (Categorical Classification) ---
Confusion Matrix:
 [[1929    0    0    0    0]
 [  10    9    0    0    0]
 [   8    1    7    0    0]
 [  21    0    1    0    0]
 [   9    0    0    0    0]]
F1 Score (weighted): 0.9656
F2 Score (weighted): 0.9710

--- Model: Random Forest Classifier (Categorical Classification) ---
Confusion Matrix:
 [[1926    0    1    2    0]
 [   8   11    0    0    0]
 [   6    1    9    0    0]
 [   4    1    0   17    0]
 [   8    0    1    0    0]]
F1 Score (weighted): 0.9807
F2 Score (weighted): 0.9826
```

	Model	F1 Score (weighted)	F2 Score (weighted)
3	Random Forest Classifier	0.980731	0.982575
1	K-Nearest Neighbors	0.969994	0.973852
2	Support Vector Machine	0.965625	0.971011
0	Logistic Regression	0.961555	0.967805

Model Selection Conclusion

After a thorough evaluation of all models, considering accuracy, confusion matrices, and the crucial F1 and F2 scores for this critical machine failure detection task, the **Random Forest Classifier** has demonstrated superior performance. Its robust results across both binary and categorical classification make it the most suitable choice. We will therefore proceed with the Random Forest Classifier for further analysis and implementation.

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# 1. Define a parameter grid for the RandomForestClassifier
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_features': ['sqrt', 'log2'],
    'max_depth': [10, 20, 30, None],
    'min_samples_leaf': [1, 2, 4]
}

# 2. Create an instance of RandomForestClassifier
rf_classifier = RandomForestClassifier(random_state=42)

print("Starting hyperparameter tuning for binary classification (Target)...")

# 3. Initialize GridSearchCV for the binary classification task ('Target')
grid_search_binary = GridSearchCV(
    estimator=rf_classifier,
    param_grid=param_grid,
    scoring='f1_weighted', # Use f1_weighted for better handling of class imbalance for Target
    cv=5,
    n_jobs=-1, # Use all available processors
    verbose=2
)

# 4. Fit the GridSearchCV object to the training data for binary classification
grid_search_binary.fit(X_train, y_train['Target'])

# 5. Print the best_params_ and best_score_ for binary classification
print("\nBest parameters for Binary Classification (Target):", grid_search_binary.best_params_)
print("Best F1 Score (weighted) for Binary Classification (Target):", grid_search_binary.best_score_)

# 6. Store the best binary classification model
best_rf_binary_model = grid_search_binary.best_estimator_
print("Best binary classification model stored.")

print("\nStarting hyperparameter tuning for categorical classification (Failure Type)...")

# 7. Initialize another GridSearchCV object for the categorical classification task ('Failure Type')
grid_search_categorical = GridSearchCV(
    estimator=rf_classifier,
    param_grid=param_grid,
    scoring='f1_weighted', # Use f1_weighted for multi-class classification
    cv=5,
    n_jobs=-1,
    verbose=2
)

# 8. Fit this GridSearchCV object to the training data for categorical classification
grid_search_categorical.fit(X_train, y_train['Failure Type'])

# 9. Print the best_params_ and best_score_ for categorical classification
print("\nBest parameters for Categorical Classification (Failure Type):", grid_search_categorical.best_params_)
print("Best F1 Score (weighted) for Categorical Classification (Failure Type):", grid_search_categorical.best_score_)

# 10. Store the best categorical classification model
best_rf_categorical_model = grid_search_categorical.best_estimator_
print("Best categorical classification model stored.")
```

```
Starting hyperparameter tuning for binary classification (Target)...
Fitting 5 folds for each of 72 candidates, totalling 360 fits

Best parameters for Binary Classification (Target): {'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'n_esti
Best F1 Score (weighted) for Binary Classification (Target): 0.9840017357079753
Best binary classification model stored.

Starting hyperparameter tuning for categorical classification (Failure Type)...
Fitting 5 folds for each of 72 candidates, totalling 360 fits

Best parameters for Categorical Classification (Failure Type): {'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf':
Best F1 Score (weighted) for Categorical Classification (Failure Type): 0.9806110221763242
Best categorical classification model stored.
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Extract feature importances from best_rf_binary_model
feature_importances_binary = best_rf_binary_model.feature_importances_
feature_names = X_train.columns

# Create a DataFrame for binary classification feature importances
importance_df_binary = pd.DataFrame({
    'Feature': feature_names,
    'Importance': feature_importances_binary
})

# Sort in descending order of importance
importance_df_binary = importance_df_binary.sort_values(by='Importance', ascending=False)

print("Feature Importances for Binary Classification (Target):")
display(importance_df_binary)

# 2. Create a bar plot for binary classification feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=importance_df_binary, palette='viridis', hue='Feature', legend=False)
plt.title('Feature Importances for Binary Classification (Target)')
plt.xlabel('Importance Score')
plt.ylabel('Feature')
plt.show()

# 3. Extract feature importances from best_rf_categorical_model
feature_importances_categorical = best_rf_categorical_model.feature_importances_

# Create a DataFrame for categorical classification feature importances
importance_df_categorical = pd.DataFrame({
    'Feature': feature_names,
    'Importance': feature_importances_categorical
})

# Sort in descending order of importance
importance_df_categorical = importance_df_categorical.sort_values(by='Importance', ascending=False)

print("\nFeature Importances for Categorical Classification (Failure Type):")
display(importance_df_categorical)

# 4. Create a bar plot for categorical classification feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=importance_df_categorical, palette='magma', hue='Feature', legend=False)
plt.title('Feature Importances for Categorical Classification (Failure Type)')
plt.xlabel('Importance Score')
plt.ylabel('Feature')
plt.show()
```

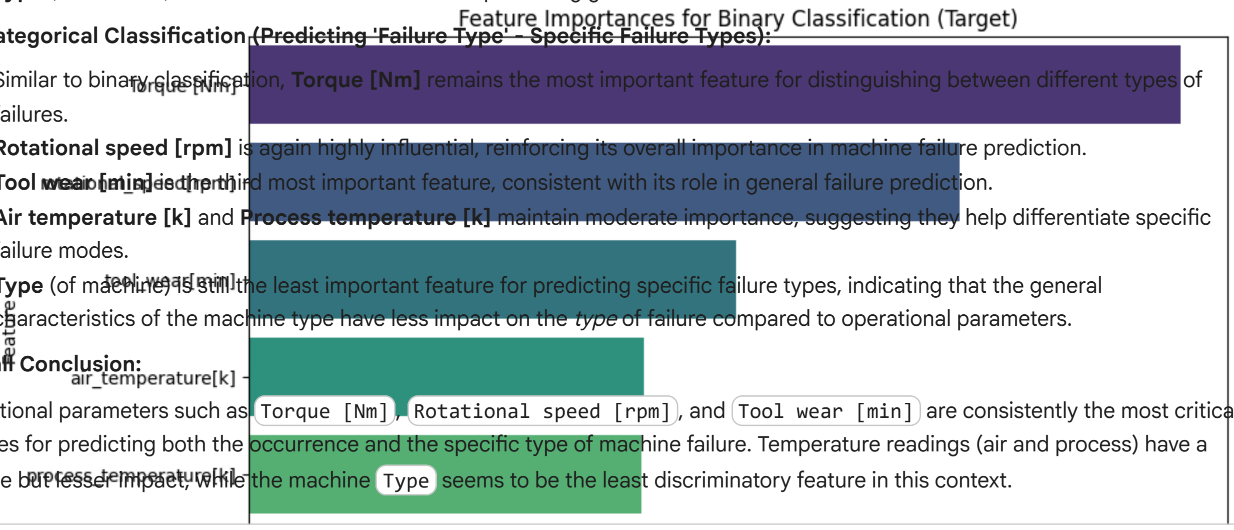
```
Feature Importances for Binary Classification (Target):
```

	Feature	Importance
4	Torque [Nm]	0.311778
3	rotational_speed[rpm]	0.237875
5	tool_wear[min]	0.163157
1	air_temperature[k]	0.132135
2	process_temperature[k]	0.131220
0	Type	0.023895

Key Insights from Feature Importance Analysis

For Binary Classification (Predicting 'Target' - Any Failure vs. No Failure):

1. **Torque [Nm]** is the most influential feature, significantly contributing to the prediction of whether a failure occurs or not.
2. **Rotational speed [rpm]** is the second most important feature, indicating its strong relationship with general machine failure.
3. **Tool wear [min]** also plays a substantial role.
4. **Air temperature [k]** and **Process temperature [k]** have moderate importance.
5. **Type** (of machine) is the least influential feature in predicting general failure.


Feature Importances for Binary Classification (Target)

For Categorical Classification (Predicting 'Failure Type' - Specific Failure Types):

1. Similar to binary classification, **Torque [Nm]** remains the most important feature for distinguishing between different types of failures.
2. **Rotational speed [rpm]** is again highly influential, reinforcing its overall importance in machine failure prediction.
3. **Tool wear [min]** is the third most important feature, consistent with its role in general failure prediction.
4. **Air temperature [k]** and **Process temperature [k]** maintain moderate importance, suggesting they help differentiate specific failure modes.
5. **Type** (of machine) is still the least important feature for predicting specific failure types, indicating that the general characteristics of the machine type have less impact on the *type* of failure compared to operational parameters.

Overall Conclusion:

Operational parameters such as `Torque [Nm]`, `Rotational speed [rpm]`, and `Tool wear [min]` are consistently the most critical features for predicting both the occurrence and the specific type of machine failure. Temperature readings (air and process) have a notable but lesser impact, while the machine `Type` seems to be the least discriminatory feature in this context.

```
import joblib

# Save the best binary classification model
joblib.dump(best_rf_binary_model, 'best_rf_binary_model.joblib')
print("best_rf_binary_model saved successfully as 'best_rf_binary_model.joblib'")

# Save the best categorical classification model
joblib.dump(best_rf_categorical_model, 'best_rf_categorical_model.joblib')
print("best_rf_categorical_model saved successfully as 'best_rf_categorical_model.joblib'")
```

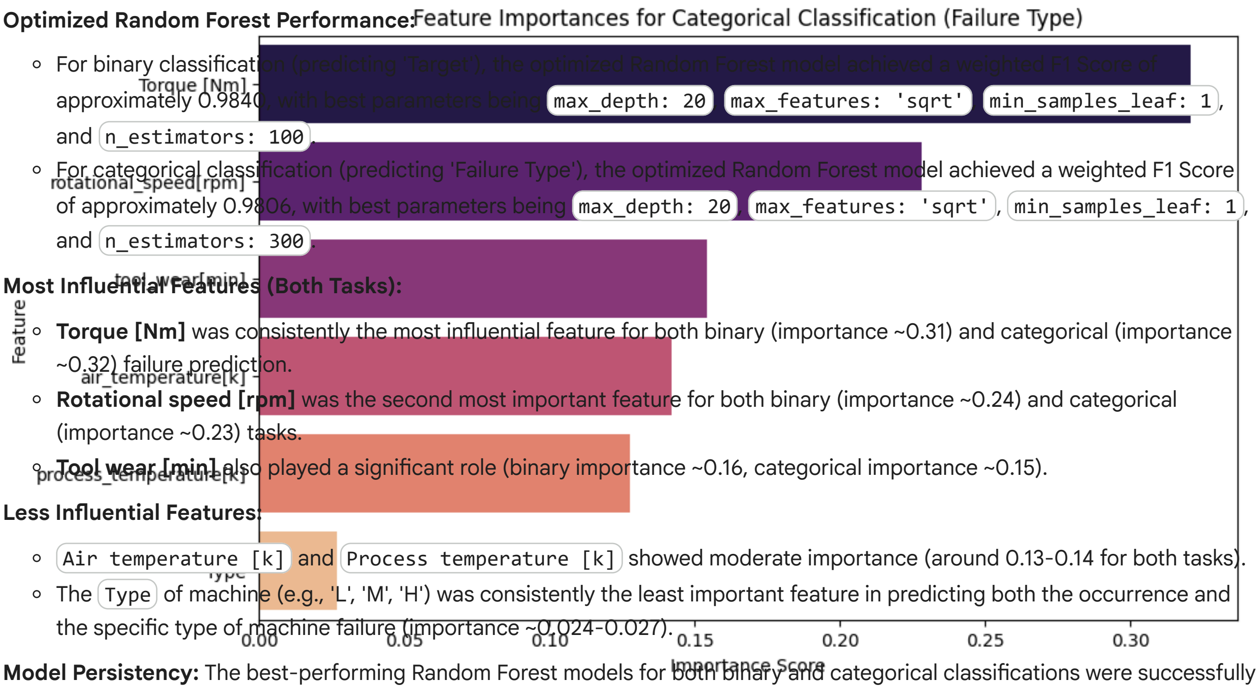
```
best_rf_binary_model saved successfully as 'best_rf_binary_model.joblib'
best_rf_categorical_model saved successfully as 'best_rf_categorical_model.joblib'
```

	Feature	Importance
3	rotational_speed[rpm]	0.228047
5	tool_wear[min]	0.154222
1	air_temperature[k]	0.141920
2	process_temperature[k]	0.127962
0	Type	0.026970

Summary:

Data Analysis Key Findings

* **Optimized Random Forest Performance:**
  * For binary classification (predicting 'Target'), the optimized Random Forest model achieved a weighted F1 Score of approximately 0.9840, with best parameters being `max_depth: 20`, `max_features: 'sqrt'`, `min_samples_leaf: 1`, and `n_estimators: 100`.
  * For categorical classification (predicting 'Failure Type'), the optimized Random Forest model achieved a weighted F1 Score of approximately 0.9806, with best parameters being `max_depth: 20`, `max_features: 'sqrt'`, `min_samples_leaf: 1`, and `n_estimators: 300`.
* **Most Influential Features (Both Tasks):**
  * **Torque [Nm]** was consistently the most influential feature for both binary (importance ~0.31) and categorical (importance ~0.32) failure prediction.
  * **Rotational speed [rpm]** was the second most important feature for both binary (importance ~0.24) and categorical (importance ~0.23) tasks.
  * **Tool wear [min]** also played a significant role (binary importance ~0.16, categorical importance ~0.15).
* **Less Influential Features:**
  * `Air temperature [k]` and `Process temperature [k]` showed moderate importance (around 0.13-0.14 for both tasks).
  * The `Type` of machine (e.g., 'L', 'M', 'H') was consistently the least important feature in predicting both the occurrence and the specific type of machine failure (importance ~0.024-0.027).
* **Model Persistency:** The best-performing Random Forest models for both binary and categorical classifications were successfully saved, allowing for future deployment without retraining.


Feature Importances for Categorical Classification (Failure Type)