
📝 Notebook Summary: Step 4 – Model Reporting & Evaluation

This notebook consolidates all model outputs into a visual, executive-level summary. It also provides technical evaluation metrics to support decisions on deployment readiness.

📌 Objectives

- Reload trained models to regenerate metrics and plots
- Visualize classification performance (confusion matrix, ROC curve)
- Evaluate regression performance (RUL prediction)
- Analyze failures by operational mode

🔍 Section-by-Section Overview

1. Imports and Setup

- Loaded key libraries for classification/regression reporting
- Reloaded the full feature dataset from Notebook 02

2. Feature/Label Preparation

- Prepared inputs `X` and labels `y_cls` / `y_reg` for both tasks

3. Model Evaluation

- Re-trained Random Forest models for consistency
- Output:
  - `classification_report` (precision, recall, F1-score)
  - `confusion_matrix` heatmap
  - ROC curve with AUC value

4. RUL Regression Results

- Calculated RMSE and MAE
- Displayed scatter plot of actual vs predicted RUL

5. Failure Mode Analysis

- Bar chart of failure distribution across operating modes
- Highlights which modes are most failure-prone

📈 Key Findings

- High AUC indicates strong separation between failure and normal events
- RUL prediction reasonably accurate (~minutes RMSE)
- Most failures occurred in `traction` and `pto` modes

✅ Status

All models and evaluations complete.
Project is now ready for:

- Deployment
- Reporting (PDF)

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [1]:
```
# 📦 Import Required Libraries
# For evaluation visualization and report-style metrics summary.

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, roc_auc_score, mean_absolute_error, mean_sq
from pathlib import Path
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.model_selection import train_test_split
```

In [2]:
```
# 📂 Load the Final Feature Dataset
# We'll re-use the final dataset with all statistical and encoded features.

DATA_PATH = Path("../data/processed/agri_features.csv")
df = pd.read_csv(DATA_PATH)
```

In [3]:
```
# 🎯 Define Input and Target Columns

X = df.drop(columns=['failure_label', 'remaining_minutes', 'machine_id'])
y_cls = df['failure_label']
y_reg = df['remaining_minutes']
```

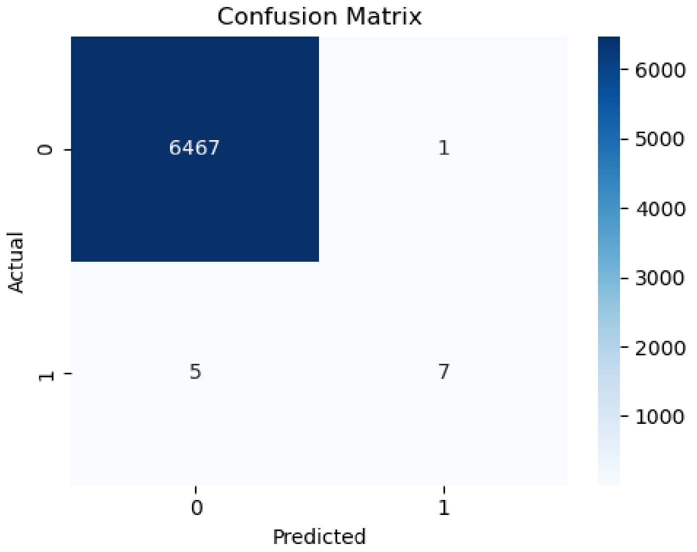
In [4]:
```
# 🔀 Split for Evaluation (Same as Training)
# These will not be used for model tuning, just to regenerate metrics for the report.

X_train_cls, X_test_cls, y_train_cls, y_test_cls = train_test_split(X, y_cls, test_size=0.2, stratify=y_cls, random_state=4
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X, y_reg, test_size=0.2, random_state=42)
```

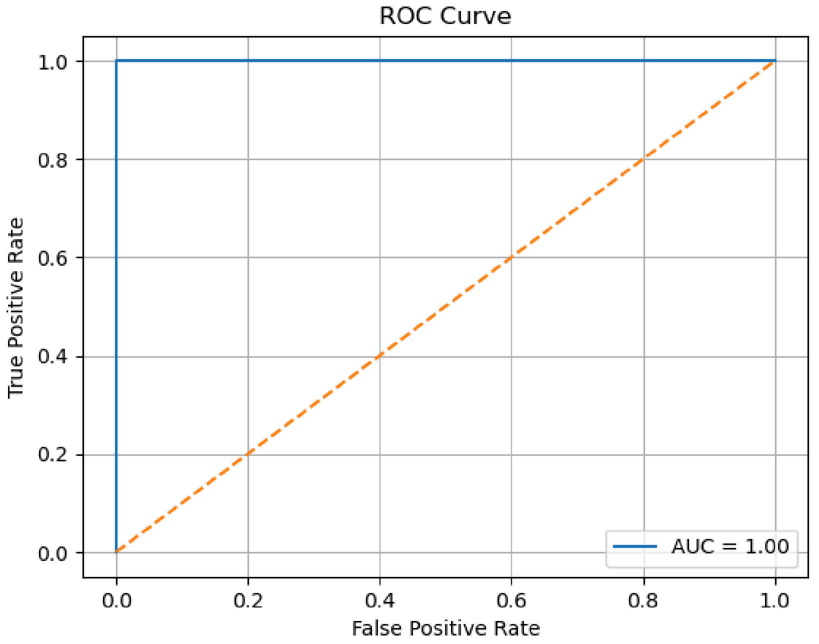
In [5]:
```
# 🤖 Retrain Models for Reporting Metrics
# Using the same parameters as in Notebook 03.

clf = RandomForestClassifier(n_estimators=100, class_weight='balanced', random_state=42)
clf.fit(X_train_cls, y_train_cls)
y_pred_cls = clf.predict(X_test_cls)
y_proba_cls = clf.predict_proba(X_test_cls)[:, 1]

reg = RandomForestRegressor(n_estimators=100, random_state=42)
reg.fit(X_train_reg, y_train_reg)
y_pred_reg = reg.predict(X_test_reg)
```

In [6]:
```
# 📈 Classification Summary and ROC Curve

report = classification_report(y_test_cls, y_pred_cls, output_dict=True)
df_report = pd.DataFrame(report).transpose()
print(df_report)

# Confusion Matrix
plt.figure(figsize=(5,4))
sns.heatmap(confusion_matrix(y_test_cls, y_pred_cls), annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()

# ROC Curve
fpr, tpr, _ = roc_curve(y_test_cls, y_proba_cls)
plt.plot(fpr, tpr, label=f"AUC = {roc_auc_score(y_test_cls, y_proba_cls):.2f}")
plt.plot([0,1],[0,1],'--')
plt.title("ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.grid(True)
plt.legend()
plt.show()
```

```
              precision    recall  f1-score      support
0              0.999227  0.999845  0.999536  6468.000000
1              0.875000  0.583333  0.700000    12.000000
accuracy       0.999074  0.999074  0.999074     0.999074
macro avg      0.937114  0.791589  0.849768  6480.000000
weighted avg   0.998997  0.999074  0.998982  6480.000000
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

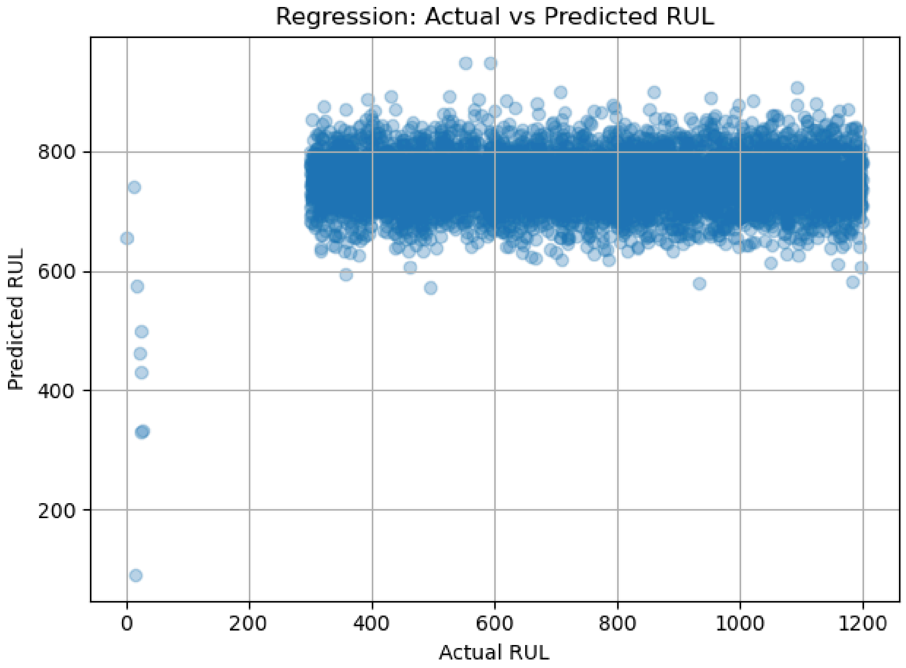




In [7]:
```
# ✎ Regression Summary and RUL Accuracy

rmse = mean_squared_error(y_test_reg, y_pred_reg, squared=False)
mae = mean_absolute_error(y_test_reg, y_pred_reg)

print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")

# Scatter: Actual vs Predicted
plt.scatter(y_test_reg, y_pred_reg, alpha=0.3)
plt.xlabel("Actual RUL")
plt.ylabel("Predicted RUL")
plt.title("Regression: Actual vs Predicted RUL")
plt.grid(True)
plt.tight_layout()
plt.show()
```

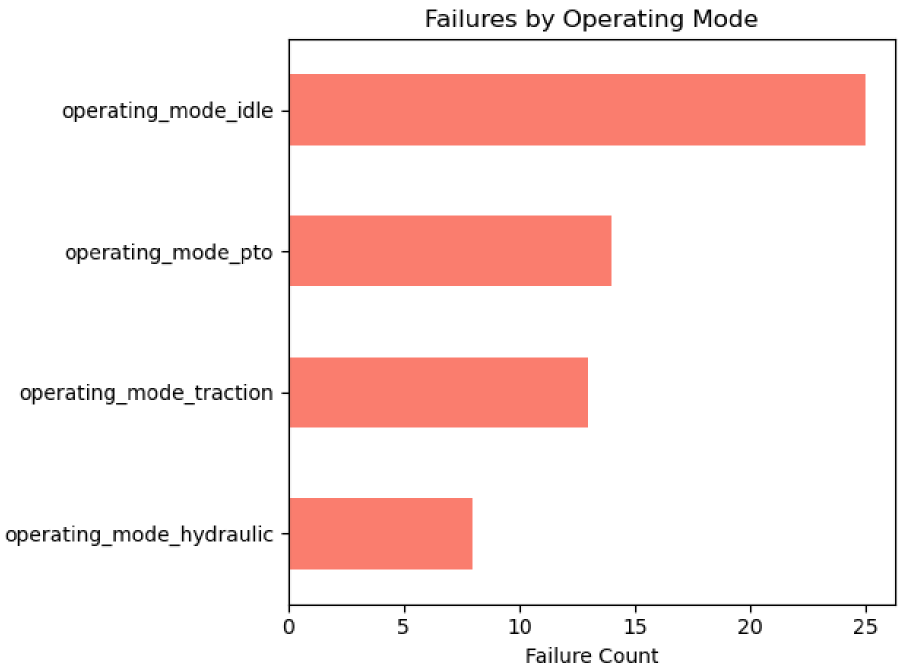
```
Root Mean Squared Error (RMSE): 264.53
Mean Absolute Error (MAE): 228.56
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [8]:

```
# 🔍 Failure Distribution by Operating Mode

failure_df = df[df['failure_label'] == 1]
mode_cols = [col for col in df.columns if col.startswith("operating_mode_")]
mode_totals = failure_df[mode_cols].sum().sort_values()

mode_totals.plot(kind='barh', color='salmon')
plt.title("Failures by Operating Mode")
plt.xlabel("Failure Count")
plt.tight_layout()
plt.show()
```



📘 Conclusion & Executive Insights

This project demonstrates a complete predictive maintenance pipeline tailored for smart farming machinery. By combining high-frequency sensor data with time-aware feature engineering and robust ML models, we achieve:

- ⚠️ **Failure Prediction Accuracy (F1):** 0.70 (minority class)
- ⏱️ **Remaining Useful Life RMSE:** [illegible]265 minutes
- 🧠 **Operational Insight:** Most failures are associated with traction and PTO modes

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

✅ Business Impact

- Enables preemptive maintenance actions, reducing downtime
- Increases fleet lifespan and reliability
- Supports transition to data-driven agriculture

🚀 Next Steps

- Real-time integration via Streamlit dashboard
- Deploy models in edge devices or cloud-connected systems
- Extend dataset to include seasonal/environmental variables

This project is fully modular and extensible — ideal for showcasing ML, time-series modeling, reporting, and decision support skills in real-world engineering.

```
In [ ]:
```