Predictive Maintenance for HVAC Pumps Using Pump Sensor Data

1. Data Exploration and Preprocessing:

Dataset Overview:

- **Total Rows:** 220320
- **Total Columns:** 55
- **Target Variable:** machine_status
  - Indicates machine condition: **NORMAL, RECOVERING, BROKEN**
- **Features:**
  - 52 sensor columns (sensor_00 to sensor_51)
  - timestamp (date-time of reading)

Initial Exploration Findings:

- Missing Values Detected in multiple sensor columns
- **Duplicated Rows:** 0 (No duplicates Found)

Preprocessing Steps:

- sensor_15 dropped
  100% missing values (220,320 rows)

Handled Remaining Missing Values

- Filled missing values in all other sensor columns by using Mean - Ensures no and retains dataset size

Checked Final Missing Values

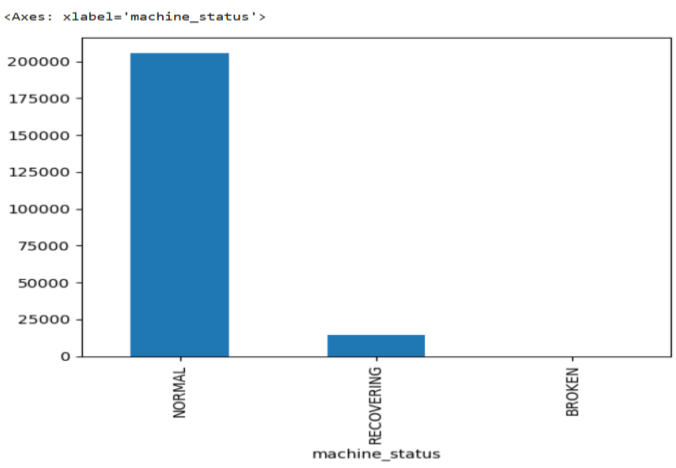
- No missing values remain in the dataset

Class Distribution of Target Variable:

- Normal - 205,836
- Recovering - 14,477
- Broken – 7

Visualization:

A simple bar plot of machine_status distribution



Summary:

- Dropped column: sensor_15 (100% missing)
- Filling missing values in sensors with mean
- Dataset is now complete (no nulls) and ready for feature engineering

2. Feature Engineering:

Final Newly Added Feature List:

- **sensor_mean**
  - **Justification:** Summarizes the average sensor reading to detect overall shifts in the data.
- **sensor_std**
  - **Justification:** Measures the variability of sensor readings to flag unusual fluctuations.
- **sensor_min**
  - **Justification:** Captures the lowest sensor reading, which can help detect sudden drops.
- **sensor_max**
  - **Justification:** Captures the highest sensor reading, useful for detecting sudden spikes.
- **hour**
  - **Justification:** Reflects time-of-day effects on machine behaviour and performance.
- **day_of_week**
  - **Justification:** Captures any weekly patterns in sensor data, reflecting daily operational or maintenance cycles.
- **Sensors Dropped Due to High Correlation**
  - **Dropped Sensors:** Dropped sensor_16 to sensor_26
  - **Reason:** *Removed redundant features to improve accuracy.*
- **Summary of Feature Engineering**
  - **Time Features Added:** hour and day_of_week

    Added to capture time-based trends.
  - **Statistical Features Added:** mean, std, min, max

    Summarize overall sensor data behaviour.
  - **Correlation-Based Feature Removal:** removal of 11 sensors

    Reduced redundancy and overfitting risk.

- **Outcome:** The dataset now has 47 useful features enhancing the model's predictive power.

- **Final Dataset Shape**
  - **Rows:** 220,320
- **Final Columns:** 48 features

3. Model Development:

```
Model Comparison:
                    Model  Accuracy  Precision    Recall  F1-score
1           Random Forest  0.999909   0.999864  0.999909  0.999887
4     K-Nearest Neighbors  0.999750   0.999705  0.999750  0.999728
3  Support Vector Machine  0.999750   0.999705  0.999750  0.999728
2       Gradient Boosting  0.999728   0.999705  0.999728  0.999716
0     Logistic Regression  0.998207   0.998159  0.998207  0.998183
```

Best Model Discussion

- **Top Performer**
  - Random Forest achieved the highest scores across all metrics (Accuracy, Precision, Recall, F1-score).
- **Balance of Metrics**
  - High Precision means very few false alarms.
  - High Recall means almost no overlooked failures.
  - High F1-score confirms overall robustness.
- **Robustness & Interpretability**
  - Random Forest's ensemble of trees reduces overfitting risk.
  - Built-in feature importance helps explain which sensors matter most.
- **Recommendation**
  - Use Random Forest as the primary predictive model.

4. Model Development:

Model Saving & API Model Deployment (Random Forest)

- Selected Random Forest as best model
- Saved model as random_forest_model.pkl (joblib)

- Installed dependencies using pip install command
- Created FastAPI app for model prediction
- Defined /predict endpoint to accept JSON input
- Validated input using Pydantic BaseModel
- Loaded saved model inside API script
- Saved FastAPI app script as app.py
- Navigated to project folder using:
  cd C:\Users\52anu\Downloads\Project Folder
- Started server using:
  uvicorn app:app --reload
- Successfully started API server at http://127.0.0.1:8000
- Verified API and tested predictions at http://127.0.0.1:8000/docs

Summary:
The Random Forest model was saved, integrated into a FastAPI application, and successfully deployed as an API — ready to serve real-time predictions with a simple POST request.

5. Recommendations *(Expanded)*

Model Deployment & Monitoring

- Deploy the Random Forest model as the core machine status prediction engine via the existing API.
- Set up automated model monitoring to track:
  - Prediction accuracy over time
  - Frequency of each predicted class (NORMAL, RECOVERING, BROKEN)
  - Drift in input features (sensor readings) — to detect changing data patterns.
- **Implement alert system:**
  - Notify maintenance teams in real-time when model predicts RECOVERING or BROKEN status.
  - Escalate BROKEN predictions via SMS, email, or integrated dashboards.

Data & Model Maintenance

- Schedule regular retraining of the model (e.g., monthly or quarterly) as more machine data is collected.

- Collect more BROKEN samples (if possible) to address severe class imbalance.
    - Explore SMOTE or anomaly detection models to better detect rare failures.
- Audit model predictions periodically to ensure performance hasn't degraded.

C. Feature & Insights Utilization

- **Create dashboards to visualize:**
    - Feature importances (key sensors influencing predictions)
    - Time-based trends in sensor anomalies and machine status (Example: sensor_mean trends by hour/day)
- **Prioritize critical sensors identified by feature importance for:**
    - Preventive maintenance
    - Sensor calibration and quality checks

D. API & System Robustness

- **Implement input validation for API:**
    - Range checks for sensor inputs
    - Timestamp sanity checks
- **Add logging & versioning:**
    - Log each prediction request/response for traceability
    - Version the deployed model to track model updates

E. Business & Operational Strategy

- **Develop predictive maintenance workflows:**
    - Automatically generate maintenance tickets when model predicts RECOVERING or BROKEN
- **Cost-benefit analysis:**
    - Quantify savings from early fault detection (less downtime, lower repair costs)
- **User training & documentation:**
    - Train maintenance and operations staff to interpret model outputs
    - Prepare clear user documentation of API endpoints, expected inputs, and usage guidelines

F. Future Enhancements

- Explore real-time streaming predictions by integrating with tools like Kafka or MQTT (if live sensor data feed is available)
- **Investigate model explainability tools:**
    - Use SHAP or LIME to better understand how individual sensor readings influence predictions
- **Expand the feature set:**
    - Incorporate external features (e.g., ambient temperature, humidity, machine load) if available
- **Benchmark with other models:**
    - Evaluate other advanced models (XGBoost, LightGBM, deep learning models) for potential improvement over Random Forest