IBM AICTE PROJECT

PREDICTIVE MAINTENANCE OF INDUSTRIAL MACHINERY

Presented By:
MOTURI SAI DINESH
LOVELY PROFESSIONAL UNIVERSITY (BTech CSE)

edunet
foundation

OUTLINE

- Problem Statement
- Proposed System/Solution
- System Development Approach
- Algorithm & Deployment
- Result (Output Image)
- Conclusion
- Future Scope
- References

PROBLEM STATEMENT

Example: In the manufacturing and industrial sector, unexpected machinery failures lead to unplanned downtime, production losses, and increased maintenance costs.

Modern machines generate vast amounts of real-time sensor data during operation. Despite this, industries still struggle to interpret these signals effectively to prevent faults.

The challenge lies in identifying patterns from sensor data that can indicate specific failure modes before they actually occur. Recognizing these signs early is crucial for ensuring safety, efficiency, and continuous operation.

PROPOSED SOLUTION

- The proposed system aims to address the challenge of anticipating machine failures before they occur by developing a predictive maintenance model for a fleet of industrial machines. This involves leveraging sensor data to identify patterns that precede a failure. The solution will consist of the following components:
- Data Collection:
  - Utilize the "Predictive Maintenance Classification" dataset from Kaggle, which contains real-time operational data.
  - The primary goal is to predict the specific 'Failure Type' of the machinery, such as Power Failure or Tool Wear Failure..
- Data Preprocessing:
  - Clean the data by removing irrelevant columns and converting the categorical 'Type' feature into a numerical format..
  - Standardize all the Sensor data using StandardScalar to ensure all features are on common scale.
- Machine Learning Algorithm:
  - Implement a Random Forest Classifier to learn the patterns that lead to machine failures..
  - Apply the SMOTE technique to balance the training data, which helps the model accurately predict even rare failures..
- Deployment:
  - Deploy the final solution on the IBM Cloud platform as required by the project.
  - Publish the model as a live, online API endpoint using the IBM Watson Machine Learning service.
- Evaluation:
  - Assess the model's performance using Accuracy for an overall score and the F1-Score for a balanced view of its effectiveness.
  - Use a Confusion Matrix to visually analyze the model's predictions and identify where it excels or makes mistakes.

SYSTEM APPROACH

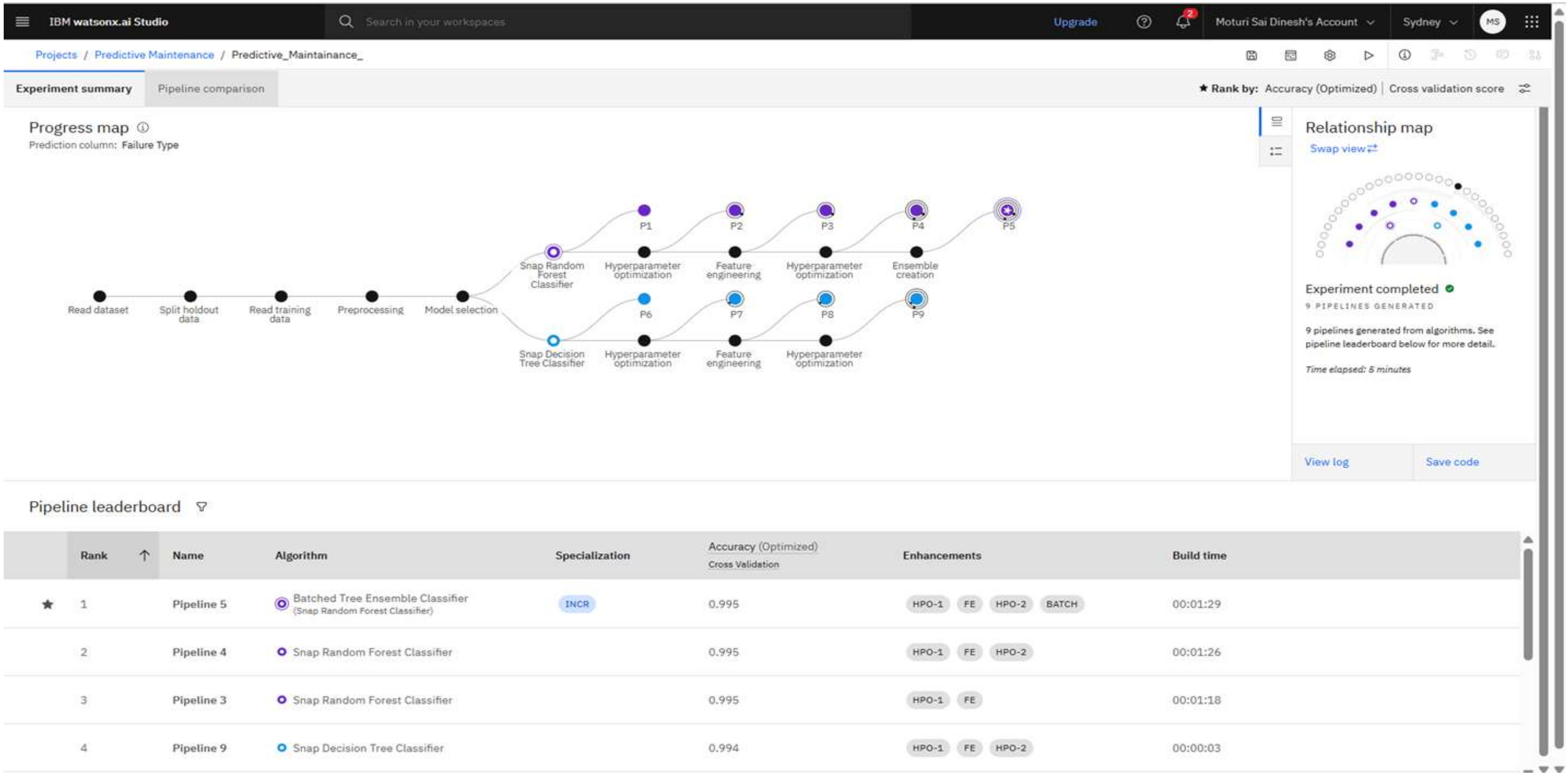
The system approach for this project involves leveraging a cloud-based data science platform to build, train, and deploy a machine learning model capable of predicting industrial machinery failures from sensor data. The entire lifecycle, from data preparation to a live API, is handled within an integrated environment

- System requirements
  - Cloud Platform: An IBM Cloud account (Lite tier is sufficient) is mandatory for hosting the services.
  - Core Services:
    - IBM Watson Studio: To create and manage the project, including the Jupyter Notebook for development.
    - IBM Cloud Object Storage (COS): To store the dataset (predictive_maintenance.csv) securely.
    - IBM Watson Machine Learning Service: To save the trained model and deploy it as a live API.
- Library required to build the model
  - The model is built using Python 3, and the following key libraries are required:
  - pandas: For loading, manipulating, and cleaning the dataset.
  - scikit-learn: The primary machine learning library used for:
    - Data splitting (train_test_split) , Feature scaling (StandardScaler) , Model creation (RandomForestClassifier). and Evaluation metrics (classification_report, accuracy_score).
  - imbalanced-learn: To use the SMOTE technique for handling the imbalanced dataset.
  - IBM-watson-machine-learning: The official IBM client library to save the model and prepare it for deployment.
  - matplotlib & seaborn: For creating visualizations like the confusion matrix to evaluate model performance.

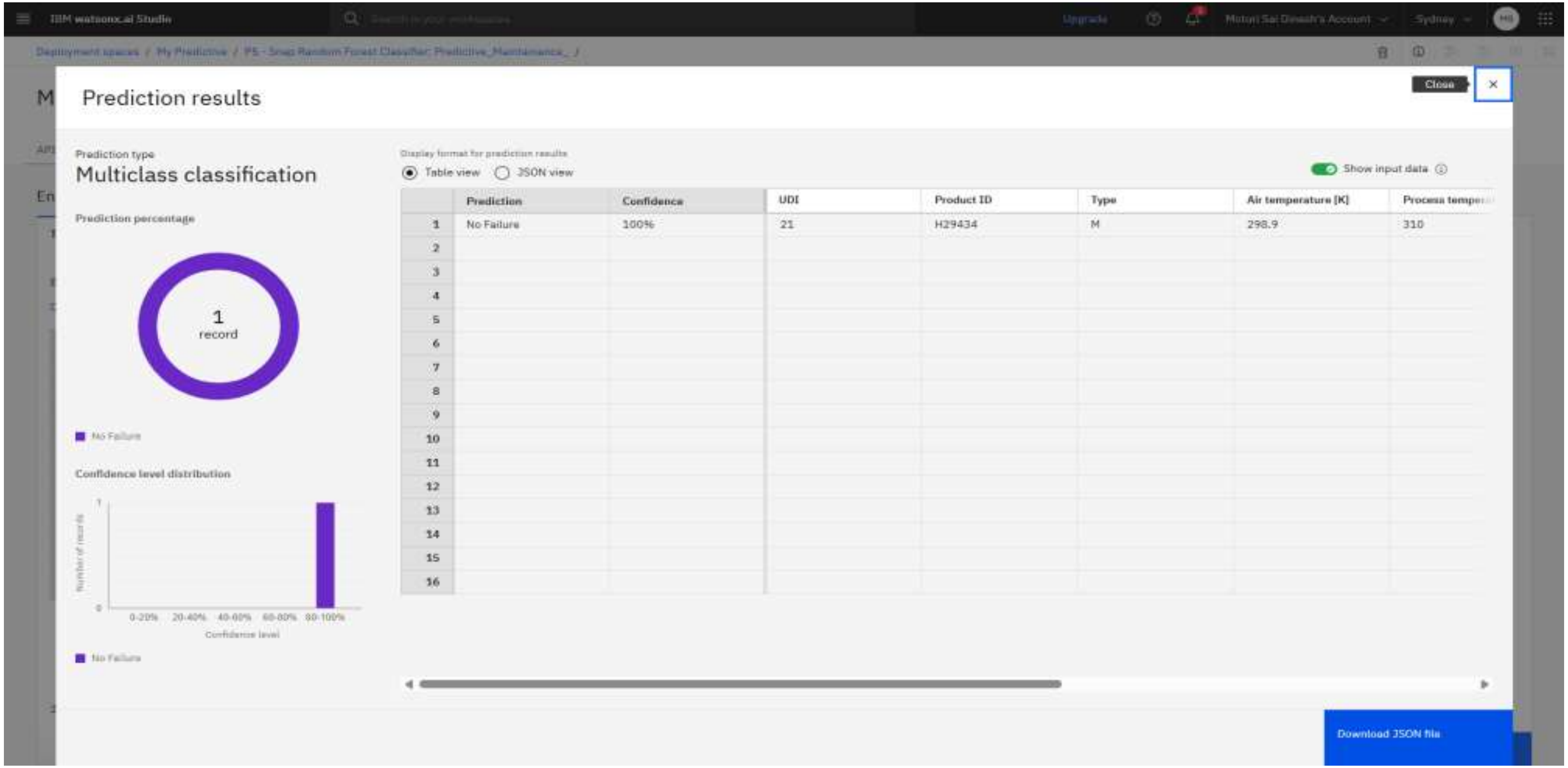
ALGORITHM & DEPLOYMENT

- **Algorithm Selection:**
  - We chose a Random Forest Classifier because it's highly accurate and excels at finding complex patterns in sensor data.
- **Data Input:**
  - The model uses real-time sensor data as input, including temperature, rotational speed, torque, and tool wear.
- **Training Process:**
  - The model is trained on 80% of the data. We use the **SMOTE** technique to balance the data, which is critical for learning to predict the rare failure types.
- **Prediction Process:**
  - The deployed API receives new sensor readings in real-time and instantly predicts a **specific failure type** or "No Failure".

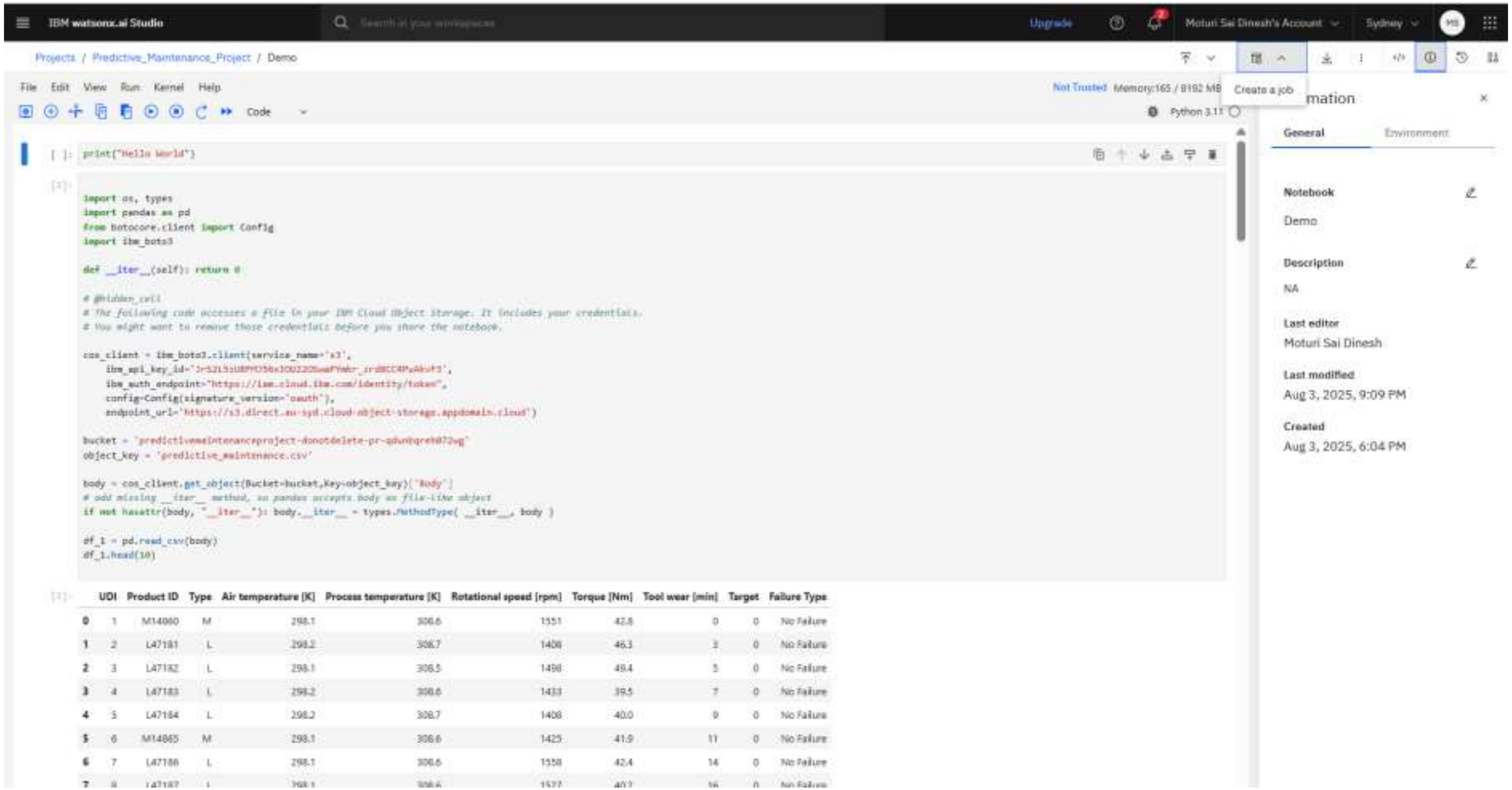
RESULT



edunet foundation



Prediction results
Close
Prediction type
Multiclass classification
Prediction percentage
1
record
No Failure
Confidence level distribution
0-20% 20-40% 40-60% 60-80% 80-100%
Confidence level
No Failure
Table view
JSON view
Show input data
Prediction
Confidence
UDI
Product ID
Type
Air temperature [K]
1 No Failure 100% 21 H29434 M 298.9 310
Download JSON file



edunet
foundation

IBM watsonx.ai Studio

Projects / Predictive_Maintenance_Project / Demo

```
In [ ]: df_1.head()
        df_1.info()
        df_1.describe()
```

```
<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: 781.44 KB
```

Out[ ]:

	UDI	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target
count	10000.00000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	5000.50000	300.004930	310.005560	1538.776100	39.986910	107.951000	0.033900
std	2886.89568	2.000259	1.483734	179.284096	9.968934	63.654147	0.180981
min	1.00000	295.300000	305.700000	1168.000000	3.800000	0.000000	0.000000
25%	2500.75000	298.300000	308.800000	1423.000000	33.200000	53.000000	0.000000
50%	5000.50000	300.100000	310.100000	1503.000000	40.100000	108.000000	0.000000
75%	7500.25000	301.500000	311.100000	1612.000000	46.800000	162.000000	0.000000
max	10000.00000	304.500000	313.800000	2886.000000	76.600000	253.000000	1.000000

```
In [ ]: print(df_1['Failure Type'].value_counts())
```

```
Failure Type
No Failure                  9652
Heat Dissipation Failure     112
Power Failure                 95
Overstrain Failure            78
Tool Wear Failure             45
Random Failures               18
Name: count, dtype: int64
```

```
In [ ]: print(df_1.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')
```

edunet
foundation

IBM watsonx.ai Studio

Search in your workspaces

Upgrade

Projects / Predictive_Maintenance_Project / Demo

In [6]:

```
import pandas as pd
from sklearn.model_selection import train_test_split

# --- Step 1: Handle Categorical Data ---
# Convert the 'Type' column ('L', 'M', 'H') into numerical format.
# We'll also drop the original 'Failure Type' column later.
df_processed = pd.get_dummies(df_1, columns=['Type'], drop_first=True)


# --- Step 2: Separate Features (X) and Target (y) ---
# X contains all columns except our target 'Failure Type' and the leaky 'Target' column.
X = df_processed.drop(['Failure Type', 'Target'], axis=1)

# y is the specific column we want to predict.
y = df_processed['Failure Type']


# --- Step 3: Split Data into Training and Testing Sets ---
# This creates the data sets you will use for model training.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# --- Step 4: Verify the results (Optional but Recommended) ---
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("\nDistribution of failure types in training data:")
print(y_train.value_counts())
```

```
Shape of X_train: (8000, 9)
Shape of X_test: (2000, 9)
```

IBM watsonx.ai Studio

Projects / Predictive_Maintenance_Project / Demo

```python
import pandas as pd
from sklearn.model_selection import train_test_split

# --- Step 1: Handle Categorical Data ---
# Convert the 'Type' column ('L', 'M', 'H') into numerical format.
# We'll also drop the original 'Failure Type' column later.
df_processed = pd.get_dummies(df_1, columns=['Type'], drop_first=True)


# --- Step 2: Separate Features (X) and Target (y) ---
# X contains all columns except our target 'Failure Type' and the leaky 'Target' column.
X = df_processed.drop(['Failure Type', 'Target'], axis=1)

# y is the specific column we want to predict.
y = df_processed['Failure Type']


# --- Step 3: Split Data into Training and Testing Sets ---
# This creates the data sets you will use for model training.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# --- Step 4: Verify the results (Optional but Recommended) ---
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("\nDistribution of failure types in training data:")
print(y_train.value_counts())
```

```
Shape of X_train: (8000, 9)
Shape of X_test: (2000, 9)

Distribution of failure types in training data:
Failure Type
No Failure                  7722
Heat Dissipation Failure      90
Power Failure                 76
Overstrain Failure            62
Tool Wear Failure             36
Random Failures               14
Name: count, dtype: int64
```

```python
# ======================================================================
# PHASE 2: DATA PREPROCESSING
# ======================================================================
print("--- Starting Phase 2: Data Preprocessing ---")

# Import libraries for data manipulation and splitting
import pandas as pd
from sklearn.model_selection import train_test_split

# Drop the identifier columns if they exist. If not, this will be skipped.
if 'UDI' in df_1.columns and 'Product ID' in df_1.columns:
    df_1 = df_1.drop(['UDI', 'Product ID'], axis=1)

# Convert the 'Type' column into numerical format using one-hot encoding
df_processed = pd.get_dummies(df_1, columns=['Type'], drop_first=True)
```

edunet foundation

IBM watsonx.ai Studio

Projects / Predictive_Maintenance_Project / Demo

```python
# Separate the data into features (X) and the target (y)
# We drop 'Target' because it leaks information about the failure.
X = df_processed.drop(['Failure Type', 'Target'], axis=1)
y = df_processed['Failure Type']

# Split data into training and testing sets (80% train, 20% test)
# stratify=y ensures both sets have a similar proportion of each failure type.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

print("Data preprocessing complete.\n")


# ==============================================================================
# PHASE 3: MODEL BUILDING, TRAINING & EVALUATION
# ==============================================================================
print("--- Starting Phase 3: Model Building & Evaluation ---")

# Install the library for handling imbalanced data.
# The '!' runs this as a command line command.
print("Installing required library 'imbalanced-learn'...")
!pip install imbalanced-learn -q
print("Library installed.\n")

# Import all necessary libraries for this phase
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from imblearn.over_sampling import SMOTE
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Scale the features. This is crucial for model performance.
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Use SMOTE to balance the training data by creating synthetic samples for rare failures.
print("Balancing the dataset with SMOTE...")
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_train)
print("Dataset balanced.\n")

# Train the final Random Forest model on the balanced data.
print("Training the final model...")
rf_model_smote = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model_smote.fit(X_train_resampled, y_train_resampled)
print("Model training complete.\n")

# --- Final Model Evaluation Results ---
print("--- Final Model Evaluation Results ---")
y_pred_smote = rf_model_smote.predict(X_test_scaled)
print(f"Final Accuracy: {accuracy_score(y_test, y_pred_smote):.4f}\n")
print("Final Classification Report:")
print(classification_report(y_test, y_pred_smote))
```

edunet foundation

IBM watsonx.ai Studio

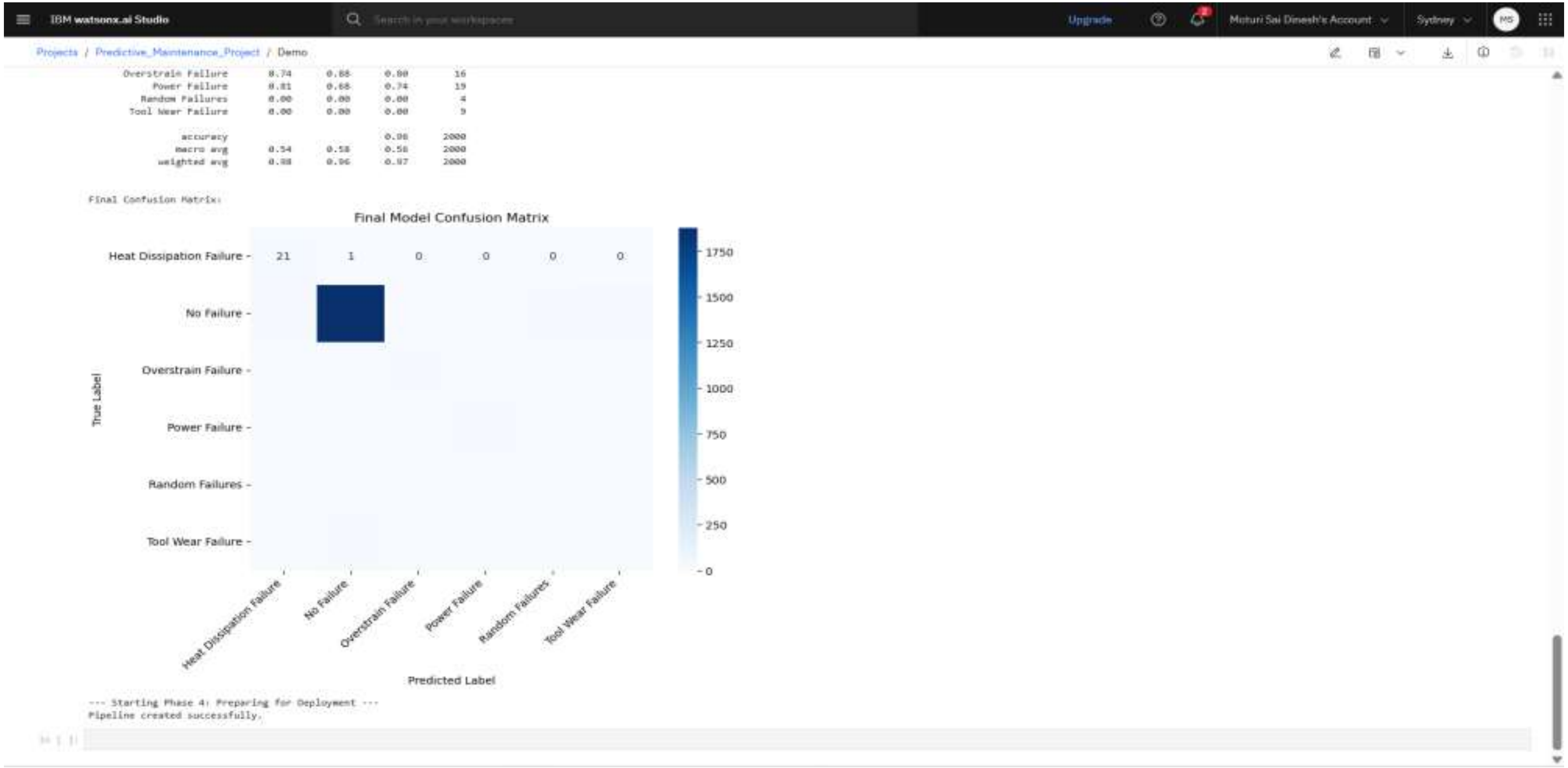
Upgrade | Moturi Sai Dinesh's Account | Sydney

Projects / Predictive_Maintenance_Project / Demo

```python
# Display the Final Confusion Matrix
print("\nFinal Confusion Matrix:")
cm = confusion_matrix(y_test, y_pred_smote, labels=rf_model_smote.classes_)
plt.figure(figsize=(8, 6))
disp = sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=rf_model_smote.classes_, yticklabels=rf_model_smote.classes_)
disp.set_xticklabels(disp.get_xticklabels(), rotation=45, ha='right')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Final Model Confusion Matrix')
plt.show()


# ======================================================================
# PHASE 4: PREPARE AND SAVE MODEL FOR DEPLOYMENT
# ======================================================================
print("\n--- Starting Phase 4: Preparing for Deployment ---")

# Import libraries for deployment
from ibm_watson_machine_learning import APIClient
from sklearn.pipeline import Pipeline
import os

# Bundle the scaler and the trained model into a single pipeline.
# This is crucial for ensuring new data is processed correctly before prediction.
deployment_pipeline = Pipeline([
    ('scaler', scaler),
    ('model', rf_model_smote)
])
print("Pipeline created successfully.")
```

```
--- Starting Phase 2: Data Preprocessing ---
Data preprocessing complete.

--- Starting Phase 3: Model Building & Evaluation ---
Installing required library 'imbalanced-learn'...
Library installed.

Balancing the dataset with SMOTE...
Dataset balanced.

Training the final model...
Model training complete.

--- Final Model Evaluation Results ---
Final Accuracy: 0.9640

Final Classification Report:
                         precision    recall  f1-score   support

Heat Dissipation Failure       0.70      0.95      0.81        22
              No Failure       0.99      0.97      0.98      1930
      Overstrain Failure       0.74      0.88      0.80        16
           Power Failure       0.81      0.68      0.74        19
         Random Failures       0.00      0.00      0.00         4
       Tool Wear Failure       0.00      0.00      0.00         9
```

edunet foundation



edunet
foundation

CONCLUSION

- This project successfully achieved its objective of developing and deploying an end-to-end predictive maintenance solution. By using a Random Forest algorithm combined with the SMOTE data balancing technique, we created a model that can accurately predict specific machine failures before they occur. The final deployed API on **IBM Watson Machine Learning** serves as a robust tool that can be integrated into real-world industrial systems to reduce downtime, lower operational costs, and enable proactive maintenance scheduling. The project demonstrates a complete machine learning lifecycle, from data preprocessing to live deployment

FUTURE SCOPE

- Real-time Integration: Connect the deployed API to live machine sensors and build a dashboard to visualize real-time operational status and failure alerts.
- Advanced Modeling: Explore more complex algorithms like XGBoost or LSTMs to potentially improve prediction accuracy further..

REFERENCES

- Dataset: Kaggle, Predictive Maintenance Classification: https://www.kaggle.com/datasets/shivamb/machine-predictive-maintenance-classificationPlatform:
- IBM Cloud & Watson Studio: https://cloud.ibm.com/.

IBM CERTIFICATIONS

In recognition of the commitment to achieve professional excellence

Getting Started with Artificial Intelligence

IBM

Moturi Sai Dinesh

Has successfully satisfied the requirements for:

Getting Started with Artificial Intelligence

Issued on: Jul 19, 2025
Issued by: IBM SkillsBuild

Verify: https://www.credly.com/badges/cd0aa684-2a87-4eea-a248-74eb0b482a42

IBM

IBM CERTIFICATIONS


In recognition of the commitment to achieve professional excellence

Moturi Sai Dinesh

Has successfully satisfied the requirements for:

Journey to Cloud: Envisioning Your Solution

Issued on: Jul 20, 2025
Issued by: IBM SkillsBuild

Verify: https://www.credly.com/badges/3a7cffb7-0559-409e-a69c-1634544efaef

IBM CERTIFICATIONS

IBM **SkillsBuild** Completion Certificate

This certificate is presented to

Moturi Sai Dinesh

for the completion of

Lab: Retrieval Augmented Generation with LangChain

(ALM-COURSE_3824998)

According to the Adobe Learning Manager system of record

Completion date: 24 Jul 2025 (GMT)

Learning hours: 20 mins

GITHUB LINK:

- https://github.com/Dinesh9831/Predictive-Maintenance-of-Industrial-Machinery

THANK YOU