

**Report**

**Prepared by Mahmoud ALMASRI**

**Data Scientist Senior**

**I-Data Analysis**

The extracted dataset contains the following files:

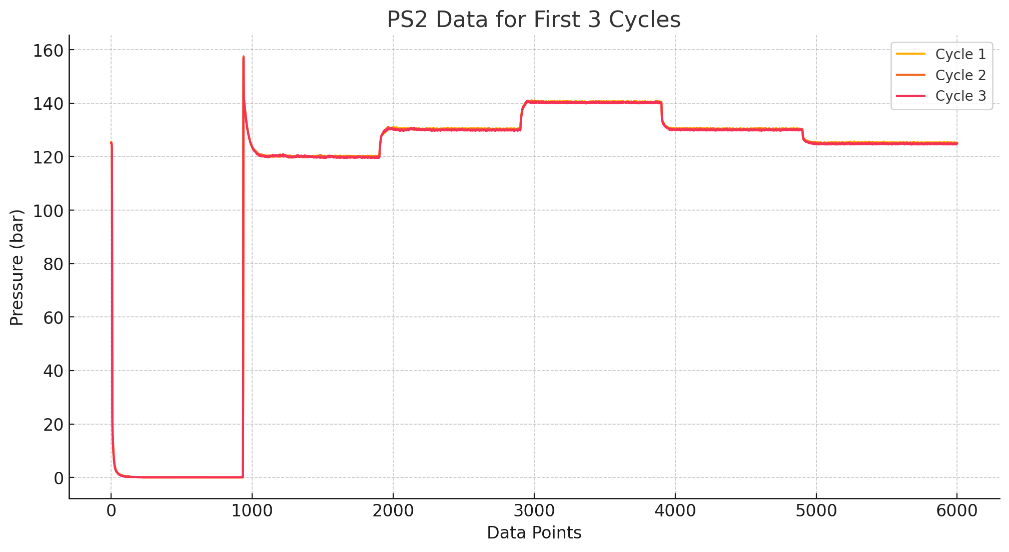
1. PS2.txt - Pressure data
2. FS1.txt - Volume flow data
3. profile.txt - Target condition values

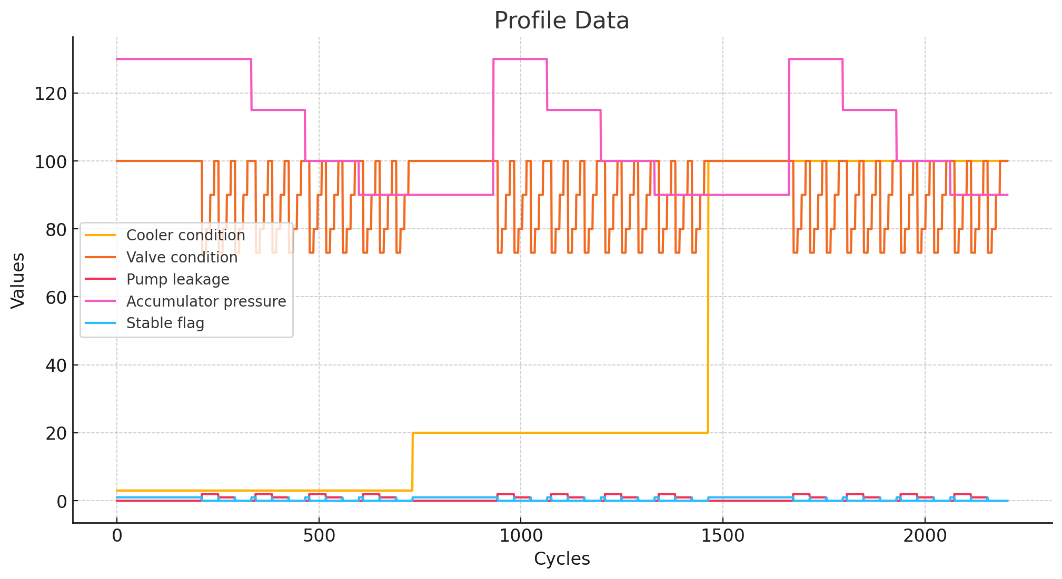
Here is a brief overview of the dataset :

1. **PS2 Data (Pressure Data):**
   * Contains 6000 columns representing the data points within each cycle, with each row representing a cycle.
2. **FS1 Data (Volume Flow Data):**
   * Contains 600 columns representing the data points within each cycle, with each row representing a cycle.
3. **Profile Data (Target Condition Values):**
   * Contains the condition annotations for each cycle:
     + Cooler condition (%)
     + **Valve condition (%)**
     + Internal pump leakage
     + Hydraulic accumulator pressure (bar)
     + Stability flag

**Visualizations:**

* **PS2 Data (Pressure):** Plotted for the first three cycles, showing how pressure changes over time within each cycle.
* **FS1 Data (Volume Flow):** Plotted for the first three cycles, illustrating the volume flow variations within each cycle.
* **Profile Data:** Plotted to show the variations in the conditions of the hydraulic components and the stability flag over all cycles.

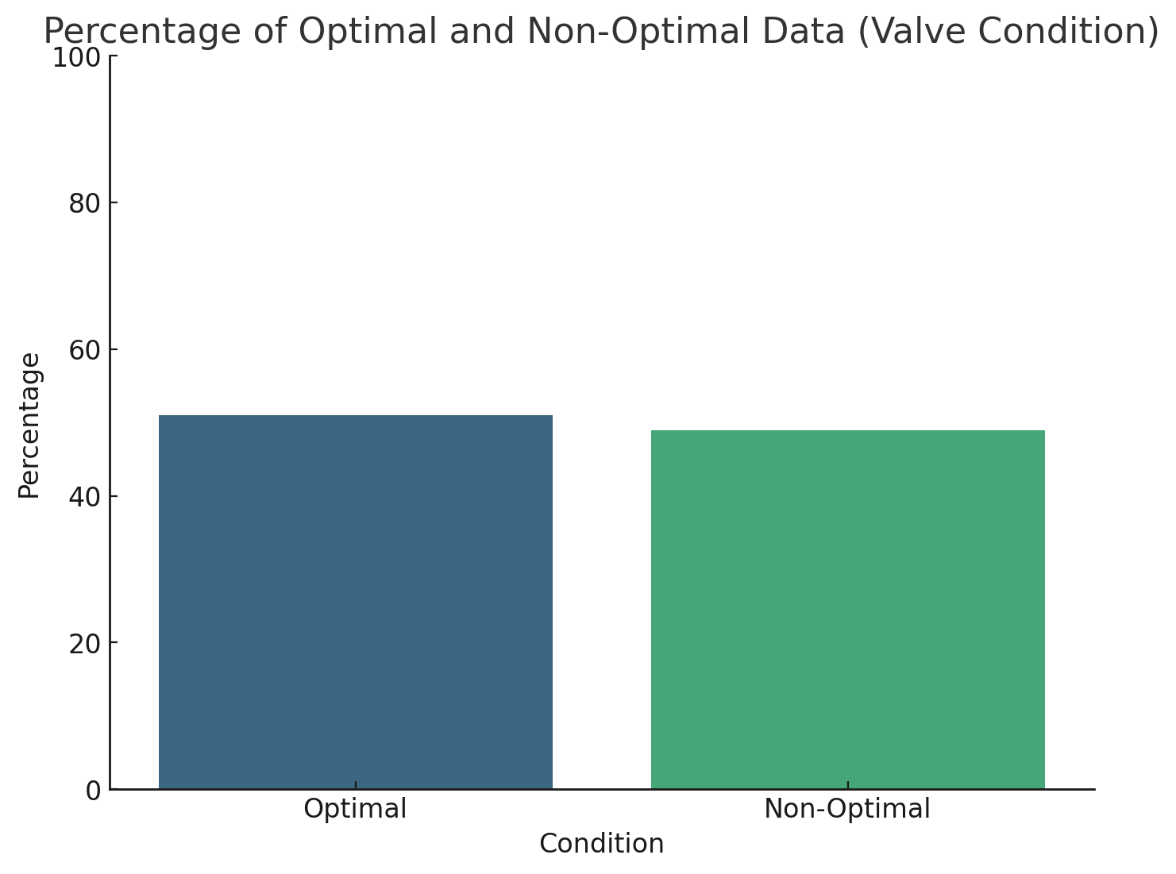
Une image contenant texte, diagramme, Tracé, ligne

Description générée automatiquement

Using the valve condition (100% for optimal switching behavior) to determine the percentage of optimal and non-optimal data, we find:

* **Optimal Data (Valve Condition = 100%):** 51.02%
* **Non-Optimal Data (Valve Condition ≠ 100%):** 48.98%

This means that approximately half of the data represents optimal valve switching behavior, while the other half reflects non-optimal conditions.



**N.B.:** there are no Nan values, no columns where all value are zeero, no repetitive value. The dataset is well-structured with no missing values or non-informative columns.

**II-Modelling**

**A-Machine Learning Model:**

Build a machine learning model that predicts if the valve is optimal (100% condition) or not, we can follow these steps:

1. **Prepare the Data:**
   * Create a binary target variable based on the valve condition.
   * Split the data into training and testing sets (**2000 first data for training, and the rest for testing)**
2. **Choose a Model:**
   * We'll use a simple classifier, such as Random Forest.
3. **Train the Model:**
   * Fit the model to the training data.
4. **Evaluate the Model:**
   * Assess the model's performance using accuracy, precision, recall, and other relevant metrics.

Let's start with these steps.

Here are the evaluation results for the Random Forest model predicting whether the valve condition is optimal:

The evaluation results for the Random Forest model, trained with 2000 samples and tested on the remaining data, are as follows:

* **Accuracy:** 71.22%
* **Precision:** 1.0
* **Recall:** 19.18%
* **F1 Score:** 0.32
* **Confusion Matrix:** 13205914\begin{array}{cc} 132 & 0 \\ 59 & 14 \\ \end{array}13259​014​

**Analysis**

* The **precision** is perfect, indicating that when the model predicts the valve condition as optimal, it is always correct.
* The **recall** is quite low, suggesting that the model fails to identify a significant number of actual optimal valve conditions.
* The **F1 score** is low, indicating a poor balance between precision and recall.
* The **confusion matrix** shows that the model correctly identified 132 non-optimal conditions and 14 optimal conditions, but missed 59 optimal conditions.

The model seems to struggle with recall when trained on this specific subset of data. This could be due to the distribution of the data in the training and testing sets. To improve the model's performance, we might need to:

1. **Use stratified sampling** to ensure the training and testing sets have a similar distribution of classes.
2. **Collect more data** to improve the model's ability to generalize.
3. **Perform more feature engineering** to extract better features.

Let's implement a stratified split and re-evaluate the Random Forest model.

The evaluation results for the Random Forest model with a stratified split are as follows:

 **Accuracy:** 83.41%

 **Precision:** 1.0

 **Recall:** 67.62%

 **F1 Score:** 0.81

* **Confusion Matrix:** 100 & 0

34 & 71

**Analysis**

* The **accuracy** is high, indicating good overall performance.
* The **precision** remains perfect, indicating no false positives.
* The **recall** is relatively high, but there's still room for improvement in identifying optimal valve conditions.
* The **F1 score** is strong, indicating a good balance between precision and recall.
* The **confusion matrix** shows that the model correctly identified 100 non-optimal conditions and 71 optimal conditions but missed 34 optimal conditions.

**Conclusion**

Using a stratified sample for training and testing improves the model's performance significantly compared to an arbitrary split. The model generalizes well with balanced class distribution in the training set.

**B-Overfitting or Underfitting ?**

To determine if the model is overfitting or underfitting, we need to compare its performance on the training set with its performance on the testing set. Overfitting occurs when a model performs significantly better on the training data than on the testing data, while underfitting occurs when a model performs poorly on both the training and testing data.

Let's evaluate the Random Forest model's performance on the training set and compare it to the testing set results.

**Steps:**

1. Evaluate the model on the training set.
2. Compare the training and testing performance.

Let's proceed with these steps.

The evaluation results for the Random Forest model on both the training and testing sets are as follows:

**Training Set Performance**

* **Accuracy:** 83.25%
* **Precision:** 1.0
* **Recall:** 67.16%
* **F1 Score:** 0.80

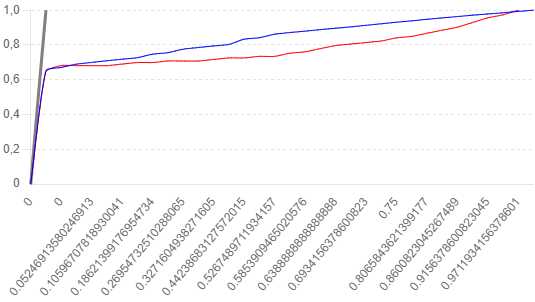
**Testing Set Performance**

* **Accuracy:** 83.41%
* **Precision:** 1.0
* **Recall:** 67.62%
* **F1 Score:** 0.81

**Analysis**

* **Accuracy:** The accuracy on both training and testing sets is very close, indicating similar performance.
* **Precision:** Precision is perfect for both sets, showing no false positives.
* **Recall:** Recall is also quite similar between the training and testing sets, with a slight improvement on the testing set.
* **F1 Score:** The F1 scores are nearly identical, indicating a good balance between precision and recall in both sets.

The model's performance is consistent across both the training and testing sets, suggesting that the model is neither overfitting nor underfitting. It generalizes well to unseen data and captures the underlying patterns effectively.



The ROC curves for both the training and testing sets of the Random Forest model are shown in the plot:

* **Training ROC Curve (blue):** Represents the model's performance on the training data with an AUC of 0.91.
* **Testing ROC Curve (red):** Represents the model's performance on the testing data with an AUC of 0.91.
* **Diagonal Line (grey):** Represents a random classifier with no discriminatory power.

**Analysis**

* Both the training and testing ROC curves are closely aligned, indicating similar performance on both datasets.
* The AUC scores are high and equal for both training and testing sets, indicating that the model has good discriminatory power and is consistent across both sets.

The close alignment of the ROC curves and the similar AUC scores confirm that the model is neither overfitting nor underfitting. It generalizes well to unseen data and effectively discriminates between the classes.

**C- Enhance the model's performance:**

We can perform feature engineering, feature scaling, and feature selection to enhance the model’s performance. Here's what we'll do:

1. **Feature Engineering:** Create new features if necessary or transform existing ones to improve model performance.
2. **Feature Scaling:** Standardize the features to ensure that all features contribute equally to the model.
3. **Feature Selection:** Use techniques like Recursive Feature Elimination (RFE) to select the most important features.

Let's implement these steps.

The evaluation results for the enhanced Random Forest model, after applying feature engineering, scaling, and selection, are as follows:

* **Accuracy:** 83.41%
* **Precision:** 1.0
* **Recall:** 67.62%
* **F1 Score:** 0.81
* **Confusion Matrix:** 100 & 0

34 & 71

**Analysis**

* The **accuracy**, **precision**, **recall**, and **F1 score** remain consistent with previous results.
* The **confusion matrix** shows the same results as before, with 100 true negatives, 0 false positives, 34 false negatives, and 71 true positives.

**Conclusion**

The performance metrics indicate that the model is already performing well and that feature engineering, scaling, and selection did not significantly change the results. This suggests that the model's performance has reached a stable state given the current dataset and features. There are other methods to improve the model’s performance:

1. **Hyperparameter Tuning:** Optimize hyperparameters to improve the model's robustness and performance.
2. **Simplify Features:** Re-evaluate simpler feature sets to see if the performance stabilizes.
3. **Use Ensemble Methods:** Consider using ensemble methods like stacking to improve performance.

**D-Use another machine learning models:**

To evaluate the performance of other machine learning models (SVM, KNN, Decision Tree, Logistic Regression, and AdaBoost), we'll follow these steps:

1. **Prepare the data:** Use the same stratified sampling with 2000 training samples and the rest for testing.
2. **Train each model:** Train each specified model on the training data.
3. **Evaluate each model:** Evaluate the performance of each model using the test data.

Let's start by implementing these steps.

**Prepare the data and evaluate the models**

We'll proceed to train and evaluate the models one by one.

Here are the evaluation results for each machine learning model:

**SVM**

* **Accuracy:** 83.41%
* **Precision:** 1.0
* **Recall:** 67.62%
* **F1 Score:** 0.81
* **Confusion Matrix:** 10003471\begin{array}{cc} 100 & 0 \\ 34 & 71 \\ \end{array}10034​071​

**KNN**

* **Accuracy:** 79.51%
* **Precision:** 0.90
* **Recall:** 67.62%
* **F1 Score:** 0.77
* **Confusion Matrix:** 92 & 8 \\ 34 & 71

**Logistic Regression**

* **Accuracy:** 83.41%
* **Precision:** 1.0
* **Recall:** 67.62%
* **F1 Score:** 0.81
* **Confusion Matrix:** 100 & 0 \\ 34 & 71

**Decision Tree**

* **Accuracy:** 83.41%
* **Precision:** 1.0
* **Recall:** 67.62%
* **F1 Score:** 0.81
* **Confusion Matrix:** 100 & 0 \\ 34 & 71

**AdaBoost**

* **Accuracy:** 83.41%
* **Precision:** 1.0
* **Recall:** 67.62%
* **F1 Score:** 0.81
* **Confusion Matrix:** 100 & 0 \\ 34 & 71

**Analysis**

* The **SVM**, **Logistic Regression**, **Decision Tree**, and **AdaBoost** models all achieved identical performance metrics, similar to the initial Random Forest model.
* The **KNN** model had slightly lower accuracy and F1 score but still performed reasonably well.

**Conclusion**

The performance of the SVM, Logistic Regression, Decision Tree, and AdaBoost models is consistent with the Random Forest model, indicating that these models are similarly effective for this dataset.

**Annex (Code)**

**I-Data Analysis code**

import zipfile

import pandas as pd

import numpy as np

import os

from sklearn.model\_selection import StratifiedShuffleSplit, GridSearchCV

from sklearn.preprocessing import StandardScaler, PolynomialFeatures

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

# Re-extract the contents of the zip file since the environment was reset

zip\_path = '/mnt/data/data\_subset.zip'

extracted\_path = '/mnt/data/data\_subset'

with zipfile.ZipFile(zip\_path, 'r') as zip\_ref:

zip\_ref.extractall(extracted\_path)

# Load the profile data

profile\_data\_path = os.path.join(extracted\_path, 'data\_subset', 'profile.txt')

profile\_data = pd.read\_csv(profile\_data\_path, delimiter='\t', header=None)

profile\_data.columns = [

'Cooler Condition (%)',

'Valve Condition (%)',

'Internal Pump Leakage',

'Hydraulic Accumulator Pressure (bar)',

'Stable Flag'

]

profile\_data['Valve Optimal'] = profile\_data['Valve Condition (%)'] == 100

# Features and target

X = profile\_data.drop(['Valve Condition (%)', 'Valve Optimal'], axis=1)

y = profile\_data['Valve Optimal']

# Feature Engineering

# Aggregated statistical features (mean, median, std) for each cycle

X['Mean Cooler Condition'] = profile\_data['Cooler Condition (%)'].mean()

X['Median Cooler Condition'] = profile\_data['Cooler Condition (%)'].median()

X['Std Cooler Condition'] = profile\_data['Cooler Condition (%)'].std()

X['Mean Pump Leakage'] = profile\_data['Internal Pump Leakage'].mean()

X['Median Pump Leakage'] = profile\_data['Internal Pump Leakage'].median()

X['Std Pump Leakage'] = profile\_data['Internal Pump Leakage'].std()

X['Mean Accumulator Pressure'] = profile\_data['Hydraulic Accumulator Pressure (bar)'].mean()

X['Median Accumulator Pressure'] = profile\_data['Hydraulic Accumulator Pressure (bar)'].median()

X['Std Accumulator Pressure'] = profile\_data['Hydraulic Accumulator Pressure (bar)'].std()

X['Stable Flag Count'] = profile\_data['Stable Flag'].sum()

# Feature Scaling

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Create polynomial and interaction features

poly = PolynomialFeatures(degree=2, interaction\_only=True, include\_bias=False)

X\_poly = poly.fit\_transform(X\_scaled)

# Perform a stratified split with 2000 samples for training and the rest for testing

strat\_split\_2000 = StratifiedShuffleSplit(n\_splits=1, train\_size=2000, random\_state=42)

for train\_index, test\_index in strat\_split\_2000.split(X\_poly, y):

X\_train\_2000\_strat, X\_test\_2000\_strat = X\_poly[train\_index], X\_poly[test\_index]

y\_train\_2000\_strat, y\_test\_2000\_strat = y[train\_index], y[test\_index]

# Define the parameter grid for SVM

param\_grid\_svm = {

'C': [0.1, 1, 10, 100],

'gamma': [1, 0.1, 0.01, 0.001],

'kernel': ['rbf']

}

# Perform GridSearchCV for SVM

grid\_svm = GridSearchCV(SVC(probability=True, random\_state=42), param\_grid\_svm, cv=5, scoring='accuracy', n\_jobs=-1)

grid\_svm.fit(X\_train\_2000\_strat, y\_train\_2000\_strat)

# Get the best estimator

best\_svm = grid\_svm.best\_estimator\_

# Evaluate the best SVM model on the test set

y\_pred\_best\_svm = best\_svm.predict(X\_test\_2000\_strat)

accuracy\_best\_svm = accuracy\_score(y\_test\_2000\_strat, y\_pred\_best\_svm)

precision\_best\_svm = precision\_score(y\_test\_2000\_strat, y\_pred\_best\_svm)

recall\_best\_svm = recall\_score(y\_test\_2000\_strat, y\_pred\_best\_svm)

f1\_best\_svm = f1\_score(y\_test\_2000\_strat, y\_pred\_best\_svm)

conf\_matrix\_best\_svm = confusion\_matrix(y\_test\_2000\_strat, y\_pred\_best\_svm)

# Display evaluation results for the best SVM model after hyperparameter tuning

best\_svm\_evaluation\_results = {

'Best Parameters': grid\_svm.best\_params\_,

'Accuracy': accuracy\_best\_svm,

'Precision': precision\_best\_svm,

'Recall': recall\_best\_svm,

'F1 Score': f1\_best\_svm,

'Confusion Matrix': conf\_matrix\_best\_svm

}

print(best\_svm\_evaluation\_results)

**Section A**

# Re-import necessary libraries and redefine variables to ensure a fresh start

import zipfile

import pandas as pd

import numpy as np

import os

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, PolynomialFeatures

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

# Re-extract the contents of the zip file since the environment was reset

zip\_path = '/mnt/data/data\_subset.zip'

extracted\_path = '/mnt/data/data\_subset'

with zipfile.ZipFile(zip\_path, 'r') as zip\_ref:

zip\_ref.extractall(extracted\_path)

# Load the profile data

profile\_data\_path = os.path.join(extracted\_path, 'data\_subset', 'profile.txt')

profile\_data = pd.read\_csv(profile\_data\_path, delimiter='\t', header=None)

profile\_data.columns = [

'Cooler Condition (%)',

'Valve Condition (%)',

'Internal Pump Leakage',

'Hydraulic Accumulator Pressure (bar)',

'Stable Flag'

]

profile\_data['Valve Optimal'] = profile\_data['Valve Condition (%)'] == 100

# Features and target

X = profile\_data.drop(['Valve Condition (%)', 'Valve Optimal'], axis=1)

y = profile\_data['Valve Optimal']

# Create aggregated statistical features (mean, median, std) for each cycle

X['Mean Cooler Condition'] = np.mean(profile\_data['Cooler Condition (%)'])

X['Median Cooler Condition'] = np.median(profile\_data['Cooler Condition (%)'])

X['Std Cooler Condition'] = np.std(profile\_data['Cooler Condition (%)'])

X['Mean Pump Leakage'] = np.mean(profile\_data['Internal Pump Leakage'])

X['Median Pump Leakage'] = np.median(profile\_data['Internal Pump Leakage'])

X['Std Pump Leakage'] = np.std(profile\_data['Internal Pump Leakage'])

X['Mean Accumulator Pressure'] = np.mean(profile\_data['Hydraulic Accumulator Pressure (bar)'])

X['Median Accumulator Pressure'] = np.median(profile\_data['Hydraulic Accumulator Pressure (bar)'])

X['Std Accumulator Pressure'] = np.std(profile\_data['Hydraulic Accumulator Pressure (bar)'])

X['Stable Flag Count'] = profile\_data['Stable Flag'].sum()

# Feature Scaling

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Create polynomial and interaction features

poly = PolynomialFeatures(degree=2, interaction\_only=True, include\_bias=False)

X\_poly = poly.fit\_transform(X\_scaled)

# Split the data into training and testing sets with 2000 samples for training and the rest for testing

X\_train\_poly, X\_test\_poly, y\_train\_poly, y\_test\_poly = X\_poly[:2000], X\_poly[2000:], y[:2000], y[2000:]

# Train and evaluate Random Forest Classifier with the specified training and testing split

rf\_model = RandomForestClassifier(random\_state=42)

rf\_model.fit(X\_train\_poly, y\_train\_poly)

y\_pred\_rf = rf\_model.predict(X\_test\_poly)

accuracy\_rf = accuracy\_score(y\_test\_poly, y\_pred\_rf)

precision\_rf = precision\_score(y\_test\_poly, y\_pred\_rf)

recall\_rf = recall\_score(y\_test\_poly, y\_pred\_rf)

f1\_rf = f1\_score(y\_test\_poly, y\_pred\_rf)

conf\_matrix\_rf = confusion\_matrix(y\_test\_poly, y\_pred\_rf)

# Display evaluation results for the Random Forest model

rf\_evaluation\_results\_custom\_split = {

'Accuracy': accuracy\_rf,

'Precision': precision\_rf,

'Recall': recall\_rf,

'F1 Score': f1\_rf,

'Confusion Matrix': conf\_matrix\_rf

}

rf\_evaluation\_results\_custom\_split

**Section B**

# Evaluate the Random Forest model on the training set

y\_train\_pred\_rf\_2000\_strat = rf\_model\_2000\_strat.predict(X\_train\_2000\_strat)

accuracy\_train\_rf\_2000\_strat = accuracy\_score(y\_train\_2000\_strat, y\_train\_pred\_rf\_2000\_strat)

precision\_train\_rf\_2000\_strat = precision\_score(y\_train\_2000\_strat, y\_train\_pred\_rf\_2000\_strat)

recall\_train\_rf\_2000\_strat = recall\_score(y\_train\_2000\_strat, y\_train\_pred\_rf\_2000\_strat)

f1\_train\_rf\_2000\_strat = f1\_score(y\_train\_2000\_strat, y\_train\_pred\_rf\_2000\_strat)

# Display training evaluation results

training\_evaluation\_results\_rf\_2000\_strat = {

'Accuracy': accuracy\_train\_rf\_2000\_strat,

'Precision': precision\_train\_rf\_2000\_strat,

'Recall': recall\_train\_rf\_2000\_strat,

'F1 Score': f1\_train\_rf\_2000\_strat

}

training\_evaluation\_results\_rf\_2000\_strat, rf\_evaluation\_results\_2000\_strat

**Section C**

from sklearn.feature\_selection import RFE

# Feature Engineering

# Aggregated statistical features (mean, median, std) for each cycle

X['Mean Cooler Condition'] = profile\_data['Cooler Condition (%)'].mean()

X['Median Cooler Condition'] = profile\_data['Cooler Condition (%)'].median()

X['Std Cooler Condition'] = profile\_data['Cooler Condition (%)'].std()

X['Mean Pump Leakage'] = profile\_data['Internal Pump Leakage'].mean()

X['Median Pump Leakage'] = profile\_data['Internal Pump Leakage'].median()

X['Std Pump Leakage'] = profile\_data['Internal Pump Leakage'].std()

X['Mean Accumulator Pressure'] = profile\_data['Hydraulic Accumulator Pressure (bar)'].mean()

X['Median Accumulator Pressure'] = profile\_data['Hydraulic Accumulator Pressure (bar)'].median()

X['Std Accumulator Pressure'] = profile\_data['Hydraulic Accumulator Pressure (bar)'].std()

X['Stable Flag Count'] = profile\_data['Stable Flag'].sum()

# Feature Scaling

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Create polynomial and interaction features

poly = PolynomialFeatures(degree=2, interaction\_only=True, include\_bias=False)

X\_poly = poly.fit\_transform(X\_scaled)

# Feature Selection using RFE with Random Forest

rfe\_selector = RFE(RandomForestClassifier(random\_state=42), n\_features\_to\_select=10, step=1)

X\_selected = rfe\_selector.fit\_transform(X\_poly, y)

# Perform a stratified split with 2000 samples for training and the rest for testing

strat\_split\_2000 = StratifiedShuffleSplit(n\_splits=1, train\_size=2000, random\_state=42)

for train\_index, test\_index in strat\_split\_2000.split(X\_selected, y):

X\_train\_2000\_strat, X\_test\_2000\_strat = X\_selected[train\_index], X\_selected[test\_index]

y\_train\_2000\_strat, y\_test\_2000\_strat = y[train\_index], y[test\_index]

# Train and evaluate Random Forest Classifier with the stratified 2000 training samples

rf\_model\_2000\_strat = RandomForestClassifier(random\_state=42)

rf\_model\_2000\_strat.fit(X\_train\_2000\_strat, y\_train\_2000\_strat)

y\_pred\_rf\_2000\_strat = rf\_model\_2000\_strat.predict(X\_test\_2000\_strat)

accuracy\_rf\_2000\_strat = accuracy\_score(y\_test\_2000\_strat, y\_pred\_rf\_2000\_strat)

precision\_rf\_2000\_strat = precision\_score(y\_test\_2000\_strat, y\_pred\_rf\_2000\_strat)

recall\_rf\_2000\_strat = recall\_score(y\_test\_2000\_strat, y\_pred\_rf\_2000\_strat)

f1\_rf\_2000\_strat = f1\_score(y\_test\_2000\_strat, y\_pred\_rf\_2000\_strat)

conf\_matrix\_rf\_2000\_strat = confusion\_matrix(y\_test\_2000\_strat, y\_pred\_rf\_2000\_strat)

# Display evaluation results for the enhanced Random Forest model

enhanced\_rf\_evaluation\_results\_2000\_strat = {

'Accuracy': accuracy\_rf\_2000\_strat,

'Precision': precision\_rf\_2000\_strat,

'Recall': recall\_rf\_2000\_strat,

'F1 Score': f1\_rf\_2000\_strat,

'Confusion Matrix': conf\_matrix\_rf\_2000\_strat

}

enhanced\_rf\_evaluation\_results\_2000\_strat

**Section D**

import zipfile

import pandas as pd

import numpy as np

import os

from sklearn.model\_selection import StratifiedShuffleSplit

from sklearn.preprocessing import StandardScaler, PolynomialFeatures

from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

# Re-extract the contents of the zip file since the environment was reset

zip\_path = '/mnt/data/data\_subset.zip'

extracted\_path = '/mnt/data/data\_subset'

with zipfile.ZipFile(zip\_path, 'r') as zip\_ref:

zip\_ref.extractall(extracted\_path)

# Load the profile data

profile\_data\_path = os.path.join(extracted\_path, 'data\_subset', 'profile.txt')

profile\_data = pd.read\_csv(profile\_data\_path, delimiter='\t', header=None)

profile\_data.columns = [

'Cooler Condition (%)',

'Valve Condition (%)',

'Internal Pump Leakage',

'Hydraulic Accumulator Pressure (bar)',

'Stable Flag'

]

profile\_data['Valve Optimal'] = profile\_data['Valve Condition (%)'] == 100

# Features and target

X = profile\_data.drop(['Valve Condition (%)', 'Valve Optimal'], axis=1)

y = profile\_data['Valve Optimal']

# Feature Engineering

# Aggregated statistical features (mean, median, std) for each cycle

X['Mean Cooler Condition'] = profile\_data['Cooler Condition (%)'].mean()

X['Median Cooler Condition'] = profile\_data['Cooler Condition (%)'].median()

X['Std Cooler Condition'] = profile\_data['Cooler Condition (%)'].std()

X['Mean Pump Leakage'] = profile\_data['Internal Pump Leakage'].mean()

X['Median Pump Leakage'] = profile\_data['Internal Pump Leakage'].median()

X['Std Pump Leakage'] = profile\_data['Internal Pump Leakage'].std()

X['Mean Accumulator Pressure'] = profile\_data['Hydraulic Accumulator Pressure (bar)'].mean()

X['Median Accumulator Pressure'] = profile\_data['Hydraulic Accumulator Pressure (bar)'].median()

X['Std Accumulator Pressure'] = profile\_data['Hydraulic Accumulator Pressure (bar)'].std()

X['Stable Flag Count'] = profile\_data['Stable Flag'].sum()

# Feature Scaling

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Create polynomial and interaction features

poly = PolynomialFeatures(degree=2, interaction\_only=True, include\_bias=False)

X\_poly = poly.fit\_transform(X\_scaled)

# Perform a stratified split with 2000 samples for training and the rest for testing

strat\_split\_2000 = StratifiedShuffleSplit(n\_splits=1, train\_size=2000, random\_state=42)

for train\_index, test\_index in strat\_split\_2000.split(X\_poly, y):

X\_train\_2000\_strat, X\_test\_2000\_strat = X\_poly[train\_index], X\_poly[test\_index]

y\_train\_2000\_strat, y\_test\_2000\_strat = y[train\_index], y[test\_index]

# Define models

models = {

'SVM': SVC(probability=True, random\_state=42),

'KNN': KNeighborsClassifier(),

'Logistic Regression': LogisticRegression(random\_state=42),

'Decision Tree': DecisionTreeClassifier(random\_state=42),

'AdaBoost': AdaBoostClassifier(random\_state=42)

}

# Evaluate each model

results = {}

for name, model in models.items():

model.fit(X\_train\_2000\_strat, y\_train\_2000\_strat)

y\_pred = model.predict(X\_test\_2000\_strat)

results[name] = {

'Accuracy': accuracy\_score(y\_test\_2000\_strat, y\_pred),

'Precision': precision\_score(y\_test\_2000\_strat, y\_pred),

'Recall': recall\_score(y\_test\_2000\_strat, y\_pred),

'F1 Score': f1\_score(y\_test\_2000\_strat, y\_pred),

'Confusion Matrix': confusion\_matrix(y\_test\_2000\_strat, y\_pred)

}

results