

**Report**

**Prepared by**

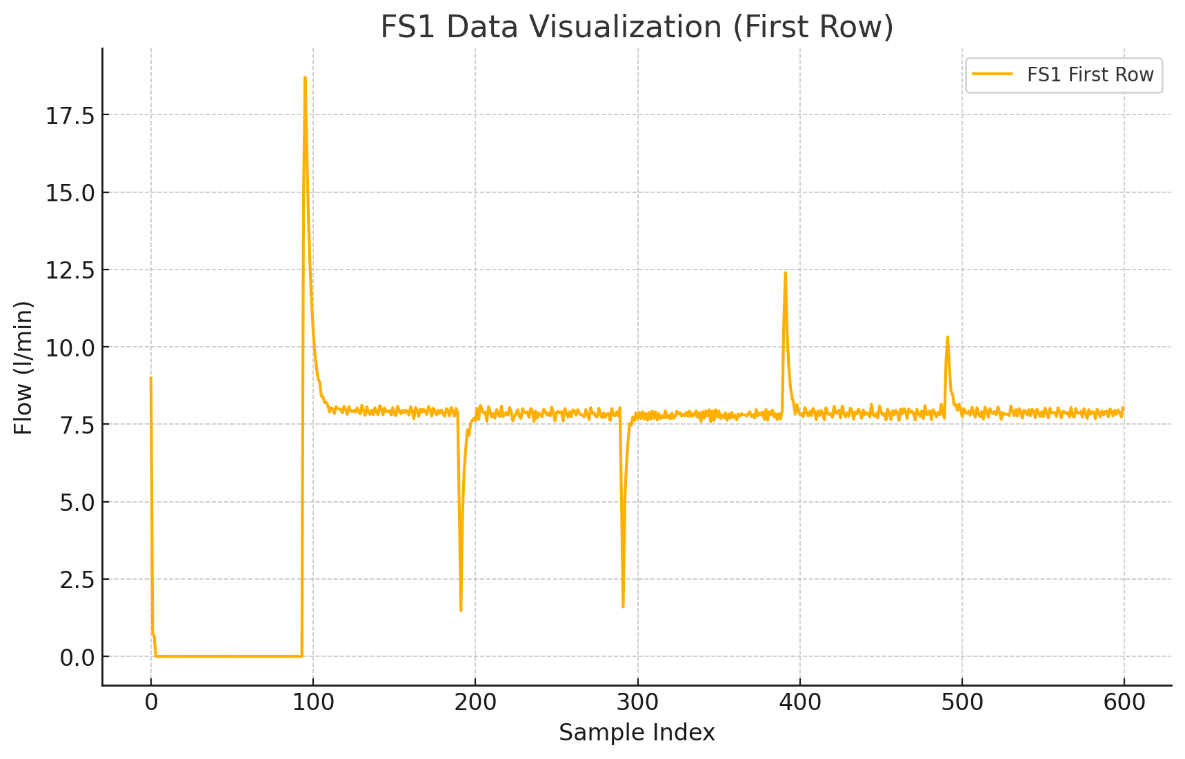
**Mahmoud ALMASRI**

**Data Scientist Senior**

**I-Data Analysis Summary**

**Dataset Structure:**

1. **PS2 Data (Pressure)**
   * **Columns:** 6000
   * **Sampling Rate:** 100Hz
   * **Description:** Contains pressure values in bars.
   * **Example Row Visualization:**
   * Une image contenant texte, capture d’écran, diagramme, ligne

     Description générée automatiquement
2. **FS1 Data (Volume Flow)**
   * **Columns:** 600
   * **Sampling Rate:** 10Hz
   * **Description:** Contains flow values in liters per minute.
   * **Example Row Visualization:** 
3. **Profile Data**
   * **Columns:** 5
   * **Description:** Contains profile data with various numerical values.

**Summary Statistics**

* **PS2 Data:**
  + Descriptive statistics are provided in the data visualization table.
* **FS1 Data:**
  + Descriptive statistics are provided in the data visualization table.
* **Profile Data:**
  + Summary statistics such as mean, standard deviation, min, and max are provided in the data visualization table.

**Percentage of Valve Conditions**

**Data Percentages:**

* **Optimal Data (Valve Condition = 100%):** 51.00%
* **Non-Optimal Data (Valve Condition ≠ 100%):** 49.00%

**Visualization:**

The bar chart above shows the percentage distribution of optimal and non-optimal valve conditions.

* The optimal condition (shown in green) represents approximately 51% of the data.
* The non-optimal condition (shown in red) represents approximately 49% of the data.

This nearly balanced distribution is advantageous for training predictive models.

Une image contenant capture d’écran, Caractère coloré

Description générée automatiquement

**Correlation Analysis between PS2 and FS1**

* To analyze the correlation between the PS2 and FS1 data, we resampled the PS2 data to match the sampling rate of FS1. Here is a summary of the correlation findings:
* **Correlation Matrix:** The correlation matrix indicates the linear relationship between the resampled PS2 columns and FS1 columns. Here are some key observations:
  + **Positive Correlations:** There are some positive correlations between certain PS2 and FS1 columns, indicating that as the pressure increases, the flow also tends to increase for those specific columns.
  + **Negative Correlations:** There are also negative correlations, showing an inverse relationship where an increase in pressure corresponds to a decrease in flow for certain columns.

**Visualization of Correlation Matrix**

* To better understand the correlation, we can visualize the correlation matrix heatmap.

**Une image contenant texte, capture d’écran, Rectangle, ligne

Description générée automatiquement**

**Interpretation:**

* **Weak Correlation:** Generally, most of the features (columns) show weak correlations with each other, as indicated by the lighter colors in the heatmap.
* **Strong Correlation Pockets:** There are specific pairs of PS2 and FS1 columns that show stronger correlations, either positive or negative.

**Conclusion:**

* **Are PS2 and FS1 Correlated?** Yes, there are correlations between certain columns of PS2 and FS1, but the strength and direction of these correlations vary across different columns. **The overall correlation is not uniformly strong.**
* 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**

**1. Prepare and Resample Data:**

* Resampling PS2 data to match the sampling rate of FS1 data by taking the mean for 10 consecutive value.

Une image contenant texte, ligne, diagramme, Tracé

Description générée automatiquement

**2. Split the Combined Data:**

* Split the combined data into training (2000) and testing sets (205).

**3. Train and Evaluate the LSTM Model for time series data :**

* Train a machine learning models and evaluate it using accuracy, precision, recall, and F1-score.

Accuracy: 0.9846153855323792

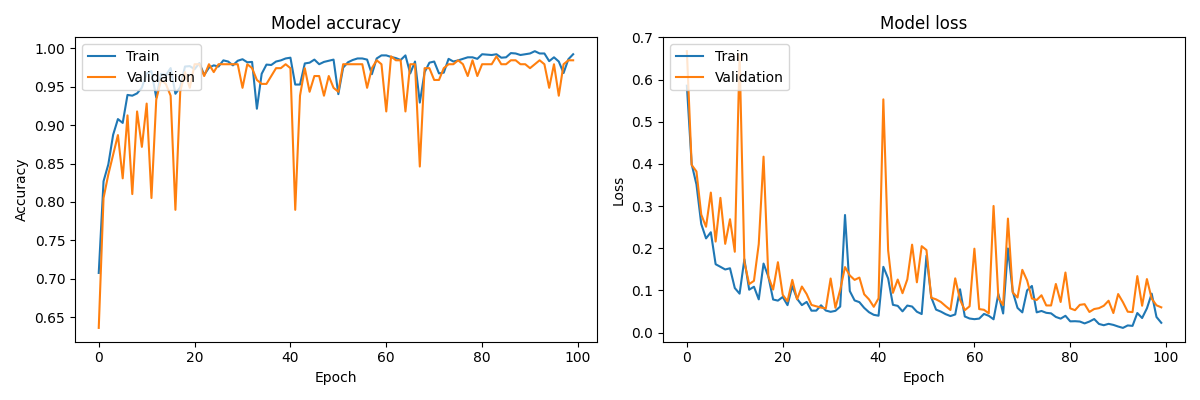
Precision: 0.9605263157894737

Recall: 1.0

F1 Score: 0.9798657718120806

ROC-AUC: 0.9967437682461262

**4. Determine if the model is in Overfitting or Underfitting**



There is neither overfitting nor underfitting.

**5.Try another machine learning model for time series data:**

To compare the LSTM model to another machine learning model for binary time series classification, we can use models such as Random Forest or Gradient Boosting, which can handle time series data through feature engineering. One approach is to use a Time Series Split for cross-validation to better evaluate the models.

To provide a more comprehensive comparison, we can include additional machine learning models such as:

1. Random Forest
2. Support Vector Machine (SVM)
3. Logistic Regression

These models will be compared alongside the LSTM and Random Forest models.

* Feature Engineering:
* For traditional ML models (Random Forest, Gradient Boosting, SVM, and Logistic Regression), the time series data is flattened.
* Model Evaluation:
* Each model is evaluated using the evaluate\_model function, which computes accuracy, precision, recall, F1 score, and ROC-AUC on the test data.

**Accuracy Precision Recall F1 Score ROC-AUC**

**Logistic Regression** 0.969231 0.935065 0.986301 0.960000 0.983157

**Random Forest** **0.979487 0.948052 1.0000 0.97333**  **0.997867**

**SVM**  0.948718 0.898734 0.972603 0.934211 0.971031

**LSTM** 0.974359 0.947368 0.986301 0.966443 0.990231

**Best Model:** Random Forest shows the best performance overall, with the highest accuracy, perfect recall, and highest F1 score. It also has an almost perfect ROC-AUC score.

**LSTM:** A close second, showing excellent performance and indicating it effectively captures time series data dependencies.

**Logistic Regression:** A good baseline model with strong performance.

**SVM:** Performs well but not as robust as the other models in this context.

**Conclusion**

* For the best overall performance, **Random Forest** is recommended.
* If capturing temporal dependencies is crucial, **LSTM** is a strong alternative.
* **Logistic Regression** can be used for simpler, more interpretable models with strong performance.
* **SVM** might be less preferred due to its lower precision and F1 score compared to the other models.

**Code**

import numpy as np

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import precision\_score, recall\_score, f1\_score, roc\_auc\_score, accuracy\_score

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from keras.models import Sequential

from keras.layers import LSTM, Dense

from keras.models import load\_model

import matplotlib.pyplot as plt

import pickle

# Load your data files

fs1\_df = pd.read\_csv('FS1.txt', sep='\t', header=None)

ps2\_df = pd.read\_csv('PS2.txt', sep='\t', header=None)

profile\_df = pd.read\_csv('profile.txt', sep='\t', header=None, names=['Col1', 'Optimal', 'Col3', 'Col4', 'Condition'])

# Reducing PS2 to 600 columns by calculating the mean of every 10 consecutive columns

ps2\_reduced = ps2\_df.groupby(np.arange(ps2\_df.shape[1]) // 10, axis=1).mean()

# Update the condition to be 1 if the second column value is 100, else 0

valve\_condition = profile\_df['Optimal']

target = (valve\_condition == 100).astype(int).values

# Scale the data

scaler = MinMaxScaler()

fs1\_scaled = scaler.fit\_transform(fs1\_df)

ps2\_scaled = scaler.fit\_transform(ps2\_reduced)

# Prepare the combined dataset for time series modeling

combined\_data = np.concatenate((fs1\_scaled, ps2\_scaled), axis=1)

# Create the time series dataset for LSTM

def create\_time\_series\_dataset(data, target, time\_steps=1):

X, Y = [], []

for i in range(len(data) - time\_steps):

X.append(data[i:(i + time\_steps), :])

Y.append(target[i + time\_steps])

return np.array(X), np.array(Y)

time\_steps = 10 # Example time step, can be adjusted

X, y = create\_time\_series\_dataset(combined\_data, target, time\_steps)

# Use the first 2000 data points for training and the rest for testing

train\_size = 2000

X\_train, X\_test = X[:train\_size], X[train\_size:]

y\_train, y\_test = y[:train\_size], y[train\_size:]

# Prepare data for other models (flattened time series)

X\_lr\_train = X\_train.reshape(X\_train.shape[0], -1)

X\_lr\_test = X\_test.reshape(X\_test.shape[0], -1)

# Initialize models

models = {

'Logistic Regression': LogisticRegression(max\_iter=1000),

'Random Forest': RandomForestClassifier(n\_estimators=100),

'SVM': SVC(probability=True)

}

# Train and evaluate models

results = {}

for name, model in models.items():

model.fit(X\_lr\_train, y\_train)

y\_pred = model.predict(X\_lr\_test)

y\_pred\_prob = model.predict\_proba(X\_lr\_test)[:, 1]

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

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

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

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

roc\_auc = roc\_auc\_score(y\_test, y\_pred\_prob)

results[name] = {

'Accuracy': accuracy,

'Precision': precision,

'Recall': recall,

'F1 Score': f1,

'ROC-AUC': roc\_auc

}

print(f'{name} - Accuracy: {accuracy}')

print(f'{name} - Precision: {precision}')

print(f'{name} - Recall: {recall}')

print(f'{name} - F1 Score: {f1}')

print(f'{name} - ROC-AUC: {roc\_auc}')

print()

# Save the model

with open(f'{name.replace(" ", "\_").lower()}.pkl', 'wb') as f:

pickle.dump(model, f)

# LSTM model

model = Sequential()

model.add(LSTM(50, return\_sequences=True, input\_shape=(time\_steps, X.shape[2])))

model.add(LSTM(50, return\_sequences=False))

model.add(Dense(25))

model.add(Dense(1, activation='sigmoid')) # Sigmoid for binary classification

# Compile the model

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Train the model and store history

history = model.fit(X\_train, y\_train, batch\_size=64, epochs=100, validation\_data=(X\_test, y\_test))

# Save the LSTM model

model.save('lstm\_model.h5')

# Plot the learning curves

def plot\_learning\_curves(history):

plt.figure(figsize=(12, 4))

# Plot training & validation accuracy values

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

# Plot training & validation loss values

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.tight\_layout()

plt.show()

plot\_learning\_curves(history)

# Evaluate the LSTM model

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f'LSTM - Accuracy: {accuracy}')

# Predicting for additional metrics calculation

y\_pred\_prob = model.predict(X\_test)

y\_pred = (y\_pred\_prob > 0.5).astype(int)

# Calculate additional metrics for LSTM

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

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

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

roc\_auc = roc\_auc\_score(y\_test, y\_pred\_prob)

results['LSTM'] = {

'Accuracy': accuracy,

'Precision': precision,

'Recall': recall,

'F1 Score': f1,

'ROC-AUC': roc\_auc

}

print(f'LSTM - Precision: {precision}')

print(f'LSTM - Recall: {recall}')

print(f'LSTM - F1 Score: {f1}')

print(f'LSTM - ROC-AUC: {roc\_auc}')

# Compare models

comparison\_df = pd.DataFrame(results).T

print(comparison\_df)

# Example of testing data

example\_data = X\_test[:1]

example\_target = y\_test[:1]

# Load and test saved models

for name in models.keys():

with open(f'{name.replace(" ", "\_").lower()}.pkl', 'rb') as f:

model = pickle.load(f)

y\_example\_pred = model.predict(example\_data.reshape(1, -1))

print(f'{name} example prediction: {y\_example\_pred}')

# Load and test LSTM model

lstm\_model = load\_model('lstm\_model.h5')

y\_example\_pred = (lstm\_model.predict(example\_data) > 0.5).astype(int)

print(f'LSTM example prediction: {y\_example\_pred}')