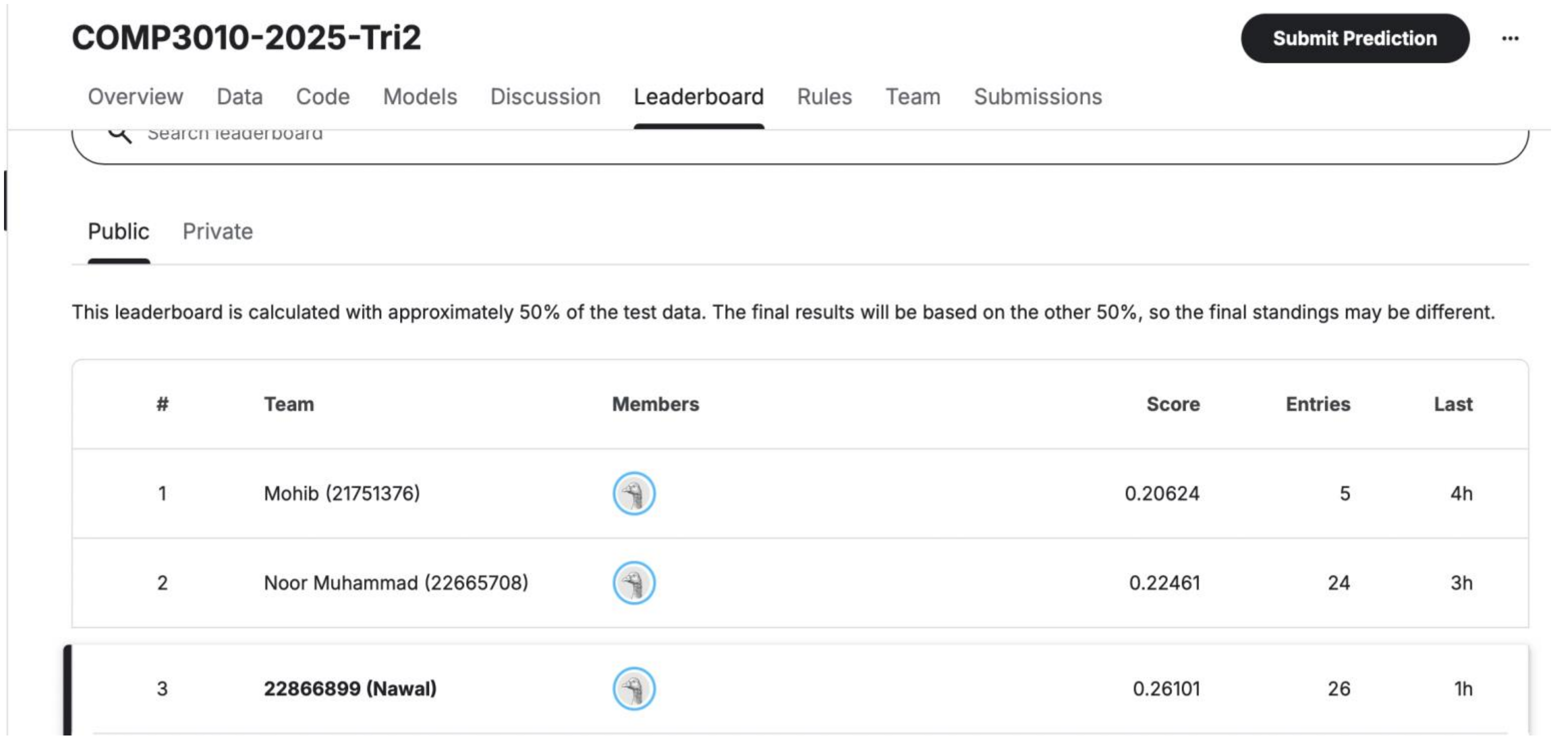

COMP3010-2025-Tri2

This leaderboard is calculated with approximately 50% of the test data. The final results will be based on the other 50%, so the final standings may be different.

#	Team	Members	Score	Entries	Last
1	Mohib (21751376)		0.20624	5	4h
2	Noor Muhammad (22665708)		0.22461	24	3h
3	**22866899 (Nawal)**		0.26101	26	1h

Name: Nawal Nadim

Student ID: 22866899

Tutor: Ms Benita

Campus: Dubai

Unit: Machine Learning

1) Data Cleaning: Describe the data issues encountered (e.g., missing values, outliers, duplicates, incorrect entries) and the specific steps taken to resolve them.

- **Dropped Unnecessary Columns:** The column 'Unnamed: 0' was removed, as it was an artifact from CSV export and did not provide useful information.

- **Removed Duplicates:** Exact duplicate rows were dropped to avoid training bias from repeated data.

- **Handled Missing Target Values:** Rows with missing values in the target column 'Target Pressure (bar)' were excluded because they cannot be used in supervised learning.

- **Dropped Empty Columns:** Columns containing only missing values were identified and removed.

- **Corrected Invalid Temperature Entries:** Negative values in temperature columns (which are invalid in Kelvin) were replaced with NaN so they could be properly imputed later.

2) Data Processing: Detail the preprocessing methods applied, such as normalization, feature engineering, and data type conversion. Provide justifications for each step.

- **Feature Selection:**
The columns *Sensor ID* and *Status* were dropped before training because they either identify the sensor (which may cause data leakage) or represent categorical/status information not directly useful or consistent for the regression task.

- **Feature Engineering:**
Created new features based on domain knowledge to enhance model input:

  - *Tank Aspect Ratio* (width divided by length) captures tank shape characteristics affecting pressure.
  - *Tank Volume (m3)* (width × length × height) represents the physical volume of the tank, important for pressure prediction.

- **Train-Validation-Test Split:**
Data split into training (~70%), validation (~15%), and test (~15%) sets using a fixed random seed to ensure reproducibility and unbiased evaluation.

- **Missing Value Imputation:**
Missing feature values imputed with the mean of the training set to maintain data consistency and prevent leakage.

- **Feature Scaling (Normalization):**
Applied standard scaling (zero mean, unit variance) to numerical features using `StandardScaler` to stabilize and speed up model training.

Model Selection

To evaluate the performance of different machine learning approaches for predicting peak pressure in BLEVE scenarios, we considered three distinct types of models: **Linear Regression**, **Random Forest**, and a **Neural Network (MLP Regressor)**.

1. Linear Regression

Reason for Selection:
Linear Regression was chosen as a **baseline model** due to its simplicity and interpretability. It assumes a linear relationship between features and the target variable, allowing us to assess how well a straightforward model performs on the dataset.

Suitability:
While it may not capture complex interactions in the data, it is fast to train and provides a useful benchmark against which more advanced models can be compared.

2. Random Forest Regressor

Reason for Selection:
Random Forest is a **non-linear**, **tree-based ensemble** model that can model complex relationships and interactions between features. It is robust to noise and handles both numerical and categorical features well.

Suitability:
Given the physical and structural complexity of BLEVE scenarios, Random Forest is well-suited to capture non-linear patterns in the dataset and has proven effective in many real-world regression tasks.

3. Neural Network (MLP Regressor)

Reason for Selection:
A Neural Network, specifically a Multi-Layer Perceptron (MLP), was chosen to model **highly non-linear patterns**. It is a powerful model that can approximate any function given sufficient data and tuning.

Suitability:

Since the dataset includes many interacting physical variables, a neural network is a suitable choice for potentially achieving the **highest accuracy**, though it requires more **computational resources and tuning** than the other models.

Rejected Models

1. Support Vector Regression (SVR)

- **Reason for Rejection:** While SVR is effective for high-dimensional spaces, it **scales poorly** with larger datasets and is computationally expensive to train.

2. Logistic Regression

- **Reason for Rejection:** Logistic Regression is inherently a **classification model**, not suited for regression tasks like predicting peak pressure (a continuous variable).

3. Gradient Boosting Models (e.g., XGBoost, LightGBM)

- **Reason for Rejection:** These models are **too similar in nature** to Random Forests (i.e., tree-based ensembles), and using both would not meet the "distinct model types" requirement.

4. K-Nearest Neighbors (KNN) Regression

- **Reason for Rejection:** KNN struggles with **high-dimensional data** and is **sensitive to feature scaling**, which can lead to poor performance if not tuned carefully.

6. Transformer Models (e.g., BERT, GPT-like models)

- **Reason for Rejection:** Transformers are designed for **sequential or textual data**, such as time series or natural language. They require a lot of data and computation and are overkill for structured numerical datasets.

5. Convolutional Neural Networks (CNNs)

- **Reason for Rejection:** CNNs are specifically designed for **spatial data** (like images), where nearby features (pixels) have local relationships. our dataset is **tabular**, with no spatial layout to exploit.

Linear Regression

We trained a Linear Regression model using scikit-learn on the training data and evaluated its performance on the training, validation, and test sets before hyperparameter tuning.

Dataset	MAPE	R² Score
Training	1.2525	0.4349
Validation	1.2578	0.4812
Testing	1.2047	0.4508

Hyperparameter Tuning:

To improve the performance of the baseline Linear Regression model, we applied hyperparameter tuning focusing on Ridge Regression, which adds L2 regularization to reduce overfitting and improve generalization.

Method:

- We used grid search with 5-fold cross-validation to find the best regularization strength parameter, alpha.
- The candidate values tested for alpha were: [0.1, 1, 10, 100, 1000, 10000].

Results:

- The best alpha found was 100.
- Performance with the tuned Ridge model on the dataset was as follows:

Dataset	MAPE	R² Score
Training	1.1549	0.3974
Validation	1.1776	0.4376
Testing	1.1450	0.3974

Interpretation:

- Hyperparameter tuning with Ridge regularization improved model generalization, but overall model performance remained limited. This suggests that while regularization helps prevent overfitting, a linear model’s capacity may be insufficient for the complexity of the data
- Therefore, more sophisticated models such as Random Forests and Neural Networks were explored for better predictive accuracy.

Random Forest

We trained a Random Forest model using scikit-learn on the training data and evaluated its performance on the training, validation, and test sets before hyperparameter tuning

Dataset	MAPE	R^2 Score
Training	0.1325	0.9477
Validation	0.2280	0.8528
Testing	0.2506	0.8317

Hyperparameter Tuning:
Improve the predictive accuracy of the baseline Random Forest Regressor by tuning key hyperparameters to reduce overfitting and improve generalization.

Method:

- Used GridSearchCV with 2-fold cross-validation on the training set.
- Tuned the following hyperparameters:
  - n_estimators: Number of trees in the forest (tested 100).
  - max_depth: Maximum depth of each tree (tested 20 and None for unlimited depth).
  - min_samples_split: Minimum number of samples required to split an internal node (tested 2).
- Set max_samples=0.8 to train each tree on 80% of samples (introduces randomness for better generalization).
- Evaluated models using R^2 scoring.

Results:

- Best hyperparameters found by grid search: {'max_depth': 20, 'min_samples_split': 2, 'n_estimators': 100}

Dataset	MAPE	R^2 Score
Training	0.1006	0.9669
Validation	0.2114	0.8482
Testing	0.2311	0.8474

The tuned Random Forest model demonstrated strong generalization performance, with consistent metrics across training, validation, and test sets. The chosen hyperparameters allowed each tree to fully grow and learn complex relationships in the data

Neural Networks (Main Model)

We trained a Random Forest model using scikit-learn on the training data and evaluated its performance on the training, validation, and test sets before hyperparameter tuning

Dataset	MAPE	R² Score
Training	0.1088	0.9803
Validation	0.1823	0.9454
Testing	0.1905	0.9402

Hyperparameter Tuning:

To improve model performance beyond the baseline Multi-Layer Perceptron (MLP) using hyperparameter tuning. The aim was to optimize architecture depth, dropout rate, and learning rate for better generalization and lower MAPE.

Method: - Conducted a **grid search** using sklearn.model_selection.ParameterGrid over the following hyperparameters:

- layer1_units: [512, 256] — number of neurons in the first dense layer.
- layer2_units: [256, 128] — number of neurons in the second dense layer.
- dropout_rate: [0.2, 0.3] — applied to prevent overfitting.
- learning_rate: [0.0001, 0.001] — controls optimization step size.

- Used early stopping with patience=20 based on validation loss.
- Each configuration was trained for a maximum of 100 epochs with a batch size of 64.
- Performance was evaluated using **Mean Absolute Percentage Error (MAPE)** on the validation set.
- The target variable was scaled using StandardScaler for more stable training, and inverse transformed for evaluation.

Results: Best hyperparameter configuration found: {'layer1_units': 512, 'layer2_units': 256, 'dropout_rate': 0.2, 'learning_rate': 0.0001}

The model with these settings was selected as the final neural network model.

Result after Hyperparameter Tuning

Dataset	MAPE	R^2 Score
Training	0.0999	0.9816
Validation	0.1336	0.9499
Testing	0.1380	0.9460

Interpretation:

- The model demonstrates **excellent performance on the training set**, with a MAPE below 0.10. This indicates that the model has effectively learned the training data patterns without overfitting.
- The **validation and test MAPEs are closely aligned**, both around 13–14%, suggesting **strong generalization** to unseen data and that the model is not simply memorizing the training data.
- The **R^2 score of 0.9460** on the test set confirms that approximately **94.6% of the variance in target pressure values** is explained by the model, which is a strong indicator of model accuracy.
- The small gap between training and test performance also shows that the **regularization (L2) and dropout**strategies were effective in preventing overfitting.

Conclusion:

The final neural network model is **highly accurate and reliable**, achieving strong performance on unseen data.

Self Reflection

This project was honestly a mix of frustration and fun. One of the biggest challenges I faced was during the preprocessing stage. I kept getting stuck trying to figure out which columns to keep, which ones to drop, and how to clean the data properly. Some features looked important at first like Sensor Id and Staus, but later I realized they were actually adding noise. It took me quite a bit of time and trial-and-error to get it right.

I was also a bit overwhelmed when it came to choosing and tuning the model.I wanted to try doing it manually to understand how each setting like layer sizes, dropout rates, and learning rates affected performance which I did but also used GridSearch which helped me explore different hyperparameter combinations and understand how they affect model performance. Trying to figure out which hyperparameters are working the best and which are not took a long time and was hard but I learned a lot from the process.

Once the model started giving better results, it felt really rewarding. I was able to see how small changes in the architecture or data processing could make a big difference. It gave me a much clearer understanding of how machine learning models work beyond just running code.

If I were to do this again, I'd definitely plan things more clearly and maybe document each experiment properly instead of just tweaking things randomly. I also think I'd try a mix of manual and automated tuning to save time while still learning.

Overall, it was a challenging but valuable experience, and it really helped me understand how much thought and effort goes into building a good machine learning model.