Machine Learning Assignment Report

Figure 1, final model prediction score

NAME + STUDENT ID: Peter Misiun (20886060)

UNIT NAME + UNIT ID: Machine Learning, COMP3010

LAB TIME: Friday 2-4PM

DUE DATE: 4/05/2025

1: Introduction

This project aims to predict the BLEVE explosions using a variety of machine learning models. These explosions occur when gas tanks burst due to pressure and can be analysed using sensors placed around the tank, leading to insights of conditions that caused the explosions. In the goal to predict the peak pressure of the gas tanks, we create machine learning models. The steps involved in creating these models includes data cleaning and preprocessing, model development and final testing against a testing dataset. Based on metrics observed within each individual model and performance against the testing data, a final model will be chosen for use within this scenario. The creation of this final model can lead to improved safety and prevent potential future damages as predictions can assist in certain alterations necessary to prevent explosions from occurring. The following report outlines the various steps undertaken to produce the final model as well as the knowledge gained throughout the creation.

2: Data Cleaning and Preprocessing

The training data provided required extensive cleaning and processing before being able to be used for model development. The steps taken to do this in order were: Initial exploration, data type conversion, identifying and handling missing values, outlier detection and treatment, duplicate removal, correcting inaccurate entries, feature engineering, feature selection and feature scaling. A more in-depth run through of these steps is outlined below.

- **Initial Exploration:**
    - Loaded in the training dataset and used functions to get a better idea of the data provided. This allowed us to see the number of rows and columns within the data, as well as the data types of each of the columns. In addition to this, a brief scroll through of the data set within excel allowed for a better understanding of certain patterns found within it.
    - This was important as it led to better understanding of what steps to take next to make sure the training data was properly cleaned and processed for model development.

- **Data Type Conversion**:
    - During the exploration, it was found that the 'Status' column contained objects. This needed to be converted to numeric data to be used for the models. Also, it was found that some entries contained spelling mistakes, seeing 'superheat' and 'saperheated' instead of 'superheated'; and 'subcoled' and 'subcool' instead of 'subcooled'.
    - To combat this, one hot encoding was employed. This sees a function created to encode two new columns, namely 'Status_Superheated' and 'Status_Subcooled'. This sees statuses referring to superheated (and those with spelling mistakes), have 1 encoded into 'Status_Superheated' and 0 encoded into 'Status_Subcooled', and vice versa for subcooled.

- With this done, the status column was removed as it's no longer needed. In addition to this, rows with an invalid status were removed from the dataset for ease of use.

- **Identifying and Handling Missing Values**:
  - During the initial exploration of the data, it was uncovered that the 'Sensor ID' column had a reoccurring pattern in which it would be in ascending order from 1 to 27 and then restart. To combat missing values within this column, a function was created to go through the values within this column, and if a missing value was found, the number in between the previous sensor ID and next sensor ID was inputted into the missing entry.
  - After this, the program would go through the data to find how many rows had missing values and the number of missing entries for these rows. Once this was done, rows with 4 or more empty entries were removed from the dataset.
  - With this done, the median of each column was imported into its respective missing entry, leading to a complete dataset with no missing entries in any of the rows.

- **Outlier Detection and Treatment:**
  - To ensure the data was free of outliers, the program would go through each of the rows and calculate the Z-scores of each entry in respect to all the entries within its column. Any rows found that had an entry with a Z-score of 3 or more were removed. With the use of while loop, this would continue to iterate 9 times over the dataset until most of the outliers were found and removed.
  - Although outliers were removed through this process, the potential for removing certain rows with valuable insights is high. However, completing 9 iterations of outlier removal allowed certain models to perform much better as opposed to only running one iteration of outlier removal or until all outliers were removed, hence why it was necessary.

- **Duplicate Removal:**
  - Duplicate rows must be removed to not falsely alter the results of the models. This was done using a 'pandas' library function 'drop_duplicates()'. This allowed for easy removal of duplicates within the dataset.

- **Correcting Inaccurate entries:**
  - This step ensures that entries within the dataset that are impossible to achieve are removed. This was done by first creating a boolean dictionary, setting conditions for each of the columns. After this, rows with invalid entries were identified and removed. Information on which columns had inaccurate entries and the amount in each is displayed leading to a better understanding.

- **Feature Engineering:**
    - To get a more comprehensive outlook on the dataset, four new columns were created. These columns were ‘Tank W / L Ratio’, ‘Obstacle Volume’, ‘Vapour Ratio’ and ‘Temperature Difference’. The entries for these new columns are created through combining other column values within an equation to produce them. After the calculations are complete, the respective entries for each row are saved within new columns.

- **Feature Selection:**
    - This step ensures that only meaningful columns are kept within the dataset. As ‘Status’ has already been removed in an earlier step, we remove the ‘Unnamed: 0’ column as it’s simply just numbering the rows without any useful information.

- **Feature Scaling:**
    - Feature scaling is important as it ensures input features are on the same scale as some inputs might have values in ranges much greater than others, leading to models to be misled by focusing on the value size rather than what they mean.
    - The program employs a ‘StandardScaler’ from sklearn to scale all features within the dataset except for 'Target Pressure (bar)' (don’t scale target feature), 'Status_Superheated', 'Status_Subcooled', 'Sensor ID' and 'Sensor Position Side' (not scaled because they are categorical variables). This scaling of variables needs to be applied to a copy of the training data to avoid modifying training data directly before splitting.

3: Model Selection

In predicting peak pressure in BLEVE scenarios, a variety of machine learning models were required to properly uncover relationships found within the data. In doing this, a more in-depth outlook of the data is produced, leading to insights unlikely to be found by using only one model.

The three models implemented within this project are linear regression, random forest and a neural network, with support vector regression being strongly considered. More detail on the specifics of model selection is presented below:

- **Linear Regression:** This is presented as the simplest of the model’s implemented. This is because linear regression models a line of best fine using the intercepts and coefficients for the data (seen as the average), and predicts values based on this line. In doing so, a computationally efficient model is produced. However, despite being simple to implement, its shortcoming in assuming the data is linearly correlated leads to non-linear relationships being misrepresented, presenting challenges for certain data sets. Implementation of this model and its visualisations were important to the project as it clearly outlined the data’s areas of non-linearity and led to decisions on what other models to implement to address this.

- **Random Forest:** This model is presented as a middle ground in terms of complexity and computational demand within the models implemented. Random Forest achieves its strong predictive capabilities though combining the outputs of multiple decision trees built during training. Its implementation within the project led to non-linear interactions and the importance of certain features being uncovered, leading to stronger test data predicting capabilities.
- **Neural Network:** This is presented as the most complex of the models implemented. This is due to its similarity to a human brain, having the ability to learn and uncover complex relationships within the data. Although being computationally expensive and requiring extensive tuning, its strengths in predictive modelling and flexibility made it a necessary addition for the models implemented. With these characteristics, neural network models can uncover relationships missed by linear regression and random forest, making it the most powerful of the models implemented.
- **Support Vector Regression:** This model was considered but was not implemented in the end. SVR's ability to model non-linear relationships (using kernel trick) made it a strong contender for models implemented within the project, however, its poor scalability within large datasets led to other models taking priority in implementation.

4: Hyperparameter Tuning

Finding the optimal arrangement of hyperparameters is known as hyperparameter tuning. This process attempts to achieve the best predictive performance possible for each model, being measured through metrics such as MSE, MAPE and R^2. Linear regression does not implement hyperparameter tuning due to an absence of hyperparameters present, as well as no regularisation implemented. However, this process is relevant to the random forest and neural network models implemented. This process is outlined as follows:

- **Random Forest:** This model sees the number of trees within the forest (n_estimators) and the maximum depth of each tree (max_depth) being tuned. For both hyperparameters, multiple different values were tested using a manual grid search and the combination that produced the best performance metrics were used.
    - **Hyperparameter n_estimator:** The values of 10, 50 and 100 were tested. This provided a wide range of different outcomes in terms of training and evaluation, allowing for models with increasing complexity and a higher likelihood of better performance without overcomplication.
    - **Hyperparameter max_depth:** The values of 10, 20 and none (unlimited) were tested. This provided a wide range of different outcomes in terms of training and evaluation, allowing for models with increasing complexity and a higher likelihood of better performance without overcomplication.
    - **Manual grid search:** This allowed all possible combinations of hyperparameters to be tested, leading to a total of 9 different configurations.

- **Evaluation:** Using metrics (MSE, MAPE, R^2) produced through testing the different combinations on the validation set, the optimal parameters could be selected.
- **Final optimal parameter values:** After evaluation, the hyperparameter values that displayed the best performance were n_estimators = 100 and max_depth = 20. This translates to a forest starting with 100 trees and a max tree depth of 20.

- **Neural Network:** This model sees the hyperparameters of the number of layers, neurons per layer, activation function, dropout rate, learning rate (optimiser), batch size, epochs, early stopping patience and loss function. Within the code, there isn't any explicit hyperparameter tuning. This is because after running once, re-running the code would take too long to find parameters. Therefore, code shows manually set parameters. The tuning process will be outlined in more detail as well as other values tested for in the following:
  - **Number of layers:** A layer represents a group of neurons which process data and output information to the next layer within the network. Our code represents 3 hidden layers within the neural network architecture. This was tested with values from 1-5, however, 3 provided the best results without overcomplication and minimised overfitting. Using 3 layers also mitigated the effects underfitting, caused by too few layers (such as 1 or 2).
  - **Number of neurons per layer:** A neuron represents a unit that receives inputs and completes operations, outputting information toward the next layer within the network allowing the model to learn. Within the 3 hidden layers, the first has 128 neurons, the second has 64 and the third has 32. The different values tested for this were 16, 32, 64, 128 and 256, each value corresponding to its own respective layer. Once again, values 32, 64 and 128 provided the best results without overcomplication and minimised overfitting. Using these values also mitigated the effects of underfitting, caused by too few neurons (such as 16).
  - **Activation function:** This is the function applied to neurons to help the model learn complex patterns. The ReLU activation function has been used for the hidden layers. The sigmoid function was also considered but was not implemented within the project due to ReLU being more suitable for moderately deep architecture (3 layers), its alignment with regression tasks and its less demanding computation.
  - **Dropout rate:** This parameter sees the percentage of neurons deactivated at random for each training iteration, leading the model to have less reliance on specific neurons. A dropout rate of 0.2 was used for both dropout layers, with values 0, 0.1, 0.2, 0.3 and 0.4 considered and tested. We see a value of 0.2 being optimal as it allows the model to truly learn, without the risk of underfitting from a high dropout rate (>0.3 means the model will uncover less relationships), or overfitting from a low dropout rate (<0.2 means the model will be too reliant on training data).

- **Learning rate (optimiser):** This is the size of the steps whilst adjusting weights during training. The learning rate of the model is shown through 'adam', with a default learning rate of 0.001. The range of values tested were 0.0001, 0.001, 0.01 and 0.1. The value of 0.001 provided a balance of training speed and convergence, whereas values such as 0.0001 has step sizes which were too small, leading to slower convergence; and a learning rate of 0.1 was too large, leading to instability and divergence due to large weight updates.
- **Batch size:** The batch size is the amount of training samples observed before adjustment of weights for a single training iteration. The batch size within the model was 32, with values 16, 32, 64 and 128 considered. This value was used due to its efficiency and stability. It also aligns with the size of the dataset, with values such as 16 being too small, capturing more noise and being slower to run, whereas values such as 128 are too large and lead to underfitting.
- **Epochs + early stopping:** During neural network training, an epoch is a complete pass through of the dataset, whereas early stopping represents the amount of non-improving validation loss stop epochs required to stop training early. The only values tested for epochs were 50 and 100. 50 epochs showed that the training would stop at this iteration, whereas with 100, it would stop prior to reaching 100 epochs. This led to using 100 as it showed the model being properly trained as needed. The only value tried for early stopping was 10. This provided confidence that minimal training improvements could be added once 10 epochs were ran without improvements.
- **Loss Function:** This is a function that uses the predicted outputs and target values to find the difference between them, ensuring training optimisation. MSE was the only loss function considered and used within the neural network model. This is due to its ability to penalise errors, leading to more accurate predictions.
- **Search Strategy:** As stated previously, the selected hyperparameter values (outlined above), have been implemented as to avoid extended running times during the operation to find optimal parameters. These parameters have been manually coded after running these searches to save time. In addition to this, the search strategy would often incorporate values that overcomplicated the model and caused over fitting, leading to poor performance on test data. To combat this, values for the neurons per layer and dropout rate were chosen based on good performance and led to a decrease in overfitting.

5: Prediction Performance

Below outlines the performance of each model according to metrics calculated and performance against the test set. All values round the nearest 4 decimal place.

	Linear Regression	Random Forest	Neural Network
Training MSE	0.0095	0.0004	0.0015
Validation MSE	0.0091	0.0029	0.0021
Training MAPE	0.5194	0.0711	0.1677
Validation MAPE	0.5138	0.1874	0.1847
Training R^2	0.6566	0.9844	0.9467
Validation R^2	0.6581	0.8902	0.9193
Testing MAPE	0.5772	0.3459	0.2476

Figure 2, table outlining performance metrics of different models

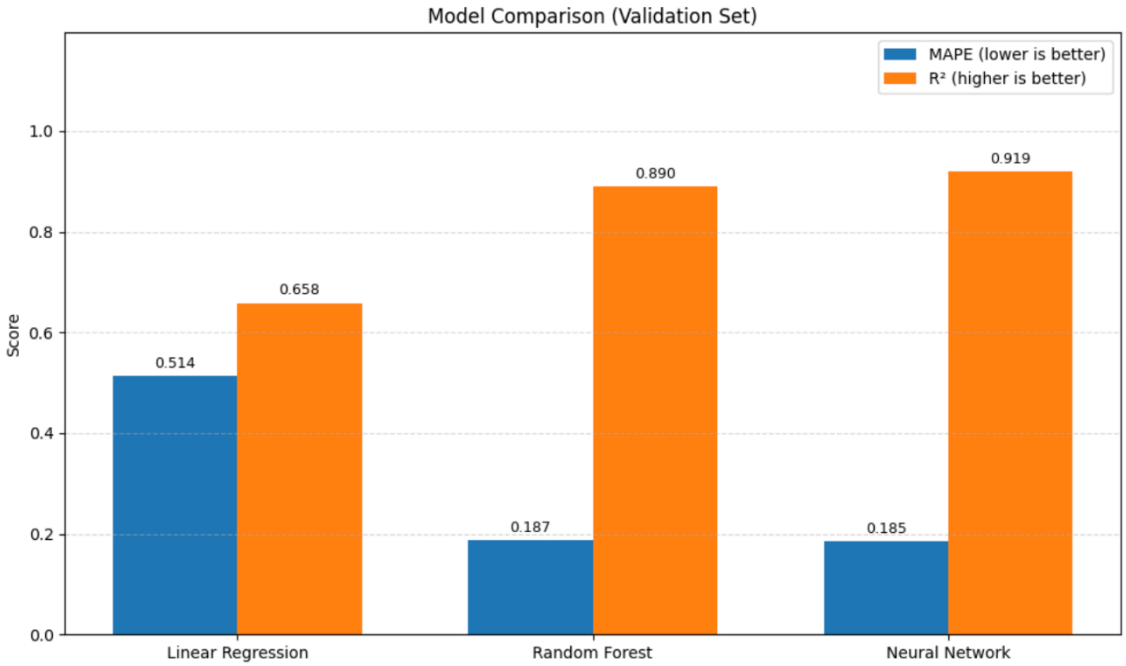

Figure 3, visualisation outlining performance metrics of different models

From the above information, it is seen that the neural network model performed the best against the test data and achieved the best validation metrics (lowest MAPE and highest R^2). Therefore, this model will be used as the final model within the project with its test prediction data included as 'prediction.csv'. Note that some of the code output within the Jupyter notebook might be different to the information above as some extra tuning was attempted before submission to no avail. In addition to this, another prediction for the neural network titled 'nn_predictions11.csv' achieved a test score of 0.2269. However, the exact values used to achieve this in terms of data preprocessing and tuning cannot be remembered. Therefore, it was most likely just good luck with the neural network training process.

6: Self Reflection

All in all, this project has allowed for a true understanding of the practical applications of machine learning within today's society, presenting itself as crucial for the betterment of many.

This project highlighted both the challenges and value of working with real-world data. This is shown as cleaning the dataset required addressing multiple issues. These issues include organising sensor IDs in ascending order from 1-27 and handling spelling errors within the 'Status' column. In addition to this, some background research was required whilst attempting to remove inaccurate entries, gaining knowledge to uncover which values truly were impossible to reach.

The modelling phase proved equally demanding. Testing three fundamentally different models and tuning their parameters demonstrated how sensitive model performance is to proper configuration. The experience reinforced how critical clean data is for reliable predictions and how machine learning can provide practical safety solutions.

In the future, the incorporation of more advanced modelling techniques would prove viable to improve the predicting capabilities required. Also, conducting deeper domain research before beginning data processing and model development would allow for a better understanding of the problem and the best course of action to complete it.

7: Conclusion

In conclusion, this project provided valuable insights into the iterative nature of machine learning workflows and the importance or seemingly menial tasks. This project is truly an eye-opening experience, allowing for a true appreciation of the usefulness of predictive modelling in solving real world problems, potentially presenting benefits to all parts of society.