Evaluation of Various Statistical Methods for Data Imputation

Greg DePaul, Mel Johnson, Alex Kermani, Samuel Van Gorden

June 2023

Abstract

The scope of this project involved applying a variety of data imputation strategies to a synthetic dataset which used machine learning techniques to create a similar dataset with randomly selected missing values. Our goal was to consider the variable "x_e_out" from the dataset as our output and the remaining variables as inputs, impute the missing data from the inputs (or work around the missing data), and use these inputs to predict the missing output values. We attempted imputation on the inputs using standard techniques such as replacement with mean or median and then used linear regression and neural networks as predictive models for the output. We also used several "variable-length" techniques which allowed us to do prediction of the output with any missing input variable entries excluded from the model. Our results, measured by root mean square error between the predicted and actual output values, showed that imputation of the input variables by mean and median worked best for the linear regression model, imputation by mode worked best with neural networks, and BERT was the best variable-length technique for this particular application, which is surprising given it's not an application typically used for numerical data.

1 Introduction

Our study looks at data generated using a deep learning model performed on the dataset from the study "On the prediction of critical heat flux using a physics-informed machine learning-aided framework" [1]. This data was generated with missing values (around 15% of the total for each variable) as part of a Kaggle competition in which the goal was to implement data imputation methods to obtain the missing values of the target variable x_e_out. Participants could then upload their predictions for the missing values and the competition was judged based on the root mean square error (RMSE) of the predicted values compared to the actual values.

One unique perspective in this competition is comparing fixed-length and variable-length imputation methods. Fixed-length data is a common case with imputation whereas variable-length data is less common. Our motivation for this project comes from having to deal with missing data in nearly all data-oriented endeavors. Real-world examples of this include cases in which researchers need to combine heterogeneous datasets that may or may not share variables; or where data is missing due to noise, participants dropping out of longitudinal studies, and errors in data collection.

Variable-length models are models that can take a variable number of parameters for calculating each response. This is useful when looking at imputation for datasets with missing values across several variables because we are able to leave out any variables with missing values for each observation we use to train the model and still perform prediction.

The fixed-length methods used included replacing missing input values (all but x_e_out) with the mean, median, maximum, minimum, mode, or fixed value. The missing values of x_e_out were then predicted using either linear regression or a neural network. The variable-length models used were conditional random fields (CRF), long short-term memory (LSTM), and bidirectional encoder representations from transformers (BERT).

The competition on Kaggle allowed participants to submit up to five predictions each day, so we were able to determine the RMSE of all of our models based on the actual datasets held by the owners of the Kaggle competition. We found that imputing missing data with modes and training a neural network gave us the overall best RMSE.

2 Description of Data

The dataset we used was provided by Kaggle and generated using a separate dataset from a different Kaggle project. These data are completely synthetic. The original study had to do with measuring critical heat flux of various solids (variable "chf_exp [MW/m2]") and utilizing prior domain knowledge, such as materials physics, and machine learning techniques to fill the gap in the results predicted from domain knowledge and actual results. In our case "x_e_out [-]" is the output variable.

The data set has uniformly removed values from each variable, making the data **missing completely at random**. This means that the missing observations were independent of the observed variables as well as unobserved parameters and that standard data imputation techniques can be used without introducing bias or statistical invalidity in our imputation [2].

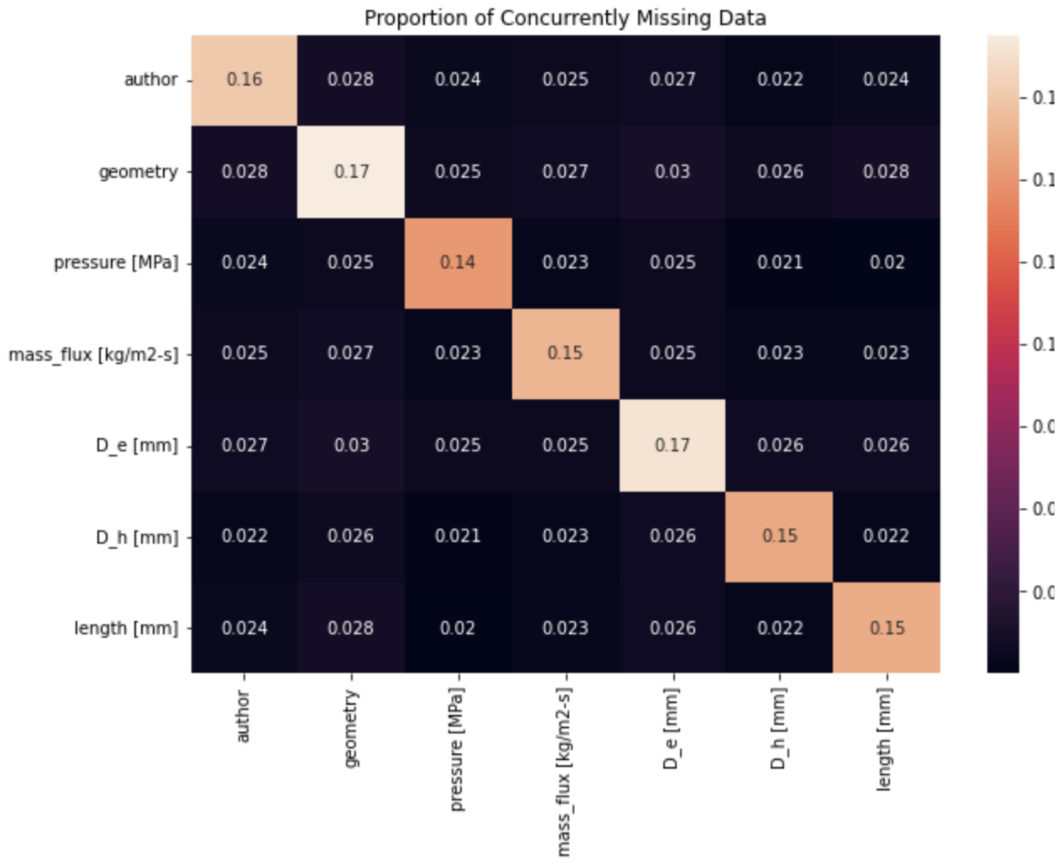
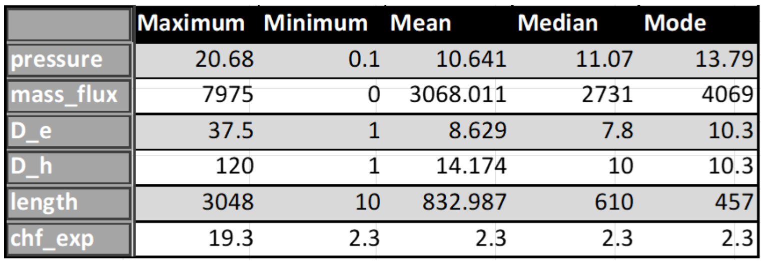


Figure 1: Fraction of missing data for each feature-to-feature combination.

The variables used in this dataset are publication author, the heating element geometry, pressure, mass flux, equilibrium concentration (our output, x_e_out), apparatus diameter, heated diameter, heated length, and critical heat flux. Below we provide summary statistics for all of the non-categorical predictor variables, which will also be used in our fixed-length imputation techniques:



	Maximum	Minimum	Mean	Median	Mode
pressure	20.68	0.1	10.641	11.07	13.79
mass_flux	7975	0	3068.011	2731	4069
D_e	37.5	1	8.629	7.8	10.3
D_h	120	1	14.174	10	10.3
length	3048	10	832.987	610	457
chf_exp	19.3	2.3	2.3	2.3	2.3

Figure 2: Summary statistics for the heat flux dataset features

While mean, median, maximum, and minimum do not apply to our categorical variables, "author" can take values ["Thompson", "Janssen", "Weatherhead", "Beus", "Williams"], with a mode of "Thompson", and "geometry" can be ["tube", "annulus", "plate"], with a mode of "tube".

3 Methods

3.1 Fixed-Length Methods

We began our initial tests using fixed-length models with standard imputation techniques such as utilizing the mean, median, mode, or a fixed value (e.g., -1, 0, or 1). These techniques were used to impute missing numerical values of all variables aside from the target variable (x_e_out). We then used the complete dataset of features in which missing values were imputed as independent variables to solve for x_e_out by implementing either a linear regression or a neural network model.

3.1.1 Imputation Statistics

- **Maximum and Minimum** - The maximum and minimum values are very simple statistics that are easy to compute as they only require a single sort to obtain both. They are likely to introduce a high amount of bias, however, and they are a bad representation of a dataset.
- **Mean Value** - The sample mean, defined as the sum of the sample values divided by the number of values in the sample, is often used in descriptive and inferential statistics due to the fact that it is easily calculable compared to other statistics. However, the mean is a non-robust statistic, meaning it is highly affected by data outliers.
- **Median Value** - The median, which is any value for which exactly half of the observations in a sample are greater than and exactly half are less than, can be a useful statistic for determining the average of a sample, especially when the data is skewed. It is a robust statistic, though it suffers from not having a simple closed-form representation. It also doesn't have a single value in some cases (e.g. the median of [1, 2, 3, 4] is any real number between 2 and 3).
- **Mode Value** - The mode is the value that occurs the most in a given sample. It can be useful when dealing with categorical or ordinal data for which the concepts of mean and median either don't exist or don't make sense. It is a robust statistic; however, it suffers when used on a sample with few repeated values or many values that repeat the same number of times.
- **Fixed Value** - Fixed value imputation involves replacing every missing value with a single chosen value, such as 1, 0, or -1. This can be useful because (assuming the fixed value does not actually occur among the non-missing values) we are able to determine which values are real and which ones are imputed from their value alone. However, it is not a very well-informed statistic since we don't use any of the existing data as we do with the aforementioned statistics.

3.1.2 Inference Models on Structured Data

- **Linear Regression** Linear regression is one of the most basic methods used for generating inferential models. These methods look to describe the variable relationships by fitting a line ($y = mx + b$) to the observed data. This method includes several assumptions, which are: there is a linear relationship between the dependent and independent variables, observations are independent, the dependent variable follows a normal distribution for any predictor value, and variability of the dependent variable is uniform across the independent variable [3]. While there were several drawbacks to using a linear regression model for this application, it serves as a good standard of measurement due to its relative simplicity and ease of implementation. This model can be represented symbolically by:

$$\text{x\_e\_out} = \underbrace{\begin{pmatrix} \ddots & \dots & \dots \\ \dots & A & \dots \\ \dots & \dots & \ddots \end{pmatrix}}_{1\times 19} \begin{pmatrix} \text{author} \\ \text{geometry} \\ \text{pressure} \\ \text{mass flux} \\ \text{D}_\text{e} \\ \text{D}_\text{h} \\ \text{length} \end{pmatrix}$$

- **Neural Network**

  A natural extension of linear regression is to use multiple matrices linked together by nonlinear activation functions. We chose to fix the same architecture for all imputation methods, using early stopping for each method. Symbolically:

$$\text{x\_e\_out} = \tanh \circ C_{1\times 8} \circ \text{relu} \circ B_{8\times 64} \circ \text{relu} \circ A_{64\times 19} \circ \begin{pmatrix} \text{author} \\ \text{geometry} \\ \text{pressure} \\ \text{mass flux} \\ \text{D}_\text{e} \\ \text{D}_\text{h} \\ \text{length} \end{pmatrix}$$

  which visually has the structure:

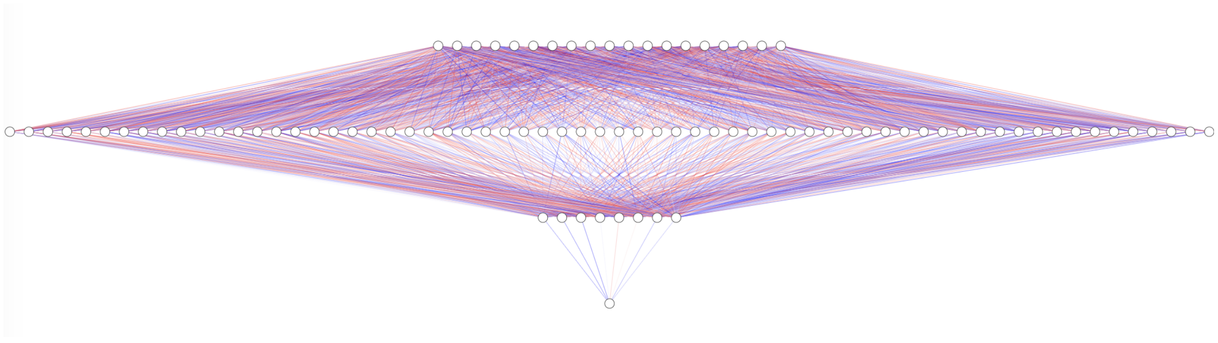


Figure 3: Neural network architecture used in x_e_out prediction

3.1.3 Unused Methods

Other standard imputation techniques considered were K Nearest Neighbors (KNN), linear interpolation, or using the next and previous values to predict the non-target features to be used as independent variables. However, many of these methods include assumptions which were not considered to be valid.

Specifically, KNN, linear interpolation, and next/previous values assume that the data is series dependent, but information provided by the supporting literature suggests the data is not series dependent [1]. The benefits and weaknesses of the imputation techniques varied for each imputation statistic used. The largest opportunity provided by fixed-length methods is the speed at which they are able to be implemented; this allows them to serve as a standard in determining the time-cost benefit of more detailed methods such as CRF and BERT.

3.2 Variable-Length Methods

Since our dataset has randomly missing values, no two inputs are guaranteed to be the same length. As such, we explored variable-length methods for predicting x_e_out. Instead of assuming the missing values, we developed models to generate output based solely on available data. This is accomplished using many-to-one models, in which a variable-length sequence is fed into a single model.

3.2.1 CRF

The first variable-length model we explored was a Conditional Random Field, which is a type of probabilistic graphical model first introduced in 2001 by Charles Sutton and Andrew McCallum [4]. It is typically used for Named Entitity Classification tasks, such as predicting the part of speech of words based on surrounding words in a sentence.

CRFs are constructed by applying a conditional probability $p(y|x)$ to a graphical structure. Because of this, it is considered a discriminative model and we don't need to consider the relationship between input variables (x). Instead we have a factor graph, in which each output variable is connected to input and other output variables by way of factor nodes, which describe their relationship.

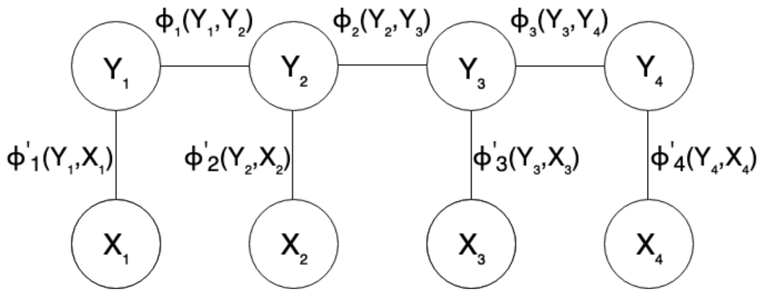


Figure 4: Graphical model representing CRF. The ϕ's represent the factor nodes (i.e. relationships between input/output variables). Notice that inputs (x variables) are not connected.

In our case, we treat each observation as a word, and the columns as features of the word. This model then goes by "moving window" over each observation to probabilistically infer the target variable. The value of x_e_out (y_i in our model) depends on the previous values of x_e_out (y_{i-1}) and the rest of the columns (features in x_{i-1}), the current observation's non-x_e_out columns (features in x_i), and the next values of x_e_out (y_{i+1}) and the rest of the columns (features in x_{i+1}).

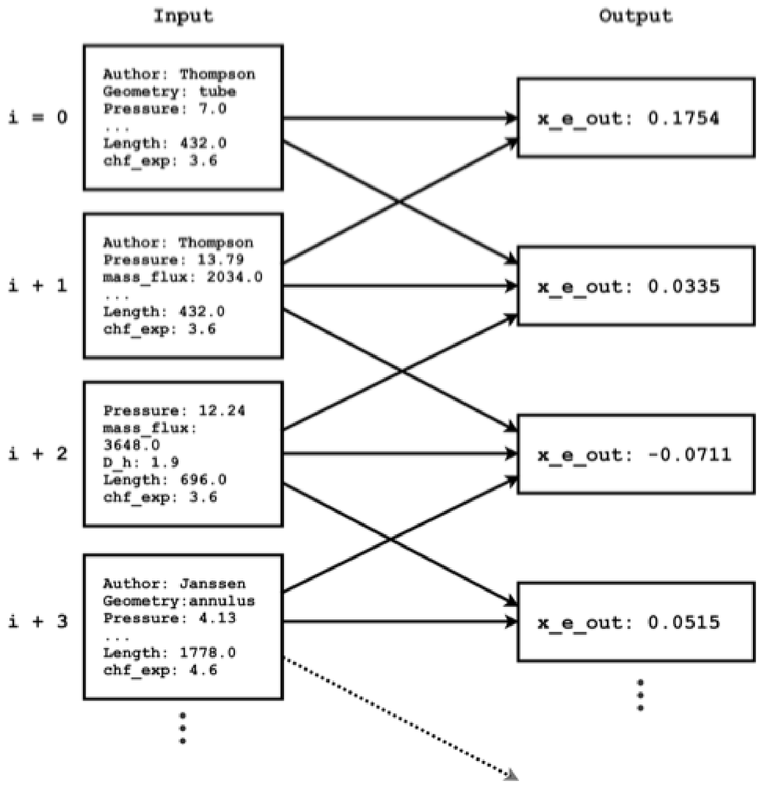


Figure 5: Block diagram for using CRF in our particular application.

We utilized the sklearn_crfsuite library in Python to fit our model and obtain predictions. While we first attempted to train the model on all observations for which x_e_out was non-missing (∼20,000 observations), the model fitting did not complete after being run for over 30 hours so we abandoned this attempt. Instead we randomly selected 5000 observations for which x_e_out was non-missing and fit our model on these observations. We ran this procedure several times, observing similar RMSE results each time. The fact that we had to only use a subset of the data could be one reason this model performed worse than the others. It would be interesting to see how this model performed if we had enough time, computing power, or a package similar to sklearnex which speeds up the execution of the sklearn library (sklearn_crfsuite is not actually part of the sklearn library) to utilize the full dataset.

3.2.2 LSTM

The second variable length model we explored was to use an LSTM which was invented in 1995 by Sepp Hochreiter and Jürgen Schmidhuber [5]. This is a recurrent network, taking in a sequence with an assumed max-length, although padding is used in order to allow for sequences less than the maximum length. In this specific heat flux example, features are encoded into a float64 array.

An LSTM then reads every word of the input and maintains the memory of already-seen features. It then exposes 64 valued features which are then fed into a dense linear perceptron. The activation function for the final output is the hyperbolic tangent.

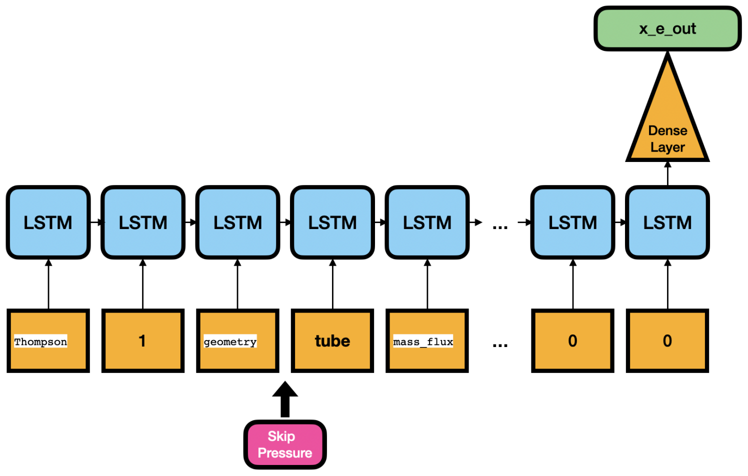


Figure 6: LSTM network with variable-length input. Note that the pressure feature is removed because it is not available.

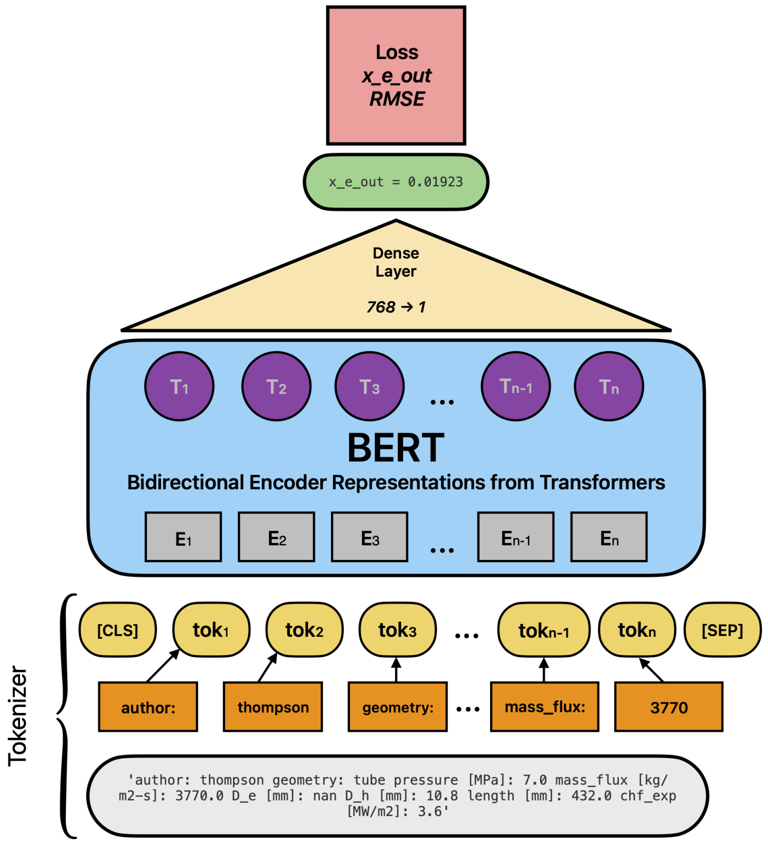
3.2.3 BERT

The final model we evaluated is called Bidirectional Encoder Representations from Transformers, or BERT. Published in 2018 [6], this model has dominated Kaggle Competitions since its release. BERT is a deep learning model widely employed in natural language processing tasks such as text classification and masked-word prediction.

However, whether this model is a suitable technique for imputing numerical data is unclear. Using BERT to impute numerical data has some potential benefits, primarily because of its ability to handle imbalanced, incomplete, and complex datasets. BERT can identify patterns within the inputs and infer the missing values based on these patterns. These benefits make BERT an interesting tool for imputing numerical data. However, BERT was not designed for numerical data, and other drawbacks of this model include high resource use and unclear interpretation of output data.

In order to harness the benefits and stymie the drawbacks, we made a modification to the BERT structure by connecting the BERT output to a single linear layer. The output of the linear layer then directly corresponds to our target variable (`x_e_out`). By making this modification, we can train our BERT model to identify relationships between the input tokens and the target variable without directly contending with the fact that this is not a typical model application. Figure 7 illustrates the model architecture and modifications.

While BERT has some benefits for imputing numerical data, it is not a purpose-built solution. Its performance depends on the specific problem requirements, data characteristics, and available resources. Thus, we sought to apply a unique and powerful model to explore the predictive power of BERT for numerical imputation.



Loss
x_e_out
RMSE
x_e_out = 0.01923
Dense Layer
768 → 1
T1
T2
T3
...
Tn-1
Tn
BERT
Bidirectional Encoder Representations from Transformers
E1
E2
E3
...
En-1
En
[CLS]
tok1
tok2
tok3
...
tokn-1
tokn
[SEP]
author:
thompson
geometry:
...
mass_flux:
3770
'author: thompson geometry: tube pressure [MPa]: 7.0 mass_flux [kg/m2-s]: 3770.0 D_e [mm]: nan D_h [mm]: 10.8 length [mm]: 432.0 chf_exp [MW/m2]: 3.6'
Tokenizer

Figure 7: Input and output data flow for the fine-tuned BERT model with an additional dense linear layer to collapse BERT output into a single value (`x_e_out`). E is an encoder, T is a transformer, RMSE is root mean squared error and is calculated on the value of `x_e_out`.

4 Results

In order to avoid cheating in Kaggle competitions, two datasets exist for evaluation. The first is a public test set in which you can submit 5 times a day to see updates on the leaderboard. The other is a private dataset that is only released when the competition ends. The private dataset determines who is the winner of the competition. We have tabulated the results of our models within the following tables. We placed 601 out of 693 teams, and have included the winning scores for comparison.

4.1 Traditional Imputation Results

Method	Public (Linear)	Private (Linear)	Public (NN)	Private (NN)
Maximum	0.093887	0.089644	0.094406	0.089772
Minimum	0.091199	0.08918	0.094219	0.093771
Mode Value	0.090305	0.08713	**0.083461**	**0.0805**
Mean Value	0.089786	**0.086574**	0.092571	0.089797
Median Value	**0.089702**	0.086661	0.083511	0.081384
Fixed Value	0.09104	0.089032	0.084949	0.083468
Competition Bests	0.074167[a]	0.072485[b]	0.074167[a]	0.072485[b]

[a] Team's solution: gradient boost regression
[b] Team's solution: tree-based, NN ensemble

4.2 Variable Length Results

Method	Public Test Set (Linear)	Private Test Set (Linear)
CRF	0.104095	0.102624
LSTM	0.092106	0.089368
Bert	**0.084648**	**0.088410**
Competition Bests	0.074167[a]	0.072485[b]

[a] Team's solution: gradient boost regression
[b] Team's solution: tree-based, NN ensemble

5 Discussion

5.1 Fixed-Length Methods

The linear regression-based fixed-length models we developed were found to score better than several models developed by other participants. Our best fixed-length method was found by utilizing the mode to impute for non-target features and training a neural network model to solve for the values of target feature (x_e_out). This result was expected because the linear regression model seeks to find the best-fit to a line while the neural network model seeks to predict the missing value over a more generalized function space. While the data may be dependent, it does not necessarily reflect a linear relationship between the independent and dependent variables. Further, categorical features such as 'author' and 'geometry' may be important factors in determining an accurate prediction, but only numerical values could imputed when using linear regression. Categorical features were included in the neural network model by one-hot encoding.

5.2 Variable-Length Methods

We found that the modified BERT model outperformed other variable-length models (CRF and LSTM) in predicting x_e_out, with the lowest RMSE for both the public and private test sets. This result suggests that the BERT model with modifications is a formidable tool for making connections between data and predicting a numerical value based on these connections.

BERT is a transduction model that uses multi-layered bidirectional transformers to capture relationships between input tokens; this may make it more effective in identifying patterns within the inputs and inferring the target output. Said differently, BERT considers all inputs simultaneously without relying on data dependencies. CRF and LSTM are sequence-based models that rely on the order and relationship between input data points to make predictions. In other words, the models require that the input data is processed sequentially, and each output calculation depends on the previous input data points.

With CRF, a sequence of observations is represented as a graph, where each observation is a node, and the graph encodes the dependencies between the observations. For example, in a natural language processing task, CRF may predict the probability of the next word based on the previous words in a sentence. In the case of the heat flux dataset, CRF predicts the missing target variable based on the relationship between the previous, current, and next observations.

LSTM is also a sequence-based model that relies on the dependencies between inputs. It is designed to capture long-term dependencies by using a memory cell and a set of gates to determine how much of the previous hidden state is relevant for the current state. This enables the model to remember important information over a longer time horizon.

While relying on data dependencies is advantageous in some cases, it also has limitations. For instance, the order in which the data points are processed can impact the model's performance, and the model may not be able to capture the relationships between input values that are non-linear or involve multiple variables.

BERT's superior performance could be attributed to its bidirectional encoder/transformer architecture, which enables it to capture complex relationships and patterns within the inputs and infer the missing values based on these patterns.

The next factor that may have contributed to BERT's performance is the availability of pre-trained models. BERT is a pre-trained model that was trained on a massive amount of unlabelled text data; this makes it possible to use transfer learning for other tasks such as numerical imputation. Transfer learning allows the model to apply the knowledge acquired from pre-training to specific tasks, thus reducing the need for large amounts of labeled data and training time. We exploited that nature and fine-tuned our BERT model on a corpus of about 21,000 observations with output variable known.

In addition, CRF and LSTM may not be able to handle the characteristics of the dataset as effectively. For example, the heat flux dataset may have had a similar amount of missing data for each feature column, but the value of each feature is not identical. Certain values were more predictive of our target variable (e.g., length, mass flux, etc.) than others (e.g., author). Despite the features having a uniform number of missing observations, the information density of the missing data varies greatly. In that sense, the missing data are indeed imbalanced.

Finally, the modifications made to the BERT model structure may have also contributed to its superior performance. We connected the large BERT output to a single output linear layer that corresponds to x_e_out, enabling it to better capture the relationships between input tokens and the target variables.

Our modified BERT model is a promising candidate for additional tuning, including expanding the training time or data, cross-validation training, or additional linear layers (with or without dropout) to get a better final prediction.

5.3 Other Competition Methods

There were a total of 693 teams participating in the Kaggle competition, many of which implemented unique methodologies to solve the problem. The highest scoring participant paired domain knowledge with an ensemble of models that have diverse internal workings such as tree-based methods, neural networks, linear models, and nearest neighbors. The winner then utilized optuna to tune their model parameters and iteratively performed missing value prediction using trees. The winner went on to note that they did not find one-hot encoding for "author" and "geometry" features, or adding PCA features as useful for this dataset. Other models which scored in the top percentile of competitors included light gradient-boosting machine (LGBM) and Random Forest. Many of the top scorers utilized a blend of frameworks and were able to draw conclusions about the individual features through exploratory data analysis. Common takeaways from the top percentile of competitors included spending more time on EDA, utilizing an ensemble of models, and utilizing feature engineering by making assumptions about the data.

6 Conclusion

For the methods we explored in this paper, we found two stood our prominently. One was a standard neural network built over the imputed structured data, where the imputation statistics was the mode. The other method was the variable length method BERT, which offers the ease of not having to choose an imputation statistic *a priori*.

We learned that certain decision tree algorithms, specifically LGBM, Random Forests, and XGBoost perform exceptionally well. We did not explore decision trees in this project, but based on the top competition team's results, they tended to perform well. We can also explore the performance of standard imputation methods such as linear regression, neural networks and random forest models. By comparing these different approaches, we can identify which method produces the best results.

We could have further improved model performance by looking for simple relationships between features. The data were all randomly missing, but a domain expert may notice some relationships. For example, certain authors only worked with specific geometries; and for tubular geometries, two dimensions are always identical. By taking these relationships into account, we could have inferred missing data instead of imputing it, thus improving the quality of our input data "for free."

It's critical to continue learning about different algorithms, imputation methods, and ways of thinking about data. With practice and perseverance, we can create models that are increasingly accurate, robust, and effective. Overall, our participation in this competition provided us with a great opportunity to learn and grow as data scientists.

References

[1] Xingang Zhao, Koroush Shirvan, Robert K. Salko, and Fengdi Guo. On the prediction of critical heat flux using a physics-informed machine learning-aided- framework. *Applied Thermal Engineering*, 164:114540, jan 2020.

[2] Bhaskaran K and Smeeth L. What is the difference between missing completely at random and missing at random? *International Journal of Epidemiology*, 43(4):1336–9, Aug 2014.

[3] Nikolaos Pandis. Linear regression. *American journal of orthodontics and dentofacial orthopedics*, 149(3):431–434, 2016.

[4] Charles Sutton and Andrew McCallum. An introduction to conditional random fields, 2010.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

A Code Appendix

A.1 BERT Model

Import Packages

[1]:
```
import tensorflow as tf
import tensorflow_hub as hub
import numpy as np
import pandas as pd
import torch
from torch.utils.data import TensorDataset, DataLoader, random_split,␣
 ↪SequentialSampler, RandomSampler
from torchmetrics import MeanSquaredError

#
from tqdm.notebook import tqdm, trange

from platform import python_version, platform

from transformers import (
    BertModel,
    BertTokenizer,
    TFBertForMaskedLM,
    BertForMaskedLM,
    AdamW,
    get_linear_schedule_with_warmup,
)

from sys import executable
print(executable)
print("platform: {}".format(platform())) #Python Platform: macOS-13.3.
 ↪1-arm64-arm-64bit
print('python ' + python_version())
print(pd.__name__, pd.__version__)
print(np.__name__, np.__version__)
print(torch.__name__, torch.__version__)
has_gpu = torch.cuda.is_available()
has_mps = getattr(torch,'has_mps',False)
device = "mps" if has_mps \
    else "gpu" if has_gpu else "cpu"

print("GPU is", "AVAILABLE" if has_gpu else "NOT available")#GPU is NOT AVAILABLE
print("MPS is", "AVAILABLE" if has_mps else "NOT available") #MPS is AVAILABLE

print("target device is {}".format(device)) #Target device is mps

'''
Functions, important globals
'''
MAX_LENGTH = 128 # for sequence length
batch_size = 32 # for training

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

```python
token_encoder_fun = lambda row: tokenizer.encode(row,
                                                 add_special_tokens=True,
                                                 padding="max_length",
                                                 #return_tensors='pt',
                                                 max_length=MAX_LENGTH,
                                                 )


class BertHeatFlux(torch.nn.Module):
    def __init__(self):
        super(BertHeatFlux, self).__init__()
        self.bert = BertModel.from_pretrained('bert-base-uncased').to(device)
        self.dropout = torch.nn.Dropout(0.3)
        # x[1].shape[-1]
        self.linear = torch.nn.Linear(768, 1)
    def forward(self, input_ids, attention_masks):
        output = self.bert(input_ids, attention_masks)
        pooled_output = output[1]
        dropout = self.dropout(pooled_output)
        out = self.linear(dropout)
        return out
```

```
/Users/ask/anaconda3/envs/STA208_BERT/bin/python
platform: macOS-13.4-arm64-arm-64bit
python 3.9.16
pandas 1.5.3
numpy 1.24.3
torch 2.0.1
GPU is NOT available
MPS is AVAILABLE
target device is mps
```

Data Processing

Training/Testing Data

[2]:
```python
# Load the numerical data you want to train BERT on
df1 = pd.read_csv("data.csv")
df2 = pd.read_csv('Data_CHF_Zhao_2020_ATE.csv')
frames = [df1, df2]
df = pd.concat(frames)

# Define the name of the column that you want to move to the end of the DataFrame
column_name = "x_e_out [-]"

# Select the column and drop it from the DataFrame
column_to_move = df[column_name]
col = df.drop(column_name, axis=1, inplace=True)

# Append the column back to the end of the DataFrame
df[column_name] = column_to_move

# select only rows where x_e out exists
#data = df[df["x_e_out [-]"].isna()]
data = df[~df["x_e_out [-]"].isna()]
```

```
# start with a small data set for speed
#data = data[0:1000]
data = data.reset_index(drop=True)

# Convert numerical values to string format to match BERT input requirement
data = data.astype(str)

data["sequence"] = ""

# Concatenate all the values in a row into a single string using the column names
# Iterate through rows and columns
for index, row in data.iterrows():
    string = ""
    for column in data.columns:
        if column == "x_e_out [-]":
            # Do not add mask tokens, simply ignore
            #string += column + ": " + '[MASK]'*4 + " "
            continue
        if column == "sequence" or column == 'id':
            continue
        string += column + ": " + str(row[column]) + " "
    masked_string = string.strip()
    data["sequence"][index] = masked_string

data.describe
```

[2]:

```
<bound method NDFrame.describe of          id       author geometry pressure
[MPa] mass_flux [kg/m2-s]  \
0         0      Thompson     tube            7.0                 3770.0
1         1      Thompson     tube            nan                 6049.0
2         2      Thompson      nan          13.79                 2034.0
3         3          Beus  annulus          13.79                 3679.0
4         5           nan      nan          17.24                 3648.0
...     ...           ...      ...            ...                    ...
23089  1861  Richenderfer    plate           1.01                 1500.0
23090  1862  Richenderfer    plate           1.01                 1500.0
23091  1863  Richenderfer    plate           1.01                 2000.0
23092  1864  Richenderfer    plate           1.01                 2000.0
23093  1865  Richenderfer    plate           1.01                 2000.0

      D_e [mm] D_h [mm] length [mm] chf_exp [MW/m2] x_e_out [-]  \
0          nan     10.8       432.0             3.6      0.1754
1         10.3     10.3       762.0             6.2     -0.0416
2          7.7      7.7       457.0             2.5      0.0335
3          5.6     15.2      2134.0             3.0     -0.0279
4          nan      1.9       696.0             3.6     -0.0711
...        ...      ...         ...             ...         ...
23089     15.0    120.0        10.0             9.4     -0.0218
23090     15.0    120.0        10.0            10.4     -0.0434
23091     15.0    120.0        10.0            10.8     -0.0109
23092     15.0    120.0        10.0            10.9     -0.0218
23093     15.0    120.0        10.0            11.5     -0.0434
```

```
                                                    sequence
0      author: Thompson geometry: tube pressure [MPa]...
1      author: Thompson geometry: tube pressure [MPa]...
2      author: Thompson geometry: nan pressure [MPa]:...
3      author: Beus geometry: annulus pressure [MPa]:...
4      author: nan geometry: nan pressure [MPa]: 17.2...
...                                                  ...
23089  author: Richenderfer geometry: plate pressure ...
23090  author: Richenderfer geometry: plate pressure ...
23091  author: Richenderfer geometry: plate pressure ...
23092  author: Richenderfer geometry: plate pressure ...
23093  author: Richenderfer geometry: plate pressure ...

[23094 rows x 11 columns]>
```

Train/Optimize

BERT Masked LM in PyTorch

[3]:
```
# Prepare data as Ax = B
# A
sequences = data["sequence"]
# B
x_e_out = data['x_e_out [-]']

tokenized_data = sequences.apply(token_encoder_fun)
tokenized_data = tokenized_data.reset_index(drop=True)

input_ids = torch.tensor(tokenized_data)

attention_masks = torch.empty( ( len(input_ids), MAX_LENGTH ) )

# Generate attention masks
for i in trange(len(input_ids)):
    tokens = input_ids[i, :]
    row_mask = [int(token_id.item() > 0) for token_id in tokens]
    row_mask = torch.tensor(row_mask).unsqueeze(0)
    attention_masks[i] = row_mask
```

[6]:
```
x_e_out = data['x_e_out [-]'].astype(float)
x_e_out = torch.tensor(x_e_out, dtype=torch.float32).reshape(-1,1)

# Combine the training inputs into a TensorDataset.
dataset = TensorDataset(input_ids, attention_masks, x_e_out)

# Create a 90-10 train-validation split.
# Calculate the number of samples to include in each set.
train_size = int(0.9 * len(dataset))
test_size = len(dataset) - train_size

# Divide the dataset by randomly selecting samples.
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])
```

```
train_dataloader = DataLoader(
            train_dataset,  # The training samples.
            sampler = RandomSampler(train_dataset), # Select batches randomly
            batch_size = batch_size # Trains with this batch size.
        )

# For validation the order doesn't matter, so we'll just read them sequentially.
test_dataloader = DataLoader(
            test_dataset, # The validation samples.
            sampler = SequentialSampler(test_dataset), # Pull out batches␣
 ↪sequentially.
            batch_size = batch_size # Evaluate with this batch size.
        )
```

[3]:
```
model = BertHeatFlux().to(device)
model.train()
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when
initializing BertModel: ['cls.predictions.transform.dense.weight',
'cls.predictions.transform.dense.bias', 'cls.predictions.bias',
'cls.seq_relationship.bias', 'cls.predictions.decoder.weight',
'cls.predictions.transform.LayerNorm.bias',
'cls.predictions.transform.LayerNorm.weight', 'cls.seq_relationship.weight']
- This IS expected if you are initializing BertModel from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of
a model that you expect to be exactly identical (initializing a
BertForSequenceClassification model from a BertForSequenceClassification model).
```

[3]:
```
BertHeatFlux(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.3, inplace=False)
  (linear): Linear(in_features=768, out_features=1, bias=True)
)
```

[13]:
```
# Step 3: Define the optimizer and loss fn
optimizer = torch.optim.AdamW(model.parameters(), lr=1e-5)
loss_fn = torch.nn.MSELoss()

# Step 4: Train the model
num_epochs = 5

for epoch in trange(num_epochs):
    for batch in tqdm(train_dataloader):
        input_ids, attention_mask, labels = [x.to(device) for x in batch]

        # Reset gradients
        optimizer.zero_grad()

        # Forward pass
        predictions = model(input_ids,
                            attention_mask)
        loss = loss_fn(predictions, labels)

        # Backward pass
        loss.backward()

        #torch.nn.utils.clip_grad

        # Update the weights
        optimizer.step()

    # Print the loss and accuracy of the model for this epoch
    print(f"Epoch {epoch+1}, Loss: {loss:.3f}")

torch.save(model, 'bert_fine-tuned-1.sav')
```

```
  0%|          | 0/5 [00:00<?, ?it/s]

  0%|          | 0/650 [00:00<?, ?it/s]

Epoch 1, Loss: 0.014

  0%|          | 0/650 [00:00<?, ?it/s]

Epoch 2, Loss: 0.007

  0%|          | 0/650 [00:00<?, ?it/s]

Epoch 3, Loss: 0.007

  0%|          | 0/650 [00:00<?, ?it/s]

Epoch 4, Loss: 0.003

  0%|          | 0/650 [00:00<?, ?it/s]

Epoch 5, Loss: 0.006
```

BertHeatFlux Model Names

- bert_fine-tuned_1.sav
  - epochs=5, training: data.csv + Data_CHF
  - loss: 0.014, 0.007, 0.007, 0.003, 0.006 ***

[32]:
```
# Evaluate the model on the testing dataset
with torch.no_grad():
  mean_squared_error = MeanSquaredError(squared=False).to(device)
  RMSE = 0
  for batch in test_dataloader:
    input_ids, attention_mask, target = [x.to(device) for x in batch]

    # Get the logits for the masked tokens
    #outputs = model(input_ids, attention_mask=attention_mask)
    preds = model(input_ids,
                  attention_mask)

    RMSE += mean_squared_error(preds, target)
  RMSE = RMSE.cpu().detach().numpy()
# Print the accuracy of the model
print("MSE: {:.3f}".format(RMSE))
```

```
  0%|          | 0/73 [00:00<?, ?it/s]

MSE: 6.045
```

Use Trained Model for Prediction on Competition Data

Uses `data_test`

Load, predict, output to file.txt

[49]:
```
# Load the numerical data you want to train BERT on
df1 = pd.read_csv("data.csv")
#df2 = pd.read_csv('Data_CHF_Zhao_2020_ATE.csv')
frames = [df1]
df = pd.concat(frames)
```

```
# Define the name of the column that you want to move to the end of the DataFrame
column_name = "x_e_out [-]"

# Select the column and drop it from the DataFrame
column_to_move = df[column_name]
col = df.drop(column_name, axis=1, inplace=True)

# Append the column back to the end of the DataFrame
df[column_name] = column_to_move

# select only rows where x_e out DOES NOT exist
data_test = df[df["x_e_out [-]"].isna()]

# start with a small data set for speed
#######################################
data_test = data_test[10000:]
#######################################
data_test = data_test.reset_index(drop=True)

# Convert numerical values to string format to match BERT input requirement
data_test = data_test.astype(str)

data_test["sequence"] = ""

# Concatenate all the values in a row into a single string using the column names
# Iterate through rows and columns
for index, row in data_test.iterrows():
    string = ""
    for column in data_test.columns:
        if column == "x_e_out [-]":
            continue
        if column == "sequence" or column == 'id':
            continue
        string += column + ": " + str(row[column]) + " "
    masked_string = string.strip()
    data_test["sequence"][index] = masked_string

sequences = data_test["sequence"]
x_e_out = data_test['x_e_out [-]']
x_e_out = data_test['x_e_out [-]'].astype(float)
x_e_out = torch.tensor(x_e_out, dtype=torch.float32).reshape(-1,1)

tokenized_data = sequences.apply(token_encoder_fun)
tokenized_data = tokenized_data.reset_index(drop=True)

input_ids = torch.tensor(tokenized_data)

attention_masks = torch.empty( ( len(input_ids), MAX_LENGTH ) )

# Generate attention masks
for i in range(len(input_ids)):
    tokens = input_ids[i, :]
    row_mask = [int(token_id.item() > 0) for token_id in tokens]
    row_mask = torch.tensor(row_mask).unsqueeze(0)
    attention_masks[i] = row_mask

eval_dataset = TensorDataset(input_ids, attention_masks) #x_e_out is nan so no point

eval_dataloader = DataLoader(
    eval_dataset,
    sampler=SequentialSampler(eval_dataset)
)

device = torch.device('cpu')
model = torch.load('bert_fine-tuned-1.sav').to(device)
model.eval()

predictions = torch.empty( ( len(input_ids), 1 ) ).to(device)
i = 0
for batch in tqdm(eval_dataloader):
    input_ids, attention_mask = [x.to(device) for x in batch]

    # Forward pass
    pred = model(input_ids.cpu(),
                attention_mask)

    # Print the loss and accuracy of the model for this epoch
    predictions[i] = pred
    i += 1

#np.savetxt('batch11.txt', predictions.cpu().detach().numpy())
```

```
  0%|          | 0/415 [00:00<?, ?it/s]
```

Combine output files into single file

[35]:
```
""" filenames = ['batch1.txt', 'batch2.txt', 'batch3.txt', 'batch4.txt','batch5.
↪txt','batch6.txt','batch7.txt',
'batch8.txt','batch9.txt','batch10.txt','batch11.txt']
with open('combined.txt', 'w') as outfile:
    for fname in filenames:
        with open(fname) as infile:
            outfile.write(infile.read()) """
```

[40]:
```
""" #data_test = df[df["x_e_out [-]"].isna()]
np.savetxt('ids.txt', data_test['id']) """
```

Combine two text files into a single csv file

[3]:
```
""" import csv

# read in data from file1.txt
with open('ids.txt', 'r') as file1:
    data1 = file1.read().splitlines()

# read in data from file2.txt
with open('combined.txt', 'r') as file2:
    data2 = file2.read().splitlines()

# combine the two lists
data = data1 + data2

# write combined data to output.csv
with open('output.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(['ID', 'x_e_out'])  # write column headers
    for i in range(len(data)):
        if i < len(data1):
            writer.writerow([data1[i], data2[i]]) """
```

Appendix

Console/Testing

```
[5]: from bertviz import model_view
```

```
[50]: from torchviz import make_dot
#device = torch.device('cpu')
#model = torch.load('bert_fine-tuned-1.sav').to(device)

make_dot(predictions.mean(), params=dict(model.named_parameters()), show_attrs=True)
```

```
[46]: """ device = torch.device('cpu')
model = torch.load('bert_fine-tuned-1.sav').to(device)
model.eval()

inp = eval_dataset[-1]
input_ids, attention_mask = [x.view(1,128) for x in inp]

    # Forward pass
outputs = model(input_ids, attention_mask)

attention = outputs[-1]  # Retrieve attention from model outputs
input_ids = input_ids.tolist()
input_ids = [i for i in input_ids[0]]
tokens = tokenizer.convert_ids_to_tokens(input_ids)  # Convert input ids to token␣
 ↪strings
model_view(attention, tokens)  # Display model view
 """
```

```
[ ]: BertHeatFlux(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.3, inplace=False)
  (linear): Linear(in_features=768, out_features=1, bias=True)
)
```

```
[47]: model.parameters
```

```
[47]: <bound method Module.parameters of BertHeatFlux(
        (bert): BertModel(
          (embeddings): BertEmbeddings(
            (word_embeddings): Embedding(30522, 768, padding_idx=0)
            (position_embeddings): Embedding(512, 768)
            (token_type_embeddings): Embedding(2, 768)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (encoder): BertEncoder(
            (layer): ModuleList(
              (0-11): 12 x BertLayer(
                (attention): BertAttention(
                  (self): BertSelfAttention(
                    (query): Linear(in_features=768, out_features=768, bias=True)
                    (key): Linear(in_features=768, out_features=768, bias=True)
                    (value): Linear(in_features=768, out_features=768, bias=True)
                    (dropout): Dropout(p=0.1, inplace=False)
                  )
                  (output): BertSelfOutput(
                    (dense): Linear(in_features=768, out_features=768, bias=True)
                    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                    (dropout): Dropout(p=0.1, inplace=False)
                  )
                )
                (intermediate): BertIntermediate(
                  (dense): Linear(in_features=768, out_features=3072, bias=True)
                  (intermediate_act_fn): GELUActivation()
                )
                (output): BertOutput(
                  (dense): Linear(in_features=3072, out_features=768, bias=True)
                  (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                  (dropout): Dropout(p=0.1, inplace=False)
                )
              )
            )
          )
          (pooler): BertPooler(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (activation): Tanh()
          )
        )
        (dropout): Dropout(p=0.3, inplace=False)
        (linear): Linear(in_features=768, out_features=1, bias=True)
      )>
```

A.2 LSTM Model

[1]:
```
import pandas as pd
from collections import defaultdict
import numpy as np
import matplotlib.pyplot as plt

import gc
import warnings
import datetime as dt
import math
import seaborn as sn

import tensorflow as tf # import tensorflow as a whole
from tensorflow import keras # only import Keras

# import the dense and input layer
from keras.layers import Dense, Input
```

```
# the sequential model object
from keras.models import Sequential

from numpy import linalg as LA
import math

from tensorflow.keras.callbacks import EarlyStopping

from sklearn.model_selection import train_test_split

import tensorflow_addons as tfa

from keras.models import Sequential
from keras.layers import Input, CuDNNLSTM, Dense
from keras.layers import Embedding
from tensorflow.keras.layers import LSTM
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint



# load a saved model
from keras.models import load_model
```

[2]:
```
#Import Data
data = pd.read_csv("data.csv")
samp_sol = pd.read_csv("sample_submission.csv")
```

[3]:
```
data_set = pd.read_csv("data.csv")
display(data_set.head())
```

```
   id    author geometry  pressure [MPa]  mass_flux [kg/m2-s]  x_e_out [-]  \
0   0  Thompson     tube            7.00               3770.0       0.1754
1   1  Thompson     tube             NaN               6049.0      -0.0416
2   2  Thompson      NaN           13.79               2034.0       0.0335
3   3      Beus  annulus           13.79               3679.0      -0.0279
4   4       NaN     tube           13.79                686.0          NaN

   D_e [mm]  D_h [mm]  length [mm]  chf_exp [MW/m2]
0       NaN      10.8        432.0              3.6
1      10.3      10.3        762.0              6.2
2       7.7       7.7        457.0              2.5
3       5.6      15.2       2134.0              3.0
4      11.1      11.1        457.0              2.8
```

[4]:
```
for k in data_set.columns:
    print("Column:", k)
    print(data_set[k].value_counts())
    print()
```

```
Column: id
0        1
21106    1
21104    1
```

```
21103    1
21102    1
        ..
10543    1
10542    1
10541    1
10540    1
31643    1
Name: id, Length: 31644, dtype: int64

Column: author
Thompson        17396
Janssen          2716
Weatherhead      2040
Beus             1604
Peskov           1084
Williams          891
Richenderfer      545
Mortimore         197
Kossolapov        101
Inasaka            46
Name: author, dtype: int64

Column: geometry
tube       21145
annulus     4381
plate        618
Name: geometry, dtype: int64

Column: pressure [MPa]
13.79    9226
6.89     4701
15.51    1117
10.34    1064
11.03     872
         ...
3.77        1
12.21       1
7.06        1
7.70        1
12.14       1
Name: pressure [MPa], Length: 144, dtype: int64

Column: mass_flux [kg/m2-s]
4069.0    963
1519.0    634
1356.0    615
2034.0    533
1000.0    418
         ...
835.0       1
967.0       1
6995.0      1
2050.0      1
```

```
1271.0      1
Name: mass_flux [kg/m2-s], Length: 733, dtype: int64

Column: x_e_out [-]
 0.0334    177
 0.0406    151
 0.0196    133
-0.0434    127
 0.0886    109
          ...
-0.0524      1
 0.0743      1
-0.4854      1
 0.0106      1
-0.6500      1
Name: x_e_out [-], Length: 1682, dtype: int64

Column: D_e [mm]
10.3    2684
10.8    2512
1.9     2499
4.7     2432
7.7     2294
5.6     2108
12.7    1433
7.8     1283
10.0    1080
9.5      881
11.1     765
15.0     663
4.6      658
23.6     647
8.5      614
11.5     602
3.0      483
6.4      420
5.7      400
12.8     355
3.6      281
37.5     262
5.0      189
1.0      153
9.3      147
22.2      99
12.4      80
8.0       32
19.8      31
1.1       18
23.5      11
1.7        8
10.1       8
10.5       6
10.4       5
13.3       4
```

```
24.6       2
22.6       2
15.2       1
10.7       1
23.7       1
11.3       1
15.9       1
Name: D_e [mm], dtype: int64

Column: D_h [mm]
10.3     2792
10.8     2598
1.9      2569
4.7      2554
7.7      2382
15.2     1578
7.8      1313
10.0     1115
9.5       904
42.3      864
11.1      790
120.0     672
23.6      662
11.5      637
38.1      611
4.6       604
5.6       570
3.0       492
15.9      446
5.7       400
22.3      386
12.8      368
3.6       285
37.5      279
24.6      240
13.3      204
1.0       165
9.3       162
96.3       99
12.4       92
11.3       73
8.0        35
19.8       31
1.1        17
23.5       13
1.7         9
10.5        9
10.4        6
15.0        6
10.1        6
22.2        4
22.6        4
12.7        2
6.4         2
```

```
40.0        1
33.3        1
10.6        1
25.6        1
14.8        1
Name: D_h [mm], dtype: int64

Column: length [mm]
457.0     3180
762.0     2364
318.0     2154
2134.0    1833
152.0     1800
          ...
1250.0       1
524.0        1
96.0         1
130.0        1
2650.0       1
Name: length [mm], Length: 70, dtype: int64

Column: chf_exp [MW/m2]
2.3     1260
2.5     1144
2.2     1134
3.6      986
2.1      980
        ...
10.4      14
9.1       14
10.2      14
12.1      13
15.6       7
Name: chf_exp [MW/m2], Length: 109, dtype: int64
```

[5]:
```
max(np.array(data_set['x_e_out [-]']))
```

[5]:
```
0.232
```

[6]:
```
more_data = pd.read_csv("Data_CHF_Zhao_2020_ATE.csv")
```

[7]:
```
for k in more_data.columns:
    print("Column:", k)
    print(more_data[k].value_counts())
    print()
```

```
Column: id
1       1
1240    1
1252    1
1251    1
1250    1
       ..
```

```
619     1
618     1
617     1
616     1
1865    1
Name: id, Length: 1865, dtype: int64

Column: author
Thompson        1202
Janssen          282
Weatherhead      162
Beus              77
Williams          51
Richenderfer      36
Mortimore         19
Peskov            17
Kossolapov        12
Inasaka            7
Name: author, dtype: int64

Column: geometry
tube       1439
annulus     378
plate        48
Name: geometry, dtype: int64

Column: pressure [MPa]
13.79    610
6.89     420
11.03     59
10.34     58
15.51     52
        ...
10.57      1
10.93      1
12.34      1
12.38      1
10.24      1
Name: pressure [MPa], Length: 114, dtype: int64

Column: mass_flux [kg/m2-s]
4069    76
1519    55
2034    42
2292    28
1533    27
        ..
903      1
852      1
783      1
4164     1
7093     1
Name: mass_flux [kg/m2-s], Length: 578, dtype: int64
```

```
Column: x_e_out [-]
 0.0886    7
 0.0145    7
 0.0990    6
-0.0187    6
-0.0071    5
          ..
 0.0257    1
 0.0140    1
 0.0363    1
 0.1789    1
-0.1041    1
Name: x_e_out [-], Length: 1360, dtype: int64

Column: D_e [mm]
7.7     188
10.3    179
1.9     150
12.7    142
4.7     137
5.6     123
10.8     92
4.6      91
8.5      88
23.6     72
7.8      70
11.5     68
11.1     68
9.5      51
15.0     48
37.5     40
9.3      31
12.8     29
5.7      28
6.4      25
3.0      20
3.6      19
5.0      19
10.0     17
1.0      17
22.2     15
12.4     12
19.8      7
8.0       6
23.5      3
1.1       3
1.7       3
23.7      1
10.4      1
10.5      1
22.6      1
Name: D_e [mm], dtype: int64

Column: D_h [mm]
```

```
7.7      188
10.3     179
1.9      150
4.7      137
10.8      92
42.3      91
4.6       78
15.2      77
23.6      72
7.8       70
11.1      68
11.5      68
38.1      51
9.5       51
22.3      50
120.0     48
5.6       46
37.5      40
24.6      38
9.3       31
12.8      29
5.7       28
15.9      25
3.0       20
13.3      19
3.6       19
1.0       17
10.0      17
96.3      14
11.3      13
12.4      12
19.8       7
8.0        6
23.5       3
1.1        3
1.7        3
23.7       1
22.2       1
22.6       1
10.4       1
10.5       1
Name: D_h [mm], dtype: int64

Column: length [mm]
457     258
762     154
1778    126
152     105
318     103
2134     96
591      70
914      68
610      63
432      61
```

```
1836     51
2743     50
10       48
696      45
295      41
1953     40
864      40
794      39
1972     38
229      37
305      36
1930     29
1524     29
625      28
216      25
737      25
1727     21
1829     20
76       19
1143     16
150      10
2591      8
100       7
25        7
51        5
1359      5
38        5
1650      5
2007      4
2711      4
565       3
114       3
35        3
43        2
890       2
660       2
841       2
1000      1
400       1
520       1
893       1
64        1
3048      1
1300      1
Name: length [mm], dtype: int64

Column: chf_exp [MW/m2]
2.3     76
3.2     66
2.1     64
2.5     64
2.2     56
        ..
13.0     1
```

```
8.5       1
7.4       1
13.3      1
11.5      1
Name: chf_exp [MW/m2], Length: 109, dtype: int64
```

[8]:
```
total_data = data_set.append(more_data).drop("id", axis=1)
```

```
/var/folders/tr/3zw9td7531z0_hhwc8105rv00000gn/T/ipykernel_11003/3739133671.py:1
: FutureWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  total_data = data_set.append(more_data).drop("id", axis=1)
```

[9]:
```
dummies_author = pd.get_dummies(total_data.author)
merged_with_authors = pd.concat([total_data, dummies_author], axis='columns')
merged_with_authors = merged_with_authors.drop(['author'], axis='columns')

dummies_geometry = pd.get_dummies(merged_with_authors.geometry)
merged_with_authors_and_geometry = pd.concat([merged_with_authors,␣
 ↪dummies_geometry], axis='columns')
finalized_data = merged_with_authors_and_geometry.drop(['geometry'], axis='columns')
```

[10]:
```
final_test = finalized_data[pd.isnull(finalized_data["x_e_out [-]"])]
final_test = final_test.fillna(0)
final_test = final_test.drop("x_e_out [-]", axis=1)
finalized_data = finalized_data.dropna(subset=["x_e_out [-]"])
finalized_data = finalized_data.fillna(0)
```

[11]:
```
submission_indices = list(final_test.index)
```

Train / Test Split

[12]:
```
X = finalized_data.drop("x_e_out [-]", axis=1)
y = finalized_data["x_e_out [-]"]
```

[13]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10,␣
 ↪random_state=42)
```

[14]:
```
def convert_to_sequences(df):

    m,n = df.shape

    sequences = np.zeros((m, 2*n))

    i = 0
    for index, row in df.iterrows():

        k = 1
        j = 0
        for item in row:
            if item != 0:
                sequences[i, j] = k
                sequences[i, j + 1] = item
                j += 2
            k += 1

        i+= 1

    return sequences
```

[15]:
```
X_train = convert_to_sequences(X_train)
X_test = convert_to_sequences(X_test)
```

[16]:
```
X_train = np.expand_dims(X_train, axis=1)
X_test = np.expand_dims(X_test, axis=1)

y_train = np.array(y_train)
y_test = np.array(y_test)
```

[17]:
```
X_train.shape
```

[17]:
```
(20784, 1, 38)
```

Construct Model

[18]:
```
model = Sequential()
model.add(LSTM(64, input_shape=(1,2*19)))
model.add(Dense(1, activation='tanh'))
```

```
2023-05-29 10:45:38.417990: I tensorflow/core/platform/cpu_feature_guard.cc:193]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  SSE4.1 SSE4.2
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

[19]:
```
model.compile(loss=['mean_squared_error'] , optimizer='adam', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 64)                26368

 dense (Dense)               (None, 1)                 65

=================================================================
Total params: 26,433
Trainable params: 26,433
Non-trainable params: 0
_________________________________________________________________
```

[20]:
```
#model = load_model('best_model.h5')
```

```
[37]: #es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=200)
      mc = ModelCheckpoint('best_model.h5', monitor='val_loss', mode='min', verbose=1,␣
       ↪save_best_only=True)
```

```
[46]: model.fit(X_train, y_train, epochs=8000, batch_size=20, verbose=2,␣
       ↪validation_split=0.1, callbacks=[mc])
```

Perform Prediction

```
[50]: X_submission = convert_to_sequences(final_test)
      X_submission = np.expand_dims(X_submission, axis=1)
```

```
[51]: y_submission = model.predict(X_submission)
```

```
326/326 [==============================] - 0s 403us/step
```

```
[52]: y_submission = list(y_submission.reshape(len(y_submission)))
```

```
[53]: df = pd.DataFrame(data={"id": submission_indices, "x_e_out [-]": y_submission})
      df.to_csv("submission.csv", sep=',',index=False)
```

A.3 CRF Model

```
[4]: %pip install sklearn-crfsuite
```

```
Collecting sklearn-crfsuite
  Downloading sklearn_crfsuite-0.3.6-py2.py3-none-any.whl (12 kB)
Requirement already satisfied: tabulate in c:\users\sam\anaconda3\lib\site-
packages (from sklearn-crfsuite) (0.8.10)
Requirement already satisfied: six in c:\users\sam\anaconda3\lib\site-packages
(from sklearn-crfsuite) (1.16.0)
Collecting python-crfsuite>=0.8.3
  Downloading python_crfsuite-0.9.9-cp310-cp310-win_amd64.whl (139 kB)
     -------------------------------------- 139.4/139.4 kB 2.1 MB/s eta 0:00:00
Requirement already satisfied: tqdm>=2.0 in c:\users\sam\anaconda3\lib\site-
packages (from sklearn-crfsuite) (4.64.1)
Requirement already satisfied: colorama in c:\users\sam\anaconda3\lib\site-
packages (from tqdm>=2.0->sklearn-crfsuite) (0.4.6)
Installing collected packages: python-crfsuite, sklearn-crfsuite
Successfully installed python-crfsuite-0.9.9 sklearn-crfsuite-0.3.6
Note: you may need to restart the kernel to use updated packages.
```

```
[3]: # Importing the necessary libraries
     import pandas as pd
     import numpy as np
     import re

     # Load the numerical data you want to train BERT on
     data = pd.read_csv('data.csv')
```

```
# Rename column names to simplify accessing
colnames = [re.sub("\s+\[.*\]", "", s) for s in data.columns]
data.columns = colnames

# Convert numerical values to string format to match BERT input requirement
data = data.astype(str)
```

[4]:
```
from sklearnex import patch_sklearn
patch_sklearn()
import sklearn_crfsuite

orig_data = data
data = data.loc[data['x_e_out'] != 'nan']
data = data.reset_index().drop('index', axis=1)
indx = np.random.choice(range(len(data)), 5000, False)
data = data.iloc[indx]
data_dicts = data.loc[:, data.columns != 'x_e_out'].to_dict('records')

# Convert inputs into dictionary form
clean_dicts = []
for dict in data_dicts:
    dict = {k: dict[k] for k in dict if not dict[k] == 'nan'}
    clean_dicts.append(dict)
clean_dicts = [clean_dicts]

y_train = [data['x_e_out'].to_list()]

# Train CRF model
crf = sklearn_crfsuite.CRF(
    algorithm='lbfgs',
    c1=0.1,
    c2=0.1,
    max_iterations=100,
    all_possible_transitions=True
)

try:
    crf.fit(clean_dicts, y_train)
except AttributeError:
    pass
```

```
Intel(R) Extension for Scikit-learn* enabled (https://github.com/intel/scikit-
learn-intelex)
```

[26]:
```
len(clean_dicts[0])
```

[26]:
```
1000
```

[10]:
```
import statistics as stat
stat.mode(x_e_out)
```

[10]:
```
'0.0334'
```

[42]:
```
orig_data.loc[0:100,'x_e_out'].to_list()
```

```
[42]: ['0.1754',
        '-0.0416',
        '0.0335',
        '-0.0279',
        '-0.0711',
        ...
        '0.0051',
        '-0.1258',
        '-0.0373',
        '-0.1209',
        '0.0886']
```

```
[3]: preds = crf.predict([orig_data.loc[:, orig_data.columns != 'x_e_out'].
      ↪to_dict('records')[0:1000]])
     preds[0]
```

```
[3]: ['0.0196',
       '-0.0071',
       '-0.0202',
       '-0.0279',
       '0.0384',
       ...
       '-0.0483',
       '-0.0103',
       '0.0384']
```

```
[5]: len(set(preds[0]).intersection(orig_data.loc[0:1000,'x_e_out'].to_list()))
```

```
[5]: 233
```

```
[4]: orig_data.loc[0:1000,'x_e_out']
```

```
[4]: 0        0.1754
     1       -0.0416
     2        0.0335
     3       -0.0279
     4           nan
              ...
     996     -0.0929
     997     -0.0483
     998         nan
     999         nan
     1000     0.0412
     Name: x_e_out, Length: 1001, dtype: object
```

```
[7]: nan_preds
```

```
[7]: [['0.0196',
        '-0.0071',
        '-0.0202',
        '-0.0279',
        '0.0384',
        ...
```

```
  '-0.0483',
  '-0.0103',
  '0.0384',
  ...]]
```

[5]:
```
nan_data = orig_data.loc[orig_data['x_e_out'] == 'nan']
nan_preds = crf.predict([orig_data.loc[:, orig_data.columns != 'x_e_out'].
 ↪to_dict('records')])
```

[60]:
```
indx = orig_data.index[orig_data['x_e_out'] == 'nan'].tolist()
nan_data.loc[:,'x_e_out'] = list(map(nan_preds[0].__getitem__, indx))
nan_data.loc[:,['id','x_e_out']].rename(columns={'x_e_out':'x_e_out [-]'}).to_csv('./
 ↪imp_data.csv', index=False)
```

```
C:\Users\Sam\AppData\Local\Temp\ipykernel_49608\2798700851.py:4:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  nan_data.loc[:,'x_e_out'] = list(map(nan_preds[0].__getitem__, indx))
```

[31]:
```
act
```

[31]:
```
['0.0396',
 '0.1434',
 '-0.0431',
 '-0.0251',
 '0.0929',
 '-0.0298',
 '0.0648',
 ...
 '0.1108',
 '0.1227',
 '0.0229',
 '-0.0158',
 '0.0491']
```

[33]:
```
indx = orig_data.index[orig_data['x_e_out'] != 'nan'].tolist()
indx = np.random.choice(indx, 1000, False)
exp_str = list(map(nan_preds[0].__getitem__, indx))
exp = [float(i) for i in exp_str]
act_str = orig_data.loc[indx, 'x_e_out'].to_list()
act = [float(i) for i in act_str]
rmse = np.sqrt(np.mean(np.square(np.subtract(exp, act))))
rmse
```

A.4 Linear Regression Model

```
[ ]: #Import Libraries
     import pandas as pd
     import numpy as np
     import math
     from sklearn.impute import KNNImputer
     from sklearn.experimental import enable_iterative_imputer
     from sklearn.impute import IterativeImputer
     from sklearn.linear_model import LinearRegression
     from sklearn.preprocessing import PolynomialFeatures
     from sklearn.pipeline import make_pipeline
```

```
[ ]: #Import Data
     data = pd.read_csv("data.csv")
     samp_sol = pd.read_csv("sample_submission.csv")
     #print(data)
```

```
[ ]: #Create imputation function
     def impute_values(dataframe):
         # Split the dataframe into complete and missing subsets
         complete_data = dataframe.dropna(subset=['x_e_out [-]'])
         complete_data.to_csv('complete_data.csv')
         missing_data = dataframe[dataframe['x_e_out [-]'].isnull()]
         missing_data.to_csv('missing_data.csv')

         # Create the regression model
         reg_model = LinearRegression()

         # Fit the regression model
         X = complete_data[['pressure [MPa]', 'mass_flux [kg/m2-s]', 'D_e [mm]', 'D_h␣
      ↪[mm]', 'length [mm]', 'chf_exp [MW/m2]']]
         y = complete_data['x_e_out [-]']
         reg_model.fit( X , y)

         #provide R**2 score and equation
         print('R**2:' , reg_model.score(X, y))

         # Predict the missing values
         predicted_values = reg_model.predict(missing_data[['pressure [MPa]', 'mass_flux␣
      ↪[kg/m2-s]', 'D_e [mm]', 'D_h [mm]', 'length [mm]', 'chf_exp [MW/m2]']])

         # Replace missing values with predicted values
         dataframe.loc[dataframe['x_e_out [-]'].isnull(), 'x_e_out [-]'] =␣
      ↪predicted_values

         # Get the regression equation
         intercept = reg_model.intercept_
         #print('Intercept:' , intercept)
         coeff = reg_model.coef_
         print('regression equation:', intercept , '+' , coeff[0] , 'P +' , coeff[1] , 'j␣
      ↪+' , coeff[2] , 'D_e +' , coeff[3] , 'D_h +' , coeff[4] , 'L +' , coeff[5] ,␣
      ↪'CHF')
```

```
    #Solve for RMSE
    N = len(y)
    y_hat = intercept + \
        (coeff[0] * complete_data['pressure [MPa]']) + \
        (coeff[1] * complete_data['mass_flux [kg/m2-s]']) + \
        (coeff[2] * complete_data['D_e [mm]']) + \
        (coeff[3] * complete_data['D_h [mm]']) + \
        (coeff[4] * complete_data['length [mm]']) + \
        (coeff[5] * complete_data['chf_exp [MW/m2]'])

    RMSE = (math.sqrt((1/N) * (sum((y - y_hat)**2))))
    print('RMSE:' , RMSE)




    # Return the updated dataframe and regression equation
    return dataframe , RMSE
```

```
[ ]: #Max
#Fill all other columns with max of the column
data_max = data.copy()
data_max['pressure [MPa]'] = data_max['pressure [MPa]'].fillna(data_max['pressure␣
 ↪[MPa]'].max())
data_max['mass_flux [kg/m2-s]'] = data_max['mass_flux [kg/m2-s]'].
 ↪fillna(data_max['mass_flux [kg/m2-s]'].max())
data_max['D_e [mm]'] = data_max['D_e [mm]'].fillna(data_max['D_e [mm]'].max())
data_max['D_h [mm]'] = data_max['D_h [mm]'].fillna(data_max['D_h [mm]'].max())
data_max['length [mm]'] = data_max['length [mm]'].fillna(data_max['length [mm]'].
 ↪max())
data_max['chf_exp [MW/m2]'] = data_max['chf_exp [MW/m2]'].fillna(data_max['chf_exp␣
 ↪[MW/m2]'].max())
#print(data_max)
data_max.to_csv('data_max.csv')




#Solve for X_e_out
data_max = impute_values(data_max)
#print(data_max)

#Put solution in new df
max_sol = samp_sol.copy()
filtered_data_max = pd.merge(max_sol['id'], data_max, on='id', how='left')
filtered_data_max = filtered_data_max.dropna(subset=['x_e_out [-]'])
#print(filtered_data_max)
max_sol['x_e_out [-]'] = filtered_data_max['x_e_out [-]']
#print(max_sol)
max_sol.to_csv('max_sol.csv')
```

[ ]:
```
#Min
#Fill Columns with min
data_min = data.copy()
data_min['pressure [MPa]'] = data_min['pressure [MPa]'].fillna(data_min['pressure␣
↪[MPa]'].min())
data_min['mass_flux [kg/m2-s]'] = data_min['mass_flux [kg/m2-s]'].
↪fillna(data_min['mass_flux [kg/m2-s]'].min())
data_min['D_e [mm]'] = data_min['D_e [mm]'].fillna(data_min['D_e [mm]'].min())
data_min['D_h [mm]'] = data_min['D_h [mm]'].fillna(data_min['D_h [mm]'].min())
data_min['length [mm]'] = data_min['length [mm]'].fillna(data_min['length [mm]'].
↪min())
data_min['chf_exp [MW/m2]'] = data_min['chf_exp [MW/m2]'].fillna(data_min['chf_exp␣
↪[MW/m2]'].min())
#print(data_min)



#Solve for X_e_out
data_min = impute_values(data_min)
#print(data_min)

#put solutions in new df
min_sol = samp_sol.copy()
filtered_data_min = pd.merge(min_sol['id'], data_min, on='id', how='left')
filtered_data_min = filtered_data_min.dropna(subset=['x_e_out [-]'])
#print(filtered_data_min)
min_sol['x_e_out [-]'] = filtered_data_min['x_e_out [-]']
#print(min_sol)
min_sol.to_csv('min_sol.csv')
```

[ ]:
```
#Mean
#Find Means of input variables
data_mean = data.copy()
data_mean['pressure [MPa]'] = data_mean['pressure [MPa]'].fillna(data_mean['pressure␣
↪[MPa]'].mean())
data_mean['mass_flux [kg/m2-s]'] = data_mean['mass_flux [kg/m2-s]'].
↪fillna(data_mean['mass_flux [kg/m2-s]'].mean())
data_mean['D_e [mm]'] = data_mean['D_e [mm]'].fillna(data_mean['D_e [mm]'].mean())
data_mean['D_h [mm]'] = data_mean['D_h [mm]'].fillna(data_mean['D_h [mm]'].mean())
data_mean['length [mm]'] = data_mean['length [mm]'].fillna(data_mean['length [mm]'].
↪mean())
data_mean['chf_exp [MW/m2]'] = data_mean['chf_exp [MW/m2]'].
↪fillna(data_mean['chf_exp [MW/m2]'].mean())


#Solve for x_e_out
data_mean = impute_values(data_mean)

#Add solutions to new df
mean_sol = samp_sol.copy()
filtered_data_mean = pd.merge(mean_sol['id'], data_mean, on='id', how='left')
filtered_data_mean = filtered_data_mean.dropna(subset=['x_e_out [-]'])
#print(filtered_data_mean)
```

```
mean_sol['x_e_out [-]'] = filtered_data_mean['x_e_out [-]']
#print(mean_sol)
mean_sol.to_csv('mean_sol.csv')
```

[ ]:
```
#Median
#Find means of input variables
data_median = data.copy()
data_median['pressure [MPa]'] = data_median['pressure [MPa]'].
 ↪fillna(data_median['pressure [MPa]'].median())
data_median['mass_flux [kg/m2-s]'] = data_median['mass_flux [kg/m2-s]'].
 ↪fillna(data_median['mass_flux [kg/m2-s]'].median())
data_median['D_e [mm]'] = data_median['D_e [mm]'].fillna(data_median['D_e [mm]'].
 ↪median())
data_median['D_h [mm]'] = data_median['D_h [mm]'].fillna(data_median['D_h [mm]'].
 ↪median())
data_median['length [mm]'] = data_median['length [mm]'].fillna(data_median['length␣
 ↪[mm]'].median())
data_median['chf_exp [MW/m2]'] = data_median['chf_exp [MW/m2]'].
 ↪fillna(data_median['chf_exp [MW/m2]'].median())



#Solve for x_e_out
data_median = impute_values(data_median)

#add solutions to new df
median_sol = samp_sol.copy()
filtered_data_median = pd.merge(median_sol['id'], data_median, on='id')
filtered_data_median = filtered_data_median.dropna(subset=['x_e_out [-]'])
#print(filtered_data_median)
median_sol['x_e_out [-]'] = filtered_data_median['x_e_out [-]']
#print(median_sol)
median_sol.to_csv('median_sol.csv')
```

[ ]:
```
#Mode
data_mode = data.copy()
data_mode['pressure [MPa]'] = data_mode['pressure [MPa]'].fillna(data_mode['pressure␣
 ↪[MPa]'].mode()[0])
data_mode['mass_flux [kg/m2-s]'] = data_mode['mass_flux [kg/m2-s]'].
 ↪fillna(data_mode['mass_flux [kg/m2-s]'].mode()[0])
data_mode['D_e [mm]'] = data_mode['D_e [mm]'].fillna(data_mode['D_e [mm]'].mode()[0])
data_mode['D_h [mm]'] = data_mode['D_h [mm]'].fillna(data_mode['D_h [mm]'].mode()[0])
data_mode['length [mm]'] = data_mode['length [mm]'].fillna(data_mode['length [mm]'].
 ↪mode()[0])
data_mode['chf_exp [MW/m2]'] = data_mode['chf_exp [MW/m2]'].
 ↪fillna(data_mode['chf_exp [MW/m2]'].mode()[0])


#Solve for x_e_out
data_mode = impute_values(data_mode)

#add solutions to new df
```

```
mode_sol = samp_sol.copy()
filtered_data_mode = pd.merge(mode_sol['id'], data_mode, on='id', how='left')
filtered_data_mode = filtered_data_mode.dropna(subset=['x_e_out [-]'])
#print(filtered_data_mode)
mode_sol['x_e_out [-]'] = filtered_data_mode['x_e_out [-]']
mode_sol.to_csv('mode_sol.csv')
```

[ ]:
```
#Fixed Value (0)
#FV_0
#Fill all other columns with max of the column
data_FV_0 = data.copy()
data_FV_0['pressure [MPa]'] = data_FV_0['pressure [MPa]'].fillna(0)
data_FV_0['mass_flux [kg/m2-s]'] = data_FV_0['mass_flux [kg/m2-s]'].fillna(0)
data_FV_0['D_e [mm]'] = data_FV_0['D_e [mm]'].fillna(0)
data_FV_0['D_h [mm]'] = data_FV_0['D_h [mm]'].fillna(0)
data_FV_0['length [mm]'] = data_FV_0['length [mm]'].fillna(0)
data_FV_0['chf_exp [MW/m2]'] = data_FV_0['chf_exp [MW/m2]'].fillna(0)
#print(data_max)

#Solve for X_e_out
data_FV_0 = impute_values(data_FV_0)

#Put solution in new df
FV_0_sol = samp_sol.copy()
filtered_data_FV_0 = pd.merge(FV_0_sol['id'], data_FV_0, on='id', how='left')
filtered_data_FV_0 = filtered_data_FV_0.dropna(subset=['x_e_out [-]'])
#print(filtered_data_FV_0)
FV_0_sol['x_e_out [-]'] = filtered_data_FV_0['x_e_out [-]']
#print(FV_0_sol)
FV_0_sol.to_csv('FV_0_sol.csv')
```

[ ]:
```
#Fixed Value (1)
#FV_1
#Fill all other columns with max of the column
data_FV_1 = data.copy()
data_FV_1['pressure [MPa]'] = data_FV_1['pressure [MPa]'].fillna(1)
data_FV_1['mass_flux [kg/m2-s]'] = data_FV_1['mass_flux [kg/m2-s]'].fillna(1)
data_FV_1['D_e [mm]'] = data_FV_1['D_e [mm]'].fillna(1)
data_FV_1['D_h [mm]'] = data_FV_1['D_h [mm]'].fillna(1)
data_FV_1['length [mm]'] = data_FV_1['length [mm]'].fillna(1)
data_FV_1['chf_exp [MW/m2]'] = data_FV_1['chf_exp [MW/m2]'].fillna(1)
#print(data_max)

#Solve for X_e_out
data_FV_1 = impute_values(data_FV_1)

#Put solution in new df
FV_1_sol = samp_sol.copy()
filtered_data_FV_1 = pd.merge(FV_1_sol['id'], data_FV_1, on='id', how='left')
filtered_data_FV_1 = filtered_data_FV_1.dropna(subset=['x_e_out [-]'])
#print(filtered_data_FV_1)
FV_1_sol['x_e_out [-]'] = filtered_data_FV_1['x_e_out [-]']
#print(FV_1_sol)
```

```
FV_1_sol.to_csv('FV_1_sol.csv')
```

[ ]:
```
#Fixed Value (-1)
#FV_-1
#Fill all other columns with max of the column
data_FV_neg = data.copy()
data_FV_neg['pressure [MPa]'] = data_FV_neg['pressure [MPa]'].fillna(-1)
data_FV_neg['mass_flux [kg/m2-s]'] = data_FV_neg['mass_flux [kg/m2-s]'].fillna(-1)
data_FV_neg['D_e [mm]'] = data_FV_neg['D_e [mm]'].fillna(-1)
data_FV_neg['D_h [mm]'] = data_FV_neg['D_h [mm]'].fillna(-1)
data_FV_neg['length [mm]'] = data_FV_neg['length [mm]'].fillna(-1)
data_FV_neg['chf_exp [MW/m2]'] = data_FV_neg['chf_exp [MW/m2]'].fillna(-1)
#print(data_max)

#Solve for X_e_out
data_FV_neg = impute_values(data_FV_neg)

#Put solution in new df
FV_neg_sol = samp_sol.copy()
filtered_data_FV_neg = pd.merge(FV_neg_sol['id'], data_FV_neg, on='id', how='left')
filtered_data_FV_neg = filtered_data_FV_neg.dropna(subset=['x_e_out [-]'])
#print(filtered_data_FV_neg)
FV_neg_sol['x_e_out [-]'] = filtered_data_FV_neg['x_e_out [-]']
#print(FV_neg_sol)
FV_neg_sol.to_csv('FV_neg_sol.csv')
```