1.Main objective of the analysis :

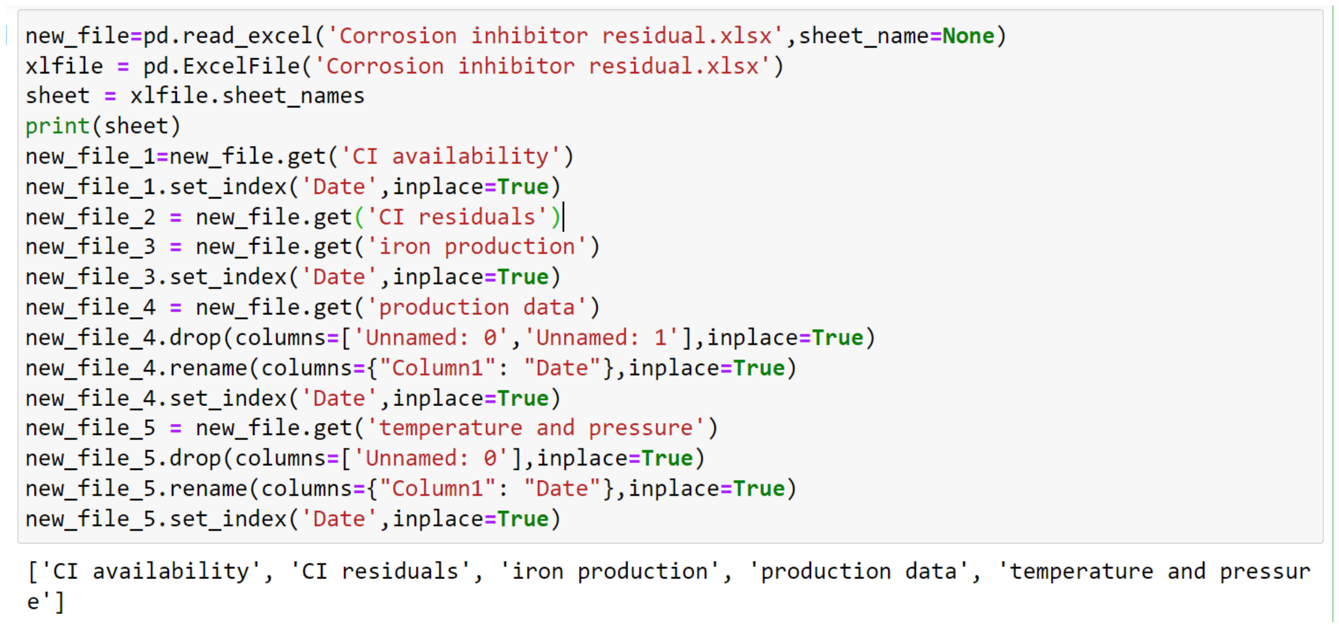
The study is aimed at **predicting the life expectancy** of oil and gas pipelines using Machine Learning. The idea is to save cost in assessing the condition of the pipelines accurately using Machine learning (ML). ML will help assess the condition of the pipeline as accurately as possible which is a <u>potential cost saving technique</u> used by stakeholders to **predict the integrity status of the pipeline.**

Focusing on Regression Analysis in **Deep Learning** using Tensorflow. We are going to build the Neural Network model using **TensorFlow's Sequential class**, where we will predict the health of the pipelines using ML as above. We will predict the 'Residual' column (**target**) using the data from the rest of 10 **variable** columns. First I will check for correlation of variables with the label column. Scaling of the data will follow, which is very essential for deep learning models. We will build our neural net using 4 layers, with '**relu**' activation for three layers and '**linear**' activation for the last layer.There will be different hyperparameters which we will tune to build different models and select the best one, using the least Mean Squared Logarithmic Error(msle).

The best model chosen by us will be used to predict the 'Residual' values for the dates belonging to the year 2022. (Prediction for 2022).

2. Brief description of the data : .

The data is contained in a file (**'Corrosion inhibitor residual.xlsx'**) with relevant data in its different sheets. The sheets contained in the file are named [<u>**'CI availability', 'CI residuals', 'iron production', 'production data', 'temperature and pressure'**</u>]

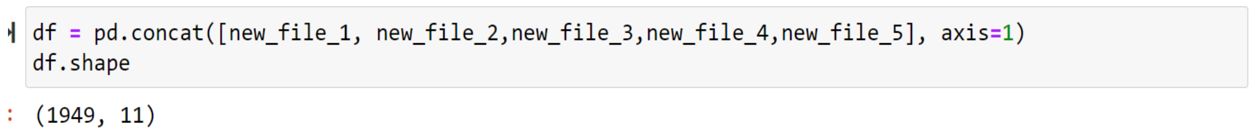


```
new_file=pd.read_excel('Corrosion inhibitor residual.xlsx',sheet_name=None)
xlfile = pd.ExcelFile('Corrosion inhibitor residual.xlsx')
sheet = xlfile.sheet_names
print(sheet)
new_file_1=new_file.get('CI availability')
new_file_1.set_index('Date',inplace=True)
new_file_2 = new_file.get('CI residuals')
new_file_3 = new_file.get('iron production')
new_file_3.set_index('Date',inplace=True)
new_file_4 = new_file.get('production data')
new_file_4.drop(columns=['Unnamed: 0','Unnamed: 1'],inplace=True)
new_file_4.rename(columns={"Column1": "Date"},inplace=True)
new_file_4.set_index('Date',inplace=True)
new_file_5 = new_file.get('temperature and pressure')
new_file_5.drop(columns=['Unnamed: 0'],inplace=True)
new_file_5.rename(columns={"Column1": "Date"},inplace=True)
new_file_5.set_index('Date',inplace=True)
```

```
['CI availability', 'CI residuals', 'iron production', 'production data', 'temperature and pressur
e']
```

The data in these files is read into data frames and concatenated after changing the index of each dataframes into ‘Date’.

The new concatenated dataframe **(df)** is the basis of our data exploration and further cleaning of the data.



```
df = pd.concat([new_file_1, new_file_2,new_file_3,new_file_4,new_file_5], axis=1)
df.shape
```

```
: (1949, 11)
```

<u>Target and Variables :</u>

Of the new data frame(df), the column ‘**Corrosion inhibitor residual, ppm (12" Infield Pipeline)’,** shortened to ‘**Residual**’ is the **Target / Label** . Rest are all **variables**. We are predicting the ‘Residual’ values which will help us predict the pipeline health and our aim is to check the effects of the **factors**(variables) like’ Iron Production’, 'Target Chemicals’ etc. on it.

<u>There are **10 variable** columns</u> :

['Liquids Production', 'Target Dosage', 'Actual Dosage',

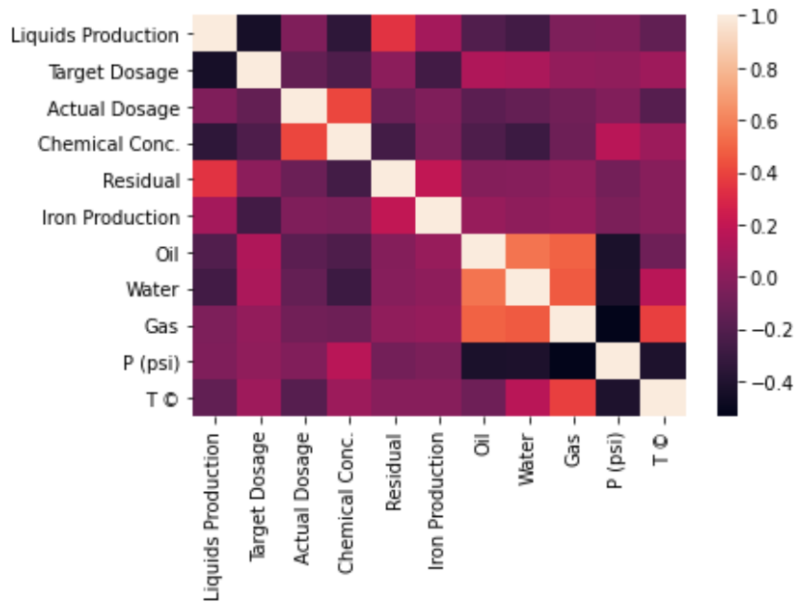
'Chemical Conc.', 'Iron Production', 'Oil', 'Water', 'Gas',

'P (psi)', 'T ©']

and **1 label** column : 'Residual'

Corrosion inhibitor residual, or 'Residual' is the measure of resistance to corrosion of the pipeline and its measurement leads to finding/predicting life expectancy of oil and gas pipelines. We will do a quick check of **correlation** between the features and the target. We will get a heatmap to find the correlation.

```
import seaborn as sns
sns.heatmap(cor_matrix)
```



After dropping all the rows with null values, we are left with **1888 rows × 11** columns in dataframe df.

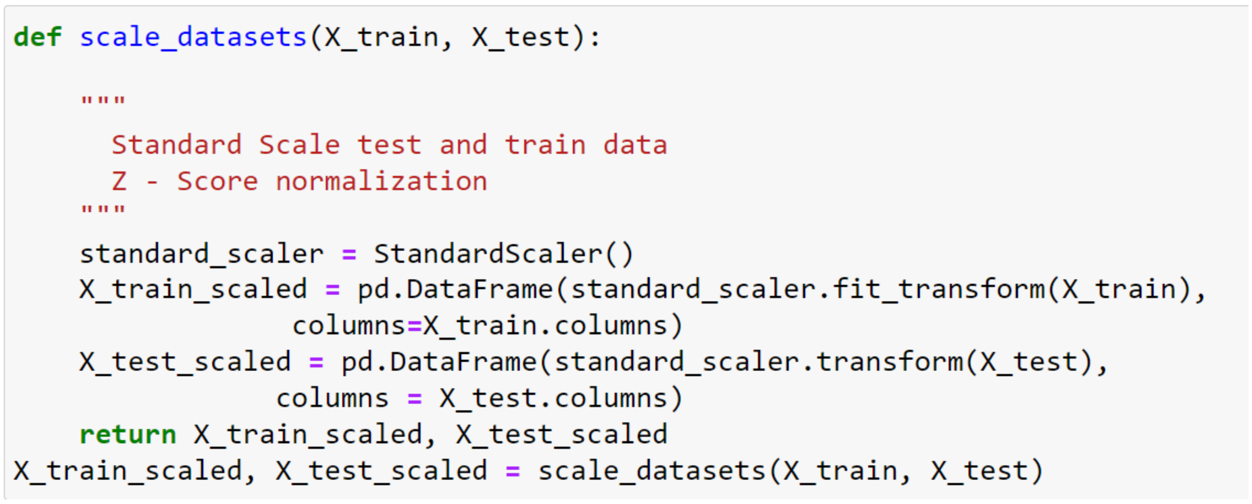
For our project we will predict the values for ‘Residual’ for the year 2022.

3. Brief summary of data exploration and actions taken for data cleaning or feature engineering.

Data exploration leads us to the following steps:

1) For data exploration we print the **(df)** and find the number of **null values in each column**. The data points from **(df)** with no null values for ‘Residual’ are separated into a dataframe **df1**. The **null values** in all the other columns of df1 are **replaced by the mean** of all columns from df. This new **df1** with no null values is **the training set**.

2) The data points with dates from the year 2022 are separated into a data frame **(df2)**, which we need predictions for. The column ‘Residual’ has all null values and is **dropped**.The **null values** in all the other columns of df2 are **replaced by the mean** of all columns from df. This new **df2** with no null values is **the test set**.
3) The **(df1)** is split up into train and validation sets with test_size=0.25.
4) The **data is scaled** using StandardScaler.



```
def scale_datasets(X_train, X_test):

    """
      Standard Scale test and train data
      Z - Score normalization
    """
    standard_scaler = StandardScaler()
    X_train_scaled = pd.DataFrame(standard_scaler.fit_transform(X_train),
                 columns=X_train.columns)
    X_test_scaled = pd.DataFrame(standard_scaler.transform(X_test),
                 columns = X_test.columns)
    return X_train_scaled, X_test_scaled
X_train_scaled, X_test_scaled = scale_datasets(X_train, X_test)
```

Summary of training at least three variations of the Deep Learning model

For building the model with the **Sequential class** we have used four layers.

The first layer :

160 hidden units/ neurons with the ReLU activation function.

The second layer :

480 hidden units with the ReLU activation function.

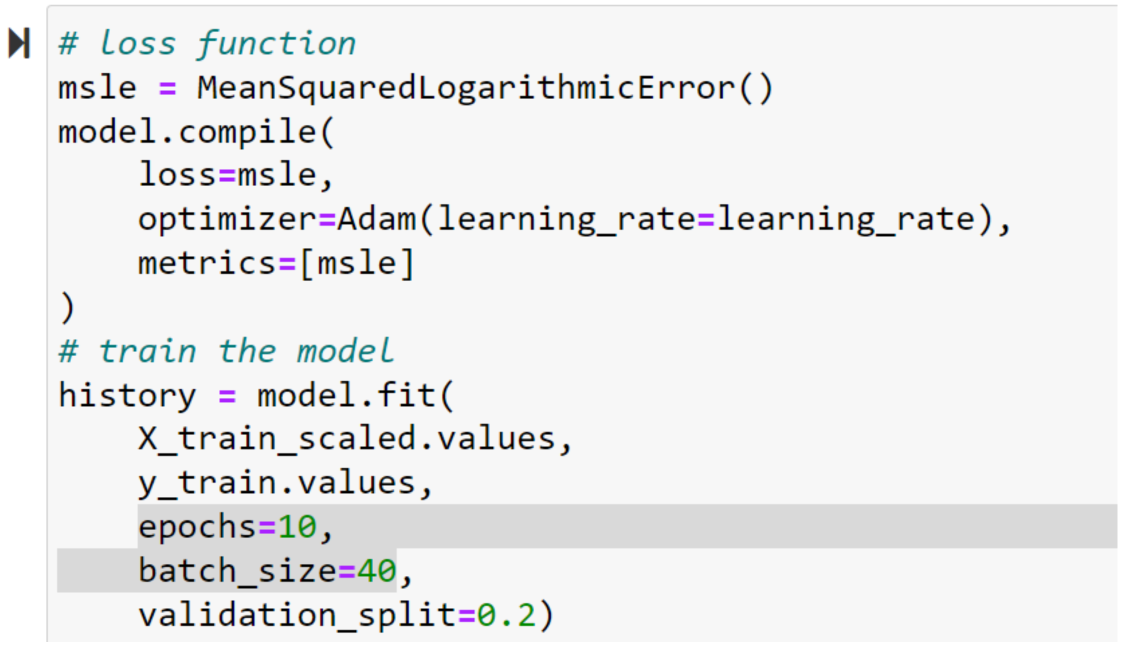
<u>The third layer :</u>

256 hidden units with the ReLU activation function.

<u>The fourth layer :</u>

1 unit with a linear activation function

To compile the model with training configurations, we use **Mean Squared Logarithmic Loss** as loss function and metric, and **Adam** loss function optimizer. For training, we use the function fit with the parameters, **features, target, epochs, batch size**, and **validation split.**



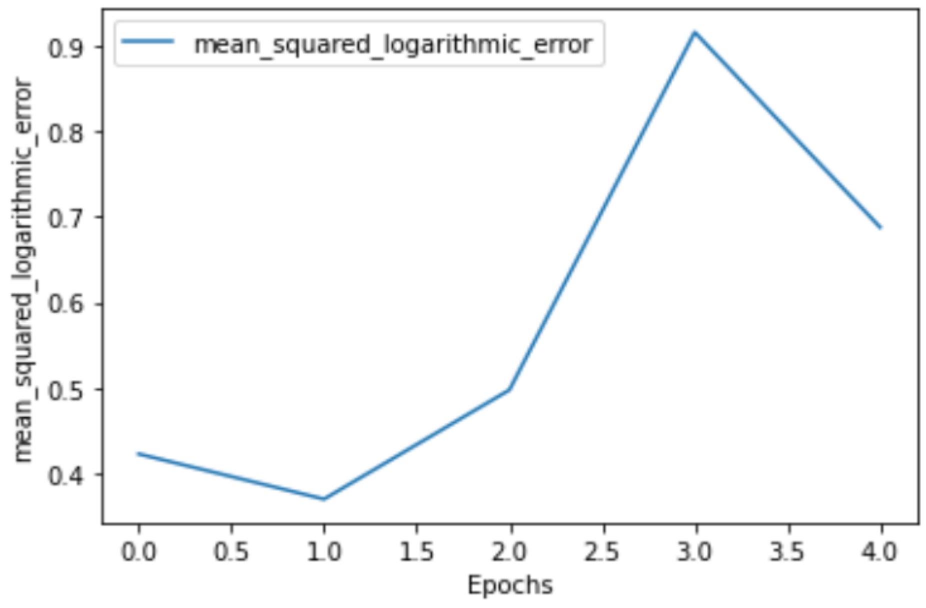
```
# loss function
msle = MeanSquaredLogarithmicError()
model.compile(
    loss=msle,
    optimizer=Adam(learning_rate=learning_rate),
    metrics=[msle]
)
# train the model
history = model.fit(
    X_train_scaled.values,
    y_train.values,
    epochs=10,
    batch_size=40,
    validation_split=0.2)
```

Hyperparameter Tuning:

Epochs and batch size are changed for three different models trained and the MLSE(mean squared logarithmic error) is plotted against the number of epochs. The number of epochs for the minimum value for MLSE for a particular batch size is noted.

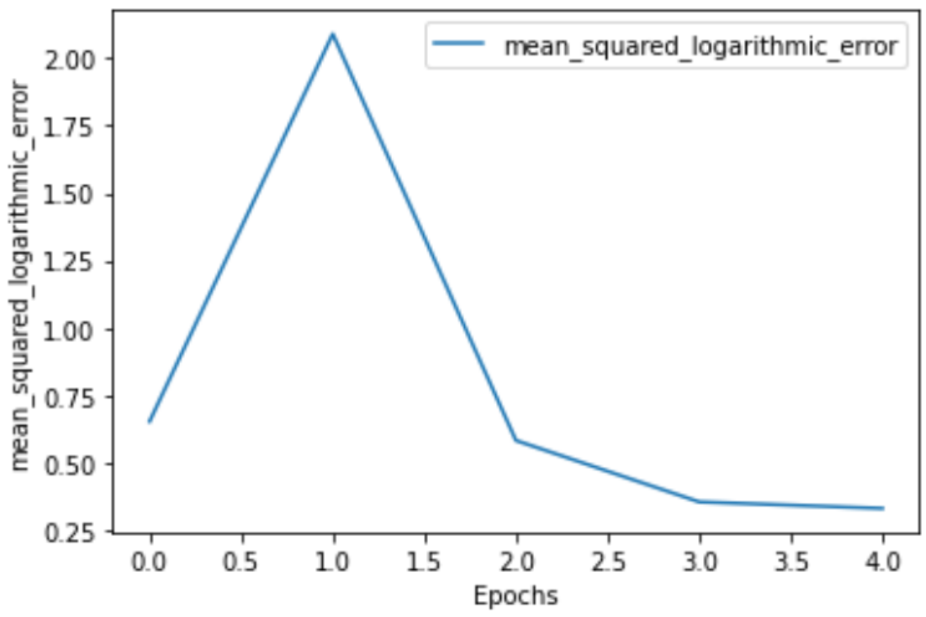
First model :

epochs=10, batch_size=40



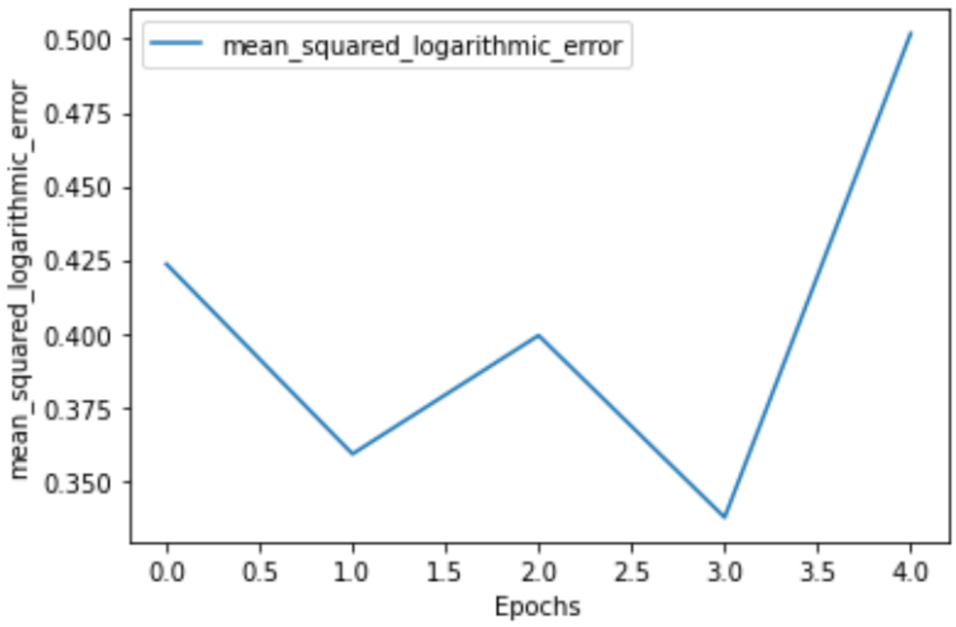
Second model :

epochs=6, batch_size=20



Third model :

epochs=5, batch_size=30



Deep Learning model recommended :

Reading the graphs above, it is evident that the model that suits our needs best in terms of accuracy are the ones with:

Serial No.	Batch size	Epochs	Validation split	Mean Squared Logarithmic Error (msle)
1.	40(whole dataset)	1	0.2	0.3 <msle<0.4
2.	20(half dataset)	3	0.2	~0.375
3.	30	3	0.2	<0.350

Of all the above, the model with batch size 30, epoch =

3 and validation split = 0.2 gives a **Mean Squared Logistic Error < 0.350**. This model seems to be the most **accurate and yields** best results in terms of our aim of prediction. So, I will build a model with above **hyperparameters**.

Summary Key Findings and Insights :.

The full batch of 40 yields the best result with a single epoch. A similar but better result is obtained with batch size of 30 and epoch=3, which is even better than using half the dataset at a time with batch size of 20 and epoch=3(3 runs of the entire batch size).

Having made provisions for **cross validation** in the code, the chances of overfitting are minimized. A **sequential model** thus built should give the best results in terms of accuracy, with a **batch size of 30 (¾ of the dataset at a time) and number of epochs=3.**

The **time taken to build the model** is also pretty reasonable, and the dataset is not too large.

Suggestions for next steps in analyzing this data

This model built using the **sequential method** looks satisfactory in terms of accuracy of prediction, but I still feel the need to get a **larger dataset for training,** so that better and accurate predictions can be made.

We can go for a different **validation split** and check for better accuracy.

We can test this model against a different metric, like **Mean Absolute Error (MAE)** in order to tune the hyperparameters of the model for better accuracy.

Pipeline data is always subject to rapid changes and safety of oilfield workers and valuable human resource is at stake while working in the field, so accuracy is of high value in this context.