Assignment 0 Report
From Data to ML and NN Models

Instructor: Alina Vereshchaka
Name: Rakshith Kumar Narasimha Murthy
UB mail id : rakshit2@buffalo.edu
Student ID: 50560326

Problem Statement:

The goal of this analysis is to predict the NOx Rate (lbs/mmBtu), which is the rate of nitrogen oxides (NOx) emissions per unit of heat input for industrial facilities. Nitrogen oxides are important pollutants that contribute to issues related to the environment such as smog and acid rain, making it crucial to understand and predict their emission rates accurately.

Dataset I have chosen:

I have chosen a dataset from https://catalog.data.gov/dataset/clean-air-markets-allowances-query-wizard data.gov, specifically from the https://campd.epa.gov/data/bulk-data-files Clean Air Markets Program Data Application (CAMPD). This application, accessible at CAMPD, allows users to view and download Clean Air Markets data hourly, monthly and yearly.

EPA's Clean Air Markets Division (CAMD) oversees several market-based regulatory programs designed to improve air quality and protect ecosystems. Notable programs include EPA's Acid Rain Program and the Cross-State Air Pollution Rule (CSAPR) Programs, which aim to reduce emissions of sulfur dioxide (SO2) and nitrogen oxides (NOx), compounds that negatively impact air quality, the environment, and public health.

From this application, I have downloaded the emissions hourly dataset for the year 2023. which includes various features related to emissions, such as SO2 Mass, Heat Input, NOx Mass, CO2 Mass, and operational parameters like Gross Load and Operating Time. The **target** variable for the analysis is the **NOx Rate (lbs/mmBtu)**.

Type of Data:

The data encountered is primarily quantitative, comprising numerical values for emissions and operational metrics. Additionally, there are categorical variables like 'Facility ID' which identify different facilities, and 'Hour', which represents the time of measurement. The dataset is structured,

meaning it is organized in a tabular format with rows representing individual records and columns representing features.

Nature of the Problem:

This problem falls under the category of supervised regression. In supervised learning, we have a dataset with input features and a corresponding target variable, and the task is to learn a mapping from inputs to the target.

Supervised Learning: We have labeled data where the target variable (NOx Rate) is known for each observation.
Regression Task: As the target variable, that is NOx Rate (lbs/mmBtu), is continuous and numerical, this makes this a regression problem.

Objective:

To build a predictive model that can estimate the NOx Rate based on various features related to emissions and operational metrics of industrial facilities.

Main Statistics:

Here are the main statistics for the dataset before preprocessing:

Head:

```
In [4]: df.head() #displaying top 5 samples
```

Out[4]:

	State	Facility Name	Facility ID	Unit ID	Associated Stacks	Date	Hour	Operating Time	Gross Load (MW)	Steam Load (1000 lb/hr)	...	Heat Input (mmBtu)	Heat Input Measure Indicator	Primary Fuel Type	Secondary Fuel Type	Unit Type	SO2 Controls	N
0	WA	Fredonia Generating Station	607	CT3	NaN	2023-01-01	0	0.0	NaN	NaN	...	NaN	NaN	Natural Gas	Diesel Oil	Combustion turbine	NaN	Inject
1	WA	Fredonia Generating Station	607	CT3	NaN	2023-01-01	1	0.0	NaN	NaN	...	NaN	NaN	Natural Gas	Diesel Oil	Combustion turbine	NaN	Inject
2	WA	Fredonia Generating Station	607	CT3	NaN	2023-01-01	2	0.0	NaN	NaN	...	NaN	NaN	Natural Gas	Diesel Oil	Combustion turbine	NaN	Inject
3	WA	Fredonia Generating Station	607	CT3	NaN	2023-01-01	3	0.0	NaN	NaN	...	NaN	NaN	Natural Gas	Diesel Oil	Combustion turbine	NaN	Inject
4	WA	Fredonia Generating Station	607	CT3	NaN	2023-01-01	4	0.0	NaN	NaN	...	NaN	NaN	Natural Gas	Diesel Oil	Combustion turbine	NaN	Inject

5 rows × 32 columns

Fig 1 : Image shows the top 5 rows with all 32 columns

In the head analysis we can see more NaN values starting from Gross Load which has to be cleaned and we can also see the many features and their values from the above snap.
Description:

```
In [7]: df.describe() #description of the datasets using count , mean, max, min etc
```

Out[7]:

	Facility ID	Associated Stacks	Hour	Operating Time	Gross Load (MW)	Steam Load (1000 lb/hr)	SO2 Mass (lbs)	SO2 Rate (lbs/mmBtu)	CO2 Mass (short tons)	CO2 Rate (short tons/mmBtu)	NOx M
count	148920.000000	0.0	148920.00000	148920.000000	106691.000000	0.0	106691.000000	106691.000000	106691.000000	106691.000000	106691.00
mean	29067.411765	NaN	11.50000	0.712389	196.353563	NaN	23.227993	0.004974	119.172485	0.062148	91.86
std	24772.816818	NaN	6.92221	0.451670	156.590432	NaN	124.728573	0.039857	161.532923	0.011508	291.04
min	607.000000	NaN	0.00000	0.000000	0.000000	NaN	0.000000	0.000000	0.005000	0.045000	0.00
25%	7870.000000	NaN	5.75000	0.000000	74.000000	NaN	0.929000	0.001000	55.000000	0.059000	11.31
50%	7999.000000	NaN	11.50000	1.000000	199.000000	NaN	1.285000	0.002000	84.900000	0.059000	14.25
75%	55482.000000	NaN	17.25000	1.000000	277.000000	NaN	2.429000	0.002000	113.500000	0.059000	19.39
max	55818.000000	NaN	23.00000	1.000000	712.000000	NaN	10307.100000	3.149000	800.700000	0.105000	3219.25


```
In [7]: df.describe() #description of the datasets using count , mean, max, min etc
```

Out[7]:

our	Operating Time	Gross Load (MW)	Steam Load (1000 lb/hr)	SO2 Mass (lbs)	SO2 Rate (lbs/mmBtu)	CO2 Mass (short tons)	CO2 Rate (short tons/mmBtu)	NOx Mass (lbs)	NOx Rate (lbs/mmBtu)	Heat Input (mmBtu)	Hg Controls
000	148920.000000	106691.000000	0.0	106691.000000	106691.000000	106691.000000	106691.000000	106691.000000	106691.000000	106691.000000	0.0
000	0.712389	196.353563	NaN	23.227993	0.004974	119.172485	0.062148	91.860099	0.027059	1666.054129	NaN
221	0.451670	156.590432	NaN	124.728573	0.039857	161.532923	0.011508	291.046074	0.049385	1492.440255	NaN
000	0.000000	0.000000	NaN	0.000000	0.000000	0.005000	0.045000	0.000000	0.000000	0.110000	NaN
000	0.000000	74.000000	NaN	0.929000	0.001000	55.000000	0.059000	11.310500	0.008000	924.400000	NaN
000	1.000000	199.000000	NaN	1.285000	0.002000	84.900000	0.059000	14.251000	0.012000	1428.700000	NaN
000	1.000000	277.000000	NaN	2.429000	0.002000	113.500000	0.059000	19.396000	0.025000	1909.600000	NaN
000	1.000000	712.000000	NaN	10307.100000	3.149000	800.700000	0.105000	3219.256000	1.499000	7634.400000	NaN

Fig 2 : Description about the features such as count, mean, std, min and max

The table contains statistical summary metrics like count, mean, standard deviation, minimum, maximum, quartiles (25th and 75th percentiles), and percentiles (10th and 90th percentiles) for each of the features.

Shape:

```
In [8]: df.shape #Number of rows and columns in the datasets

Out[8]: (148920, 32)
```

Fig 3 : Image shows the shape that is rows and columns count

The above raw dataset contains 148920 rows and 32 columns which includes both categorical and numerical features in the dataset.

Description Object

```
In [11]: df.describe(include = "object")
```

Out[11]:

	State	Facility Name	Unit ID	Date	SO2 Mass Measure Indicator	SO2 Rate Measure Indicator	CO2 Mass Measure Indicator	CO2 Rate Measure Indicator	NOx Mass Measure Indicator	NOx Rate Measure Indicator	Heat Input Measure Indicator	Primary Fuel Type	Secondary Fuel Type	Unit Type	S Contr
count	148920	148920	148920	148920	106691	106691	106691	106691	106691	106691	106691	148920	35040	148920	87
unique	1	11	12	365	3	1	3	1	1	2	3	3	1	3	
top	WA	Encogen Generating Station	CT3	2023-01-01	Measured	Calculated	Measured	Calculated	Calculated	Measured	Measured	Natural Gas	Diesel Oil	Combined cycle	W Limesto
freq	148920	26280	17520	408	94555	106691	105781	106691	106691	105829	105781	122640	35040	122640	87

Fig 4 : Description of object data type

In the above image we can see the analysis of the columns which have object data type

Info

```
In [6]: df.info() #the values in the features and type of the feature  #sanity check
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 148920 entries, 0 to 148919
Data columns (total 32 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   State                           148920 non-null  object
 1   Facility Name                   148920 non-null  object
 2   Facility ID                     148920 non-null  int64
 3   Unit ID                         148920 non-null  object
 4   Associated Stacks               0 non-null       float64
 5   Date                            148920 non-null  object
 6   Hour                            148920 non-null  int64
 7   Operating Time                  148920 non-null  float64
 8   Gross Load (MW)                 106691 non-null  float64
 9   Steam Load (1000 lb/hr)         0 non-null       float64
 10  SO2 Mass (lbs)                  106691 non-null  float64
 11  SO2 Mass Measure Indicator      106691 non-null  object
 12  SO2 Rate (lbs/mmBtu)            106691 non-null  float64
 13  SO2 Rate Measure Indicator      106691 non-null  object
 14  CO2 Mass (short tons)           106691 non-null  float64
 15  CO2 Mass Measure Indicator      106691 non-null  object
 16  CO2 Rate (short tons/mmBtu)     106691 non-null  float64
 17  CO2 Rate Measure Indicator      106691 non-null  object
 18  NOx Mass (lbs)                  106691 non-null  float64
 19  NOx Mass Measure Indicator      106691 non-null  object
 20  NOx Rate (lbs/mmBtu)            106691 non-null  float64
 21  NOx Rate Measure Indicator      106691 non-null  object
 22  Heat Input (mmBtu)              106691 non-null  float64
 23  Heat Input Measure Indicator    106691 non-null  object
 24  Primary Fuel Type               148920 non-null  object
 25  Secondary Fuel Type             35040 non-null   object
 26  Unit Type                       148920 non-null  object
 27  SO2 Controls                    8760 non-null    object
 28  NOx Controls                    148920 non-null  object
 29  PM Controls                     8760 non-null    object
 30  Hg Controls                     0 non-null       float64
 31  Program Code                    148920 non-null  object
dtypes: float64(12), int64(2), object(18)
```

Fig 5 : Information about features, values and their data types

The total columns in the above snap contains information on hourly emission data's and their column operations for a specific time period. It includes various features related to fuel consumption, electricity generation, and emissions. This data contains the mixture of float, object and int data types.

Missing value count:

```
In [9]: #finding missing value
        df.isnull().sum()

Out[9]: State                          0
        Facility Name                  0
        Facility ID                    0
        Unit ID                        0
        Associated Stacks         148920
        Date                           0
        Hour                           0
        Operating Time                 0
        Gross Load (MW)            42229
        Steam Load (1000 lb/hr)   148920
        SO2 Mass (lbs)             42229
        SO2 Mass Measure Indicator 42229
        SO2 Rate (lbs/mmBtu)       42229
        SO2 Rate Measure Indicator 42229
        CO2 Mass (short tons)      42229
        CO2 Mass Measure Indicator 42229
        CO2 Rate (short tons/mmBtu) 42229
        CO2 Rate Measure Indicator 42229
        NOx Mass (lbs)             42229
        NOx Mass Measure Indicator 42229
        NOx Rate (lbs/mmBtu)       42229
        NOx Rate Measure Indicator 42229
        Heat Input (mmBtu)         42229
        Heat Input Measure Indicator 42229
        Primary Fuel Type              0
        Secondary Fuel Type       113880
        Unit Type                      0
        SO2 Controls              140160
        NOx Controls                   0
        PM Controls               140160
        Hg Controls               148920
        Program Code                   0
        dtype: int64
```

Fig 6 : Counting the NULL values in the dataset

The above snap shows the null values in the columns and their count of every feature

Missing Value Percentage:

```
In [10]: df.isnull().sum()/df.shape[0]*100 #getting percentage of missing value
```

```
Out[10]: State                              0.000000
         Facility Name                      0.000000
         Facility ID                        0.000000
         Unit ID                            0.000000
         Associated Stacks                100.000000
         Date                               0.000000
         Hour                               0.000000
         Operating Time                     0.000000
         Gross Load (MW)                   28.356836
         Steam Load (1000 lb/hr)          100.000000
         SO2 Mass (lbs)                    28.356836
         SO2 Mass Measure Indicator        28.356836
         SO2 Rate (lbs/mmBtu)              28.356836
         SO2 Rate Measure Indicator        28.356836
         CO2 Mass (short tons)             28.356836
         CO2 Mass Measure Indicator        28.356836
         CO2 Rate (short tons/mmBtu)       28.356836
         CO2 Rate Measure Indicator        28.356836
         NOx Mass (lbs)                    28.356836
         NOx Mass Measure Indicator        28.356836
         NOx Rate (lbs/mmBtu)              28.356836
         NOx Rate Measure Indicator        28.356836
         Heat Input (mmBtu)                28.356836
         Heat Input Measure Indicator      28.356836
         Primary Fuel Type                  0.000000
         Secondary Fuel Type               76.470588
         Unit Type                          0.000000
         SO2 Controls                      94.117647
         NOx Controls                       0.000000
         PM Controls                       94.117647
         Hg Controls                      100.000000
         Program Code                       0.000000
         dtype: float64
```

Fig 7 : Counting the percentage of missing values

Calculated the percentage of missing values helps us to identify features in my dataset that have a high percentage of missing values (NaN values). I then removed these features as they have a threshold (e.g., 75%) of missing values.

Preprocessing Techniques: Data Cleaning

For the emissions dataset, I have applied several preprocessing techniques to ensure the data was clean and suitable for training machine learning models. Below i have explained the steps I took to preprocess my dataset

- **Handling Missing Values:** Checked for any missing values in the dataset and dropped some of the columns with more than 75% missing values because using mean or median imputation wouldn't accurately represent the missing data. This decision was made because mean, median, or mode imputation might not accurately represent the missing data, especially for features with a large portion missing, which could lead to inaccurate imputation techniques. Additionally, including features with many missing values can negatively impact the performance of machine learning models, reducing their overall performance.

```
In [11]: #dropping features as it contains 100% null values or above 75% null values
         drop_columns = [4,9,25,27,29,30]
         df.drop(df.columns[drop_columns], axis=1, inplace=True)
```

Fig 8 : Dropping columns which had more than 75% Null values

- **Handling Missing Values in Rows:** Dropping the rows containing null values for features such as Gross Load (MW), SO2 Mass (lbs), SO2 Mass Measure Indicator, SO2 Rate (lbs/mmBtu), SO2 Rate Measure Indicator, CO2 Mass (short tons), CO2 Mass Measure Indicator, CO2 Rate (short tons/mmBtu), CO2 Rate Measure Indicator, NOx Mass (lbs), NOx Mass Measure Indicator, NOx Rate (lbs/mmBtu), NOx Rate Measure Indicator, Heat Input (mmBtu), and Heat Input Measure Indicator is crucial for the analysis. These features are considered important, and removing rows with missing values ensures data integrity for subsequent analysis.

```
In [15]: df_cleaned = df.dropna()  #Dropping the rows which have null values as the Gross Load (MW),SO2 Mass (lbs), SO2 Mass Measure Indic
         #SO2 Rate (lbs/mmBtu),SO2 Rate Measure Indicator, CO2 Mass (short tons), CO2 Mass Measure Indicator,CO2 Rate (short tons/mmBtu),
          #NOx Mass (lbs), NOx Mass Measure Indicator, NOx Rate (lbs/mmBtu), NOx Rate Measure Indicator, Heat Input (mmBtu), Heat Input Me
         #is important features for the analysis
```

Fig 9 : Dropping missing values in rows

- **Handling mismatched string formats:** The column ‘Unit ID’ has a values which were repeated with different name so we have to format this string and converting the string to upper case and checking if it has a non-alphanumeric characters and after handling we can see the change in the count of values and the format.

```
In [19]: df_cleaned['Unit ID'].value_counts(dropna=False)  #cheking if it has a mismatched string formats

Out[19]: Unit ID
         CT-1     14832
         1        14136
         CT2      13085
         CT1      11412
         CT3       9271
         F1CT      7378
         BW22      7155
         CTG1      7046
         2         6836
         CT-1A     6506
         CT-1B     6129
         CT4       2905
         Name: count, dtype: int64
```

Fig 10 : Before handling mismatched strings

4 Handling mismatched string formats

```
In [20]: df_cleaned['Unit ID'] = df_cleaned['Unit ID'].str.replace(r'[^A-Za-z0-9]+', '')  # Removing non-alphanumeric characters
         df_cleaned['Unit ID'] = df_cleaned['Unit ID'].str.upper()  # Convert to uppercase
         df_cleaned['Unit ID'] = df_cleaned['Unit ID'].str.replace('CT1', 'CT-1') #replacing with correct strings
         df_cleaned['Unit ID'] = df_cleaned['Unit ID'].str.replace('CT2', 'CT-2')
         df_cleaned['Unit ID'] = df_cleaned['Unit ID'].str.replace('CT3', 'CT-3')
         df_cleaned['Unit ID'] = df_cleaned['Unit ID'].str.replace('CT4', 'CT-4')
         df_cleaned['Unit ID'] = df_cleaned['Unit ID'].str.replace('CTG1', 'CTG-1')
         df_cleaned['Unit ID'] = df_cleaned['Unit ID'].str.replace('BW22', 'BW-22')
```

Fig 11 : Code used to handle mismatched string

```
In [21]: df_cleaned['Unit ID'].value_counts(dropna=False) #after formatting
Out[21]: Unit ID
         CT-1     26244
         1        14136
         CT-2     13085
         CT-3      9271
         F1CT      7378
         BW-22     7155
         CTG-1     7046
         2         6836
         CT-1A     6506
         CT-1B     6129
         CT-4      2905
         Name: count, dtype: int64
```

Fig 12 : Column ‘Unit ID’ after handling string

- **Handling Outliers:** In handling outliers for the 'SO2 Rate (lbs/mmBtu)' feature, it was observed that some values exceeded the expected range, reaching up to 3 while it should ideally be around 1. To address this issue, an imputation method was employed. Values greater than 1 were replaced with the median to mitigate the impact of outliers and ensure that the data aligns more closely with the expected distribution. This approach helps in maintaining the integrity of the data and ensures that the 'SO2 Rate (lbs/mmBtu)' feature remains within a reasonable range for subsequent analysis.

```
In [22]: df_cleaned['SO2 Rate (lbs/mmBtu)'].value_counts(dropna=False) #checking the outliers
Out[22]: SO2 Rate (lbs/mmBtu)
         0.002     51701
         0.001     43188
         0.000      4657
         0.049       246
         0.046       236
                    ...
         3.002         1
         0.131         1
         0.664         1
         0.179         1
         0.560         1
         Name: count, Length: 131, dtype: int64
```

Fig 13 : SO2 values before handling we can see values > 1

5 Handling outliers using impute outliers method

```
In [23]: #as we see the value in SO2 Rate which is greater than 1 we replace those with median to avoid outliers
         outliers = df_cleaned['SO2 Rate (lbs/mmBtu)'] > 1.0
         replacement_value = df_cleaned.loc[~outliers, 'SO2 Rate (lbs/mmBtu)'].median()
         df_cleaned.loc[outliers, 'SO2 Rate (lbs/mmBtu)'] = replacement_value
```

Fig 14 : Code for the above imputer method

```
In [24]: df_cleaned['SO2 Rate (lbs/mmBtu)'].value_counts(dropna=False)

Out[24]: SO2 Rate (lbs/mmBtu)
         0.002    51732
         0.001    43188
         0.000     4657
         0.049      246
         0.046      236
                  ...
         0.301        1
         0.560        1
         0.336        1
         0.131        1
         0.932        1
         Name: count, Length: 119, dtype: int64
```

Fig 15 : SO2 values after using median for outliers

- **Identifying uncorrelated features:** In this step, the focus is on identifying features with low correlation to the target variable using a correlation matrix and a predefined threshold. Features that exhibit weak correlations with the target variable will be dropped from the analysis. By evaluating the strength of the relationships between features and the target variable, we can streamline the dataset and improve the relevance of the remaining features for subsequent analysis.

7) Identifing uncorrelated or unrelated features.

```
In [40]: #identifying the features which are not related using co relation matrix and also dropping that feature
         numeric_columns = df_cleaned.select_dtypes(include=[np.number]).columns.tolist()
         correlation_matrix = df_cleaned[numeric_columns].corr()

In [41]: target = 'NOx Rate (lbs/mmBtu)'
         correlation_with_target = correlation_matrix[target].drop(target)

In [42]: threshold = 0.1
         low_correlation_features = correlation_with_target[abs(correlation_with_target) < threshold].index.tolist()

In [43]: print(f"Features with low correlation to the target variable:{low_correlation_features}")
         Features with low correlation to the target variable:['Hour']

In [44]: df_cleaned = df_cleaned.drop(columns=low_correlation_features) #dropping low correlation features

In [45]: print("Original dataset shape:", df_cleaned.shape)
         Original dataset shape: (106691, 25)
```

Fig 16 : identifying unrelated features

- **Converting categorical features into numerical using One Hot Encoding:** This step is a crucial preprocessing step for analysis. By transforming categorical variables into numerical representations. We enable machine learning models to effectively utilize these features for prediction. This process enhances the model's predictive capability by providing it with a structured input format that captures the categorical information present in the dataset. Ultimately, this facilitates better model performance and accuracy in predictive tasks.

8. Converting categorical features into numerical using One-Hot Encoding

```
In [46]: df_cleaned = pd.get_dummies(df_cleaned, columns=['Unit ID'])

In [47]: df_cleaned['Heat Input Measure Indicator'].replace('', 'Substitute', inplace=True)
         df_cleaned = pd.get_dummies(df_cleaned, columns=['Heat Input Measure Indicator'])
```

Fig 17 : Using get_dummies converting ‘Unit Id’ and ‘Heat Input Measure Indicator’ into numerical values

```
In [49]: df_cleaned = pd.get_dummies(df_cleaned, columns=['Facility Name'])

In [50]: df_cleaned = pd.get_dummies(df_cleaned, columns=['SO2 Mass Measure Indicator'])

In [51]: df_cleaned = pd.get_dummies(df_cleaned, columns=['CO2 Mass Measure Indicator'])

In [52]: df_cleaned = pd.get_dummies(df_cleaned, columns=['Primary Fuel Type'])

In [53]: df_cleaned = pd.get_dummies(df_cleaned, columns=['Unit Type'])

In [54]: df_cleaned = pd.get_dummies(df_cleaned, columns=['NOx Controls'])

In [55]: df_cleaned = pd.get_dummies(df_cleaned, columns=['Program Code'])
```

Fig 18 : Using get_dummies converting ‘Facility name’, ‘SO2 Mass Measure Indicator’, ‘CO2 Mass Measure Indicator’, ‘Primary Fuel Type’, ‘Unit Type’,’NOx Controls’, ‘Program Code’ to numerical features.

- **Identifying columns with single unique value:** Identifying and subsequently dropping columns with a single unique value, such as "State", is an essential preprocessing step for model development. These columns, having only one unique value, do not provide any meaningful variation in the dataset and therefore offer no predictive power to the model. By removing such columns, we streamline the dataset and focus the model's attention on features that actually contribute to the prediction process, thus improving the model's efficiency and interpretability.

identifying columns with single unique value and dropping them as it won't help for the model like "State"

```
In [48]: single_value_columns = [col for col in df_cleaned.columns if df_cleaned[col].nunique() == 1]
         df_cleaned.drop(columns=single_value_columns, inplace=True)
```

Fig 19 : Dropping columns with single unique value

- **Converting boolean columns to integers:** Converting boolean columns to integers, where True is represented as 1 and False as 0, ensures uniformity in the feature representation for model training. By transforming boolean values into numerical equivalents, we standardize the data format across all features, allowing the model to interpret and utilize them consistently during analysis. This process helps maintain the integrity of the dataset and ensures that all features contribute equally to the model's predictive capability, thereby enhancing the overall performance and interpretability of the model.

```
In [57]: # Convert boolean columns to integers
         for col in df_cleaned.columns:
             if df_cleaned[col].dtype == 'bool':
                 df_cleaned[col] = df_cleaned[col].astype(int)
```

Fig 20 : Boolean to integer

- **Normalizing non categorical features using MinMaxScaler:** Normalizing non-categorical features using MinMaxScaler is a standard preprocessing technique. By scaling numerical features to a specific range, typically between 0 and 1, MinMaxScaler ensures that all features are on a comparable scale. This prevents features with larger magnitudes from dominating the model training process and allows for more stable and efficient convergence during optimization. Normalization using MinMaxScaler also helps to improve the performance of certain algorithms, particularly those sensitive to feature scaling, such as gradient descent-based methods. Overall, this preprocessing step facilitates better model training and enhances the model's ability to generalize to new data.

9) Normalize non-categorical features using MinMaxScaler

```
In [59]: from sklearn.preprocessing import MinMaxScaler
         numerical_columns = df_cleaned.select_dtypes(include='number')
         scaler = MinMaxScaler()
         scaled_data = scaler.fit_transform(numerical_columns)
         df_cleaned = pd.DataFrame(scaled_data, columns=numerical_columns.columns)
```

Fig 21 : Using MinMaxScaler to scale the features

- **Choosing target Y and X features for model training :** Selecting Target (Y) and Feature (X) Variables for Model Training: I have chosen the features and target values for my model.

10) Choosing target Y and features X ¶

```
features = df_cleaned[['Gross Load (MW)','SO2 Mass (lbs)','Heat Input (mmBtu)','NOx Mass (lbs)','CO2 Rate (short tons/mmBtu)',
                       'CO2 Mass (short tons)','SO2 Rate (lbs/mmBtu)','Program Code_ARP', 'Program Code_ARP, MATS','NOx Controls
       'NOx Controls_Dry Low NOx Burners,Selective Catalytic Reduction',
       'NOx Controls_Dry Low NOx Burners,Water Injection,Selective Catalytic Reduction',
       'NOx Controls_Low NOx Burner Technology w/ Closed-coupled/Separated OFA',
       'NOx Controls_Selective Catalytic Reduction',
       'NOx Controls_Selective Catalytic Reduction,Steam Injection',
       'NOx Controls_Water Injection,Selective Catalytic Reduction','Primary Fuel Type_Coal',
       'Primary Fuel Type_Natural Gas',
       'Primary Fuel Type_Pipeline Natural Gas', 'Unit Type_Combined cycle',
       'Unit Type_Combustion turbine', 'Unit Type_Tangentially-fired','Primary Fuel Type_Coal',
       'Primary Fuel Type_Natural Gas',
       'Primary Fuel Type_Pipeline Natural Gas','SO2 Mass Measure Indicator_Measured',
       'SO2 Mass Measure Indicator_Measured and Substitute',
       'SO2 Mass Measure Indicator_Substitute',
       'CO2 Mass Measure Indicator_Measured',
       'CO2 Mass Measure Indicator_Measured and Substitute',
       'CO2 Mass Measure Indicator_Substitute','Facility Name_Centralia',
       'Facility Name_Chehalis Generation Facility',
       'Facility Name_Encogen Generating Station',
       'Facility Name_Ferndale Generating Station',
       'Facility Name_Frederickson Power LP',
       'Facility Name_Fredonia Generating Station',
       'Facility Name_Goldendale Generating Station',
       'Facility Name_Grays Harbor Energy Center',
       'Facility Name_Mint Farm Generating Station',
       'Facility Name_River Road', 'Facility Name_Sumas Generating Station',  'Heat Input Measure Indicator_Measured',
       'Heat Input Measure Indicator_Measured and Substitute',
       'Heat Input Measure Indicator_Substitute', 'Unit ID_1', 'Unit ID_2', 'Unit ID_BW-22',
       'Unit ID_CT-1', 'Unit ID_CT-1A', 'Unit ID_CT-1B', 'Unit ID_CT-2',
       'Unit ID_CT-3', 'Unit ID_CT-4', 'Unit ID_CTG-1', 'Unit ID_F1CT',]]
target = df_cleaned['NOx Rate (lbs/mmBtu)']
print(len(features.columns))
```

```
56
```

Fig 21 : Selected Features and target variable

- **Splitting the Dataset:** Splitting the dataset into training, validation, and test sets to assess model performance and prevent overfitting. The data is typically divided into three subsets: training set, used to train the model validation set, used to tune hyperparameters and evaluate model performance during training and test set, used to assess the final model's performance on unseen data. The split ratio is 80% for training, 10% for validation, and 10% for testing. This ensures that an adequate amount of data is allocated for model training while still allowing for robust evaluation. By employing this splitting strategy, we can develop a model that generalizes well to unseen data and produces reliable predictions in real-world scenarios.

11) Spliting the dataset into training, testing and validation sets

```
]: # Split the dataset into training (80%) and testing + validation (20%)
   X_train, X_temp, y_train, y_temp = train_test_split(features, target, test_size=0.2, random_state=42)

   # Further split the temporary dataset into testing (50%) and validation (50%)
   X_test, X_validation, y_test, y_validation = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

Fig 22 : Dataset split into train, test and validation sets

- **Shape of the resulting dataset:** The resulting dataset comprises the shapes of the training set (X_train, y_train), test set (X_test, y_test), and validation set (X_val, y_val).

12) shape of X_train, y_train, X_test, y_test, X_validation, y_validation

```
[64]: # Print the shapes of the resulting datasets
print("Shape of X_train:", X_train.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_test:", y_test.shape)
print("Shape of X_validation:", X_validation.shape)
print("Shape of y_validation:", y_validation.shape)
```

```
Shape of X_train: (85352, 56)
Shape of y_train: (85352,)
Shape of X_test: (10669, 56)
Shape of y_test: (10669,)
Shape of X_validation: (10670, 56)
Shape of y_validation: (10670,)
```

Fig 23 : shape of X_train, y_train, X_test, y_test, X_validation, y_validation

- **Data Cleaning:** cleaning the data has been performed by removing any irrelevant columns, handling outliers, and ensuring that the data types were appropriate for analysis.

Data Visualization:

Plot 1: The subplot displays histogram plots for each numerical column in the DataFrame df_cleaned, presenting the distribution and counts of values.

Analysis: Facility ID, Heat Input and Hour exhibit higher counts compared to other features, as they have significant relevance and frequency within the dataset. The distributions for Facility ID and Hour appear well-distributed without significant skewness, indicating a uniform occurrence across different values. On the other hand features like Operating System, SO2 mass, SO2 rate, NOx mass, and NOx rate exhibit relatively lower counts and less variation. This suggests that these variables may have more concentrated values or less diversity within the dataset. Overall, the histograms provide valuable insights into the distributional characteristics of numerical features, highlighting both prominent and less varying variables within the dataset."

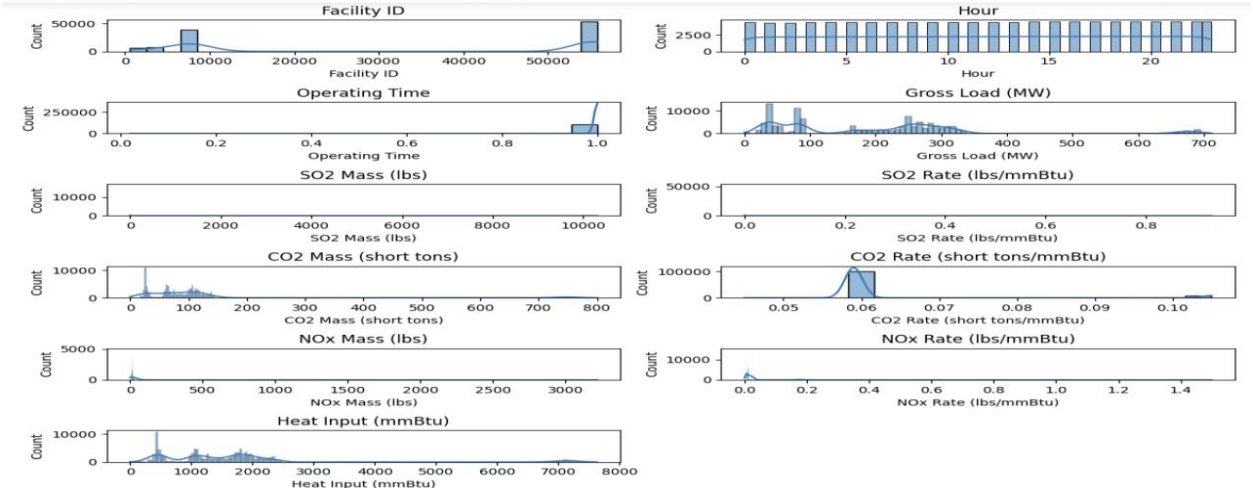


Fig 24 : plotting histplot for all features with numerical values

Plot 2: This box plot compares the NOx rate distributions across different primary fuel types, namely natural gas, coal, and pipeline natural gas.

Analysis: Natural gas exhibits the widest range of NOx rates, spanning from 0 to higher values. This indicates significant variability in NOx emissions associated with natural gas usage, potentially influenced by factors such as combustion conditions or fuel composition. The distribution of NOx rates for natural gas shows a relatively uniform spread of values, suggesting a diverse range of emission levels across different facilities or operating conditions utilizing this primary fuel type. In contrast, coal shows a narrower range of NOx rates, with values typically starting later and ending earlier on the plot. This suggests a more constrained range of NOx emissions associated with coal combustion compared to natural gas. Pipeline natural gas falls between natural gas and coal in terms of NOx rate distribution. The values tend to start earlier than coal but conclude around 0.4, indicating a moderate variability in NOx emissions relative to the other fuel types. These findings highlight the distinct NOx emission characteristics associated with different primary fuel types, with natural gas exhibiting the most variability and coal showing relatively constrained emissions. Understanding these patterns is crucial for assessing the environmental impact and regulatory compliance of facilities utilizing various fuel sources."

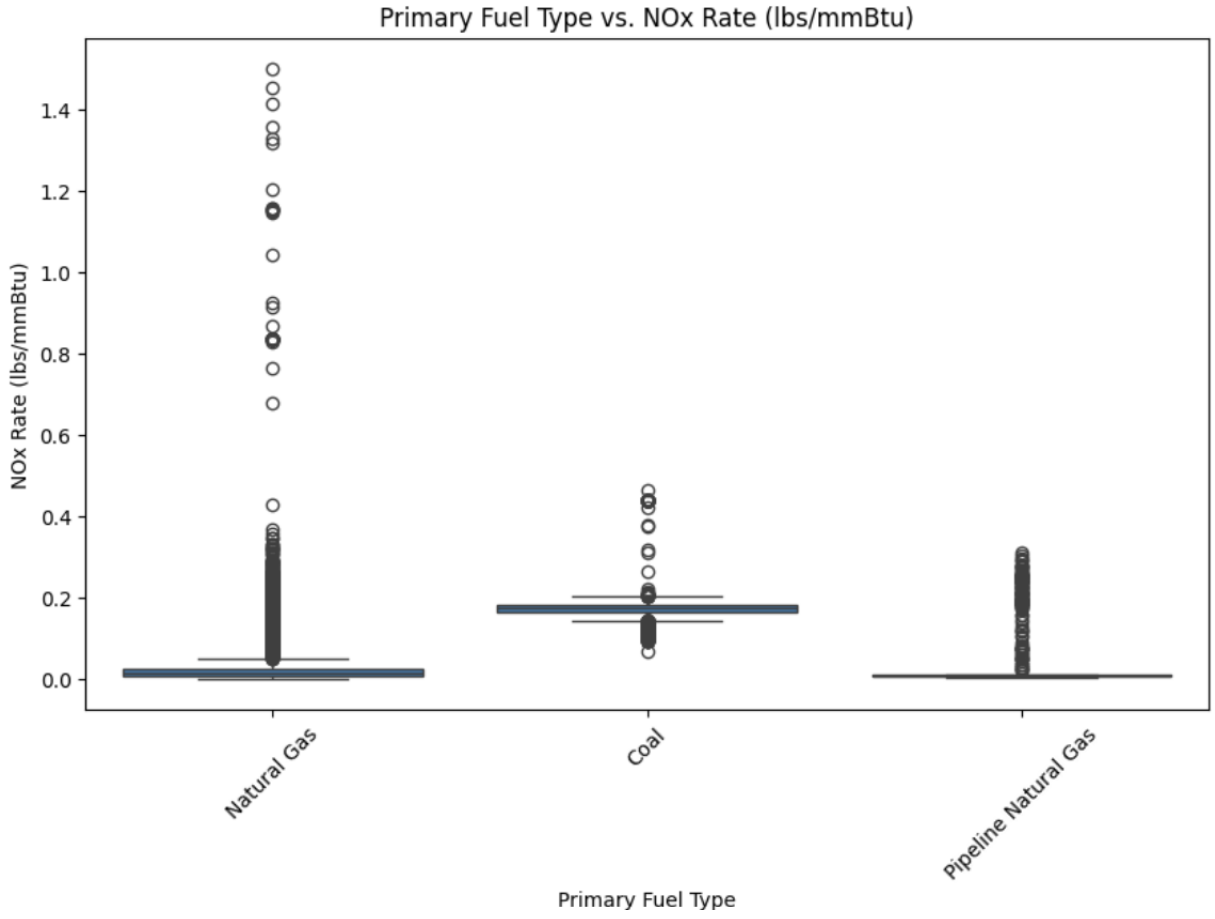


Fig 25 : Box plot for Primary Fuel Type vs NOx Rate

Plot 3: This scatter plot illustrates the relationship between Gross load and NOx rate

Analysis : The majority of data points appear clustered in the lower range of both Gross load and NOx rate, particularly between 0 and 0.4. This concentration suggests a prevalent pattern where facilities with lower Gross load tend to exhibit lower NOx emissions. As Gross load increases beyond 0.4, the density of data points decreases, indicating fewer instances of higher NOx rates at higher Gross load levels. This pattern suggests a potential inverse relationship between Gross load and NOx emissions, with higher operational loads correlating with reduced NOx emissions per unit of load. However, it's essential to note the presence of scattered data points in the upper range of both variables, indicating exceptions to the general trend. These outliers could represent facilities with unique operational characteristics or external factors influencing NOx emissions independently of Gross load. Overall, the scatter plot provides valuable insights into the relationship between Gross load and NOx rate, highlighting the prevalence of lower emissions at lower load levels and the potential for variability in NOx emissions under specific operational conditions or facility characteristics.

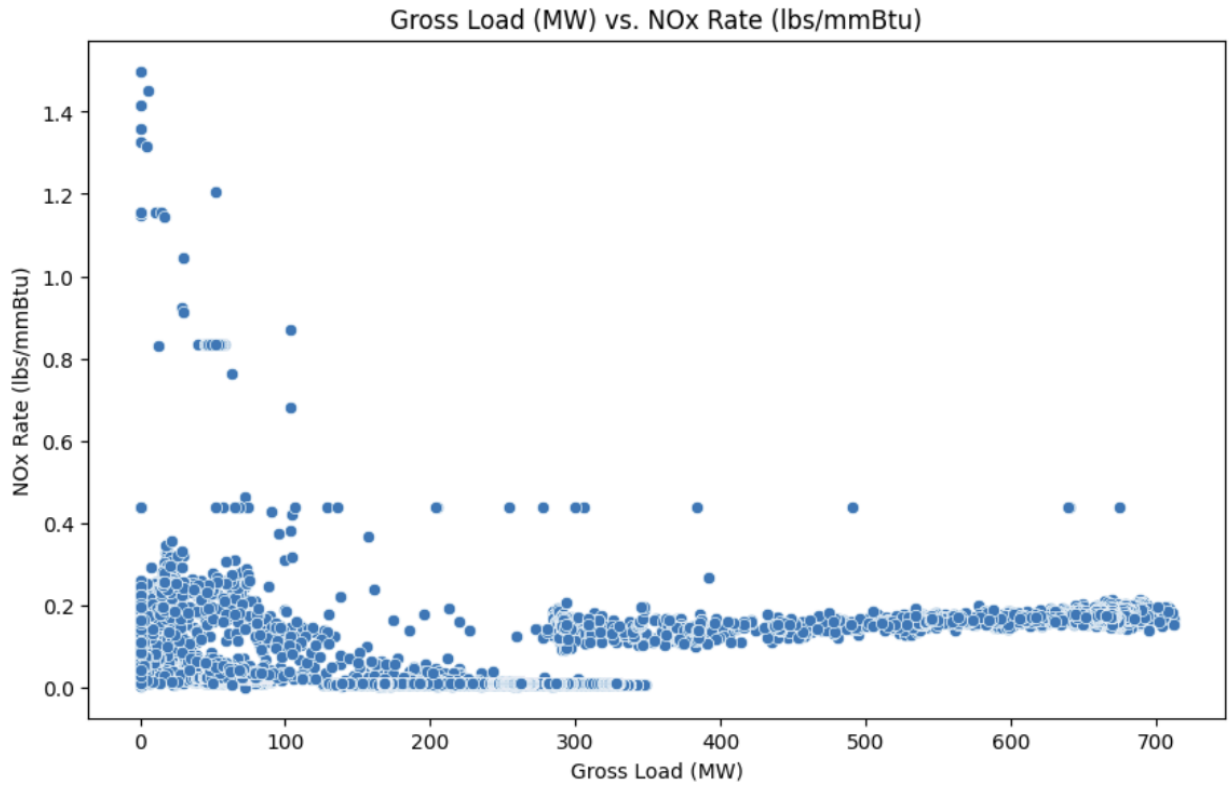


Fig 26 : Scatter Plot (Gross Load vs. NOx Rate)

Plot 4: This line plot depicts the temporal variation of NOx rate over a one-year period, from January 2023 to January 2024.

Analysis: The plot reveals fluctuations in NOx rate over time, with the data points scattered across the x-axis, indicating variability in NOx emissions throughout the year.

A distinct pattern emerges from the data, notably a peak in NOx emissions occurring between May and July 2023. This period of elevated NOx rates suggests a potential seasonal trend or specific environmental conditions contributing to increased emissions during these months. The temporal distribution of NOx rate underscores the importance of monitoring and analyzing emissions trends over time to identify periods of heightened environmental impact or regulatory concern. While the plot provides valuable insights into the temporal dynamics of NOx emissions, further investigation is warranted to understand the underlying factors driving the observed patterns. Factors such as seasonal variations in industrial activity, weather conditions, or regulatory changes may influence NOx emissions and contribute to the observed fluctuations. Overall, the line plot serves as a valuable tool for assessing the temporal trends in NOx emissions and identifying potential periods of heightened environmental risk or regulatory compliance challenges.

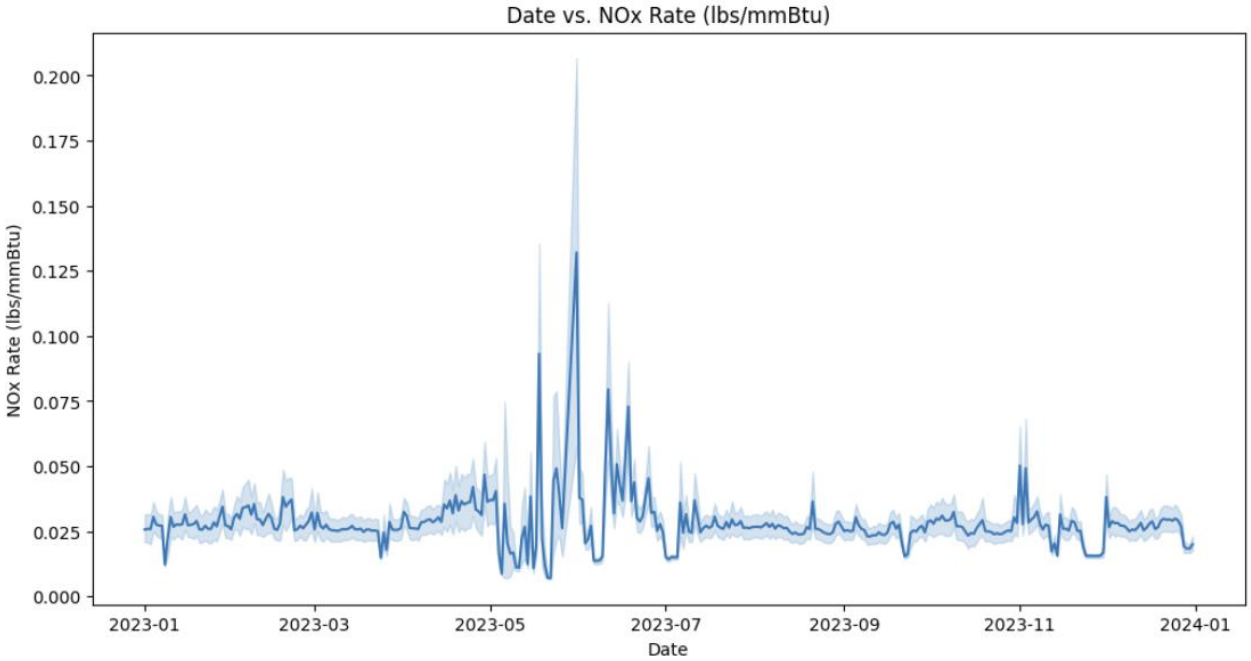


Fig 27 : Line Plot of Date vs. NOx Rate

Plot 5 : This scatter plot illustrates the relationship between heat input and CO2 mass

Analysis: The plot spans the range of heat input from 0 to 8000 and CO2 mass from 0 to 800, showcasing the full spectrum of data points within these ranges. A clear linear trend is discernible in the data, indicating a proportional relationship between heat input and CO2 mass. As heat input increases, there is a corresponding increase in CO2 mass, suggesting a direct impact of heat input on CO2 emissions. The linearity of the relationship implies that CO2 emissions scale linearly with heat input, reflecting the fundamental combustion process where the amount of CO2 generated is directly influenced by the amount of fuel burned. Notably, the scatter plot allows for the identification of outliers or deviations from the general trend, which could signify anomalies in the combustion process or the presence of external factors affecting CO2 emissions. Understanding the relationship between heat input and CO2 mass is crucial for assessing the environmental impact of energy generation processes and informing strategies for emissions reduction and efficiency improvement. Overall, the scatter plot provides valuable insights into the dependency of CO2

emissions on heat input, highlighting the importance of monitoring and optimizing combustion processes to mitigate environmental impact.

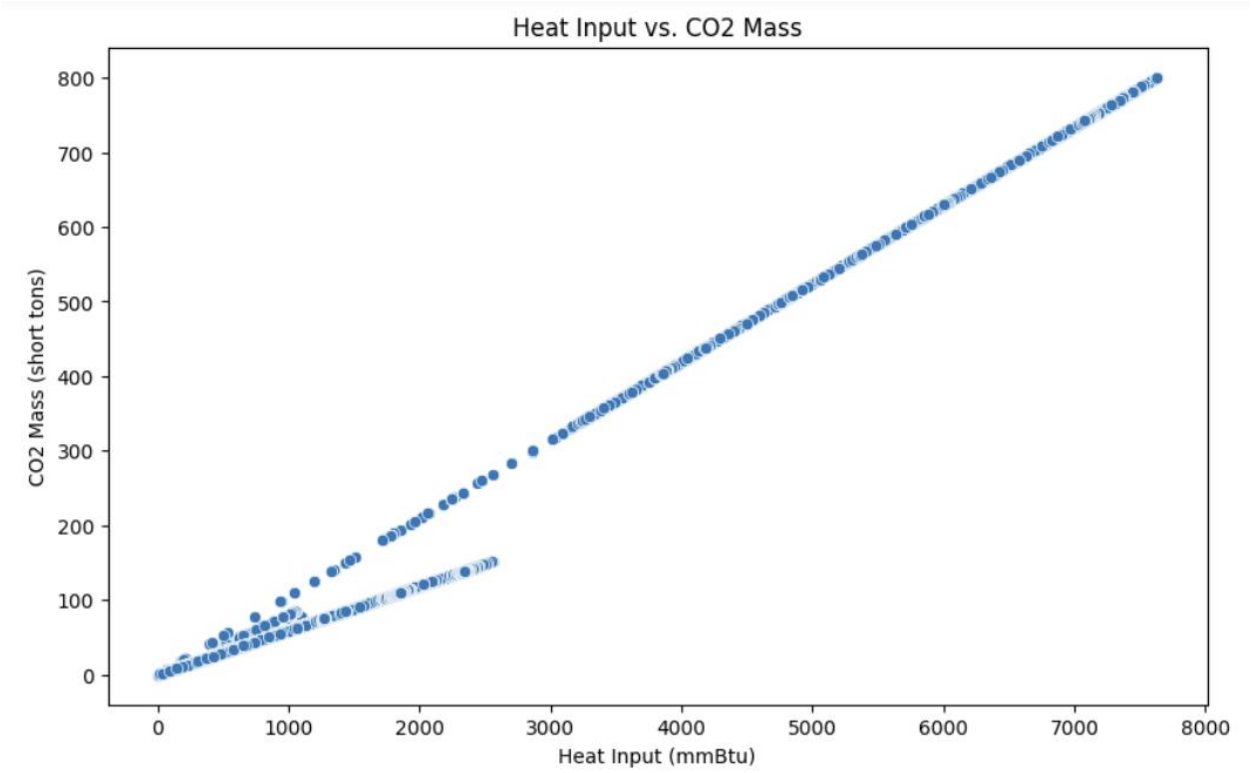


Fig 28 : Scatter Plot of Heat Input vs. CO2 Mass

Plot 6 : This violin plot compares the distribution of NOx rates across different unit types, including combustion turbine, tangentially fired, and combined cycle.

Analysis: Each violin represents the distribution of NOx rates for a specific unit type. The width of the violin corresponds to the density of data points, with wider sections indicating higher frequency of NOx rates within that range. The plot reveals distinct patterns in NOx rate distributions for each unit type. Notably, there is considerable variation in NOx rates among the unit types, with combined cycle exhibiting the widest spread of NOx rates. Combustion turbines show a moderate range of NOx rates, indicating some variability but less pronounced than combined cycle. This suggests differences in combustion technologies or operational parameters between combustion turbines and combined cycle units. Tangentially fired units display the least variation in NOx rates among the three unit types, with a narrower range of values concentrated around lower NOx rates. This could reflect the efficiency of tangentially fired combustion systems in controlling emissions compared to other technologies. The range of NOx rates spans from 0 to 1.4 across all unit types, indicating potential regulatory compliance considerations and environmental impact associated with NOx emissions from these facilities.

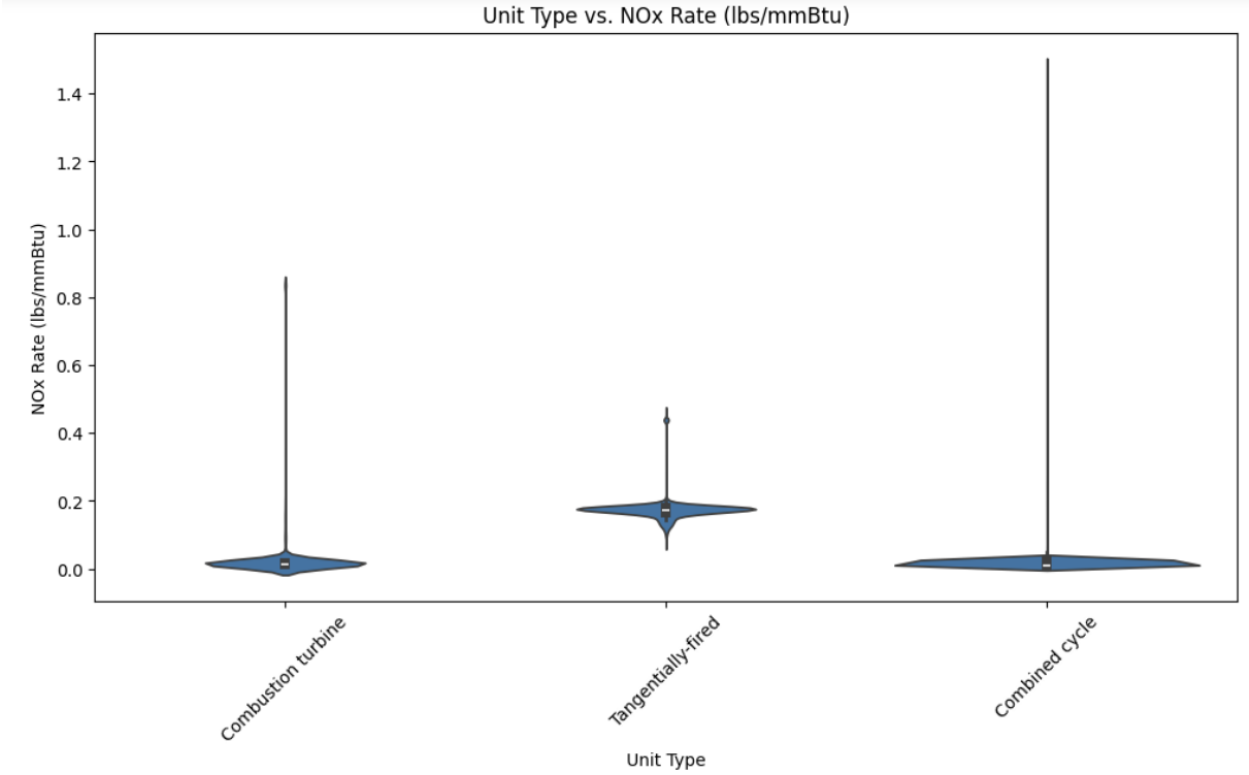


Fig 29 : Violin Plot of Unit Type vs. NOx Rate

Plot 7 : This correlation matrix provides a comprehensive overview of the relationships between numerical features within the dataset.

Analysis:

Positive correlations: Gross load, SO2 mass, SO2 rate, CO2 mass, CO2 rate, NOx rate, NOx mass, and heat input exhibit positive correlations with the target variable. A positive correlation implies that as these numerical features increase, the target variable (presumably emissions or some related metric) also tends to increase. For example, higher values of Gross load, which represents the total electrical load on a power system, are associated with increased emissions, as more energy generation typically leads to higher emissions.
Similarly, higher values of SO2 mass, SO2 rate, CO2 mass, CO2 rate, NOx rate, NOx mass, and heat input are linked to increased emissions of sulfur dioxide (SO2), carbon dioxide (CO2), nitrogen oxides (NOx), and heat generation, respectively.

Negative correlations:Facility ID and Hour exhibit negative correlations with the target variable.
A negative correlation suggests that as the values of these features increase, the target variable tends to decrease, and vice versa. In this context, Facility ID and Hour may represent categorical or temporal variables that influence emissions in an inverse manner.
For instance, Facility ID could represent different power plants, where certain plants may have more advanced emission control technologies or operate under stricter environmental regulations,

leading to lower emissions. Hour could represent the time of day, with lower emissions observed during off-peak hours when energy demand is reduced.

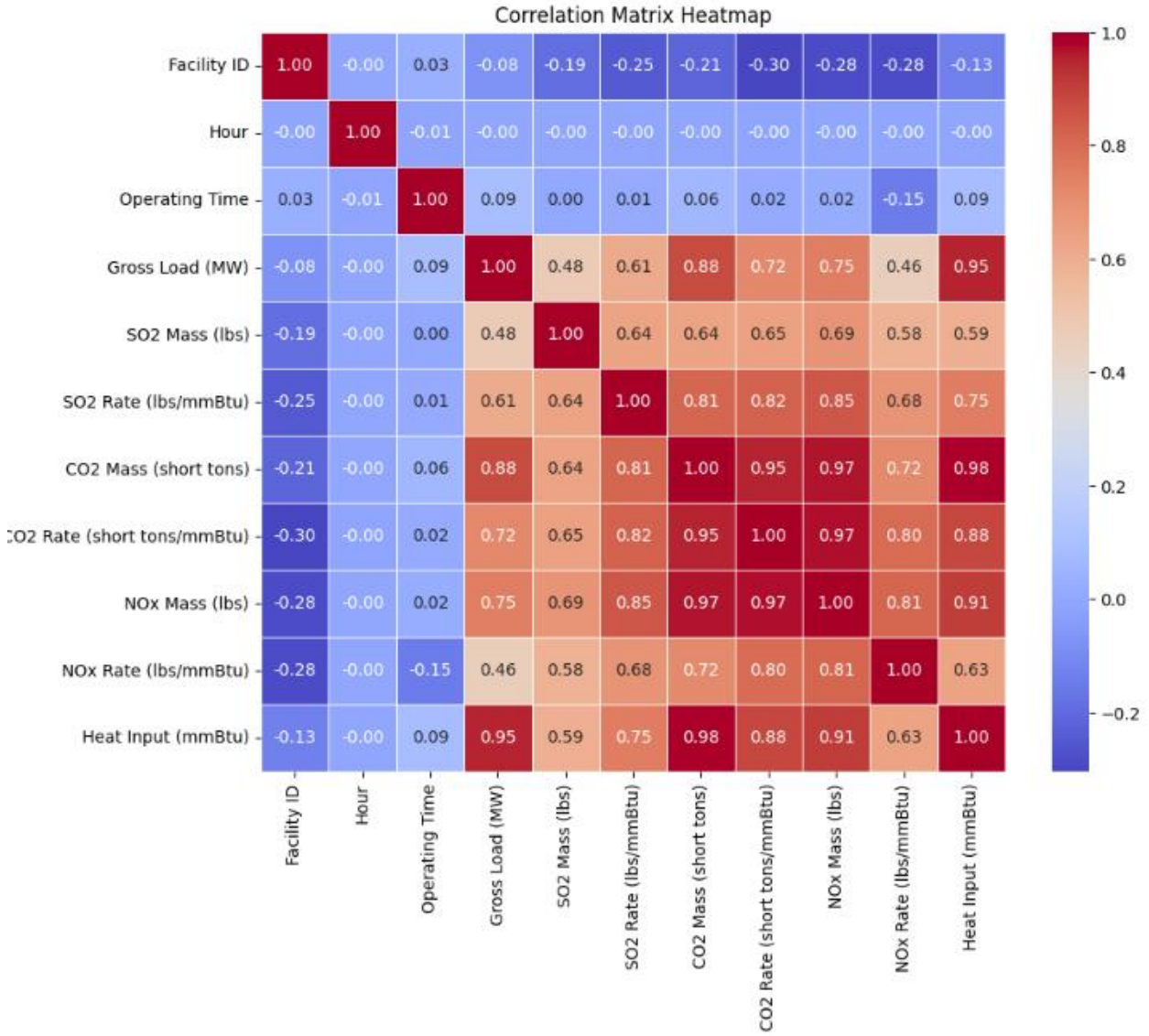


Fig 30 : Correlation matrix for the numerical columns using heatmap

Machine Learning Models

1)KNeighborsRegressor:

The K-Nearest Neighbors (KNN) algorithm is a simple, non-parametric algorithm used for both classification and regression tasks. In the case of regression, KNN predicts the target variable by averaging the values of its k nearest neighbors.

Mathematical Representation of KNN Regression:
Distance Calculation: Calculate the distance between the new data point and each data point in the training set using a distance metric (e.g., Euclidean distance, Manhattan distance, etc.).

Selecting Neighbors: Select the k nearest neighbors to the new data point based on the calculated distances.

Prediction: For regression, predict the target variable of the new data point by averaging the target values of its k nearest neighbors.

How KNN Regression Works:
Distance Calculation: KNN regression begins by calculating the distance between the new data point and each data point in the training set. This is typically done using a distance metric such as Euclidean distance or Manhattan distance.

$$d2 = (x_2 - x_1)^2 + (y_2 - y_1)^2$$

Selecting Neighbors: The algorithm then selects the k nearest neighbors to the new data point based on the calculated distances. These neighbors are the data points with the smallest distances to the new data point.

Prediction: For regression, the algorithm predicts the target variable of the new data point by averaging the target values of its k nearest neighbors. The predicted value is the average of the target values of the k nearest neighbors.

Key Features of KNN Regression:
Simple: KNN regression is easy to understand and implement, making it suitable for beginners.
Non-parametric: KNN regression makes no assumptions about the underlying data distribution, making it suitable for datasets with complex relationships.
Flexibility: KNN regression can be used for both regression and classification tasks.
Localized Learning: KNN regression focuses on local patterns in the data, which can capture complex relationships that global models might miss.
Advantages of KNN Regression:
KNN regression makes no assumptions about the data, making it robust to outliers and non-linear relationships. KNN regression produces interpretable results that can be easily understood by stakeholders. KNN regression can be applied to a wide range of regression problems with different types of data.

Disadvantages of KNN Regression:
KNN regression can be computationally expensive, especially for large datasets or high-dimensional data. KNN regression is sensitive to noise and outliers in the data, which can lead to

inaccurate predictions. KNN regression performance can degrade in high-dimensional spaces due to the curse of dimensionality.

2)Linear Regression:

By fitting a linear equation to observed data, linear regression is a statistical technique used to model the connection between one or more independent variables (features) and a dependent variable (goal). It makes the assumption that the independent and dependent variables have a linear relationship.

The relationship between x and y in a simple linear regression with one independent variable (y) and one dependent variable (x) is represented by the following equation of a straight line:

$$y = \beta_0 + \beta_1 + \epsilon$$

The dependent variable is called y, the independent variable is called x, the line's intercept is called the y-intercept, the slope of the line is called $\beta 0$ is the change in y for a unit change in x, and the error term, or residuals, is called noise ϵ.

How Linear Regression Operates:

Model Training: By estimating the coefficients (β), linear regression fits a straight line to the observed data points that reduce the sum of squared residuals (the vertical gap between the observed and projected values of y).
Prediction: By changing the values of the independent variables (x) in the linear equation, the trained model may be used to predict the target variable (y) for new data points.

Advantages of Linear Regression: The coefficients produced by linear regression are simple to understand and show how each independent variable affects the dependent variable. This facilitates stakeholders' understanding and explanation of the links between the variables. The algorithm of linear regression is simple and easy to understand, making it appropriate for novices and those who are not experts in statistics to learn about. Large datasets can be handled with comparatively less processing resources thanks to linear regression's high computational efficiency. When working with datasets that have a lot of features, it is really helpful. Simple linear regression (one independent variable), multiple linear regression (many independent variables), and polynomial regression (non-linear relationships) are only a few of the data and relationship types that can be handled by linear regression.

Disadvantages of Linear Regression: The independent and dependent variables in a linear regression are assumed to be related. Predictions using linear regression may be skewed or wrong if the relationship is non-linear. Data outliers can disproportionately affect the calculated coefficients and produce biased predictions in linear regression. Performing preprocessing on the data and reducing or eliminating the impact of anomalies is crucial prior to constructing a linear regression model. Linear regression makes the assumption that the variance of the residuals, or the discrepancy between the dependent variable's observed and projected values.

3)Ridge Regression:

Ridge regression is a kind of linear regression where regularization is used to deal with dataset multicollinearity and overfitting. Large coefficients are penalized by adding a penalty term to the ordinary least squares (OLS) cost function. A regularization parameter governs this penalty term (λ), known as the shrinkage parameter.

Mathematical Representation: A penalty term is added by edge regression to the ordinary least squares (OLS) cost function.

$$\text{Cost}(w) = \sum i = 1n\,(y\,i - y\,\hat{}\,i\,)2 + \lambda \sum j = 1\; p\; w\; j\; 2$$

The cost function that needs to be minimized is denoted by Cost(w), the observed value of the dependent variable for the ith data point is denoted by yi. The number of features (independent variables) is p, the regularization parameter is λ λ, the coefficients of the linear regression model are w j, and the projected value of the dependent variable for the ith data point is i.

How Ridge Regression Works: Ridge regression penalizes large coefficients by adding a penalty term to the OLS cost function. By encouraging the model to discover coefficients with smaller magnitudes, this penalty term helps to prevent overfitting and effectively lower the variance of the model. Ridge regression uses closed-form solutions or gradient descent as optimization approaches to minimize the updated cost function. The strength of regularization is determined by the regularization parameter (λ), where bigger values of λ result in stronger regularization and smaller coefficients. By using the learnt coefficients to the features of the new data points, the Ridge regression model may be trained and then utilized to produce predictions on fresh data points.

Important Ridge Regression Features:L2 regularization, which penalizes high coefficients and lessens overfitting, is used into ridge regression. Ridge regression reduces the coefficients of correlated characteristics toward one another, making it resistant to multicollinearity. Ridge regression enhances the model's generalization performance on unknown data by penalizing big coefficients, hence preventing overfitting.

Advantages: Reduces the tendency for overfitting by reducing the coefficients to zero.successfully manages multicollinearity by stabilizing the coefficients. predicts more accurately and robustly than standard least squares regression.
Disadvantages: Causes the model to become biased by making the coefficients smaller than zero. needs to have the regularization parameter (λ) adjusted in order to balance variance and bias. Compared to standard least squares regression, it could be harder to interpret because

4)Bayesian Regression:

A probabilistic method for regression modeling that enables the estimation of prediction uncertainty is called Bayesian regression. In contrast to conventional regression techniques that yield point estimates for the model parameters, Bayesian regression yields posterior distributions over the parameters that encompass the estimates' mean as well as their associated uncertainty.

Mathematical Representation: The model parameters in Bayesian regression are seen as random variables with prior distributions, which stand in for our assumptions about the parameters prior to data observation. Given the model parameters, the likelihood function shows the likelihood of observing the data. We generate the posterior distribution over the parameters, which reflects our revised beliefs following data observation, by combining the prior and likelihood using the Bayes theorem.

$$y_i = \alpha + \beta x_i + \epsilon_i$$

How Bayesian Regression Works: The first step in using Bayesian regression is to define prior distributions across the model parameters. Our preconceived notions about the parameters before seeing the data are encoded by these priors. Given the model parameters, the likelihood function shows the likelihood of observing the data. It measures how well the observed data are explained by the model. The posterior distribution over the parameters is obtained by combining the prior and likelihood using the Bayes theorem. Our revised opinions about the parameters following the observation of the data are represented by this posterior distribution. Bayesian regression yields posterior distributions over the model parameters that encompass both the estimates' mean and uncertainty. This makes it possible to estimate the uncertainty of forecasts, which is very helpful in decision making

Key Features of Bayesian Regression: By offering posterior distributions over the model parameters, Bayesian regression enables the estimation of prediction uncertainty. Bayesian regression enables the analysis to take into account preexisting beliefs or knowledge regarding the model parameters. Bayesian regression is appropriate for a broad range of regression issues since it can handle different kinds of data and relationships.

Advantages of Bayesian Regression: One of the benefits of using Bayesian Regression is that it may give estimates of prediction uncertainty, which is useful when making decisions in the face of uncertainty. permits past information or opinions to be incorporated into the analysis and adaptability in managing various data kinds and interactions.

Disadvantages : The drawbacks of using Bayesian Regression computationally demanding: The computation of posterior distributions over the parameters in Bayesian regression can be computationally costly, particularly for big datasets or intricate models. Prerequisite distributions must be specified: The results of Bayesian regression may be affected by the subjective specification of prior distributions over the model parameters.

5) Polynomial Regression:

Regression analysis that represents the relationship between the independent variable (predictor) and the dependent variable (target) as an nth-degree polynomial is known as polynomial regression. By fitting a curve to the data, polynomial regression can capture non-linear correlations, in contrast to linear regression, which presumes a linear relationship between the variables.

Mathematical Representation: An nth-degree polynomial equation represents the relationship between the independent variable (x) and the dependent variable (y) in polynomial regression.

$$y=\beta 0+\beta 1 x+\beta 2 x 2+\ldots+\beta n x n+\epsilon$$

Where:

The dependent variable y is the target and the independent variable x is the predictor the polynomial's degree is n the coefficients of the polynomial terms are βn β 0 , βn^2 1 ,…,β n and the error term ϵ is the residuals signifying the discrepancy between the observed and predicted values of y.

Working of Polynomial Regression: Model Fitting By estimating the coefficients (β 0 ,β 1 ,…,βn β 0,β 1,…,β n) that minimize the sum of squared residuals (the vertical distance between the observed and anticipated values of y), polynomial regression fits a polynomial curve to the data. The intricacy of the curve is based on the degree of the polynomial (n). Although they are more likely to overfit, higher-degree polynomials are able to capture more intricate relationships. The degree of the polynomial is usually chosen using methods for evaluating models, including cross-validation.

Key Features of Polynomial Regression: By fitting a curve to the data, polynomial regression can capture nonlinear correlations between the variables.A wide range of regression issues with various data kinds and relationships can be solved using polynomial regression. The polynomial term coefficients shed light on the nature and direction of the variable-to-variable interaction.

Advantages: Polynomial regression is appropriate for datasets with non-linear patterns because it may capture nonlinear correlations between the variables. The modeling of intricate interactions between the variables is made possible by the versatility and flexibility of polynomial regression. Interpretable information about the relationship between the variables can be obtained from the coefficients of the polynomial terms**.**

Disadvantages: Higher degree polynomials have a higher tendency to overfit, particularly when their degree is excessively high in comparison to the size of the dataset. The model's complexity rises with higher-degree polynomials, making it harder to understand and more prone to overfitting. Polynomial regression exhibits sensitivity to outliers in the data, potentially producing biased predictions due to their disproportionate influence on the fitted curve.

Mean Squared Error Loss and R2 Score for all Models

Mean Squared ErrorLoss Value: The loss value represents how well the model is performing during training. It typically measures the discrepancy between the actual target values and the predicted values generated by the model. A lower loss value indicates better performance.

Accuracy: Accuracy measures the overall correctness of the model's predictions. It represents the proportion of correct predictions out of the total number of predictions made by the model. Higher accuracy values indicate better performance.

For KNeighborsRegressor: The Mean squared error loss and R2 score for KNN model

```
# Evaluation metrics
model_knn_mse_test = mean_squared_error(y_test, model_knn_predict_test) #for test data
print(f"MSE for test data : {model_knn_mse_test}")
```

```
MSE for test data : 0.09356287098496685
```

```
model_knn_r2_score_test = r2_score(y_test, model_knn_predict_test) #r2 score for test data
print(f"R2 score for test data : {model_knn_r2_score_test}")
```

```
R2 score for test data : 0.793985234716573
```

```
# Evaluation metrics
model_knn_mse_validation = mean_squared_error(y_validation, model_knn_predict_validation) #for validation set
print(f"MSE for validation data : {model_knn_mse_validation}")
```

```
MSE for validation data : 0.11244569357597431
```

```
model_knn_r2_score_validation = r2_score(y_validation, model_knn_predict_validation) #r2 score for validation data
print(f"R2 score for validation data : {model_knn_r2_score_validation}")
```

```
R2 score for validation data : 0.7490578725682342
```

Fig 31 : MSE and R2 score for KNN

For LinearRegression: The Mean squared error loss and R2 score for linear regression model

```
# evaluation metrics for test set
model_linear_mse_test = mean_squared_error(y_test, model_linear_predict_test)
print(f"MSE for test data : {model_linear_mse_test}")
```

```
MSE for test data : 0.00020831520116174716
```

```
model_linear_r2_score_test = r2_score(y_test, model_linear_predict_test) #Using R2 score
print(f"R2 score for test data : {model_linear_r2_score_test}")
```

```
R2 score for test data : 0.8061745912871335
```

```
model_linear_mse_validation = mean_squared_error(y_validation, model_linear_predict_validation) #For validation dataset
print(f"MSE for validation data : {model_linear_mse_validation}")
```

```
MSE for validation data : 0.00018564650573977485
```

```
model_linear_r2_score_validation = r2_score(y_validation, model_linear_predict_validation)
print(f"R2 score for validation data : {model_linear_r2_score_validation}")
```

```
R2 score for validation data : 0.8242146066095026
```

Fig 32 : MSE and R2 score for Linear

For Ridge Regression: The Mean squared error loss and R2 score for ridge regression model

```
# evaluation metrics for test set
model_ridge_mse_test = mean_squared_error(y_test, model_ridge_predict_test)
print(f"MSE for test data : {model_ridge_mse_test}")
```

```
MSE for test data : 0.00020119922345103453
```

```
model_ridge_r2_score_test = r2_score(y_test, model_ridge_predict_test) #Using R2 score
print(f"R2 score for test data : {model_ridge_r2_score_test}")
```

```
R2 score for test data : 0.812795602526249
```

```
model_ridge_mse_validation = mean_squared_error(y_validation, model_ridge_predict_validation) #For validation dataset
print(f"MSE for validation data : {model_ridge_mse_validation}")
```

```
MSE for validation data : 0.00019557467840432483
```

```
model_ridge_r2_score_validation = r2_score(y_validation, model_ridge_predict_validation)
print(f"R2 score for validation data : {model_ridge_r2_score_validation}")
```

```
R2 score for validation data : 0.8148137954790575
```

Fig 33 : MSE and R2 score for Ridge

For Bayesian Regression: The Mean squared error loss and R2 score for bayesianregression model

```
# evaluation metrics for test set
model_bayesian_mse_test = mean_squared_error(y_test, model_bayesian_predict_test)
print(f"MSE for test data : {model_bayesian_mse_test}")
```

```
MSE for test data : 0.00020830455014692726
```

```
model_bayesian_r2_score_test = r2_score(y_test, model_bayesian_predict_test) #Using R2 score
print(f"R2 score for test data : {model_bayesian_r2_score_test}")
```

```
R2 score for test data : 0.806184501448702
```

```
model_bayesian_mse_validation = mean_squared_error(y_validation, model_bayesian_predict_validation) #For validation dataset
print(f"MSE for validation data : {model_bayesian_mse_validation}")
```

```
MSE for validation data : 0.00018567462363780097
```

```
model_bayesian_r2_score_validation = r2_score(y_validation, model_bayesian_predict_validation)
print(f"R2 score for validation data : {model_bayesian_r2_score_validation}")
```

```
R2 score for validation data : 0.8241879822690867
```

Fig 34 : MSE and R2 score for Bayesian

For Polynomial Features forRegression: The Mean squared error loss and R2 score for polynomial regression model

```
# evaluation metrics for test set
model_poly_mse_test = mean_squared_error(y_test, model_poly_predict_test)
print(f"MSE for test data : {model_poly_mse_test}")
```

```
MSE for test data : 0.00020847184349523745
```

```
model_poly_r2_score_test = r2_score(y_test, model_poly_predict_test) #Using R2 score
print(f"R2 score for test data : {model_poly_r2_score_test}")
```

```
R2 score for test data : 0.8060288445334585
```

```
model_poly_mse_validation = mean_squared_error(y_validation, model_poly_predict_validation) #For validation dataset
print(f"MSE for validation data : {model_poly_mse_validation}")
```

```
MSE for validation data : 0.00018606123175613363
```

```
model_poly_r2_score_validation = r2_score(y_validation, model_poly_predict_validation)
print(f"R2 score for validation data : {model_poly_r2_score_validation}")
```

```
R2 score for validation data : 0.8238219098784524
```

Fig 35 : MSE and R2 score for Polynomial

Plots For Prediction vs actual test data

1) Plot for KNN regressor : plotting scatter plot for Actual vs Predicted Target for KNN regressor

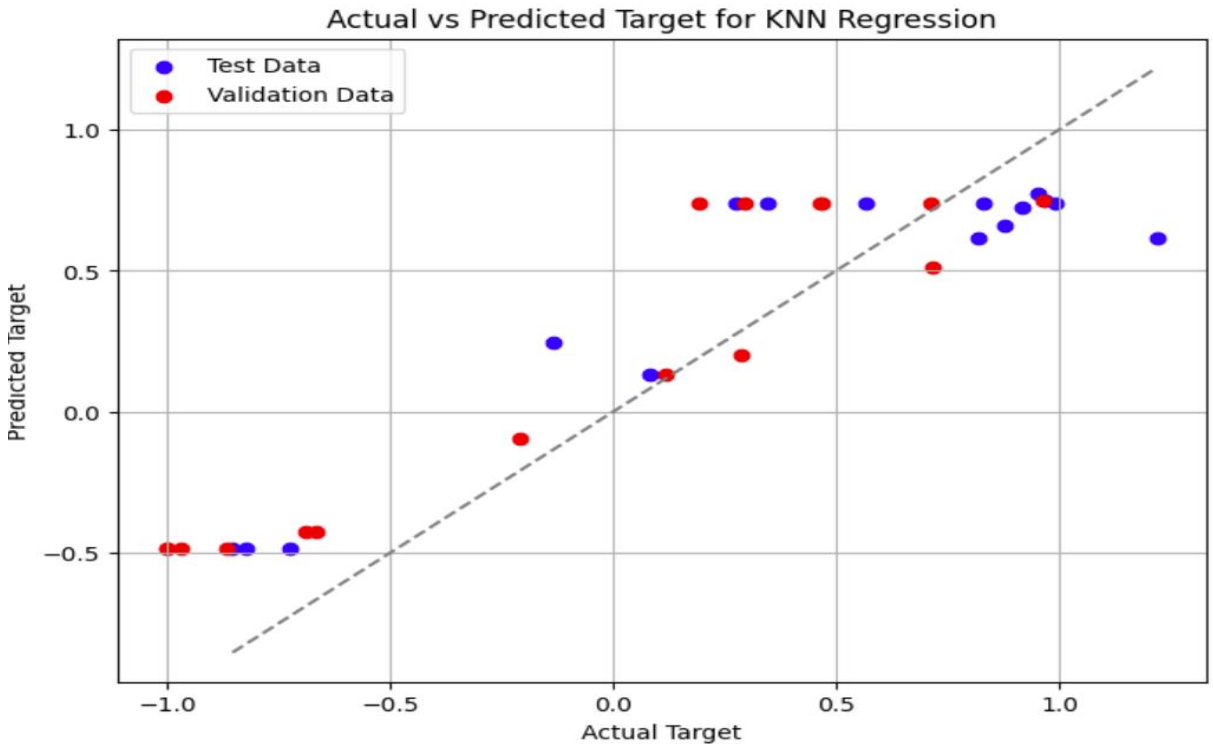


Fig 36 : Scatter Plot for Actual Vs Predicted Target for KNN

2) Plot for Linear Regression : plotting scatter plot for Actual vs Predicted Target for KNN regressor

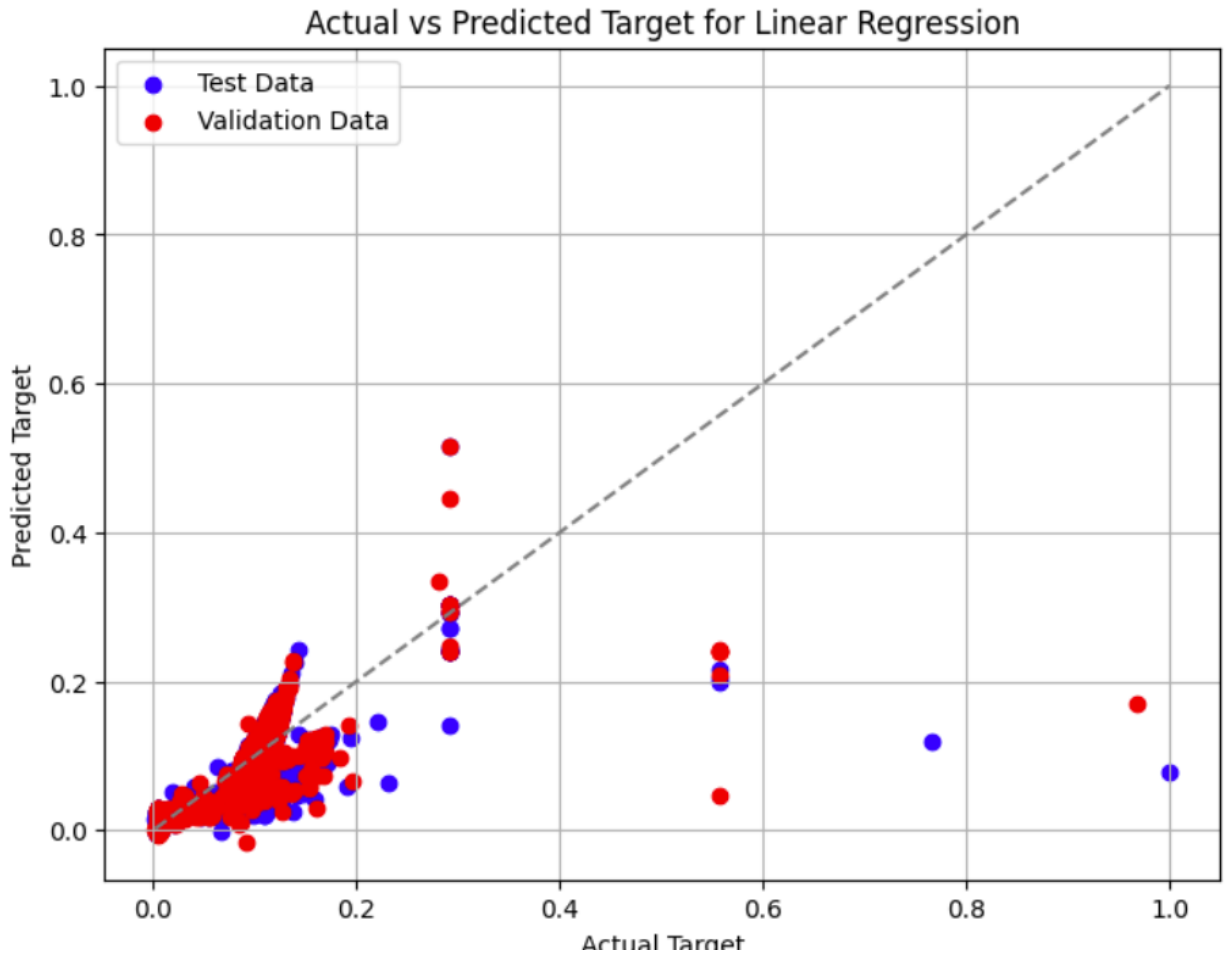


Fig 37 : Scatter Plot for Actual Vs Predicted Target for Linear

3) Plot for Ridge Regression : plotting scatter plot for Actual vs Predicted Target for Ridge regression

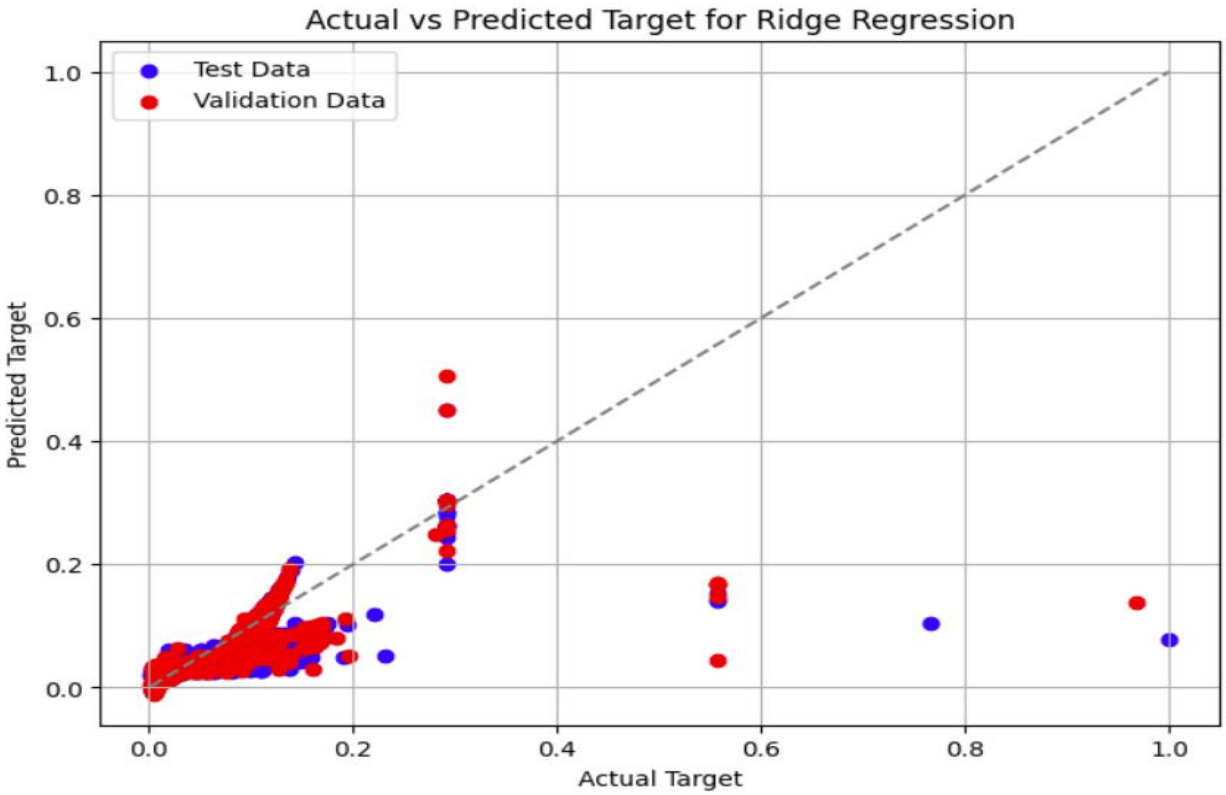


Fig 38 : Scatter Plot for Actual Vs Predicted Target for Ridge

4) Plot for Bayesian Regression : plotting scatter plot for Actual vs Predicted Target for Bayesian regression

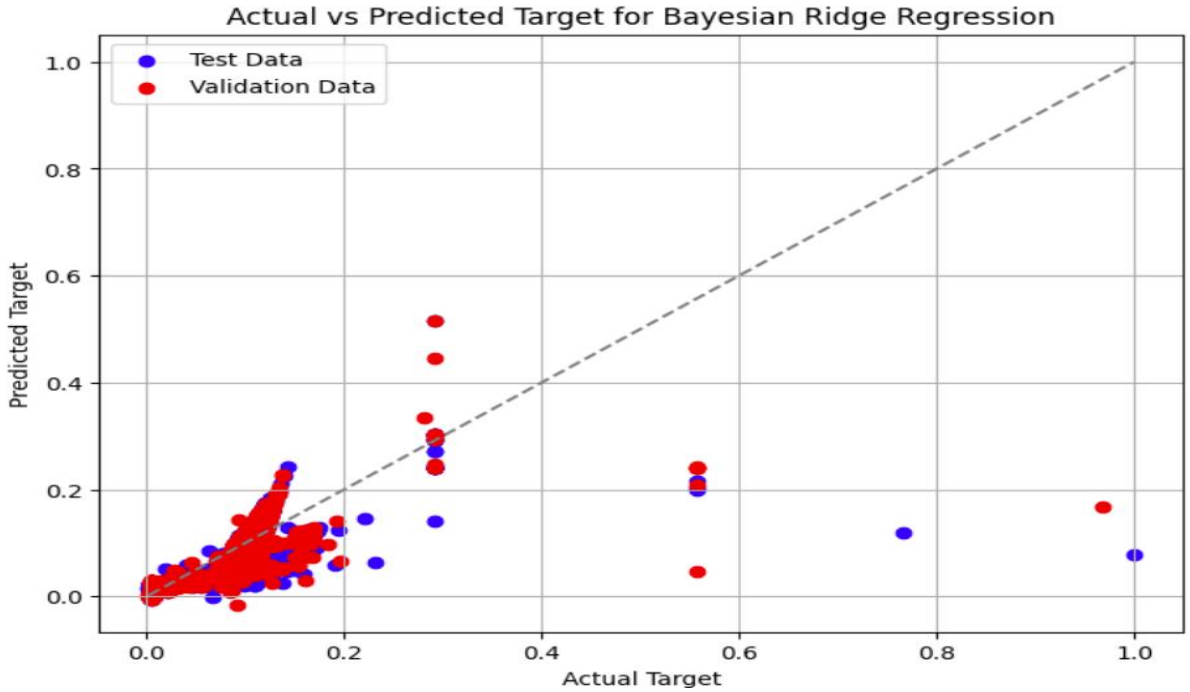


Fig 39 : Scatter Plot for Actual Vs Predicted Target for Bayesian

5) Plot for Polynomial Regression : plotting scatter plot for Actual vs Predicted Target for Polynomial regression

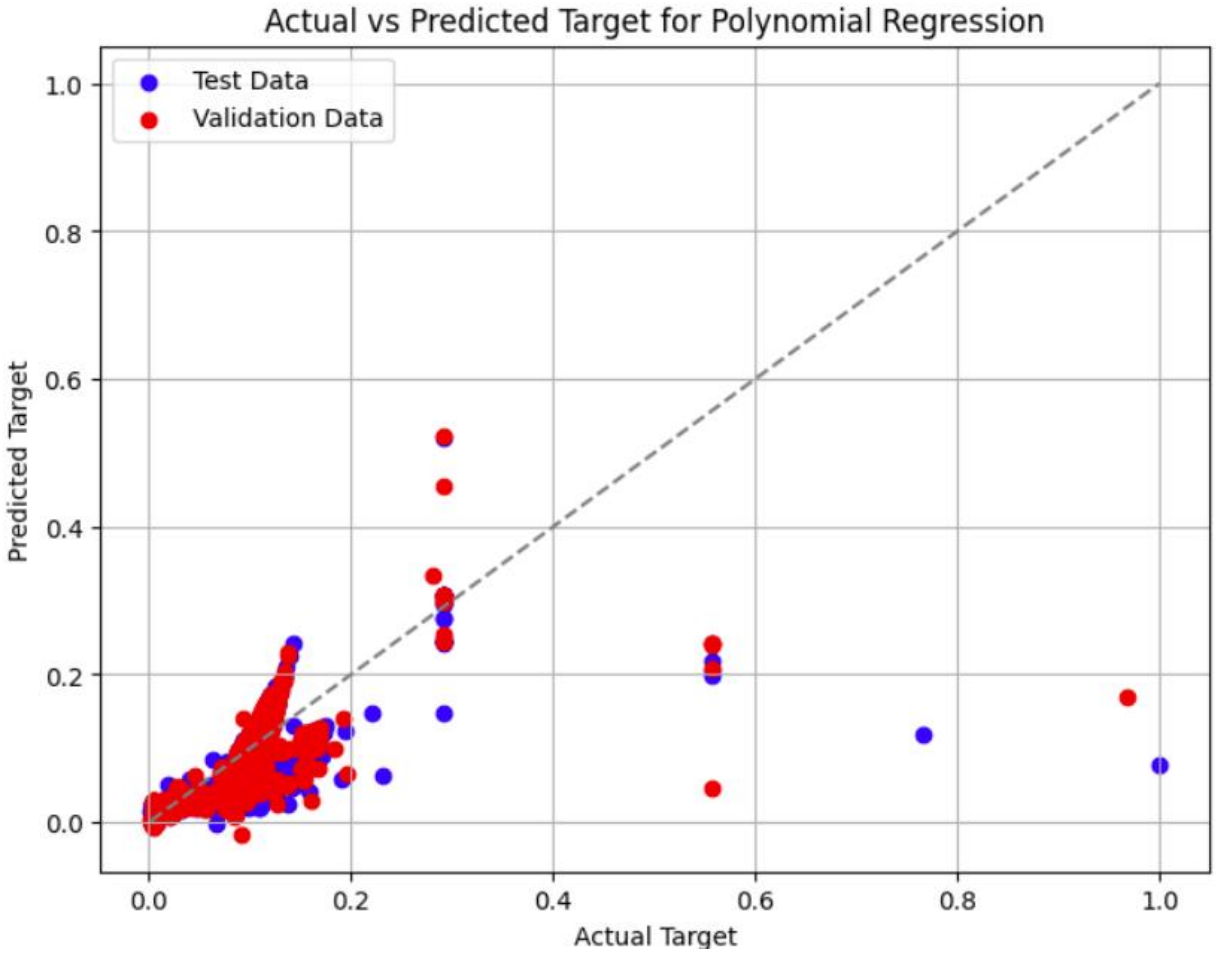


Fig 39 : Scatter Plot for Actual Vs Predicted Target for Polynomial

Neural Network Structure

Layer of Input:

Size of Input: The number of features in the dataset is the same as the number of neurons in the input layer. The number of columns in the feature matrix features, or input_size, determines this. Layers That Are Hidden First Hidden Layer has 128 neurons in total Activation Function: Rectified Linear Unit (ReLU) Goal. This layer allows the network to learn intricate patterns by converting the input data into a higher-dimensional space. The second hidden layer has 64 neurons. Activation Function: ReLU Goal: Strengthens the model's capacity to discover complex associations in the data by processing the characteristics that were retrieved by the preceding layer. The third hidden layer has 32 neurons. ReLU is the activation function. Its goal is to decrease the dimensionality of the features without losing any crucial information. There is one neuron. There is no activation function (linear output) Goal In this regression exercise, generates the network's final output, which is a single continuous value. Layer Interconnections The input layer and the

first hidden layer are joined by the fully connected (dense) layer, or fc1. fc2 A fully connected layer that establishes a connection between the first and second hidden layers. The fully connected layer known as fc3 is what links the second and third hidden layers together. fc4 A fully connected layer that joins the output layer and the third hidden layer. Functions of Activation After every hidden layer, the Rectified Linear Unit (ReLU) activation function is applied. ReLU gives the model non-linearity, which helps it learn more intricate functions. The definition of ReLU mathematically as f(x)=max(0,x) + max(0,x) where positive values are kept and all negative values are set to zero. MSE, or mean squared error, is the loss function. The average squared difference between the expected and actual values is measured by this loss function. Regression tasks frequently employ it since it penalizes excessive errors.

Optimizer:

Adam (Adaptive Moment Estimation): Adam is an optimization algorithm that adjusts the learning rate for each parameter. It combines the advantages of two other extensions of stochastic gradient descent: Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). The learning rate is set to 0.001 in this case.

Training Process:

Number of Epochs: The model is trained for 100 epochs.

Training Loop:

For each epoch, the model processes the training data in a forward pass.

The loss is calculated using the MSE loss function.

The gradients are computed via backpropagation.

The optimizer updates the model parameters to minimize the loss.

The training loss is recorded at each epoch for monitoring the training progress.

This fully connected feedforward neural network is designed to effectively learn from the input features and make accurate continuous predictions in a regression setting. The use of ReLU activation functions, a structured hidden layer configuration, and the Adam optimizer helps in achieving a model that generalizes well to unseen data.

```
# Define the model
class Net(nn.Module):
    def __init__(self, input_size, hidden_sizes, output_size):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_sizes[0])
        self.fc2 = nn.Linear(hidden_sizes[0], hidden_sizes[1])
        self.fc3 = nn.Linear(hidden_sizes[1], hidden_sizes[2])
        self.fc4 = nn.Linear(hidden_sizes[2], output_size)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = self.fc4(x)
        return x
```

Fig 40 : Structure of neural network

Training the model using the above classification and initialize the models

```
# Training configuration
input_size = len(features.columns)
hidden_sizes = [128, 64, 32]
output_size = 1
learning_rate = 0.001
num_epochs = 100
```

```
# Initialize the model
model = Net(input_size, hidden_sizes, output_size)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

Fig 41 : Configuration and model initialization

Training and Evaluation: There are 100 training epochs for the model. In every period, the subsequent actions are carried out: Forward Pass To obtain the output predictions, the input data is sent through the network. Loss Calculation The Mean Squared Error (MSE) loss function is used to compute the difference between the expected outputs and the actual target values. Backward Pass Backpropagation is used to calculate the gradients of the loss with respect to the network parameters. Update of Parameters In order to reduce loss, the Adam optimizer updates the network parameters.

```
# Lists to store training loss
train_losses = []
```

```
# Training the model
for epoch in range(num_epochs):
    model.train()
    optimizer.zero_grad()
    outputs = model(X_train_tensor)
    loss = criterion(outputs, y_train_tensor)
    loss.backward()
    optimizer.step()
    train_losses.append(loss.item())

    if (epoch+1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')
```

```
Epoch [10/100], Loss: 0.0005
Epoch [20/100], Loss: 0.0004
Epoch [30/100], Loss: 0.0004
Epoch [40/100], Loss: 0.0003
Epoch [50/100], Loss: 0.0003
Epoch [60/100], Loss: 0.0003
Epoch [70/100], Loss: 0.0003
Epoch [80/100], Loss: 0.0003
Epoch [90/100], Loss: 0.0003
Epoch [100/100], Loss: 0.0003
```

Fig 42 : Train the model

Results and Analysis: Results and Analysis

After training, the model is evaluated on the test set. The performance metrics are Mean Squared Error (MSE) Evaluates the average squared difference between the predicted and actual test values. R-squared (R2) Score Measures the proportion of variance in the dependent variable that is predictable from the independent variables. It indicates how well the predictions match the actual values. Training Loss Plot The plot below shows the training loss over the 100 epochs, indicating how the model's performance improves over time:

```
# Evaluate the model on the test set
model.eval()
with torch.no_grad():
    y_pred_test = model(X_test_tensor)
```

```
# Calculate metrics
mse_test = mean_squared_error(y_test_tensor.numpy(), y_pred_test.numpy())
r2_test = r2_score(y_test_tensor.numpy(), y_pred_test.numpy())
print("Best Model - Test Set:")
print("Mean Squared Error:", mse_test)
print("R2 Score:", r2_test)
```

```
Best Model - Test Set:
Mean Squared Error: 0.00022485422
R2 Score: 0.7907859832048416
```

Fig: 43 : Evaluation

Training Loss Plot:
The plot below shows the training loss over the 100 epochs, indicating how the model's performance improves over time

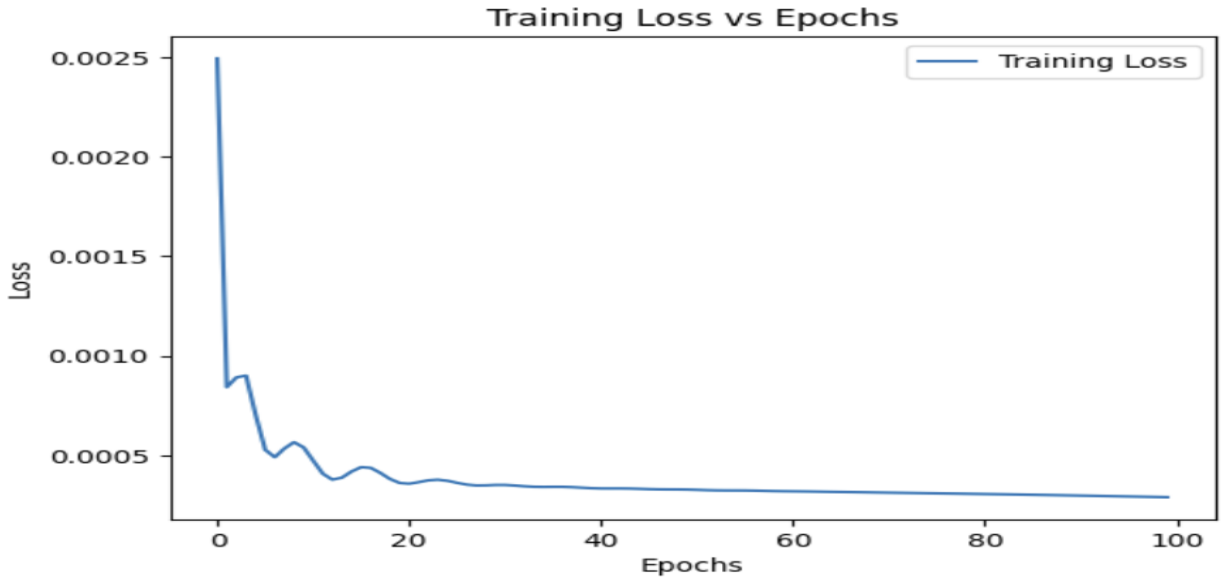


Fig 44 : Training Loss vs Epochs

Compared the Evaluation of all the models and plotted the graph for all the models with their mse and r2 score

```
# Dictionary to store results and appending all the mse and r2 score
results = {
    'Model': [],
    'MSE_Test': [],
    'R2_Test': []
}

results['Model'].append('KNN Regression')
results['MSE_Test'].append(model_knn_mse_test)
results['R2_Test'].append(model_knn_r2_score_test)

results['Model'].append('Linear Regression')
results['MSE_Test'].append(model_linear_mse_test)
results['R2_Test'].append(model_linear_r2_score_test)

results['Model'].append('Ridge Regression')
results['MSE_Test'].append(model_ridge_mse_test)
results['R2_Test'].append(model_ridge_r2_score_test)

results['Model'].append('Bayesian Ridge Regression')
results['MSE_Test'].append(model_bayesian_mse_test)
results['R2_Test'].append(model_bayesian_r2_score_test)

results['Model'].append('Neural Network')
results['MSE_Test'].append(mse_test)
results['R2_Test'].append(r2_test)
```

Fig 45 : Dic to store the values of MSE and R2

```
print("Results:")
for model, mse, r2 in zip(results['Model'], results['MSE_Test'], results['R2_Test']):
    print(f"Model: {model}, MSE: {mse}, R2: {r2}")
```

```
Results:
Model: KNN Regression, MSE: 0.09356287098496685, R2: 0.793985234716573
Model: Linear Regression, MSE: 0.00020831520116174716, R2: 0.8061745912871335
Model: Ridge Regression, MSE: 0.00020119922345103453, R2: 0.812795602526249
Model: Bayesian Ridge Regression, MSE: 0.00020830455014692726, R2: 0.806184501448702
Model: Neural Network, MSE: 0.00022485421504825354, R2: 0.7907859832048416
```

Fig 46 : Results

Plotting the graph for all the models and their accuracy score

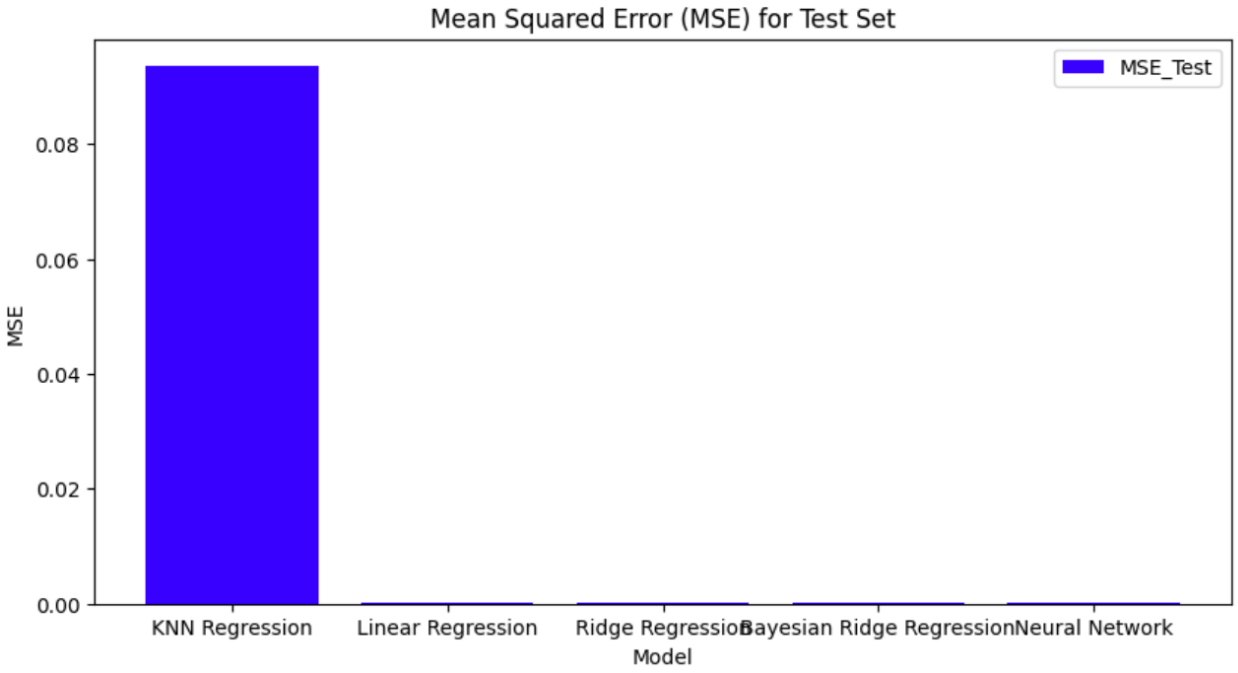


Fig 47: MSE plot for Test set

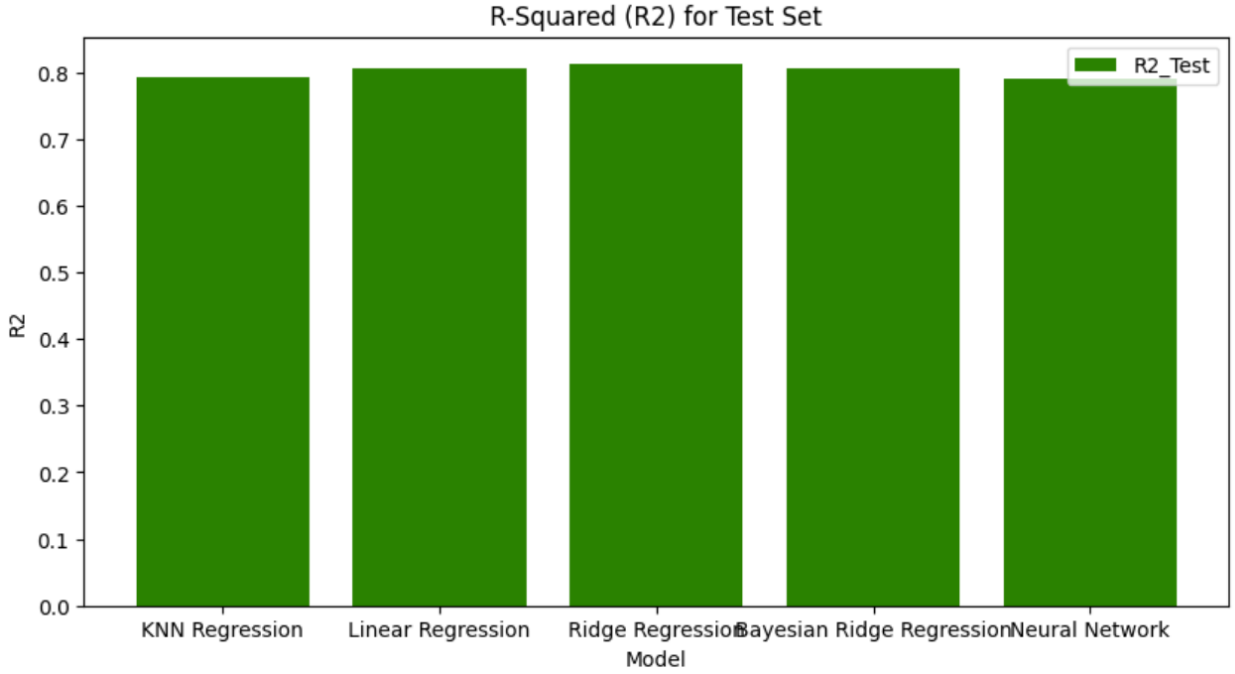


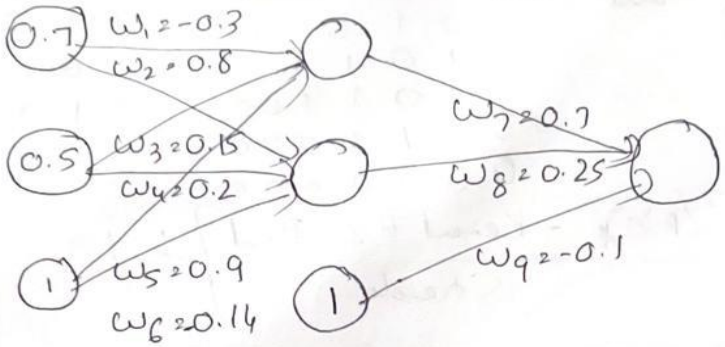
Fig 48: R2 score plot for test set

Part 2 : Deep Learning Theoretical Part

Name :- RAKSHITH KUMAR NARASIMHA MURTHY
Student ID :- 50560326 mail :- rakshit2@buffalo.edu

Part-II : Deep Learning Theoritical Part.

1. Forward backward Pass



Hidden layer : ReLu
Output layer
Learning rate = 0.03
Target(y) = 0.5

1. Forward Pass.

$h_1 = 0.7 \times w_1 + 0.5 \times w_3 + 1 \times 0.9$
$= 0.7 \times 0.3 + 0.5 \times 0.15 + 1 \times 0.9$
$= -0.21 + 0.075 + 0.9$ ReLu:
$h_1 = 0.765$

Value of the hidden layer node 1 $h_1 = 0.765$.

$h_2 = 0.7 \times w_2 + 0.5 \times w_4 + 1 \times w_6$
$= 0.7 \times 0.8 + 0.5 \times 0.2 + 1 \times -0.14$
$= 0.56 + 0.1 - 0.14$
$h_2 = 0.52$ ReLu:

Value of the hidden layer node 2 $h_2 = 0.52$.

$\hat{y} = h_1 \times w_7 + h_2 \times w_8 + 1 \times w_9$
$= 0.765 * 0.7 + 0.52 \times 0.25 + 1 \times 0.1)$
$\hat{y} = 0.5355 + 0.13 - 0.1$
$\hat{y} = 0.5655$

2) Estimate the MSE

$MSE = \frac{1}{n} \sum_{i=1}^{n} (y - \hat{y}_i)^2$ $\frac{1}{n} \sum_{i=1}^{n}$ (actual - Predicted)

For only One data, $(y - \hat{y})^2$

$MSE = (0.5 - 0.5655)^2$

$= 0.00429025$

3) Backward Propogation (to find gradient)

$Error = \hat{y} - y = 0.5655 - 0.5$

$= 0.0655$

Gradient for Output layer weight.

$\frac{\partial L}{\partial w_7} = Error \cdot h_1 = 0.0655 \times 0.765$

$= 0.0501$

$\frac{\partial L}{\partial w_8} = Error \times h_2 = 0.0655 \times 0.52 = 0.034$

$\frac{\partial L}{\partial w_9} = Error \times h_3 = 0.0655 \times 1 = 0.0655$

$\delta h_1 = Error \times w_7 * RELU(h_1) = 0.0655 \times 0.7 \times 1$

$= 0.04585$

$\delta h_2 = Error \times w_8 \times RELU(h_2) = 0.0655 \times 0.25 \times 1$

$= 0.016375$

$Relu(h) = 1$ if $h_1, h_2 > 0$.

Gradients for hidden layer weights $w_1, w_2, w_3, w_4, w_5, w_6$

$\frac{\partial L}{\partial w_1} = \delta h_1 \times x_1 = 0.04585 \times 0.7 = 0.03209$

$\frac{\partial L}{\partial w_3} = \delta h_2 \times x_2 = 0.04585 \times 0.5 = 0.022$

$\frac{\partial L}{\partial w_5} = \delta h_1 \times x_3 = 0.04585 \times 1 = 0.04585$

$\frac{\partial L}{\partial w_2} = \delta h_2 \times x_1 = 0.016375 \times 0.7$

$= 0.0114$

$\frac{\partial L}{\partial w_4} = \delta h_2 \times i_2 = 0.01637 \times 0.5 = 0.0081$

$\frac{\partial L}{\partial w_8} = \delta h_2 \times i_3 = 0.01637 \times 1 = 0.01637$

4) Update the weights & bias.

$w_1 = w_1 - \alpha \frac{\partial L}{\partial w_i}$

Learning rate(α) = 0.03

$W_1 = W_1 - 0.03 \frac{\partial L}{\partial w_1} = -0.3 - 0.03 \times 0.0321$

$w_1 = -0.3009$

$w_2 = w_2 - 0.03 \frac{\partial L}{\partial w_2} = 0.8 - 0.03 \times 0.0114$

$w_2 = 0.7996$

$w_4 = w_4 - 0.03 \frac{\partial L}{\partial w_4} = 0.2 - 0.03 \times 0.0818$

$w_4 = 0.19$

$w_3 = w_3 - 0.03 \frac{\partial L}{\partial w_3} = 0.15 - 0.03 \times 0.0229$

$= 0.15 - 0.00068$

$w_3 = 0.1493$

$w_4 = w_4 - 0.03 \frac{\partial L}{\partial w_4} = 0.2 - 0.03 \times 0.00818$

$w_4 = 0.1997$

$w_5 = w_5 - 0.03 \frac{\partial L}{\partial w_5} = 0.9 - 0.03 \times 0.0458$

$w_5 = -0.8986$

$w_6 = w_6 - 0.03 \frac{\partial L}{\partial w_6} = -0.14 - 0.03 \times 0.01637$

$w_6 = -0.1405$

$w_7 = w_7 - 0.03 \frac{\partial L}{\partial w_7} = 0.7 - 0.03 \times 0.050$

$w_7 = 0.6985$

$W_8 = W_8 - 0.03 \frac{\partial L}{\partial W_8} = 0.25 - 0.0010$

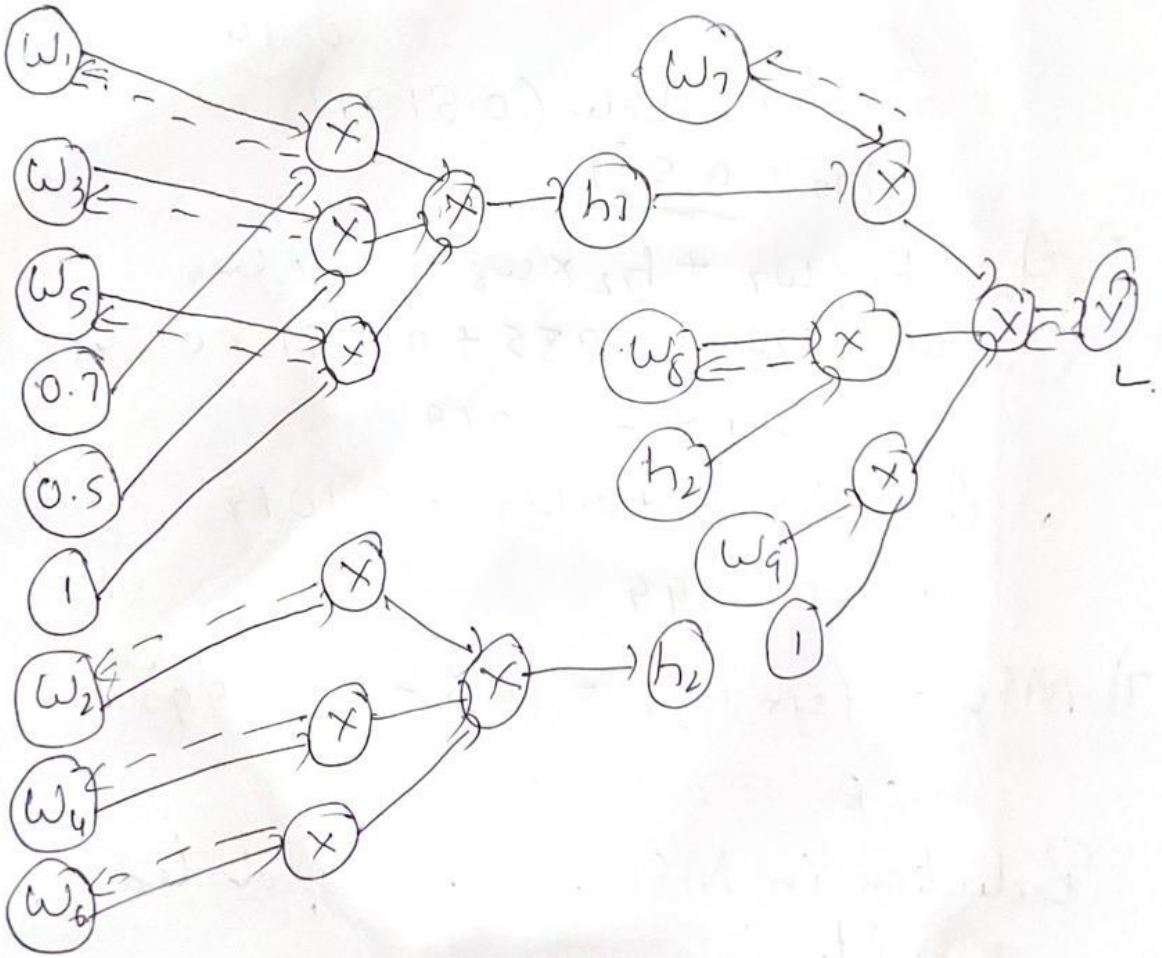
$W_8 = 0.2489$

$W_9 = W_9 - 0.03 \frac{\partial L}{\partial W_9} = -0.1 - 0.03 \times 0.0655$

$W_9 = -0.1019$

5) Computational Graph

Forward Pass & Backward Propagation



6) Forward Pass using updated Weight

$h_1 = 0.7 \times w_1 + 0.5 \times w_3 + 1 \times w_5$

$= 0.7 \times -0.3009 + 0.5 \times 0.1493 + 1 \times 0.8986$

$= -0.2106 + 0.0746 + 0.8986$

$h_1 = 0.7626 \quad = \text{Relu}(0.7626)$

$h_1 = 0.7626$

$h_2 = 0.7 \times w_2 + 0.5 \times w_4 + 1 \times w_6$

$= 0.7 \times 0.7996 + 0.5 \times 0.1997 + (-1) \times 0.1405$

$= 0.5197 = \text{Relu}(0.5191)$

$h_2 = 0.5191$

$\hat{y}_2 = h_1 \times w_7 + h_2 \times w_8 + h_3 \times w_9$

$= 0.7626 \times 0.6985 + 0.5191 \times 0.2489 + 1 \times -0.1019$

$\hat{y}_2 = 0.5326 + 0.1292 - 0.1019$

$\hat{y}_2 = 0.5599$

7) $MSE = (y - \hat{y}_2)^2 = (0.5 - 0.5599)^2$

$MSE = 0.0035$

Reduction in MSE for the updated weight is

$= MSE_1 - MSE_2$

$= 0.00429 - 0.0035$

$= 0.00079$ (Reduction in loss)

Derivation of tanh

The hyperbolic tangent (tanh) activation function has following form:

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Tanh :-

To Prove the derivation of Tanh which has the form,

$$f'(x) = 1 - f(x)^2$$

$u = e^x$ and $v = e^{-x}$

$$\tanh = \frac{u - v}{u + v}$$

differentiate both

$$g(x) = \frac{h(x)}{k(x)} \Rightarrow g'(x) = \frac{h'(x)k(x) - h(x)k'(x)}{[k(x)]^2}$$

$h(x) = u - v$

$k(x) = u + v$

$$\frac{d}{dx} e^x = e^x \cdot u \frac{d}{dx} e^{-x} = -v$$

$$\left(\frac{u}{v}\right)' = \frac{u'v - uv'}{v^2}$$

$$u' = \frac{d(e^x - e^{-x})}{\partial x} = e^x + e^{-x}$$

$$v' = \frac{d(e^x + e^{-x})}{\partial x} = e^x - e^{-x}$$

Apply Quotient rule.

$$y' = \frac{u'v - uv'}{v^2} = \frac{(e^x + e^{-x})(e^x + e^{-x}) - (e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2}$$

$$y' = \frac{(e^x + e^{-x})^2 - (e^x - e^{-x})^2}{(e^x + e^{-x})^2}$$

Apply $(a+b)^2 = a^2 + b^2 + 2ab$

$(a-b)^2 = a^2 + b^2 - 2ab$

$$y' = \frac{(e^{2x} + e^{2x} + 2 \cdot e^x e^{-x}) - (e^{2x} + e^{-2x} - 2e^x e^{-x})}{(e^x + e^{-x})^2}$$

$$y' = \frac{e^{2x} + e^{-2x} + 2 - e^{2x} - e^{-2x} + 2}{(e^x + e^{-x})^2} = \frac{4}{(e^x - e^{-x})^2} \quad -(2)$$

As tanh value $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

We can take square to future Calculate

$$\tan^2 h(x) = \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2} = \frac{e^{2x} - 2 + e^{-2x}}{e^{2x} + 2 + e^{-2x}} \quad -(3)$$

$$= \frac{e^{2x} + 2 + e^{-2x} - e^{2x} + 2 - e^{-2x}}{e^{2x} + 2 + e^{-2x}} = \frac{4}{e^{2x} + 2 + e^{-2x}} \quad -(4)$$

Equation (4) & (2) $y' = \frac{4}{(e^x - e^{-x})^2} = 1 - \tan^2 h(x)$

$\tanh(x) = 1 - \tan^2 h(x) \Rightarrow \boxed{f'(x) = 1 - f^2(x)}$

$1 - \tan^2 h(x) = \tanh'(x)$ So

Part 3 :OCTMNIST Classification :

1)Description of Neural Network:

The OCTMNIST dataset is a collection of 109,309 valid optical coherence tomography (OCT) images, specifically aimed at detecting and classifying retinal diseases. Each image in the dataset has a resolution of 28x28 pixels and is associated with one of four classes. The dataset is part of the MedMNIST collection, which includes various medical imaging datasets for diverse machine learning tasks.

The neural network is a Convolutional Neural Network (CNN) designed for image classification. It consists of three convolutional layers followed by two fully connected layers. The first convolutional layer has 16 output channels, the second has 32, and the third has 64, each with a kernel size of 3x3, batch normalization, ReLU activation, and 2x2 max pooling. The first fully connected layer has 128 output features and uses ReLU activation and a dropout of 0.5 to prevent overfitting. The second fully connected layer outputs the final class predictions.

```
# Define a CNN model
class CNN(nn.Module):
    def __init__(self, in_channels, num_classes):
        super(CNN, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(in_channels, 16, kernel_size=3, padding=1),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.layer2 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.layer3 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.fc1 = nn.Sequential(
            nn.Linear(64 * 3 * 3, 128),
            nn.ReLU(),
            nn.Dropout(0.5)
        )
        self.fc2 = nn.Linear(128, num_classes)

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        x = self.fc2(x)
        return x

model = CNN(in_channels=n_channels, num_classes=n_classes)
```

Fig 49: CNN for octmnist classification

2) Impacts of techniques applied:

Regularization techniques such as L2 regularization (applied via weight decay), dropout, and early stopping are crucial for improving the generalization performance of neural networks. L2 regularization helps prevent overfitting by penalizing large weights, promoting simpler model architectures. Dropout, applied with a probability of 0.5, forces the network to learn redundant representations, aiding in preventing overfitting by introducing randomness during training. Early stopping, with a patience of 5 epochs, monitors the validation loss and halts training when the model's performance on the validation set stops improving, thereby preventing overfitting. Together, these techniques enhance the model's ability to generalize well to unseen data while mitigating the risk of overfitting.

```
# Define loss function and optimizer with L2 regularization
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE, weight_decay=WEIGHT_DECAY)
```

Fig 50 : L2 regularization

```
# Train the model
model, history = train(model, train_loader, val_loader, criterion, optimizer, epochs=NUM_EPOCHS)
```

```
100%|██████████| 762/762 [11:04<00:00,  1.15it/s]
Epoch [1/10], Train Loss: 0.5472, Train Accuracy: 81.32%, Val Loss: 0.5595, Val Accuracy: 78.96%
100%|██████████| 762/762 [13:43<00:00,  1.08s/it]
Epoch [2/10], Train Loss: 0.4036, Train Accuracy: 86.25%, Val Loss: 0.4458, Val Accuracy: 85.05%
100%|██████████| 762/762 [12:49<00:00,  1.01s/it]
Epoch [3/10], Train Loss: 0.3603, Train Accuracy: 87.60%, Val Loss: 0.3293, Val Accuracy: 88.16%
100%|██████████| 762/762 [13:57<00:00,  1.10s/it]
Epoch [4/10], Train Loss: 0.3372, Train Accuracy: 88.38%, Val Loss: 0.3301, Val Accuracy: 88.50%
100%|██████████| 762/762 [15:26<00:00,  1.22s/it]
Epoch [5/10], Train Loss: 0.3153, Train Accuracy: 89.08%, Val Loss: 0.2609, Val Accuracy: 90.67%
100%|██████████| 762/762 [11:47<00:00,  1.08it/s]
Epoch [6/10], Train Loss: 0.3008, Train Accuracy: 89.69%, Val Loss: 0.2533, Val Accuracy: 91.06%
100%|██████████| 762/762 [10:49<00:00,  1.17it/s]
Epoch [7/10], Train Loss: 0.2883, Train Accuracy: 90.21%, Val Loss: 0.3309, Val Accuracy: 88.08%
100%|██████████| 762/762 [14:31<00:00,  1.14s/it]
Epoch [8/10], Train Loss: 0.2766, Train Accuracy: 90.44%, Val Loss: 0.3010, Val Accuracy: 89.19%
100%|██████████| 762/762 [14:52<00:00,  1.17s/it]
Epoch [9/10], Train Loss: 0.2669, Train Accuracy: 90.89%, Val Loss: 0.2290, Val Accuracy: 91.73%
100%|██████████| 762/762 [17:35<00:00,  1.38s/it]
Epoch [10/10], Train Loss: 0.2558, Train Accuracy: 91.16%, Val Loss: 0.2330, Val Accuracy: 91.79%
```

Fig 51 : Model Training and accuracy

```python
# Test the model
def test(model, test_loader, criterion):
    model.eval()
    test_loss = 0.0
    correct = 0
    total = 0
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for inputs, targets in test_loader:
            outputs = model(inputs)
            loss = criterion(outputs, targets.squeeze().long())
            test_loss += loss.item()
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets.squeeze().long()).sum().item()
            all_preds.extend(predicted.cpu().numpy())
            all_labels.extend(targets.squeeze().long().cpu().numpy())
threshold = 0.1
low_correlation_features = correlation_with_target[abs(correlation_with_target) < threshold].index.tolist()
    test_loss /= len(test_loader)
    test_accuracy = 100. * correct / total
    print(f'Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.2f}%')
    return all_labels, all_preds, test_loss, test_accuracy
```

Fig 52 : Testing the model

3) Results and Relevant Graphs :

The training and validation metrics provide insights into the performance of the model during training. Tracking training accuracy and loss helps monitor how well the model learns from the training data, while validation accuracy and loss indicate its generalization capability. Finally, testing accuracy and loss offer a comprehensive evaluation of the model's performance on unseen data, crucial for assessing its real-world effectiveness. These metrics collectively facilitate understanding and fine-tuning of the model's behavior throughout the training process.

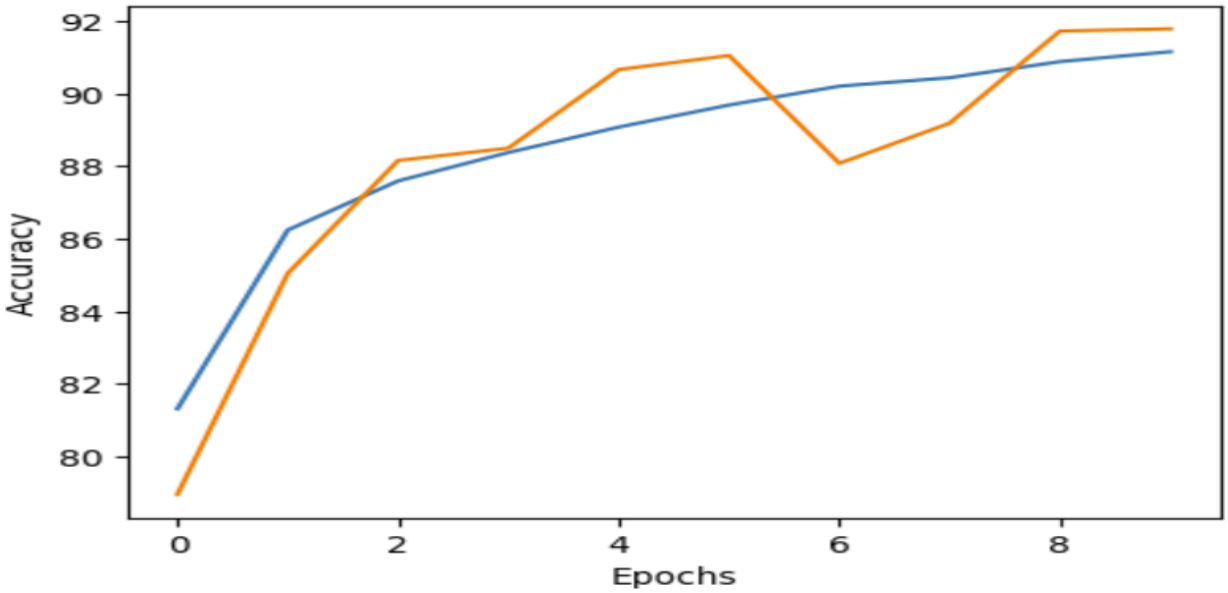


Fig 53 : Epochs vs Accuracy

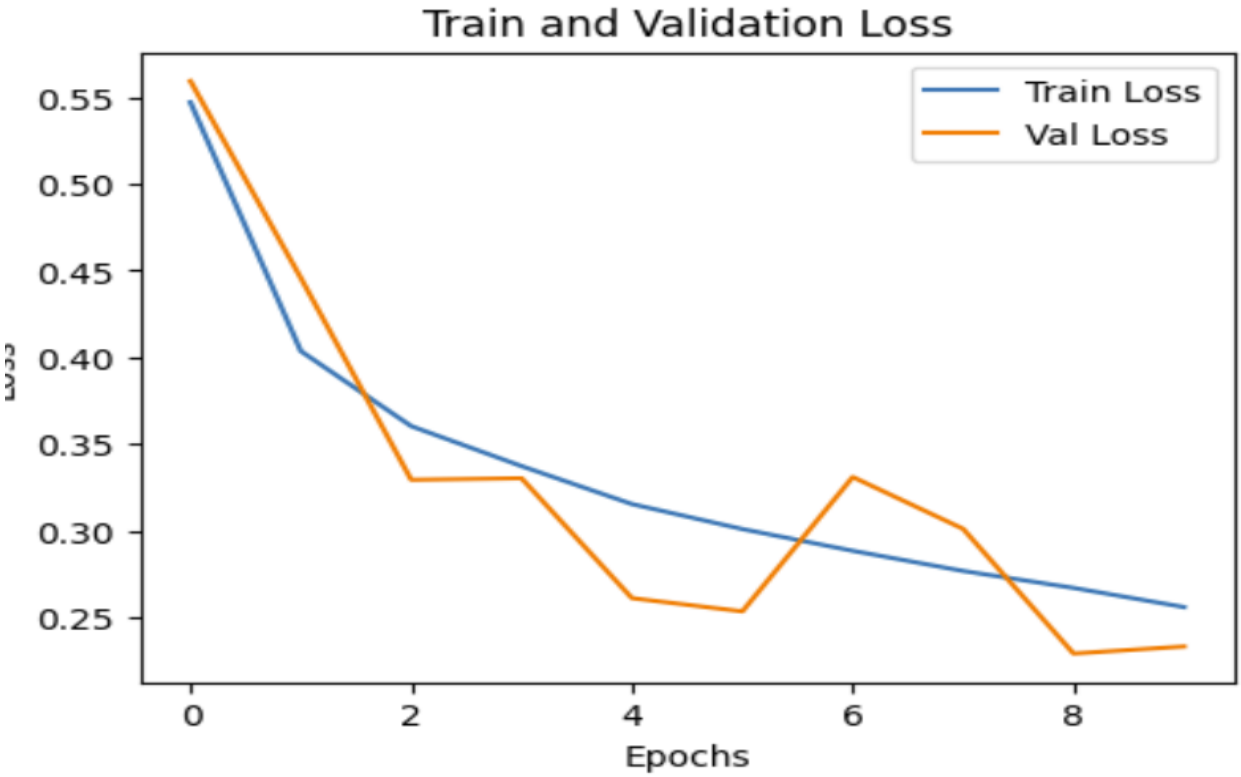


Fig 54 : Train and validation loss

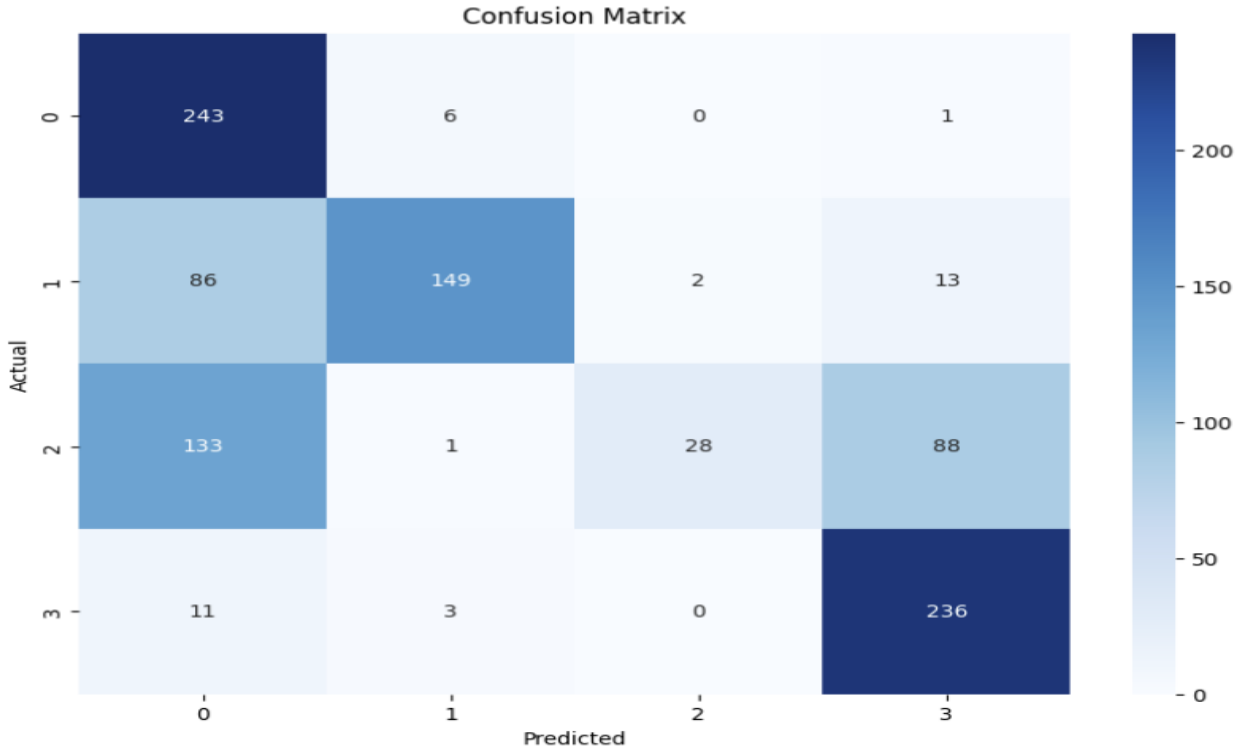


Fig 55 : confusion matrix

```
# Calculate and report other evaluation metrics
precision = precision_score(all_labels, all_preds, average='weighted')
recall = recall_score(all_labels, all_preds, average='weighted')

print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
```

```
Precision: 0.7706
Recall: 0.6560
```

Fig 56: Evaluation metrics

References:

1) https://github.com/pyjanitor-devs/pyjanitor
2) http://localhost:8501/
3) Data.gov Home - Data.gov
4) https://catalog.data.gov/dataset/clean-air-markets-allowances-query-wizard
5) https://www.youtube.com/watch?v=GP-2634exqA
6) https://towardsdatascience.com/demo-your-model-with-streamlit-a76011467dfb
7) https://www.geeksforgeeks.org/deploy-a-machine-learning-model-using-streamlit-library/
8) https://medium.com/deep-math-machine-learning-ai/chapter-1-complete-linear-regression-with-math-25b2639dde23
9) https://github.com/MBKraus/Predicting_real_estate_prices_using_scikit-learn
10) https://www.geeksforgeeks.org/implementation-of-k-nearest-neighbors-from-scratch-using-python/
11) https://github.com/SurbhiJainUSC/Linear-Regression-and-KNN-Regression
12) https://www.geeksforgeeks.org/implementation-of-bayesian-regression/
13) https://github.com/krasserm/bayesian-machine-learning/blob/dev/bayesian-linear-regression/bayesian_linear_regression.ipynb
14) https://github.com/lmc2179/Polynomial-Regression/blob/master/Regression.py
15) https://github.com/SiddhantAttavar/PolynomialRegression
16) https://github.com/bluebehree/polynomial-regression
17) https://github.com/geekquad/Lasso-Ridge-Regression-and-Elastic_Net-Regularization-from-Scratch
18) https://github.com/akaashagarwal/ridge-regression
19) https://github.com/aroques/ridge-regression
20) https://www.youtube.com/watch?v=Y90NTNG_yJg
21) https://www.youtube.com/watch?v=a7_1-HNNAFw
22) https://www.youtube.com/watch?v=qbuZDQDx6zU
23) https://www.youtube.com/watch?v=Q81RR3yKn30&t=23s
24) https://www.youtube.com/watch?v=VqKq78PVO9g
25) https://www.youtube.com/watch?v=9lRv01HDU0s
26) https://www.youtube.com/watch?v=oWq6aVv5mC8&t=1s

27) https://www.youtube.com/watch?v=oWq6aVv5mC8&t=1s
28) https://www.youtube.com/watch?v=Bkw-boJQz9s&list=PL3Dh_99BJkCEhE7Ri8W6aijiEqm3ZoGRq
29) https://www.youtube.com/watch?v=mozBidd58VQ
30) https://www.youtube.com/watch?v=gBw0u_5u0qU
31) https://github.com/MedMNIST/MedMNIST/blob/main/examples/getting_started.ipynb
32) https://medmnist.com/
33) Early stopping - Wikipedia
34) f1_score — scikit-learn 1.5.0 documentation
35) Precision and recall - Wikipedia