Chapter 1: Data Analysis of a Combustion System

Introduction

This document contains the steps to perform data analysis on some industrial combustion data before creating a machine learning models to predict system performance under different inputs.

Prerequisites

Developing and performing the tasks in this tutorial requires several pieces of software and supporting libraries. The tutorial was developed and tested using Python 3.10.10, though this specific version may not be required so long as the supporting libraries are compatible. These libraries include:

Library	Version Used	Description
numpy	1.24.2	Fundamental package for array computing in Python
pandas	1.5.3	Powerful data structures for data analysis, time series, and statistics
scikit-learn	1.2.2	A set of python modules for machine learning and data mining
lightgbm	3.3.5	Light Gradient-Boosting Machine, machine learning framework
plotly	5.14.1	An open-source, interactive data visualization library for Python

While the exact version numbers above are not required, older or newer versions may become incompatible with some methods used in this tutorial. Other libraries may be needed depending on how the problems/tasks are completed.

In addition to the python environment details listed above, the dataset `process-data.h5` will be needed to perform this tutorial. It should be located in the `data/` folder.

Background

The dataset `process-data.h5` contains the measurement data from the test of a simple air-fired combustion system over the course of several months. The system is a methane-fueled combustion chamber. There are several thermocouples within the chamber to measure the gas temperature (labeled `tc1` to `tc4`), as well as a flue thermocouple to measure the exhaust gas temperature (labeled `tcf`). The flue also has a gas analyzer to determine the composition of the exhaust gas.

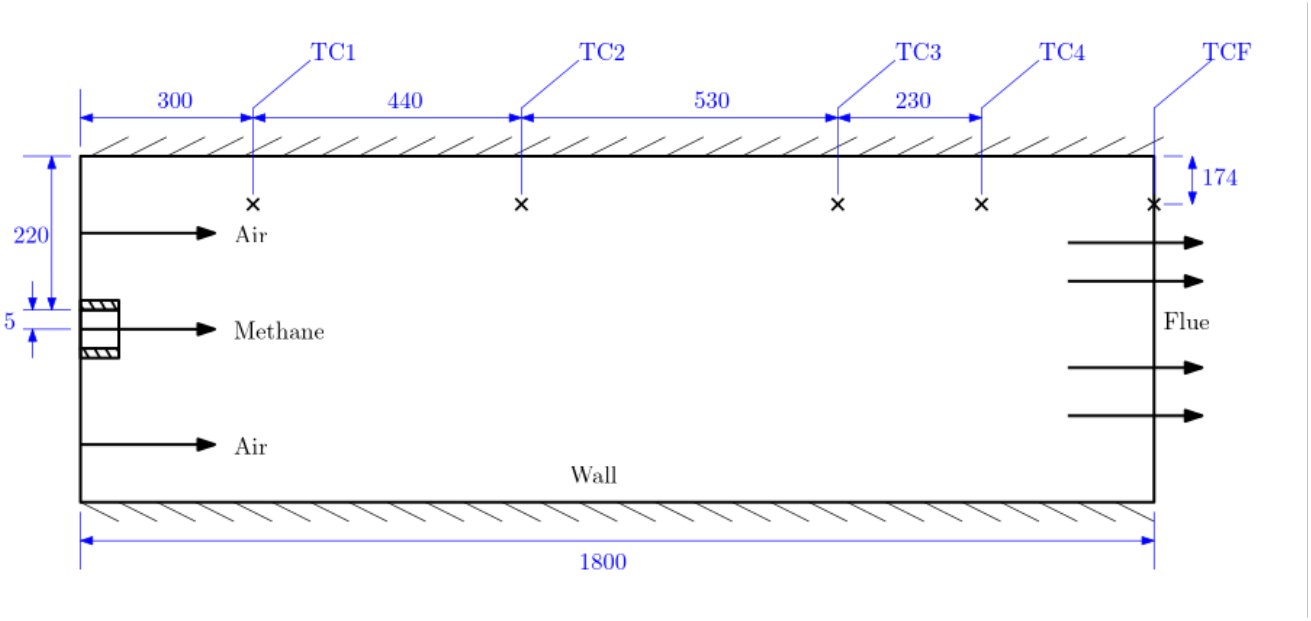

Figure 1: System Domain

The thermocouples are known to have an expected accuracy of +/- 3°C, but can potentially fail due to high temperature environment of the testing chamber. The flue gas analyzer will sample a small stream of the exhaust gasses and perform a chemical composition analysis to determine the approximate concentration of several key gases. Currently, the system will measure the levels of *water vapor*, *carbon dioxide*, *oxygen*, *NOx*, and *unburnt methane*.

Problem Description

The purpose of the combustion test is to examine the performance of a burner. To this end, the operators have performed several test runs by adjusting four factors:

1. The amount of air provided to the system
2. The amount of methane fuel provided to the system
3. The oxygen content of the air stream
4. The temperature of the air stream

However, running these operations can be expensive and time-consuming. The operators are interested in the development of a machine learning program that provide information about different burner set-points without having to run those studies. Therefore, the tasks are:

1. Clean up the data to remove failed sensors and downtimes.
2. Explore the cleaned data and plot relevant behaviors
3. Build a linear regression model to predict NOx emissions and examine the results
    - build addtional models to predict the flue temperature and co2 emissions
4. Build a more complex model using LightGBM and compare the results to the linear regression model
    - repeat for flue temperature and co2 emissions
5. Identify an optimal configuration to minimize NOx emissions

Data Description

The data is provided as a *hdf5* file, which can be loaded via `pandas` using the following methods to create a new DataFrame `df` containing the raw process data:

```
''' Read process data into a pandas DataFrame '''
import numpy as np
import pandas as pd

# read process data from hdf5 file
df = pd.read_hdf('../data/process-data.h5')

# examine the columns and first five rows
df.head()
```

The data has the following columns:

column	description
timestamp	timestamp of data record
air.flow	flow rate of air in SCFH
air.temp	air temperature in °C
air.frac	air mole fraction of oxygen
fuel.flow	flow rate of methane fuel in SCFH
tc1... tcf	thermocouple measurements in °C
f.h2o ... f.co	flue species exhaust in tonne/hr
f.nox	flue NOx exhaust as a dry volume fraction
spec	process procedure parameter
stoppage	True/False signal of system shutoff
hub	process procedure parameter
shift	work shift
trial	process trial number

Task 1: Data Cleanup

Before performing analysis on the data, the data must be cleaned to remove invalid entries. Exploring the data will show three primary issues that need to be resolved:

1. Remove instances of dead/failed thermocouples
2. Removing downtime periods where the process is shut down
3. Removing invalid entries such as `NaN` or empty values.

For the first tasks of cleaning up the thermocouple data, we must first determine what criteria to use.

```
# examine statistics of thermocouple data
df[['tc1','tc2','tc3','tc4','tc4']].describe()
```

	tc1	tc2	tc3	tc4	tc4
count	133921.000000	133921.000000	133921.000000	133921.000000	133921.000000
mean	99.168950	443.279934	1554.673210	1604.034289	1604.034289
std	559.469778	695.425814	863.410878	542.676655	542.676655
min	22.000000	27.000000	25.000000	28.000000	28.000000
25%	30.000000	131.000000	1085.000000	1218.000000	1218.000000
50%	57.000000	171.000000	1328.000000	1482.000000	1482.000000
75%	99.000000	345.000000	1866.000000	1912.000000	1912.000000
max	9999.000000	9999.000000	9999.000000	3039.000000	3039.000000

Examining the thermocouple columns using `df.describe()`, we see that all thermocouple share similar minimum values of around `20C`, which is roughly room temperature. This may be a valid measurement if the process is not running, as the thermocouples will read the ambient air temperature. However, several of the thermocouples appear to have maximum readings of `9999C`, which is not a valid temperature. Depending on the control logic and type of thermocouple, failed thermocouples can register as a variety of things. Nonphysical values such as `9999C` are a common indicator of failed thermocouples.

To remove these values, a new DataFrame will be created that is derived from a `query()` of the invalid entries. The condition of `tc < 5000` will be used as this can exclude both the `9999C` failure condition and any potential invalid entries that may have been created as the thermocouple failed while also reducing the likelihood of accidentally removing valid entries. A thorough analysis of the data may be needed to improve the criteria.

```
''' Clean up raw data by removing bad thermocouple entries '''
# get the shape of the raw dataframe
print('Before filtering: ',df.shape) # returns (133921,21)pandoc

# create a new dataframe using only rows where all thermocouple entries are valid
fildf = df.query('tc1 < 5000 and tc2 < 5000 and tc3 < 5000 \
     and tc4 < 5000 and tcf < 5000').reset_index(drop=True)

# get new dataframe shape
print('After filtering: ',fildf.shape)  # returns (131943,21)
```

Note that `query()` is being used here in a reverse way, with the criteria being defined to *exclude* invalid entries rather than *select* them. You may choose to used different logic or methods for filtering values. After filtering, the number of rows has decreased from `133921` to `131943`, a loss of about 2000 rows or 1.4% of the data. If data filtering results in significant loss of data, make sure to examine your filtering methods. If the methods are valid or necessary, the dataset may not be sufficient for analysis, depending on needs.

Proceed with the remaining data cleanup tasks listed above before continuing to the next stage. After completing data cleanup, write the filtered data to a new hdf5 file:

```
# write filtered dataframe to hdf5 file
fildf.to_hdf('../data/clean-data.h5',key='df',mode='w')
```

Task 2: Data Analysis and Visualization

Before moving on to building the regression model, it is important to analyze the data. While the data in this study is relatively simple, machine learning models for complex real-world industrial data may require exploration of the dataset to identify key features and relationships. Data visualization is also key, as it can reveal important concepts and connections between the input and output parameters.

One simple first step is to explore the input parameters of the data. As mentioned above, the combustion tests in this dataset were preformed by altering four input values: the air flow rate, air temperature, air oxygen fraction, and the fuel flow rate. In combustion, the air-fuel ratio is critical to the performance of the combustion system. Therefore, for the first step a plot of the air and fuel configurations will be created.

```
''' Create air-fuel ratio scatter plot '''
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go

# read dataframe
df = pd.read_hdf('../data/clean-data.h5','df')

## create plot using plotly graph objects
# initalize plot
af_spots = go.Figure()

# add air and fuel markers
af_spots.add_trace(
    go.Scatter(
        x = df['air.flow'],
        y = df['fuel.flow'],
        mode='markers',
        name='air-fuel'
    )
)

# update figure size and title
af_spots.update_layout(width=800,height=800,title='Air-Fuel Distribution')

# add axis labels
af_spots.update_xaxes(title='Air Flow Rates (SCFH)')
af_spots.update_yaxes(title='Fuel Flow Rates (SCFH)')

# save figure as static image
af_spots.write_image('../fig/air-fuel_plot.jpg')
```

This produces a plot like the one below:

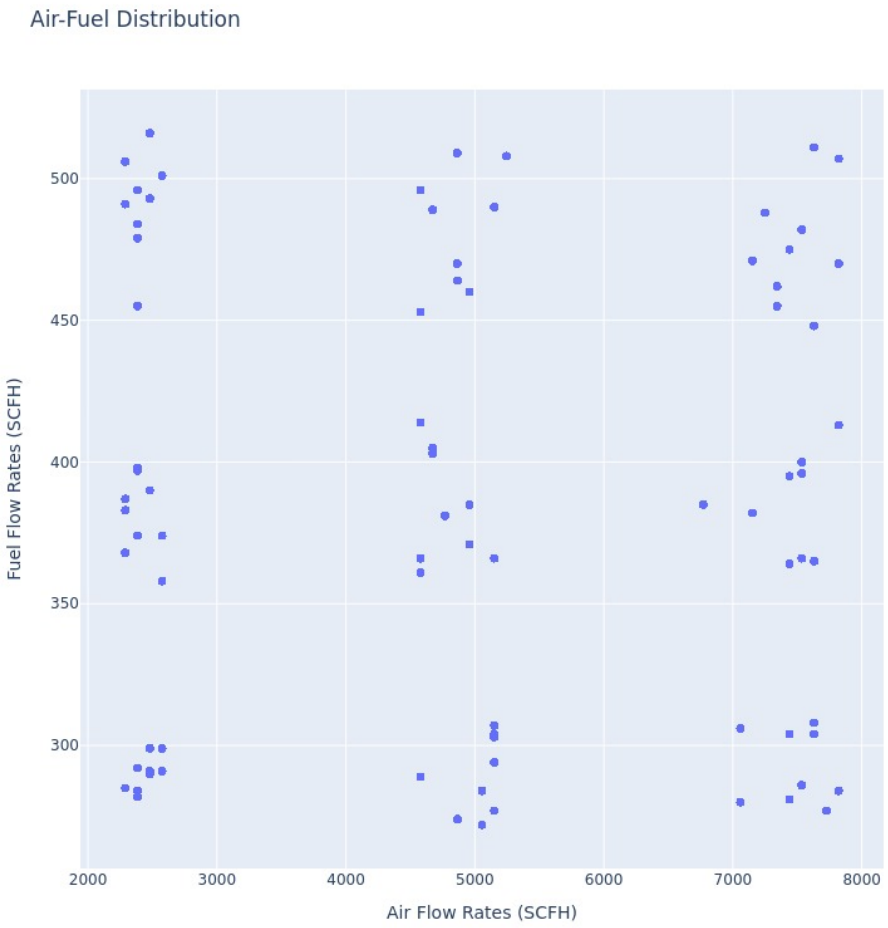

Figure 2: Air-Fuel Ratio Plot

From looking at the plot, it would appear that there nine overall combinations being tested, but that the variance of the air and fuel flow rates is somewhat high. Examine some other input combinations.

It is also important to explore the relationship between the input parameters and the output parameters. For example, here is the same plot as above but with the markers colored by the flue temperature. In the interactive version of this plot, the user can also hover over the marker values to see additional data (in this case the temperature value).

```
''' Create air-fuel ratio scatter plot with temperature coloration '''
af_tcf = go.Figure()
af_tcf.add_trace(
    go.Scatter(
        x = df['air.flow'],
        y = df['fuel.flow'],

        # define marker coloring and colorscale
        marker = dict(
            size = 12,
            color = df['tcf'],
            colorbar = dict(title=dict(text="TCF")),
            showscale=True
        ),
        # show temperature value on hover
        hovertext=df['tcf'],
```

```
        mode='markers',
        name='air-fuel'

    )
)
af_tcf.update_layout(width=800,height=800,title='Air-Fuel Distribution')
af_tcf.update_xaxes(title='Air Flow Rates (SCFH)')
af_tcf.update_yaxes(title='Fuel Flow Rates (SCFH)')

# display plot
af_tcf.show()

# save plot as image
af_tcf.write_image('../fig/airfuel_temp_plot.jpg')
```

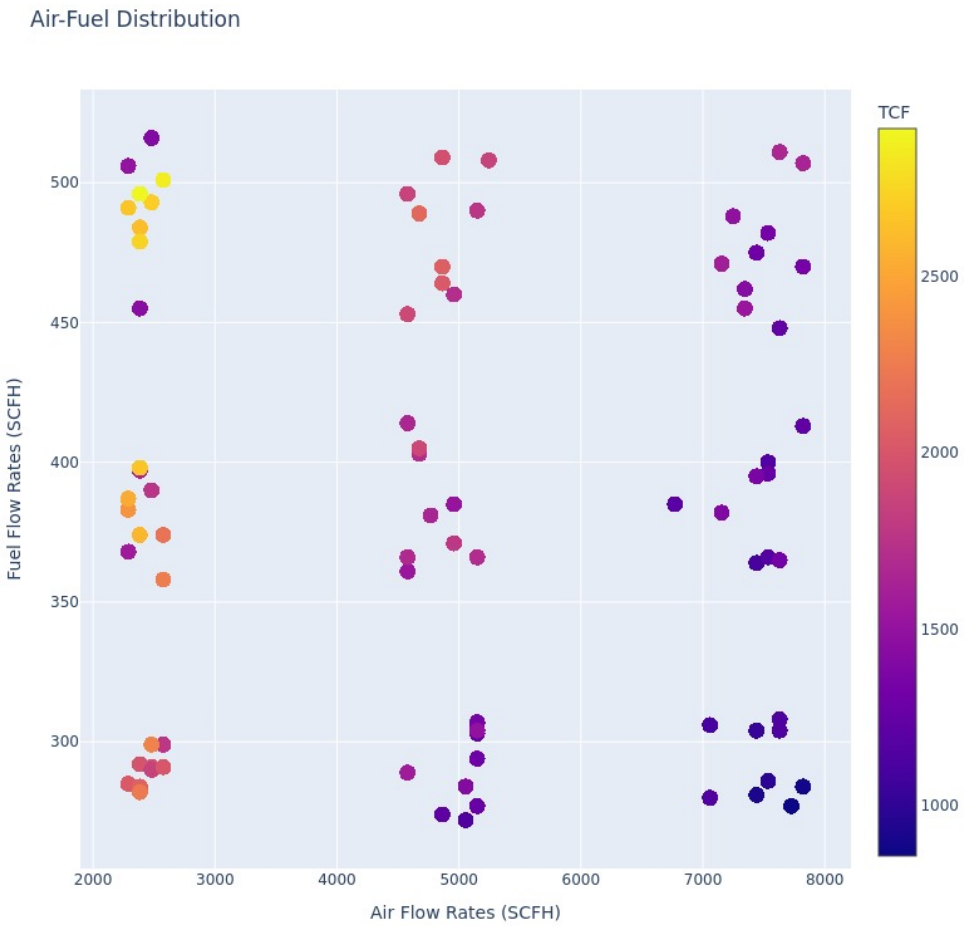

Figure 3: Air-Fuel Ratios with Flue Temperatures

Looking at the figure, a clear relationship can be seen between the air-fuel ratio and the resulting flue temperature. It would appear that the air flow rates of between 2000 and 3000 SCFH are enough for most of the fuel flow rates in this study, as higher air flow rates seem to result in decreasing temperatures. Each cluster of air-fuel ratios also seems to show a range of values, likely due to the influence of the other two values.

Additional Tasks/Questions

- Explore a few other input combinations and analyze their impact on the output values
- How does the air-fuel ratio impact the NOx production?
- How about air temperature or oxygen fraction?

Task 3: Linear Regression

With the data cleaned up and examined, it is now time to build a simple linear regression model to analyze the data. For this task, the `scikit-learn` library will be used.

The data has been divided into two sets: a training set containing 80% of the input data that will be used to fit the model, and a testing dataset that is the remaining 20% that will be used to judge the model accuracy. The model is initialized and then fit using `model.fit()`. Finally, the `score()` method is used to evaluate the R^2 of the model. The R^2 is the "coefficient of determination" that provides statistical information about how well a model fits the data. It ranges from `[0-1]`, with `1.0` being a perfect fit.

```
''' Load data for linear regression model '''
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import plotly.graph_objects as go

# read cleaned data
df = pd.read_hdf('../data/clean-data.h5')
df.head()

# list of input features
inputs = ['air.flow','air.temp','air.frac','fuel.flow']

# list of output features
columns = ['tc1','tc2','tc3','tc4','tcf','f.h2o','f.co2',
    'f.o2','f.ch4','f.co','f.nox']

# selected output feature
output = ['f.nox']
```

Here, the cleaned data has been imported and arrays of the input feature column names has been created. The list of output columns is created as `columns`, and the specific target output has been created as `output`. This will be passed to `train_test_split()` function to divide up the input and output data into training datasets and testing datasets. No validation dataset will be used in this example.

```
''' Build linear regression model using sklearn.LinearRegression '''
# split the data into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(
    df[inputs],
    df[output],
    test_size=0.2,
    random_state=42)

# initiate the linear regression model
model = LinearRegression()

# fit the linear model using the input data
model.fit(X_train,y_train)

# determine r2 score for model
score = model.score(X_test,y_test)
```

```
display('R2: ' + str(score)) # returns R2 = 0.099
```

It looks like the R^2 of the linear regression model for NOx is very poorly-fitted, with a value of `0.099`. To explore why the model's R^2 value is so low, further analysis can be done. Statistical analysis can be performed to explore deviations in the data and predictions, but a simple first step is to create a *parity plot* that will visualize the true values against the predicted outputs.

```
''' Evaluate model performance with a parity plot '''
# predict output for test data and put into a dataframe
predictions = model.predict(X_test)
predictionsDF = pd.DataFrame(predictions,columns=output)

# Plot parity of actual versus predicted values
parity = go.Figure()

# add test v. predicted markers
parity.add_trace(
    go.Scatter(
        x=y_test['f.nox'],
        y=predictionsDF['f.nox'],
        mode='markers',
        name='results'
    )
)

# add parity line
parity.add_trace(
    go.Scatter(
        x=y_test['f.nox'],
        y=y_test['f.nox'],
        name='parity'
    )
)

# update layout and title and set height and width to a square ratio
parity.update_layout(height=800,width=800,
    title="NOx Actual vs. Predicted")

#display figure
parity.show()
```

Note that in the `update_layout()` method, the height and width are defined such that the plot is a square aspect ratio. This is helpful for parity plots as the performance is being compared in a linear faction against a 45^circ line that represents optimal performance.

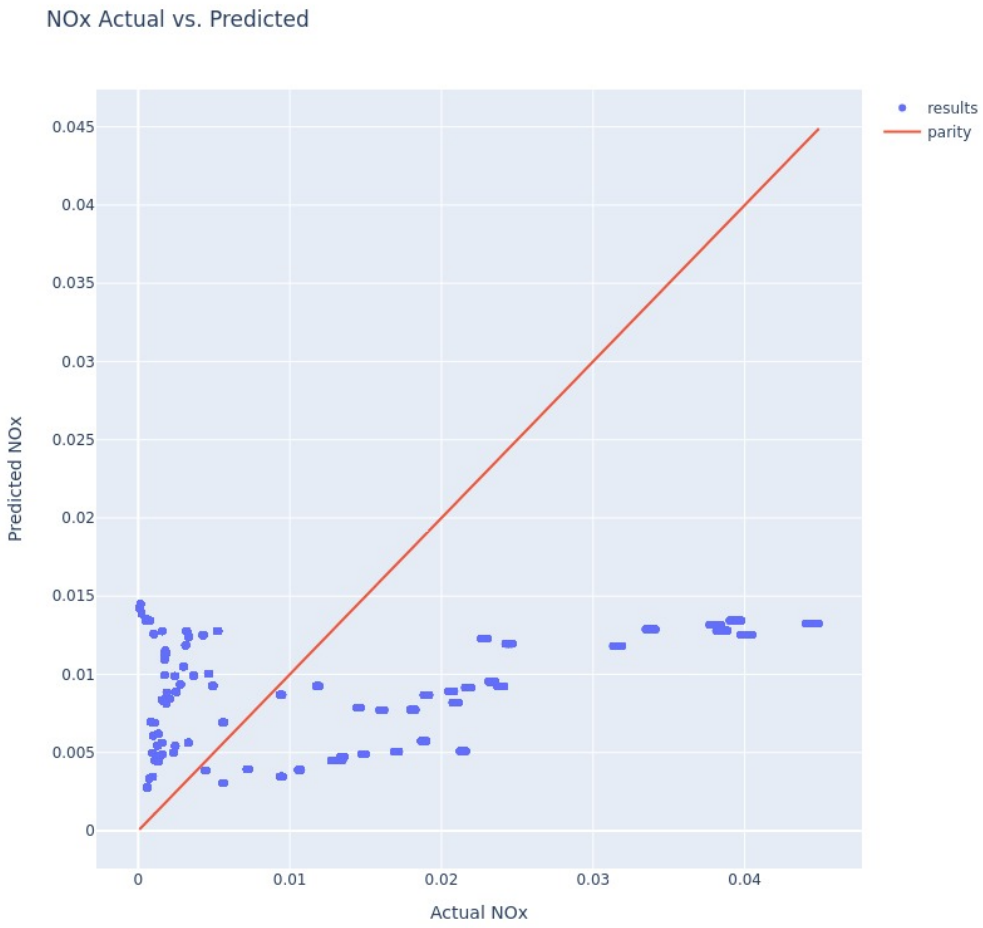

Figure 4: Linear Regression NOx Parity Plot

From the parity plot, it can be seen that the model almost completely fails to predict the NOx values. The linear regression model might not be a good choice for predicting the NOx levels generated by the combustion system.

Additional Tasks/Questions

- Build the linear regression models for the flue CO2 emission and the flue temperature data
- Why might the linear regression approach show poor performance with this dataset?
- Can the linear regression model be improved?

Task 4: Build a LightGBM Model for Prediction

For this task, build a new model using the `LightGBM` toolkit. Follow instructions from the LightGBM Documentation or other online sources.

Objectives

1. Build a LightGBM model to predict the flue NOx emissions
2. Compare the accuracy of the LightGBM model to the linear regression model
   - Compare the Mean Absolute Error (MAE) and R^2 performance
   - Visually compare the resulting parity plots
3. Build additional LightGBM models for the flue CO2 and flue temperatures
   - Compare with linear regression results and parity plots

Additional Tasks/Questions

- How does the performance of the LightGBM model compare to the linear regression model?
- If the performance is different, why?

Task 5: Optimization

In an industrial setting, machine learning and data analysis is rarely done without some specific purpose. In this case, the purpose of a combustion test would be to determine the behavior of the input parameters on the resulting output data, usually to improve performance by finding optimal configurations. However, "optimal" does not mean the same thing for each of the outputs. For example, operators might want to minimize fuel usage while maximizing the flue temperatures to save money on fuel. Similarly, the resulting pollutants of the combustion process (such as CO2 and NOx) need to be minimized as much as they can while still meeting process requirements.

For this step, find an optimization library that can use the trained LightGBM model to:

1. Optimize inputs to minimize NOx emissions with a fuel flow rate of 282 SCFH
2. Optimize inputs to minimize CO2 emissions with a fuel flow rate of 282 SCFH

Make sure to include logical input ranges, as the model might (correctly) determine that the best way to minimze emissions is to just turn the system off! For the four inputs, use the following ranges:

Input	Min	Max
Air Flow	1500	10000
Air Temp	20	200
Air Oxygen Fraction	0.21	1.0
Fuel Flow	282	282

Additional Tasks/Questions

- Does the resulting combination of input parameters make physical sense? Explain why.
- What is the optimal configuration if the air oxygen fraction is also fixed to `0.21`?

Presenting Results

After completing the above tasks, please prepare a slide deck containing the problem description and the results of each task. Make sure to include your own interpretation of the meaning behind the results and behavior of the system. When your own code is used, please provide relevant code snippets when it would help explain your methods.

You are also encouraged to explore additional methods to approaching these problems based on your experience and ideas!