Chapter 1: Data Analysis of a Combustion System

Introduction

This is a description of the tutorial

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. In addition to the python environment details listed above, the dataset process-data.h5 will be needed to perform this tutorial.

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.

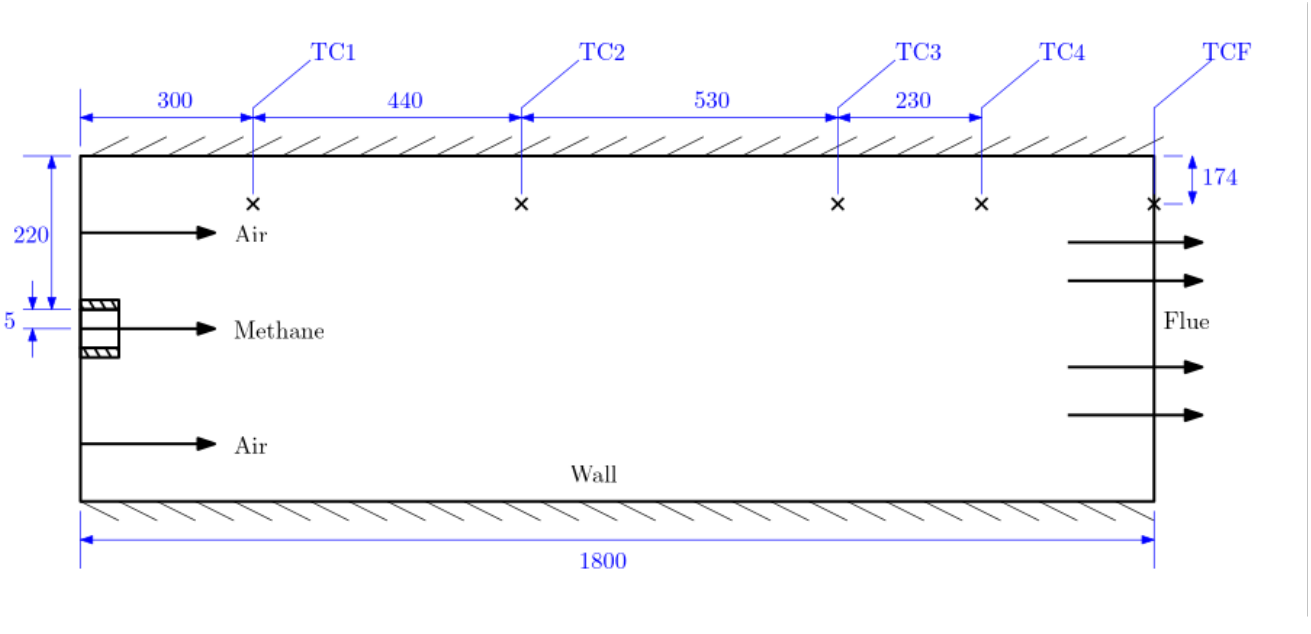

Figure 1: domain-fig

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 a second model 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 an *hdf5* file, which can be loaded in *pandas* via the following methods to create a new DataFrame `df` containing the raw process data:

```
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

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.

```
# 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')
```

Data Analysis

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.