Python Assignment

Topic: LED Transient Thermal Data 4

Group 12

Anh Khoa Dang

An Quang Nguyen

Thi Van Khanh Nguyen

To accomplish the task, we developed 2 different pythons files. One to do linear regression on data of each file and then output the estimated voltage at t=0 into a csv file, which is called ‘Voltage estimation’. The second python file will use information from the output csv to develop regression from polynomial of second order to find out the sensitivity function (V(T)) as the task requires, which is called ‘Sensitivity function’.

You can find the full version of the code as well as output plots at the submission package in the zip form. Within this documentation we include only snippets from the code.

1. Voltage estimation

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import linregress
import os

#List of temperature from txt file
temps = [26.8, 30.4, 34, 36, 37.8, 40.6, 45.7, 54.7, 65.5, 67.8, 69.0, 72.2, 82.2, 87.1, 100.2, 106.8, 111.0, 115.1, 116.8]
file_number_list = [1,2,3,4,5,6,7,8,10,11,12,14,16,17,19,20,21,22,23]
switch_indices = []
in_stable_indices = []
out_stable_indices = []
```

We started by defining the variables and lists that might be needed later. This includes temps and file_number_list, extracted from temps.txt file.

The indices list for switch, in_stable and end_stable will record the indices of switch moment, begin of stable phase and end of stable phase.

```python
# Create the output plots directory for voltage and current
voltage_output_directory = 'voltage output plots'
if not os.path.exists(voltage_output_directory):
    os.makedirs(voltage_output_directory)
current_output_directory = 'current output plots'
if not os.path.exists(current_output_directory):
    os.makedirs(current_output_directory)


#Function compute slope
def compute_slopes(current, time_intervals):
    delta_current = np.diff(current)
    delta_time = np.diff(time_intervals)
    slopes = delta_current / delta_time
    return slopes
```

Define output directory for voltage plots and current plots later.

Then we define the function to compute the slopes for all entries within the files and store the slopes in a list called ‘’slopes’’.

```python
#Function find switch point, stable zone of current
def allthingscurrent(current,time_intervals):
    switch_index = None
    stable_begin_index = None
    stable_end_index = None

    found_switch = False
    found_stable_begin = False
    found_stable_end = False
    slopes = compute_slopes(current, time_intervals)
    from collections import Counter
    # Detect switch, stable begin, and stable end
    for i in range(len(slopes)):
        if slopes[i] > 8000 and not found_switch:
            switch_index = i
            found_switch = True
            break

    if found_switch:
        for i in range(switch_index, len(slopes)):
            if abs(slopes[i]) < 100: streak += 1
            else:streak = 0

            if streak == 50 and not found_stable_begin and 0.73 > current[i + 1] > 0.729:
                stable_begin_index = i -50
                found_stable_begin = True
                break

    if found_stable_begin:
        for i in range(stable_begin_index, len(slopes)):
            if slopes[i] < -1000 and not found_stable_end:
                stable_end_index = i
                found_stable_end = True
                break
    return switch_index, stable_begin_index, stable_end_index
```

In this snippet, we develop a function called ‘allthingscurrent’. The point of this function is to return the indices of switch moment t=0, begin of stable phase and end of stable phase for each file.

Code explanation:

- To find the moment of switch, we identify when the current abruptly changes significantly. This can be found using the slopes stored in the slopes list. We identify for all files at the moment of switch slopes can be 8000 or bigger, so we use that as threshold.

- Once the moment of switch index is found, we find the moment where current enter stable phase. To do this, we identify the moment where the slopes remain relatively low for a long period. That indicates the current has settled down. To do this we have a variable called ‘streak’, identify the streak of entries whose slopes remained relatively low. When streak is 50, it could be suggested that the current has entered stable phase 50 entries ago, and that is begin of stable phase or in_stable_index.

- Similarly, once in_stable_index is found, we identify when stable phase ends. This is identified when the slopes is suddenly significantly negative. We used the threshold -1000 and the result turns out quite accurate.

```
def savedata(filenumber,t0_voltage):
    try:
        estimated_voltages = np.loadtxt("estimated_voltages.csv", delimiter=",")
    except FileNotFoundError:
        estimated_voltages = np.empty((0, 6))
    if filenumber == 1:
        estimated_voltages = np.empty((0, 6))
    for i in range(0,len(file_number_list)):
        if file_number_list[i]== filenumber:
            temperature =temps[i]
            switch_index = switch_indices[i]
            in_stable_index = in_stable_indices[i]
            out_stable_index = out_stable_indices[i]
    new_data = np.array([[filenumber, t0_voltage, temperature,switch_index,in_stable_index, out_stable_index]])
    estimated_voltages = np.vstack((estimated_voltages, new_data))
    np.savetxt("estimated_voltages.csv", estimated_voltages, delimiter=",",fmt="%f")
```

- Function savedata is defined. This function saves switch moment, begin and end of stable phase indices and voltages information of each file to a csv file. If there is not yet a file, one will be created, if there is a file, new information is added on top of the old file. However, when there is a file but file number is 1, it indicates that users run program again after output has been created, in that case savedata will write on a completely new file. This is to avoid data duplication within csv file.

Problem faced: We could not add the label before adding the information. The error was estimated voltages array must be (0,6) for all the csv, as far as we understand. The solution is to create a readme.txt file that indicates the information of the saved data. (see next)

```
#Create readme file
def create_readme():
    readme_file = 'estimated_value.readme.txt'

    if not os.path.exists(readme_file):
        content = 'Label for estimated_value.csv: "File number, Estimated voltage, Temperature, Switch index, Begin of stable phase index

        with open(readme_file, 'w') as f:
            f.write(content)
```

```
voltage, Temperature, Switch index, Begin of stable phase index, End of stable phase index"
```

```python
# plot voltage function
def plotvoltage(filenumber):
    # Load CSV data
    csv_path = f'data_set/turn_on{filenumber}.csv'
    df_csv = pd.read_csv(csv_path, skiprows=8)
    voltage = df_csv.iloc[:, 1]
    current = df_csv.iloc[:, 14]
    time_intervals = df_csv.iloc[:, 23]

    # Get indices for current analysis
    switch_index, in_stable_index, out_stable_index = allthingscurrent(current, time_intervals)

    # Add the index to indices list
    switch_indices.append(switch_index)
    in_stable_indices.append(in_stable_index)
    out_stable_indices.append(out_stable_index)
    # Adjust time intervals relative to the switch index
    adjusted_time_intervals = time_intervals - time_intervals[switch_index]

    # Calculate the square root of time, handle negative values by taking absolute value before sqrt
    sqrt_time = np.sqrt(np.abs(adjusted_time_intervals)) * np.sign(adjusted_time_intervals)

    # Record values for the stable phase
    voltage_recorded = voltage[in_stable_index:out_stable_index]
    sqrt_time_recorded = sqrt_time[in_stable_index:out_stable_index]

    # Perform linear regression
    res = linregress(sqrt_time_recorded, voltage_recorded)
    extrapolated_voltage_at_t0 = res.slope * np.sqrt(0) + res.intercept
    # Calculate extrapolated voltage at t = 0
    t_0 = np.array([[0]])
    extrapolated_voltage = res.intercept + res.slope * np.sqrt(t_0)
```

- Next, the function to plot voltage. The goal is to plot how voltage changes during the time recorded in the data, and note all the indices where moment of switch t=0, begin and end of stable phase happen. We first start by extracting data from csv files to a data frame called ‘plotvoltage’. From this data frame, voltage and current data is extracted as column number 1 and 14.

- Next, from the defined function ‘allthingscurrent’ we were able to locate the important indices and store them in respective list (switch_indices, in_stable_indices and out_stable_indices).

- The moment of switch is set as t=0, therefore we minus all entries of time in the data from the moment of t=0. The next thing is to square root this adjusted time list. However, since t=0 is not the start of the data set, entries which stand before t=0 will have negative time value. And negative numbers cannot be root squared. This is where initially we had a problem. The solution is to square root the absolute value, and later return the sign (plus or minus) of the instance after it is squared.

- Voltage_recorded and sqrt_time_recorded are objects to record only the part of voltage and sqrt_time where current has settled down (stable phase). The beginning and ending indices of stable phase are found in the previous step. These recorded values are then used to do linear regression, to find out the fitted line.

- From the linear regression we were able to do extrapolation to find the estimated value of voltage at t=0. This value, along with other indices of each file are later stored in csv file.

```
def plotvoltage(filenumber):

    #Extend fitted line
    t_extended = np.linspace(0, 0.01, 500)
    sqrt_time_extended = np.sqrt(t_extended)
    extrapolated_voltage_extended = res.intercept + res.slope * sqrt_time_extended

    # Plot the entire voltage series against the square root of time
    plt.plot(sqrt_time, voltage, label='Entire Voltage Series')

    # Highlight specific points
    plt.scatter(sqrt_time[switch_index], voltage[switch_index], color='green', marker='o', label='Power Switch')
    plt.plot(sqrt_time_recorded, voltage_recorded, 'o', label='Stable Phase Data')
    plt.scatter(0, extrapolated_voltage, color='black', marker='x', label='Extrapolated t=0')

    # Plot the extended fitted line
    plt.plot(sqrt_time_extended, extrapolated_voltage_extended, 'r', label='Extended Fitted Line')

    # Labels and title
    plt.xlabel('Square Root of Adjusted Time')
    plt.ylabel('Voltage')
    plt.title('Voltage vs. Square Root of Adjusted Time')
    plt.legend()
    plt.grid(True)



# Save the plot in the 'output plots' directory
    plot_filename = os.path.join(voltage_output_directory, f'voltage_plot_{filenumber}.png')
    plt.savefig(plot_filename)
    plt.show()

    return filenumber, extrapolated_voltage_at_t0
```

- Next, linspace is used to generate extra points to extend the fitted line. Voltage vs square rooted time is then plotted for the whole period, highlighting the important

points. Next, the extended fitted line is also plotted to see where it intercepts t=0. That point is the estimated extrapolated value of voltage.

- Finally, plot is saved into voltage output directory defined earlier for later uses.
- The function returns file number and the estimated voltage at t=0.

```
def plotcurrent(filenumber):

    csv_path = f'data_set/turn_on{filenumber}.csv'
    df_csv = pd.read_csv(csv_path, skiprows=8)


    voltage = df_csv.iloc[:, 1]
    current = df_csv.iloc[:, 14]
    time_intervals = df_csv.iloc[:, 23]
    switch_index, in_stable_index, out_stable_index = allthingscurrent(current,time_intervals)
    if switch_index is not None:
        current_recorded = current
        adjusted_time_intervals = time_intervals - time_intervals[switch_index]
    sqrt_time = np.sqrt(np.abs(adjusted_time_intervals)) * np.sign(adjusted_time_intervals)
    plt.plot(sqrt_time, current_recorded)

    if switch_index is not None:
        current_recorded = current
        adjusted_time_intervals = time_intervals - time_intervals[switch_index]
        sqrt_time = np.sqrt(np.abs(adjusted_time_intervals)) * np.sign(adjusted_time_intervals)
        plt.plot(sqrt_time, current_recorded)

        # Highlight specific points using adjusted times and square root of time
        plt.scatter(sqrt_time[switch_index], current[switch_index], color='green', marker='o', label='Power Switch')

        plt.scatter(sqrt_time[in_stable_index], current[in_stable_index], color='blue', marker='o', label='Start Stable')

        plt.scatter(sqrt_time[out_stable_index], current[out_stable_index], color='red', marker='o', label='End Stable')

    plt.xlabel('Square Root of Time')
    plt.ylabel('Current')
    plt.title('Current vs. Square Root of Time')
    plt.grid(True)
    plt.legend()
```

```
    plot_filename = os.path.join(current_output_directory, f'current_plot_{filenumber}.png')
    plt.savefig(plot_filename)

    plt.show()
    plt.show
    return filenumber
```

- Plotcurrent function is defined similarly to plotvoltage. The only difference is it is used for current data and no need for linear regression.

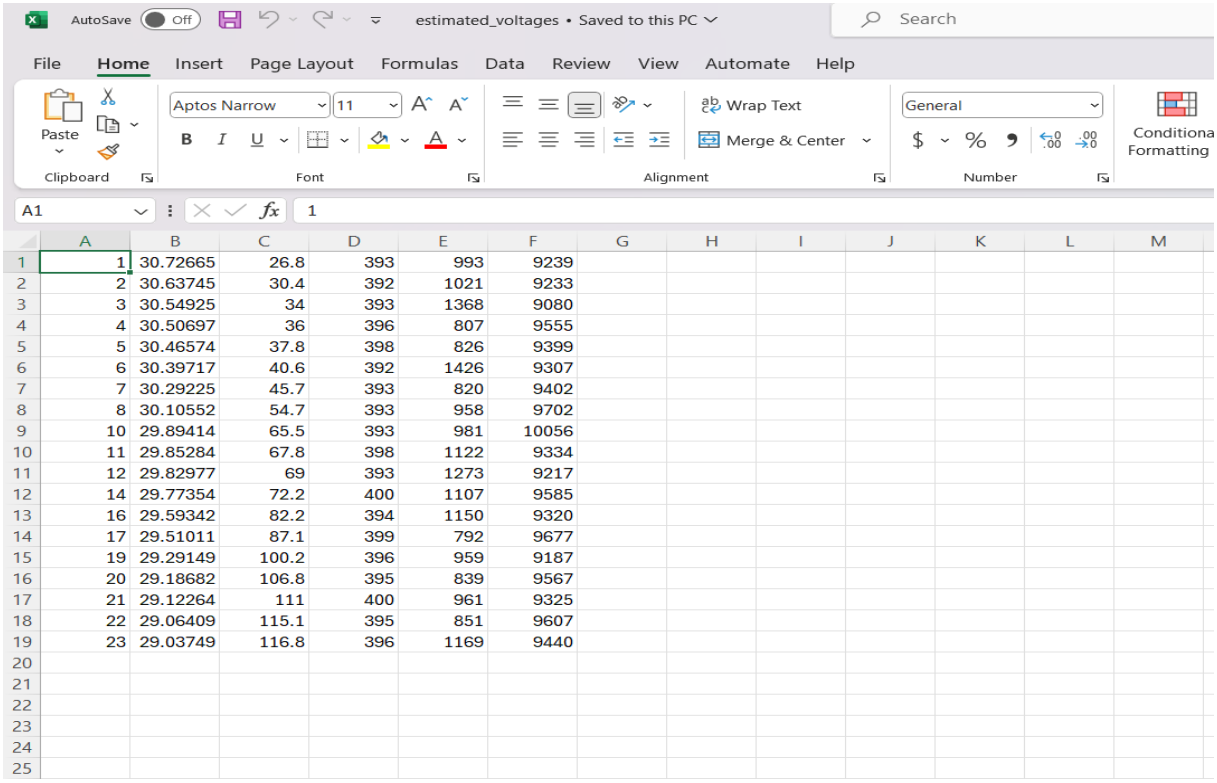
```python
def main():

    for i in range (1,24):
        try:
            filenumber, t0_voltage = plotvoltage(i)
            savedata(filenumber,t0_voltage)
            plotcurrent(i)
        except FileNotFoundError:
            pass
    create_readme()

main()
```

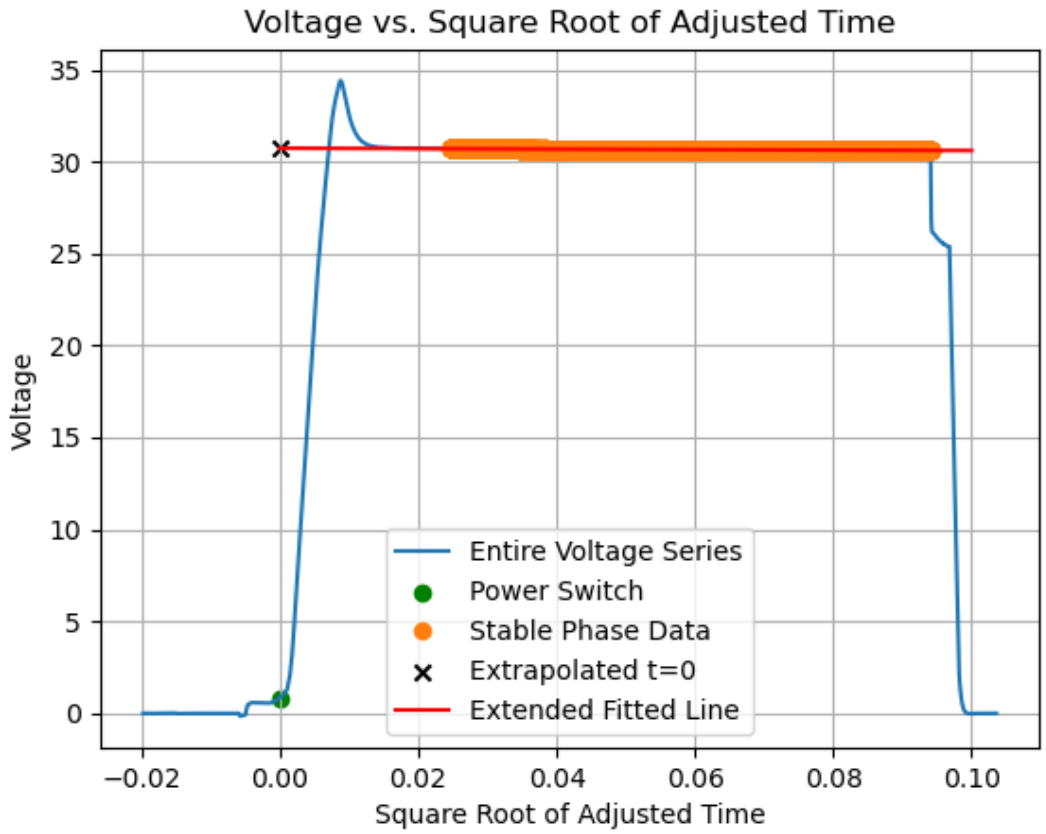
- Finally, in main() function, we loop 23 times, if the index of the loop corresponds to an existing turnon file, plotvoltage(), plotcurrent() for the file is performed. Results from plotvoltage (the estimated voltage at t=0) is then stored using savedata().
- Create_readme() is put outside the loop as it only needs to be created once.

Output results of main():

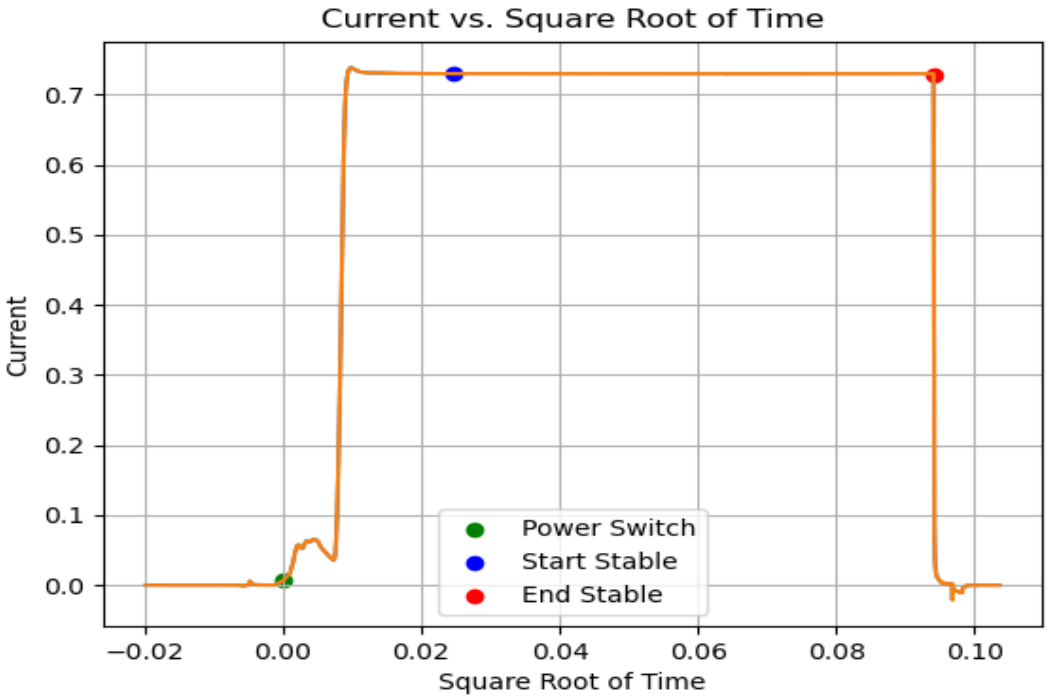


	A	B	C	D	E	F	G	H	I	J	K	L	M
1	1	30.72665	26.8	393	993	9239							
2	2	30.63745	30.4	392	1021	9233							
3	3	30.54925	34	393	1368	9080							
4	4	30.50697	36	396	807	9555							
5	5	30.46574	37.8	398	826	9399							
6	6	30.39717	40.6	392	1426	9307							
7	7	30.29225	45.7	393	820	9402							
8	8	30.10552	54.7	393	958	9702							
9	10	29.89414	65.5	393	981	10056							
10	11	29.85284	67.8	398	1122	9334							
11	12	29.82977	69	393	1273	9217							
12	14	29.77354	72.2	400	1107	9585							
13	16	29.59342	82.2	394	1150	9320							
14	17	29.51011	87.1	399	792	9677							
15	19	29.29149	100.2	396	959	9187							
16	20	29.18682	106.8	395	839	9567							
17	21	29.12264	111	400	961	9325							
18	22	29.06409	115.1	395	851	9607							
19	23	29.03749	116.8	396	1169	9440							
20													
21													
22													
23													
24													
25													

The above entries are entered into estimated_voltage.csv.



Voltage plots are stored in folder ‘Voltage output plots’. In the picture is the plot for turnon_1. Other plots for other files can all be found in the mentioned folder.



Similarly, output plots for current, stored in folder ‘Current output plots’. This is an example for file turnon_1.

2. Sensitivity function

Sensitivity function.py > ...

```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import os


sf_output_directory = 'Sensitivity Function'
if not os.path.exists(sf_output_directory):
    os.makedirs(sf_output_directory)


# Load data from your CSV file
df = pd.read_csv('estimated_voltages.csv')
temperature = df.iloc[:, 2]
voltage = df.iloc[:, 1]

# Perform quadratic regression
coefficients = np.polyfit(temperature, voltage, 2)
a, b, c = coefficients        #  V(T) = a*T^2 + b*T + c

# Function return
def sensitivity_function(T):
    return a * T**2 + b * T + c

# Plot the data points
plt.scatter(temperature, voltage, label='Data')

# Plot the fitted curve
T_range = np.linspace(min(temperature), max(temperature), 100)
plt.plot(T_range, sensitivity_function(T_range), color='red', label='Fitted Curve (Quadratic)')
plt.xlabel('Temperature')
plt.ylabel('Estimated Voltage')
plt.title('Voltage vs. Temperature')
plt.legend()

# Show the plot
plt.show()
```

- First, output directory for sensitivity function is defined. Then, load the data from the estimated_voltages.csv file that was the output of the voltage estimation program. We only need temperature and voltage columns.
- As suggested by supervisor, we performed a polynomial of second order (quadratic regression) to identify the sensitivity function. The coefficients is stored in a b c variables.
- Next, using extracted data, we scattered all pairs of temperature-voltage as well as the fitted curve that passes through all of them. The figure is then saved in the defined output directory.

```
#Save the result to a txt file
sensitivity_function_text1 = f"Estimated sensitivity function: V(T) = {a:.4f} * T^2 + {b:.4f} * T + {c:.4f}"
sensitivity_function_text2 = f"Coefficients (a, b, c): {coefficients}"
with open(os.path.join(sf_output_directory, 'Sensitivity_function.txt'), 'w') as file:
    file.write(sensitivity_function_text1 + '\n')
    file.write(sensitivity_function_text2)
```

- Finally, the sensitivity function is also saved in txt file named 'Sensitivity_function.txt'

3. Task division

The second python file, named 'Sensitivity function' was wholly developed by Thi Van Khanh Nguyen.

'Voltage estimation is developed by Anh Khoa Dang and An Quang Nguyen

Anh Khoa Dang is responsible for the following functions: compute_slopes(), allthingscurrent(), plotvoltage()

An Quang Nguyen is responsible for the following functions: savedata(), create_readme(), plotcurrent(), main()

Although each member is mainly responsible for the mentioned task, the general plan of development that comes before any function is written is discussed by all members, and every problem a member has during the development is collectively solved by the team.

Imports from different libraries and the initial list, as well as output directory are discussed and implemented by both 3 people.

4. Conclusion

In conclusion, it was a challenging task. We did not have a background in electrical engineering, so it was also a problem at first to understand the tasks. Furthermore, at the beginning, the stable phase of the current was not identified correctly, leading to voltage not behaving linearly. This leads to linear regression being unable to be performed, causing quite confusion at first. That being said, this was a very valuable learning experience for our team. In the end, we were able to overcome the challenges and complete the assignment.