Name : Aman Sharma
Roll No : 2021010

WN_Assignment_4

1)

```
# code to find the amplitude and phase of every subscarrier
def amp_phase(df):
    from math import sqrt, atan2
    amp = []

    d = np.array(df)
    for j in range(len(d)):
        imaginary = []
        real = []
        amplitudes = []

        for i in range(len(d[j])):
            if i % 2 == 0:
                imaginary.append(d[j][i])
            else:
                real.append(d[j][i])

        for i in range(int(len(d[0]) / 2)):
            amplitude = sqrt(imaginary[i]**2 + real[i]**2)


            amplitudes.append(amplitude)

        amp.append(amplitudes)

    amp = pd.DataFrame(amp)
    amp = amp.reset_index(drop=True)

    print("amp_phase completed")

    return amp
```

Here is the code for computed Amplitude.

I used the formula

amplitude = sqrt(imaginary[i]**2 + real[i]**2)

2)

```
# call Here
out_rem_amp=hample_filter(Normalized_amp)
```

58] ✓ 2.9s

```
hample_filter completed
```

```
def denoise(df):
    dwt = pd.DataFrame()

    for i in range(len(df.iloc[0])):
        signal = df[i]
        coeff = pywt.wavedec(signal, wavelet='db4', mode="per")
        #coeff
        d = np.mean(np.absolute(coeff[-1] - np.mean(coeff[-1], axis=None)), axis=None)
        sigma = (1/0.6475) * d
        #sigma
        uthresh = sigma * np.sqrt(2 * np.log(len(signal)))
        #uthresh
        coeff[1:] = (pywt.threshold(i, value=uthresh, mode='hard') for i in coeff[1:])
        filter = pywt.waverec(coeff, wavelet='db4', mode='per')
        #filter1 = pd.DataFrame(filter)
        dwt[i]= filter
    dwt = dwt[:-1]
    print("denoise completed")
    return dwt
```

59] ✓ 0.2s

```
# Call Here
denoised_amp=denoise(out_rem_amp)
print(denoised_amp.shape)
```

60] ✓ 0.1s

```
denoise completed
(41595, 114)
```

I used the normalized data in hample_filter and then used that dataframe in denoised function.

Here are some of the values of the denoised data.

```
    # Call Here
    denoised_amp=denoise(out_rem_amp)
    print(denoised_amp)

 ✓  0.2s
```

```
Output exceeds the size limit. Open the full output data in a text editor
denoise completed
           0         1         2         3         4         5         6    \
0      0.248575  0.252260  0.247869  0.263384  0.260823  0.235170  0.239478
1      0.248477  0.252163  0.247704  0.263228  0.260660  0.234755  0.239048
2      0.248373  0.252060  0.247530  0.263064  0.260489  0.234318  0.238595
3      0.248277  0.251965  0.247369  0.262912  0.260330  0.233919  0.238182
4      0.248194  0.251884  0.247233  0.262782  0.260195  0.233588  0.237839
...         ...       ...       ...       ...       ...       ...       ...
41590  0.249103  0.252782  0.248757  0.264222  0.261700  0.237414  0.241804
41591  0.248944  0.252625  0.248487  0.263967  0.261433  0.236726  0.241090
41592  0.248856  0.252537  0.248338  0.263827  0.261286  0.236349  0.240699
41593  0.248784  0.252467  0.248218  0.263713  0.261167  0.236046  0.240385
41594  0.248755  0.252438  0.248170  0.263668  0.261120  0.235928  0.240264

           7         8         9    ...       104       105       106  \
0      0.236249  0.250526  0.243641  ...  0.299805  0.274300  0.273218
1      0.235759  0.250079  0.243180  ...  0.299671  0.274125  0.273040
2      0.235242  0.249608  0.242694  ...  0.299530  0.273941  0.272852
3      0.234772  0.249178  0.242250  ...  0.299403  0.273775  0.272682
4      0.234383  0.248820  0.241881  ...  0.299299  0.273640  0.272544
...         ...       ...       ...  ...       ...       ...       ...
41590  0.238903  0.252946  0.246138  ...  0.300525  0.275242  0.274179
41591  0.238088  0.252202  0.245370  ...  0.300307  0.274956  0.273888
41592  0.237642  0.251796  0.244951  ...  0.300186  0.274799  0.273727
41593  0.237284  0.251469  0.244614  ...  0.300089  0.274671  0.273597
...
```

3)

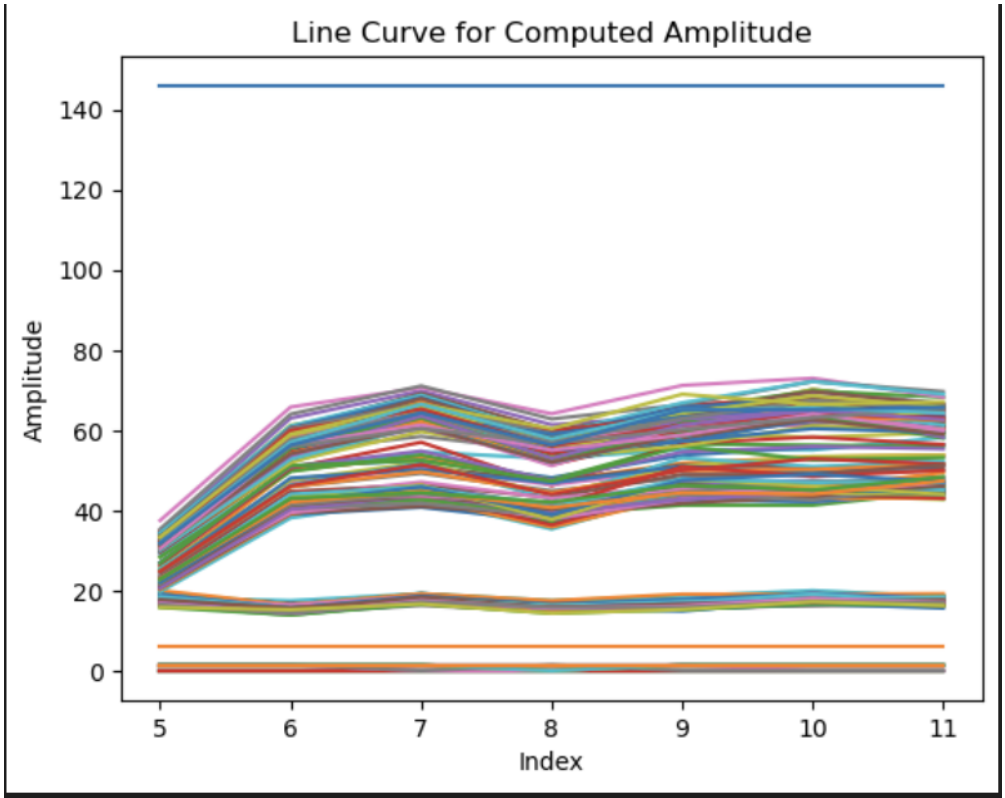
For visualization,

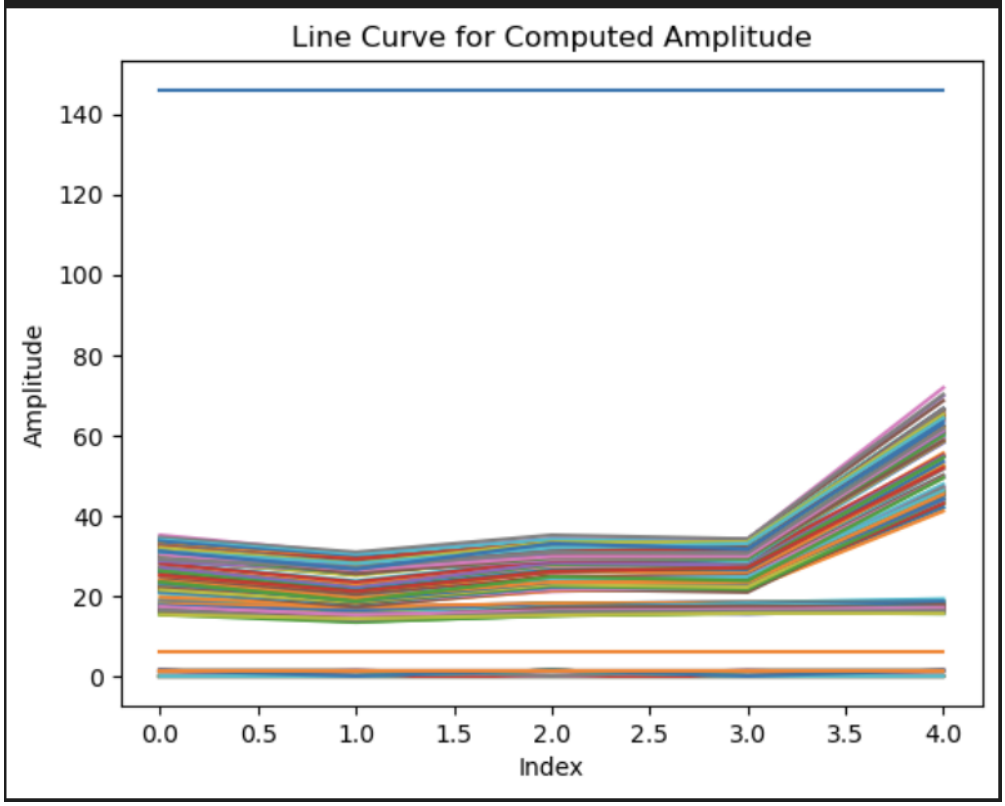
I tried different things, Here is the basic code for the plot. You can change the values in square bracket to choose rows and column accordingly and can visualize the data.

```
# Write code to display the line curve for computed amplitude.



# Create the line plot
plt.plot(Normalized_amp[5:12])

# Label the axes
plt.xlabel('Index')
plt.ylabel('Amplitude')

# Set the title
plt.title('Line Curve for Computed Amplitude')

# Show the plot
plt.show()
plt.plot(Normalized_amp[:5])

# Label the axes
plt.xlabel('Index')
plt.ylabel('Amplitude')

# Set the title
plt.title('Line Curve for Computed Amplitude')

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

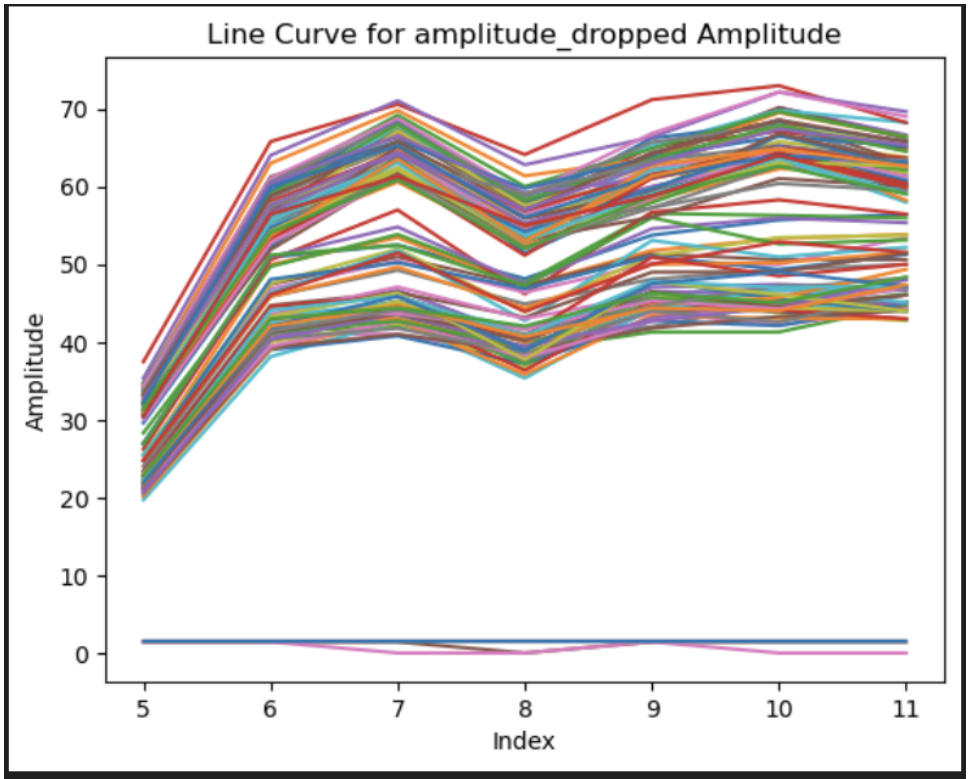
Like

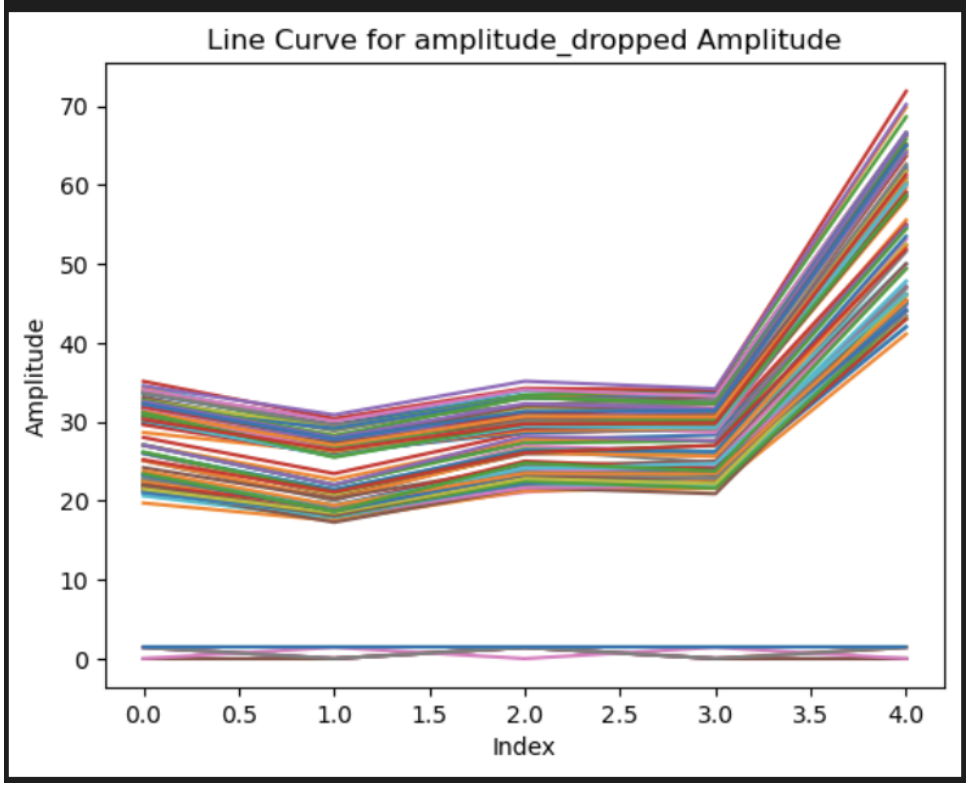
First,


Line Curve for Computed Amplitude
Amplitude
Index
0
20
40
60
80
100
120
140
5
6
7
8
9
10
11

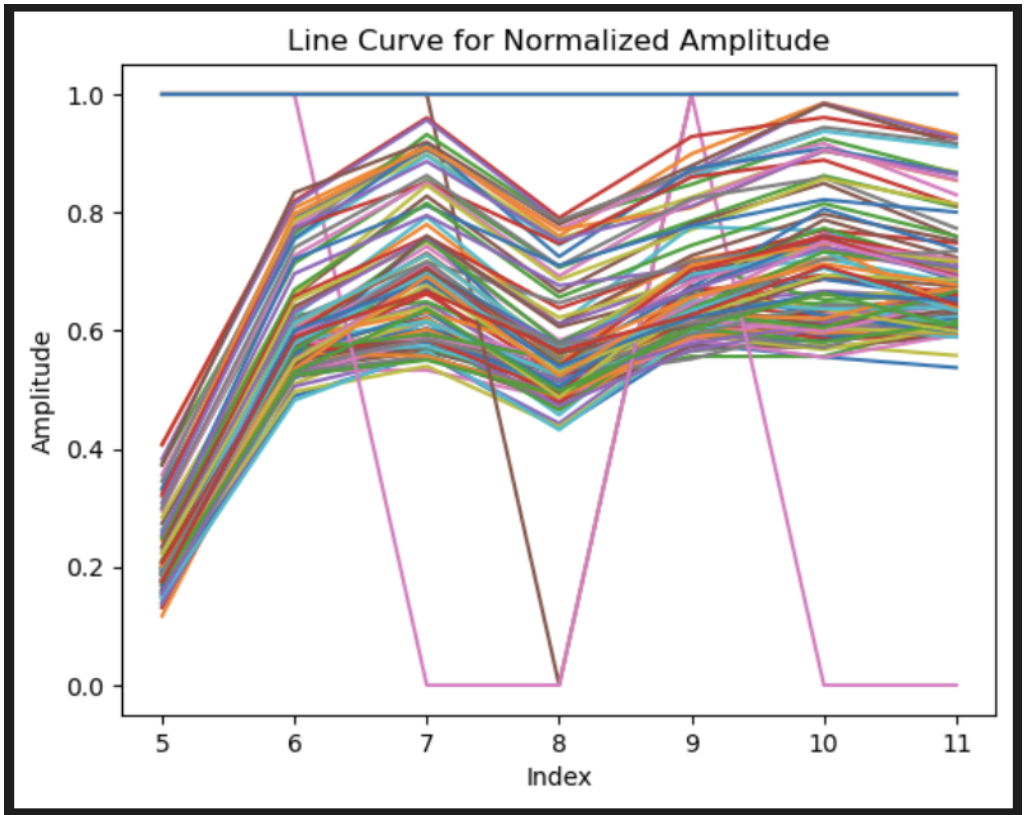

Line Curve for Computed Amplitude
Amplitude
Index
0
20
40
60
80
100
120
140
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

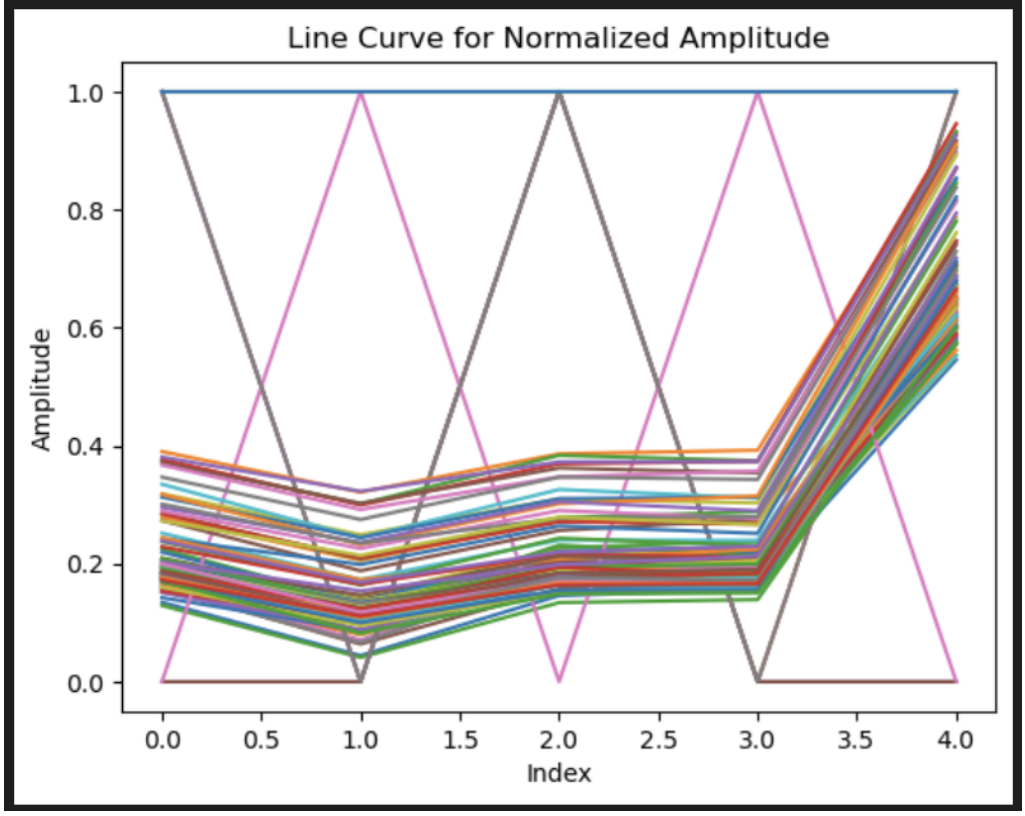
Second,


Line Curve for amplitude_dropped Amplitude
Amplitude
Index
0
10
20
30
40
50
60
70
5
6
7
8
9
10
11

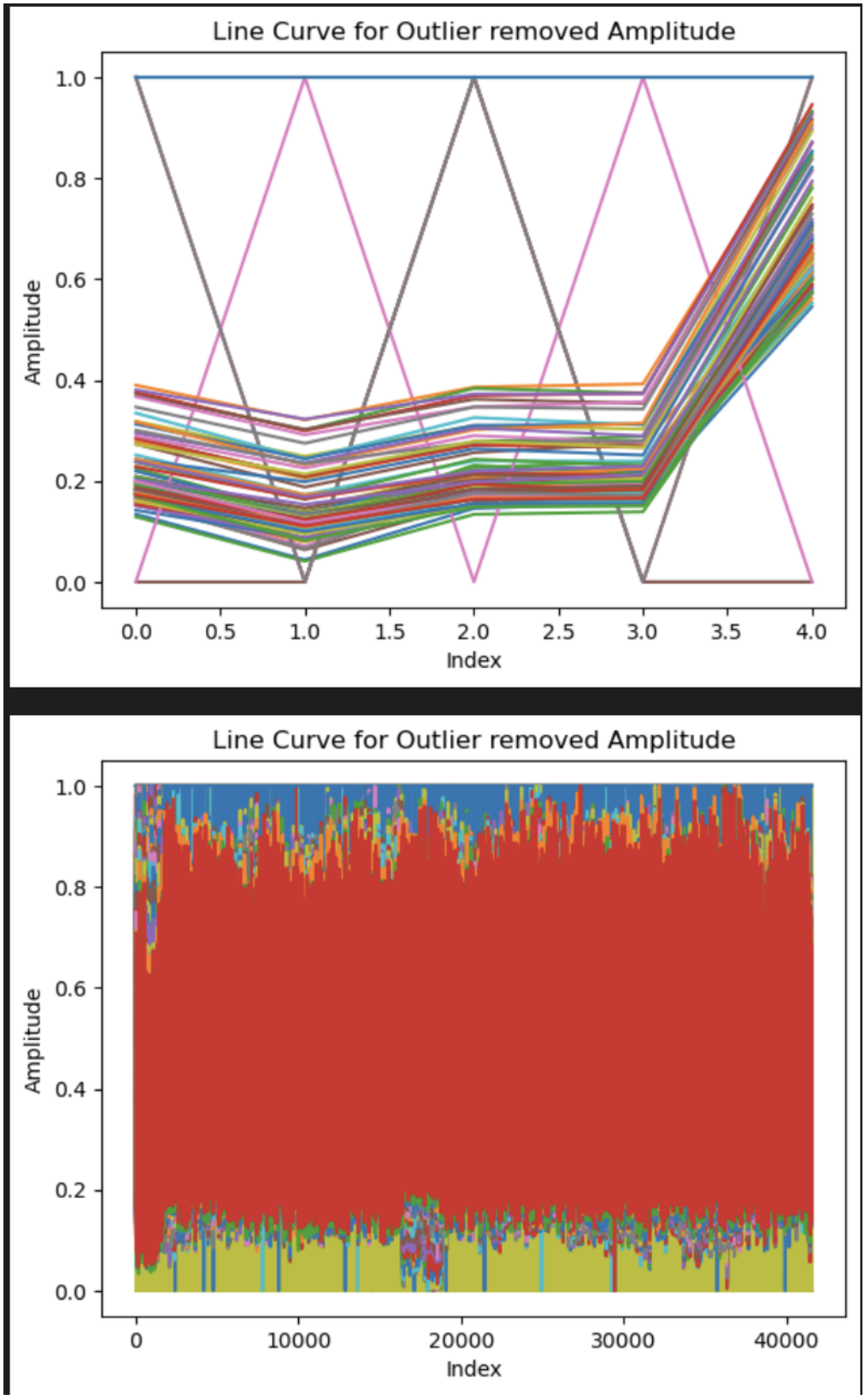

Line Curve for amplitude_dropped Amplitude
Amplitude
Index
0
10
20
30
40
50
60
70
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

Third,

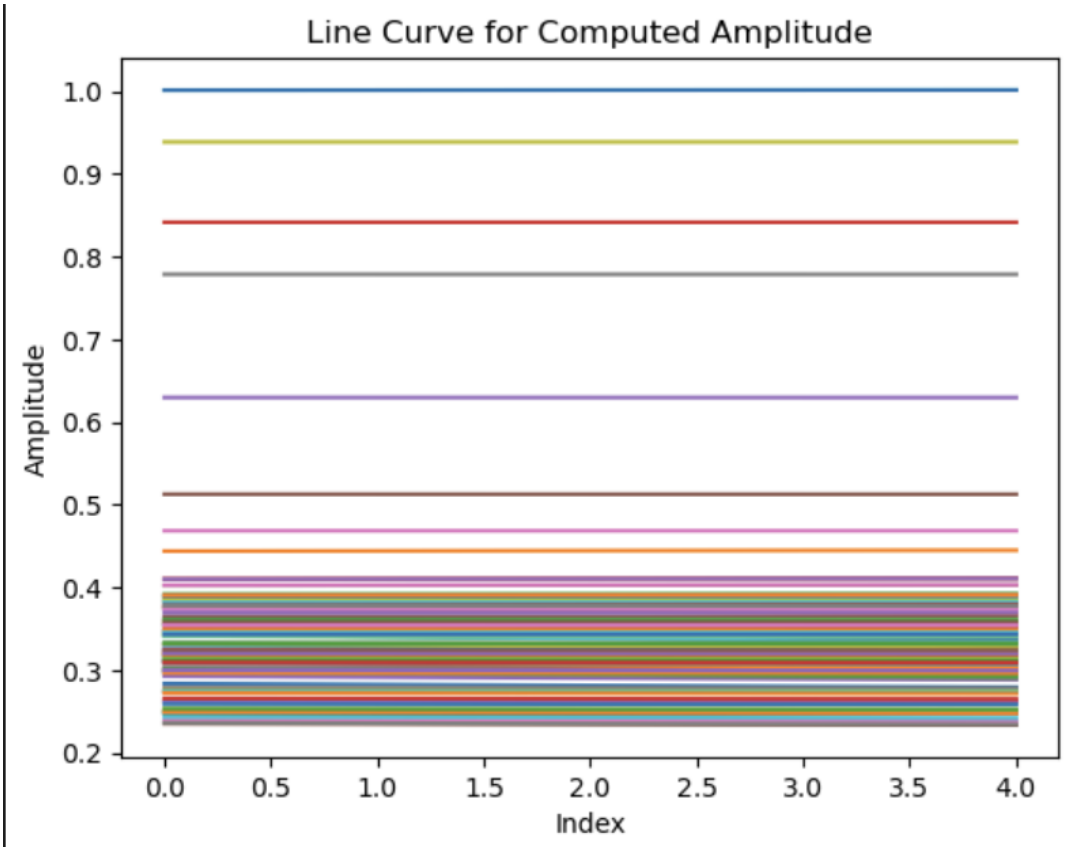

Line Curve for Normalized Amplitude
Amplitude
Index
1.0
0.8
0.6
0.4
0.2
0.0
5
6
7
8
9
10
11

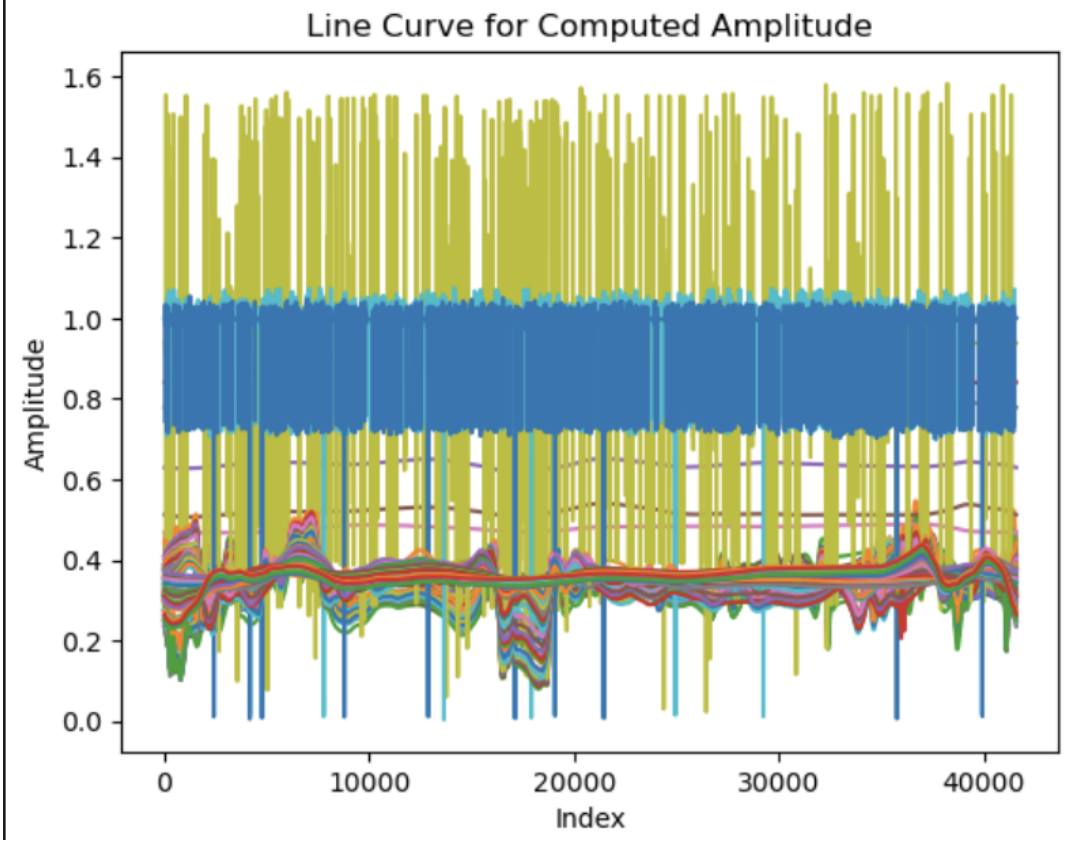

Line Curve for Normalized Amplitude
Amplitude
Index
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

Fourth,

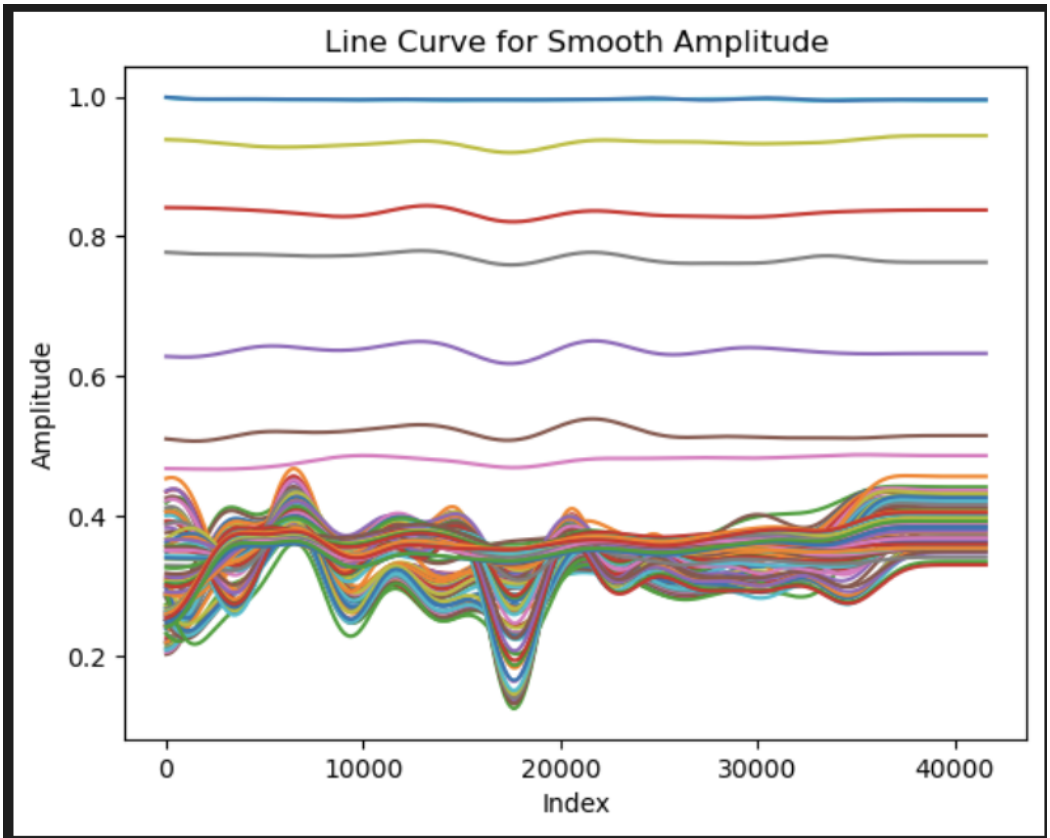

Line Curve for Outlier removed Amplitude
Amplitude
Index
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Line Curve for Outlier removed Amplitude
Amplitude
Index
1.0
0.8
0.6
0.4
0.2
0.0
0
10000
20000
30000
40000

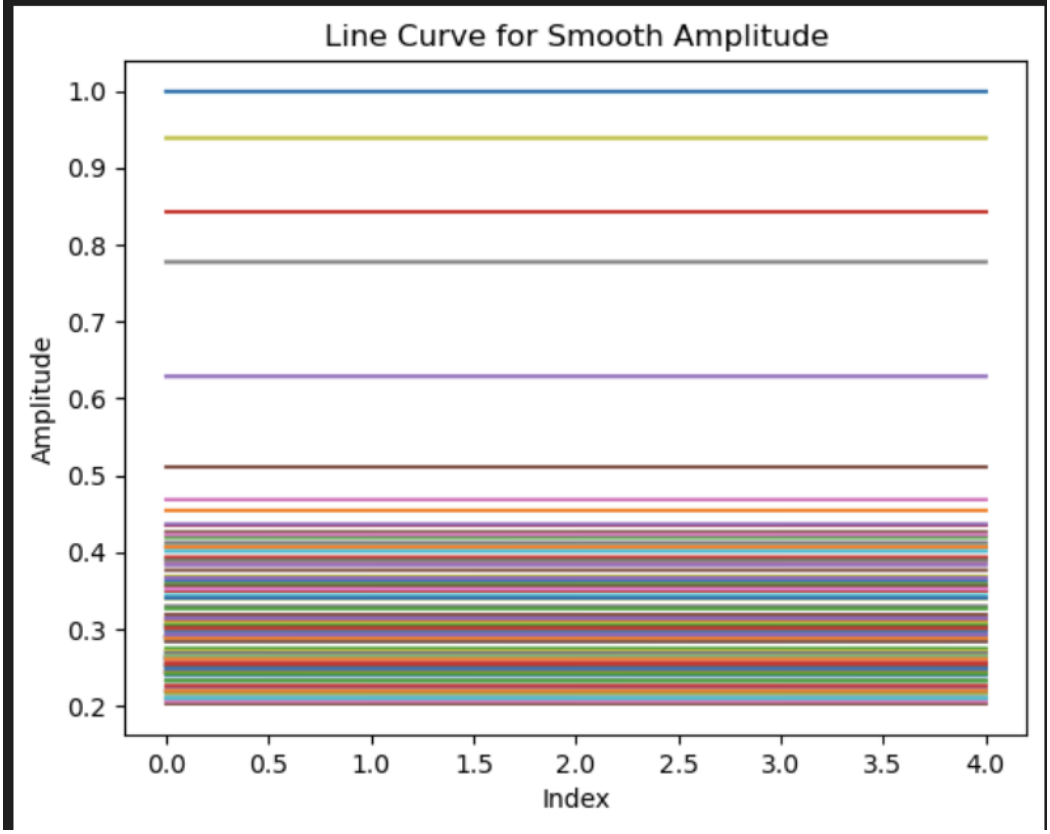
Fifth,


Line Curve for Computed Amplitude
Amplitude
Index
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Line Curve for Computed Amplitude
Amplitude
Index
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
10000
20000
30000
40000

Sixth,


Line Curve for Smooth Amplitude
Amplitude
1.0
0.8
0.6
0.4
0.2
0
10000
20000
30000
40000
Index


Line Curve for Smooth Amplitude
Amplitude
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Index

4)
For proper dimensioning, I used PCA and then reshaped it to (-1,10,10)

```
# Make input shape as:
# First reduce the column dimension to 100
# Then make reshape to 10 X 10.

from sklearn.decomposition import PCA
import pandas as pd

X = smooth_amp.iloc[:, :114].values


n_components = 100
pca = PCA(n_components=n_components)
X_pca = pca.fit_transform(X)
df_reduced = pd.DataFrame(data=X_pca)
```

✓ 0.4s

```
df_reduced.shape
ten_data=df_reduced.values.reshape(-1,10,10)
print(ten_data.shape)
```

✓ 0.2s

5)

My engagement score for
mayank_ISA.csv is 0.75
mayank_relation.csv is 0.42