# Novel and Performant Quench Analysis Tools for Superconducting Magnets

Samarth Chitgopekar & Shreekar Earanti

QuarkNet Presentations

2 August 2022

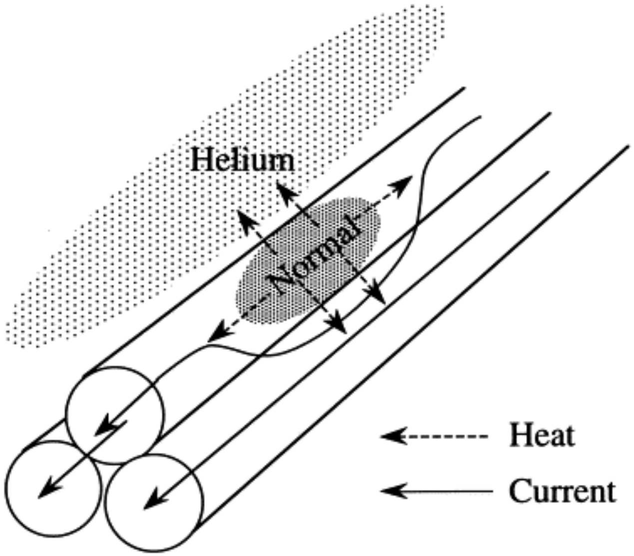
# Presentation Outline

- Background & Significance
- Problem
- Data
- Teamwork
- Solution Overview
- Previous Work
- Back end Challenges
- Front end Challenges
- Documentation
- Future Expandability
- Summary

# Background

- Particle accelerators rely on superconductors to function
- Superconducting magnets are electromagnets and can undergo *quenches*
- A *quench* is a process where a heat source in the magnet coils causes a portion of them to become resistive
- A resistive superconductor isn't a superconductor at all
  - It is key to understand how to prevent quenches, as they are *irreversible*

Quench visualization in superconducting cables




source: https://ars.els-cdn.com/content/image/1-s2.0-S001122759800006X-gr1.gif

# Significance & Motivation

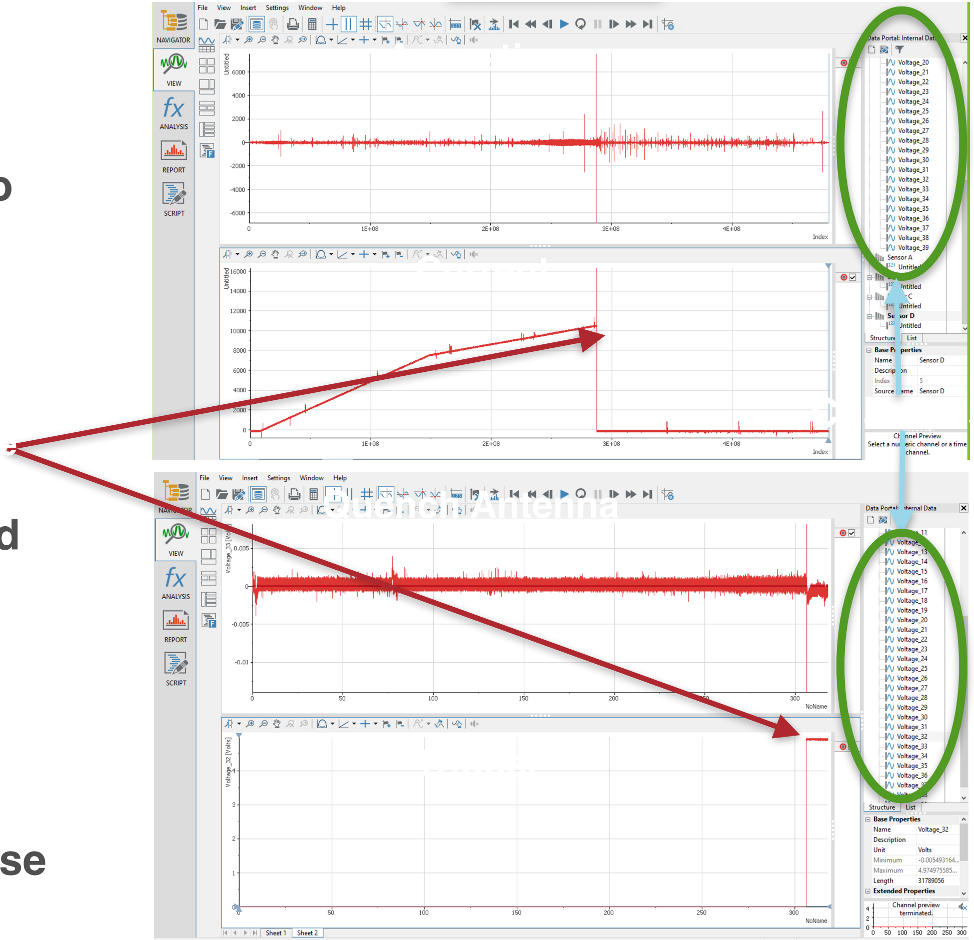
- *Quenches* affect both time and resources
  - Superconducting magnets need time to recover after a quench
    - Can take minutes to hours
  - More liquid helium is required for the superconducting magnet after a quench
    - Can become very expensive over time as more helium is needed
- Although general causes of quenches are known, specific causes are difficult to find
- These issues call for an accurate analysis of solutions to understand more about *quenches*, what might cause them to occur, and how they might be prevented.

# Problem

- **Previous research has used machine learning models to try to understand and predict quenches**
- **There is a need to understand *what* specifically caused a particular quench**
- **Scientists at FNAL have a new quality, amount, and type of data from sensors surrounding magnets as they undergo quenches**
  - **Lack the tools to adequately analyze it**

# Data

- TDMS Files
  - Specialized data file format, difficult to interact with, but very optimized
- Acoustic Data
- Current Data
- Quench Antenna Data
  - Detect magnetic field deviation caused by a change in current from a quench
- Trigger Data
- Graphs displayed in DIAdem
  - Doesn't have features we need for in-depth analysis - Our solution fixes these needs

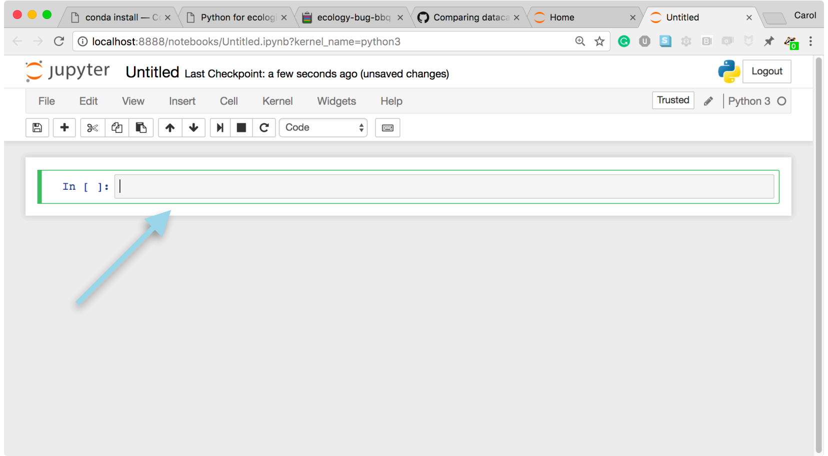


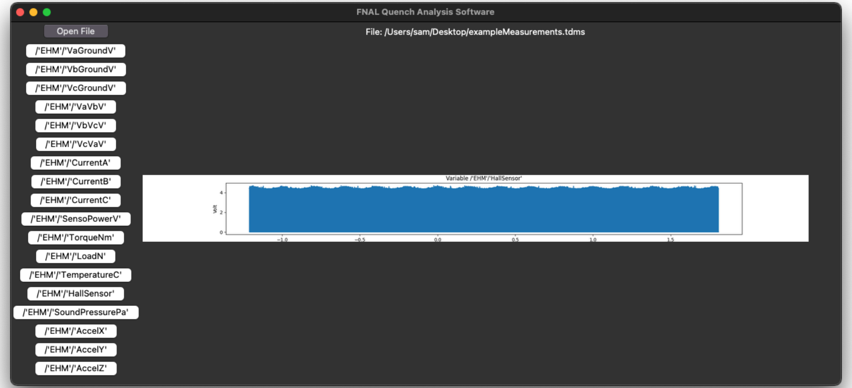
# Working as a team

- Split work based on strengths
  - Shreekar: Data Analysis & Visualization
  - Sam: Frontend Tools
- Met every other day to communicate updates and process on the corresponding assignments
- Prioritizing which features from the backend to implement in the frontend
- Worked together if there were challenges that arose on either end of work
- Combined our two parts for a full fledged tool

# Solution to understanding new Quench Data

- Python Programming Language based data visualization and analysis tool
- Jupyter Notebook
  - Web application for running Python code and functions in 'cells', or 'notebook' format allowing for flexibility and easy of use in analysis and visualization
  - Granular control over specific variables and functions
- TKinter Desktop Python Application
  - Easy to use GUI
  - Mass-Deployable






Fermilab

## Back end Solution Overview

Back end = Functions for data processing/analysis called upon by frontend when interacted with in the Graphical user interface

Backend can act independently in Jupyter Notebooks.

- TDMS File Opener
- Channel Viewer
- Time Frame Control
- Multiple Channel Viewer
- Zero & Smoothing
- TDMS File to CSV File Download

## Front end Solution Overview

Front end = Call backend logic as needed and display data to end user via a Graphical User Interface

- Native TDMS file selector
- Display channels
- Show normalized channel plot
- Packaged as a no-code desktop application

## Connection

Front end *calls* on Back end functions for material actions.

# Literature Review

- Sujay (2020)
  - Examined one specific feature of the acoustic data, the strengths of acoustic events to minimum quench energy (MQE) (theoretical prediction).
  - Looked for an understanding of why certain events found in the acoustics trigger a quench while others do not.
- Kiernan (2021)
  - Built various analysis tools to help understand the measured sensor data.
    - Created an event detection tool that takes a ON and OFF threshold from the user to isolate a quench event, among other utilities.

# Opening TDMS Files

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from nptdms import TdmsFile
import random
import string
from IPython import display
```

```python
def open_file_list(path):
    with TdmsFile.read(path) as tdms_file:
        all_groups = tdms_file.groups()

    all_groups = tdms_file.groups()
    data_frame = tdms_file.as_dataframe()
    cLen = len(data_frame.columns)
    return data_frame.columns.values.tolist()
```

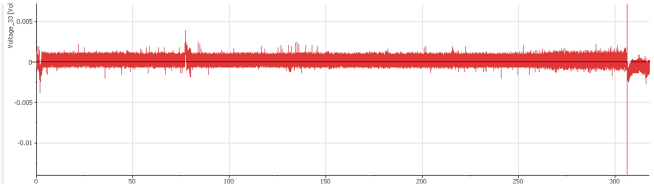
```python
def open_file_df(path):
    with TdmsFile.read(path) as tdms_file:
        all_groups = tdms_file.groups()

    all_groups = tdms_file.groups()
    data_frame = tdms_file.as_dataframe()
    cLen = len(data_frame.columns)
    return data_frame
```

- NPTDMS Package in Python
- Two Functions
  - Opening a TDMS file channels into a list
  - Opening TDMS files into a DataFrame
- Reused in other functions
- Tested with ten test TDMS files with varying TDMS file structures
  - Previous tools were not compatible with significant variations in TDMS files
- First step in a larger process
  - Opening files is critical 1st step in data analysis

# Time Adjustment

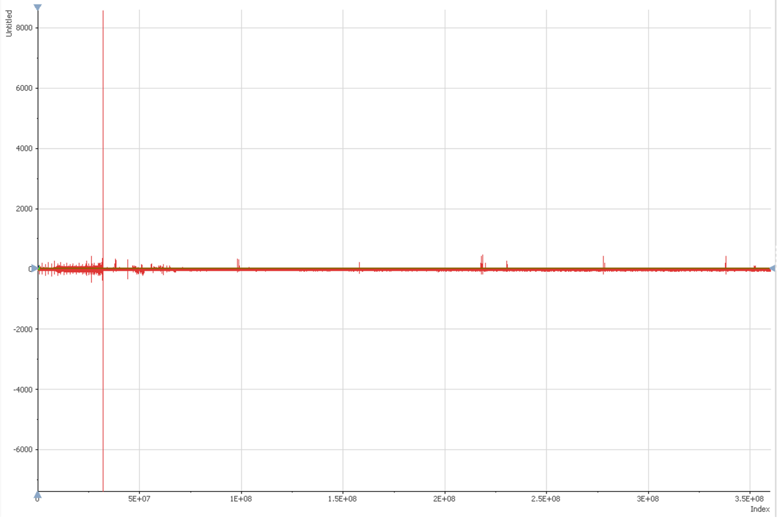
- Data is recorded 100 to 1000000 times per second based on sensor(can be adjusted)
  - Quench Antenna Sensor: 100,000 Data points per second
  - Acoustic Sensor: 1,000,000 Data points per second
- Data is adjusted to match in seconds
- Function works whether time has been previously normalized or not
  - Functions independently

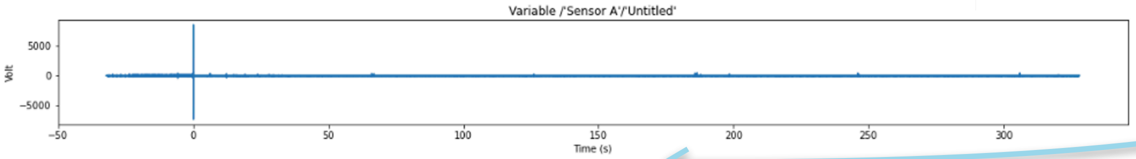



For example: Original time data is multiplied by 0.000001(0.000001=1e-6) because 1,000,000 samples are collected every second from this acoustic sensor

1,000,000*0.000001=1

```
time_range = np.multiply(time_range, 1e-6, out=time_range, casting="unsafe")
```



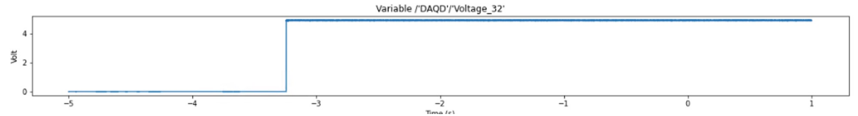


Fermilab

# Zeroing

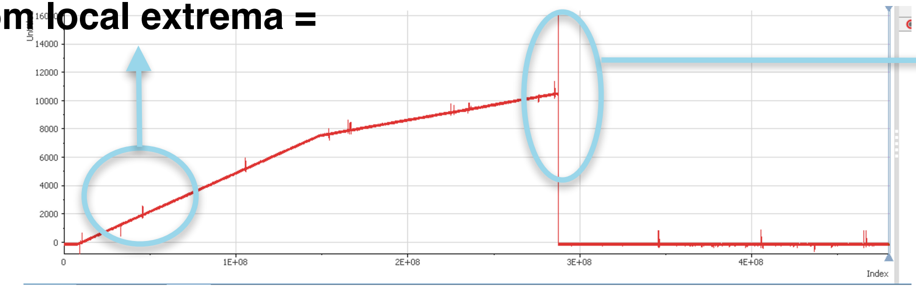
## Trigger Data

```
: dataoutputTimeRange("/'DAQD'/'Voltage_32'", r"D:\QA\MBHSM03\DAQD_MBHSM03_20220302_002\AllCards.tdms", -5, 1)
```




## Current Data

**Random local extrema = Noise**




**Distinct spike = Quench detection time**

- Can't use standard derivative analysis because of noise
- Upon filtering out the noise, we can use the derivatives to isolate the quench

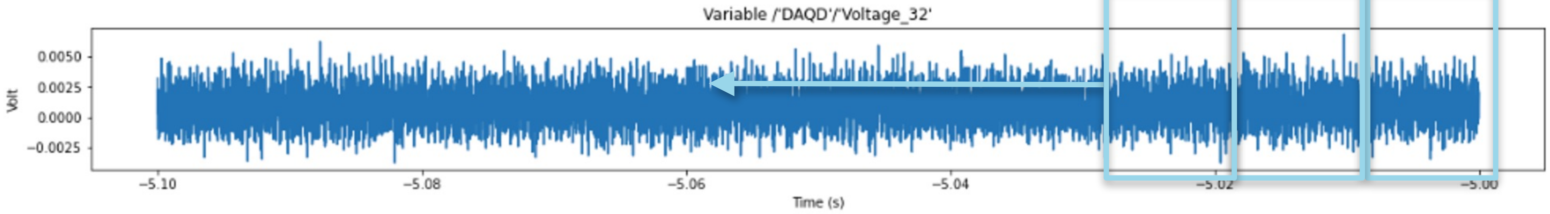
# Data Smoothing

- Window needs to be large enough to encapsulate enough random noise for a consistent average
- Window still needs to be relatively small to preserve the accuracy of the 0 (target: ~1ms).

**Windows of fixed-time Δt**

**Trigger Data**

```
dataoutputTimeRange("/'DAQD'/'Voltage_32'", r"D:\QA\MBHSM03\DAQD_MBHSM03_20220302_002\AllCards.tdms", -5.1, -5)
```




- Create several fixed-time windows
- Replace all of the measurements inside the window with the average value over the window
- With the new 'smoothed' plot, we can use derivative analysis to identify the quench time
- **Non-manual tool not previously available**

**Distinct spike = Quench**



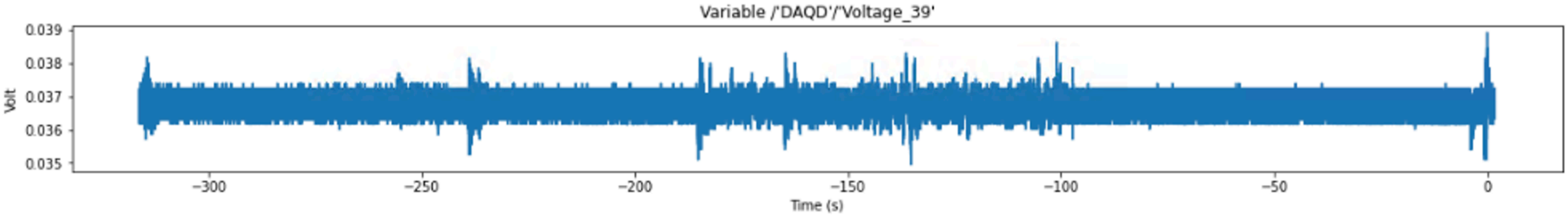
Fermilab

# TDMS File Channel Viewing

```
def dataoutput(channel, path):
    ##user will select for example "/'Untitled'/'PXI1Slot8/ai3'"
    ##we then select that column from df, and output it
    data_frame=open_file(path)
    df2 = data_frame[[channel]]
    #Add time axis relative to the time when quench happens
    time_range = np.asarray(range(df2.shape[0]))
    #Center around the max value (quench happens at 0 time)
    max_index = np.argmax(df2)
    time_range -= max_index
    time_range = time_range.astype('float32')
    #Multiply by datarate
    time_range = np.multiply(time_range, 1e-5, out=time_range, casting="unsafe")
    ##loading time into the existing df
    df2 = pd.DataFrame(data = {channel: df2[channel],"time": time_range})
    startTime = min(df2["time"])
    endTime = max(df2["time"])

    df2 = df2[(df2["time"] > startTime) & (df2["time"] < endTime)]
    plt.figure(figsize=(20,2))
    plt.plot(df2["time"], df2[channel])
    plt.xlabel("Time (s)")
    plt.ylabel("Volt")
    #plt.ylim([-1.5,1.5])
    #plt.xlim([-650,100])
    plt.title("Variable {}".format(channel))
    plt.show()
    ##imgPath=(''.join(random.choices(string.ascii_lowercase, k=5)))
    ##plt.savefig("{}.svg".format(imgPath), format="svg")
    ##return imgPath
    return plt
```

- **File path and specific channel name passed in**
- **Zero Algorithm finds Quench and centers graph**
- **Graph is plotted**
- **Data can also be plotted without being normalized around the Quench**



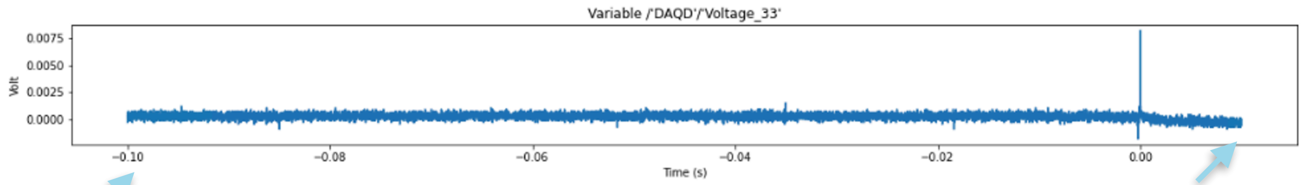

**Full data time range plotted from a Quench Antenna Sensor**

# Time Frame Control in Graphing

- Initial steps identical to normal graphing
- Starting time and ending time in seconds passed in
- Graph is created from the starting value to end value
- Milli/microseconds can also be inputted for precision
  - Events occur in the order of milliseconds

```
def dataoutputTimeRange(channel, path, startTime, endtime):
    ##user will select for example "/'Untitled'/'PXI1Slot8/ai3'"
    ##we then select that column from df, and output it
    data_frame=open_file(path)
    df2 = data_frame[[channel]]
    #Add time axis relative to the time when quench happens
    time_range = np.asarray(range(df2.shape[0]))
    #Center around the max value (quench happens at 0 time)
    max_index = np.argmax(df2)
    time_range -= max_index
    time_range = time_range.astype('float32')
    #Multiply by datarate
    time_range = np.multiply(time_range, 1e-5, out=time_range, casting="unsafe")
    ##loading time into the existing df
    df2 = pd.DataFrame(data = {channel: df2[channel],"time": time_range})
    startTime = startTime
    endTime = endtime
    df2 = df2[(df2["time"] > startTime) & (df2["time"] < endTime)]
    plt.figure(figsize=(20,2))
    plt.plot(df2["time"], df2[channel])
    plt.xlabel("Time (s)")
    plt.ylabel("Volt")
    #plt.ylim([-1.5,1.5])
    #plt.xlim([-650,100])
    plt.title("Variable {}".format(channel))
    plt.show()
    ##imgPath=(''.join(random.choices(string.ascii_lowercase, k=5)))
    ##plt.savefig("{}.svg".format(imgPath), format="svg")
    ##return imgPath
    return plt
```

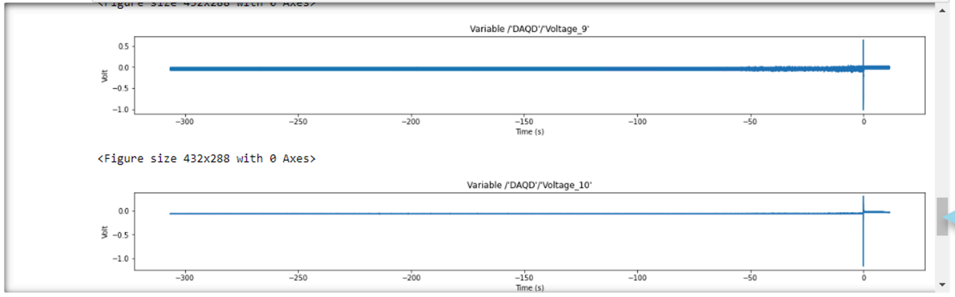
```
dataoutputTimeRange("/'DAQD'/'Voltage_33'", r"D:\QA\MBHSM03\DAQD_MBHSM03_20220302_002\AllCards.tdms", -0.1, 0.01)
```



# Multiple Sensor Channel Viewer

```
def multiDataOutput(path):
    cList = open_file2(path)
    data_frame=open_file(path)
    pathList = []
    for item in cList:
        df2 = data_frame[[item]]
        #Add time axis relative to the time when quench happens
        time_range = np.asarray(range(df2.shape[0]))
        #Center around the max value (quench happens at 0 time)
        max_index = np.argmax(df2)
        time_range -= max_index
        time_range = time_range.astype('float32')
        #Multiply by datarate
        time_range = np.multiply(time_range, 1e-5, out=time_range, casting="unsafe")
        ##Loading time into the existing df
        df2 = pd.DataFrame(data = {item: df2[item],"time": time_range})
        startTime = min(df2["time"])
        endTime = max(df2["time"])
        df2 = df2[(df2["time"] > startTime) & (df2["time"] < endTime)]
        plt.figure(figsize=(20,2))
        plt.plot(df2["time"], df2[item])
        plt.xlabel("Time (s)")
        plt.ylabel("Volt")
        plt.title("Variable {}".format(item))
        plt.show()
        imgPath=(''.join(random.choices(string.ascii_lowercase, k=5)))
        plt.savefig("{}.svg".format(imgPath), format="svg")
        pathList.append(imgPath)
    return pathList
```

```
In [*]: l2 = multiDataOutput(r"D:\QA\MBHSM03\DAQD_MBHSM03_20220302_002\AllCards.tdms")
        for i in l2:
            display.Image(i)
```




- Initial steps identical
- Each graph is assigned random path and saved
- Loop opens each image path and displays it
- Allows for multiple sensor channels of data to be displayed at once allowing for further data analysis
- Time Frame can be changed for all plots at once
- Tool not previously available

# TDMS File to CSV File Download

```
def data_CSV_download(channel, path):
    data_frame=open_file(path)
    df2 = data_frame[[channel]]
    time_range = np.asarray(range(df2.shape[0]))

    max_index = np.argmax(df2)
    time_range -= max_index
    time_range = time_range.astype('float32')

    time_range = np.multiply(time_range, 1e-5, out=time_range, casting="unsafe")

    df2 = pd.DataFrame(data = {channel: df2[channel],"time": time_range})
    csvPath=(''.join(random.choices(string.ascii_lowercase, k=5)))
    df2.to_csv("{}.csv".format(csvPath))
    return csvPath
```

- **Channel and file path passed in**
- **Data is centered around Quench**
- **Random path assigned to CSV and is saved**
- **Future Expansion: Time Frame manipulation of CSV data**
- **Tool not previously available**

# Back end Solution Challenges

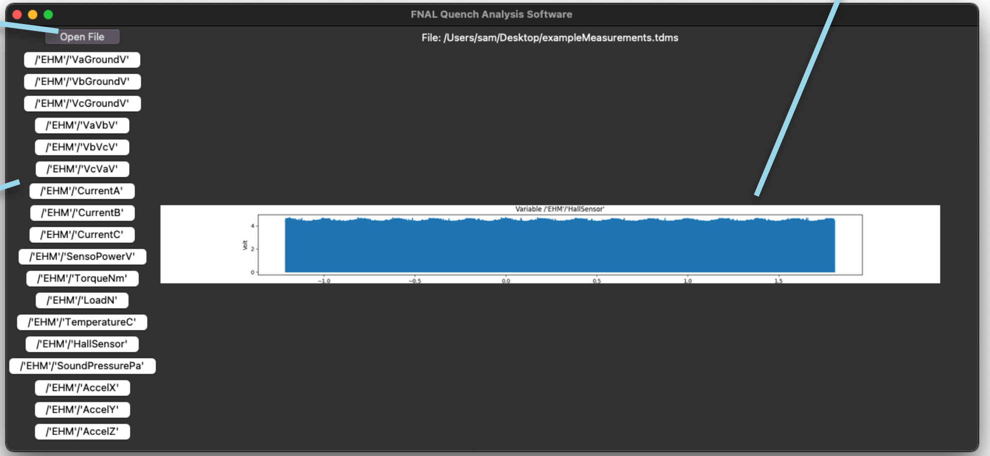
- RAM Capacity Problem
  - When processing larger files over 2.5 Gigabytes, RAM issues arose
    - When a TDMS file was 3 GB, Python was using 22 GB
    - Future Potential Solutions:
      - Batch Processing
      - Data Bit Conversion
      - Data Compression
- TDMS File Variability
  - TDMS Files can vary largely due to their structure flexibility
  - Took us many iterations of functions to create a tool to open all kinds of TDMS files

# Front end Tool

TDMS file selector button

Channel list (clickable buttons to show plot)

Normalized data plot




Instead of interacting with the previously discussed backend via Python, users can access the same functionality via the frontend (clicking instead of coding).

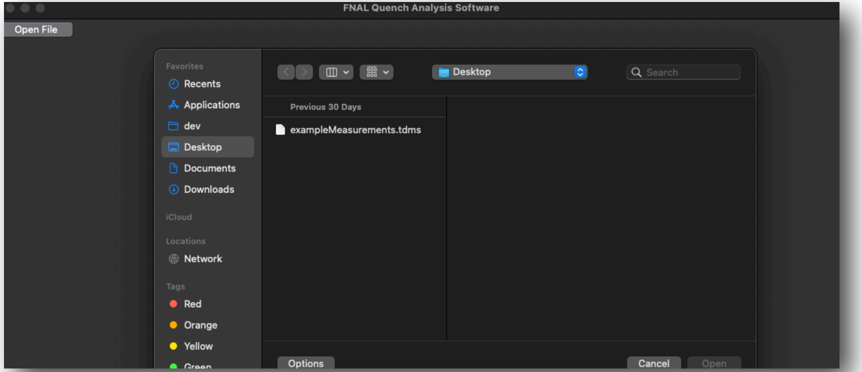
Python and TKinter powered Graphical User Interface

# Front end Challenges

- Initial approach used a local web server to view plot images
  - Very difficult to send large datasets over HTTP
  - Had to write in HTML, JS, CSS, and Python
- TKinter Plot rendering bug when screen sharing
- Validating that a TDMS file existed in the user-provided location before opening
  - Could cause crashes if incorrect paths were provided
- Dynamically rendering the appropriate number of channel selection buttons based on the TDMS file
- Multi-channel/Cross-file channel viewing
  - *Implemented in backend, still in progress for the frontend*

# Front end Solutions

- Desktop application built with TKinter allowed for the entire project (frontend and backend) to stay in Python
  - Improves extensibility for the tool down the line
- Native file selector allowed validation for the files the user selected
  - Ensures existence and that it is a valid TDMS file
- Instead of having to start a web server, users can simply run the desktop application
  - Similar user experience to DIAdem




Sample file selecting workflow


Fermilab

# Package Documentation

- At a high-level, we will be providing a Word Doc that contains information on the following:
  - How to use our tool
    - Building the executable (compiling code into a desktop app)
    - Launching the app
  - How to extend our tool
    - Adding more functionality to the backend
    - Installing dependencies
  - A code walkthrough (explanation) of our current frontends and backends

```
cycler==0.11.0
fonttools==4.34.4
kiwisolver==1.4.4
matplotlib==3.5.2
npTDMS==1.5.0
numpy==1.23.1
packaging==21.3
pandas==1.4.3
Pillow==9.2.0
pyparsing==3.0.9
python-dateutil==2.8.2
pytz==2022.1
six==1.16.0
```

requirements.txt
dependency file

# Future Expansion

- Code foundation can be easily built upon
- Documentation is available for future expansions
- Machine Learning
  - Tools could be utilized to prepare data for machine learning
- Deployment of TKinter desktop application
- Addition of functions to Jupyter Notebook files
- Parallel opening of files for cross analysis

# Summary

- **Problem: *Quenches* are a significant issue with superconducting magnets and require significant resources to resolve, therefore analysis and understanding is critical.**
- **Our Solution: Python based data analysis and visualization tools used with magnet sensor data in order to better understand and analyze *quenches*.**
  - **Backend Functions and Processing Tools**
  - **Frontend Graphical User Interface**
- **Future Developments of our Package:**
  - **Machine Learning and more analysis functions**