RAVATA SOLUTIONS

Summary Report 11/04

James Hizon

Summary of Report

Within this internship, I was able to observe which IDE would be best for Python and SQL. I spent time preprocessing data, used pandas, matplotlib and plotly for data visualization, and integrated Jupyter Notebook with MySQL to store information from the data collected through experimentation with IVF Biotech device at Ravata Solutions.

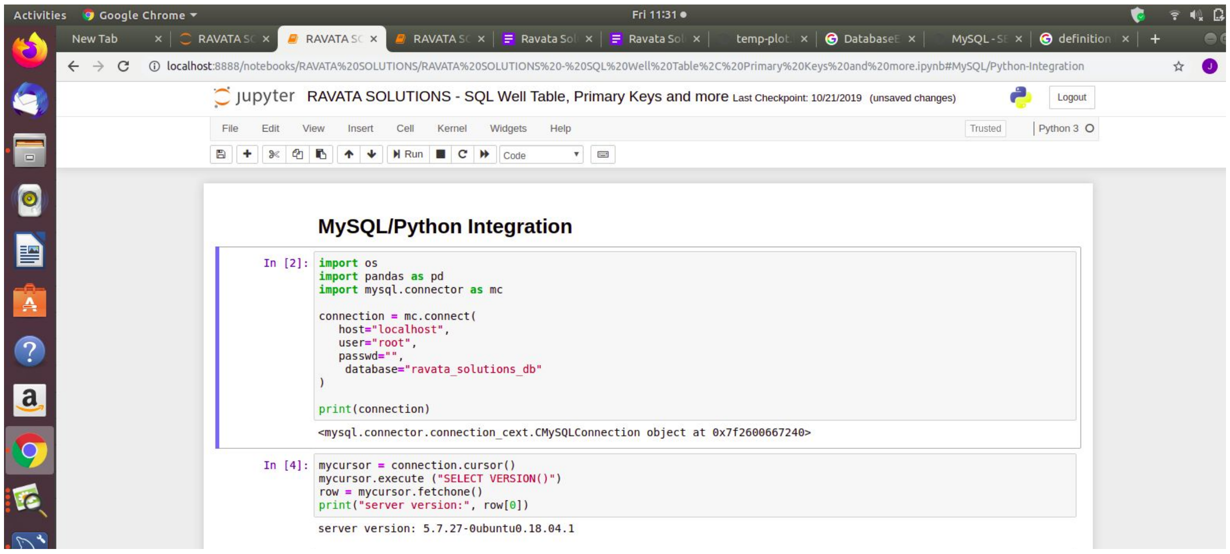
Report Body

Table of Contents:

1. Integrated Development Environments:

The main two IDE's we used in this internship were Jupyter Notebook and MySQL Workbench. Given a programming language, the purpose of an IDE is to allow more in-depth understanding of the code we are working with. Jupyter Notebook is great for note-taking within our code, displaying error messages for debugging, and can split our code into sections for organization. To use Jupyter Notebook after installing, simply type in command prompt: jupyter notebook.

Sample of Jupyter Notebook Usage:

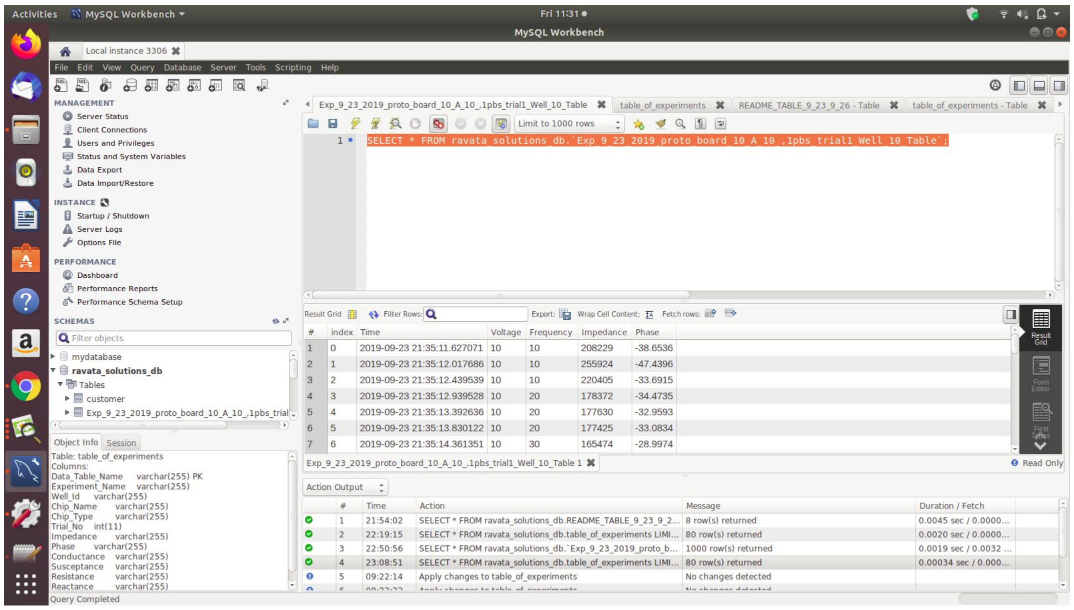

Activities
Google Chrome
Fri 11:31
New Tab
RAVATA SC
Ravata Sol
temp-plot
DatabaseE
MySQL - SE
definition
localhost:8888/notebooks/RAVATA%20SOLUTIONS/RAVATA%20SOLUTIONS%20-%20SQL%20Well%20Table%2C%20Primary%20Keys%20and%20more.ipynb#MySQL/Python-Integration
jupyter RAVATA SOLUTIONS - SQL Well Table, Primary Keys and more Last Checkpoint: 10/21/2019 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

MySQL/Python Integration

```
In [2]: import os
        import pandas as pd
        import mysql.connector as mc

        connection = mc.connect(
           host="localhost",
           user="root",
           passwd="",
            database="ravata_solutions_db"
        )

        print(connection)

        <mysql.connector.connection_cext.CMySQLConnection object at 0x7f2600667240>

In [4]: mycursor = connection.cursor()
        mycursor.execute ("SELECT VERSION()")
        row = mycursor.fetchone()
        print("server version:", row[0])

        server version: 5.7.27-0ubuntu0.18.04.1
```

MySQL Workbench enables us to work with the SQL (Structured Query Language). Here, the user can create their own database to store information, visualize primary key/foreign key relationships via EER diagrams, create and alter tables accordingly, and extract information when needed.

Sample Usage of MySQL Workbench through SELECT * Command:



```
SELECT * FROM ravata_solutions_db.`Exp_9_23_2019_proto_board_10_A_10_,1pbs_trial1_Well_10_Table`;
```

#	index	Time	Voltage	Frequency	Impedance	Phase
1	0	2019-09-23 21:35:11.627071	10	10	208229	-38.6536
2	1	2019-09-23 21:35:12.017686	10	10	255924	-47.4396
3	2	2019-09-23 21:35:12.439539	10	10	220405	-33.6915
4	3	2019-09-23 21:35:12.939528	10	20	178372	-34.4735
5	4	2019-09-23 21:35:13.392636	10	20	177630	-32.9593
6	5	2019-09-23 21:35:13.830122	10	20	177425	-33.0834
7	6	2019-09-23 21:35:14.361351	10	30	165474	-28.9974

#	Time	Action	Message	Duration / Fetch
1	21:54:02	SELECT * FROM ravata_solutions_db.README_TABLE_9_23_9_2...	8 row(s) returned	0.0045 sec / 0.0000...
2	22:19:15	SELECT * FROM ravata_solutions_db.table_of_experiments LIMI...	80 row(s) returned	0.0020 sec / 0.0000...
3	22:50:56	SELECT * FROM ravata_solutions_db.`Exp_9_23_2019_proto_b...	1000 row(s) returned	0.0019 sec / 0.0032 ...
4	23:08:51	SELECT * FROM ravata_solutions_db.table_of_experiments LIMI...	80 row(s) returned	0.00034 sec / 0.000...
5	09:22:14	Apply changes to table_of_experiments	No changes detected	

```
Table: table_of_experiments
Columns:
Data_Table_Name     varchar(255) PK
Experiment_Name     varchar(255)
Well_Id     varchar(255)
Chip_Name        varchar(255)
Chip_Type        varchar(255)
Trial_No    int(11)
Impedance        varchar(255)
Phase       varchar(255)
Conductance    varchar(255)
Susceptance    varchar(255)
Resistance     varchar(255)
Reactance      varchar(255)
```

2. Data Preprocessing:

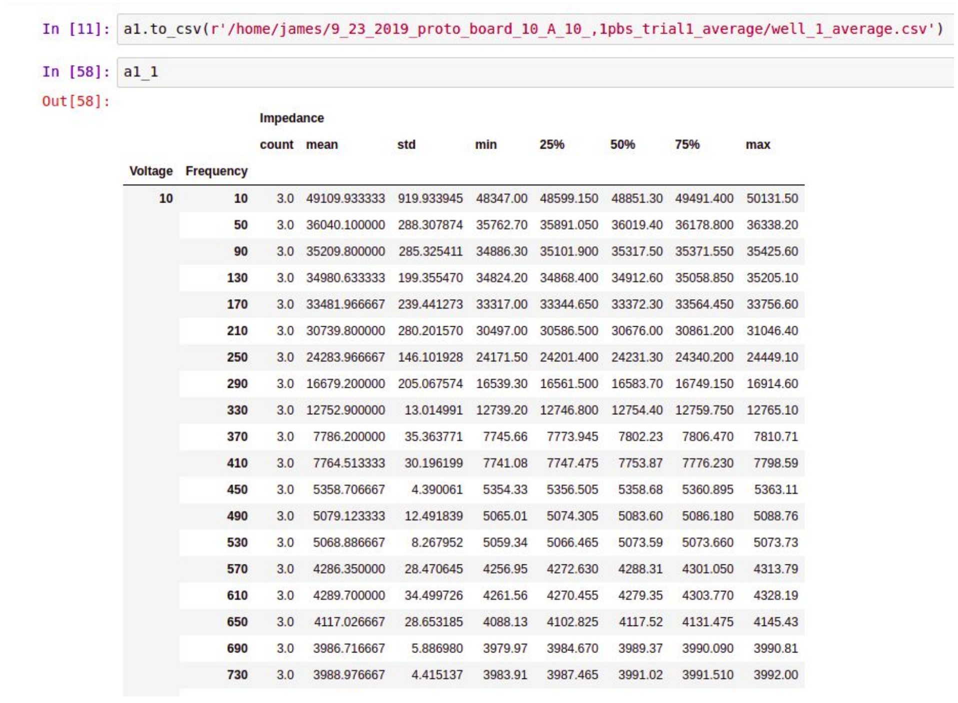
Part of data science involves preprocessing data. If we cannot clean up data, we will not be able to plot accurately. I first began by performing steps on creating a single dataframe, one well of a given feature (i.e., Well 1 for Impedance values). Afterwards, I would create a function, so that I can just speed up the process. I don't recall creating a list, but I know that if I were to create a list, I can just iterate through that list to call the "preprocessing data frame" function. I did this, when I wanted to iterate through multiple folders to create multiple data frames via a list of ".csv" file names.

Data Preprocessing Sample Codes:

I created a function that preprocesses/cleans up data for each ".csv" file using pandas.

```
# Function for Preprocessing Impedance dataframe:
def impedance_df_maker(filename):
    #df = pd.read_csv('Well_1.csv')
    df = pd.read_csv(filename)
    df_drop_step1 = df.drop(['Bias Voltage', 'Phase', 'Time'], axis=1)
    df_drop_step2 = df_drop_step1.dropna(axis=1)
    df = df_drop_step2
    df['Frequency'] = df['Frequency'].map(lambda x: x.rstrip('KHZ'))
    df['Voltage'] = df['Voltage'].map(lambda x: x.rstrip(' MV'))
    df['Frequency'] = pd.to_numeric(df['Frequency'])
    df['Voltage'] = pd.to_numeric(df['Voltage'])
    df = df.groupby(['Voltage','Frequency'])
    return df.describe()

# Call function:
a1_1 = impedance_df_maker('Well_1.csv')
a1.to_csv(r'/home/james/9_23_2019_proto_board_10_A_10_,1pbs_trial1_average/well_1_average.csv'
)
a1_1
```

OUTPUT (Dataframe of Impedance values):



```
In [11]: a1.to_csv(r'/home/james/9_23_2019_proto_board_10_A_10_,1pbs_trial1_average/well_1_average.csv')

In [58]: a1_1
Out[58]:
```

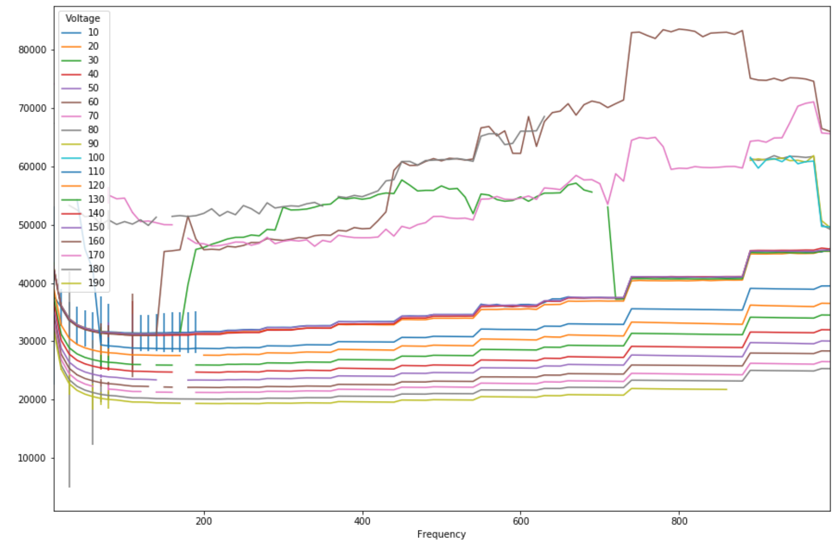
		Impedance							
		count	mean	std	min	25%	50%	75%	max
Voltage	Frequency								
10	10	3.0	49109.933333	919.933945	48347.00	48599.150	48851.30	49491.400	50131.50
	50	3.0	36040.100000	288.307874	35762.70	35891.050	36019.40	36178.800	36338.20
	90	3.0	35209.800000	285.325411	34886.30	35101.900	35317.50	35371.550	35425.60
	130	3.0	34980.633333	199.355470	34824.20	34868.400	34912.60	35058.850	35205.10
	170	3.0	33481.966667	239.441273	33317.00	33344.650	33372.30	33564.450	33756.60
	210	3.0	30739.800000	280.201570	30497.00	30586.500	30676.00	30861.200	31046.40
	250	3.0	24283.966667	146.101928	24171.50	24201.400	24231.30	24340.200	24449.10
	290	3.0	16679.200000	205.067574	16539.30	16561.500	16583.70	16749.150	16914.60
	330	3.0	12752.900000	13.014991	12739.20	12746.800	12754.40	12759.750	12765.10
	370	3.0	7786.200000	35.363771	7745.66	7773.945	7802.23	7806.470	7810.71
	410	3.0	7764.513333	30.196199	7741.08	7747.475	7753.87	7776.230	7798.59
	450	3.0	5358.706667	4.390061	5354.33	5356.505	5358.68	5360.895	5363.11
	490	3.0	5079.123333	12.491839	5065.01	5074.305	5083.60	5086.180	5088.76
	530	3.0	5068.886667	8.267952	5059.34	5066.465	5073.59	5073.660	5073.73
	570	3.0	4286.350000	28.470645	4256.95	4272.630	4288.31	4301.050	4313.79
	610	3.0	4289.700000	34.499726	4261.56	4270.455	4279.35	4303.770	4328.19
	650	3.0	4117.026667	28.653185	4088.13	4102.825	4117.52	4131.475	4145.43
	690	3.0	3986.716667	5.886980	3979.97	3984.670	3989.37	3990.090	3990.81
	730	3.0	3988.976667	4.415137	3983.91	3987.465	3991.02	3991.510	3992.00

The code above was used, when I had to preprocess data for multiple ".csv" files that examines Impedance values through ".describe()" function, and stores data inside another folder. I repeated this process for 10 wells, and then switched to a different working directory.

3. Data Visualization:

Part of the data visualization process involved working with different "plotting" packages within Python. I explored plotting through "matplotlib" and "plotly".

The following is a plot, using the "matplotlib" package.



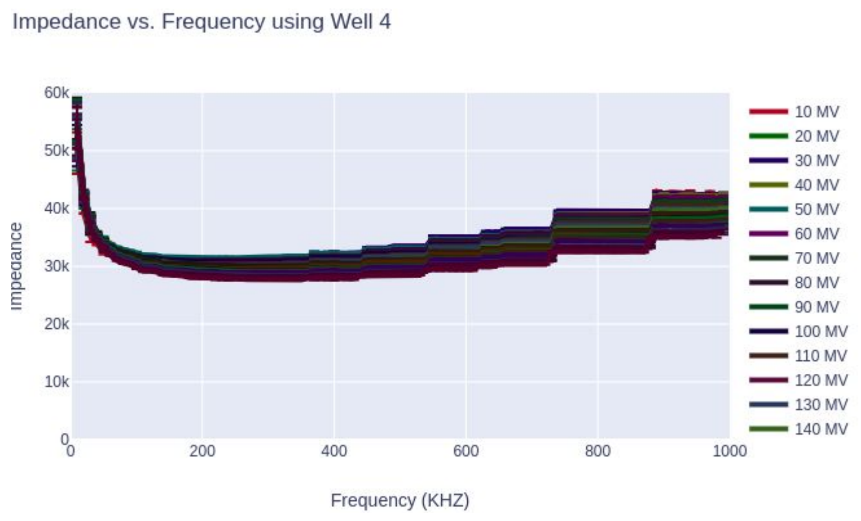
After plotting using “matplotlib” package, I desired to try plotting with another package. From my Udemy course, “Python for Data Science and Machine Learning Bootcamp,” I learned about interactive plotting using plotly.

I first began the data visualization of “plotly” through copy and paste method. I was still not the best with using for loops to iterate with lists. Thus, here is a sample of the code I used to produce one graph using plotly. Here is where I plotted Impedance vs. Frequency values, such that each line is a different measurement of Voltage in MV. The code includes the usage of dataframes from “pandas” package, and working with numpy arrays as x-values to plot the data. In addition, I had to manually change the color for each line, but realized I did not need to do so.

```
# WELL 4 CODE:
fig_well4_impedance = go.Figure(data=go.Scatter(
    x= np.arange(10,1000,10), # MANUALLY CREATED A SERIES OF RANGE 1, 1869. OR # USE
df['col'].value_count
    y= df_b4_Impedance.loc[10, np.arange(10,1000,10)],
  name = '10 MV',
    error_y=dict(
      type='data', # value of error bar given in data coordinates
      array = c4_Impedance_unstacked.stack(),
      visible=True),
  line = dict(color='firebrick', width=4)))

fig_well4_impedance.update_xaxes(range=[0, 1000])
fig_well4_impedance.update_yaxes(range=[0, 60000])
fig_well4_impedance.update_layout(title='Impedance vs. Frequency using Well 4',
            xaxis_title='Frequency (KHZ)',
            yaxis_title='Impedance')
```

```
fig_well4_impedance.add_trace(go.Scatter(
      x= np.arange(10,1000,10), # MANUALLY CREATED A SERIES OF RANGE 1, 1869. OR # USE
df['col'].value_count
      y= df_b4_Impedance.loc[20, np.arange(10,1000,10)],
   name = '20 MV',
      error_y=dict(
         type='data', # value of error bar given in data coordinates
         array = c4_Impedance_unstacked.stack(),
         visible=True),
   line = dict(color='rgb(0, 90, 0)', width=4)))
###... Continue pattern from 30 MV to 280 MV ...
fig_well4_impedance.add_trace(go.Scatter(
      x= np.arange(10,1000,10), # MANUALLY CREATED A SERIES OF RANGE 1, 1869. OR # USE
df['col'].value_count
      y= df_b4_Impedance.loc[290, np.arange(10,1000,10)],
   name = '290 MV',
      error_y=dict(
         type='data', # value of error bar given in data coordinates
         array = c4_Impedance_unstacked.stack(),
         visible=True),
   line = dict(color='rgb(80, 15, 38)', width=4)))
```

Outcome of plot:



The difficulty of using plotly, was that I had to add one line at a time. The best solution is to use a “for loop”, to iterate in using another line per voltage.

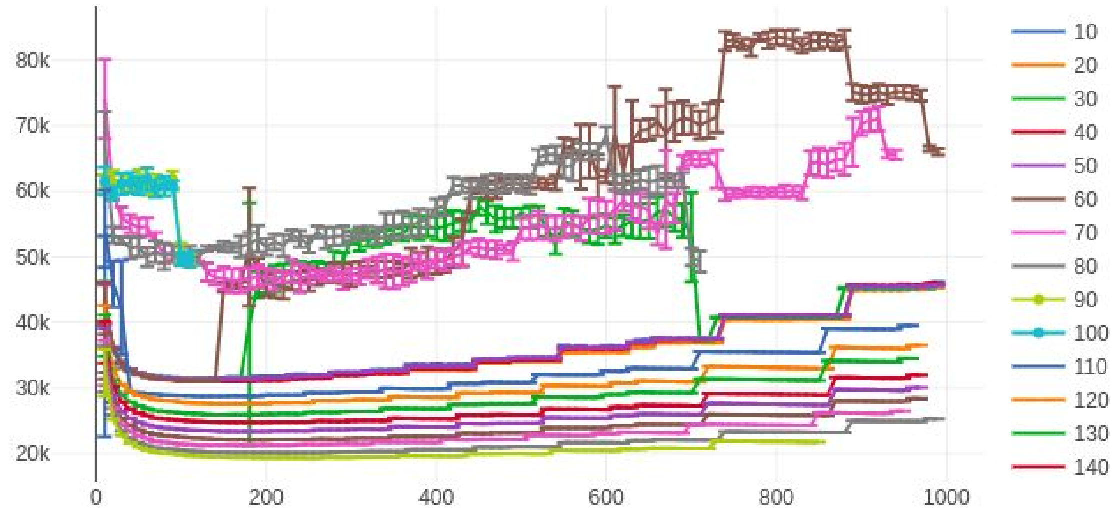
I learned the importance of for loops, and used it often, especially when I detected patterns that I wanted to iterate through. I believed the first step of iterating would be to start producing a few values at a time, and after observing the patterns being repeated, iterate along those patterns. The key data structure to use would be “lists,” for iteration.

Here is my new code, by which I created a list to plot Impedance vs. Frequency for Well 1:

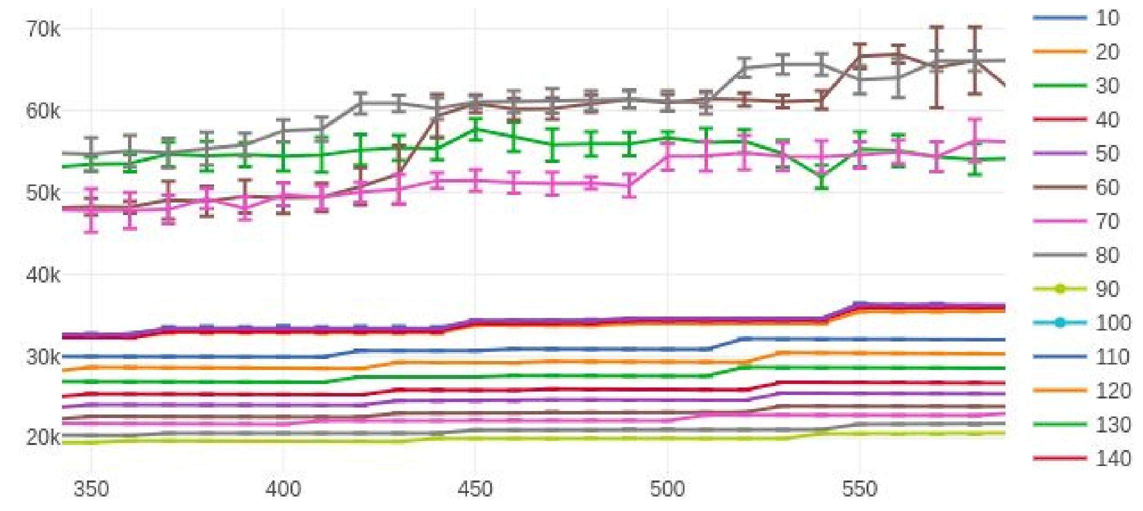
```
### Impedance vs. Frequency for Well 1
container = []
for i in a.index:
   traces = (go.Scatter(
      x= np.arange(10,1000,10), # MANUALLY CREATED A SERIES OF RANGE 1, 1869. OR # USE
df['col'].value_count
      y= df_b_Impedance.loc[i, np.arange(10,1000,10)],
      name = i,
      error_y=dict(
         type='data', # value of error bar given in data coordinates
         array = c_Impedance_unstacked.stack()[i],
         visible=True)))
   container.append(traces)

fig = dict(data=container)
pyo.plot(fig, validate=False) # Offline Plotting
```

Note: If I need to create a plot for Well 2, all I need to change is the "y-value" from df_b_Impedance.loc[...] to df_b2_Impedance.loc[...]. In addition, If I need to switch to another variable, I would do the same type of preprocessing work for the data frames of Phase, Conductance, Susceptance, Resistance, and Reactance.

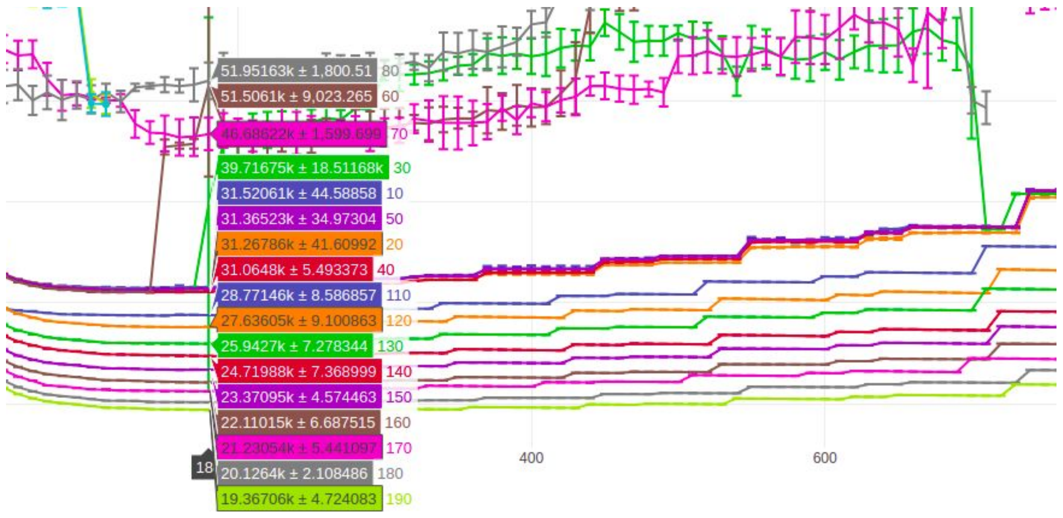
Output Result:



Zoom-in:



Hover-Over View:



What I like about plotly, is that, we can hover over a certain x-value (Frequency value), and simultaneously compare values within different voltage values. In the above plot, I included error-bars to visualize the range of standard deviation along each frequency value depicted (per 10 MV).

4. Python (Jupyter Notebook)/MySQL:

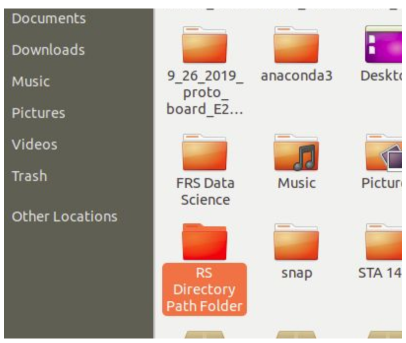
Using another Udemy course, "The Business Intelligence Analyst Course 2019," I learned how to apply what I learned about MySQL and applying commands inside Jupyter Notebook. I began by reviewing the main SQL commands, and how to use them. Then, I created a database to store all of the information within 80 ".csv" files. Here, each ".csv" file is a table on its own. Then, we created a big README table, by which we first created one README table through a single dataframe uploaded from a single "README.txt" file. Afterwards, I iterated through 8 different "README.txt" files to create 8 README tables inside MySQL. Finally, I used "SELECT" and "INSERT INTO" commands to insert all those values inside a big “SQL table.” With SQL table, I linked it to another SQL table called, “table_of_experiment” via primary key/foreign key relationship under the “Experiment_Name” Column for each table.

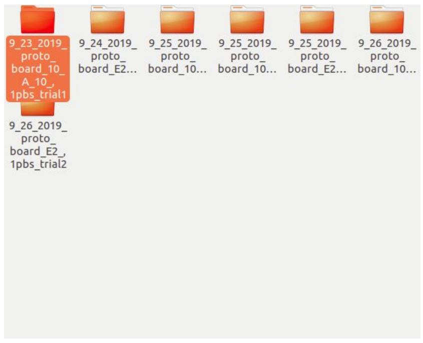
Code for Connecting to MySQL through Jupyter Notebook:

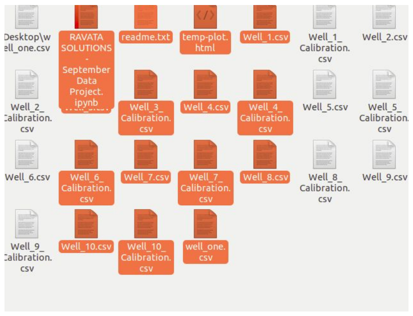
```
import mysql.connector as mc
from sqlalchemy import types, create_engine
connection = mc.connect(
   host="localhost",
   user="root",
   passwd="",
    database="ravata_solutions_db")
mycursor = connection.cursor()
mycursor.execute ("SELECT VERSION()")
row = mycursor.fetchone()

# MySQL Connection
MYSQL_USER = 'root'
MYSQL_PASSWORD = ''
MYSQL_HOST_IP = '127.0.0.1'
MYSQL_PORT = '3306'
MYSQL_DATABASE = 'ravata_solutions_db'
engine =
create_engine('mysql+mysqlconnector://'+MYSQL_USER+':'+MYSQL_PASSWORD+'@'+MYSQL_HOST_IP
+':'+MYSQL_PORT+'/'+MYSQL_DATABASE, echo=False)
```

Directory Folders:

We placed all of our data inside one directory path folder called, “RS Directory Path Folder,” which contains 8 separate folders. Each of the 8 folders contains a README.txt file and “.csv” file information for each well number.




9_23_2019_proto_board_10_A_10_1pbs_trial1
9_24_2019_proto_board_E2...
9_25_2019_proto_board_10...
9_25_2019_proto_board_10...
9_25_2019_proto_board_E2...
9_26_2019_proto_board_10...
9_26_2019_proto_board_E2_1pbs_trial2



Code for iterating through each folder:

```
# SET WORKING DIRECTORY (FILE OPEN PATH TO FOLDER WITH DATA):
# LIST OF FILE NAMES:
path = '/home/james/RS Directory Path Folder/'
folder = os.fsencode(path)
filenames = []
for file in os.listdir(folder):
    filename = os.fsdecode(file)
    #print(filename)
    filename_list_object = str('/home/james/RS Directory Path Folder/') + filename
    filenames.append(filename_list_object)

filenames = [filenames[-1], filenames[-4], filenames[-7], filenames[-6], filenames[-3], filenames[-8],
filenames[-2], filenames[-5]]
filenames
```

OUTPUT (List of Working Directories):

```
['/home/james/RS Directory Path Folder/9_23_2019_proto_board_10_A_10_,1pbs_trial1',
 '/home/james/RS Directory Path Folder/9_24_2019_proto_board_E2_,1pbs_trial1',
 '/home/james/RS Directory Path Folder/9_25_2019_proto_board_10_A_10_,1pbs_trial1',
 '/home/james/RS Directory Path Folder/9_25_2019_proto_board_10_A_10_,1pbs_trial2',
 '/home/james/RS Directory Path Folder/9_25_2019_proto_board_E2_,1pbs_trial1',
 '/home/james/RS Directory Path Folder/9_26_2019_proto_board_10_A_10_,1pbs_trial1',
 '/home/james/RS Directory Path Folder/9_26_2019_proto_board_E2_,1pbs_trial1',
 '/home/james/RS Directory Path Folder/9_26_2019_proto_board_E2_,1pbs_trial2']
```

Process of iterating through multiple folders via function/for loop:

```
### Preprocessing DF Function:
def preprocess_fromdf(filename):
    df = pd.read_csv(filename)
    df = df.drop(['Bias Voltage','Conductance', 'Susceptance', 'Resistance','Reactance', 'Capacitance
Parallel','Quality Factor','Capacitance Series','Dissipation Factor','Inductance Series','ESR'], axis=1)
    df['Frequency'] = df['Frequency'].map(lambda x: x.rstrip('KHZ'))
    df['Voltage'] = df['Voltage'].map(lambda x: x.rstrip(' MV'))
    df['Frequency'] = pd.to_numeric(df['Frequency'])
    df['Voltage'] = pd.to_numeric(df['Voltage'])
    return df

### LIST OF CSV FILES:
csvFiles_List = []
for i in range(10):
    csvFiles_List += ['Well_'+str(i+1)+'.csv']
csvFiles_List
```

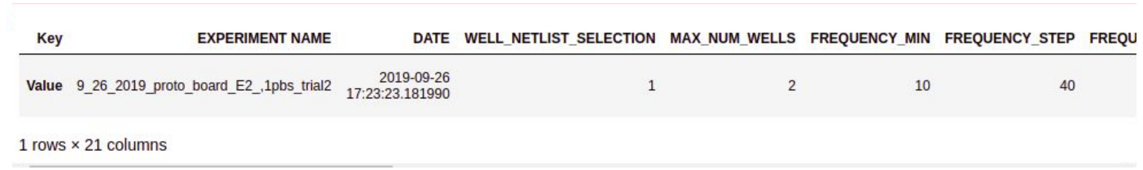
```
### Iterate through all files inside 8 folders to create SQL table:
for file in os.listdir(folder):
    filename = os.fsdecode(file)
    counter = 0
    for file in csvFiles_List:
        df = preprocess_fromdf(file)
        counter += 1
        df.to_sql('Exp_' + str(filename) + 'Well_' + str(counter)+'_Table', con=engine)
```

SQL README Table Preprocessing Code:

Given that it wouldn't be feasible to store design tables to contain information in the format of dictionaries, the next approach was to have each Well ".csv" file be its own table.

```
readme_df_list = []
for file in filenames:
    os.chdir(file) # path -> working directory
    os.scandir(path=file)
    readme_df_list += [pd.read_csv('readme.txt', delimiter=": ", header=None)] # sep, or delimiter?

count=0
for rm in readme_df_list: # Each rm - is a dataframe.
    rm.columns = ['Key', 'Value']
    rm = rm.set_index('Key')
    rm = rm.T
    rm = rm.reset_index(drop=True)
    count += 1
    rm.to_sql("README_SAMPLE_C" + str(count), con=engine)

sqlInsrtSelect_List = []
for i in range(7):
    sqlInsrtSelect_List += [
        """
        INSERT INTO  ravata_solutions_db.README_SAMPLE_C1
        SELECT *
        FROM ravata_solutions_db.README_SAMPLE_C""" + str(i+2) + """
        """]
for query in sqlInsrtSelect_List:
    mycursor.execute(query)
    connection.commit()
```

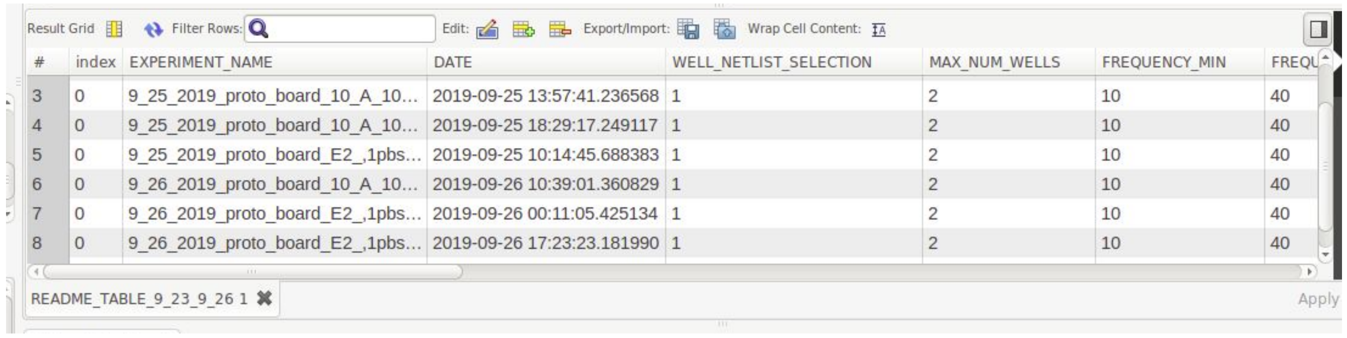
Format of single README Dataframe:



Key	EXPERIMENT NAME	DATE	WELL_NETLIST_SELECTION	MAX_NUM_WELLS	FREQUENCY_MIN	FREQUENCY_STEP	FREQU
Value	9_26_2019_proto_board_E2_,1pbs_trial2	2019-09-26 17:23:23.181990	1	2	10	40	

1 rows × 21 columns

Output of "Big" README Table in MySQL (after iterating INSERT INTO/SELECT commands):

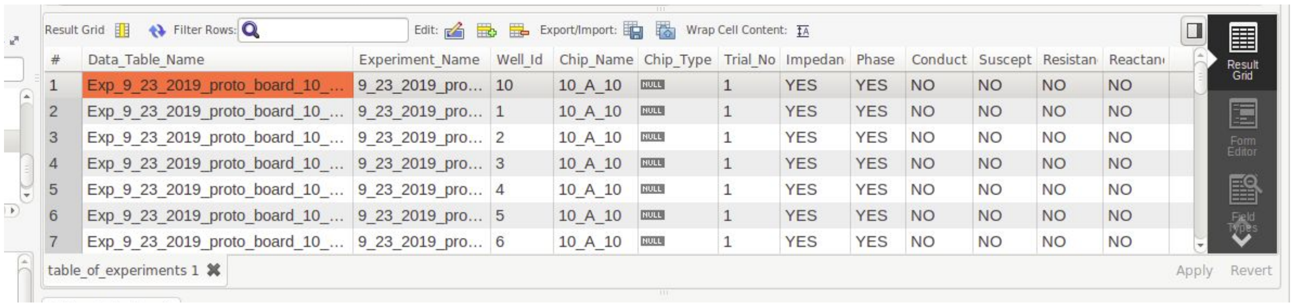


#	index	EXPERIMENT_NAME	DATE	WELL_NETLIST_SELECTION	MAX_NUM_WELLS	FREQUENCY_MIN	FREQU
3	0	9_25_2019_proto_board_10_A_10...	2019-09-25 13:57:41.236568	1	2	10	40
4	0	9_25_2019_proto_board_10_A_10...	2019-09-25 18:29:17.249117	1	2	10	40
5	0	9_25_2019_proto_board_E2_,1pbs...	2019-09-25 10:14:45.688383	1	2	10	40
6	0	9_26_2019_proto_board_10_A_10...	2019-09-26 10:39:01.360829	1	2	10	40
7	0	9_26_2019_proto_board_E2_,1pbs...	2019-09-26 00:11:05.425134	1	2	10	40
8	0	9_26_2019_proto_board_E2_,1pbs...	2019-09-26 17:23:23.181990	1	2	10	40

README_TABLE_9_23_9_26 1

Table of Experiments:

The challenge with creating this table was knowing how to set-up this table. It was difficult in understanding how the table was to be designed. It was crucial to manually write out how we wanted to design this table along with other tables. Another challenge with creating this table was distinguishing when to use INSERT INTO, or UPDATE/SET/WHERE Commands. The INSERT INTO Command would be used when we wanted to insert values for multiple rows and multiple columns. I believe, I used UPDATE/SET/WHERE when I needed to just update rows with same values on a given condition, i.e., Chip_Name = 10_A_10.



#	Data_Table_Name	Experiment_Name	Well_Id	Chip_Name	Chip_Type	Trial_No	Impedan	Phase	Conduct	Suscept	Resistan	Reactan
1	Exp_9_23_2019_proto_board_10_...	9_23_2019_pro...	10	10_A_10	NULL	1	YES	YES	NO	NO	NO	NO
2	Exp_9_23_2019_proto_board_10_...	9_23_2019_pro...	1	10_A_10	NULL	1	YES	YES	NO	NO	NO	NO
3	Exp_9_23_2019_proto_board_10_...	9_23_2019_pro...	2	10_A_10	NULL	1	YES	YES	NO	NO	NO	NO
4	Exp_9_23_2019_proto_board_10_...	9_23_2019_pro...	3	10_A_10	NULL	1	YES	YES	NO	NO	NO	NO
5	Exp_9_23_2019_proto_board_10_...	9_23_2019_pro...	4	10_A_10	NULL	1	YES	YES	NO	NO	NO	NO
6	Exp_9_23_2019_proto_board_10_...	9_23_2019_pro...	5	10_A_10	NULL	1	YES	YES	NO	NO	NO	NO
7	Exp_9_23_2019_proto_board_10_...	9_23_2019_pro...	6	10_A_10	NULL	1	YES	YES	NO	NO	NO	NO

table_of_experiments 1

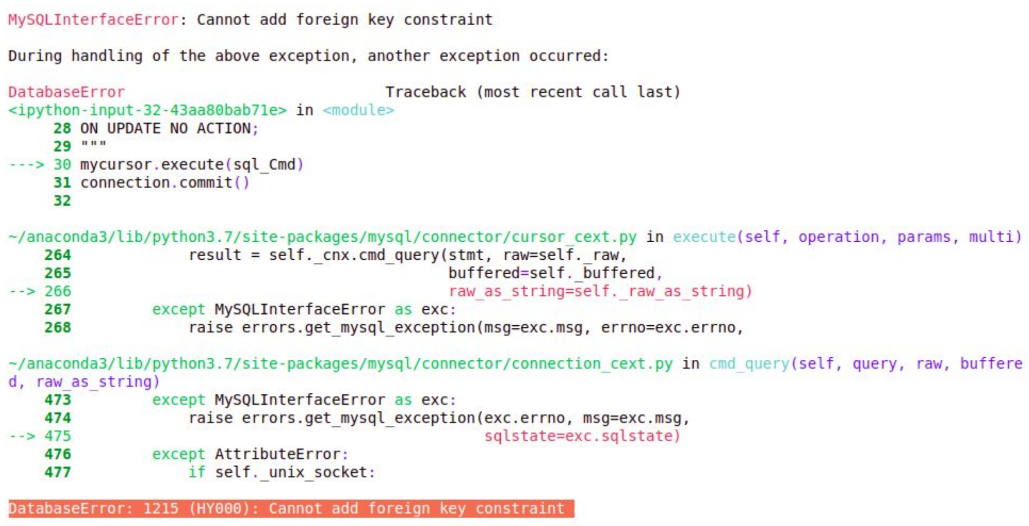
Primary Key/Foreign Key Relationship:

Here, we see that the table_of_experiments table has a foreign key on "Experiment_Name," which references the "EXPERIMENT_NAME" column from README_TABLE_9_23_9_26.

Debugging/Coding for PK/FK Relationship:

Before producing the primary key/foreign key relationship, I encountered an error to debug.

Errors are in the following format:



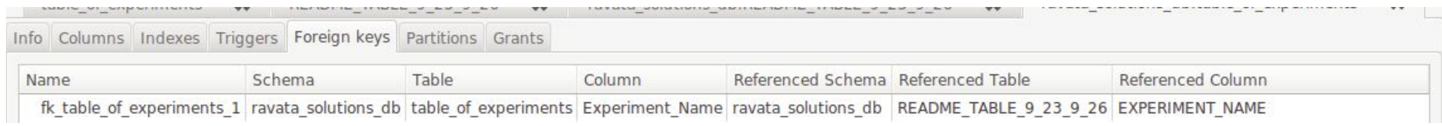
```
MySQLInterfaceError: Cannot add foreign key constraint

During handling of the above exception, another exception occurred:

DatabaseError                             Traceback (most recent call last)
<ipython-input-32-43aa80bab71e> in <module>
     28 ON UPDATE NO ACTION;
     29 """
---> 30 mycursor.execute(sql_Cmd)
     31 connection.commit()
     32 

~/anaconda3/lib/python3.7/site-packages/mysql/connector/cursor_cext.py in execute(self, operation, params, multi)
    264             result = self._cnx.cmd_query(stmt, raw=self._raw,
    265                                          buffered=self._buffered,
--> 266                                          raw_as_string=self._raw_as_string)
    267         except MySQLInterfaceError as exc:
    268             raise errors.get_mysql_exception(msg=exc.msg, errno=exc.errno,

~/anaconda3/lib/python3.7/site-packages/mysql/connector/connection_cext.py in cmd_query(self, query, raw, buffere
d, raw_as_string)
    473         except MySQLInterfaceError as exc:
    474             raise errors.get_mysql_exception(exc.errno, msg=exc.msg,
--> 475                                              sqlstate=exc.sqlstate)
    476         except AttributeError:
    477             if self._unix_socket:

DatabaseError: 1215 (HY000): Cannot add foreign key constraint
```

The solution was to add code beforehand:
SET FOREIGN_KEY_CHECKS = 0;

Then, I can either, manually or enter code to create primary key/foreign key relationships.

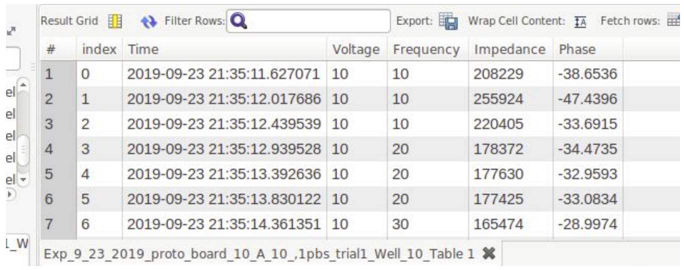
CODE:

```
ALTER TABLE `ravata_solutions_db`.`table_of_experiments`
ADD CONSTRAINT `fk_table_of_experiments_1`
  FOREIGN KEY (`Experiment_Name`)
  REFERENCES `ravata_solutions_db`.`README_TABLE_9_23_9_26` (`EXPERIMENT_NAME`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION;
```



Name	Schema	Table	Column	Referenced Schema	Referenced Table	Referenced Column
fk_table_of_experiments_1	ravata_solutions_db	table_of_experiments	Experiment_Name	ravata_solutions_db	README_TABLE_9_23_9_26	EXPERIMENT_NAME

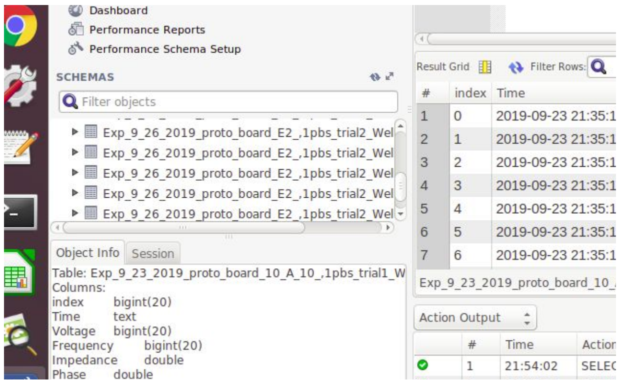
Sample Experiment Table (originally from .csv file):



#	index	Time	Voltage	Frequency	Impedance	Phase
1	0	2019-09-23 21:35:11.627071	10	10	208229	-38.6536
2	1	2019-09-23 21:35:12.017686	10	10	255924	-47.4396
3	2	2019-09-23 21:35:12.439539	10	10	220405	-33.6915
4	3	2019-09-23 21:35:12.939528	10	20	178372	-34.4735
5	4	2019-09-23 21:35:13.392636	10	20	177630	-32.9593
6	5	2019-09-23 21:35:13.830122	10	20	177425	-33.0834
7	6	2019-09-23 21:35:14.361351	10	30	165474	-28.9974

Exp_9_23_2019_proto_board_10_A_10_.1pbs_trial1_Well_10_Table 1

Where to find all of our tables inside MySQL Workbench:



Exp_9_26_2019_proto_board_E2_,1pbs_trial2_Wel
Exp_9_26_2019_proto_board_E2_,1pbs_trial2_Wel
Exp_9_26_2019_proto_board_E2_,1pbs_trial2_Wel
Exp_9_26_2019_proto_board_E2_,1pbs_trial2_Wel
Exp_9_26_2019_proto_board_E2_,1pbs_trial2_Wel

```
Table: Exp_9_23_2019_proto_board_10_A_10_,1pbs_trial1_W
Columns:
index       bigint(20)
Time        text
Voltage     bigint(20)
Frequency       bigint(20)
Impedance       double
Phase       double
```

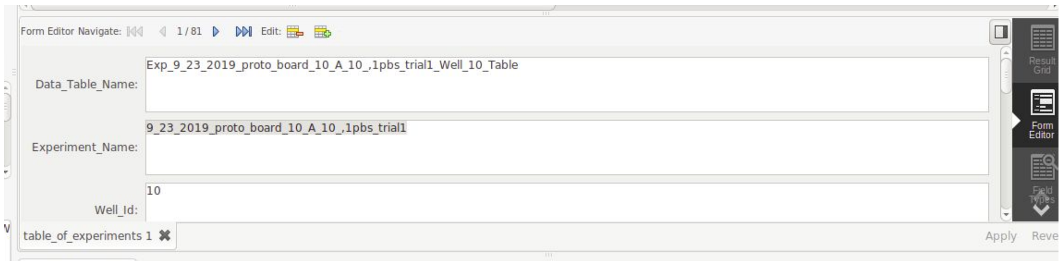
#	index	Time
1	0	2019-09-23 21:35:1
2	1	2019-09-23 21:35:1
3	2	2019-09-23 21:35:1
4	3	2019-09-23 21:35:1
5	4	2019-09-23 21:35:1
6	5	2019-09-23 21:35:1
7	6	2019-09-23 21:35:1

Exp_9_23_2019_proto_board_10_

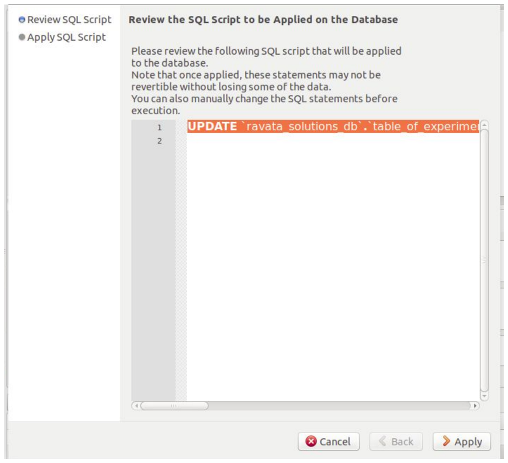
	#	Time	Action
✓	1	21:54:02	SELEC

Process of Iterating Commands through MySQL/Jupyter Notebook:

Let's say, we would want to change the experiment name for not just one, but multiple values.
A good idea is to manually insert a value inside MySQL Workbench and review SQL script. Following this, we are capable of observing which pattern we would iterate over. And then, we can create a list of commands to insert multiple values into multiple rows (i.e., using UPDATE/SET).



Data_Table_Name:	Exp_9_23_2019_proto_board_10_A_10_,1pbs_trial1_Well_10_Table
Experiment_Name:	9_23_2019_proto_board_10_A_10_,1pbs_trial1
Well_Id:	10



Review the SQL Script to be Applied on the Database

Please review the following SQL script that will be applied to the database.
Note that once applied, these statements may not be revertible without losing some of the data.
You can also manually change the SQL statements before execution.

```
UPDATE `ravata_solutions_db`.`table_of_experime
```

In the above SQL code for review, the code may look as follows:

```
UPDATE `ravata_solutions_db`.`table_of_experiments`
    SET `Data_Table_Name`='Exp_9_23_2019_proto_board_10_A_10_,1pbs_trial1_Well_10_Table'
    WHERE `Data_Table_Name`='Exp_9_23_10_pb_10_A_10_trial1_Well_10_Table';
```

Here, we replace “10” with “”””str(i+1)”””” if we want to iterate.

Then, use the following sample code:

```
sqlUpdate_Cmd_List = []
for i in range(10):
   sqlUpdate_Cmd_List += [
      """
      UPDATE `ravata_solutions_db`.`table_of_experiments`
      SET
`Data_Table_Name`='Exp_9_23_2019_proto_board_10_A_10_,1pbs_trial1_Well_"""+str(i+1)+"""_Table'
      WHERE `Data_Table_Name`='Exp_9_23_10_pb_10_A_10_trial1_Well_"""+str(i+1)+"""_Table';
      """ ]
for query in sqlUpdate_Cmd_List:
   mycursor.execute(query)
   connection.commit()
```

Process of Updating via MySQL:

Here, what I attempted first was manually performing steps. Given the SQL code, I would copy and paste inside Jupyter Notebook in order to iterate the code multiple times if needed.

Within working inside the database, I had to use a list of MySQL Commands multiple times, so that I can iterate between multiple rows within a table, and speed up the process of database design. Iterating through a list of UPDATE/SET Commands is how I created the “table_of_experiments” table originally.

Conclusion

This internship, so far, has taught me a variety of skills. From time management to attempting to meet deadlines, I faced numerous challenges. Overall, I enjoyed the experience, given it is my first internship and being allowed to face real-world challenges with a start-up company in the BioTech industry. I have learned to become more confident as a programmer in the Python language for Data Science, in addition to MySQL.

A majority of the problems that I faced, had to do with debugging errors that I frequently came in contact with. Stepping into the internship, I would not have known how to quickly debug errors. But, I was taught well on how to debug errors, where if I were given any error message, I can perform a quick search on Google that references an answer on Stack Overflow, and model other people’s code online through pseudo code. I learned, in addition, how to be patient in working with cleaning big data for data visualization.

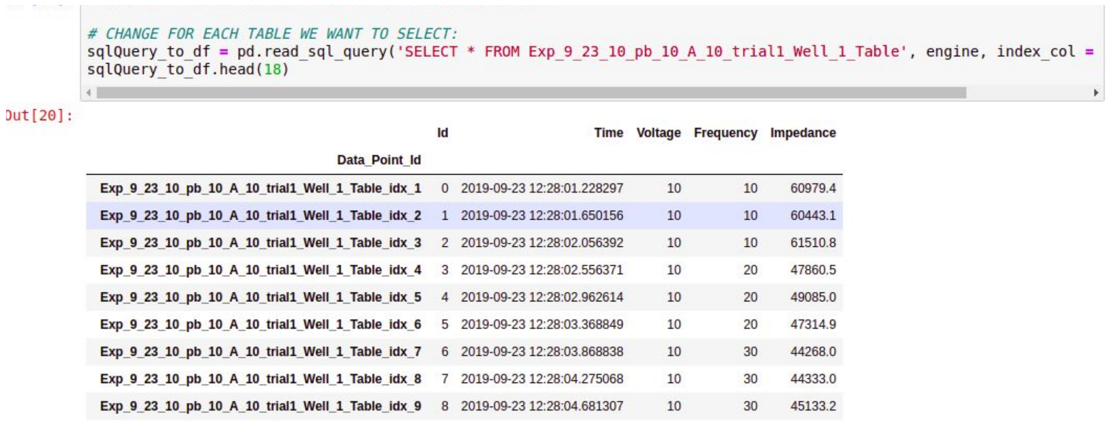
Another difficulty I faced, was trying to understand what is the desired output, say, in terms of database design. There were ideas of what we wanted to be done, in terms of creating the database and storing information, yet the solution at hand was slightly different than what our proposed plan of database design. An example was beginning with trying to use SQL tables to store dictionaries from Python.

Following this, laptop crashes can happen frequently, when working with big data. Thus, one has to be both patient, and learn how to condense code, when possible. Always important, is to have a back-up plan when computer problems arise. It is inevitable to not run into computer problems, crashes, but it is always good to know that there is always a solution out there. As an intern who is on the verge of applying for full-time positions in the future, having learned to overcome the hurdles, I believe that this internship has prepared me for what is to come.

Next Steps

The steps following creation of MySQL database, would be to attempt to use the “SELECT” statement to pull information from different Experiment tables, given a condition. I would do this by first examining the connections between the “README” table and the “table_of_experiments” table. Afterwards, I can use the “pd.read_sql()” or “pd.read_sql_query” function inside pandas to read a given SQL table and create a dataframe. I would most likely have to create multiple data frames, and “merge”, “join”, or “groupby”, in the preprocessing phase, in order to create a big dataframe for plotting. I can just recall how I made a function in the past, so that I can more speedily preprocess data ready for plotting. Moreover, I am ready to work with more data and iterating to add to the big “README” table and then I will add to the preexisting “table_of_experiments” table.

Maybe we want to try something like this:



```
# CHANGE FOR EACH TABLE WE WANT TO SELECT:
sqlQuery_to_df = pd.read_sql_query('SELECT * FROM Exp_9_23_10_pb_10_A_10_trial1_Well_1_Table', engine, index_col =
sqlQuery_to_df.head(18)
```

Out[20]:

	Id	Time	Voltage	Frequency	Impedance
Data_Point_Id					
Exp_9_23_10_pb_10_A_10_trial1_Well_1_Table_idx_1	0	2019-09-23 12:28:01.228297	10	10	60979.4
Exp_9_23_10_pb_10_A_10_trial1_Well_1_Table_idx_2	1	2019-09-23 12:28:01.650156	10	10	60443.1
Exp_9_23_10_pb_10_A_10_trial1_Well_1_Table_idx_3	2	2019-09-23 12:28:02.056392	10	10	61510.8
Exp_9_23_10_pb_10_A_10_trial1_Well_1_Table_idx_4	3	2019-09-23 12:28:02.556371	10	20	47860.5
Exp_9_23_10_pb_10_A_10_trial1_Well_1_Table_idx_5	4	2019-09-23 12:28:02.962614	10	20	49085.0
Exp_9_23_10_pb_10_A_10_trial1_Well_1_Table_idx_6	5	2019-09-23 12:28:03.368849	10	20	47314.9
Exp_9_23_10_pb_10_A_10_trial1_Well_1_Table_idx_7	6	2019-09-23 12:28:03.868838	10	30	44268.0
Exp_9_23_10_pb_10_A_10_trial1_Well_1_Table_idx_8	7	2019-09-23 12:28:04.275068	10	30	44333.0
Exp_9_23_10_pb_10_A_10_trial1_Well_1_Table_idx_9	8	2019-09-23 12:28:04.681307	10	30	45133.2

In addition, even though I only have 4 weeks left of school at Davis, I do not mind staying a bit afterwards to just gain more experience with machine learning/deep learning/predictive modeling, in case I am unable to do so in the next four weeks. My proposal is to see, based off data collected, how can we predict future outcomes based off adjustment to chips, number of wells, etc.