MAIN_Pascal_Tutorial1_V1

June 22, 2023

1 Quick Tutorial on Python Interaction with the HGCAL INT2R database

1.1 Ali Al Kadhim - FSU

1.2 May 2023

[135]:
```
import cx_Oracle
import sys
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
import argparse
import os
import IPython
from IPython.display import Image, display
import time
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
# USE PYTHON3  !

os.environ['PASCAL']='/home/PASCAL_REPO'
```

[136]:
```
print(os.environ['PASCAL'])
PASCAL=os.environ['PASCAL']
OUTPUT_DIR=os.path.join(PASCAL,'outputs')
QUERY_DIR=os.path.join(PASCAL,'queries')
LOG_DIR=os.path.join(PASCAL,'logs')
IMAGE_DIR=os.path.join(PASCAL,'images')
```

```
/home/PASCAL_REPO
```

Make sure you've run `bash tunnel.sh username` before starting this or in the same terminal that is running this notebook

[137]:
```
############################## Some helper functions ###############
# def printException(exception):
#       error, = exception.args
```

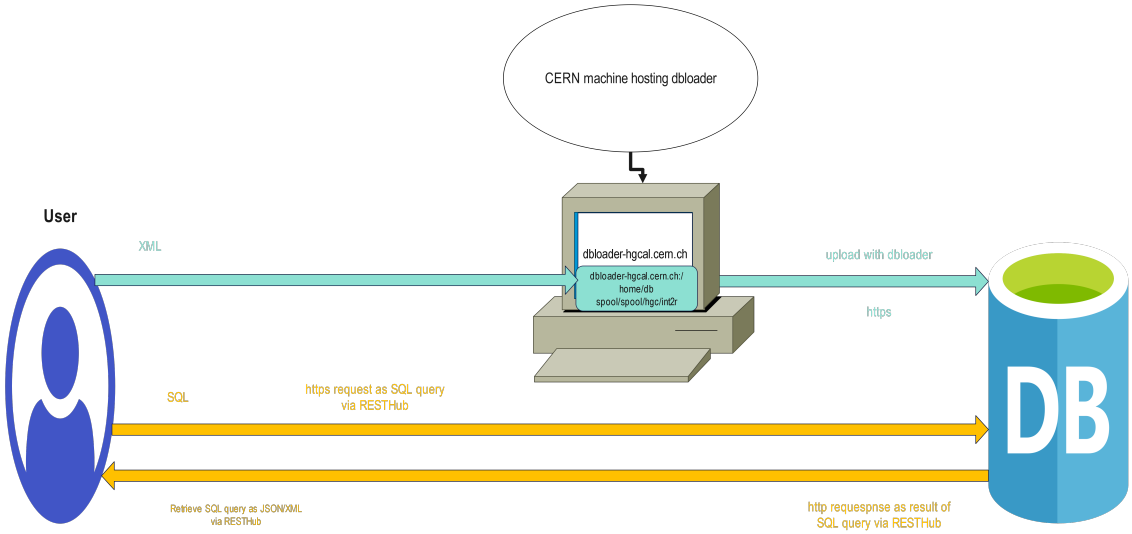
```
#   printf("Error code = %s\n",error.code)
#   printf("Error message = %s\n",error.message)

def printf(format,*args):
      sys.stdout.write(format % args)

def show_jupyter_image(image_filename, width=1300, height=300):
    """Show a saved image directly in jupyter. Make sure image_filename is in␣
 ↪your IQN_BASE !"""
    display(Image(os.path.join(os.environ['PASCAL'], 'images', image_filename),␣
 ↪width=width, height=height))

def makeDictFactory(cursor):
    columnNames = [d[0] for d in cursor.description]
    def createRow(*args):
        return dict(zip(columnNames, args))
    return createRow

def output_type_handler(cursor, name, default_type, size, precision, scale):
    if default_type == cx_Oracle.DB_TYPE_VARCHAR:
        return cursor.var(default_type, size, arraysize=cursor.arraysize,
                          encoding_errors="replace")

# cursor.outputtypehandler = output_type_handler


############################## Some decorators ##############################
def SourcePASCAL(func):
    def _func(*args):
        import os
        from common.utility.source import source

        env = {}
        env.update(os.environ)
        env.update(source(os.environ["PASCAL"]))
        func(*args, env=env)

    return _func

def debug(func):
    """Print the function signature and return value"""
    import functools

    @functools.wraps(func)
    def wrapper_debug(*args, **kwargs):
        args_repr = [repr(a) for a in args]
        kwargs_repr = [f"{k}={v!r}" for k, v in kwargs.items()]
```

```
        signature = ", ".join(args_repr + kwargs_repr)
        print(f"Calling {func.__name__}({signature})")
        values = func(*args, **kwargs)
        print(f"{func.__name__!r} returned {values!r}")
        return values

    return wrapper_debug

def make_interactive(func):
    """make the plot interactive"""
    import functools

    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        plt.ion()
        output = func(*args, **kwargs)
        plt.ioff()
        return output

    return wrapper

def timer(func):
    """Print the runtime of the decorated function"""
    import functools
    import time
    @functools.wraps(func)
    def wrapper_timer(*args, **kwargs):
        start_time = time.perf_counter()    # 1
        value = func(*args, **kwargs)
        end_time = time.perf_counter()      # 2
        run_time = end_time - start_time    # 3
        print(f"\nFINISHED {func.__name__!r} in {run_time:.4f} SECS")
        return value
    return wrapper_timer

# from IPython.core.magic import register_cell_magic

# @register_cell_magic
def write_and_run(line, cell):
    """write the current cell to a file (or append it with -a argument) as well␣
 ↪as execute it
    use with %%write_and_run at the top of a given cell"""
    argz = line.split()
    file = argz[-1]
    mode = "w"
    if len(argz) == 2 and argz[0] == "-a":
        mode = "a"
```

```
    with open(file, mode) as f:
        f.write(cell)
    # get_ipython().run_cell(cell)
############################  ############################
```

1.2.1 Screenshot of dbolader diagram

[138]:
```
display(Image(os.path.join(IMAGE_DIR, 'RESTHUB_dbloader_diagram.png')))
```



1.2.2 Screenshot of XML template -> SQLDeveloper view

Previously, the recommended way to read data from HGCAL DB was using SQLDeveloper. But it is very tedious to install/configure and not fast or convenient enough.

1.3 Define Connection Configurations

The database is a Oracle Database, with access permitted at port 10131. For Int2R, The service name, username, and passwork are 'int2r_lb.cern.ch', 'HGCAL_Reader_2016' and 'CMS_HGC_PRTTYPE_HGCAL_READER', respectively.

[140]:
```
HOST='localhost'
PORT = '10131'
SERVICE_NAME='int2r_lb.cern.ch'
PASS=r'HGCAL_Reader_2016'
USER=r'CMS_HGC_PRTTYPE_HGCAL_READER'
```

make TNS connection

[141]:
```
DSN_TNS = cx_Oracle.makedsn(HOST, PORT, service_name=SERVICE_NAME)
```

Create the connection only once for faster eqxecution

[142]:
```
try:
    conn = cx_Oracle.connect(user=USER, password=PASS, dsn=DSN_TNS,
                        # encoding="UTF-8"
                        )
    print("\nGreat! You've successfully Connected to the Database\n")
except Exception:
    print("\nUnfortunately, you have not connected to the database. Do you have␣
 ↪access permission from Umesh or Haffeez?\n")
```

```
Great! You've successfully Connected to the Database
```

2 Quick Tutorial on SQL queries in HGCAL INT2R

SQL for data-retrieval in a nutshell:

`SELECT column_name FROM table_name WHERE condition;`

Look at the Documentation here for explanation of the schemas, tables and attributes. Since this is made from SQC-related tables perspective, we will be using the tables in the `CMS_HGC_CORE_COND` schema.

For the HGCAL DB data, focusing on Si snesors, we should focus on two kinds of tables:

- `*_CONSTRUCT`: DESCRIBES HOW ALL PARTS/CHILDREN ARE RELATED TO EACH OTHER.
- `*_COND`: CONDITIONS DATA, WHICH IS DATA DESCRIBING A MEASUREMENT OR A TEST ON A PART.

You should think of the `CMS_HGC_CORE_COND.COND_DATA_SETS` table as the master table that has a record of everything that was uploaded, and the `CONDITION_DATA_SET_ID` as the successful upload ID that relates schemas and tables.

2.0.1 Part 1: Sample Queries to Get Started

We will Ue cx_Oracle for the interface with the DB, all the dependencies and configurations have been configured in the Pascal docker image

We will execute any SQL query (that permitted by cx_Oracle), and return output as pandas DataFrame

2.0.2 Example 1: Execute SQL Query Directly

[143]:
```
@timer
def execute_query(QUERY, maxrows ='all', outformat='DF', saveformat=None,␣
 ↪outstring=None):
```

```
    """

    Args:
    outformat: the format of the result. DF = pandas.DataFrame(),
    """
    # conn = None
    try:
    # if you want a new connection and close it at the end, uncomment below
      conn = cx_Oracle.connect(user=USER, password=PASS, dsn=DSN_TNS,
                               # encoding="UTF-8"
                               )


      cursor = conn.cursor()

    except Exception as err:
      print('Connection error')
      print(err)
    finally:
      if conn:
            if maxrows=='all':
            # execute
                cursor.execute(QUERY)
            # rows = cursor.execute(QUERY2)
            # cursor.execute(QUERY_TIME_3)
            # conn.commit()
            else:
                cursor.execute(QUERY,offset=0, maxnumrows=maxrows)

            # try:
              # rows=cursor.fetall()
            # except Exception as err:
              # print(err)
            columnNames = [d[0] for d in cursor.description]
            print('\nCOLUMN NAMES:\n', columnNames)
            # for row in rows:
                  # print(list(row))
            #return result as a dictionary
            result = [dict(zip(columnNames, row)) for row in cursor.fetchall()]
            # result=None
            cursor.close()
            conn.close()

    if outformat=='DF':
        df =pd.DataFrame(result)
```

```
    if saveformat=='CSV':
        df.to_csv(os.path.join(OUTPUT_DIR, f'{outstring}.csv'))
    # print(df.head())
    return df
```

The SQL query

```
select * from CMS_HGC_CORE_COND.KINDS_OF_CONDITIONS;
```

translates to

"show all columns and rows from the KINDS_OF_CONDITIONS table in the CMS_HGC_CORE_COND account"

---

[144]:
```
execute_query(QUERY="select * from CMS_HGC_CORE_COND.KINDS_OF_CONDITIONS")
```

```
COLUMN NAMES:
 ['KIND_OF_CONDITION_ID', 'IS_RECORD_DELETED', 'NAME', 'EXTENSION_TABLE_NAME',
'RECORD_INSERTION_TIME', 'RECORD_INSERTION_USER', 'RECORD_LASTUPDATE_TIME',
'RECORD_LASTUPDATE_USER', 'COMMENT_DESCRIPTION', 'CATEGORY_NAME']

FINISHED 'execute_query' in 0.5709 SECS
```

[144]:

	KIND_OF_CONDITION_ID	IS_RECORD_DELETED	NAME	EXTENSION_TABLE_NAME
0	2640	F	HGC Sensor Manufacturer IV Test	HGC_SENSOR_IV
1	1100	F	Hamamatsu-S10938-4956 Sensor IV Test	TEST_SENSOR_IV
2	1120	F	Hamamatsu-S10938-4956 Sensor CV Test	TEST_SENSOR_CV
3	1140	F	Hamamatsu-S10938-4956 Sensor Test Conds	TEST_SENSOR_CONDITIONS
4	18640	F	HGC Sensor Flatness Data	FLATNS_SENSOR_DATA
..	…	…	…	…
82	17080	F	SiPM HGCROC RAM Retention Time	HGCROC_RAM_RETENTION
83	17100	F		
84	17120	F		
85	17140	F		
86	4220	F		

```
83             HD HGCROC DACB Conveyor Test  HGCROC_DACB_CONVEYOR_TEST
84             LD HGCROC DACB Conveyor Test  HGCROC_DACB_CONVEYOR_TEST
85           SiPM HGCROC DACB Conveyor Test  HGCROC_DACB_CONVEYOR_TEST
86       HGC Six Inch Proto Module Assembly        HGC_PRTO_MOD_ASMBLY

   RECORD_INSERTION_TIME RECORD_INSERTION_USER RECORD_LASTUPDATE_TIME  \
0    2018-02-17 07:07:52                Umesh    2020-03-07 10:00:28
1    2017-09-14 06:44:30                Umesh    2020-03-07 10:11:03
2    2017-09-14 06:44:34                Umesh    2020-03-07 10:11:03
3    2017-09-14 06:44:37                Umesh    2020-03-07 10:11:03
4    2022-10-10 12:18:42                Umesh                    NaT
..                   …                    …                    …
82   2022-06-25 04:39:05                Umesh                    NaT
83   2022-06-25 05:11:06                Umesh    2022-06-25 05:16:37
84   2022-06-25 05:11:06                Umesh    2022-06-29 00:48:51
85   2022-06-25 05:11:07                Umesh    2022-06-29 00:48:55
86   2018-12-01 19:38:58                Umesh    2020-03-07 11:34:58

   RECORD_LASTUPDATE_USER                          COMMENT_DESCRIPTION  \
0       CMS_HGC_CORE_COND                           HGC Sensor IV Test
1       CMS_HGC_CORE_COND          Hamamatsu-S10938-4956 Sensor IV Test
2       CMS_HGC_CORE_COND          Hamamatsu-S10938-4956 Sensor CV Test
3       CMS_HGC_CORE_COND  Hamamatsu-S10938-4956 Sensor Test Conditions
4                    None                       HGC Sensor Flatness Data
..                    …                                             …
82                   None                SiPM HGCROC RAM Retention Time
83      CMS_HGC_CORE_COND                  HD HGCROC_DACB_Conveyor_TEST
84      CMS_HGC_CORE_COND                  LD HGCROC_DACB_Conveyor_TEST
85      CMS_HGC_CORE_COND                SiPM HGCROC_DACB_Conveyor_TEST
86      CMS_HGC_CORE_COND           HGC Six Inch Proto Module Assembly

    CATEGORY_NAME
0     MEASUREMENT
1     MEASUREMENT
2     MEASUREMENT
3     MEASUREMENT
4            None
..            …
82           None
83           None
84           None
85           None
86  CONFIGURATION

[87 rows x 10 columns]
```

[10]:
```
df=execute_query(QUERY="select * from CMS_HGC_CORE_COND.KINDS_OF_CONDITIONS")
df.head()
```

```
COLUMN NAMES:
 ['KIND_OF_CONDITION_ID', 'IS_RECORD_DELETED', 'NAME', 'EXTENSION_TABLE_NAME',
'RECORD_INSERTION_TIME', 'RECORD_INSERTION_USER', 'RECORD_LASTUPDATE_TIME',
'RECORD_LASTUPDATE_USER', 'COMMENT_DESCRIPTION', 'CATEGORY_NAME']

FINISHED 'execute_query' in 0.6342 SECS
```

[10]:

	KIND_OF_CONDITION_ID	IS_RECORD_DELETED	NAME	EXTENSION_TABLE_NAME	RECORD_INSERTION_TIME	RECORD_INSERTION_USER	RECORD_LASTUPDATE_TIME	RECORD_LASTUPDATE_USER	COMMENT_DESCRIPTION	CATEGORY_NAME
0	2640	F	HGC Sensor Manufacturer IV Test	HGC_SENSOR_IV	2018-02-17 07:07:52	Umesh	2020-03-07 10:00:28	CMS_HGC_CORE_COND	HGC Sensor IV Test	MEASUREMENT
1	1100	F	Hamamatsu-S10938-4956 Sensor IV Test	TEST_SENSOR_IV	2017-09-14 06:44:30	Umesh	2020-03-07 10:11:03	CMS_HGC_CORE_COND	Hamamatsu-S10938-4956 Sensor IV Test	MEASUREMENT
2	1120	F	Hamamatsu-S10938-4956 Sensor CV Test	TEST_SENSOR_CV	2017-09-14 06:44:34	Umesh	2020-03-07 10:11:03	CMS_HGC_CORE_COND	Hamamatsu-S10938-4956 Sensor CV Test	MEASUREMENT
3	1140	F	Hamamatsu-S10938-4956 Sensor Test Conds	TEST_SENSOR_CONDITIONS	2017-09-14 06:44:37	Umesh	2020-03-07 10:11:03	CMS_HGC_CORE_COND	Hamamatsu-S10938-4956 Sensor Test Conditions	MEASUREMENT
4	18640	F	HGC Sensor Flatness Data	FLATNS_SENSOR_DATA	2022-10-10 12:18:42	Umesh	NaT	None	HGC Sensor Flatness Data	None

Save desired output of query into a convenient format (`.csv` in this example)

[14]:
```
! ls /home/PASCAL_REPO
```

```
build_and_push.sh  logs           pascal_run_command.sh    test_tunnel.sh
Dockerfile         miscillaneous  queries                  tunnel.sh
images             notebooks      README.md                utils
interact.sh        outputs        start_jupyter_server.sh
```

[15]:
```
# confirm that there is nothing currently in the outputs directory
! ls /home/PASCAL_REPO/outputs
```

```
KIND_OF_CONDITIONS_QUERY.csv
```

2.1 Example 1.1: save output as .csv file

[28]:
```
df=execute_query(QUERY="select * from CMS_HGC_CORE_COND.KINDS_OF_CONDITIONS",
                 saveformat='CSV',
                 outstring='KIND_OF_CONDITIONS_QUERY')
df.head()
```

```
COLUMN NAMES:
 ['KIND_OF_CONDITION_ID', 'IS_RECORD_DELETED', 'NAME', 'EXTENSION_TABLE_NAME',
'RECORD_INSERTION_TIME', 'RECORD_INSERTION_USER', 'RECORD_LASTUPDATE_TIME',
'RECORD_LASTUPDATE_USER', 'COMMENT_DESCRIPTION', 'CATEGORY_NAME']

FINISHED 'execute_query' in 0.5635 SECS
```

[28]:

	KIND_OF_CONDITION_ID	IS_RECORD_DELETED	NAME	EXTENSION_TABLE_NAME	RECORD_INSERTION_TIME	RECORD_INSERTION_USER	RECORD_LASTUPDATE_TIME	RECORD_LASTUPDATE_USER	COMMENT_DESCRIPTION	CATEGORY_NAME
0	2640	F	HGC Sensor Manufacturer IV Test	HGC_SENSOR_IV	2018-02-17 07:07:52	Umesh	2020-03-07 10:00:28			
1	1100	F	Hamamatsu-S10938-4956 Sensor IV Test	TEST_SENSOR_IV	2017-09-14 06:44:30	Umesh	2020-03-07 10:11:03			
2	1120	F	Hamamatsu-S10938-4956 Sensor CV Test	TEST_SENSOR_CV	2017-09-14 06:44:34	Umesh	2020-03-07 10:11:03			
3	1140	F	Hamamatsu-S10938-4956 Sensor Test Conds	TEST_SENSOR_CONDITIONS	2017-09-14 06:44:37	Umesh	2020-03-07 10:11:03			
4	18640	F	HGC Sensor Flatness Data	FLATNS_SENSOR_DATA	2022-10-10 12:18:42	Umesh	NaT			

```
  RECORD_LASTUPDATE_USER                              COMMENT_DESCRIPTION  \
0      CMS_HGC_CORE_COND                                HGC Sensor IV Test
1      CMS_HGC_CORE_COND          Hamamatsu-S10938-4956 Sensor IV Test
2      CMS_HGC_CORE_COND          Hamamatsu-S10938-4956 Sensor CV Test
3      CMS_HGC_CORE_COND  Hamamatsu-S10938-4956 Sensor Test Conditions
4                   None                        HGC Sensor Flatness Data

  CATEGORY_NAME
0   MEASUREMENT
1   MEASUREMENT
2   MEASUREMENT
3   MEASUREMENT
4          None
```

[29]:
```
! ls outputs
```

```
KIND_OF_CONDITIONS_QUERY.csv
```

2.2 Example 2: Quickly Query All Upload Attempts by a User

cx_oracle becomes very powerful as the SQL commands can be combined with python syntax.

See all attempts to upload and their success/failure status and their logs. Replace "Ali" with the name of the user who aploaded something, and you should see it, whether it was uploaded successfully and a log file associated with it!

```
select * from CMS_HGC_CORE_MANAGEMNT.CONDITIONS_DATA_AUDITLOG where RECORD_LASTUPDATE_USER LIKE
```

[57]:
```
def auditlog(username):
    AUDITLOG_ALI="select * from CMS_HGC_CORE_MANAGEMNT.CONDITIONS_DATA_AUDITLOG␣
 ↪where RECORD_LASTUPDATE_USER LIKE '{}%'".format(username)
    AUDITLOG_ALI_DF=execute_query(AUDITLOG_ALI,outformat='DF', saveformat=None,␣
 ↪outstring=None)
    # print(AUDITLOG_ALI_DF)
    return AUDITLOG_ALI_DF
```

`username=''` displays all usernames. Put your name to see info on what you uploaded, like `Ali`

[58]:
```
widgets.interact(auditlog, username='')
```

```
interactive(children=(Text(value='', description='username'), Output()),␣
 ↪_dom_classes=('widget-interact',))
```

[58]:
```
<function __main__.auditlog(username)>
```

2.3 Predifined queries from file in PASCAL

There are a bunch of predefined query templates that we think are useful for people.

2.4 Queries List (not all documeted yet)

Filename	description	jupyter usage example	terminal usage example
'CV_FULL.sql'	Plot all bias voltage vs capactivance (CV) for every cell at every voltage step	get_query_from_file('CV_FULL.sql')	
——-	—	—-	—
'IV_FULL.sql'	Plot all bias voltage vs current (IV) for every cell at every voltage step	get_query_from_file('IV_FULL.sql')	

[59]:
```
def get_query_from_file(query_file):
    query_file_path = os.path.join(QUERY_DIR,query_file)
    query_f = open(query_file_path)
    QUERY = query_f.read()
    # print(QUERY)
    query_f.close()

    return QUERY
```

Recall that the first step in uploading is registering the parts.

See the registered wafer that has serial number "100113":

```
select * from CMS_HGC_CORE_CONSTRUCT.PARTS where SERIAL_NUMBER='100113';
```

[66]:
```
def registered_parts(sensor_id):
    cmd = """sed -i "s/'SOME_SENSOR_SERIAL_NUMBER'/'%s'/g" %s/queries/
 ↪registered_parts.sql""" % (str(sensor_id), str(PASCAL) )
    os.system(cmd)
    registered_parts_query = get_query_from_file('registered_parts.sql')
    query_out = execute_query(registered_parts_query)
    cmd = """sed -i "s/'%s'/'SOME_SENSOR_SERIAL_NUMBER'/g" %s/queries/
 ↪registered_parts.sql""" % (str(sensor_id), str(PASCAL) )
    os.system(cmd)
    return query_out
```

[65]:
```
interact(registered_parts, sensor_id='100113')
```

```
interactive(children=(Text(value='100113', description='sensor_id'), Output()),␣
 ↪_dom_classes=('widget-interact…
```

[65]: <function __main__.registered_parts(sensor_id)>

3 Example 3: Plot CV test for all cells at all voltage steps

3.0.1 XML Template for Table: HGC_CERN_SENSOR_CV

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ROOT xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<HEADER>
    <TYPE>
        <EXTENSION_TABLE_NAME>HGC_CERN_SENSOR_CV</EXTENSION_TABLE_NAME>
        <NAME>HGC CERN Sensor CV Test</NAME>
    </TYPE>
    <RUN>
        <RUN_NAME>CERN HPK_8in_271_4003 CV Test</RUN_NAME>
        <RUN_BEGIN_TIMESTAMP>2018-05-14 00:00:00</RUN_BEGIN_TIMESTAMP>
        <RUN_END_TIMESTAMP>2018-05-14 00:00:00</RUN_END_TIMESTAMP>
        <INITIATED_BY_USER>Florian Pitters</INITIATED_BY_USER>
        <LOCATION>CERN</LOCATION>
        <COMMENT_DESCRIPTION>CV Test at CERN</COMMENT_DESCRIPTION>
    </RUN>
</HEADER>
<DATA_SET>
        <PART>
            <KIND_OF_PART>HPK Eight Inch 271 Sensor Cell</KIND_OF_PART>
            <SERIAL_NUMBER>HPK_8in_271_4003-010</SERIAL_NUMBER>
        </PART>

        <DATA>
            <VOLTS>25</VOLTS>
            <CPCTNCE_PFRD>103.752</CPCTNCE_PFRD>
            <ERR_CPCTNC_PFRD>0.00333346</ERR_CPCTNC_PFRD>
            <TOT_CURNT_NANOAMP>-1980</TOT_CURNT_NANOAMP>
            <ACTUAL_VOLTS>-25</ACTUAL_VOLTS>
            <ORG_CPCTNC_PFRD>207.857</ORG_CPCTNC_PFRD>
            <TEMP_DEGC>23.5</TEMP_DEGC>
            <HUMIDITY_PRCNT>44.7</HUMIDITY_PRCNT>
            <IMP_OHM>207.857</IMP_OHM>
            <PHS_RAD>23.5</PHS_RAD>
            <TIME_SEC>44.7</TIME_SEC>
            <CELL_NR>40</CELL_NR>
        </DATA>
            /*  .   */
            /*  .   */
            /*  .   */
        <DATA>
            <VOLTS>25</VOLTS>
```

```
            <CPCTNCE_PFRD>103.752</CPCTNCE_PFRD>
            <ERR_CPCTNC_PFRD>0.00333346</ERR_CPCTNC_PFRD>
            <TOT_CURNT_NANOAMP>-1980</TOT_CURNT_NANOAMP>
            <ACTUAL_VOLTS>-25</ACTUAL_VOLTS>
            <ORG_CPCTNC_PFRD>207.857</ORG_CPCTNC_PFRD>
            <TEMP_DEGC>23.5</TEMP_DEGC>
            <HUMIDITY_PRCNT>44.7</HUMIDITY_PRCNT>
            <IMP_OHM>207.857</IMP_OHM>
            <PHS_RAD>23.5</PHS_RAD>
            <TIME_SEC>44.7</TIME_SEC>
            <CELL_NR>40</CELL_NR>
        </DATA>
</DATA_SET>ERR_CURNT_NANOAMP
</ROOT>
```

3.0.2 Let us get the 'CV_FULL.sql' SQL query template

[67]:
```
CV_QUERY_REP = get_query_from_file('CV_FULL.sql')
print(CV_QUERY_REP)
```

```
SELECT SNSRPRT.SERIAL_NUMBER SCRATCHPAD_ID,
SNSRPRT.NAME_LABEL SENSOR_ID,
SNSRCEL.SERIAL_NUMBER SCRATCHPAD_ID_CELL,
HGCSNSRCV.VOLTS,
HGCSNSRCV.CPCTNCE_PFRD,
HGCSNSRCV.ERR_CPCTNC_PFRD,
HGCSNSRCV.TOT_CURNT_NANOAMP,
HGCSNSRCV.ACTUAL_VOLTS,
HGCSNSRCV.ORG_CPCTNC_PFRD,
HGCSNSRCV.TEMP_DEGC,
HGCSNSRCV.HUMIDITY_PRCNT,
HGCSNSRCV.IMP_OHM,
HGCSNSRCV.PHS_RAD,
HGCSNSRCV.TIME_SECS,
HGCSNSRCV.CELL_NR
FROM CMS_HGC_CORE_CONSTRUCT.KINDS_OF_PARTS SNSRKOP
INNER JOIN CMS_HGC_CORE_CONSTRUCT.PARTS SNSRPRT
ON SNSRKOP.KIND_OF_PART_ID = SNSRPRT.KIND_OF_PART_ID
INNER JOIN CMS_HGC_CORE_CONSTRUCT.PHYSICAL_PARTS_TREE SNSRPHPRT
ON SNSRPRT.PART_ID = SNSRPHPRT.PART_PARENT_ID
INNER JOIN CMS_HGC_CORE_CONSTRUCT.PARTS SNSRCEL
ON SNSRPHPRT.PART_ID = SNSRCEL.PART_ID
INNER JOIN CMS_HGC_CORE_CONSTRUCT.KINDS_OF_PARTS CELLKOP
ON SNSRCEL.KIND_OF_PART_ID = CELLKOP.KIND_OF_PART_ID
INNER JOIN CMS_HGC_CORE_COND.COND_DATA_SETS CONDS
ON SNSRCEL.PART_ID = CONDS.PART_ID
INNER JOIN CMS_HGC_CORE_COND.KINDS_OF_CONDITIONS SNSRCVKOC
ON CONDS.KIND_OF_CONDITION_ID = SNSRCVKOC.KIND_OF_CONDITION_ID
```

```
INNER JOIN CMS_HGC_HGCAL_COND.HGC_CERN_SENSOR_CV HGCSNSRCV
ON CONDS.CONDITION_DATA_SET_ID = HGCSNSRCV.CONDITION_DATA_SET_ID

WHERE CONDS.IS_RECORD_DELETED = 'F'
AND SNSRCVKOC.NAME = 'HGC CERN Sensor CV'
AND SNSRCVKOC.IS_RECORD_DELETED = 'F'
AND SNSRPRT.SERIAL_NUMBER = 'SOME_SENSOR_SERIAL_NUMBER'
ORDER BY CELL_NR, VOLTS;
```

In the query above, `AS` is implicit, so `SELECT SNSRPRT.SERIAL_NUMBER CERNSNSR` is the same as `SELECT SNSRPRT.SERIAL_NUMBER AS CERNSNSR`. I do some more selections and renaming (e.g. SERIAL_NUMBER to SCRATCHPAD_ID and ordering based on cell number and voltage. The output here has the same columns that we expect to see on the XML template for a full CV test.

- Query CV table by scratchpad by me influenced from the Umesh one above. This shows all the columns that we uplokaded data for in our XML template, for the sensor with a given serial number (scratchpad ID).

3.0.3 Plot and Interact

[68]:
```
def get_cmap(n, name='hsv'):
    '''Returns a function that maps each index in 0, 1, ..., n-1 to a distinct
    RGB color; the keyword argument name must be a standard mpl colormap name.
 ↪'''
    # colors = mcolors.TABLEAU_COLORS
    # dl = list(colors.items())
    # print(dl[0][1])
    return plt.cm.get_cmap(name, n)
```

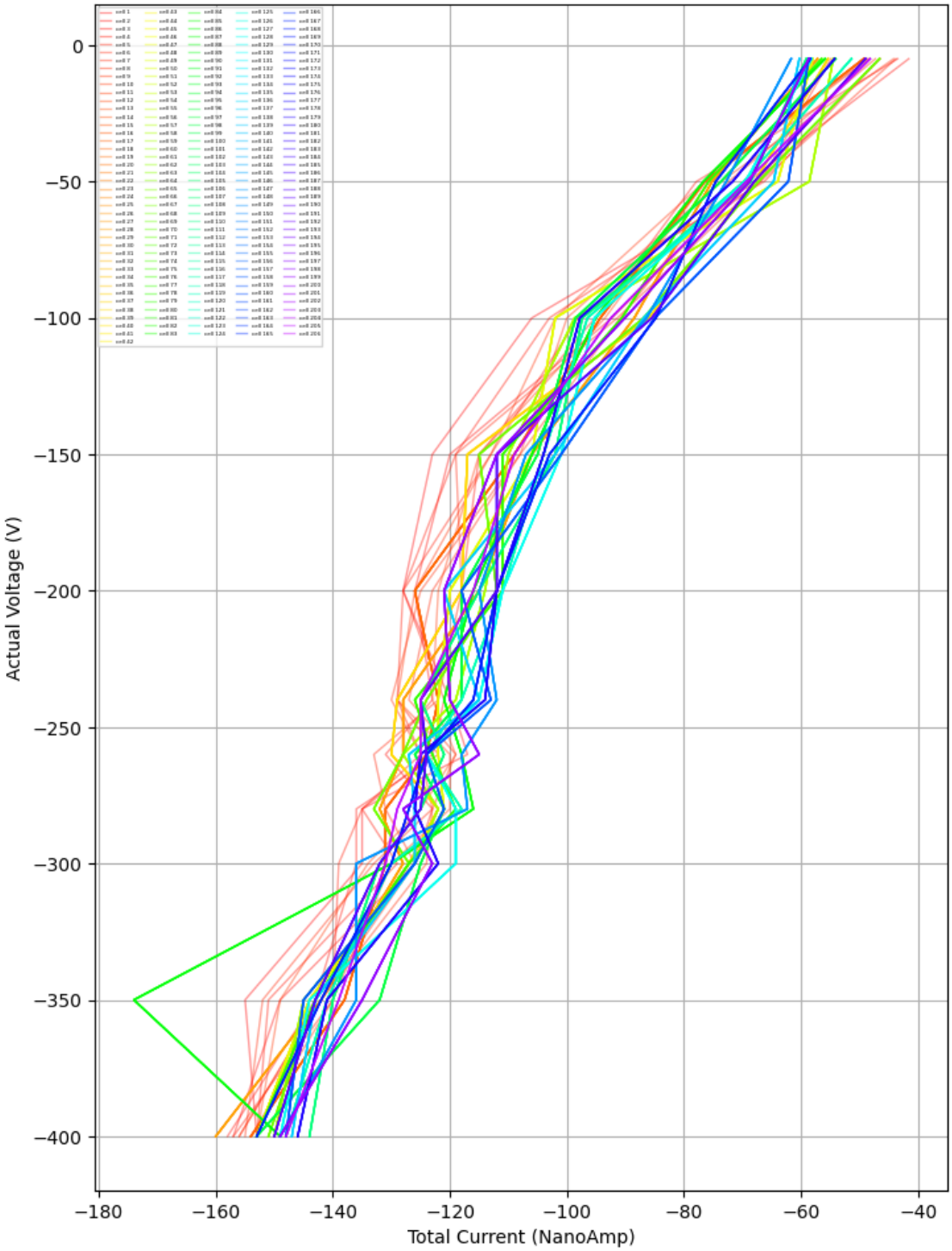
[69]:
```
def plot_CV(sensor_id, saveplot):
    # convert SOME_SENSOR_SERIAL_NUMBER in the file to the sensor ID
    cmd = """sed -i "s/'SOME_SENSOR_SERIAL_NUMBER'/'%s'/g" /home/PASCAL_REPO/
 ↪queries/CV_FULL.sql""" % str(sensor_id)
    os.system(cmd)
    CV_QUERY_REP=get_query_from_file(query_file='CV_FULL.sql')
    # remove the last ";" from the sql command in the file to make it␣
 ↪executable here
    length=len(CV_QUERY_REP)
    # print(length-1)
    # print(CV_QUERY_REP[:-1])
    CV_QUERY_REP=CV_QUERY_REP[:length-2]
    # print(CV_QUERY_REP[:length-3])
    QUERY_OUT = execute_query(CV_QUERY_REP)
    print(QUERY_OUT.head())

    # now convert back to the original SOME_SENSOR_SERIAL_NUMBER
```

```
    cmd = """sed -i "s/'%s'/'SOME_SENSOR_SERIAL_NUMBER'/g" /home/PASCAL_REPO/
↪queries/CV_FULL.sql""" % str(sensor_id)
    os.system(cmd)

    start_time = time.perf_counter()
    #measure the time for plotting
    max_cells=QUERY_OUT['CELL_NR'].max()
    # serial_number='100383'#aka SCRATCHPAD_ID
    serial_number=sensor_id
    fig,ax=plt.subplots(figsize=(25/3,35/3))
    # colors = mcolors.TABLEAU_COLORS
    # color_list = list(colors.items())
    # color_index = 0
    # cmap = get_cmap(QUERY_OUT.shape[1])
    cmap = get_cmap(255)

    # index = (index + 1) % len(my_list)
    for ind, cell_nr in enumerate(range(1,max_cells)):
        # keep looping back and forth in the colors list
        # color_index = (color_index + 1) % len(colors)
        # color = colors[color_index]
        # print('color index = ', color_index)
        plt.plot(QUERY_OUT['TOT_CURNT_NANOAMP'][QUERY_OUT['CELL_NR']==cell_nr],
                 QUERY_OUT['ACTUAL_VOLTS'][QUERY_OUT['CELL_NR']==cell_nr],
                 label = f'cell {cell_nr}',
                alpha=0.4,
                 # color = color_list[color_index][1],
                 # Randomly pick out a color from the cmap
                 color = cmap(ind),
                 linewidth=1.0


                )
        plt.ylabel('Actual Voltage (V)'); plt.xlabel('Total Current (NanoAmp)')
        plt.legend(ncol=5, fontsize=3)
    plt.grid()
    fig.suptitle(serial_number)
    plt.show()
    end_time = time.perf_counter()
    run_time = end_time - start_time
    print(f"Finished all plotting in {run_time:.4f} secs")
    if saveplot != 'False':
        imagename= '%s.pdf' % str(saveplot)
        print('\nOkay, saveing image to %s\n', os.path.join(IMAGE_DIR,␣
↪imagename))
        plt.savefig(os.path.join(IMAGE_DIR, imagename))
```

[133]:
```
plot_CV(sensor_id='100113', saveplot='False')
```

```
COLUMN NAMES:
 ['SCRATCHPAD_ID', 'SENSOR_ID', 'SCRATCHPAD_ID_CELL', 'VOLTS', 'CPCTNCE_PFRD',
'ERR_CPCTNC_PFRD', 'TOT_CURNT_NANOAMP', 'ACTUAL_VOLTS', 'ORG_CPCTNC_PFRD',
'TEMP_DEGC', 'HUMIDITY_PRCNT', 'IMP_OHM', 'PHS_RAD', 'TIME_SECS', 'CELL_NR']

FINISHED 'execute_query' in 1.6268 SECS
  SCRATCHPAD_ID SENSOR_ID SCRATCHPAD_ID_CELL  VOLTS  CPCTNCE_PFRD  \
0        100113   N8738_1           100113_0 -400.0      198.7342
1        100113   N8738_1           100113_0 -350.0      198.8951
2        100113   N8738_1           100113_0 -300.0      198.6748
3        100113   N8738_1           100113_0 -280.0      198.6927
4        100113   N8738_1           100113_0 -260.0      198.7315

   ERR_CPCTNC_PFRD  TOT_CURNT_NANOAMP  ACTUAL_VOLTS  ORG_CPCTNC_PFRD  \
0         0.012571             -153.0       -400.00         198.7342
1         0.018476             -155.0       -350.00         198.8951
2         0.010903             -135.0       -300.00         198.6748
3         0.010768             -135.0       -280.00         198.6927
4         0.010782             -117.0       -260.01         198.7315

   TEMP_DEGC  HUMIDITY_PRCNT   IMP_OHM   PHS_RAD  TIME_SECS  CELL_NR
0       23.5            14.4  452706.7 -1.085433    3504.51        1
1       23.5            14.5  452473.3 -1.084877    3144.23        1
2       23.5            14.7  452656.0 -1.086213    2792.69        1
3       23.5            15.0  452578.7 -1.086367    2447.97        1
4       23.5            15.2  452444.3 -1.086560    2104.63        1
```

100113


0
−50
−100
−150
−200
−250
−300
−350
−400
Actual Voltage (V)
−180
−160
−140
−120
−100
−80
−60
−40
Total Current (NanoAmp)

```
Finished all plotting in 22.0707 secs
```

[48]:
```
interact(plot_CV, sensor_id='100113', saveplot='False')
```

```
interactive(children=(Text(value='100113', description='sensor_id'),␣
↪Text(value='False', description='saveplot…
```

[48]:
```
<function __main__.plot_CV(sensor_id, saveplot)>
```

---

4 Example 4: Plot IV test for all cells at all voltage steps

4.0.1 XML Template for Table: HGC_CERN_SENSOR_IV

Kind of condition: HGC CERN Sensor IV Test

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ROOT xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<HEADER>
    <TYPE>
        <EXTENSION_TABLE_NAME>HGC_CERN_SENSOR_IV</EXTENSION_TABLE_NAME>
        <NAME>HGC CERN Sensor IV Test</NAME>
    </TYPE>

    <RUN>
        <RUN_NAME>Your Run Name</RUN_NAME>
<!-- Enter your timestamp -->
        <RUN_BEGIN_TIMESTAMP>2018-05-14 00:00:00</RUN_BEGIN_TIMESTAMP>
        <RUN_END_TIMESTAMP>2018-05-14 00:00:00</RUN_END_TIMESTAMP>
        <INITIATED_BY_USER>Your Name</INITIATED_BY_USER>
        <LOCATION>CERN</LOCATION>
        <COMMENT_DESCRIPTION>Your Comments</COMMENT_DESCRIPTION>
    </RUN>
</HEADER>
<DATA_SET>
        <PART>
            <KIND_OF_PART>120um Si Sensor HD Full</KIND_OF_PART>
            <SERIAL_NUMBER>XXXXXXXXXXXXXXXXXXX</SERIAL_NUMBER>
        </PART>

        <DATA>
<VOLTS>-25</VOLTS>
<CURNT_NANOAMP>7.609905</CURNT_NANOAMP>
<ERR_CURNT_NANOAMP>0.01653122</ERR_CURNT_NANOAMP>
<TOT_CURNT_NANOAMP>-2000</TOT_CURNT_NANOAMP>
<ACTUAL_VOLTS>-25</ACTUAL_VOLTS>
```

```
<TIME_SECS>7.609905</TIME_SECS>
<TEMP_DEGC>23</TEMP_DEGC>
<HUMIDITY_PRCNT>7.609905</HUMIDITY_PRCNT>
<CELL_NR>YYYY</CELL_NR>
        </DATA>
                .
                .
                .
        <DATA>
<VOLTS>-225</VOLTS>
<CURNT_NANOAMP>7.609905</CURNT_NANOAMP>
<ERR_CURNT_NANOAMP>0.01653122</ERR_CURNT_NANOAMP>
<TOT_CURNT_NANOAMP>-2000</TOT_CURNT_NANOAMP>
<ACTUAL_VOLTS>-25</ACTUAL_VOLTS>
<TIME_SECS>7.609905</TIME_SECS>
<TEMP_DEGC>23</TEMP_DEGC>
<HUMIDITY_PRCNT>7.609905</HUMIDITY_PRCNT>
<CELL_NR>YYYY</CELL_NR>
        </DATA>
</DATA_SET>

</ROOT>
```

[83]:
```
def plot_IV(sensor_id, saveplot):
    # convert SOME_SENSOR_SERIAL_NUMBER in the file to the sensor ID
    cmd = """sed -i "s/'SOME_SENSOR_SERIAL_NUMBER'/'%s'/g" /home/PASCAL_REPO/
 ↪queries/IV_FULL.sql""" % str(sensor_id)
    os.system(cmd)
    IV_QUERY_REP=get_query_from_file(query_file='IV_FULL.sql')

    # remove the last ";" from the sql command in the file to make it␣
 ↪executable here
    length=len(IV_QUERY_REP)
    # print(length-1)
    # print(CV_QUERY_REP[:-1])
    IV_QUERY_REP=IV_QUERY_REP[:length-1]
    # print(CV_QUERY_REP[:length-3])
    QUERY_OUT = execute_query(IV_QUERY_REP)
    print(QUERY_OUT.head())

    # now convert back to the original SOME_SENSOR_SERIAL_NUMBER
    cmd = """sed -i "s/'%s'/'SOME_SENSOR_SERIAL_NUMBER'/g" /home/PASCAL_REPO/
 ↪queries/IV_FULL.sql""" % str(sensor_id)
    os.system(cmd)

    start_time = time.perf_counter()
    #measure the time for plotting
```

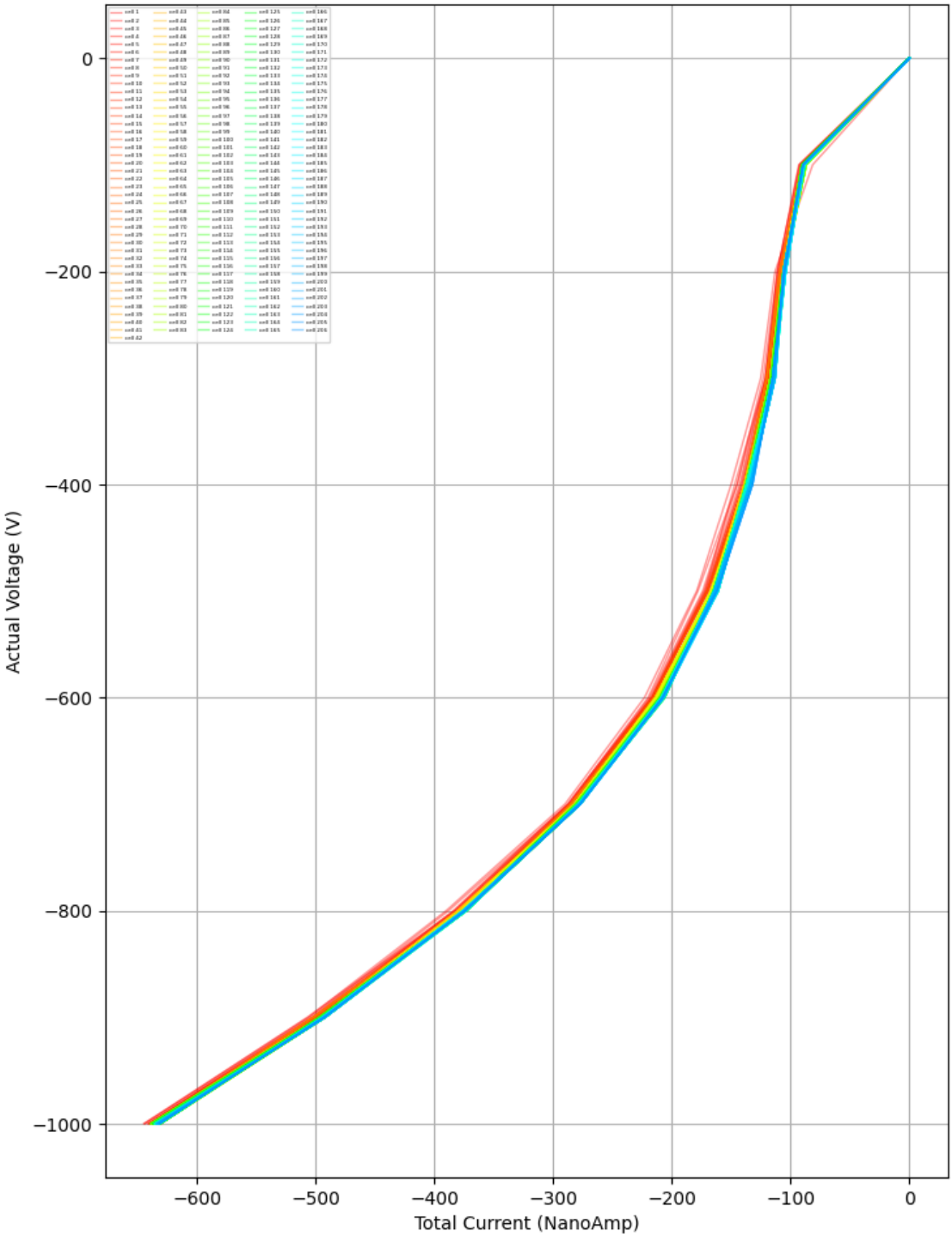
```
    max_cells=QUERY_OUT['CELL_NR'].max()
    # serial_number='100383'#aka SCRATCHPAD_ID
    serial_number=sensor_id
    fig,ax=plt.subplots(figsize=(25/3,35/3))
    # colors = mcolors.TABLEAU_COLORS
    # color_list = list(colors.items())
    # color_index = 0
    # cmap = get_cmap(QUERY_OUT.shape[1])
    cmap = get_cmap(355)

    # index = (index + 1) % len(my_list)
    for ind, cell_nr in enumerate(range(1,max_cells)):
        # keep looping back and forth in the colors list
        # color_index = (color_index + 1) % len(colors)
        # color = colors[color_index]
        # print('color index = ', color_index)
        plt.plot(QUERY_OUT['TOT_CURNT_NANOAMP'][QUERY_OUT['CELL_NR']==cell_nr],
                 QUERY_OUT['ACTUAL_VOLTS'][QUERY_OUT['CELL_NR']==cell_nr],
                 label = f'cell {cell_nr}',
                alpha=0.4,
                 # color = color_list[color_index][1],
                 # Randomly pick out a color from the cmap
                 color = cmap(ind),
                 linewidth=1.0


                )
        plt.ylabel('Actual Voltage (V)'); plt.xlabel('Total Current (NanoAmp)')
        plt.legend(ncol=5, fontsize=3)
    plt.grid()
    fig.suptitle(serial_number)
    plt.show()
    end_time = time.perf_counter()
    run_time = end_time - start_time
    print(f"Finished all plotting in {run_time:.4f} secs")
    if saveplot != 'False':
        imagename= '%s.pdf' % str(saveplot)
        print('\nOkay, saveing image to %s\n', os.path.join(IMAGE_DIR,␣
↪imagename))
        plt.savefig(os.path.join(IMAGE_DIR, imagename))
```

[134]:
```
plot_IV( sensor_id='100383', saveplot='False')
```

```
COLUMN NAMES:
 ['SCRATCHPAD_ID', 'SENSOR_ID', 'SCRATCHPAD_ID_CELL', 'VOLTS', 'CURNT_NANOAMP',
'ERR_CURNT_NANOAMP', 'TOT_CURNT_NANOAMP', 'ACTUAL_VOLTS', 'TIME_SECS',
```

```
'TEMP_DEGC', 'HUMIDITY_PRCNT', 'CELL_NR']

FINISHED 'execute_query' in 1.3529 SECS
  SCRATCHPAD_ID SENSOR_ID SCRATCHPAD_ID_CELL   VOLTS  CURNT_NANOAMP  \
0        100383  OBA46983           100383_0 -1000.0      -2.417423
1        100383  OBA46983           100383_0  -900.0      -1.973708
2        100383  OBA46983           100383_0  -800.0      -1.577893
3        100383  OBA46983           100383_0  -700.0      -1.249481
4        100383  OBA46983           100383_0  -600.0      -1.003410

   ERR_CURNT_NANOAMP  TOT_CURNT_NANOAMP  ACTUAL_VOLTS  TIME_SECS  TEMP_DEGC  \
0           0.013538             -645.0      -1000.00    3424.23       20.5
1           0.000896             -506.0       -900.07    3071.97       20.5
2           0.000640             -390.0       -800.05    2718.58       20.5
3           0.000661             -290.0       -700.05    2375.16       20.5
4           0.000185             -223.0       -600.08    2035.23       20.5

   HUMIDITY_PRCNT  CELL_NR
0             8.7        1
1             9.5        1
2            10.4        1
3            11.5        1
4            12.7        1
```

100383


0
−200
−400
−600
−800
−1000
Actual Voltage (V)
−600
−500
−400
−300
−200
−100
0
Total Current (NanoAmp)

```
Finished all plotting in 22.1127 secs
```

[84]:
```
interact(plot_IV, sensor_id='100383', saveplot='False')
```

```
interactive(children=(Text(value='100383', description='sensor_id'),␣
↪Text(value='False', description='saveplot…
```

[84]:
```
<function __main__.plot_IV(sensor_id, saveplot)>
```

---

5 More Examples

5.0.1 - You can execute these from the command line: do `python pascalutils.py --help` to see what you're able to execute from the terminal.

5.0.2 Show IV Summary Data from the `CMS_HGC_HGCAL_COND.HGC_CERN_SENSOR_IV_SUMRY` table

[124]:
```
QUERY_IV_SUM="""
SELECT SNSRPRT.SERIAL_NUMBER SCRATCHPAD_ID,
SNSRPRT.NAME_LABEL SENSOR_ID,
SNSRCEL.SERIAL_NUMBER SCRATCHPAD_ID_CELL,
HGCSNSRCV.PASS,
FROM CMS_HGC_CORE_CONSTRUCT.KINDS_OF_PARTS SNSRKOP
INNER JOIN CMS_HGC_CORE_CONSTRUCT.PARTS SNSRPRT
ON SNSRKOP.KIND_OF_PART_ID = SNSRPRT.KIND_OF_PART_ID
INNER JOIN CMS_HGC_CORE_CONSTRUCT.PHYSICAL_PARTS_TREE SNSRPHPRT
ON SNSRPRT.PART_ID = SNSRPHPRT.PART_PARENT_ID
INNER JOIN CMS_HGC_CORE_CONSTRUCT.PARTS SNSRCEL
ON SNSRPHPRT.PART_ID = SNSRCEL.PART_ID
INNER JOIN CMS_HGC_CORE_COND.COND_DATA_SETS CONDS
ON SNSRCEL.PART_ID = CONDS.PART_ID
INNER JOIN CMS_HGC_CORE_COND.KINDS_OF_CONDITIONS SNSRCVKOC
ON CONDS.KIND_OF_CONDITION_ID = SNSRCVKOC.KIND_OF_CONDITION_ID

INNER JOIN CMS_HGC_HGCAL_COND.HGC_CERN_SENSOR_IV_SUMRY HGCSNSRCV
ON CONDS.CONDITION_DATA_SET_ID = HGCSNSRCV.CONDITION_DATA_SET_ID

WHERE CONDS.IS_RECORD_DELETED = 'F'
AND SNSRCVKOC.NAME = 'HGC CERN Sensor IV Summary'
AND SNSRCVKOC.IS_RECORD_DELETED = 'F'
"""
```

[131]:
```
def show_IV_summary(sensor_ID):
    #just print the dataframe from the execute_query(<IV SUMMARY DF for␣
↪Sensor_ID>)
```

```
    # QUERY="""SELECT * FROM CMS_HGC_CORE_COND.HGC_CERN_SENSOR_IV_SUMRY WHERE␣
 ↪SERIAL_NUMBER= '%s' """ % str(sensor_ID)
    QUERY="""SELECT * FROM CMS_HGC_HGCAL_COND.HGC_CERN_SENSOR_IV_SUMRY """
    # QUERY=QUERY_IV_SUM

    OUT = execute_query(QUERY)
    return OUT

def show_CV_summary(sensor_ID):
    #just print the dataframe from the execute_query(<IV SUMMARY DF for␣
 ↪Sensor_ID>)
    pass
```

[132]:
```
interact(show_IV_summary, sensor_ID='100113')
```

```
interactive(children=(Text(value='100113', description='sensor_ID'), Output()),␣
 ↪_dom_classes=('widget-interact…
```

[132]:
```
<function __main__.show_IV_summary(sensor_ID)>
```

[59]:
```
QUERY_CV_ALL="""
SELECT SNSRPRT.SERIAL_NUMBER SCRATCHPAD_ID,
SNSRPRT.NAME_LABEL SENSOR_ID,
SNSRCEL.SERIAL_NUMBER SCRATCHPAD_ID_CELL,
HGCSNSRCV.VOLTS,
HGCSNSRCV.CPCTNCE_PFRD,
HGCSNSRCV.ERR_CPCTNC_PFRD,
HGCSNSRCV.TOT_CURNT_NANOAMP,
HGCSNSRCV.ACTUAL_VOLTS,
HGCSNSRCV.ORG_CPCTNC_PFRD,
HGCSNSRCV.TEMP_DEGC,
HGCSNSRCV.HUMIDITY_PRCNT,
HGCSNSRCV.IMP_OHM,
HGCSNSRCV.PHS_RAD,
HGCSNSRCV.TIME_SECS,
HGCSNSRCV.CELL_NR
FROM CMS_HGC_CORE_CONSTRUCT.KINDS_OF_PARTS SNSRKOP
INNER JOIN CMS_HGC_CORE_CONSTRUCT.PARTS SNSRPRT
ON SNSRKOP.KIND_OF_PART_ID = SNSRPRT.KIND_OF_PART_ID
INNER JOIN CMS_HGC_CORE_CONSTRUCT.PHYSICAL_PARTS_TREE SNSRPHPRT
ON SNSRPRT.PART_ID = SNSRPHPRT.PART_PARENT_ID
INNER JOIN CMS_HGC_CORE_CONSTRUCT.PARTS SNSRCEL
ON SNSRPHPRT.PART_ID = SNSRCEL.PART_ID
INNER JOIN CMS_HGC_CORE_CONSTRUCT.KINDS_OF_PARTS CELLKOP
ON SNSRCEL.KIND_OF_PART_ID = CELLKOP.KIND_OF_PART_ID
INNER JOIN CMS_HGC_CORE_COND.COND_DATA_SETS CONDS
ON SNSRCEL.PART_ID = CONDS.PART_ID
```

```
INNER JOIN CMS_HGC_CORE_COND.KINDS_OF_CONDITIONS SNSRCVKOC
ON CONDS.KIND_OF_CONDITION_ID = SNSRCVKOC.KIND_OF_CONDITION_ID
INNER JOIN CMS_HGC_HGCAL_COND.HGC_CERN_SENSOR_CV HGCSNSRCV
ON CONDS.CONDITION_DATA_SET_ID = HGCSNSRCV.CONDITION_DATA_SET_ID

WHERE CONDS.IS_RECORD_DELETED = 'F'
AND SNSRCVKOC.NAME = 'HGC CERN Sensor CV'
AND SNSRCVKOC.IS_RECORD_DELETED = 'F'
AND SNSRPRT.SERIAL_NUMBER = '100383'
ORDER BY CELL_NR, VOLTS
"""
```

[60]:
```
CV_DF_100383=execute_query(QUERY_CV_ALL)
```

```
COLUMN NAMES:
 ['SCRATCHPAD_ID', 'SENSOR_ID', 'SCRATCHPAD_ID_CELL', 'VOLTS', 'CPCTNCE_PFRD',
'ERR_CPCTNC_PFRD', 'TOT_CURNT_NANOAMP', 'ACTUAL_VOLTS', 'ORG_CPCTNC_PFRD',
'TEMP_DEGC', 'HUMIDITY_PRCNT', 'IMP_OHM', 'PHS_RAD', 'TIME_SECS', 'CELL_NR']
Finished 'execute_query' in 0.5936 secs
```

[37]:
```
CV_DF_100383
```

[37]:

	SCRATCHPAD_ID	SENSOR_ID	SCRATCHPAD_ID_CELL	VOLTS	CPCTNCE_PFRD	\
0	100383	OBA46983	100383_0	-400.0	197.2013	
1	100383	OBA46983	100383_0	-350.0	197.2013	
2	100383	OBA46983	100383_0	-300.0	197.1981	
3	100383	OBA46983	100383_0	-280.0	197.2315	
4	100383	OBA46983	100383_0	-260.0	197.2694	
..	…	…	…	…	…	
226	100383	OBA46983	100383_0	-200.0	185.7436	
227	100383	OBA46983	100383_0	-150.0	185.7315	
228	100383	OBA46983	100383_0	-100.0	185.7157	
229	100383	OBA46983	100383_0	-50.0	185.6920	
230	100383	OBA46983	100383_0	-5.0	187.2945	

	ERR_CPCTNC_PFRD	TOT_CURNT_NANOAMP	ACTUAL_VOLTS	ORG_CPCTNC_PFRD	\
0	0.005712	-140.0	-400.01	197.2013	
1	0.008718	-140.0	-350.01	197.2013	
2	0.004396	-113.0	-300.06	197.1981	
3	0.002299	-120.0	-280.04	197.2315	
4	0.005476	-123.0	-260.02	197.2694	
..	…	…	…	…	
226	0.001425	-110.0	-199.99	185.7436	
227	0.004674	-96.6	-150.03	185.7315	
228	0.002590	-87.1	-99.99	185.7157	
229	0.005135	-70.7	-49.98	185.6920	

```
230         0.003127              -42.0         -5.00         187.2945

     TEMP_DEGC  HUMIDITY_PRCNT   IMP_OHM   PHS_RAD  TIME_SECS  CELL_NR
0         20.6             6.9  455926.3 -1.086680     736.73        1
1         20.6             7.0  455904.0 -1.086773     663.56        1
2         20.6             7.0  455873.0 -1.086933     590.44        1
3         20.6             7.1  455791.7 -1.086950     522.53        1
4         20.6             7.2  455691.3 -1.087003     454.63        1
..         …               …        …        …         …            …
226       20.6             7.4  495557.3 -1.044223     348.05      201
227       20.6             7.4  495571.3 -1.044287     274.70      201
228       20.6             7.5  495588.7 -1.044373     201.49      201
229       20.6             7.7  495610.7 -1.044517     128.16      201
230       20.6             7.7  487855.7 -1.057053      56.02      201

[231 rows x 15 columns]
```

plotting function with option to save

- Query IV table by scratchpad by me influenced from the Umesh one above. This shows all the columns that we uplokaded data for in our XML template, for the sensor with serial number (scratchpad ID) 100383:

```
SELECT SNSRPRT.SERIAL_NUMBER SCRATCHPAD_ID,
SNSRPRT.NAME_LABEL SENSOR_ID,
SNSRCEL.SERIAL_NUMBER SCRATCHPAD_ID_CELL,
!! <DATA>
HGCSNSRIV.VOLTS,
HGCSNSRIV.CURNT_NANOAMP,
HGCSNSRIV.ERR_CURNT_NANOAMP,
HGCSNSRIV.TOT_CURNT_NANOAMP,
HGCSNSRIV.ACTUAL_VOLTS,
HGCSNSRIV.TIME_SECS,
HGCSNSRIV.TEMP_DEGC,
HGCSNSRIV.HUMIDITY_PRCNT,
HGCSNSRIV.CELL_NR
FROM CMS_HGC_CORE_CONSTRUCT.KINDS_OF_PARTS SNSRKOP
INNER JOIN CMS_HGC_CORE_CONSTRUCT.PARTS SNSRPRT
ON SNSRKOP.KIND_OF_PART_ID = SNSRPRT.KIND_OF_PART_ID
INNER JOIN CMS_HGC_CORE_CONSTRUCT.PHYSICAL_PARTS_TREE SNSRPHPRT
ON SNSRPRT.PART_ID = SNSRPHPRT.PART_PARENT_ID
INNER JOIN CMS_HGC_CORE_CONSTRUCT.PARTS SNSRCEL
ON SNSRPHPRT.PART_ID = SNSRCEL.PART_ID
INNER JOIN CMS_HGC_CORE_CONSTRUCT.KINDS_OF_PARTS CELLKOP
ON SNSRCEL.KIND_OF_PART_ID = CELLKOP.KIND_OF_PART_ID
INNER JOIN CMS_HGC_CORE_COND.COND_DATA_SETS CONDS
ON SNSRCEL.PART_ID = CONDS.PART_ID
```

```
INNER JOIN CMS_HGC_CORE_COND.KINDS_OF_CONDITIONS SNSRIVKOC
ON CONDS.KIND_OF_CONDITION_ID = SNSRIVKOC.KIND_OF_CONDITION_ID
INNER JOIN CMS_HGC_HGCAL_COND.HGC_CERN_SENSOR_IV HGCSNSRIV
ON CONDS.CONDITION_DATA_SET_ID = HGCSNSRIV.CONDITION_DATA_SET_ID

WHERE CONDS.IS_RECORD_DELETED = 'F'
AND SNSRIVKOC.NAME = 'HGC CERN Sensor IV'
AND SNSRIVKOC.IS_RECORD_DELETED = 'F'
AND SNSRPRT.SERIAL_NUMBER = '100383'
ORDER BY CELL_NR, VOLTS;
```

[24]:
```
IV_QUERY_ALL="""SELECT SNSRPRT.SERIAL_NUMBER SCRATCHPAD_ID,
SNSRPRT.NAME_LABEL SENSOR_ID,
SNSRCEL.SERIAL_NUMBER SCRATCHPAD_ID_CELL,
!! <DATA>
HGCSNSRIV.VOLTS,
HGCSNSRIV.CURNT_NANOAMP,
HGCSNSRIV.ERR_CURNT_NANOAMP,
HGCSNSRIV.TOT_CURNT_NANOAMP,
HGCSNSRIV.ACTUAL_VOLTS,
HGCSNSRIV.TIME_SECS,
HGCSNSRIV.TEMP_DEGC,
HGCSNSRIV.HUMIDITY_PRCNT,
HGCSNSRIV.CELL_NR
FROM CMS_HGC_CORE_CONSTRUCT.KINDS_OF_PARTS SNSRKOP
INNER JOIN CMS_HGC_CORE_CONSTRUCT.PARTS SNSRPRT
ON SNSRKOP.KIND_OF_PART_ID = SNSRPRT.KIND_OF_PART_ID
INNER JOIN CMS_HGC_CORE_CONSTRUCT.PHYSICAL_PARTS_TREE SNSRPHPRT
ON SNSRPRT.PART_ID = SNSRPHPRT.PART_PARENT_ID
INNER JOIN CMS_HGC_CORE_CONSTRUCT.PARTS SNSRCEL
ON SNSRPHPRT.PART_ID = SNSRCEL.PART_ID
INNER JOIN CMS_HGC_CORE_CONSTRUCT.KINDS_OF_PARTS CELLKOP
ON SNSRCEL.KIND_OF_PART_ID = CELLKOP.KIND_OF_PART_ID
INNER JOIN CMS_HGC_CORE_COND.COND_DATA_SETS CONDS
ON SNSRCEL.PART_ID = CONDS.PART_ID
INNER JOIN CMS_HGC_CORE_COND.KINDS_OF_CONDITIONS SNSRIVKOC
ON CONDS.KIND_OF_CONDITION_ID = SNSRIVKOC.KIND_OF_CONDITION_ID
INNER JOIN CMS_HGC_HGCAL_COND.HGC_CERN_SENSOR_IV HGCSNSRIV
ON CONDS.CONDITION_DATA_SET_ID = HGCSNSRIV.CONDITION_DATA_SET_ID

WHERE CONDS.IS_RECORD_DELETED = 'F'
AND SNSRIVKOC.NAME = 'HGC CERN Sensor IV'
AND SNSRIVKOC.IS_RECORD_DELETED = 'F'
AND SNSRPRT.SERIAL_NUMBER = '100383'
ORDER BY CELL_NR, VOLTS
"""
```

```
IV_DF_100383=execute_query(QUERY_CV_ALL)
```

```
COLUMN NAMES:
 ['SCRATCHPAD_ID', 'SENSOR_ID', 'SCRATCHPAD_ID_CELL', 'VOLTS', 'CPCTNCE_PFRD',
'ERR_CPCTNC_PFRD', 'TOT_CURNT_NANOAMP', 'ACTUAL_VOLTS', 'ORG_CPCTNC_PFRD',
'TEMP_DEGC', 'HUMIDITY_PRCNT', 'IMP_OHM', 'PHS_RAD', 'TIME_SECS', 'CELL_NR']
```

[25]:
```
IV_DF_100383
```

[25]:

	SCRATCHPAD_ID	SENSOR_ID	SCRATCHPAD_ID_CELL	VOLTS	CPCTNCE_PFRD	\
0	200144	N8741_9	200144_0	-200.0	206.7537	
1	200144	N8741_9	200144_0	-170.0	206.8677	
2	200144	N8741_9	200144_0	-140.0	206.9698	
3	200144	N8741_9	200144_0	-120.0	207.0888	
4	200144	N8741_9	200144_0	-100.0	207.9651	
…	…	…	…	…	…	
1651	200144	N8741_9	200144_0	-120.0	376.4265	
1652	200144	N8741_9	200144_0	-100.0	376.4440	
1653	200144	N8741_9	200144_0	-80.0	376.4557	
1654	200144	N8741_9	200144_0	-50.0	376.4726	
1655	200144	N8741_9	200144_0	-5.0	379.6343	

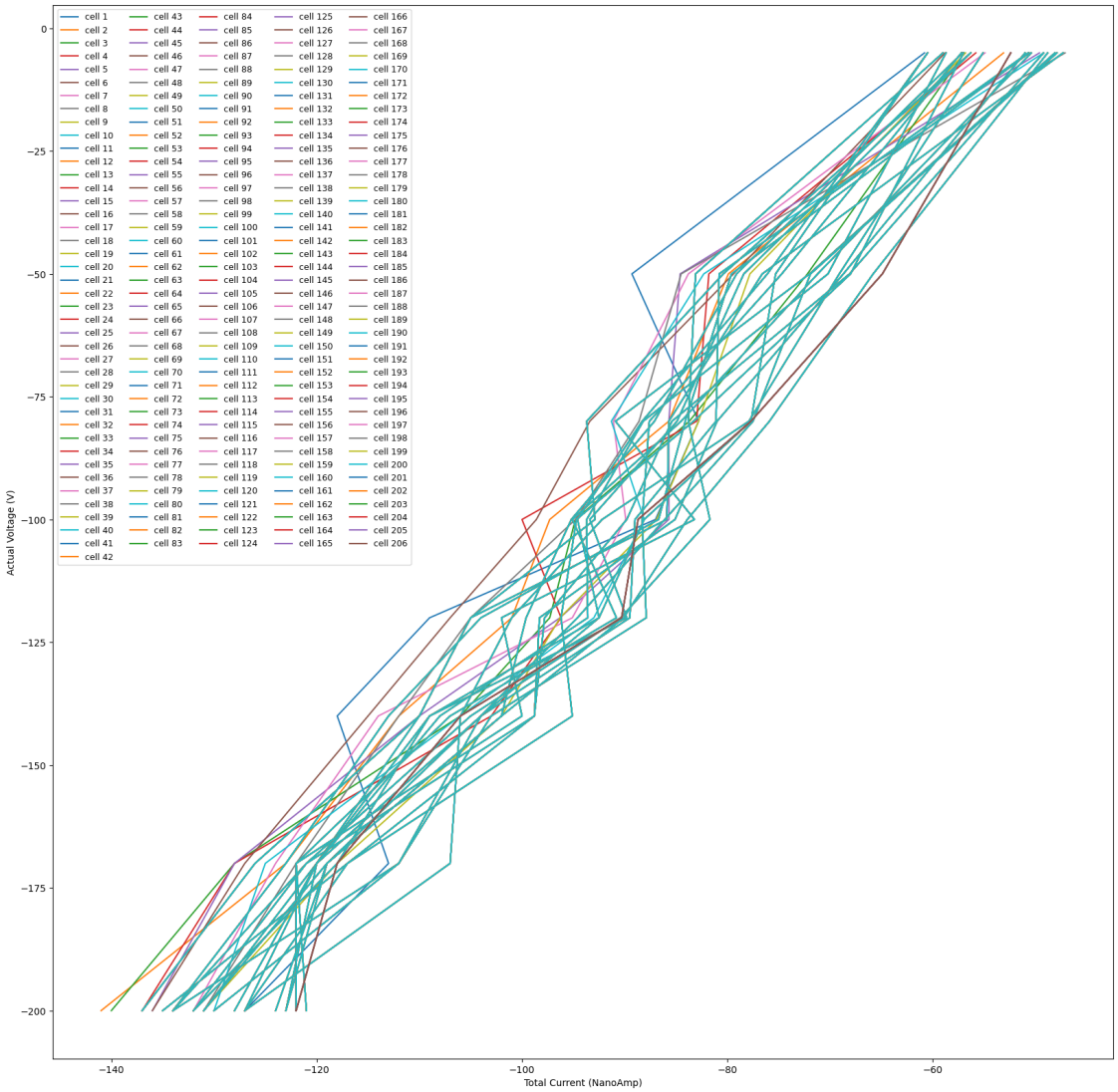
	ERR_CPCTNC_PFRD	TOT_CURNT_NANOAMP	ACTUAL_VOLTS	ORG_CPCTNC_PFRD	\
0	0.008206	-127.0	-200.00	206.7537	
1	0.008204	-113.0	-170.01	206.8677	
2	0.011109	-118.0	-140.01	206.9698	
3	0.010926	-109.0	-120.01	207.0888	
4	0.009217	-87.0	-100.00	207.9651	
…	…	…	…	…	
1651	0.015038	-90.3	-120.01	376.4265	
1652	0.012988	-88.7	-100.00	376.4440	
1653	0.017328	-77.7	-80.00	376.4557	
1654	0.014044	-64.9	-50.00	376.4726	
1655	0.008660	-52.4	-5.00	379.6343	

	TEMP_DEGC	HUMIDITY_PRCNT	IMP_OHM	PHS_RAD	TIME_SECS	CELL_NR
0	25.0	4.1	429026.7	-1.113217	2433.29	1
1	25.0	4.0	428748.0	-1.113417	2086.77	1
2	25.0	4.1	428464.7	-1.113757	1740.31	1
3	25.0	4.1	428149.0	-1.114087	1396.65	1
4	25.0	4.3	425807.3	-1.116663	1053.54	1
…	…	…	…	…	…	…
1651	25.0	4.1	217549.3	-1.332513	1707.02	207
1652	25.0	4.1	217538.3	-1.332530	1363.39	207
1653	25.0	4.3	217530.0	-1.332560	1020.25	207
1654	24.9	4.4	217519.0	-1.332583	674.62	207
1655	24.9	4.4	215574.7	-1.335133	330.27	207

```
[1656 rows x 15 columns]
```

[26]:
```
max_cells=IV_DF_100383['CELL_NR'].max()
serial_number='100383'#aka SCRATCHPAD_ID
fig,ax=plt.subplots(figsize=(20,20))
for cell_nr in range(1,max_cells):
    plt.
 ↪plot(IV_DF_100383['TOT_CURNT_NANOAMP'][IV_DF_100383['CELL_NR']==cell_nr],␣
 ↪IV_DF_100383['ACTUAL_VOLTS'][IV_DF_100383['CELL_NR']==cell_nr], label =␣
 ↪f'cell {cell_nr}')
    plt.ylabel('Actual Voltage (V)'); plt.xlabel('Total Current (NanoAmp)')
    plt.legend(ncol=5, fontsize=9)
fig.suptitle(serial_number)
```

[26]:
```
Text(0.5, 0.98, '100383')
```



- See the names of all the tables in the `CMS_HGC_HGCAL_COND` account (and the number of rows in each)

```
select table_name, num_rows
from all_tables
  where owner='CMS_HGC_HGCAL_COND';
```

- see everything in the `hcg_cern_sensor_ivl` table:

```
select * from CMS_HGC_HGCAL_COND.hgc_cern_sensor_iv;
```

- See the uploaded registered parts (wafers), ordered by the time they were uploaded to the database:

```
select * from CMS_HGC_CORE_CONSTRUCT.PARTS order by RECORD_INSERTION_TIME ASC;
```

- See the registered wafer that has serial number "100113":

```
select * from CMS_HGC_CORE_CONSTRUCT.PARTS where SERIAL_NUMBER='100113';
```

- See the uploaded wafer kind of part ID that was uploaded by the user "`Alex%`" (i.e. it matches any user name that starts with "Alex").

```
select  KIND_OF_PART_ID, NAME_LABEL
from CMS_HGC_CORE_CONSTRUCT.PARTS Where RECORD_INSERTION_USER LIKE 'Alex%';
```

For the HGCAL data we sometimes need to use some `Inner join` commands in our SQL query. Basically `inner join` lets you join your initial table with another table, at a particular field that is the same in both tables.

- stupid way to see the first CV table that I uploaded.

```
select * from CMS_HGC_HGCAL_COND.HGC_CERN_SENSOR_CV
INNER JOIN CMS_HGC_CORE_COND.COND_DATA_SETS
ON CMS_HGC_HGCAL_COND.HGC_CERN_SENSOR_CV.CONDITION_DATA_SET_ID = CMS_HGC_CORE_COND.COND_DATA_SE
where CMS_HGC_CORE_COND.COND_DATA_SETS.RECORD_INSERTION_USER LIKE '%Ali%'
ORDER BY CELL_NR;
```

[ ]:
```
QUERY2_DF = execute_query(QUERY2)
QUERY2_DF.head()
df = execute_query("""SELECT * FROM CMS_HGC_CORE_COND.HGC_CERN_SENSOR_IV where␣
 ↪SERIAL_NUMBER='100113"""")
df.head()
```