In [35]:
```
%matplotlib notebook

import matplotlib.pyplot as plt #import matplotlib library
from matplotlib.animation import FuncAnimation # import animations for matplot

from datetime import datetime # datetime library

import pandas as pd # pandas library

from sqlalchemy.dialects.postgresql import JSON # Allows SQLAlchemy to parse t
from sqlalchemy import ( # Sqlaclhemy desired content
    MetaData,
    Table, Column,
    Integer, Numeric, String,
    DateTime,
    ForeignKey,
    Select,
    create_engine)
from sqlalchemy.orm import sessionmaker # Sqlalchemy ORM desired content
from sqlalchemy.orm.exc import NoResultFound # Sqlalchemy NoResultFound

from time import sleep # sleep
```

In [36]:
```
 metadata = MetaData() # Set SQLAlchemy MetaData
```

In [37]:
```
# Obtain database password
dbpass = ''
with open('../lunacapture/.dbpass') as f:
    dbpass = f.readlines()
```

In [38]:
```
# Create the SQLAlchemy Engine
engine = create_engine('postgresql://postgres:' + dbpass[0] + '@localhost:5432
```

In [39]:
```
# Create the connection to the engine
connection = engine.connect()
```

In [40]:
```
# Create a SQLAlchemy Session
Session = sessionmaker(bind = engine)
session = Session()
```

In [41]:
```
# Build a SQLAlchemy model of the PostgreSQL database
test_conn = Table('test_conn', metadata,
                  Column('id', Integer(), primary_key=True),
                  Column('robot_json', JSON),
                  Column('created_at', DateTime(timezone=False), default=date
```

In [42]:
```
# List of possible data points to observe
data_record_names = [
    'boom',
    'drive_encoder_left',
    'drive_encoder_right',
    'dump',
    'epoch_time',
    'fork',
    'loc_angle',
```

```
    'loc_x',
    'loc_y',
    'power_boom',
    'power_dump',
    'power_fork',
    'power_left',
    'power_right',
    'power_spin',
    'power_stick',
    'power_tilt',
    'power_tool',
    'spin',
    'state_state',
    'stick',
    'tilt',
    'vibe'
]
```

In [43]:
```
sub_plot_0_name = data_record_names[10]
sub_plot_1_name = data_record_names[3]
```

In [44]:
```
print("Sub Plot 1: ", sub_plot_0_name, "\nSub Plot 2: ", sub_plot_1_name)
```

```
Sub Plot 1:  power_dump
Sub Plot 2:  dump
```

In [45]:
```
# Create an empty pandas dataframe with columns to match
df = pd.DataFrame()

df['id'] = ''
df[sub_plot_0_name] = ''
df[sub_plot_1_name] = ''
df['datetime_database'] = ''
```

In [46]:
```
def retrieve_data(show_print):
    # Prepare to select from the test_conn table
    results = None

    if df.empty:
        s = test_conn.select()
        rp = connection.execute(s)
        results = rp.fetchall()
        # To Do: Too many nests. Factor into separate functions
        if show_print:
            for result in results:
                print(result[0], ", ", result[1][sub_plot_0_name], ", ", resul

    else:
        results = session.query(test_conn).order_by(test_conn.c.id.desc()).fir

        # To Do: Search in reverse until a familiar id is found, then stop and
        # Possible code below
        # stop_at_id = df.iloc[-1]['id']
        # results = session.query(test_conn).order_by(test_conn.c.id.desc()).f

        if show_print:
            print(results[0], ", ", results[1][sub_plot_0_name], ", ", results
```

```
    if (results == None):
        raise SystemExit("The postgresql database is empty.")

    # If the dataframe is empty, add this first result
    if (len(df) == 0):
        # To Do: Lift below into its own function, so as not to repeat
        for result in results:
            df.loc[len(df)] = [result[0], result[1][sub_plot_0_name], result[1

    # To Do: Change so that all results that are not yet in dataframe are adde
    # up to the last one added

    # Otherwise, only add if the id in the result is different from the last i
    elif (results[0] != df.iloc[-1]['id']):
        df.loc[len(df)] = [results[0], results[1][sub_plot_0_name], results[1]
```

In [47]:
```
retrieve_data(True)
```

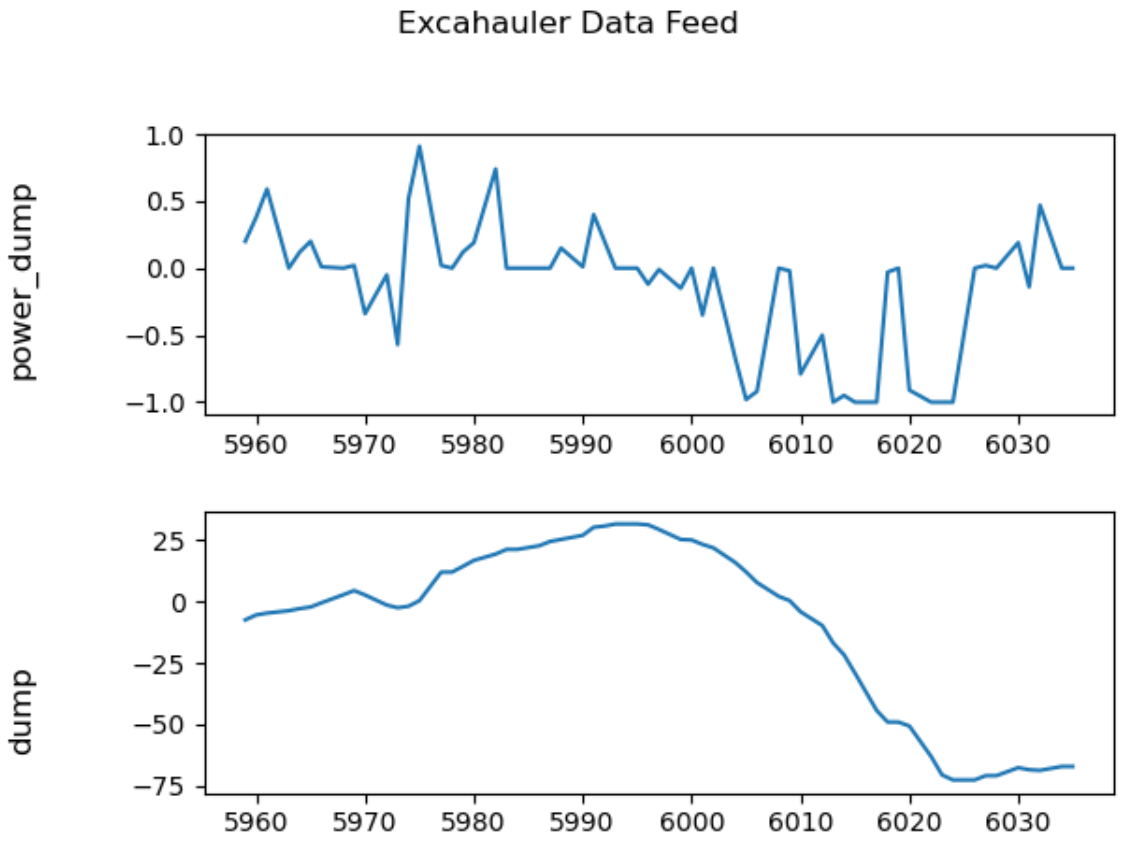
```
5850 ,  0.0 ,  0.0 ,  2023-05-15 00:31:44.604783
5851 ,  0.0 ,  0.0 ,  2023-05-15 00:31:45.109984
5852 ,  0.0 ,  0.0 ,  2023-05-15 00:31:45.615021
5853 ,  0.0 ,  0.0 ,  2023-05-15 00:31:46.119688
5854 ,  0.0 ,  0.0 ,  2023-05-15 00:31:46.623877
5855 ,  0.0 ,  0.0 ,  2023-05-15 00:31:47.129096
5856 ,  0.0 ,  0.0 ,  2023-05-15 00:31:47.634164
5818 ,  0.0 ,  0.0 ,  2023-05-15 00:31:28.485150
5819 ,  0.0 ,  0.0 ,  2023-05-15 00:31:28.989446
5820 ,  0.0 ,  0.0 ,  2023-05-15 00:31:29.493172
5821 ,  0.0 ,  0.0 ,  2023-05-15 00:31:29.996647
5822 ,  0.0 ,  0.0 ,  2023-05-15 00:31:30.500185
5823 ,  0.0 ,  0.0 ,  2023-05-15 00:31:31.003693
5824 ,  0.0 ,  0.0 ,  2023-05-15 00:31:31.507161
5825 ,  0.0 ,  0.0 ,  2023-05-15 00:31:32.010672
5826 ,  0.0 ,  0.0 ,  2023-05-15 00:31:32.514129
5827 ,  0.0 ,  0.0 ,  2023-05-15 00:31:33.017871
5828 ,  0.0 ,  0.0 ,  2023-05-15 00:31:33.521505
5829 ,  0.0 ,  0.0 ,  2023-05-15 00:31:34.024528
5830 ,  0.0 ,  0.0 ,  2023-05-15 00:31:34.528308
5831 ,  0.0 ,  0.0 ,  2023-05-15 00:31:35.031852
5832 ,  0.0 ,  0.0 ,  2023-05-15 00:31:35.535876
5833 ,  0.0 ,  0.0 ,  2023-05-15 00:31:36.039891
5834 ,  0.0 ,  0.0 ,  2023-05-15 00:31:36.545076
5835 ,  0.0 ,  0.0 ,  2023-05-15 00:31:37.048679
5836 ,  0.0 ,  0.0 ,  2023-05-15 00:31:37.552391
5837 ,  0.0 ,  0.0 ,  2023-05-15 00:31:38.055954
5838 ,  0.0 ,  0.0 ,  2023-05-15 00:31:38.559931
5839 ,  0.0 ,  0.0 ,  2023-05-15 00:31:39.063432
5840 ,  0.0 ,  0.0 ,  2023-05-15 00:31:39.566995
5841 ,  0.0 ,  0.0 ,  2023-05-15 00:31:40.070158
5842 ,  0.0 ,  0.0 ,  2023-05-15 00:31:40.573914
5843 ,  0.0 ,  0.0 ,  2023-05-15 00:31:41.077444
5844 ,  0.0 ,  0.0 ,  2023-05-15 00:31:41.580963
5845 ,  0.0 ,  0.0 ,  2023-05-15 00:31:42.084546
5846 ,  0.0 ,  0.0 ,  2023-05-15 00:31:42.589013
5847 ,  0.0 ,  0.0 ,  2023-05-15 00:31:43.092621
5848 ,  0.0 ,  0.0 ,  2023-05-15 00:31:43.595622
5849 ,  0.0 ,  0.0 ,  2023-05-15 00:31:44.100445
```

In [48]:
```
def update_graph_length(graph_initial_cell, graph_x_length):
    if (len(df) > graph_x_length):
```

```
        graph_initial_cell = len(df) - graph_x_length
    return graph_initial_cell
```

In [49]:
```
fig, axs = plt.subplots(2)
fig.suptitle("Excahauler Data Feed")
plt.ion()

fig.show()
fig.canvas.draw()
```



In [50]:
```
# Initialize the graph starting point at 0
graph_initial_cell = 0

# Set the intended graph length
graph_x_length = 60
```

In [ ]:
```
# Endless while loop to display graph
while True:
    retrieve_data(False)
    axs[0].clear()
    axs[1].clear()
    graph_initial_cell = update_graph_length(graph_initial_cell, graph_x_lengt
    axs[0].plot(df.iloc[graph_initial_cell:len(df)]['id'], df.iloc[graph_initi
    axs[1].plot(df.iloc[graph_initial_cell:len(df)]['id'], df.iloc[graph_initi

    axs[0].set_title(sub_plot_0_name, rotation=90, x=-0.2, y=0.1)
    axs[1].set_title(sub_plot_1_name, rotation=90, x=-0.2, y=0.1)
    fig.tight_layout(pad=2.0)

    fig.canvas.draw()
```

```
    sleep(500/1000)
    # clear_output(wait=True)
```

In [ ]: