data_analysis

May 9, 2021

1 database interface & plotting examples

[1]:
```
%load_ext autoreload
%autoreload 2
%matplotlib inline
import boto3
import base64
import os
from botocore.exceptions import ClientError
import json
import psycopg2
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
import pickle
import sys
import traceback
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
```

[2]:
```
class DB:
    """database interface class"""
    @staticmethod
    def connect(params: dict) -> [psycopg2.extensions.connection, psycopg2.
 ↪extensions.cursor]:
        """
            @brief: connects to the database

            @params:
                params: dictionary of db connection parameters

            @returns:
                db: the database
                cur: the cursor
        """
        if "datasource.username" in params:
            temp = {
                "user": params["datasource.username"],
                "password": params["datasource.password"],
                "database": params["datasource.database"],
                "host": params["datasource.url"],
                "port": params["datasource.port"]
            }
            params = temp
        try:
            print("[INFO] connecting to db.")
            db = psycopg2.connect(**params)
            print("[INFO] connected.")
            cur = db.cursor()
        except Exception as e:
            print("[ERROR] failed to connect to db.")
            print(e)
            return []
        return [db, cur]

    @staticmethod
    def execute(sql_query: str, database: psycopg2.extensions.connection) -> pd.
↪DataFrame:
        """
            @brief: shorthand sql style execution

            @params:
                sql_query: the query string to execute
                database: the database to execute on

            @returns: a pandas table of the query results
        """
        try:
            if('insert' in sql_query):
                print("insert here")
                pd.read_sql_query(sql_query, database)
            else:
                return pd.read_sql_query(sql_query, database)
        except Exception as e:
            print(e)
            print(traceback.print_exc())
            if ('NoneType' in str(e)):
                print("ignoring error")
            return pd.DataFrame()

    @staticmethod
    def get_tables(db: psycopg2.extensions.connection) -> pd.DataFrame:
        """Returns a DataFrame of the tables in a given database"""
        return DB.execute("""SELECT table_name FROM information_schema.tables␣
 ↪WHERE table_schema = 'public'""", db)

    @staticmethod
    def get_fields(tb: str, db: psycopg2.extensions.connection) -> pd.DataFrame:
        """Returns the fields (column headers) for a given table"""
        return DB.execute("""SELECT column_name FROM INFORMATION_SCHEMA.COLUMNS␣
 ↪WHERE table_name = '{}';""".format(tb), db)


class Utils:
    """
        @brief: static class for utility functions

        @definitions:
            get_aws_secret(secret_name, region_name)
    """

    @staticmethod
    def get_aws_secret(secret_name: str="", region_name: str="us-east-1") -> {}:
        """
            @brief: retrieves a secret stored in AWS Secrets Manager. Requires␣
 ↪AWS CLI and IAM user profile properly configured.

            @input:
                secret_name: the name of the secret
                region_name: region of use, default=us-east-1

            @output:
                secret: dictionary
        """
        client = boto3.session.Session().client(service_name='secretsmanager',␣
 ↪region_name=region_name)
        secret = '{"None": "None"}'
        if (len(secret_name) < 1):
            print("[ERROR] no secret name provided.")
        else:
            try:
                res = client.get_secret_value(SecretId=secret_name)
                if 'SecretString' in res:
                    secret = res['SecretString']
                elif 'SecretBinary' in res:
                    secret = base64.b64decode(res['SecretBinary'])
                else:
                    print("[ERROR] secret keys not found in response.")
            except ClientError as e:
                print(e)

        return json.loads(secret)

    @staticmethod
    def get_config(filename: str=r'', section: str='postgresql') -> {}:
        """
            @brief: [DEPRECIATED] parses a database configuration file

            @params:
                filename: configuraiton file with .ini extension
                section: the type of db

            @returns:
                config: dictionary of database configuration settings
        """
        from configparser import ConfigParser
        parser = ConfigParser()
        config = {}

        try:
            parser.read(filename)
        except:
            print("[ERROR] failed to read file. does it exist?")
            return config

        if parser.has_section(section):
            params = parser.items(section)
            for param in params:
                config[param[0]] = param[1]
        else:
            print('[ERROR] Section {0} not found in the {1} file'.
 ↪format(section, filename))
            return config

        return config
```

[3]:

```
params = Utils.get_aws_secret("/secret/uav_db")
db, cur =  DB.connect(params)
del(params)
DB.get_tables(db)
```

```
[INFO] connecting to db.
[INFO] connected.
```

[3]:

```
                  table_name
0                   model_tb
```

```
1                        uav_tb
2                eqc_battery_tb
3                   eq_motor_tb
4    degradation_parameter_tb
5                    mission_tb
6           pg_stat_statements
7             battery_sensor_tb
8              flight_sensor_tb
9                 experiment_tb
10               twin_params_tb
11                trajectory_tb
```

1.1 get a list of experiments & mission_ids

[71]:

```
def get_all_experiments(res='all'):
    experiments_df = DB.execute("""select et.* from experiment_tb et;""",␣
 ↪database=db)

    mission_ids = list(experiments_df['mission_ids'].values)
    mission_ids = [idx.split('-') for idx in mission_ids]
    mission_ids = [np.arange(int(x), int(y)) for x, y in mission_ids]
    mission_idx = [np.arange(1, len(x)+1) for x in mission_ids]
    assert len(mission_idx) == len(mission_ids), "[ERROR] index mappings should␣
 ↪be of same length"
    experiments = [(x, y) for x,y in zip(mission_ids, mission_idx)]
    if res is 'all':
        return experiments, mission_ids, mission_idx
    if res is 'mission':
        return mission_ids, mission_idx
    if res is 'experiments':
        return experiments
    else:
        return []
```

1.2 access the mission and degradation data for a given experiment

[791]:

```
experiment = 0 # 0 based
print(experiments_df['notes'].iloc[experiment])
mission_ids = experiments[experiment][0]
mission_idx = experiments[experiment][1]

mission_data_df = DB.execute(f"""select mt.* from mission_tb mt where mt.id >=␣
 ↪{mission_ids[0]} and mt.id <= {mission_ids[-1]} order by mt.id asc;""",␣
 ↪database=db)
mission_data_df['idx'] = mission_idx
mission_data_df = mission_data_df.drop(columns={'dt_start', 'dt_stop'})
```

```
print(len(mission_data_df))
mission_data_df.head()
```

```
first experiment with degradation curves downsampled to about 100 missions.
motor degradation was too high
51
```

```
[791]:    id  trajectory_id  stop_code  prior_rul  flight_time   distance   z_end  \
       0   1              3          3       18.0      17.8342  1301.9481  0.4780
       1   2              3          3       18.0      17.8371  1302.1040  0.4361
       2   3              3          3       18.0      17.8325  1301.9721  0.4868
       3   4             15          3       18.0      17.6346  1283.8769  0.4784
       4   5             11          3       18.0      16.3617  1218.7356  0.4833

           v_end  avg_pos_err  max_pos_err  std_pos_err  avg_ctrl_err  max_ctrl_err  \
       0  4.0083       1.2455       3.2464       0.6710        0.1012        3.1186
       1  3.9500       1.2476       3.2242       0.6687        0.0929        3.1642
       2  4.0095       1.3173       3.4164       0.6756        0.1317        3.1493
       3  4.0084       1.3431       3.7596       0.8461        0.0705        3.2933
       4  3.9880       1.3421       3.4620       0.7247        0.1588        3.5650

          std_ctrl_err  battery_id  uav_id  idx
       0        0.9820           2       1    1
       1        0.9797           2       1    2
       2        1.0203           2       1    3
       3        1.0571           2       1    4
       4        1.0172           2       1    5
```

1.2.1 view summary statistics of the mission data

```
[588]: mission_data_df.describe().transpose()
```

```
[588]:                 count         mean         std       min         25%  \
       id             131.0   696.000000   37.960506  631.0000   663.50000
       trajectory_id  131.0    11.946565    2.954301    3.0000    10.00000
       stop_code      131.0     2.816794    0.508414    1.0000     3.00000
       prior_rul      131.0    16.613964    1.263457   13.7188    15.57330
       flight_time    131.0    14.790105    2.057571    1.0396    13.93420
       distance       131.0  1093.139221  147.703723   87.3520  1047.79255
       z_end          131.0     0.517110    0.063170    0.4182     0.48805
       v_end          131.0     3.683755    0.438303    2.4711     3.48870
       avg_pos_err    131.0     1.336629    0.055145    1.2296     1.31875
       max_pos_err    131.0     3.740719    0.353017    3.1439     3.52835
       std_pos_err    131.0     0.759966    0.041454    0.6431     0.72890
       avg_ctrl_err   131.0     0.150467    0.093535    0.0649     0.09575
       max_ctrl_err   131.0     3.563545    0.282051    2.8748     3.47740
       std_ctrl_err   131.0     1.038957    0.035560    0.9071     1.01525
```

```
battery_id     131.0     2.000000    0.000000    2.0000     2.00000
uav_id         131.0     1.000000    0.000000    1.0000     1.00000
idx            131.0    66.000000   37.960506    1.0000    33.50000

                    50%          75%        max
id             696.0000   728.50000   761.0000
trajectory_id   13.0000    14.00000    20.0000
stop_code        3.0000     3.00000     3.0000
prior_rul       17.0758    17.69250    18.5685
flight_time     14.8288    16.35630    19.6329
distance      1083.3364  1218.55020  1449.1421
z_end            0.5126     0.55095     0.9627
v_end            3.9057     3.99375     4.0377
avg_pos_err      1.3399     1.36310     1.7696
max_pos_err      3.6659     3.87080     5.1263
std_pos_err      0.7604     0.77515     0.8935
avg_ctrl_err     0.1154     0.18465     0.7350
max_ctrl_err     3.6113     3.75055     4.4279
std_ctrl_err     1.0455     1.06105     1.1433
battery_id       2.0000     2.00000     2.0000
uav_id           1.0000     1.00000     1.0000
idx             66.0000    98.50000   131.0000
```

```
[180]: degradation_data_df = DB.execute(f"""select dpt.* from degradation_parameter_tb␣
        ↪dpt where dpt.mission_id >= {mission_ids[0]} and dpt.mission_id <=␣
        ↪{mission_ids[-1]} order by dpt.mission_id asc;""", database=db)
       degradation_data_df = degradation_data_df.fillna(0)
       degradation_data_df.head()
```

```
[180]:    id  mission_id      q_deg  q_var  q_slope  q_intercept     r_deg    r_var  \
       0  631         631  15.000000   0.90      0.0          0.0  0.001100  0.00100
       1  632         632  15.000000   0.90      0.0          0.0  0.001100  0.00100
       2  633         633  13.913262   0.89      0.0          0.0  0.001771  0.00099
       3  634         634  14.897995   0.88      0.0          0.0  0.000586  0.00098
       4  635         635  15.500000   0.87      0.0          0.0  0.000896  0.00097

          r_slope  r_intercept     m_deg    m_var  m_slope  m_intercept  battery_id  \
       0      0.0          0.0  0.237100  0.02000      0.0          0.0           2
       1      0.0          0.0  0.237100  0.02000      0.0          0.0           2
       2      0.0          0.0  0.251445  0.01975      0.0          0.0           2
       3      0.0          0.0  0.232806  0.01950      0.0          0.0           2
       4      0.0          0.0  0.216816  0.01925      0.0          0.0           2

          motor2_id  uav_id
       0          2       1
       1          2       1
       2          2       1
```

```
3           2       1
4           2       1
```

1.3 view summary statistics of the degradation data

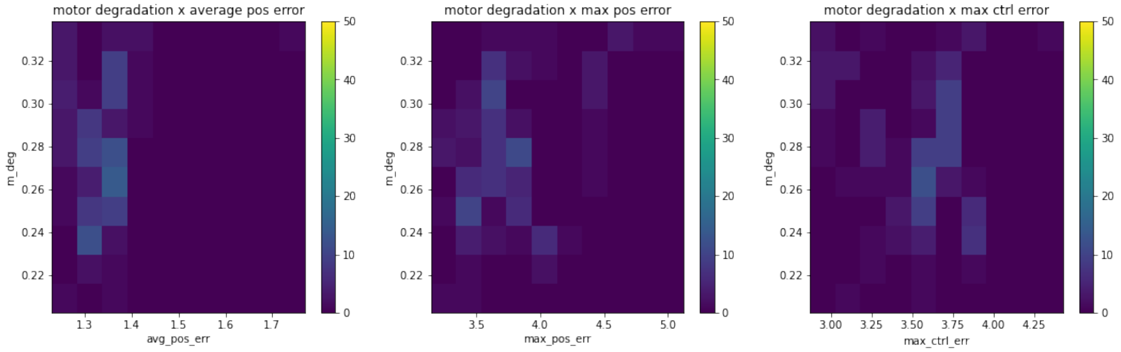
[8]:
```
degradation_data_df.describe().transpose()
```

[8]:
```
             count        mean        std         min         25%         50%  \
id           131.0  696.000000  37.960506  631.000000  663.500000  696.000000
mission_id   131.0  696.000000  37.960506  631.000000  663.500000  696.000000
q_deg        131.0   13.493197   1.257465   10.785498   12.545135   13.626380
q_var        131.0    0.417557   0.216912    0.200000    0.250000    0.260000
q_slope      131.0   -0.028664   0.051856   -0.163271   -0.060285   -0.032770
q_intercept  131.0   15.278515   4.375167    0.000000   14.321135   15.567552
r_deg        131.0    0.013608   0.017023    0.000100    0.001694    0.005009
r_var        131.0    0.000419   0.000306    0.000100    0.000100    0.000360
r_slope      131.0    0.000422   0.000517   -0.000241    0.000043    0.000167
r_intercept  131.0   -0.031403   0.045675   -0.173827   -0.057739   -0.005386
m_deg        131.0    0.276982   0.030475    0.202736    0.253572    0.275532
m_var        131.0    0.008502   0.005004    0.002500    0.005000    0.005000
m_slope      131.0    0.000757   0.001378   -0.006784    0.000243    0.000908
m_intercept  131.0    0.207710   0.069023    0.000000    0.179722    0.214890
battery_id   131.0    2.000000   0.000000    2.000000    2.000000    2.000000
motor2_id    131.0    2.000000   0.000000    2.000000    2.000000    2.000000
uav_id       131.0    1.000000   0.000000    1.000000    1.000000    1.000000

                    75%         max
id           728.500000  761.000000
mission_id   728.500000  761.000000
q_deg         14.561224   15.500000
q_var          0.585000    0.900000
q_slope       -0.003733    0.261229
q_intercept   17.371333   22.736088
r_deg          0.020550    0.064233
r_var          0.000685    0.001000
r_slope        0.000800    0.001830
r_intercept    0.000000    0.006323
m_deg          0.300006    0.338491
m_var          0.012125    0.020000
m_slope        0.001418    0.005162
m_intercept    0.251678    0.373551
battery_id     2.000000    2.000000
motor2_id      2.000000    2.000000
uav_id         1.000000    1.000000
```

1.4 some data exploration

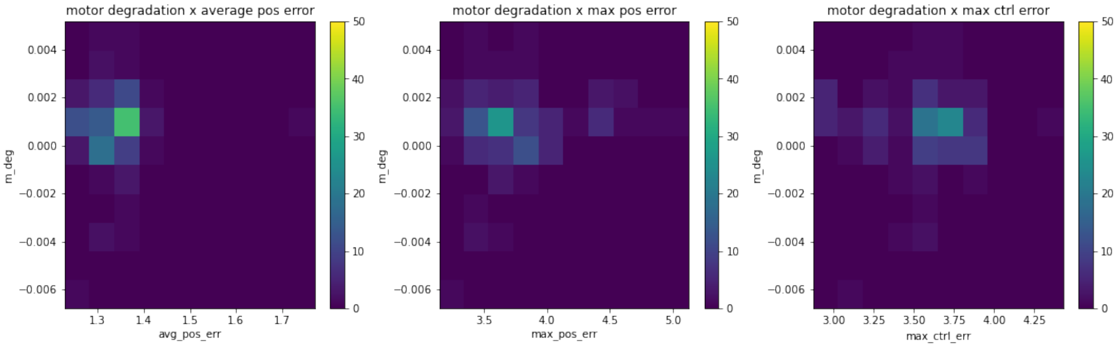
1.4.1 looking at correlations between average position error and motor degradation

[9]:
```
plt.figure(figsize=(18,5))
plt.subplot(1,3,1)
plt.hist2d(mission_data_df['avg_pos_err'], degradation_data_df['m_deg'],␣
 ↪bins=(10,10), vmax=50)
plt.colorbar()
plt.xlabel('avg_pos_err')
plt.ylabel('m_deg')
plt.title("motor degradation x average pos error")

plt.subplot(1,3,2)
plt.hist2d(mission_data_df['max_pos_err'], degradation_data_df['m_deg'],␣
 ↪bins=(10,10), vmax=50)
plt.colorbar()
plt.xlabel('max_pos_err')
plt.ylabel('m_deg')
plt.title("motor degradation x max pos error")

plt.subplot(1,3,3)
plt.hist2d(mission_data_df['max_ctrl_err'], degradation_data_df['m_deg'],␣
 ↪bins=(10,10), vmax=50)
plt.colorbar()
plt.xlabel('max_ctrl_err')
plt.ylabel('m_deg')
plt.title("motor degradation x max ctrl error")

plt.show()
```



1.4.2 what about correlations with the degradation rate of change?

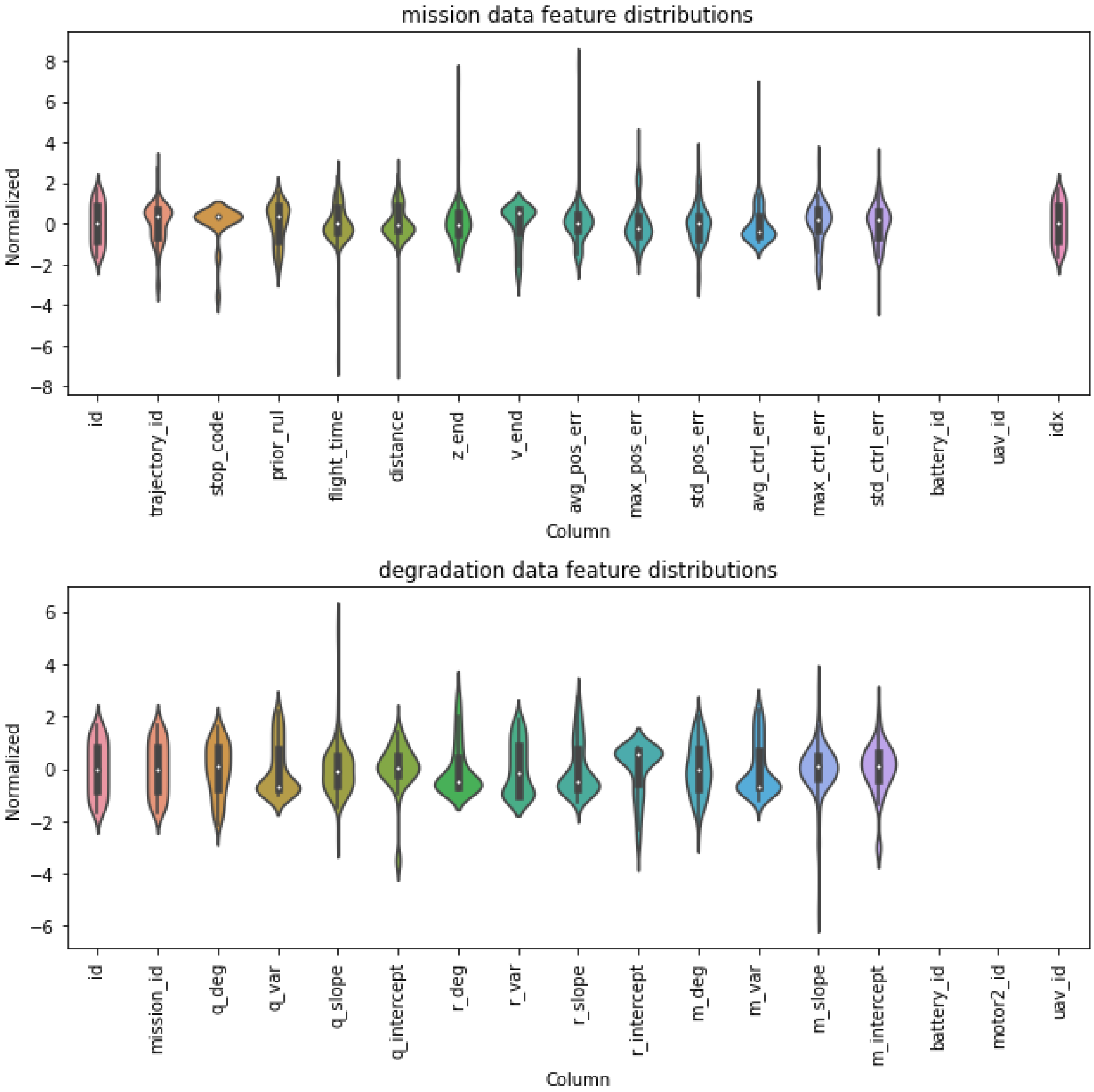
[10]:
```
assert degradation_data_df['m_slope'].isnull().values.any() == False, "[WARN]␣
 ↪fillna values on the m_slope column"
plt.figure(figsize=(18,5))
plt.subplot(1,3,1)
plt.hist2d(mission_data_df['avg_pos_err'], degradation_data_df['m_slope'],␣
 ↪bins=(10,10), vmax=50)
plt.colorbar()
plt.xlabel('avg_pos_err')
plt.ylabel('m_deg')
plt.title("motor degradation x average pos error")

plt.subplot(1,3,2)
plt.hist2d(mission_data_df['max_pos_err'], degradation_data_df['m_slope'],␣
 ↪bins=(10,10), vmax=50)
plt.colorbar()
plt.xlabel('max_pos_err')
plt.ylabel('m_deg')
plt.title("motor degradation x max pos error")

plt.subplot(1,3,3)
plt.hist2d(mission_data_df['max_ctrl_err'], degradation_data_df['m_slope'],␣
 ↪bins=(10,10), vmax=50)
plt.colorbar()
plt.xlabel('max_ctrl_err')
plt.ylabel('m_deg')
plt.title("motor degradation x max ctrl error")

plt.show()
```



2 view feature distributions

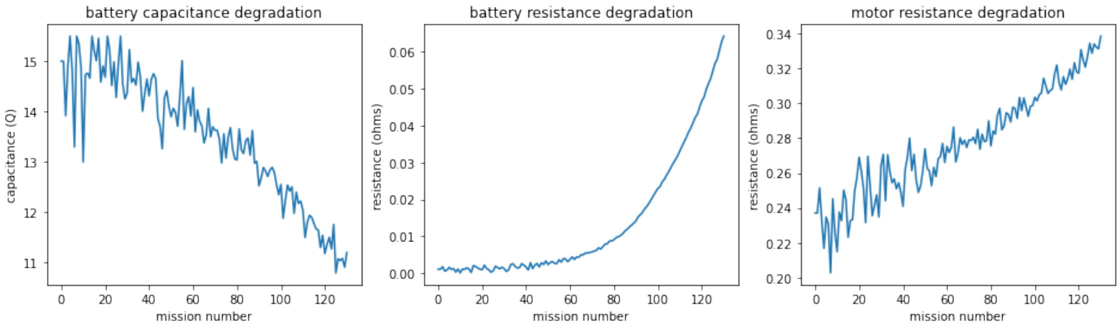
[11]:

```
degradation_data_normalized = (degradation_data_df - degradation_data_df.
 ↪mean()) / degradation_data_df.std()
mission_data_normalized = (mission_data_df - mission_data_df.mean()) /␣
 ↪mission_data_df.std()

plt.figure(figsize=(9, 9))
plt.subplot(2,1,1)
mission_plt = mission_data_normalized.melt(var_name='Column',␣
 ↪value_name='Normalized')
ax = sns.violinplot(x='Column', y='Normalized', data=mission_plt)
_ = ax.set_xticklabels(mission_data_df.keys(), rotation=90)
plt.title("mission data feature distributions")

plt.subplot(2,1,2)
degradation_plt = degradation_data_normalized.melt(var_name='Column',␣
 ↪value_name='Normalized')
ax = sns.violinplot(x='Column', y='Normalized', data=degradation_plt)
_ = ax.set_xticklabels(degradation_data_df.keys(), rotation=90)
plt.title("degradation data feature distributions")
plt.tight_layout()
```



3 view degradation parameter plots for a single experiment

[12]:
```
plt.figure(figsize=(16,4))
plt.subplot(1,3,1)
degradation_data_df['q_deg'].plot()
plt.title('battery capacitance degradation')
plt.xlabel('mission number')
plt.ylabel('capacitance (Q)')

plt.subplot(1,3,2)
degradation_data_df['r_deg'].plot()
plt.title('battery resistance degradation')
plt.xlabel('mission number')
plt.ylabel('resistance (ohms)')

plt.subplot(1,3,3)
degradation_data_df['m_deg'].plot()
plt.title('motor resistance degradation')
plt.xlabel('mission number')
plt.ylabel('resistance (ohms)')

plt.show()
```



4 view RUL (flight time) estimation plots for all experiments

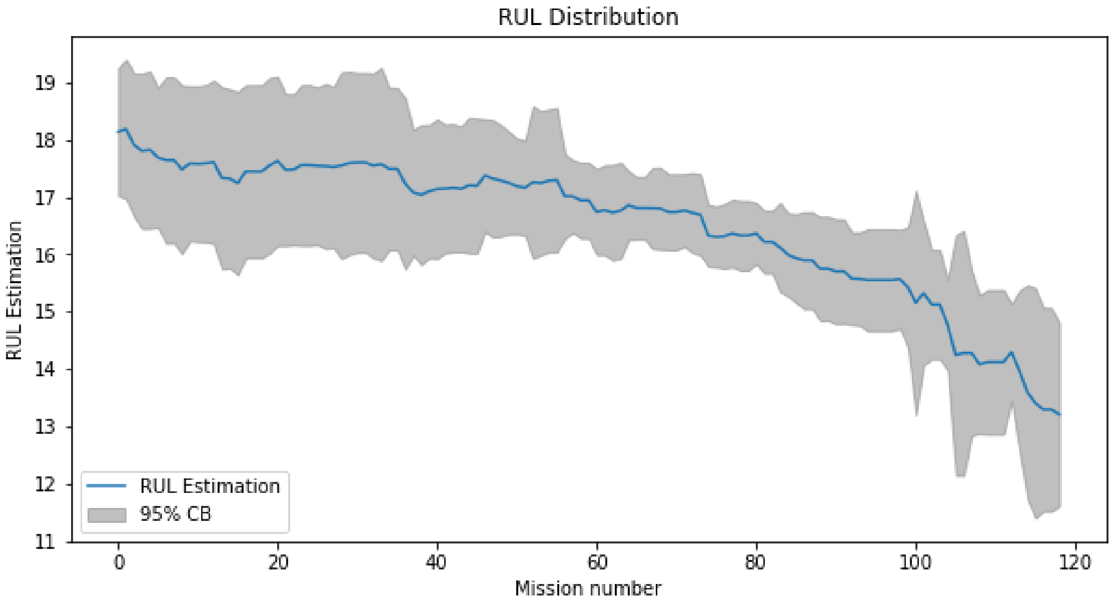
[8]:

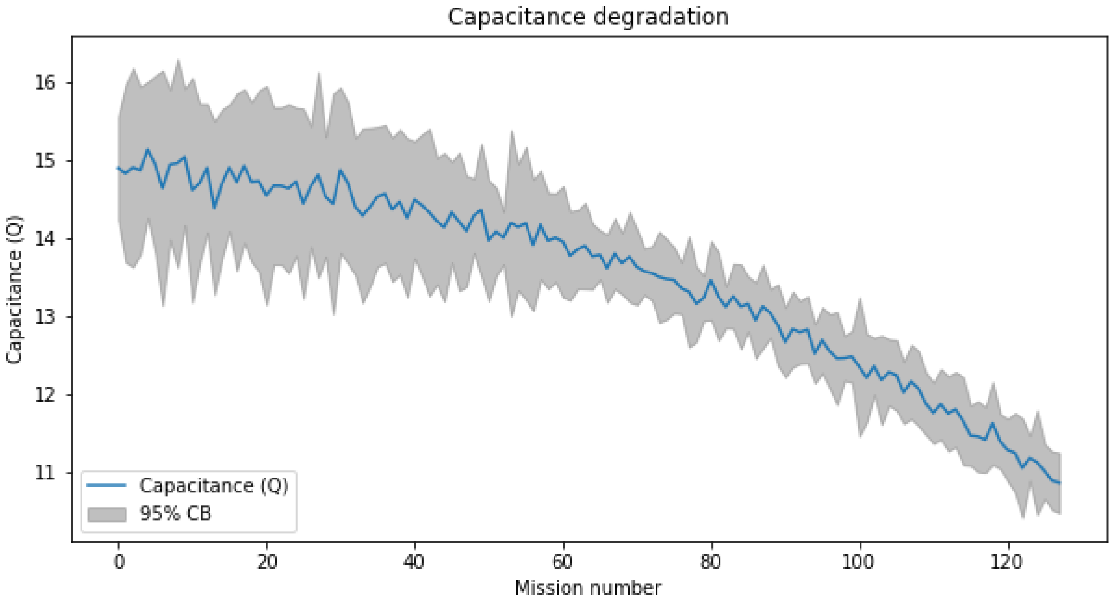
```
# get the rul from each experiment
ruls = []
q_degs = []
r_degs = []
m_degs = []
errs = []

def get_samples(vals, exclude=[0,2]):
    samples = []
    for i in range(0, max(len(val) for val in vals)):
        temp = []
        for j in range(0, len(vals)):
            if j in exclude:
                continue
            if(i < len(vals[j])):
                temp.append(vals[j][i])
        samples.append(temp)
    return samples


def plot_distribution(samples=[],
                      return_distribution=True,
                      label="RUL Estimation",
                      title="RUL Distribution",):
    mus = []
    stds = []

    # samples is a multi-dimensional list, where each index represents a run number
    # which always starts at 0 and increments until end of life, and at each index is
    # a list of rul estimations from all experiments for that mission number
    # for example, there are 9 rul estimates at run number = 19, which contain
    # [16.5433, 18.9283, 17.7767, 18.2302, 17.0758, 16.7842, 17.5781, 17.4173, 17.0094]
    count = tf.Variable(0)
    for sample in samples:
        s = tf.convert_to_tensor(sample, dtype=tf.float32)
        count = count + len(s)
        mu = tf.math.reduce_mean(s, axis=0)
        std = tf.math.reduce_std(s, axis=0)
        #print(mu, std)
        mus.append(mu)
        stds.append(std)
    mu_t = tf.convert_to_tensor(mus, dtype=tf.float32)
    std_t = tf.convert_to_tensor(stds, dtype=tf.float32)


    plt.figure(figsize=(10,5))

    x = tf.range(0, mu_t.shape[0], delta=1)
    plt.fill_between(x,
                     mu_t-2*std_t,
                     mu_t+2*std_t,
                     color='grey',
                     alpha=.5, label="95% CB")
    plt.plot(x, mu_t, label=label)
    plt.ylabel(label)
    plt.xlabel('Mission number')
    plt.title(title)
    plt.legend(loc=3)
    #plt.text(-4, 11.8, f"*Calculated from data on {count} missions", backgroundcolor='white')
    plt.show()

    if return_distribution:
        return [mu_t, std_t]
```

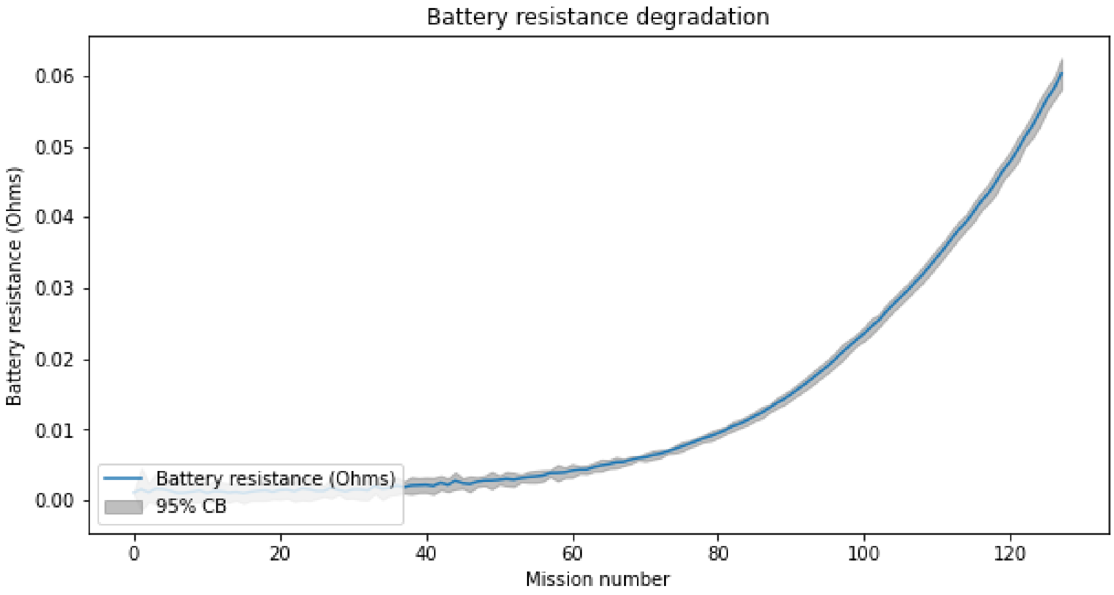
```
for i in range(0, len(experiments)):
    print(i, experiments_df['notes'].iloc[i])
    mission_ids = experiments[i][0]
    mission_idx = experiments[i][1]

    mission_data_df = DB.execute(f"""select mt.* from mission_tb mt where mt.id␣
 ↪>= {mission_ids[0]} and mt.id <= {mission_ids[-1]} order by mt.id asc;""",␣
 ↪database=db)
    mission_data_df['idx'] = mission_idx
    mission_data_df = mission_data_df.drop(columns={'dt_start', 'dt_stop'})

    degradation_data_df = DB.execute(f"""select dpt.* from␣
 ↪degradation_parameter_tb dpt where dpt.mission_id >= {mission_ids[0]} and␣
 ↪dpt.mission_id <= {mission_ids[-1]} order by dpt.mission_id asc;""",␣
 ↪database=db)
    degradation_data_df = degradation_data_df.fillna(0)

    ruls.append(mission_data_df['prior_rul'].values)
    errs.append(mission_data_df['avg_pos_err'].values)
    q_degs.append(degradation_data_df['q_deg'].values)
    r_degs.append(degradation_data_df['r_deg'].values)
    m_degs.append(degradation_data_df['m_deg'].values)

exclude=[0,2]
1 in exclude



samples=get_samples(ruls, exclude=[0,2])
rul_mu, rul_std = plot_distribution(samples=samples[9:-4],␣
 ↪return_distribution=True)

samples=get_samples(q_degs, exclude=[0,2])
qd_mu, qd_std = plot_distribution(samples=samples[:-4],
                                 return_distribution=True,
                                 label="Capacitance (Q)",
                                 title="Capacitance degradation")

samples=get_samples(r_degs, exclude=[0,2])
rd_mu, rd_std = plot_distribution(samples=samples[:-4],
                                 return_distribution=True,
                                 label="Battery resistance (Ohms)",
                                 title="Battery resistance degradation")

samples=get_samples(m_degs, exclude=[0,2])
```

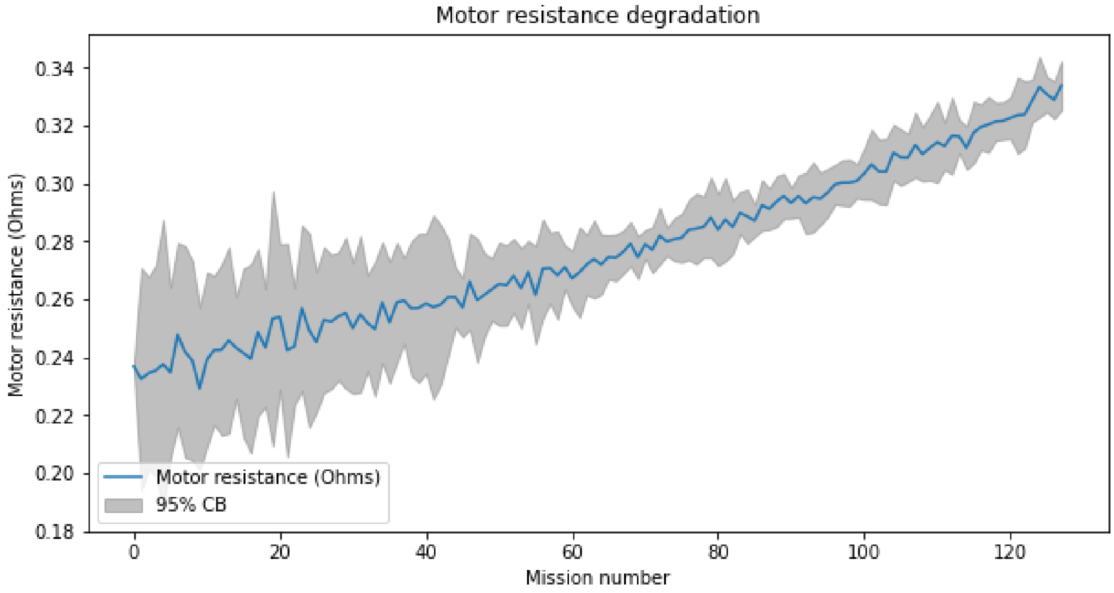
```
md_mu, md_std = plot_distribution(samples=samples[:-4],
                                  return_distribution=True,
                                  label="Motor resistance (Ohms)",
                                  title="Motor resistance degradation")

samples=get_samples(errs, exclude=[0,2])
ed_mu, ed_std = plot_distribution(samples=samples[:-4],
                                  return_distribution=True,
                                  label="avg pos err (m)",
                                  title="avg position error")
```

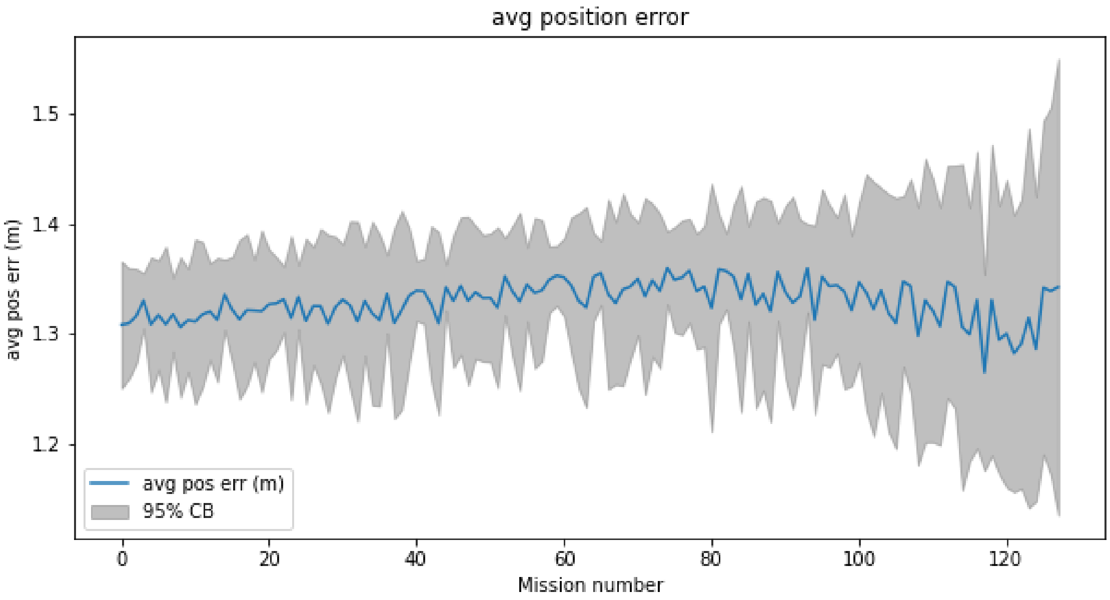
```
0 first experiment with degradation curves downsampled to about 100 missions.
motor degradation was too high
1 second experiment with motor degradation back to original (500 cycles) and
battery degradation at half (180 cycles)
2 third experiment exact repeat of second experiment
3 4th experiment, allowed for better rul updates - still seeing true system
failures before digital twin failures
4 failed to write flight data for mission 526, error during simulation, matlab
crashed and the simulation restarted form scratch with a new experiment
5 now simulating digital twin 4x and using mean values, includes random
trajectory exploration of path > rul time, stopped before experiment finished
6 simulating digital twin 4x, random trajectory exploration, digital twin does
not inform true system, mission 742 (and others), why did true system fail when
it had worse degradradation parameters than the digital twin? are there
trajectories with higher crash rates? (trajectory 10)
7 same as above, digital twin informs true system, but in some cases the true
system still did exploration - computer restarted in the middle of the
experiment
8 same as above, digital twin informs true system, true system doesnt explore
9 same as above, digital twin informs true system, true system doesnt explore,
but true system failed several times in the end?
10 same as above, there are still some true system failures
11 same as above, there are still some true system failures
12 same as above, there are still some true system failures
13 same as above, decreased initial variance some, increased exploration rate
```


RUL Distribution
RUL Estimation
19
18
17
16
15
14
13
12
11
0
20
40
60
80
100
120
Mission number
RUL Estimation
▨ 95% CB


Capacitance degradation
Capacitance (Q)
16
15
14
13
12
11
0
20
40
60
80
100
120
Mission number
Capacitance (Q)
▨ 95% CB


Battery resistance degradation
Battery resistance (Ohms)
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0
20
40
60
80
100
120
Mission number
Battery resistance (Ohms)
95% CB


Motor resistance degradation
Motor resistance (Ohms)
0.34
0.32
0.30
0.28
0.26
0.24
0.22
0.20
0.18
0
20
40
60
80
100
120
Mission number
Motor resistance (Ohms)
95% CB

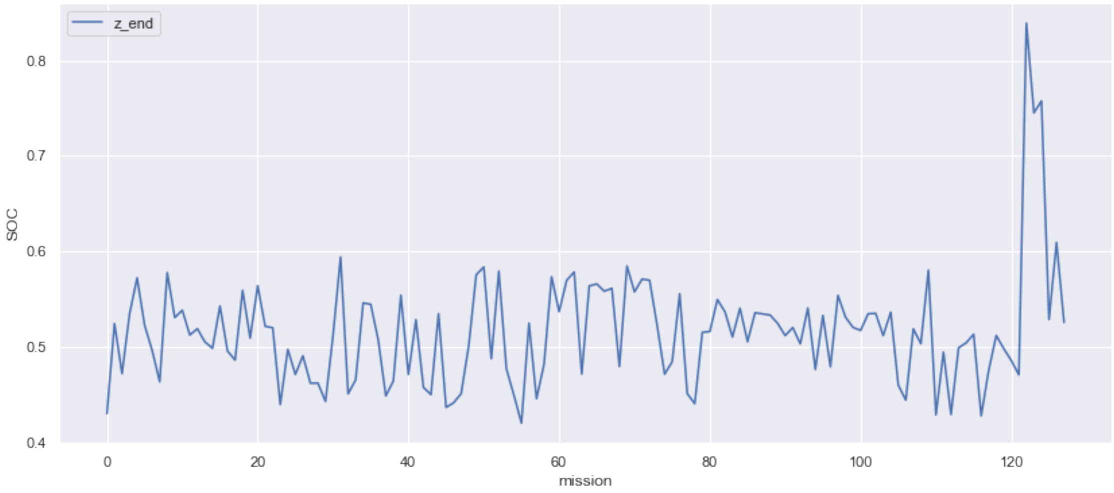


[794]:
```
mission_data_df.head()
mission_data_df['index'] = mission_data_df.index
mission_data_df.head()
```

[794]:
```
   id  trajectory_id  stop_code  prior_rul  flight_time   distance   z_end  \
0   1              3          3       18.0      17.8342  1301.9481  0.4780
1   2              3          3       18.0      17.8371  1302.1040  0.4361
2   3              3          3       18.0      17.8325  1301.9721  0.4868
3   4             15          3       18.0      17.6346  1283.8769  0.4784
4   5             11          3       18.0      16.3617  1218.7356  0.4833

    v_end  avg_pos_err  max_pos_err  std_pos_err  avg_ctrl_err  max_ctrl_err  \
0  4.0083       1.2455       3.2464       0.6710        0.1012        3.1186
1  3.9500       1.2476       3.2242       0.6687        0.0929        3.1642
2  4.0095       1.3173       3.4164       0.6756        0.1317        3.1493
3  4.0084       1.3431       3.7596       0.8461        0.0705        3.2933
4  3.9880       1.3421       3.4620       0.7247        0.1588        3.5650

   std_ctrl_err  battery_id  uav_id  idx  index
0        0.9820           2       1    1      0
1        0.9797           2       1    2      1
2        1.0203           2       1    3      2
3        1.0571           2       1    4      3
4        1.0172           2       1    5      4
```

[820]:
```
#mission_data_df['z_end'] = mission_data_df['z_end']/200
plt.figure(figsize=(14,6))
```

```python
plt.plot(mission_data_df['z_end'], label='z_end')
#plt.scatter(x=mission_data_df.index, y=mission_data_df['trajectory_id'])
axe = plt.gca()
plt.legend()
plt.xlabel('mission')
plt.ylabel('SOC')
plt.show()
```



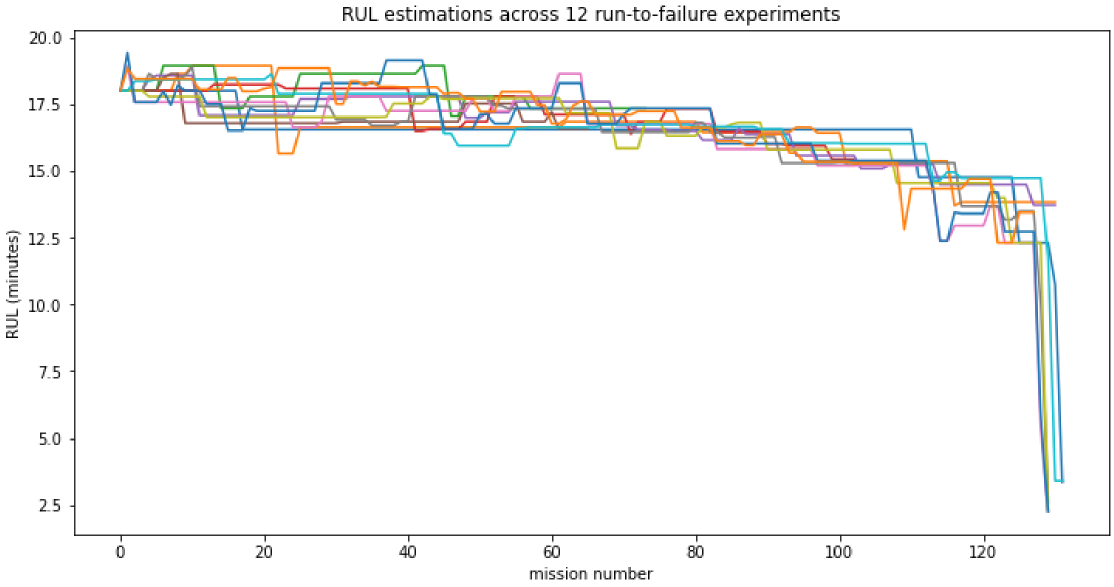
[808]:

```python
mission_data_df['trajectory_id'].unique().min()
```

[808]: 3

[9]:

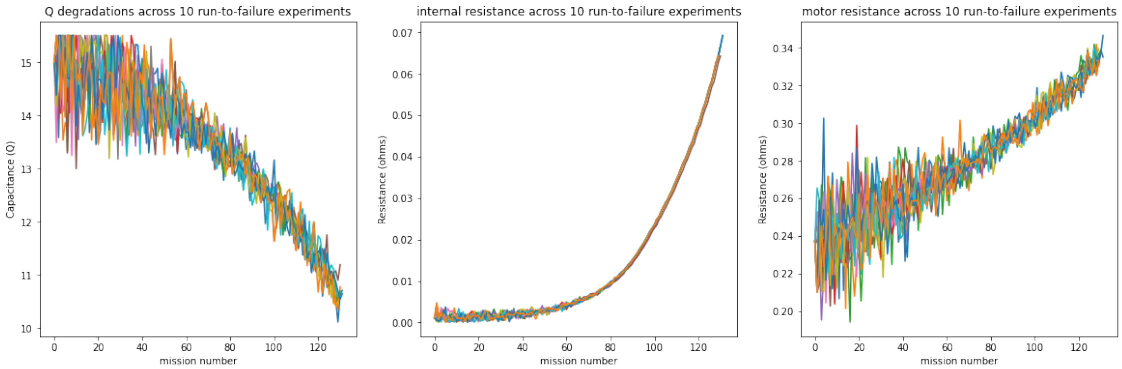
```python
plt.figure(figsize=(12,6))
for i in range(0, len(ruls)):
    if i == 0 or i == 2:
        continue
    plt.plot(ruls[i])


plt.title(f"RUL estimations across {len(ruls)-2} run-to-failure experiments")
plt.ylabel("RUL (minutes)")
plt.xlabel("mission number")
plt.show()
```



5 view degradation parameter plots for all runs

[19]:

```
plt.figure(figsize=(20,6))
plt.subplot(1,3,1)
i = 0
for q in q_degs:
    if i == 0:
        i = 1
        continue
    plt.plot(q)

plt.title("Q degradations across 10 run-to-failure experiments")
plt.ylabel("Capacitance (Q)")
plt.xlabel("mission number")

plt.subplot(1,3,2)
i = 0
for r in r_degs:
    if i == 0:
        i = 1
        continue
    plt.plot(r)

plt.title("internal resistance across 10 run-to-failure experiments")
plt.ylabel("Resistance (ohms)")
plt.xlabel("mission number")
plt.subplot(1,3,3)
i = 0
for m in m_degs:
    if i == 0:
        i = 1
        continue
    plt.plot(m)

plt.title("motor resistance across 10 run-to-failure experiments")
plt.ylabel("Resistance (ohms)")
plt.xlabel("mission number")

plt.show()
```



6 look at the twin parameter data

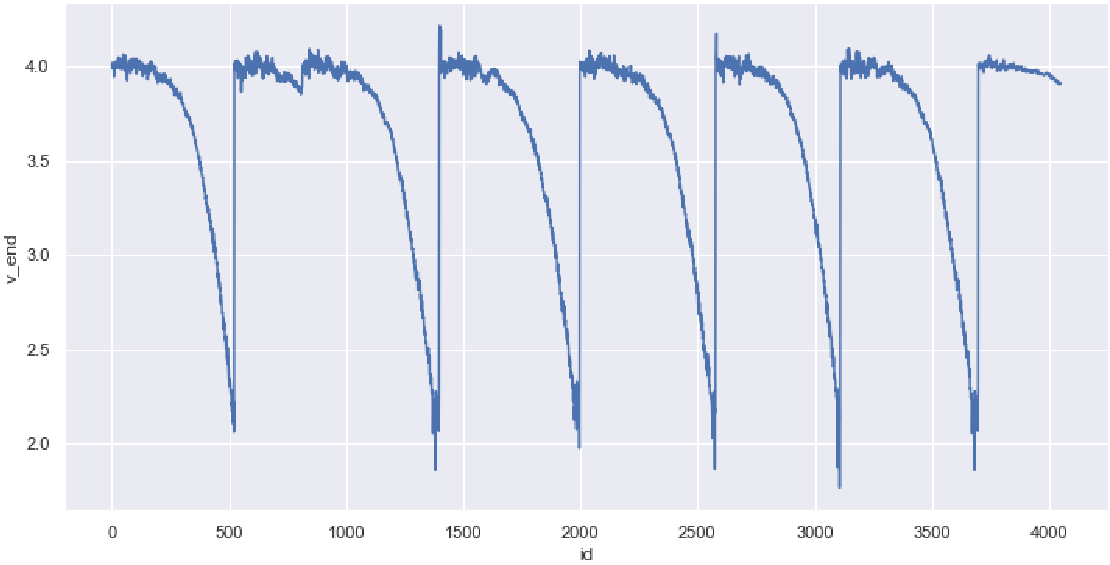
[20]:
```
twin_params_df = DB.execute("""select tpt.* from twin_params_tb tpt;""",␣
 ↪database=db)
twin_params_df.tail()
```

[20]:
```
        id  mission_id  trajectory_id    rul_hat  flight_time     distance  \
4043  4044        1551             20  17.461667    17.426667  1320.547308
4044  4045        1551             14  16.769167    14.834167  1099.691784
4045  4046        1551             14  16.769167    14.833333  1099.451839
4046  4047        1551             14  16.769167    14.833333  1099.399413
4047  4048        1551             14  16.769167    14.833333  1099.838904

         v_end     z_end   avg_err     q_deg      r_deg     m_deg  stop1  \
4043  3.904420  0.405843  1.453637  0.004846  12.859870  0.276714      1
4044  3.904003  0.489788  1.499260  0.004846  12.859870  0.276714      0
4045  3.910385  0.505285  1.485872  0.004844  13.263996  0.280576      0
```

```
4046  3.909785  0.499963  1.503915  0.004841  13.120078  0.274096      0
4047  3.914312  0.510384  1.511213  0.004714  13.409883  0.272416      0

      stop2  stop3  uav_id
4043      0      0       1
4044      0      1       1
4045      0      1       1
4046      0      1       1
4047      0      1       1
```

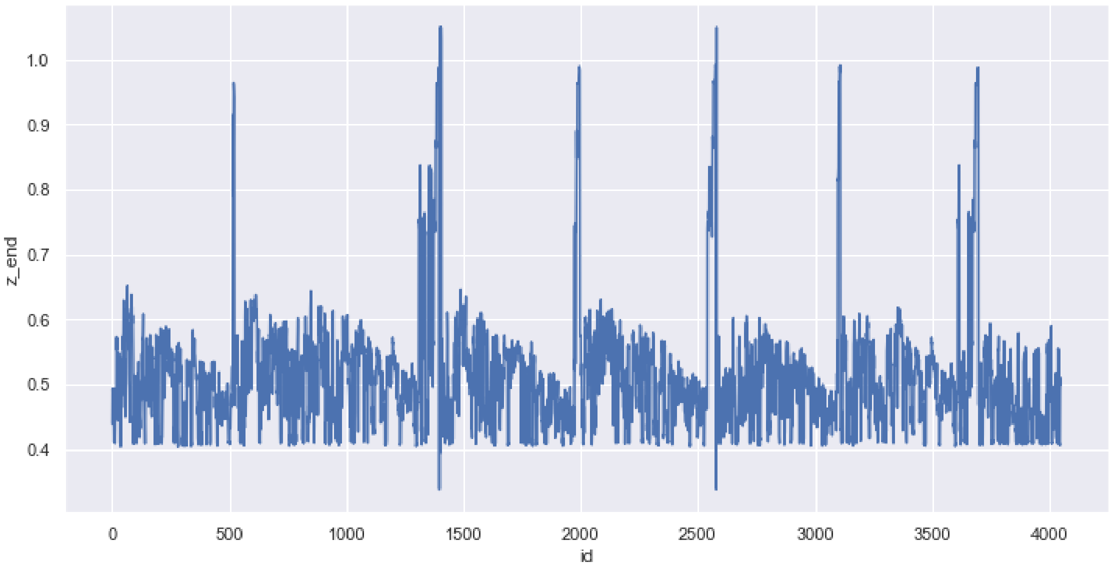
6.1 view voltage ending

[21]:
```
sns.set_theme(style='darkgrid')
plt.figure(figsize=(12,6))
sns.lineplot(data=twin_params_df, x="id", y="v_end")
plt.show()
```



7 view ending charge

[22]:
```
sns.set_theme(style='darkgrid')
plt.figure(figsize=(12,6))
sns.lineplot(data=twin_params_df, x="id", y="z_end")
plt.show()
```



8 view degradation slopes

[23]:
```
degradation_data_df.head()
```

[23]:
```
     id  mission_id      q_deg  q_var  q_slope  q_intercept     r_deg  \
0  1360        1360  15.000000   0.90      0.0          0.0  0.001100
1  1361        1361  15.500000   0.89      0.0          0.0  0.004678
2  1362        1362  15.250537   0.88      0.0          0.0  0.000808
3  1363        1363  14.300126   0.87      0.0          0.0  0.002619
4  1364        1364  15.500000   0.86      0.0          0.0  0.000940

     r_var  r_slope  r_intercept     m_deg    m_var  m_slope  m_intercept  \
0  0.00100      0.0          0.0  0.237100  0.02000      0.0          0.0
1  0.00099      0.0          0.0  0.210102  0.01975      0.0          0.0
2  0.00098      0.0          0.0  0.222026  0.01950      0.0          0.0
3  0.00097      0.0          0.0  0.245294  0.01925      0.0          0.0
4  0.00096      0.0          0.0  0.223809  0.01900      0.0          0.0

   battery_id  motor2_id  uav_id
0           2          2       1
1           2          2       1
2           2          2       1
3           2          2       1
4           2          2       1
```

[14]:
```
%matplotlib inline
from IPython.display import HTML
```

```
import matplotlib.animation
import numpy as np

from scipy.signal import medfilt


def animate_degradation(deg, slope, intercept):
    # First set up the figure, the axis, and the plot element we want to animate
    fig, ax = plt.subplots()

    line, = ax.plot([], [], lw=2)
    plt.plot(deg)

    # initialization function: plot the background of each frame
    def init():
        line.set_data([], [])
        return (line,)

    # animation function. This is called sequentially
    def animate(i):
        x = tf.range(start=i-2, limit=i+2, delta=.25)
        qm = slope[i]
        qb = intercept[i]
#          qm = degradation_data_df['q_slope'].iloc[i]
#          qb = degradation_data_df['q_intercept'].iloc[i]
        qy = qm * x + qb
        line.set_data(x, qy)
        return (line,)

    # call the animator. blit=True means only re-draw the parts that have␣
 ↪changed.
    anim = matplotlib.animation.FuncAnimation(fig, animate, init_func=init,␣
 ↪frames=len(slope), interval=50, blit=True)

    return HTML(anim.to_html5_video())
```

9 single run to failure experiment battery capacitance degradation

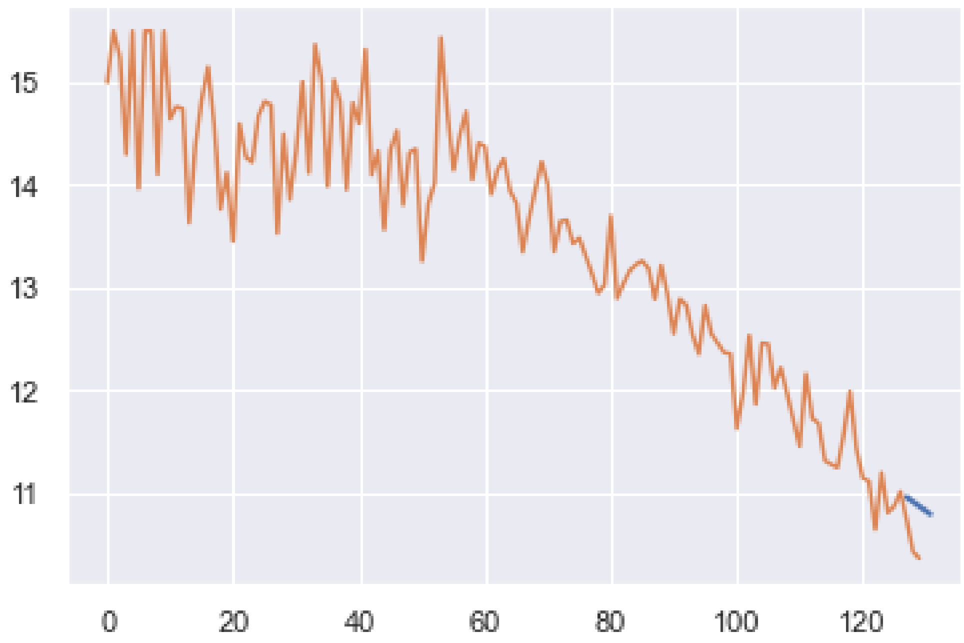
- 128 missions in this case

[182]:

```
slope_filt = medfilt(degradation_data_df['q_slope'].values, 9)
inter_filt = medfilt(degradation_data_df['q_intercept'].values, 9)
animate_degradation(degradation_data_df['q_deg'], slope_filt, inter_filt)
```

[182]:

```
<IPython.core.display.HTML object>
```



10 single run to failure experiment battery resistance degradation

- 128 missions in this case

[28]:
```
slope_filt = medfilt(degradation_data_df['r_slope'].values, 9)
inter_filt = medfilt(degradation_data_df['r_intercept'].values, 9)
animate_degradation(degradation_data_df['r_deg'], slope_filt, inter_filt)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[28]:
```
<IPython.core.display.HTML object>
```

11 single run to failure experiment motor degradation

[29]:
```
slope_filt = medfilt(degradation_data_df['m_slope'].values, 9)
inter_filt = medfilt(degradation_data_df['m_intercept'].values, 9)
animate_degradation(degradation_data_df['m_deg'], slope_filt, inter_filt)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

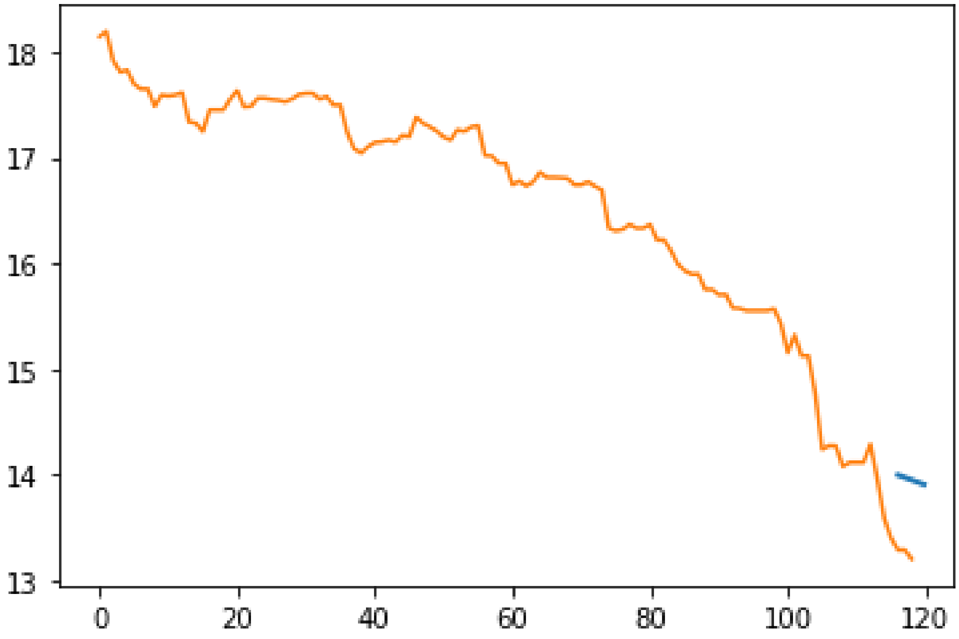
[29]:
```
<IPython.core.display.HTML object>
```

12 rul estimate of 14 experiments (1620 missions)

12.0.1 view slope

[13]:
```
lookback = 8
horizon = 4
slopes = []
ints = []
for i in range(0, len(rul_mu)):
    if i <= lookback:
        slopes.append(0)
        ints.append(0)
    else:
        x = tf.range(start=i-lookback, limit=i, delta=1)
        y = rul_mu[x[0].numpy():x[-1].numpy()+1]
        z = np.polyfit(x.numpy(), y.numpy(), 1)
        slopes.append(z[0])
        ints.append(z[1])

slope_filt = medfilt(np.array(slopes), 9)
inter_filt = medfilt(np.array(ints), 9)
animate_degradation(rul_mu, slope_filt, inter_filt)
```

[13]:
```
<IPython.core.display.HTML object>
```



[ ]:
```
```

13 clustering analysis

[104]:

```
import scipy.cluster.hierarchy as hc
from sklearn.cluster import AgglomerativeClustering

stop1 = twin_params_df.pop('stop1')
stop2 = twin_params_df.pop('stop2')
stop3 = twin_params_df.pop('stop3')
```

[110]:

```
degradation_data_df = DB.execute(f"""select dpt.* from degradation_parameter_tb␣
 ↪dpt order by dpt.mission_id asc;""", database=db)
degradation_data_df = degradation_data_df.fillna(0)
print(len(degradation_data_df))
degradation_data_df.head()
```

```
1553
```

[110]:

```
   id  mission_id      q_deg  q_var  q_slope  q_intercept     r_deg    r_var  \
0   1           1  15.000000   0.90      0.0          0.0  0.001100  0.00100
1   2           2  13.877173   0.89      0.0          0.0  0.003257  0.00099
2   3           3  15.500000   0.88      0.0          0.0  0.001059  0.00098
3   4           4  14.934967   0.87      0.0          0.0  0.001094  0.00097
4   5           5  13.927817   0.86      0.0          0.0  0.001899  0.00096

   r_slope  r_intercept     m_deg    m_var  m_slope  m_intercept  battery_id  \
0      0.0          0.0  0.237100  0.02500      0.0          0.0           2
1      0.0          0.0  0.250991  0.02475      0.0          0.0           2
2      0.0          0.0  0.171652  0.02450      0.0          0.0           2
3      0.0          0.0  0.272745  0.02425      0.0          0.0           2
4      0.0          0.0  0.264777  0.02400      0.0          0.0           2

   motor2_id  uav_id
0          2       1
1          2       1
2          2       1
3          2       1
4          2       1
```

14 min max scale to range [0,1]

[183]:

```
twin_params_df = DB.execute("""select tpt.* from twin_params_tb tpt;""",␣
 ↪database=db)
degradation_data_df = DB.execute(f"""select dpt.* from degradation_parameter_tb␣
 ↪dpt where dpt.mission_id >= {mission_ids[0]} and dpt.mission_id <=␣
 ↪{mission_ids[-1]} order by dpt.mission_id asc;""", database=db)
degradation_data_df = degradation_data_df.fillna(0)
```

```
twin_params_df = (twin_params_df - twin_params_df.min()) / (twin_params_df.
 ↪max() - twin_params_df.min())
degradation_data_df = (degradation_data_df - degradation_data_df.min()) /␣
 ↪(degradation_data_df.max() - degradation_data_df.min())
X1 = twin_params_df[['v_end', 'z_end', 'avg_err']]
X2 = degradation_data_df[['q_deg', 'q_slope', 'q_intercept', 'r_deg',␣
 ↪'r_slope', 'r_intercept', 'm_deg', 'm_slope', 'm_intercept']]
#X2 = X2[X2['q_slope'] > 0]
Xs = [X1.values, X2.values]
twin_params_df = DB.execute("""select tpt.* from twin_params_tb tpt;""",␣
 ↪database=db)
degradation_data_df = DB.execute(f"""select dpt.* from degradation_parameter_tb␣
 ↪dpt where dpt.mission_id >= {mission_ids[0]} and dpt.mission_id <=␣
 ↪{mission_ids[-1]} order by dpt.mission_id asc;""", database=db)
degradation_data_df = degradation_data_df.fillna(0)
```

[ ]:

```

```

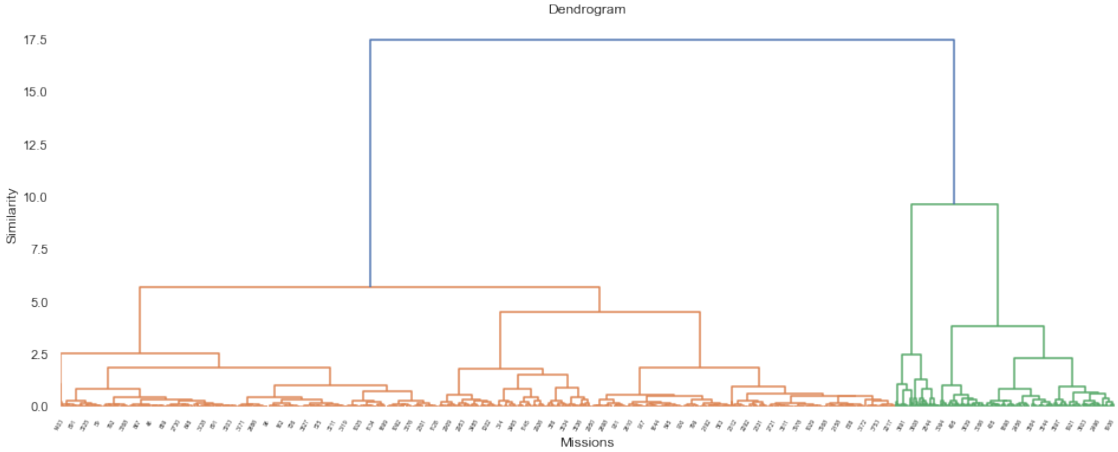
[184]:

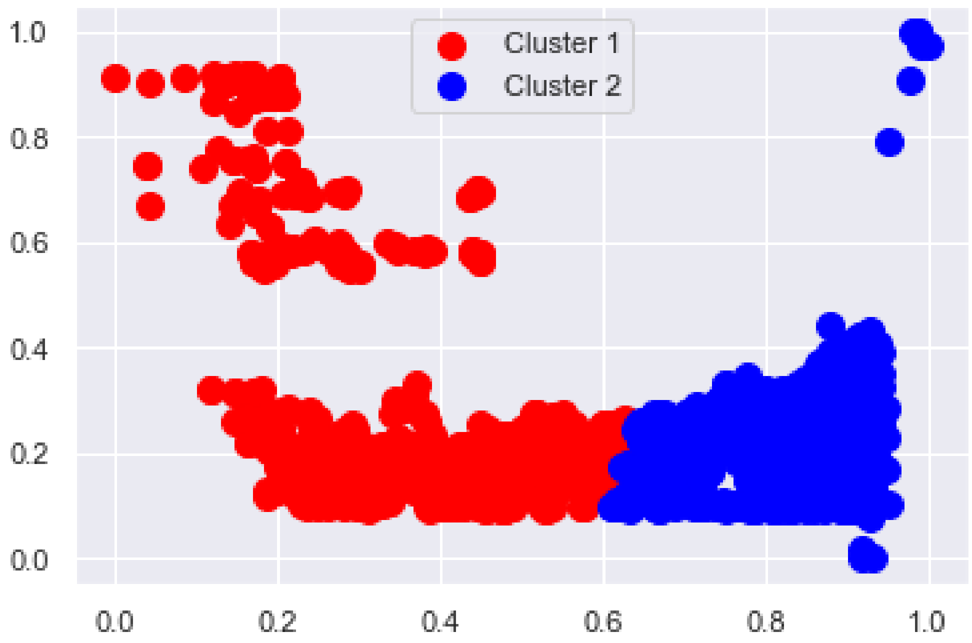
```
%matplotlib inline
names = ["twin params", "degradation data"]
j = 0

keep_clusters = []

for X in Xs:

    print(f"******************** {names[j]} ******************")
    j = j + 1
    plt.figure(figsize=(16,6))
    dendrogram = hc.dendrogram(hc.linkage(X, method='ward'))
    plt.title('Dendrogram')
    plt.xlabel('Missions')
    plt.ylabel("Similarity")
    ax = plt.gca()
    [l.set_visible(False) for (i,l) in enumerate(ax.xaxis.get_ticklabels()) if␣
 ↪i % 50 != 0]
    plt.xticks(rotation=60)
    plt.show()

    clust2 = AgglomerativeClustering(n_clusters=2, affinity='euclidean',␣
 ↪linkage='ward')
    y_pred = clust2.fit_predict(X)
    plt.scatter(X[y_pred == 0, 0], X[y_pred == 0, 1], s = 100, c = 'red', label␣
 ↪= 'Cluster 1')
    plt.scatter(X[y_pred == 1, 0], X[y_pred == 1, 1], s = 100, c = 'blue',␣
 ↪label = 'Cluster 2')
```

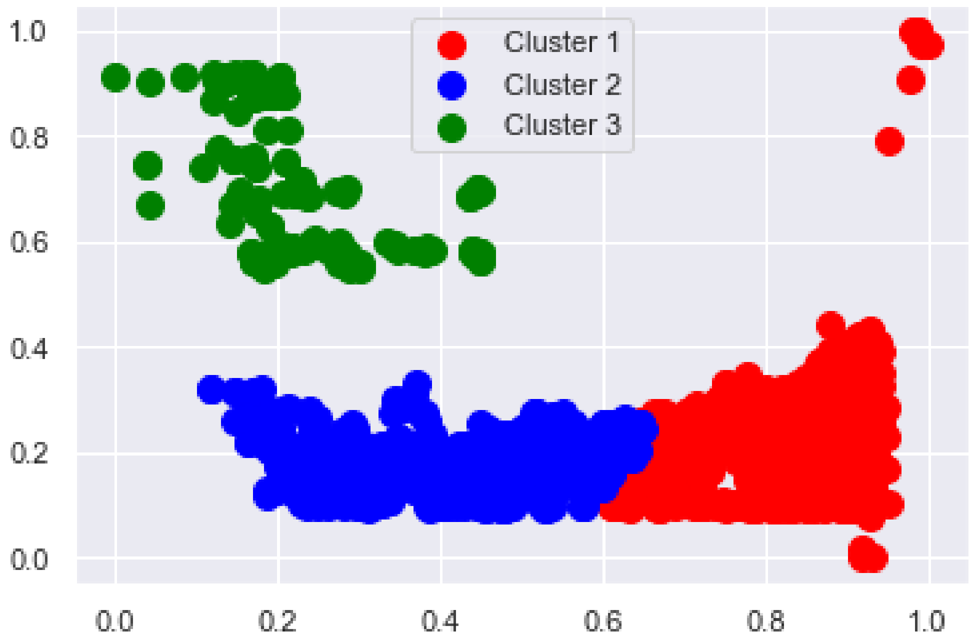
```
    plt.legend()
    plt.show()

    clust3 = AgglomerativeClustering(n_clusters=3, affinity='euclidean',␣
↪linkage='ward')
    y_pred = clust3.fit_predict(X)
    plt.scatter(X[y_pred == 0, 0], X[y_pred == 0, 1], s = 100, c = 'red', label␣
↪= 'Cluster 1')
    plt.scatter(X[y_pred == 1, 0], X[y_pred == 1, 1], s = 100, c = 'blue',␣
↪label = 'Cluster 2')
    plt.scatter(X[y_pred == 2, 0], X[y_pred == 2, 1], s = 100, c = 'green',␣
↪label = 'Cluster 3')
    plt.legend()
    plt.show()
    keep_clusters.append(clust3)

    clust4 = AgglomerativeClustering(n_clusters=4, affinity='euclidean',␣
↪linkage='ward')
    y_pred = clust4.fit_predict(X)
    plt.scatter(X[y_pred == 0, 0], X[y_pred == 0, 1], s = 100, c = 'red', label␣
↪= 'Cluster 1')
    plt.scatter(X[y_pred == 1, 0], X[y_pred == 1, 1], s = 100, c = 'blue',␣
↪label = 'Cluster 2')
    plt.scatter(X[y_pred == 2, 0], X[y_pred == 2, 1], s = 100, c = 'green',␣
↪label = 'Cluster 3')
    plt.scatter(X[y_pred == 3, 0], X[y_pred == 3, 1], s = 100, c = 'cyan',␣
↪label = 'Cluster 4')
    plt.legend()
    plt.show()

    clust5 = AgglomerativeClustering(n_clusters=5, affinity='euclidean',␣
↪linkage='ward')
    y_pred = clust5.fit_predict(X)
    plt.scatter(X[y_pred == 0, 0], X[y_pred == 0, 1], s = 100, c = 'red', label␣
↪= 'Cluster 1')
    plt.scatter(X[y_pred == 1, 0], X[y_pred == 1, 1], s = 100, c = 'blue',␣
↪label = 'Cluster 2')
    plt.scatter(X[y_pred == 2, 0], X[y_pred == 2, 1], s = 100, c = 'green',␣
↪label = 'Cluster 3')
    plt.scatter(X[y_pred == 3, 0], X[y_pred == 3, 1], s = 100, c = 'cyan',␣
↪label = 'Cluster 4')
    plt.scatter(X[y_pred == 4, 0], X[y_pred == 4, 1], s = 100, c = 'magenta',␣
↪label = 'Cluster 5')
    plt.legend()
    plt.show()
```

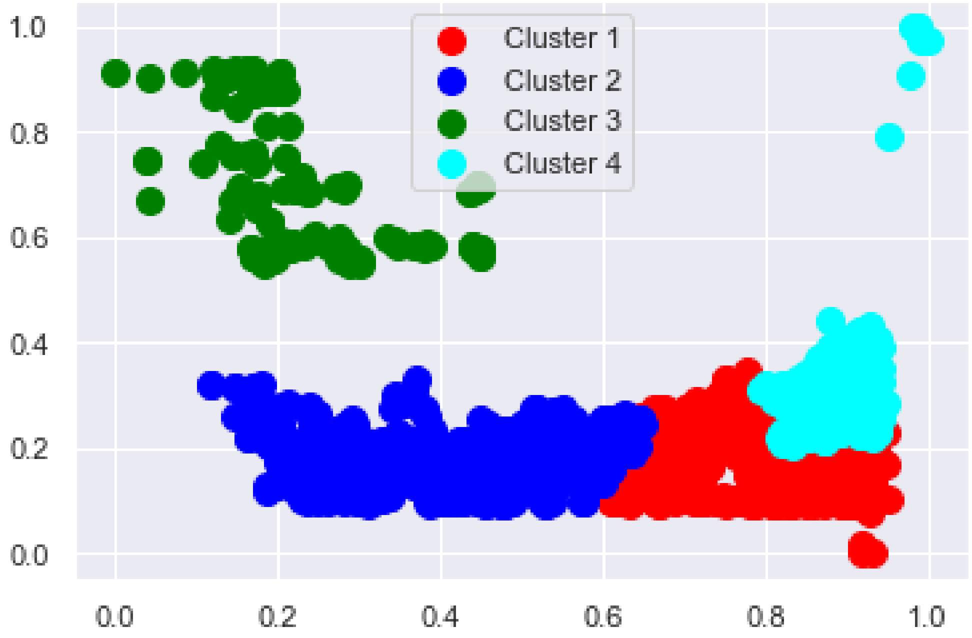
```python
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    km = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=31)
    km.fit(X)
    wcss.append(km.inertia_)

plt.plot(range(1,11), wcss)
plt.xlabel("Number of Clusters")
plt.ylabel("WCSS")
plt.title("KMeans Clustering Elbow Method")
plt.show()
print("*********************************************************")
```

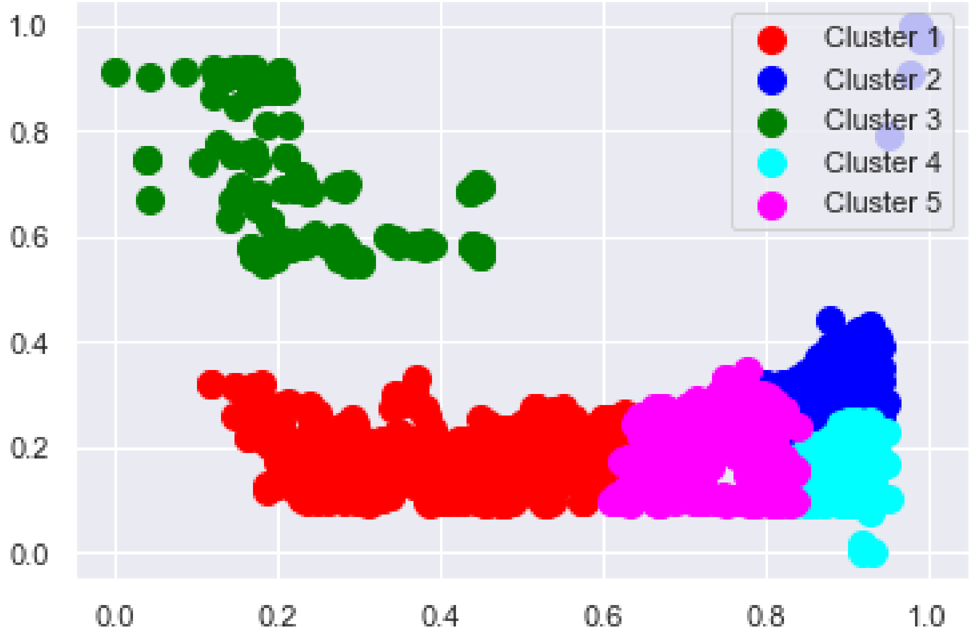
```
********************* twin params *******************
```




Cluster 1
Cluster 2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

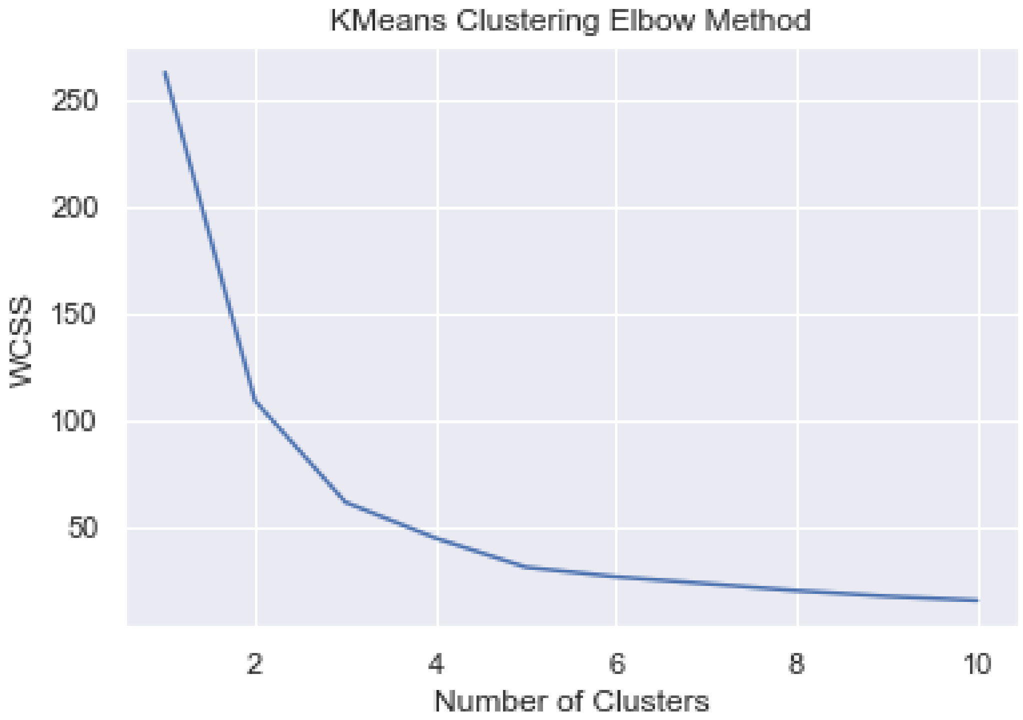

Cluster 1
Cluster 2
Cluster 3
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0



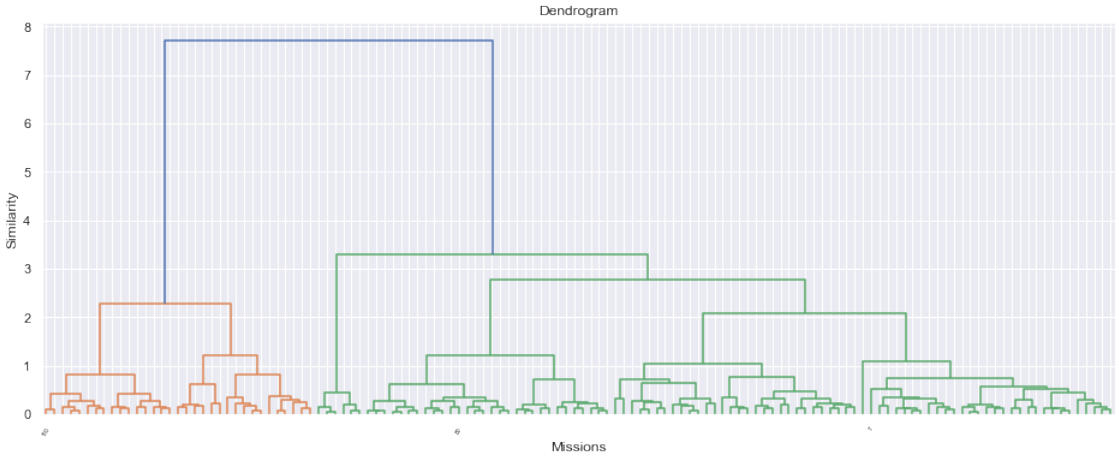


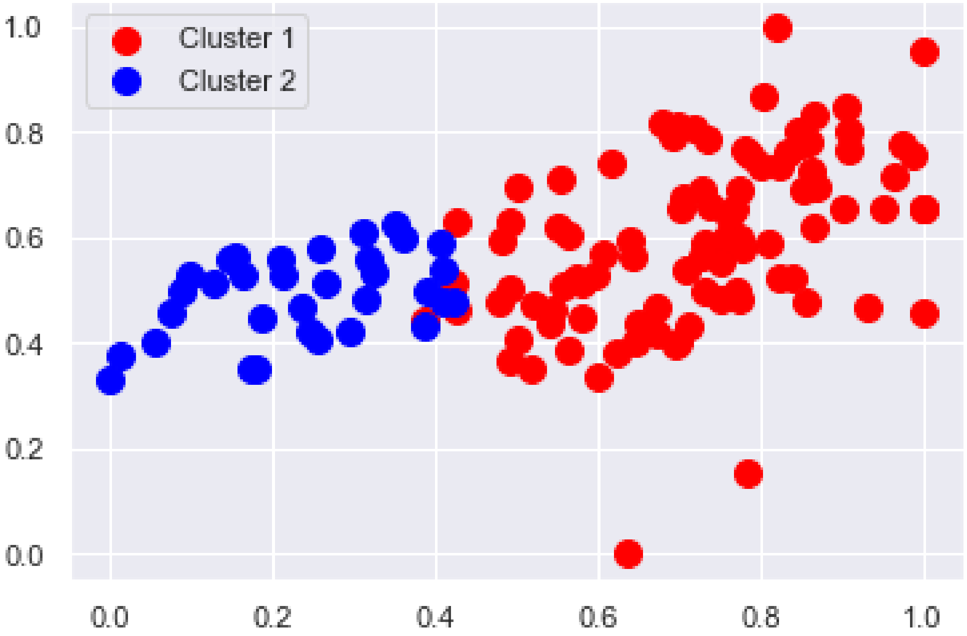
```
C:\Users\darrahts\anaconda3\envs\tf2x\lib\site-
packages\sklearn\cluster\_kmeans.py:882: UserWarning: KMeans is known to have a
memory leak on Windows with MKL, when there are less chunks than available
threads. You can avoid it by setting the environment variable
```

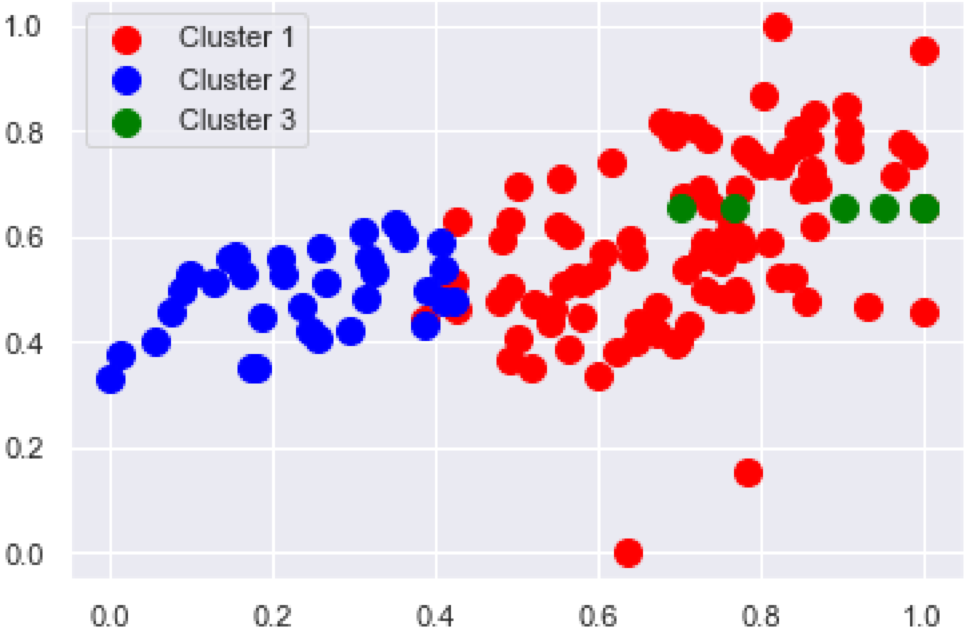
```
OMP_NUM_THREADS=16.
  f"KMeans is known to have a memory leak on Windows "
```

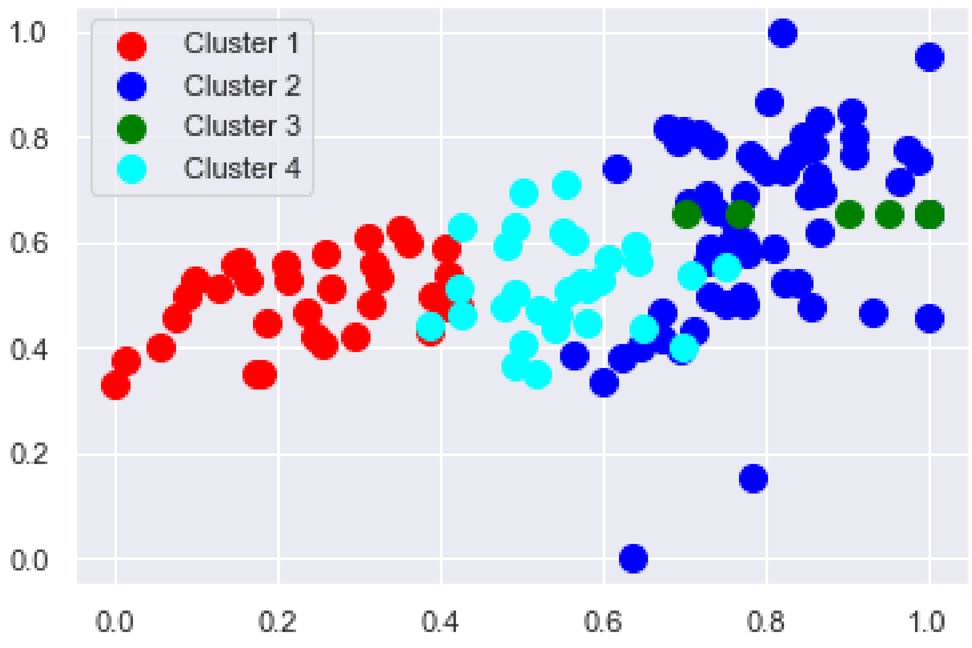


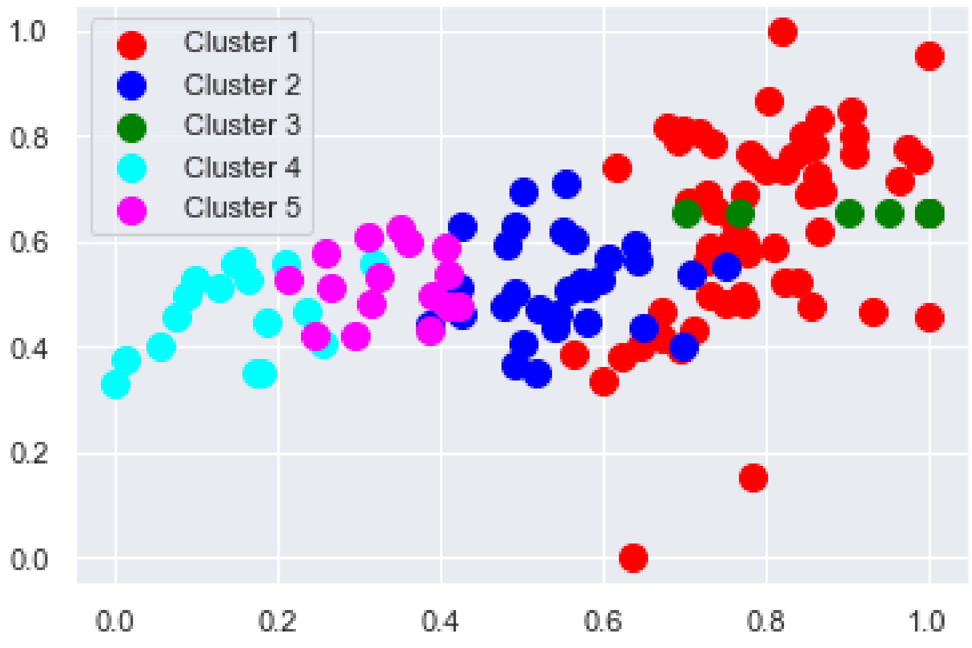
```
*********************************************************
********************* degradation data *******************
```




Cluster 1
Cluster 2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

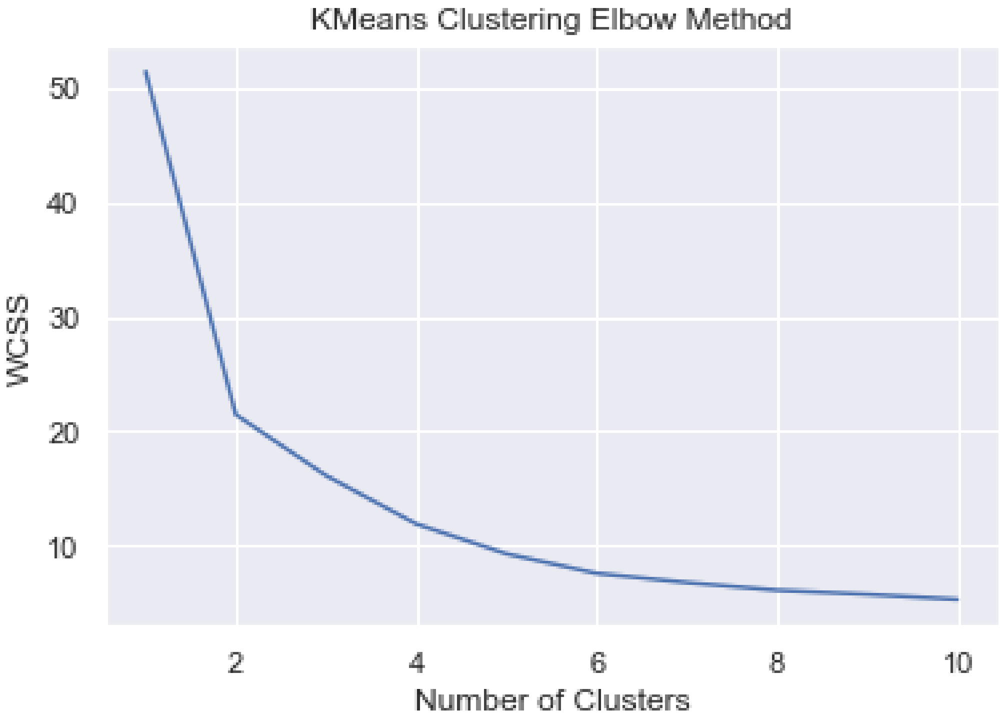

Cluster 1
Cluster 2
Cluster 3
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0





```
C:\Users\darrahts\anaconda3\envs\tf2x\lib\site-
packages\sklearn\cluster\_kmeans.py:882: UserWarning: KMeans is known to have a
memory leak on Windows with MKL, when there are less chunks than available
threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
```

```
f"KMeans is known to have a memory leak on Windows "
```



```
********************************************************
```

[185]:
```
len(mission_data_df)
```

[185]:
```
130
```

[218]:
```
sum(mission_data_df['cluster'] == 3)
```

[218]:
```
6
```

[367]:
```
mission_data_df['cluster'] = keep_clusters[1].labels_
mission_data_df['cluster'] = mission_data_df['cluster'] + 1
len(mission_data_df[mission_data_df['stop_code'] == mission_data_df['cluster']])
```

[367]:
```
11
```

[239]:
```
twin_params_df['cluster'] = keep_clusters[0].labels_
twin_params_df['cluster'] = twin_params_df['cluster'] + 1
```

[240]:
```
print(sum(stop1), sum(stop2), sum(stop3))
print(twin_params_df['cluster'].value_counts())
```

```
429 90 3531
1    3227
2     675
3     169
Name: cluster, dtype: int64
```

[246]:

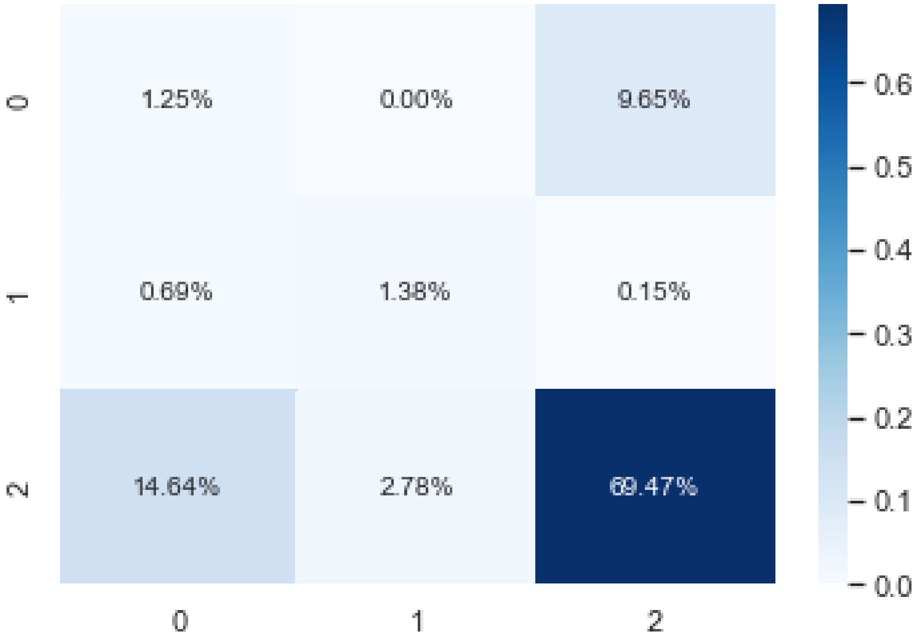
```
```

```
429 90 3531
3    3227
1     675
2     169
Name: cluster, dtype: int64
```

[368]:

```
mission_data_df['cluster'] = keep_clusters[1].labels_
mission_data_df['cluster'] = mission_data_df['cluster'] + 1



twin_params_df['cluster'] = keep_clusters[0].labels_
twin_params_df['cluster'] = twin_params_df['cluster'] + 1
temp = twin_params_df[['stop1', 'stop2', 'stop3', 'cluster']].iloc[:]
temp['cluster'].replace({1:4}, inplace=True)
temp['cluster'].replace({2:1}, inplace=True)
temp['cluster'].replace({3:2}, inplace=True)
temp['cluster'].replace({4:3}, inplace=True)

print(sum(stop1), sum(stop2), sum(stop3))
print(temp['cluster'].value_counts())


actual = np.ones(len(temp), dtype=int)
for i in range(0, len(actual)):
    if temp['stop1'].iloc[i]:
        actual[i] = 1
    elif temp['stop2'].iloc[i]:
        actual[i] = 2
    elif temp['stop3'].iloc[i]:
        actual[i] = 3
    else:
        print("error")

from sklearn.metrics import confusion_matrix
import seaborn as sns
cm = confusion_matrix(actual, temp['cluster'].values)/len(temp['cluster'])
sns.heatmap(cm, annot=True, fmt=".2%", cmap='Blues')

from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

# predict everything is 3
null_hypothesis = np.ones((len(temp['cluster']),))*3

res = null_hypothesis - actual
res[res != 0] = 1
sum(res)
print(1-sum(res)/len(res))

res = temp['cluster'].values - actual
res[res != 0] = 1
sum(res)
print(1-sum(res)/len(res))
```

```
429 90 3531
3    3227
1     675
2     169
Name: cluster, dtype: int64
0.868828297715549
0.7209530827806436
```



15 Feature selection of mission + degradation data on rul

[23]:

```
def normalize(data_df):
    data_df = data_df[:]
    data_df = (data_df - data_df.min()) / (data_df.max() - data_df.min())
    return data_df

def plot_loss(history):
  plt.plot(history.history['loss'], label='loss')
  plt.plot(history.history['val_loss'], label='val_loss')
  plt.ylim([0, 1])
  plt.xlabel('Epoch')
  plt.ylabel('Error [RUL]')
  plt.legend()
  plt.grid(True)
```

[48]:

```
data_df = DB.execute("select dpt.*, mt.* from degradation_parameter_tb dpt join␣
 ↪mission_tb mt on mt.id = dpt.mission_id order by mission_id desc;", db)
data_df.head()
data_df[data_df.isnull().any(axis=1)]
data_df = data_df.fillna(0)
data_df[data_df.isnull().any(axis=1)]
data_df = data_df[data_df['prior_rul'] > 0]
```

[16]:

```
data_df.head()
```

[16]:

```
     id  mission_id      q_deg  q_var    q_slope  q_intercept     r_deg  \
1  1619        1619  10.478152   0.25  -0.092328    22.517134  0.062589
2  1618        1618  10.650947   0.25  -0.089567    22.193118  0.060135
3  1617        1617  11.382871   0.25  -0.049539    17.418654  0.058179
4  1616        1616  10.892924   0.25  -0.088574    22.104463  0.056724
5  1615        1615  11.035111   0.25  -0.037697    15.916682  0.054831

    r_var   r_slope  r_intercept  …  z_end   v_end  avg_pos_err  \
1  0.0001  0.001715    -0.159339  …  0.5254  2.3546       1.4515
2  0.0001  0.001640    -0.150143  …  0.6092  0.9836       1.3294
3  0.0001  0.001602    -0.145503  …  0.5285  2.4704       1.3498
4  0.0001  0.001571    -0.141748  …  0.7575  2.5439       1.2452
5  0.0001  0.001510    -0.134410  …  0.7451  2.5898       1.2200

   max_pos_err  std_pos_err  avg_ctrl_err  max_ctrl_err  std_ctrl_err  \
1       3.6725       0.6726        0.2236        3.2601        1.0946
2       5.2724       0.7164        0.2933        3.4855        1.0051
3       3.3762       0.6866        0.4337        3.4760        0.9718
4       3.7023       0.7655        0.6271        3.7969        0.8008
5       3.4668       0.6874        0.6801        3.4391        0.7194
```

```
  battery_id uav_id
1          2      1
2          2      1
3          2      1
4          2      1
5          2      1

[5 rows x 35 columns]
```

[44]:

```python
y = data_df.pop('prior_rul').values
data_df.pop('id')
mid = data_df.pop('mission_id')
stp = data_df.pop('stop_code')
data_df.pop('dt_start')
data_df.pop('dt_stop')
data_df.pop('battery_id')
data_df.pop('uav_id')
data_df.pop('motor2_id')
data_df = normalize(data_df)
X = data_df.values
data_df.head()
```

[44]:

```
      q_deg     q_var   q_slope  q_intercept     r_deg  r_var   r_slope  \
0  0.707970  0.071429  0.409214     0.767338  0.907051    0.0  0.550832
1  0.676010  0.071429  0.390650     0.824576  0.884597    0.0  0.542681
2  0.687158  0.071429  0.394543     0.812710  0.849917    0.0  0.524989
3  0.734379  0.071429  0.450984     0.637870  0.822267    0.0  0.515983
4  0.702769  0.071429  0.395943     0.809464  0.801696    0.0  0.508736

   r_intercept     m_deg     m_var  …  flight_time  distance     z_end  \
0     0.186165  0.553211  0.130435  …     0.044131  0.045223  0.946235
1     0.206358  0.548056  0.130435  …     0.591060  0.607022  0.209217
2     0.249686  0.551477  0.130435  …     0.487923  0.446705  0.349140
3     0.271548  0.546834  0.130435  …     0.624761  0.633463  0.214393
4     0.289241  0.550787  0.130435  …     0.300987  0.294383  0.596761

      v_end  avg_pos_err  max_pos_err  std_pos_err  avg_ctrl_err  \
0  0.471584     0.676363     0.013230     0.001674      0.897781
1  0.452585     0.631890     0.317453     0.192800      0.588320
2  0.007340     0.511488     0.865814     0.245186      0.631763
3  0.490192     0.531604     0.215897     0.209544      0.719272
4  0.514062     0.428459     0.327667     0.303911      0.839816

   max_ctrl_err  std_ctrl_err
0      0.464782      0.163442
1      0.540715      0.379447
2      0.592345      0.294989
```

```
3      0.590169      0.263565
4      0.663674      0.102199

[5 rows x 23 columns]
```

[47]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.
 ↪SelectKBest.html
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.feature_selection import mutual_info_regression
from sklearn.feature_selection import chi2

methods = ['mutual info', 'f regression']


freg = f_regression(X, y)
best = SelectKBest(score_func=mutual_info_regression, k=12)
best_features = best.fit(X, y)
mutual_info = pd.DataFrame(best_features.scores_)
cols_df = pd.DataFrame(data_df.columns)
scores_df = pd.concat([cols_df, mutual_info, pd.DataFrame(freg[0])], axis=1)

scores_df.columns = ['parameter',methods[0], methods[1]]
scores_df.sort_values(by=methods[1], ascending=False)
```

[47]:

```
        parameter  mutual info  f regression
7     r_intercept     0.942607   2478.294092
4           r_deg     0.936711   2206.971420
14       distance     0.863893   2184.476822
16          v_end     0.930825   2091.648712
13    flight_time     0.880300   2081.693699
0           q_deg     0.708063   1852.502793
6         r_slope     0.891900   1416.915508
20   avg_ctrl_err     0.684204   1115.891968
15          z_end     0.302888    741.415600
5           r_var     0.705459    638.317807
1           q_var     0.535559    433.205780
8           m_deg     0.779617    347.202145
19    std_pos_err     0.552487    316.006052
9           m_var     0.576155    312.552632
3     q_intercept     0.221744    261.467892
22   std_ctrl_err     0.536165    209.723995
17    avg_pos_err     0.440959     78.896569
2         q_slope     0.206351     78.145442
21   max_ctrl_err     0.522129     15.928874
18    max_pos_err     0.534925      1.151482
10        m_slope     0.261464      0.506439
11    m_intercept     0.229048      0.257465
```

```
12  trajectory_id     0.649876      0.211133
```

16 Model evaluation

[239]:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers.experimental import preprocessing
import seaborn as sns

# json dict to hold model information
model_results = {}
```

16.0.1 get the data_df back in shape after altering it from feature selection methods

[406]:

```
def get_data(db):
    data_df = DB.execute("select dpt.*, mt.* from degradation_parameter_tb dpt␣
 ↪join mission_tb mt on mt.id = dpt.mission_id order by mission_id desc;", db)
    data_df = data_df.fillna(0)
    #data_df[data_df.isnull().any(axis=1)]
    data_df = data_df[data_df['prior_rul'] > 0]
    return data_df

data_df = get_data(db)
data_df.head()
```

[406]:

```
     id  mission_id      q_deg  q_var    q_slope  q_intercept     r_deg  \
0  1620        1620  10.973540   0.25 -0.079162    20.954117  0.064178
1  1619        1619  10.478152   0.25 -0.092328    22.517134  0.062589
2  1618        1618  10.650947   0.25 -0.089567    22.193118  0.060135
3  1617        1617  11.382871   0.25 -0.049539    17.418654  0.058179
4  1616        1616  10.892924   0.25 -0.088574    22.104463  0.056724

    r_var   r_slope  r_intercept  …   z_end   v_end  avg_pos_err  \
0  0.0001  0.001749    -0.163624  …  0.9668  2.4131       1.4966
1  0.0001  0.001715    -0.159339  …  0.5254  2.3546       1.4515
2  0.0001  0.001640    -0.150143  …  0.6092  0.9836       1.3294
3  0.0001  0.001602    -0.145503  …  0.5285  2.4704       1.3498
4  0.0001  0.001571    -0.141748  …  0.7575  2.5439       1.2452

   max_pos_err  std_pos_err  avg_ctrl_err  max_ctrl_err  std_ctrl_err  \
0       2.7849       0.5128        0.7201        2.9286        0.8657
1       3.6725       0.6726        0.2236        3.2601        1.0946
2       5.2724       0.7164        0.2933        3.4855        1.0051
3       3.3762       0.6866        0.4337        3.4760        0.9718
4       3.7023       0.7655        0.6271        3.7969        0.8008
```

```
  battery_id uav_id
0          2      1
1          2      1
2          2      1
3          2      1
4          2      1

[5 rows x 35 columns]
```

[365]:

```
data_df.columns
```

[365]:

```
Index(['id', 'mission_id', 'q_deg', 'q_var', 'q_slope', 'q_intercept', 'r_deg',
       'r_var', 'r_slope', 'r_intercept', 'm_deg', 'm_var', 'm_slope',
       'm_intercept', 'battery_id', 'motor2_id', 'uav_id', 'id', 'dt_start',
       'dt_stop', 'trajectory_id', 'stop_code', 'prior_rul', 'flight_time',
       'distance', 'z_end', 'v_end', 'avg_pos_err', 'max_pos_err',
       'std_pos_err', 'avg_ctrl_err', 'max_ctrl_err', 'std_ctrl_err',
       'battery_id', 'uav_id'],
      dtype='object')
```

16.0.2 Remove columns not needed for training

[384]:

```
data_df.pop('id')
mid = data_df.pop('mission_id')
stp = data_df.pop('stop_code')
trj = data_df.pop('trajectory_id')
data_df.pop('dt_start')
data_df.pop('dt_stop')
data_df.pop('battery_id')
data_df.pop('uav_id')
data_df.pop('motor2_id')
```

[384]:

```
0       2
1       2
2       2
3       2
4       2
       ..
1615    2
1616    2
1617    2
1618    2
1619    2
Name: motor2_id, Length: 1618, dtype: int64
```

[385]:

```
train_df = data_df.sample(frac=.8, random_state=tf.random.uniform(shape=[],␣
 ↪minval=0, maxval=9999, dtype='int32').numpy())
train_mid = mid.loc[train_df.index]
test_df = data_df.drop(train_df.index)
test_mid = mid.loc[test_df.index]
train_labels = train_df.pop('prior_rul')
test_labels = test_df.pop('prior_rul')

normalizer = preprocessing.Normalization()
normalizer.adapt(np.array(train_df))
```

16.0.3 Data features, 'prior_rul' is the predictor and removed for training

[368]:

```
data_df.columns.tolist()
```

[368]:

```
['q_deg',
 'q_var',
 'q_slope',
 'q_intercept',
 'r_deg',
 'r_var',
 'r_slope',
 'r_intercept',
 'm_deg',
 'm_var',
 'm_slope',
 'm_intercept',
 'prior_rul',
 'flight_time',
 'distance',
 'z_end',
 'v_end',
 'avg_pos_err',
 'max_pos_err',
 'std_pos_err',
 'avg_ctrl_err',
 'max_ctrl_err',
 'std_ctrl_err']
```

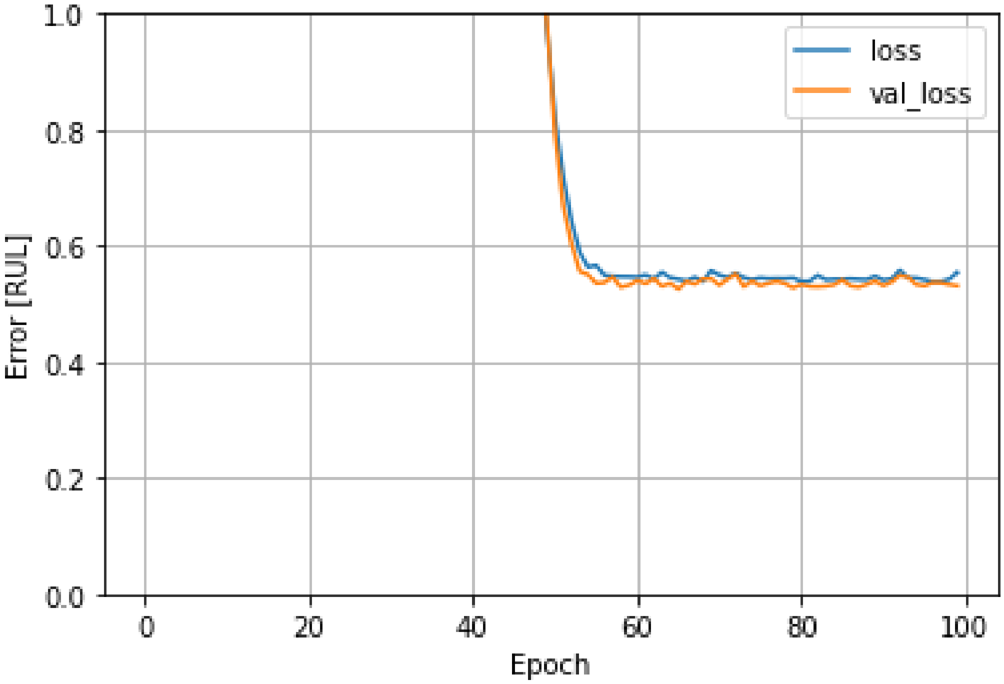
17 the model

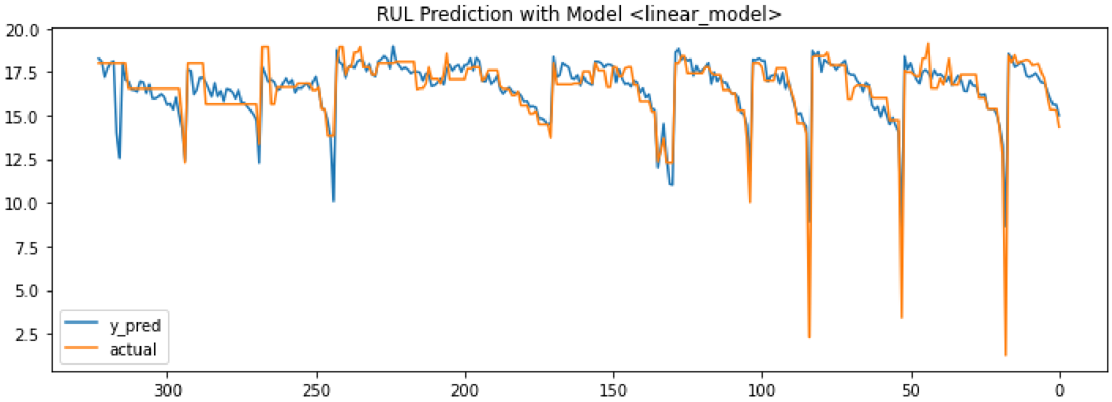
[387]:

```
normalizer = preprocessing.Normalization()
normalizer.adapt(np.array(train_df))

learning_rate = .01
layer_1_units = 1
```

```
epochs          = 100
val_split       = .2


linear_model = tf.keras.Sequential([normalizer, tf.keras.layers.
 ↪Dense(units=layer_1_units)])

names = ['linear_model']
models = [linear_model]

for model in zip(models, names):

    model[0].compile(optimizer=tf.optimizers.Adam(learning_rate=learning_rate),␣
 ↪loss='mean_absolute_error')

    history = linear_model.fit(train_df, train_labels, verbose=0,␣
 ↪epochs=epochs, validation_split=val_split)

    y_pred = model[0].predict(test_df)

    plot_loss(history)

    model_results[model[1]] = model[0].evaluate(test_df, test_labels, verbose=1)

    plt.figure(figsize=(12,4))
    plt.plot(y_pred, label='y_pred')
    plt.plot(test_labels.values, label='actual')
    plt.legend(loc='best')
    plt.title(f"RUL Prediction with Model <{model[1]}>")
    ax = plt.gca()
    ax.invert_xaxis()
    plt.show()
```

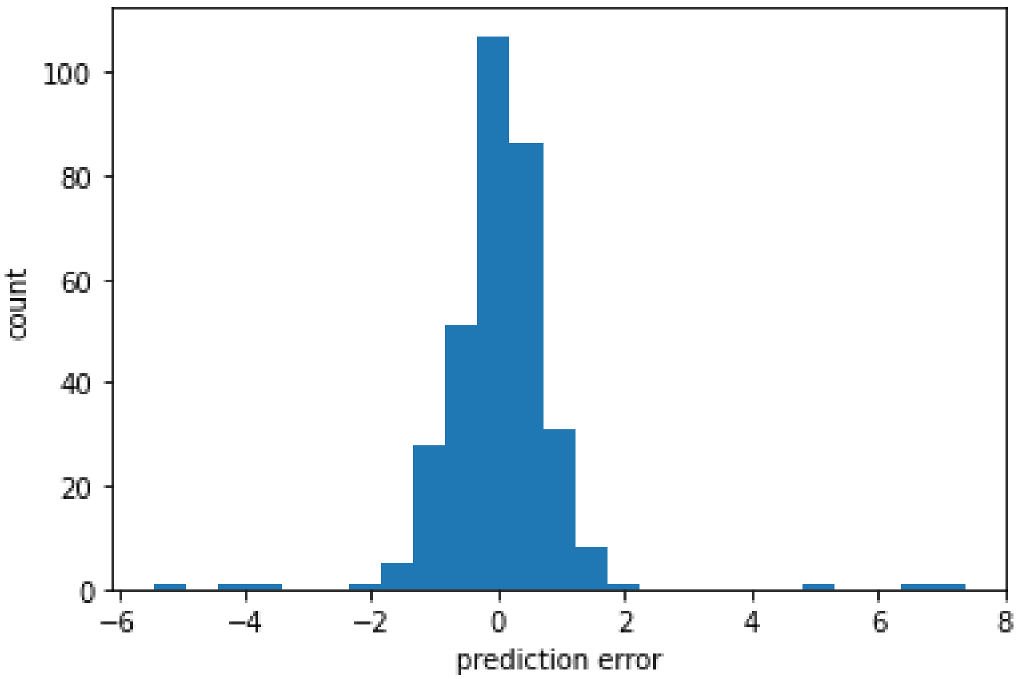
```
11/11 [==============================] - 0s 455us/step - loss: 0.5660
```





17.1 Lets look at the error distribution

[401]:
```
error = y_pred.flatten() - test_labels.values
plt.hist(error, bins=25)
plt.xlabel('prediction error')
_ = plt.ylabel('count')
plt.show()
```



17.1.1 The above plot is test data from training a linear model (one fully connected layer) to predict RUL from the true system and degradation data using all parameters. There are 1,620 total missions, this test sample represents 20% of that. The training data consists of appx 1300 records, where each record contains data from a single mission. Over the last 14 experiments, the UAV typically reaches EOL at around the 130th mission, so the data set contains approximately 10 samples per mission index in range [0, ~130]. The features are listed below

17.2 now reconstruct the mission indices to plot a distribution with 95% confidence bound

[370]:

```
mission_ids, mission_idx = get_all_experiments(res='mission')
mission_ids = np.concatenate(mission_ids)
mission_idx = np.concatenate(mission_idx)
data_df['mission_id'] = mid
data_df.head()
```

[370]:

```
       q_deg  q_var   q_slope  q_intercept     r_deg   r_var   r_slope  \
0  10.973540   0.25 -0.079162    20.954117  0.064178  0.0001  0.001749
1  10.478152   0.25 -0.092328    22.517134  0.062589  0.0001  0.001715
2  10.650947   0.25 -0.089567    22.193118  0.060135  0.0001  0.001640
3  11.382871   0.25 -0.049539    17.418654  0.058179  0.0001  0.001602
4  10.892924   0.25 -0.088574    22.104463  0.056724  0.0001  0.001571
```

```
   r_intercept     m_deg  m_var  …   distance   z_end   v_end  avg_pos_err  \
0    -0.163624  0.335395  0.005  …    69.4641  0.9668  2.4131       1.4966
1    -0.159339  0.332270  0.005  …   890.4648  0.5254  2.3546       1.4515
2    -0.150143  0.334344  0.005  …   656.1817  0.6092  0.9836       1.3294
3    -0.145503  0.331529  0.005  …   929.1057  0.5285  2.4704       1.3498
4    -0.141748  0.333926  0.005  …   433.5821  0.7575  2.5439       1.2452

   max_pos_err  std_pos_err  avg_ctrl_err  max_ctrl_err  std_ctrl_err  \
0       2.7849       0.5128        0.7201        2.9286        0.8657
1       3.6725       0.6726        0.2236        3.2601        1.0946
2       5.2724       0.7164        0.2933        3.4855        1.0051
3       3.3762       0.6866        0.4337        3.4760        0.9718
4       3.7023       0.7655        0.6271        3.7969        0.8008

   mission_id
0        1620
1        1619
2        1618
3        1617
4        1616

[5 rows x 24 columns]
```

[ ]:

```

```

[373]:

```python
test_df['mission_id'] = test_mid
midx = []
for i in range(0, len(test_df)):
    _id = test_df['mission_id'].iloc[i]
    res = np.where(mission_ids == _id)[0]
    if len(res) > 0:
        midx.append(mission_idx[res[0]])
    else:
        print(i, _id, res)
        test_df = test_df[test_df['mission_id'] != _id]
len(midx)
len(test_df)
test_df['midx'] = midx
test_df = test_df.sort_values(by='midx')
```

17.3 Now that we are sorted by mission index, we can aggregate the predictions

[374]:

```python
test_df.head(10)
```

[374]:

```
         q_deg  q_var  q_slope  q_intercept     r_deg    r_var  r_slope  \
789  15.000000   0.90      0.0          0.0  0.001100  0.00100      0.0
260  15.000000   0.90      0.0          0.0  0.001100  0.00100      0.0
```

```
128   13.801244   0.59      0.0          0.0  0.000639  0.00049       0.0
1092  15.000000   0.90      0.0          0.0  0.001100  0.00100       0.0
1564  15.134816   0.89      0.0          0.0  0.002357  0.00099       0.0
126   14.949655   0.57      0.0          0.0  0.002205  0.00047       0.0
1165  14.097361   0.88      0.0          0.0  0.000801  0.00098       0.0
987   13.913262   0.89      0.0          0.0  0.001771  0.00099       0.0
1429  14.997578   0.87      0.0          0.0  0.003058  0.00097       0.0
1616  14.934967   0.87      0.0          0.0  0.001094  0.00097       0.0

      r_intercept     m_deg    m_var  …   z_end   v_end  avg_pos_err  \
789           0.0  0.237100  0.02000  …  0.5178  4.0103       1.3181
260           0.0  0.237100  0.02000  …  0.5178  4.0103       1.3181
128           0.0  0.233529  0.00975  …  0.4300  4.0183       1.3256
1092          0.0  0.237100  0.02000  …  0.4193  4.0051       1.3134
1564          0.0  0.210237  0.02475  …  0.5883  3.9815       1.3331
126           0.0  0.223310  0.00925  …  0.4718  3.9788       1.3164
1165          0.0  0.218997  0.01950  …  0.4437  4.0150       1.2580
987           0.0  0.251445  0.01975  …  0.5282  3.9930       1.3243
1429          0.0  0.210809  0.02425  …  0.4752  3.9560       1.2685
1616          0.0  0.272745  0.02425  …  0.4784  4.0084       1.3431

      max_pos_err  std_pos_err  avg_ctrl_err  max_ctrl_err  std_ctrl_err  \
789        3.3425       0.7242        0.1407        3.4590        1.0065
260        3.3425       0.7242        0.1407        3.4590        1.0065
128        3.5643       0.8502        0.0899        3.4557        1.0424
1092       3.4085       0.7838        0.1056        3.4022        1.0262
1564       3.4316       0.7823        0.1275        3.4188        1.0398
126        3.5386       0.8467        0.0956        3.5081        1.0411
1165       3.2919       0.6774        0.1050        3.0655        0.9882
987        3.8746       0.7925        0.1165        3.8520        1.0365
1429       3.2803       0.6735        0.1458        3.0855        0.9923
1616       3.7596       0.8461        0.0705        3.2933        1.0571

      mission_id  midx
789          831     1
260         1360     1
128         1492     1
1092         528     2
1564          56     2
126         1494     3
1165         454     3
987          633     3
1429         191     4
1616           4     4

[10 rows x 24 columns]
```

[375]:

```
test_mids = test_df.pop('mission_id')
test_midx = test_df.pop('midx')
```

[376]:

```
y_pred = model[0].predict(test_df)
results = pd.DataFrame()
results['mission_id'] = test_mids
results['mission_idx'] = test_midx
results['actual'] = test_labels
results['y_pred'] = y_pred
results.head()
```

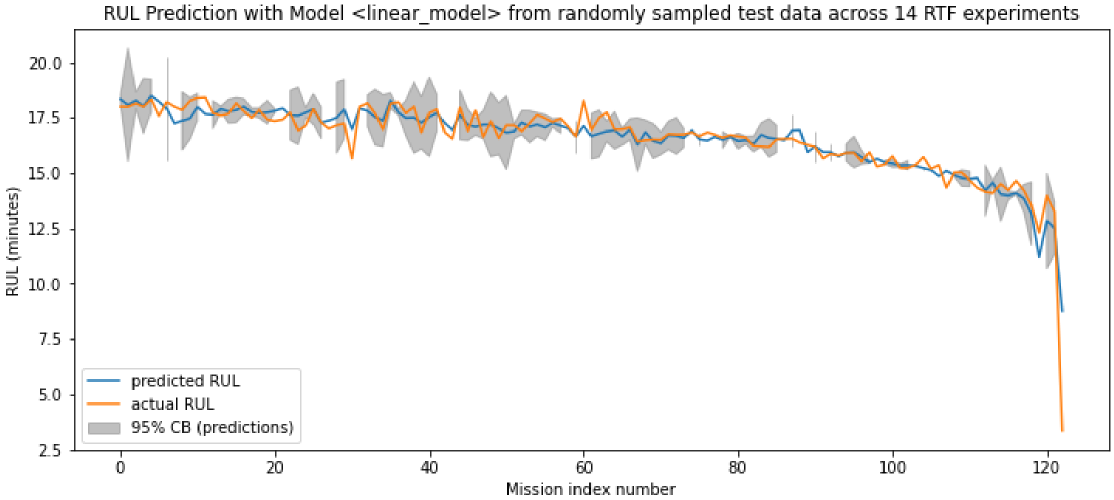
[376]:

```
      mission_id  mission_idx  actual     y_pred
789          831            1    18.0  18.284552
260         1360            1    18.0  18.284552
128         1492            1    18.0  18.461681
1092         528            2    18.0  18.984953
1564          56            2    18.0  17.171000
```

17.4 make sure we have values for each mission index in the test data

[377]:

```
def update_missing(results):
    lst = results['mission_idx'].unique().tolist()
    missing = [x for x in range(lst[0], lst[-1]+1) if x not in lst]
    for i in range(0, len(missing)):
        results.loc[len(results)] = [-1, missing[i]]



def find_missing(lst):
    return [x for x in range(lst[0], lst[-1]+1) if x not in lst]

find_missing(results['mission_idx'].unique().tolist())

def plot_pred_dist(results):
    pred_mus = tf.convert_to_tensor(results.groupby("mission_idx").
 ↪mean()['y_pred'].values, dtype='float32')
    pred_sds = tf.convert_to_tensor(results.groupby("mission_idx").
 ↪std()['y_pred'].values, dtype='float32')

    act_mus = tf.convert_to_tensor(results.groupby("mission_idx").
 ↪mean()['actual'].values, dtype='float32')
    act_sds = tf.convert_to_tensor(results.groupby("mission_idx").
 ↪std()['actual'].values, dtype='float32')

    plt.figure(figsize=(12,5))

    x = tf.range(0, pred_mus.shape[0], delta=1)
    plt.fill_between(x,
                    pred_mus-2*pred_sds,
                    pred_mus+2*pred_sds,
                    color='grey',
                    alpha=.5, label="95% CB (predictions)")
    plt.plot(x, pred_mus, label="predicted RUL")
    plt.plot(x, act_mus, label="actual RUL")
    plt.ylabel("RUL (minutes)")
    plt.xlabel('Mission index number')
    plt.title(f"RUL Prediction with Model <{model[1]}> from randomly sampled␣
 ↪test data across 14 RTF experiments")
    plt.legend(loc=3)
    #plt.text(-4, 11.8, f"*Calculated from data on {count} missions",␣
 ↪backgroundcolor='white')
    plt.show()

plot_pred_dist(results)
```



[382]:
```
len(test_labels)
```

[382]: 324

[ ]:

[378]:
```
# plt.figure(figsize=(12,4))
# plt.plot(y_pred, label='y_pred')
# #plt.plot(test_labels.values, label='actual')
# plt.legend(loc='best')
# plt.title(f"RUL Prediction with Model <{model[1]}> on test data across 14 RTF␣
 ↪experiments")
```

```
# ax = plt.gca()
# plt.show()
```

17.5 look at trajectory data

[411]:
```
data_df = get_data(db)
print(data_df.columns.tolist())
data_df.head()
```

```
['id', 'mission_id', 'q_deg', 'q_var', 'q_slope', 'q_intercept', 'r_deg',
'r_var', 'r_slope', 'r_intercept', 'm_deg', 'm_var', 'm_slope', 'm_intercept',
'battery_id', 'motor2_id', 'uav_id', 'id', 'dt_start', 'dt_stop',
'trajectory_id', 'stop_code', 'prior_rul', 'flight_time', 'distance', 'z_end',
'v_end', 'avg_pos_err', 'max_pos_err', 'std_pos_err', 'avg_ctrl_err',
'max_ctrl_err', 'std_ctrl_err', 'battery_id', 'uav_id']
```

[411]:
```
     id  mission_id      q_deg  q_var   q_slope  q_intercept     r_deg  \
0  1620        1620  10.973540   0.25 -0.079162    20.954117  0.064178
1  1619        1619  10.478152   0.25 -0.092328    22.517134  0.062589
2  1618        1618  10.650947   0.25 -0.089567    22.193118  0.060135
3  1617        1617  11.382871   0.25 -0.049539    17.418654  0.058179
4  1616        1616  10.892924   0.25 -0.088574    22.104463  0.056724

    r_var   r_slope  r_intercept  ...   z_end   v_end  avg_pos_err  \
0  0.0001  0.001749    -0.163624  ...  0.9668  2.4131       1.4966
1  0.0001  0.001715    -0.159339  ...  0.5254  2.3546       1.4515
2  0.0001  0.001640    -0.150143  ...  0.6092  0.9836       1.3294
3  0.0001  0.001602    -0.145503  ...  0.5285  2.4704       1.3498
4  0.0001  0.001571    -0.141748  ...  0.7575  2.5439       1.2452

   max_pos_err  std_pos_err  avg_ctrl_err  max_ctrl_err  std_ctrl_err  \
0       2.7849       0.5128        0.7201        2.9286        0.8657
1       3.6725       0.6726        0.2236        3.2601        1.0946
2       5.2724       0.7164        0.2933        3.4855        1.0051
3       3.3762       0.6866        0.4337        3.4760        0.9718
4       3.7023       0.7655        0.6271        3.7969        0.8008

  battery_id uav_id
0          2      1
1          2      1
2          2      1
3          2      1
4          2      1

[5 rows x 35 columns]
```

[412]:

```
trajectory_df = pd.DataFrame()
trajectory_df['trajectory_id'] = data_df.pop('trajectory_id')
trajectory_df['stop_code'] = data_df.pop('stop_code')
trajectory_df.head()
```

[412]:

```
   trajectory_id  stop_code
0              7          3
1             18          3
2              4          2
3              4          3
4             17          3
```

[416]:

```
trajectory_df = trajectory_df.sort_values(['trajectory_id', 'stop_code'])
trajectory_df.head(10)
```

[416]:

```
      trajectory_id  stop_code
1137              2          1
1092              2          1
1065              2          1
1059              2          1
1124              2          1
394               2          1
389               2          1
387               2          1
383               2          1
377               2          1
```

[554]:

```
traj_df = trajectory_df.groupby(['trajectory_id', 'stop_code']).
 ↪agg({'trajectory_id': ['min'], 'stop_code': ['min', 'count']})

trajectory_id = 2
count = 0

totals = []

for i in range(0, len(traj_df)):
    tid = int(traj_df.iloc[i].values[0])
    sc = int(traj_df.iloc[i].values[1])
    ct = int(traj_df.iloc[i].values[2])

    if tid == trajectory_id:
        count = count + ct
    else:
        totals.append(count)
        count = ct
        trajectory_id = tid
totals.append(count)
```

```
trajectory_id = 2
j = 0

newtotals = []

for i in range(0, len(traj_df)):
    tid = int(traj_df.iloc[i].values[0])
    sc = int(traj_df.iloc[i].values[1])
    if tid == trajectory_id:
        newtotals.append(totals[j])
    else:
        j = j + 1
        trajectory_id = tid
        newtotals.append(totals[j])

traj_df['totals'] = newtotals

pcts = []
for i in range(0, len(traj_df)):
    pct = traj_df.iloc[i].values[2] / traj_df.iloc[i].values[3]
    pcts.append(pct)

traj_df['pcts'] = pcts

trajectory_id = 2

ids = []

for i in range(0, len(traj_df)):
    tid = int(traj_df.iloc[i].values[0])
    if tid == trajectory_id:
        ids.append(tid)
    else:
        trajectory_id = tid
        ids.append(tid)
traj_df['ids'] = ids
traj_df.head()


traj_df
```

```
[554]:                         trajectory_id stop_code       totals      pcts ids
                                         min       min count                   
       trajectory_id stop_code                                                 
       2             1                     2         1    12     16  0.750000   2
                     3                     2         3     4     16  0.250000   2
```

```
3              1                      3         1   15    129  0.116279   3
               3                      3         3  114    129  0.883721   3
4              2                      4         2    4      9  0.444444   4
               3                      4         3    5      9  0.555556   4
5              1                      5         1    1     14  0.071429   5
               2                      5         2    2     14  0.142857   5
               3                      5         3   11     14  0.785714   5
6              1                      6         1   10     16  0.625000   6
               3                      6         3    6     16  0.375000   6
7              3                      7         3   16     16  1.000000   7
8              1                      8         1    2     43  0.046512   8
               2                      8         2    3     43  0.069767   8
               3                      8         3   38     43  0.883721   8
9              1                      9         1    2    117  0.017094   9
               2                      9         2    6    117  0.051282   9
               3                      9         3  109    117  0.931624   9
10             2                     10         2   64    235  0.272340  10
               3                     10         3  171    235  0.727660  10
11             1                     11         1    9    244  0.036885  11
               2                     11         2    5    244  0.020492  11
               3                     11         3  230    244  0.942623  11
13             2                     13         2    1    267  0.003745  13
               3                     13         3  266    267  0.996255  13
14             1                     14         1    1    336  0.002976  14
               2                     14         2    1    336  0.002976  14
               3                     14         3  334    336  0.994048  14
15             1                     15         1   20    117  0.170940  15
               2                     15         2    3    117  0.025641  15
               3                     15         3   94    117  0.803419  15
16             2                     16         2    2     13  0.153846  16
               3                     16         3   11     13  0.846154  16
17             3                     17         3    7      7  1.000000  17
18             2                     18         2    3     15  0.200000  18
               3                     18         3   12     15  0.800000  18
19             2                     19         2    1      9  0.111111  19
               3                     19         3    8      9  0.888889  19
20             1                     20         1    7     15  0.466667  20
               3                     20         3    8     15  0.533333  20
```

[555]:

```
ids = traj_df.pop('ids')
tmin = traj_df.pop('trajectory_id')
totals = traj_df.pop('totals')
scs = traj_df.pop('stop_code')
traj_df
```

[555]:

```
                              pcts
```

```
trajectory_id stop_code
2             1            0.750000
              3            0.250000
3             1            0.116279
              3            0.883721
4             2            0.444444
              3            0.555556
5             1            0.071429
              2            0.142857
              3            0.785714
6             1            0.625000
              3            0.375000
7             3            1.000000
8             1            0.046512
              2            0.069767
              3            0.883721
9             1            0.017094
              2            0.051282
              3            0.931624
10            2            0.272340
              3            0.727660
11            1            0.036885
              2            0.020492
              3            0.942623
13            2            0.003745
              3            0.996255
14            1            0.002976
              2            0.002976
              3            0.994048
15            1            0.170940
              2            0.025641
              3            0.803419
16            2            0.153846
              3            0.846154
17            3            1.000000
18            2            0.200000
              3            0.800000
19            2            0.111111
              3            0.888889
20            1            0.466667
              3            0.533333
```

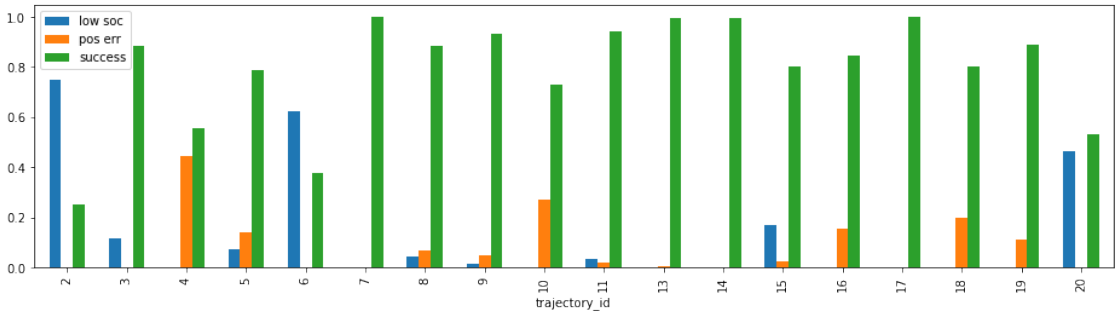
[ ]:

[556]:
```
traj_us = traj_df.unstack().fillna(0)
traj_us.head()
```

```
[556]:                    pcts

        stop_code            1         2         3
        trajectory_id
        2             0.750000  0.000000  0.250000
        3             0.116279  0.000000  0.883721
        4             0.000000  0.444444  0.555556
        5             0.071429  0.142857  0.785714
        6             0.625000  0.000000  0.375000
```

```
[579]: fig, ax = plt.subplots()
       traj_us.plot(kind='bar', width=.6, align='center', ax=ax, figsize=(16,4))
       ax.legend(["low soc", "pos err", "success"])
```

```
[579]: <matplotlib.legend.Legend at 0x26211c27588>
```



```
[586]: trajectory_df = DB.execute("select tt.* from trajectory_tb tt order by␣
        ↪path_time desc;", db)
       trajectory_df.sort_values(by='path_time', ascending=False)
```

```
[586]:     id  path_distance  path_time  risk_factor  \
        0   12        1641.73      21.05         0.01
        1    1        1637.12      20.99         0.01
        2    2        1553.56      19.92         0.01
        3   20        1532.93      19.65         0.01
        4    6        1466.96      18.81         0.01
        5    3        1390.64      17.83         0.01
        6   15        1375.04      17.63         0.01
        7   11        1277.49      16.38         0.01
        8   14        1155.53      14.81         0.01
        9   13        1087.02      13.94         0.01
        10  10        1058.49      13.57         0.01
        11   9        1049.45      13.45         0.01
        12   8        1038.04      13.31         0.01
        13   5        1037.58      13.30         0.01
```

```
14   4         974.22      12.49         0.01
15  18         918.98      11.78         0.01
16  16         497.71       6.38         0.01
17  17         469.96       6.03         0.01
18  19         337.51       4.33         0.01
19   7          70.94       0.91         0.01

                                  x_waypoints  \
0           [450.0, 330.0, 80.0, 120.0, 45.0]
1   [30.0, 200.0, 100.0, 410.0, 450.0, 200.0]
2          [30.0, 100.0, 410.0, 450.0, 200.0]
3    [70.0, 200.0, 100.0, 440.0, 120.0, 45.0]
4          [30.0, 200.0, 100.0, 410.0, 200.0]
5                 [30.0, 100.0, 410.0, 200.0]
6    [260.0, 440.0, 330.0, 80.0, 120.0, 45.0]
7                 [450.0, 330.0, 80.0, 120.0]
8           [260.0, 330.0, 80.0, 120.0, 45.0]
9                  [330.0, 80.0, 120.0, 45.0]
10                       [450.0, 330.0, 80.0]
11                      [200.0, 440.0, 350.0]
12                      [200.0, 450.0, 200.0]
13                [30.0, 200.0, 410.0, 450.0]
14                [30.0, 200.0, 100.0, 410.0]
15                              [440.0, 80.0]
16                             [260.0, 440.0]
17                             [260.0, 330.0]
18                              [45.0, 120.0]
19                                     [70.0]

                                  y_waypoints  \
0          [50.0, 230.0, 375.0, 240.0, 165.0]
1    [150.0, 235.0, 350.0, 380.0, 50.0, 30.0]
2           [150.0, 350.0, 380.0, 50.0, 30.0]
3   [90.0, 345.0, 350.0, 380.0, 240.0, 165.0]
4          [150.0, 235.0, 350.0, 380.0, 30.0]
5                 [150.0, 350.0, 380.0, 30.0]
6   [80.0, 190.0, 230.0, 375.0, 240.0, 165.0]
7                 [50.0, 230.0, 375.0, 240.0]
8          [80.0, 230.0, 375.0, 240.0, 165.0]
9                [230.0, 375.0, 240.0, 165.0]
10                       [50.0, 230.0, 375.0]
11                      [345.0, 380.0, 150.0]
12                        [235.0, 50.0, 30.0]
13                [150.0, 235.0, 380.0, 50.0]
14               [150.0, 235.0, 350.0, 380.0]
15                             [190.0, 375.0]
16                              [80.0, 190.0]
```

```
17                              [80.0, 230.0]
18                             [165.0, 240.0]
19                                     [90.0]

                                         x_ref_points  \
0   [49.74, 49.58, 49.52, 49.55, 49.66, 49.85, 50…
1   [50.7, 51.4, 52.09, 52.78, 53.45, 54.12, 54.78…
2   [49.17, 48.52, 48.02, 47.67, 47.46, 47.38, 47…
3   [50.68, 51.33, 51.97, 52.58, 53.18, 53.76, 54…
4   [50.51, 51.04, 51.59, 52.14, 52.7, 53.27, 53.8…
5   [51.07, 52.08, 53.04, 53.94, 54.79, 55.59, 56…
6   [48.97, 48.18, 47.62, 47.27, 47.12, 47.15, 47…
7   [49.98, 50.01, 50.1, 50.25, 50.45, 50.69, 50.9…
8   [49.18, 48.53, 48.05, 47.72, 47.54, 47.5, 47.5…
9   [50.28, 50.6, 50.98, 51.4, 51.85, 52.35, 52.86…
10  [49.9, 49.87, 49.91, 50.01, 50.18, 50.4, 50.67…
11  [49.93, 49.91, 49.95, 50.04, 50.17, 50.35, 50…
12  [49.59, 49.33, 49.21, 49.22, 49.34, 49.58, 49…
13  [50.81, 51.61, 52.39, 53.15, 53.9, 54.62, 55.3…
14  [50.59, 51.2, 51.8, 52.4, 53.01, 53.61, 54.2, …
15  [49.9, 49.94, 50.12, 50.43, 50.86, 51.39, 52.0…
16  [50.2, 50.49, 50.85, 51.28, 51.78, 52.34, 52.9…
17  [50.29, 50.66, 51.11, 51.62, 52.2, 52.83, 53.5…
18  [50.09, 50.26, 50.53, 50.86, 51.26, 51.73, 52…
19  [50.69, 51.36, 52.01, 52.63, 53.23, 53.81, 54…

                                         y_ref_points  sample_time  reward
0   [24.45, 24.12, 23.98, 24.04, 24.28, 24.69, 25…            1     1.0
1   [24.81, 24.94, 25.37, 26.1, 27.09, 28.33, 29.7…            1     1.0
2   [24.46, 24.11, 23.94, 23.95, 24.12, 24.46, 24…            1     1.0
3   [24.77, 24.75, 24.92, 25.28, 25.82, 26.52, 27…            1     1.0
4   [25.94, 26.94, 27.99, 29.1, 30.25, 31.45, 32.6…            1     1.0
5   [25.49, 26.0, 26.52, 27.07, 27.63, 28.22, 28.8…            1     1.0
6   [24.68, 24.56, 24.62, 24.86, 25.26, 25.81, 26…            1     1.0
7   [24.1, 23.4, 22.88, 22.54, 22.38, 22.37, 22.52…            1     1.0
8   [25.25, 25.57, 25.95, 26.4, 26.92, 27.5, 28.14…            1     1.0
9   [25.29, 25.78, 26.46, 27.32, 28.34, 29.53, 30…            1     1.0
10  [24.62, 24.4, 24.32, 24.38, 24.57, 24.89, 25.3…            1     1.0
11  [24.1, 23.39, 22.86, 22.52, 22.34, 22.31, 22.4…            1     1.0
12  [25.86, 26.73, 27.61, 28.49, 29.38, 30.27, 31…            1     1.0
13  [25.0, 25.21, 25.62, 26.22, 26.98, 27.92, 29.0…            1     1.0
14  [25.83, 26.71, 27.65, 28.63, 29.66, 30.74, 31…            1     1.0
15  [25.8, 26.76, 27.86, 29.1, 30.45, 31.92, 33.49…            1     1.0
16  [25.03, 25.31, 25.82, 26.55, 27.47, 28.58, 29…            1     1.0
17  [24.72, 24.68, 24.87, 25.28, 25.88, 26.68, 27…            1     1.0
18  [24.32, 24.02, 24.07, 24.44, 25.11, 26.05, 27…            1     1.0
19  [25.56, 26.16, 26.8, 27.48, 28.2, 28.96, 29.77…            1     1.0
```

[ ]:

```
```

[ ]:

```
```

[379]:

```
# conn = psycopg2.connect(dbname="tsdb", user="postgres",
#           password="8rK2Q@99Ad0uo!Wb",host="144.126.248.145",port=5432)
# cur = conn.cursor()
```

[583]:

```
# params = Utils.get_aws_secret("/secret/uav_db")
# db, cur =  DB.connect(params)
# del(params)
# DB.get_tables(db)
```

```
[INFO] connecting to db.
[INFO] connected.
```

[583]:

```
                   table_name
0                    model_tb
1                      uav_tb
2              eqc_battery_tb
3                 eq_motor_tb
4     degradation_parameter_tb
5                  mission_tb
6          pg_stat_statements
7           battery_sensor_tb
8            flight_sensor_tb
9               experiment_tb
10             twin_params_tb
11              trajectory_tb
```