```
#!/usr/bin/env python
# coding: utf-8

# # Code Part :  'Battery Health Management for small Satellites' Project

# ### Jingyi Chen (jc2498), Shuo Feng (sf587), Bingjie Huang (bh572), Lina Takemaru
(llt45)

# ## 2. Data Description

# In[ ]:


#%%Loading package and data
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from sklearn.metrics import mean_squared_error
from sklearn import linear_model
from sklearn.neural_network import MLPRegressor
#%%
from mat4py import loadmat

data = loadmat('RW9.mat')
#Pulling out all the data from raw dataset

data2=data['data']

step=data2['step']

#Eight features
comment=pd.Series(step['comment'])
Type=pd.Series(step['type'])
current=pd.Series(step['current'])
time=pd.Series(step['time'])
relativeTime=pd.Series(step['relativeTime'])
voltage=pd.Series(step['voltage'])
temperature=pd.Series(step['temperature'])
date=pd.Series(step['date'])
#This df is our whole Dataframe
df=pd.DataFrame([comment,Type,date,current,time,relativeTime,voltage,temperature])
df=df.T
df.columns=['comment','Type','date','current','time','relativeTime','voltage','temperature']
#%%
# df.to_csv('df_RW9.csv')
# df=pd.read_csv('df_RW9.csv',index_col=0)
#%%There are 15 different comment types
# comment_type = []
# for i in comment:
#     comment_type.append(i)
# comment_type = set(comment_type)
# comment_type=list(comment_type)

comment_type=df['comment'].unique()


fig, (ax1, ax2) = plt.subplots(2, 1,figsize=(10,20))
plt.rcParams.update({'font.size': 24})

index=[index for index,value in enumerate(comment) if value=='reference discharge']
```

```
newlist=[]
for i in index:
    # for j in range(len(relativeTime[i])):
    #     newlist=[relativeTime[i][j]]
    ax1.plot(relativeTime[i],current[i])


ax1.set_title("Reference Discharge for Current")
ax1.set_xlabel('Time(s)')
ax1.set_ylabel('Current(A)')
for i in index:

    ax2.plot(relativeTime[i],voltage[i])


ax2.set_title("Reference Discharge for Voltage")
ax2.set_xlabel('Time(s)')
ax2.set_ylabel('Voltage(V)')


#%%
#Pulsed load (rest+ discharge) one cycle for current and voltage
#Dataset index 4:30
pulsed_cycle_current=[]
pulsed_cycle_voltage=[]
plused_time=[]
for i in range(4,30):
    plused_time+=time[i]
    pulsed_cycle_current+=current[i]
    pulsed_cycle_voltage+=voltage[i]


plused_time_final=[]
for i in range(len(plused_time)):
    plused_time_final.append((plused_time[i]-plused_time[0])/60)

plt.rcParams.update({'font.size': 24})
fig, (ax1, ax2) = plt.subplots(2, 1,figsize=(10,20))
ax1.plot(plused_time_final,pulsed_cycle_current)
ax1.set_title("Pulsed Load One Cycle for Current")
ax1.set_xlabel('Time(minutes)')
ax1.set_ylabel('Current(A)')
ax2.plot(plused_time_final,pulsed_cycle_voltage)
ax2.set_title("Pulsed Load One Cycle for Voltage")
ax2.set_xlabel('Time(minutes)')
ax2.set_ylabel('Voltage(V)')
#%%
#Random walk (rest+discharge+charge) one cycle for current and voltage
#Dataset index 30:101
# df_RW = df[(df['comment'] == 'rest (random walk)') | (df['comment'] == 'charge
(random walk)') | (df['comment'] == 'discharge (random walk)') ]
# df_RDC = df[df['comment'] == 'reference discharge']
random_walk_current=[]
random_walk_voltage=[]


RW_time=[]



for i in range(31,100):
    RW_time+=time[i]
```

```
    random_walk_current+=current[i]
    random_walk_voltage+=voltage[i]

RW_time_final=[]
for i in range(len(RW_time)):
    RW_time_final.append((RW_time[i]-RW_time[0])/60)

plt.rcParams.update({'font.size': 24})
fig, (ax1, ax2) = plt.subplots(2, 1,figsize=(10,20))
ax1.plot(RW_time_final,random_walk_current)
ax1.set_title("RW One Cycle for Current: Discharge + Rest")
ax1.set_xlabel('Time(minutes)')
ax1.set_ylabel('Current(A)')
ax2.plot(RW_time_final,random_walk_voltage)
ax2.set_title("RW One Cycle for Voltage: Discharge + Rest")
ax2.set_xlabel('Time(minutes)')
ax2.set_ylabel('Voltage(V)')
#%%


#Plotting all relativeTime vs Voltage for different Comment Type


fig =plt.figure(figsize = (16,14))

for count,j in enumerate(comment_type):
    index=[index for index,value in enumerate(comment) if value==j]
    for i in index:
        ax =fig.add_subplot(4,4,count+1)
        ax.plot(relativeTime[i],voltage[i])
        plt.title(str(j))
fig.show()

#Plotting all relativeTime vs Current for different Comment Type


fig =plt.figure(figsize = (16,14))

for count,j in enumerate(comment_type):
    index=[index for index,value in enumerate(comment) if value==j]
    for i in index:
        ax =fig.add_subplot(4,4,count+1)
        ax.plot(relativeTime[i],current[i])
        plt.title(str(j))
fig.show()

#Plotting all relativeTime vs Temperature for different Comment Type


fig =plt.figure(figsize = (16,14))

for count,j in enumerate(comment_type):
    index=[index for index,value in enumerate(comment) if value==j]
    for i in index:
        ax =fig.add_subplot(4,4,count+1)
        ax.plot(relativeTime[i],temperature[i])
        plt.title(str(j))
fig.show()


# ## 3.3 Battery Voltage Forecast
```

```
# ## 3.1 Feature Measurement & 3.2 SOH Analysis

# ### SOH on Reference Discharge Period

# In[ ]:



# # State of Health Analysis on Reference Discharge Period

# ## Prepare Data Set

# In[2]:


import pandas as pd
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import statsmodels.api as sm
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn import linear_model
from sklearn.neural_network import MLPRegressor
from mat4py import loadmat
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score


# In[3]:


data = loadmat('RW9.mat')


# In[4]:


#Pulling out all the data from raw dataset

data2=data['data']

step=data2['step']


# In[5]:


#Eight features
comment=pd.Series(step['comment'])
Type=pd.Series(step['type'])
current=pd.Series(step['current'])
time=pd.Series(step['time'])
relativeTime=pd.Series(step['relativeTime'])
voltage=pd.Series(step['voltage'])
temperature=pd.Series(step['temperature'])
date=pd.Series(step['date'])
#This df is our whole Dataframe
df=pd.DataFrame([comment,Type,date,current,time,relativeTime,voltage,temperature])
df=df.T
df.columns=['comment','Type','date','current','time','relativeTime','voltage','temperature']
```

```
#%%There are 15 different comment types
comment_type = []
for i in comment:
    comment_type.append(i)
comment_type = set(comment_type)
comment_type=list(comment_type)


# ## Pulling Out Data for Reference Discharge Period

# In[7]:


# Pulling out data of reference dsicharge
df_RDC = np.array(df[df['comment'] == 'reference discharge'])
df_RDC=pd.DataFrame(df_RDC)
df_RDC.columns=['comment','Type','date','current','time','relativeTime','voltage','temperature']

size_RDC=[]
for i in range(len(df_RDC)):
    size_RDC.append(len(df_RDC['relativeTime'][i]))



from statistics import mean
df_RDC['current_avg'] = df_RDC['current'].map(mean)
df_RDC['time_avg'] = df_RDC['time'].map(mean)
df_RDC['relativeTime_avg'] = df_RDC['relativeTime'].map(mean)
df_RDC['voltage_avg'] = df_RDC['voltage'].map(mean)
df_RDC['temperature_avg'] = df_RDC['temperature'].map(mean)


# # State of Health on Reference Discharge Period

# # Features

# In[8]:


# discharging time
df_RDC['duration'] = 0
for i in range(0,len(df_RDC)):
    df_RDC['duration'][i] = max(df_RDC['relativeTime'][i]) - 
min(df_RDC['relativeTime'][i])


# In[9]:


# ploting discharing time against number of cycles

duration = df_RDC['duration'].copy()
duration = [i for i in duration if i != 1199]

plt.figure(figsize=(8, 6))

plt.plot(duration, alpha
         = 0.6)
#plt.hlines(np.arange(100,601,100),0,184,colors='black', alpha = 0.3, 
linestyles='dashed',)

plt.rc('font', family='Arial')
```

```
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.ylabel('Discharging Time (Seconds)',weight='bold')
plt.xlabel('Number of Cycles',weight='bold')
plt.title('Discharging Time vs. No.cycles for Reference Discharge
Period',weight='bold')
plt.show()



####plt.gca().xaxis.set_major_locator(ticker.MultipleLocator(15))


# ## Internal Resistence

# In[10]:


# internal resistence

df_RDC['internal_resistence'] = pd.Series()
for i in range(0,80):
    df_RDC['internal_resistence'][i] = (df_RDC['voltage'][0][0] - df_RDC['voltage'][i]
[0] )/df_RDC['current_avg'][i]



# In[11]:


# plot internal resistance against cycles
r = df_RDC['internal_resistence']

plt.figure(figsize=(8, 6))

plt.plot(r.index,r,alpha = 0.6)
plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.ylabel('Internal Resistance (Ohms)',weight='bold')
plt.xlabel('Number of Cycles',weight='bold')
plt.title('Internal Resistance vs. No.cycles for Reference Discharge
Period',weight='bold')
plt.show()


# In[12]:


# state of charge voltage-based on load period

# state of charge voltage based
df_RDC['soc'] = (df_RDC['voltage_avg'] - min(df_RDC['voltage_avg']))/
(max(df_RDC['voltage_avg']) - min(df_RDC['voltage_avg']))
soc = df_RDC['soc']
plt.figure(figsize=(8, 6))
```

```
plt.plot( soc, alpha = 0.6)
plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.ylabel('State of Charge (%)',weight='bold')
plt.xlabel('Number of Cycles',weight='bold')
plt.title('State of Charge vs. No.Cycles on Reference Discharge Period',weight='bold')
plt.show()


# In[13]:


# capacity on reference discharge


df_RDC['capacity'] = (df_RDC['current_avg'] *  df_RDC['duration'] )
capacity = df_RDC['capacity']
plt.figure(figsize=(8, 6))

plt.plot( capacity, alpha = 0.6)


plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.ylabel('Capacity',weight='bold')
plt.xlabel('Number of Cycles',weight='bold')
plt.title('Capacity vs. No.Cycles on Reference Discharge Period',weight='bold')
plt.show()


# In[14]:


# discharge energy
discharge_energy = list()
for i in range(len(df_RDC)):
    q =  (df_RDC['current_avg'][i] * df_RDC['duration'][i] * df_RDC['voltage_avg'][i])
    discharge_energy.append(q)

plt.figure(figsize=(8, 6))

plt.plot( discharge_energy, alpha = 0.6)

plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.ylabel('Discharged Energy',weight='bold')
plt.xlabel('Number of Cycles',weight='bold')
plt.title('Discharged Energy vs. No.Cycles on Reference Discharge 
Period',weight='bold')
```

```
plt.show()


# In[15]:



# discharge voltage range
discharge_range = list()
for i in range(0,80):
    v_r =  (max(df_RDC['voltage'][i]) - min(df_RDC['voltage'][i]))
    discharge_range.append(v_r)

plt.figure(figsize=(8, 6))

plt.plot( discharge_range, alpha = 0.6)

plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.ylabel('Discharged Voltage Range',weight='bold')
plt.xlabel('Number of Cycles',weight='bold')
plt.title('Discharged Voltage Range vs. No.Cycles on Reference Discharge
Period',weight='bold')
plt.show()


# # Define Failure Threshold

# In[17]:


# define failure threshold


cutoff = 0.25 * soc[0]

unhealthy = np.ma.masked_where(soc > cutoff, soc)
healthy = np.ma.masked_where(soc <= cutoff, soc)

# plot soc against cycles
plt.figure(figsize=(8, 6))

plt.plot(soc.index,healthy,soc.index,unhealthy, alpha = 0.6)
plt.ylabel('Cutoff State of Charge',fontsize = 12)
plt.xlabel('Number of Cycles',fontsize = 12)
plt.title('Cutoff State of Charge vs. No.cycles on Reference Discharge Period', c =
'black',fontsize = 14,weight='bold')
plt.hlines(np.arange(0.0,1,0.1),0,80,colors='black', alpha = 0.3,
linestyles='dashed',)

plt.text(-7,-0.5, 'https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-
repository/\nBrian Bole, Chetan Kulkarni, and Matthew Daigle, \n“Adaptation of an
Electrochemistry-based Li-Ion Battery Model to Account for Deterioration Observed
Under Randomized Use”, \nin the proceedings of the Annual Conference of the
Prognostics and Health Management Society, 2014',fontsize=7)
plt.show()
```

```
# scatter plot

plt.figure(figsize=(8, 6))

plt.plot(soc.index,healthy,'o', soc.index,unhealthy,'o', alpha = 0.6)
plt.hlines(0.25,0,80,colors='red', linestyles='dashed',)

plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.ylabel('State of Charge (%)',weight='bold')
plt.xlabel('Number of Cycles',weight='bold')
plt.title('Cutoff SOC vs. No.cycles for Reference Discharge Period',weight='bold')
plt.show()


# In[18]:


# find the cutoff cycle, the nearest cycle of the cutoff soc
cutoff_cycle = (np.abs(soc-cutoff)).argmin()
print('cutoff cycle is No.', cutoff_cycle)

# plot failure based on cutoff cycle


# In[19]:


# plot failure based on cutoff cycle


plt.figure(figsize=(8, 6))
plt.plot(soc[0:65].index,soc[0:65],'o',soc[65:80].index,soc[65:80],'o')

plt.vlines(65,0,1, colors='red',linestyles='dashed')
plt.xticks(list(plt.xticks()[0]) + [65])


plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.ylabel('State of Charge (%)',weight='bold')
plt.xlabel('Number of Cycles',weight='bold')
plt.title('Failure Threshold Cycle for Reference Discharge Period',weight='bold')
plt.show()


# # Cycle Estimation

# ## Prepare Estimation Dataframe

# In[23]:


# State of Health dataframe on reference discharge period
```

```
# soh
# soc
# discharge_energy
c = list()
v_ocv = list()
v_bat = list()
cycle = np.arange(1,81)
d = list()
t = list()
for i in range(0,80):
    c.append(df_RDC['current_avg'][i])
    v_ocv.append(df_RDC['voltage_avg'][0])
    v_bat.append(df_RDC['voltage_avg'][i])
    d.append(df_RDC['duration'][i])
    t.append(df_RDC['temperature_avg'][i])

features =
['current','voltage_ocv','voltage_bat','internal_resistance','duration','temperature','soc','dischar
soh_rdc =
pd.DataFrame([c,v_ocv,v_bat,r,d,t,soc,discharge_energy,discharge_range,capacity,cycle])
soh_rdc = np.transpose(soh_rdc)
soh_rdc.columns =
['current','voltage_ocv','voltage_bat','internal_resistance','duration','temperature','soc','dischar

np.random.seed(10)
threshold = np.random.rand(len(soh_rdc)) < 0.8
train = soh_rdc[threshold]
test = soh_rdc[~threshold]

X = train[features].astype(float)
Y = train['cycle'].astype(float)

X_test = test[features].astype(float)
Y_test = test['cycle'].astype(float)


# In[24]:


# select features
def minAIC_OLS(X,Y):
    variables = X.columns
    model = sm.OLS(Y,X[variables]).fit()
    while True:
        print(f'old model AIC：{model.aic}')
        maxp = np.max(model.pvalues)
        newvariables = variables[model.pvalues < maxp]
        removed = variables[model.pvalues == maxp].values
        print(f'consider a model with these variables removed:{removed}')
        newmodel = sm.OLS(Y,X[newvariables]).fit()
        print(f'new model AIC :{newmodel.aic}')
        if newmodel.aic < model.aic:
            model = newmodel
            variables = newvariables
        else:
            break
    return model, variables


# In[25]:
```

```
model_new, features_new = minAIC_OLS(X,Y)
model_new.summary()
print(features_new)


# ## Random Forest

# In[26]:


from statistics import mean
from sklearn import tree
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score


# # Iterate on Depth

# In[35]:


depth = [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
accuracy = list()
accuracy_modified = list()
mse = []
r2 = []
for d in depth:
    clf = DecisionTreeRegressor(random_state=0, criterion = "mse", splitter = "best",
max_depth = d)
    model = clf.fit(X[features_new], Y) # Use the training data to build
    y_test_pred = np.round(model.predict(X_test[features_new]))
    m = mean_squared_error(Y_test, y_test_pred)
    r = r2_score(Y_test, y_test_pred)

    mse.append(m)
    r2.append(r)
    print('depth:',d)
    print ("Test MSE on Reference Discharge Period ", m)
    print ("R2 on Reference Discharge Period ", r)
# cycle prediction accuracy
    y_pred = model.predict(soh_rdc[features_new].astype(float))
    y_pred = np.round(y_pred)

    soh_rdc['predict'] = y_pred

    correct = 0
    actual = list()
    pred = list()
    for i in range(len(soh_rdc)):
        if soh_rdc['predict'][i] == soh_rdc['cycle'][i]:
            correct += 1
        else:
            actual.append(soh_rdc['cycle'][i])
            pred.append(soh_rdc['predict'][i])

    acc = correct/80
    print('Accuracy for reference discharge cycle prediction before modified is:',acc)
    accuracy.append(acc)

# if prediction error is within one cycle, roughly considered as correct
```

```
    for i in range(len(soh_rdc)):
        if abs(soh_rdc['predict'][i] - soh_rdc['cycle'][i]) == 1:
            correct += 1

    acc_mod = correct/80
    print('Accuracy for cycle prediction after modified is:',acc_mod)
    accuracy_modified.append(acc_mod)
    print('-' * 50)


# ## From output parameters, the best model is random forest with depth = 9

# In[41]:



# best depth is 9
clf = DecisionTreeRegressor(random_state=0, criterion = "mse", splitter = "best",
max_depth = 9)
model = clf.fit(X[features_new], Y) # Use the training data to build
y_test_pred = np.round(model.predict(X_test[features_new]))
m = mean_squared_error(Y_test, y_test_pred)
rscore = r2_score(Y_test, y_test_pred)


print('depth:',9)
print ("Test MSE on Reference Discharge Period ", m)
print ("R2 on Reference Discharge Period ", rscore)

y_pred = model.predict(soh_rdc[features_new].astype(float))
y_pred = np.round(y_pred)

soh_rdc['predict'] = y_pred


# ## Compare Estimation with Actual Condition

# In[42]:


#actual data


plt.figure(figsize=(8, 6))
plt.plot(soc[0:65].index,soc[0:65],'o',label = 'Actual Health')
plt.plot(soc[65:80].index,soc[65:80],'o',label = 'Actual Failure')
plt.legend()

plt.vlines(65,0,1, colors='red',linestyles='dashed')

plt.xticks(list(plt.xticks()[0]) + [65])


plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.ylabel('State of Charge (%)',weight='bold')
plt.xlabel('Number of Cycles',weight='bold')
plt.title('Actual Health Condition on Reference Discharge Period',weight='bold')
```

```
plt.show()

# predicted data



plt.figure(figsize=(8, 6))

plt.plot(y_pred[0:65],soc[0:65],'o',label = 'Estimated Health', c = 'green')
plt.plot(y_pred[65:80],soc[65:80],'o',label = 'Estimated Failure', c = 'red')
plt.legend()

plt.vlines(65,0,1, colors='red',linestyles='dashed')
plt.xticks(list(plt.xticks()[0]) + [65])
plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.ylabel('State of Charge (%)',weight='bold')
plt.xlabel('Number of Cycles',weight='bold')
plt.title('Estimated Health Condition on Reference Discharge Period',weight='bold')
plt.show()

# compare actual and predicted


plt.figure(figsize=(8, 6))

plt.plot(soh_rdc['cycle'][0:65],y_pred[0:65],'o',label = 'Health', c = 'green')
plt.plot(soh_rdc['cycle'][65:80],y_pred[65:80],'o',label = 'Failure', c = 'red')
plt.legend()

plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.ylabel('Estimated',weight='bold')
plt.xlabel('Actual',weight='bold')
plt.title('Actual vs. Estimated on Reference Discharge Period',weight='bold')
plt.show()


# ## Cycle Prediciton Accuracy

# In[43]:


# cycle prediciton accuracy
correct = 0
actual = list()
pred = list()
for i in range(len(soh_rdc)):
    if soh_rdc['predict'][i] == soh_rdc['cycle'][i]:
        correct += 1
    else:
        actual.append(soh_rdc['cycle'][i])
        pred.append(soh_rdc['predict'][i])
```

```
accuracy = correct/80
print('Accuracy for cycle prediction before modified is:',accuracy)

# if prediction error is within one cycle, roughly considered as correct

for i in range(len(soh_rdc)):
    if abs(soh_rdc['predict'][i] - soh_rdc['cycle'][i]) == 1:
        correct += 1

accuracy = correct/80
print('Accuracy for cycle prediction after modified is:',accuracy)


# ## Application Model on Real-time Input

# In[44]:


np.random.seed(20)

sample_data = soh_rdc.sample(n = 10)
sample_data = sample_data.reset_index(drop = True)

sample_x = sample_data[features_new].astype(float)
sample_y = sample_data['cycle']

sample_y_pred = np.round(model.predict(sample_x))

import sys
from termcolor import colored, cprint

for i in range(len(sample_y_pred)):
    print('For the follwing input data on reference discharge period:')
    print(sample_x.iloc[i].to_string())
    print(colored('The estimated current cycle is at'), colored('No.','green'),
colored(int(sample_y_pred[i]),'green'))
    if sample_y_pred[i] < 65:
        print(colored('There are'), colored(65 - int(sample_y_pred[i]),'blue'),
colored('cycles left to reach failure threshold'))
    else:
        print(colored('There are'), colored(int(sample_y_pred[i]) - 65,'red'),
colored('cycles overused beyond the failure threshold'))

    print(colored('The actual current cycle is at'), colored('No.','green'),
colored(int(int(sample_y[i])),'green'))
    if sample_y[i] < 65:
        print(colored('There are actual'), colored(65 - int(sample_y[i]) ,'blue'),
colored('cycles left to reach failure threshold'))

    else:
        print(colored('There are actual'), colored(int(sample_y[i]) - 65,'red'),
colored('cycles overused beyond the failure threshold'))

    if sample_y_pred[i] == sample_y[i]:
        print(colored('Correct Estimation','yellow'))
    else:
        print(colored('Error Estimation', 'yellow'))

print('------------------------------------------------------------------------------------------')
```

```
# ### SOH on Pulsed Load Period
#

# In[ ]:



# # State of Health Analysis on Pulsed Load Period

# In[3]:


import pandas as pd
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import statsmodels.api as sm
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn import linear_model
from sklearn.neural_network import MLPRegressor
from mat4py import loadmat
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score


# In[4]:


data = loadmat('RW9.mat')


# In[5]:


#Pulling out all the data from raw dataset

data2=data['data']

step=data2['step']


# In[6]:


#Eight features
comment=pd.Series(step['comment'])
Type=pd.Series(step['type'])
current=pd.Series(step['current'])
time=pd.Series(step['time'])
relativeTime=pd.Series(step['relativeTime'])
voltage=pd.Series(step['voltage'])
temperature=pd.Series(step['temperature'])
date=pd.Series(step['date'])
#This df is our whole Dataframe
df=pd.DataFrame([comment,Type,date,current,time,relativeTime,voltage,temperature])
df=df.T
df.columns=['comment','Type','date','current','time','relativeTime','voltage','temperature']
#%%There are 15 different comment types
comment_type = []
for i in comment:
```

```
    comment_type.append(i)
comment_type = set(comment_type)
comment_type=list(comment_type)


# # Pulling Out Data for Pulsed Load period

# In[7]:


#pulling out data for the second period - pulsed period (pulse load(discharge) +
pulse load(rest) )

 #Type: Pulsed Load(PL)
 #Pulling out all the data related to the PL
df_PL = np.array(df[(df['comment'] == 'pulsed load (discharge)') | (df['comment'] ==
'pulsed load (rest)') ])
df_PL=pd.DataFrame(df_PL)
df_PL.columns=['comment','Type','date','current','time','relativeTime','voltage','temperature']


size_PL=[]
for i in range(len(df_PL)):
    size_PL.append(len(df_PL['relativeTime'][i]))
df_PL

#%Pulling out all observations of each cycle of Pulsed Load

all_comment = list()
all_Type = list()
all_date = list()

all_Type = [df_PL['Type'][j]  for j in range(len(df_PL)) for _ in
range(len(df_PL['voltage'][j])) ]
all_comment = [df_PL['comment'][j]  for j in range(len(df_PL)) for _ in
range(len(df_PL['voltage'][j])) ]
all_date = [df_PL['date'][j]  for j in range(len(df_PL)) for _ in
range(len(df_PL['voltage'][j])) ]
#all_comment.append(df2['comment'][i])

#columns we need to pull out for each list in each row i for i in range(len(df)):
all_current = list()
all_time = list()
all_relativetime = list()
all_voltage = list()
all_temperature = list()


for i in range(len(df_PL)):
    for j in range(len(df_PL['current'][i])):

        all_current.append(df_PL['current'][i][j])
        all_time.append(df_PL['time'][i][j])
        all_relativetime.append(df_PL['relativeTime'][i][j])
        all_temperature.append(df_PL['temperature'][i][j])
        #all_Type.append(df2['Type'][i])
        #all_date.append(df2['date'][i])
        all_voltage.append(df_PL['voltage'][i][j])
        #all_comment.append(df2['comment'][i])

pulsed_load_ds = np.transpose(np.array([all_comment,all_Type,all_date,
all_current,all_time, all_relativetime,all_voltage, all_temperature ]))
```

```
pulsed_load_df = pd.DataFrame(pulsed_load_ds)

pulsed_load_df.columns = 
['comment','Type','date','current','time','relativeTime','voltage','temperature']

pulsed_load_df  #It's related to all the cycles of the 'refrence discharge' by 
pulling out all observations respect to the second for each row


# # Average Some Features for Pulsed Load Period

# In[8]:


from statistics import mean
df_PL['current_avg'] = df_PL['current'].map(mean)
df_PL['time_avg'] = df_PL['time'].map(mean)
df_PL['relativeTime_avg'] = df_PL['relativeTime'].map(mean)
df_PL['voltage_avg'] = df_PL['voltage'].map(mean)
df_PL['temperature_avg'] = df_PL['temperature'].map(mean)


# # Features

# In[9]:


# discharging time
df_PL['duration'] = 0
for i in range(0,len(df_PL)):
    df_PL['duration'][i] = max(df_PL['relativeTime'][i]) - min(df_PL['relativeTime']
[i])


# In[10]:


# ploting discharing time against number of cycles

duration = df_PL['duration'].copy()
duration = [i for i in duration if i != 1199]

plt.figure(figsize=(8, 6))

plt.plot(duration, alpha
         = 0.6)

#plt.text(-10, -60, 'https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-
repository/\nBrian Bole, Chetan Kulkarni, and Matthew Daigle, \n“Adaptation of an 
Electrochemistry-based Li-Ion Battery Model to Account for Deterioration Observed 
Under Randomized Use”, \nin the proceedings of the Annual Conference of the 
Prognostics and Health Management Society, 2014',fontsize=7)
plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.ylabel('Discharging Time (Seconds)',weight='bold')
plt.xlabel('Number of Cycles',weight='bold')
plt.title('Discharging Time vs. No.cycles for Pulsed Load Period',weight='bold')
```

```
plt.show()


# ## Internal Resistence

# In[11]:


# internal resistence avg

df_PL['internal_resistence_avg'] = pd.Series()
for i in range(0,368,2):
    df_PL['internal_resistence_avg'][i] = (df_PL['voltage_avg'][i] -
df_PL['voltage_avg'][i+1])/df_PL['current_avg'][i+1]



# In[29]:


# plot internal resistance against cycles
r = df_PL['internal_resistence_avg'].replace(-np.inf, np.nan, regex=True).dropna()
r = r.reset_index(drop = True)

plt.figure(figsize=(8, 6))

plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.plot(r.index,r,alpha = 0.6)

plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.ylabel('Internal Resistance (Ohms)',weight='bold')
plt.xlabel('Number of Cycles',weight='bold')
plt.title('Internal Resistance vs. No.cycles for Pulsed Load Period',weight='bold')
plt.show()


# In[13]:


# state of charge voltage-based on load period

# state of charge voltage based
df_PL['soc'] = (df_PL['voltage_avg'] - min(df_PL['voltage_avg']))/
(max(df_PL['voltage_avg']) - min(df_PL['voltage_avg']))
# extract soc for only load phase
soc = []
for i in range(1,368,2):
    soc.append(df_PL['soc'][i])

plt.figure(figsize=(8, 6))

plt.plot( soc, alpha = 0.6)

plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
```

```
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.ylabel('State of Charge (%)',weight='bold')
plt.xlabel('Number of Cycles',weight='bold')
plt.title('State of Charge vs. No.Cycles on Pulsed Load Period',weight='bold')
plt.show()


# In[14]:


# discharge energy
discharge_energy = list()
for i in range(1,368, 2):
    q =  (df_PL['current_avg'][i] * df_PL['duration'][i] * df_PL['voltage_avg'][i])
    discharge_energy.append(q)

plt.figure(figsize=(8, 6))

plt.plot( discharge_energy, alpha = 0.6)
plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.ylabel('Discharged Energy',weight='bold')
plt.xlabel('Number of Cycles',weight='bold')
plt.title('Discharged Energy vs. No.Cycles on Pulsed Load Period',weight='bold')
plt.show()


# In[15]:



# discharge voltage range
discharge_range = list()
for i in range(1,368, 2):
    q =  (max(df_PL['voltage'][i]) - min(df_PL['voltage'][i]))
    discharge_range.append(q)

plt.figure(figsize=(8, 6))

plt.plot( discharge_range, alpha = 0.6)
plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.ylabel('Discharged Voltage Range',weight='bold')
plt.xlabel('Number of Cycles',weight='bold')
plt.title('Discharged Voltage Range vs. No.Cycles on Pulsed Load 
Period',weight='bold')
plt.show()


# In[16]:
```

```
# capacity on Pulsed Load Period
capacity = []
for i in range(1,368,2):

    cap = (df_PL['current_avg'][i] *  df_PL['duration'][i] )
    capacity.append(cap)
plt.figure(figsize=(8, 6))

plt.plot( capacity, alpha = 0.6)
plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.ylabel('Capacity',weight='bold')
plt.xlabel('Number of Cycles',weight='bold')
plt.title('Capacity vs. No.Cycles on Pulsed Load Period',weight='bold')
plt.show()


# # Define Failure Threshold

# In[17]:


# define failure threshold


cutoff = 2 * r[0]

unhealthy = np.ma.masked_where(r > cutoff, r)
healthy = np.ma.masked_where(r <= cutoff, r)

# plot internal resistance against cycles
plt.figure(figsize=(8, 6))

plt.plot(r.index,healthy,r.index,unhealthy, alpha = 0.6)
plt.ylabel('Cutoff Internal Resistance (Ohms)',fontsize = 12)
plt.xlabel('Number of Cycles',fontsize = 12)
plt.title('Cutoff Internal Resistance vs. No.cycles for Pulsed Load Period', c = 
'black',fontsize = 14,weight='bold')
plt.hlines(np.arange(0.15,0.40,0.05),0,184,colors='black', alpha = 0.3, 
linestyles='dashed',)

plt.text(-7,0.05, 'https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-
repository/\nBrian Bole, Chetan Kulkarni, and Matthew Daigle, \n“Adaptation of an 
Electrochemistry-based Li-Ion Battery Model to Account for Deterioration Observed 
Under Randomized Use”, \nin the proceedings of the Annual Conference of the 
Prognostics and Health Management Society, 2014',fontsize=7)
plt.show()


# scatter plot

plt.figure(figsize=(8, 6))

plt.plot(r.index,healthy,'o', r.index,unhealthy,'o', alpha = 0.6)

plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
```

```
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.ylabel('Internal Resistance (Ohms)',weight='bold')
plt.xlabel('Number of Cycles',weight='bold')
plt.hlines(0.34,0,184,colors='red', linestyles='dashed')

plt.title('Cutoff Internal Resistance vs. No.cycles for Pulsed Load
Period',weight='bold')
plt.show()


# In[18]:


# find the cutoff cycle, the nearest cycle of the cutoff internal resistance
cutoff_cycle = (np.abs(r-cutoff)).argmin()
print('cutoff cycle is No.', cutoff_cycle)


# In[19]:


# plot failure based on cutoff cycle

plt.figure(figsize=(8, 6))

plt.plot(r[0:150].index,r[0:150],'o',r[150:184].index,r[150:184],'o')
plt.vlines(150, 0.13, 0.4, colors='red',linestyles='dashed')


plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.ylabel('Internal Resistance (Ohms)',weight='bold')
plt.xlabel('Number of Cycles',weight='bold')
plt.title('Failure Threshold Cycle on Pulsed Load Period',weight='bold')
plt.show()


# # Cycle Estimation

# ## Prepare Estimation Dataframe

# In[30]:


# State of Health dataframe
# soh
# soc
# discharge_energy
c = list()
v_ocv = list()
v_bat = list()
cycle = np.arange(1,185)
d = list()
t = list()
for i in range(0,368,2):
    c.append(df_PL['current_avg'][i+1])
```

```
    v_ocv.append(df_PL['voltage_avg'][i])
    v_bat.append(df_PL['voltage_avg'][i+1])
    d.append(df_PL['duration'][i+1])
    t.append(df_PL['temperature_avg'][i+1])

features =
['current','voltage_ocv','voltage_bat','internal_resistance','duration','temperature','soc','dischar
soh_df =
pd.DataFrame([c,v_ocv,v_bat,r,d,t,soc,discharge_energy,discharge_range,capacity,cycle])
soh_df = np.transpose(soh_df)
soh_df.columns =
['current','voltage_ocv','voltage_bat','internal_resistance','duration','temperature','soc','dischar

np.random.seed(1)
threshold = np.random.rand(len(soh_df)) < 0.8
train = soh_df[threshold]
test = soh_df[~threshold]

X = train[features].astype(float)
Y = train['cycle'].astype(float)

X_test = test[features].astype(float)
Y_test = test['cycle'].astype(float)


# In[22]:


# select features
def minAIC_OLS(X,Y):
    variables = X.columns
    model = sm.OLS(Y,X[variables]).fit()
    while True:
        print(f'old model AIC：{model.aic}')
        maxp = np.max(model.pvalues)
        newvariables = variables[model.pvalues < maxp]
        removed = variables[model.pvalues == maxp].values
        print(f'consider a model with these variables removed:{removed}')
        newmodel = sm.OLS(Y,X[newvariables]).fit()
        print(f'new model AIC :{newmodel.aic}')
        if newmodel.aic < model.aic:
            model = newmodel
            variables = newvariables
        else:
            break
    return model, variables


# ## Variables Selection

# In[23]:


model_new, features_new = minAIC_OLS(X,Y)
model_new.summary()
print(features_new)


# ## Random Forest

# In[24]:
```

```
from statistics import mean
from sklearn import tree
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score


# In[26]:


depth = [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
accuracy = list()
accuracy_modified = list()
mse = []
r2 = []
for d in depth:
    clf = DecisionTreeRegressor(random_state=0, criterion = "mse", splitter = "best",
max_depth = d)
    model = clf.fit(X[features_new], Y) # Use the training data to build
    y_test_pred = np.round(model.predict(X_test[features_new]))
    m = mean_squared_error(Y_test, y_test_pred)
    rscore = r2_score(Y_test, y_test_pred)

    mse.append(m)
    r2.append(rscore)
    print('depth:' ,d)
    print ("Test MSE Pulsed Load ", m)
    print ("R2 Pulsed Load ", rscore)

    y_pred = model.predict(soh_df[features_new].astype(float))
    y_pred = np.round(y_pred)

    soh_df['predict'] = y_pred

# Accuracy Calculation
    correct = 0
    actual = list()
    pred = list()
    for i in range(len(soh_df)):
        if soh_df['predict'][i] == soh_df['cycle'][i]:
            correct += 1
        else:
            actual.append(soh_df['cycle'][i])
            pred.append(soh_df['predict'][i])

    acc = correct/184
    print('Accuracy for cycle prediction before modified is:',acc)
    accuracy.append(acc)

# if prediction error is within one cycle, roughly considered as correct

    for i in range(len(soh_df)):
        if abs(soh_df['predict'][i] - soh_df['cycle'][i]) == 1:
            correct += 1

    acc_mod = correct/184
    print('Accuracy for cycle prediction after modified is:',acc_mod)
    accuracy_modified.append(acc_mod)
    print('-' * 50)
```

```
# ## From output parameters, best model is random forest with depth = 11

# In[32]:


clf = DecisionTreeRegressor(random_state=0, criterion = "mse", splitter = "best",
max_depth = 11)
model = clf.fit(X[features_new], Y) # Use the training data to build
y_test_pred = np.round(model.predict(X_test[features_new]))
m = mean_squared_error(Y_test, y_test_pred)
rscore = r2_score(Y_test, y_test_pred)


print('depth:' ,11)
print ("Test MSE Pulsed Load ", m)
print ("R2 Pulsed Load ", rscore)

y_pred = model.predict(soh_df[features_new].astype(float))
y_pred = np.round(y_pred)

soh_df['predict'] = y_pred


# ## Compare with Actual Condition

# In[33]:


#Actual Data

plt.figure(figsize=(8, 6))

plt.plot(r[0:150].index,r[0:150],'o',label = 'Actual Health')
plt.plot(r[150:184].index,r[150:184],'o',label = 'Actual Failure')

plt.vlines(150,0.1,0.4, colors='red',linestyles='dashed')
plt.xticks(list(plt.xticks()[0]) + [150])
plt.legend()


plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.ylabel('Internal Resistance',weight='bold')
plt.xlabel('Number of Cycles',weight='bold')
plt.title('Actual Health Condition on Pulsed Load Period',weight='bold')
plt.show()


# Predicted Data
y_pred = model.predict(soh_df[features_new].astype(float))
y_pred = np.round(y_pred)



plt.figure(figsize=(8, 6))

plt.plot(y_pred[0:150],r[0:150],'o',label = 'Estimated Health', c = 'green')
plt.plot(y_pred[150:184],r[150:184],'o',label = 'Estimated Failure', c = 'red')
```

```
plt.legend()

plt.vlines(150,0.1,0.4, colors='red',linestyles='dashed')
plt.xticks(list(plt.xticks()[0]) + [150])


plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.ylabel('Internal Resistance',weight='bold')
plt.xlabel('Number of Cycles',weight='bold')
plt.title('Estimated Health Condition on Pulsed Load Period',weight='bold')
plt.show()

# Comparsion


plt.figure(figsize=(8, 6))

plt.plot(soh_df['cycle'][0:150],y_pred[0:150],'o',label = 'Health', c = 'green')
plt.plot(soh_df['cycle'][150:184],y_pred[150:184],'o',label = 'Failure', c = 'red')
plt.legend()


plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.ylabel('Estimated',weight='bold')
plt.xlabel('Actual',weight='bold')
plt.title('Actual vs. Estimated on Pulsed Load Period',weight='bold')
plt.show()


# ## Cycle Prediciton Accuracy

# In[34]:


# cycle prediciton accuracy
correct = 0
actual = list()
pred = list()
for i in range(len(soh_df)):
    if soh_df['predict'][i] == soh_df['cycle'][i]:
        correct += 1
    else:
        actual.append(soh_df['cycle'][i])
        pred.append(soh_df['predict'][i])

accuracy = correct/184
print('Accuracy for cycle prediction before modified is:',accuracy)

# if prediction error is within one cycle, roughly considered as correct

for i in range(len(soh_df)):
    if abs(soh_df['predict'][i] - soh_df['cycle'][i]) == 1:
```

```
        correct += 1

accuracy = correct/184
print('Accuracy for cycle prediction after modified is:',accuracy)


# ## Application Model on Real-time Input

# In[35]:


from termcolor import colored, cprint


# In[36]:


np.random.seed(7)

sample_data = soh_df.sample(n = 10)
sample_data = sample_data.reset_index(drop = True)

sample_x = sample_data[features_new].astype(float)
sample_y = sample_data['cycle']

sample_y_pred = np.round(model.predict(sample_x))

for i in range(len(sample_y_pred)):
    print('For the follwing input data on pulsed load period:')
    print(sample_x.iloc[i].to_string())
    print(colored('The estimated current cycle is at'), colored('No.','green'),
colored(int(sample_y_pred[i]),'green'))
    if sample_y_pred[i] < 150:
        print(colored('There are estimated'), colored(150 - int(sample_y_pred[i]),
'blue'), colored('cycles left to reach failure threshold'))
    else: print(colored('There are estimated'), colored(int(sample_y_pred[i]) -
150,'red'), colored('cycles overused beyond failure threshold'))

    print('The actual No. cycle is: ', sample_y[i])
    print(colored('The actual current cycle is at'), colored('No.','green'),
colored(int(sample_y[i]), 'green'))
    if int(sample_y[i]) < 150:
        print(colored('There are actual'), colored(150 - int(sample_y[i]), 'blue'),
colored('cycles left to reach failure threshold'))
    else:
        print(colored('There are actual'), colored(int(sample_y[i]) - 150, 'red'),
colored('cycles overused beyond failure threshold'))
    if sample_y_pred[i] == sample_y[i]:
        print(colored('Correct Estimation','yellow'))
    else:
        print(colored('Error Estimation', 'yellow'))

print('-------------------------------------------------------------------------------------------')


# ## 3.3 Battery Voltage Forecast

# In[ ]:


# In[1]:
```

```python
import pandas as pd
import numpy as np
from mat4py import loadmat
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.linear_model import LinearRegression
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.linear_model import Lasso
from sklearn import model_selection
from sklearn.model_selection import cross_val_score
from sklearn.metrics import r2_score
import statsmodels.api as sm
import tensorflow as tf
from sklearn.metrics import mean_squared_error


# In[2]:


data = loadmat('RW9.mat')
data2=data['data']

step=data2['step']

#Eight features
comment=pd.Series(step['comment'])
Type=pd.Series(step['type'])
current=pd.Series(step['current'])
time=pd.Series(step['time'])
relativeTime=pd.Series(step['relativeTime'])
voltage=pd.Series(step['voltage'])
temperature=pd.Series(step['temperature'])
date=pd.Series(step['date'])


# In[3]:


#%%This df is our whole Dataframe
df=pd.DataFrame([comment,Type,date,current,time,relativeTime,voltage,temperature])
df=df.T
df.columns=['comment','Type','date','current','time','relativeTime','voltage','temperature']


# In[4]:


#%%There are 15 different comment types
comment_type = []
for i in comment:
    comment_type.append(i)
comment_type = set(comment_type)
comment_type=list(comment_type)


# In[5]:
```

```
df_RW = np.array(df[(df['comment'] == 'rest (random walk)') | (df['comment'] ==
'discharge (random walk)') | (df['comment'] == 'charge (random walk)')])

df_RW=pd.DataFrame(df_RW)
df_RW.columns=['comment','Type','date','current','time','relativeTime','voltage','temperature']


# To prevent the out of memory stuff comes out, I select the df_RW with indexs $100 *
1^i$ where i $\in$ $[0, 1126]$.

# In[6]:


index = [i for i in range(1,len(df_RW), 100)]
df_RW_selected = df_RW[df_RW.index.isin(index)]


# In[7]:



for i in range(len(df_RW_selected)):
        if  isinstance(df_RW_selected.iloc[i,3], float) or
isinstance(df_RW_selected.iloc[i,3], int): #check for if current is float or int
            df_RW_selected.iloc[i,3] = [df_RW_selected.iloc[i,3]]
        if  isinstance(df_RW_selected.iloc[i,4], float) or
isinstance(df_RW_selected.iloc[i,4], int):  #check for if time is float or int
            df_RW_selected.iloc[i,4] = [df_RW_selected.iloc[i,4]]
        if  isinstance(df_RW_selected.iloc[i,5], float) or
isinstance(df_RW_selected.iloc[i,5], int): #check for if relativeTime is float
            df_RW_selected.iloc[i,5] = [df_RW_selected.iloc[i,5]]
        if  isinstance(df_RW_selected.iloc[i,6], float) or
isinstance(df_RW_selected.iloc[i,6], int):  #check for if voltage is float  or int
            df_RW_selected.iloc[i,6] = [df_RW_selected.iloc[i,6]]
        if  isinstance(df_RW_selected.iloc[i,7], float) or
isinstance(df_RW_selected.iloc[i,7], int):  #check for if temperature is float or int
            df_RW_selected.iloc[i,7] = [df_RW_selected.iloc[i,7]]
df_RW_selected.head()  #one thing to check if the comment is in the order in its
original data set (df)


# In[8]:


df_RW_selected = df_RW_selected.reset_index()


# ### Code for 3.3.2 Forecasting battery voltage on the random walk period
#
# Method:
# 1. train other variables(others than time)'s residuals on regression
# 2. try exponential smoothing on the residuals and time.

# In[9]:


random_walk_df_forcast = df_RW_selected.copy()
random_walk_df_forcast['date'] = pd.to_datetime(random_walk_df_forcast['date'],
format = '%d-%b-%Y %H:%M:%S', errors = 'ignore')
random_walk_df_forcast = random_walk_df_forcast[['date', 'Type', 'current', 'time',
'voltage', 'temperature']]
```

```
random_walk_df_forcast = random_walk_df_forcast.sort_values(by=['date']).copy()
random_walk_df_forcast.head()
date_index = random_walk_df_forcast.index

random_walk_df_forcast.head()


# In[10]:


#Taking average on each observation for the columns 'current', 'time', 'voltage',
'temperature':
import statisticsCode for

#current:

mean_current = list()
for l in random_walk_df_forcast['current']:
    mean_current.append(statistics.mean(l))

#time:

mean_time = list()
for l in random_walk_df_forcast['time']:
    mean_time.append(statistics.mean(l))

#voltage:

mean_voltage = list()
for l in random_walk_df_forcast['voltage']:
    mean_voltage.append(statistics.mean(l))

#temperature:

mean_temperature = list()
for l in random_walk_df_forcast['temperature']:
    mean_temperature.append(statistics.mean(l))

random_walk_df_forcast = random_walk_df_forcast.drop(['current','time', 'voltage',
'temperature'], axis = 1)

random_walk_df_forcast['mean_current'] = np.array(mean_current)
random_walk_df_forcast['mean_time(s)'] = np.array(mean_time)
random_walk_df_forcast['mean_voltage'] = np.array(mean_voltage)
random_walk_df_forcast['mean_temperature'] = np.array(mean_temperature)

#check for dimension match:
len(random_walk_df_forcast),len(mean_current), len(mean_time), len(mean_voltage),
len(mean_temperature)


# In[11]:


# Adding Day Month and Year in separate columns
d = pd.to_datetime(random_walk_df_forcast.index)
random_walk_df_forcast['Month'] = random_walk_df_forcast['date'].dt.month
random_walk_df_forcast['Year'] = random_walk_df_forcast['date'].dt.year
random_walk_df_forcast['Day'] = random_walk_df_forcast['date'].dt.day
random_walk_df_forcast['mean_current'] =
random_walk_df_forcast['mean_current'].astype(float)
random_walk_df_forcast['mean_time(s)'] =
```

```
random_walk_df_forcast['mean_time(s)'].astype(float)
random_walk_df_forcast['mean_voltage'] =
random_walk_df_forcast['mean_voltage'].astype(float)
random_walk_df_forcast['mean_temperature'] =
random_walk_df_forcast['mean_temperature'].astype(float)

random_walk_df_forcast = random_walk_df_forcast.reset_index(drop=True).copy()

random_walk_df_forcast['time(h)'] =
random_walk_df_forcast['mean_time(s)'].astype(float).div(3600)


# #### Explortary data analysis

# In[12]:


from importlib import reload
reload(plt)

plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.figure(figsize = (8, 6))
plt.ylabel('Mean Voltage',weight='bold')
plt.xlabel('Date',weight='bold')
plt.title('Voltage vs. Date on Random Walk Period',weight='bold')
plt.plot(random_walk_df_forcast['date'],random_walk_df_forcast['mean_voltage'] ,alpha=0.6)
plt.legend()
plt.show()


# We can see that from March to May(when current = 0), the mean_voltage's range is
much smaller than ranges in other months where current != 0.

# In[13]:


#Here, we choose the discharge and charge type and throw away data with the rest type:
random_walk_df_forcast = random_walk_df_forcast[(random_walk_df_forcast['Type'] ==
'D' ) | (random_walk_df_forcast['Type'] == 'C' ) ]
random_walk_df_forcast.describe()


# In[14]:


from importlib import reload
reload(plt)

plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.figure(figsize = (8, 6))
plt.ylabel('Mean Voltage',weight='bold')
plt.xlabel('Date',weight='bold')
```

```
plt.title('Voltage vs. Date on Random Walk Period ',weight='bold')
plt.plot(random_walk_df_forcast['date'],random_walk_df_forcast['mean_voltage'] ,alpha=0.6)
plt.legend()
plt.show()


# In[15]:


random_walk_df_forcast[['Month','mean_temperature']].groupby('Month').agg({'mean_temperature' :
{'max', 'min', 'mean'}})


# In[16]:


random_walk_df_forcast[['mean_voltage','mean_temperature']].corr()


# In[17]:


random_walk_df_forcast.query('mean_temperature > 40')
[['mean_voltage','mean_temperature']].corr()


# In[18]:


random_walk_df_forcast.query('mean_temperature < 40')
[['mean_voltage','mean_temperature']].corr()


# Do the same thing for the variable - mean_current as follows:

# In[19]:


random_walk_df_forcast[['Month','mean_current']].groupby('Month').agg({'mean_current' :
{'max', 'min', 'mean'}})


# In[20]:


random_walk_df_forcast.query('mean_current > 4')
[['mean_voltage','mean_current']].corr()


# In[21]:


random_walk_df_forcast.query('mean_current < - 4')
[['mean_voltage','mean_current']].corr()


# In[22]:


def high_temp(temp):
    if temp > 40:
        return 1
```

```
    return 0

def high_current(curr): #when current is greather than 4, we treat it as
high_current(disregard the discharge or charge period)
    if abs(curr) > 4:
        return 1
    return 0



# In[23]:


random_walk_df_forcast['high_temperature'] =
random_walk_df_forcast['mean_temperature'].apply(high_temp)
random_walk_df_forcast['high_current'] =
random_walk_df_forcast['mean_current'].apply(high_current)


# In[24]:


high_temp_obs = random_walk_df_forcast[random_walk_df_forcast['high_temperature'] ==
1]
plt.figure(figsize=(10,3))
plt.plot(high_temp_obs['date'], high_temp_obs['mean_temperature'], '.')
plt.xlabel('date')
plt.ylabel('temperature')
plt.title('high temperature(> 40)\'s observation')


# In[25]:


high_current_obs = random_walk_df_forcast[random_walk_df_forcast['high_current'] == 1]
plt.figure(figsize=(10,3))
plt.plot(high_current_obs['date'], high_current_obs['mean_current'], '.')
plt.xlabel('date')
plt.ylabel('current')
plt.title('high current(> 4)\'s observation')


# In[26]:


high_current_obs.groupby('Month')['mean_current'].mean()


# Quoted from the description of the data: "The RW operation is composed of a series
of charging or discharging current setpoints that are selected at random from the set
{-4.5A, -3.75A, -3A, -2.25A, -1.5A, -0.75A, 0.75A, 1.5A, 2.25A, 3A, 3.75A, 4.5A}.
Negative currents are associated with charging and positive currents indicate
discharging."
#
# We can see that the in the first two months(January and Feburay), the mean value of
the currents in the high_current_observations is approximating to 0 (0.003341 and
0.023557 showed above), which means that the current can reach up to high current
under discharge period and charge period. However, from the mean value of those in
the May and June(small dataset in June, not sinigifncant to show), we can see the
mean values( -0.965147 and -1.454667) are negative, which means that there is more
likely to reach up to high current in charge period than discharge period. This
finding may be caused by the battery degration respect to time. Further analysis is
```

```
needed.

# #### Let's create the month_bins for every two months:

# In[27]:


random_walk_df_forcast['month_bins'] = pd.cut(random_walk_df_forcast['Month'], bins =
3, labels = False)
random_walk_df_forcast = random_walk_df_forcast.drop(['Year',
'mean_time(s)','Day','time(h)'], axis=1)
Code for
random_walk_df_forcast = random_walk_df_forcast.rename(columns={"date": "ds",
"mean_voltage": "y"})


# In[28]:



#data spliting again:

train = random_walk_df_forcast[(random_walk_df_forcast['ds'] < '2014-05-01 00:00:00')]
valid = random_walk_df_forcast[(random_walk_df_forcast['ds'] >= '2014-05-01
00:00:00')]
train.head()
train.shape, valid.shape


# #### 1. Predict the voltage's residual using Holt-Winters (ExponentialSmoothing)
with  trend and seasonality.

# #### data cleaning

# In[29]:


df_Holt = random_walk_df_forcast.copy().drop(['Month','Type','month_bins'], axis =1)
df_Holt['ds'] = df_Holt['ds'].dt.date
df_Holt.head()


# In[30]:


df_Holt['ds'] = pd.to_datetime(df_Holt['ds'])


# In[31]:



train = df_Holt[(df_Holt['ds'] < '2014-05-01 00:00:00')]
valid = df_Holt[(df_Holt['ds'] >= '2014-05-01 00:00:00')]
train = sm.add_constant(train)
valid = sm.add_constant(valid)
X_train = train.drop(['ds', 'y'], axis = 1)
y_train = train['y']

X_valid =  valid.drop(['ds', 'y'], axis = 1)
y_valid = valid['y']
```

```
train.shape, valid.shape, X_train.shape, y_train.shape, X_valid.shape, y_valid.shape


# Firstly, we try to find the reduced significant variables and train the
variables(others than time)'s residuals on regression

# In[32]:


def minAIC_OLS_model(X,y):
    variables = X.columns
    model = sm.OLS(y,X[variables]).fit()
    while True:
        print(f"old model's aic: {model.aic}")
        maxp = np.max(model.pvalues)
        newvariables = variables[model.pvalues < maxp]
        removed_variable = variables[model.pvalues == maxp].values
        print(f'considering a model with these variables removed: {removed_variable}')
        newmodel = sm.OLS(y,X[newvariables]).fit()
        print(f"new model's aic: {newmodel.aic}")
        if newmodel.aic < model.aic:
            model = newmodel
            variables = newvariables
        else:
            break
    return model,variables

new_train_linear_model, linear_variables = minAIC_OLS_model(X_train, y_train)

new_linear_model = sm.OLS(y_valid,X_valid[linear_variables]).fit()
results = new_linear_model.summary()
results


# The model for predicting Voltage selected by minAIC_OLS is OLS and  all variables
in the model are statistically significant at the 95% level under the test data(so we
dropped variables except the variable of mean_current).

# In[33]:


X = pd.concat([X_train[linear_variables],X_valid[linear_variables]], axis = 0)
Y = pd.concat([y_train,y_valid], axis = 0)
model = sm.OLS(Y, X).fit()
model.summary()


# In[34]:


residuals = model.resid
residuals


# In[35]:


d = {"date" : df_Holt['ds'],"residual" : np.abs(residuals)}
# d = {"date" : df_Holt['ds'],"residual" : (residuals)}

df_HoltLinear = pd.DataFrame(data = d)
df_HoltLinear.head()
```

```
df_HoltLinear['date'] = df_HoltLinear['date'].dt.date
df_HoltLinear.head()


# In[38]:


ts = df_HoltLinear.set_index('date') #dataframe to time series


# To see if it is stationary time series:

# In[39]:


from importlib import reload
reload(plt)

plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.figure(figsize = (8, 6))
plt.ylabel('Residuals(returned by the best OLS model)',weight='bold')
plt.xlabel('Date',weight='bold')
plt.title('Time Series Residuals of Voltage',weight='bold')
plt.plot(ts ,alpha=0.6)
plt.legend()
plt.show()


# Check ACF and PACF:

# In[40]:


#Check ACF and PACF:
from matplotlib import pyplot
from statsmodels.graphics.tsaplots import plot_acf

series = ts
plot_acf(series)
pyplot.show()


# The plot above is good since there is no significant correlation among the elements
of a time series.

# In[41]:


from statsmodels.graphics.tsaplots import plot_pacf
plot_pacf(series)
pyplot.show()


# In[42]:


# Split the data into training and test
```

```
import datetime
train = df_HoltLinear[(df_HoltLinear['date'] < datetime.date(2014, 5, 1))]
valid = df_HoltLinear[(df_HoltLinear['date']  >=  datetime.date(2014, 5, 1))]

train.shape, valid.shape


# In[43]:


## Holt Double Exponential Linear
from statsmodels.tsa.holtwinters import Holt, ExponentialSmoothing
n = len(valid)
train_voltage_residual = train['residual']
valid_voltage_residual = valid['residual']


# In[ ]:



#Hyperparameter tuning
trend_list = ['add', 'mul']
seasonal_list = ['add', 'mul']
smoothing_level_list = [0,0.3, 0.5,0.8, 1.0]
smoothing_trend_list = [0,0.3, 0.5,0.8, 1.0]
damping_trend_list = [0,0.3, 0.5,0.8, 1.0]
seaonsal_period_list = [2, 7, 10, 14]
smoothing_seasonal_list = [0,0.3, 0.5,0.8, 1.0]
best_parameters = {'trend:' : None, 'seasonal:' : None, 'smoothing_level:' : None,
'smoothing_trend:' : None, 'damping_trend:' : None, 'seasonal_periods:': None,
'smoothing_seasonal:' : None}

mse_valid = float('inf')
for trend in trend_list:
    for seasonal in seasonal_list:
        for sl in smoothing_level_list:
            for st in smoothing_trend_list:
                for seasonal_period in seaonsal_period_list:
                    for dt in damping_trend_list:
                        for ss in smoothing_seasonal_list:
                            fit_Holt = ExponentialSmoothing(train_voltage_residual,
trend = trend, seasonal = seasonal, seasonal_periods =
seasonal_period).fit(damping_trend = dt, smoothing_seasonal = ss,smoothing_level =
sl,smoothing_trend = st)
                            fore_Holt = fit_Holt.forecast(valid.shape[0])
                            if np.sum(np.isnan(fore_Holt)) == 0:

                                if mean_squared_error(valid_voltage_residual,
fore_Holt) < mse_valid:
                                    best_parameters['trend:'] =  trend
                                    best_parameters['seasonal:'] = seasonal
                                    best_parameters['smoothing_level:'] = sl
                                    best_parameters['smoothing_trend:'] = st
                                    best_parameters['seasonal_periods:'] =
seasonal_period
                                    best_parameters['damping_trend:'] = dt
                                    best_parameters['smoothing_seasonal:'] = ss

                                    mse_valid =
mean_squared_error(valid_voltage_residual, fore_Holt)
```

```
# We did hyperparameter tuning on the following parameters: trend type = 'add',
seasonal type = 'mul', smoothing level = 0.5,
# smoothing trend = 0, damping trend = 0, seasonal periods = 10 and smoothing
seasonal = 0.

# In[46]:


# voltage_residual = df_HoltLinear['residual']
fit_Holt = ExponentialSmoothing(train_voltage_residual, trend = 'add', seasonal =
'mul', seasonal_periods = 10).fit(damping_trend = 0, smoothing_level = 0.5,
smoothing_seasonal = 0, remove_bias = True, smoothing_trend = 0)

fore_Holt = fit_Holt.forecast(valid.shape[0])


# Plot the fit, forecasts, and original df_HoltLinear data.

# In[47]:


plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.figure(figsize = (8, 6))
plt.scatter(range(len(df_HoltLinear)), df_HoltLinear['residual'], marker='.',
color='black', label = 'original data') #original data
plt.plot(range(len(train_voltage_residual)), fit_Holt.fittedvalues, color='red',
label = 'fitted values') #fitted value in training data
fore_Holt.plot(color='blue', label ='forecasting values') #forecasted value on test
plt.xlabel('Cycle', weight='bold')
plt.ylabel('Residuals(returned by the best OLS model)',weight='bold')
plt.title("Forecasting of Voltage's Residuals on Validation" ,weight='bold')
# plt.ylim((0,.5))
plt.legend()
plt.show()
sum_square_error = fit_Holt.sse
sum_square_error


# In[48]:


from sklearn.metrics import mean_squared_error
mean_squared_error(valid_voltage_residual, fore_Holt)


# #### Forecasting for the next 100 cycles:

# In[49]:


fore_Holt_next100 = fit_Holt.forecast(valid.shape[0] + 100)

plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
```

```
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.figure(figsize = (8, 6))

plt.scatter(range(len(df_HoltLinear)), df_HoltLinear['residual'], marker='.',
color='black', label = 'original data') #original data
plt.plot(range(len(train_voltage_residual)), fit_Holt.fittedvalues, color='red',
label = 'fitted values') #fitted value in training data
fore_Holt_next100.plot(color='blue', label = 'forecasting values(include validation
set)') #forecasted value on test

plt.xlabel('Cycle', weight='bold')
plt.ylabel('Residuals(returned by the best OLS model)',weight='bold')
plt.title("Forecasting of Voltage's Residuals on  next 100 cycles",weight='bold')

plt.legend()
plt.show()


# Therefore, according to the mean current's information in a day in future, we can
use the OLS model done before(as the model of the training set above) to get fitted
value by using the formula( fitted_value =  β0+β1  * mean_current) and then add the
fitted_value on the residual returned by the forecasting model above, and so get
prediction on the volatge value(fitted_value +- residual) in some future days!

#
# ### Code for 3.3.3 Forecasting battery voltage on the reference discharge period
#
# Method:
# 1. OLS with Exponential Smoothing method;
# 2. Holt's Linear Trend / Exponential Trend / Damped Trend models;
# 3. Prophet Forecasting.

# In[755]:


reference_discharge_df_forcast = df.copy()
reference_discharge_df_forcast =  reference_discharge_df_forcast
[reference_discharge_df_forcast['comment'] == 'reference discharge']
reference_discharge_df_forcast['date'] =
pd.to_datetime(reference_discharge_df_forcast['date'], format = '%d-%b-%Y %H:%M:%S',
errors = 'ignore')
reference_discharge_df_forcast = reference_discharge_df_forcast[['date',  'current',
'time', 'voltage', 'temperature']]

reference_discharge_df_forcast =
reference_discharge_df_forcast.sort_values(by=['date']).copy()
reference_discharge_df_forcast.head()
date_index = reference_discharge_df_forcast.index


# Predict durations on each clycle, except prediction on voltage.

# In[756]:


#Taking average on each observation for the columns 'current', 'time', 'voltage',
'temperature':
import statistics

#current:
```

```
mean_current = list()
for l in reference_discharge_df_forcast['current']:
    mean_current.append(statistics.mean(l))

#time:

mean_time = list()
for l in reference_discharge_df_forcast['time']:
    mean_time.append(statistics.mean(l))

#voltage:

mean_voltage = list()
for l in reference_discharge_df_forcast['voltage']:
    mean_voltage.append(statistics.mean(l))

#temperature:

mean_temperature = list()
for l in reference_discharge_df_forcast['temperature']:
    mean_temperature.append(statistics.mean(l))

reference_discharge_df_forcast =
reference_discharge_df_forcast.drop(['current','time', 'voltage', 'temperature'],
axis = 1)

reference_discharge_df_forcast['mean_current'] = np.array(mean_current)
reference_discharge_df_forcast['mean_time(s)'] = np.array(mean_time)
reference_discharge_df_forcast['mean_voltage'] = np.array(mean_voltage)
reference_discharge_df_forcast['mean_temperature'] = np.array(mean_temperature)

#check for dimension match:
len(reference_discharge_df_forcast),len(mean_current), len(mean_time),
len(mean_voltage), len(mean_temperature)


# In[758]:



# Adding Day Month and Year in separate columns
d = pd.to_datetime(reference_discharge_df_forcast.index)
reference_discharge_df_forcast['Month'] =
reference_discharge_df_forcast['date'].dt.month

reference_discharge_df_forcast['mean_current'] =
reference_discharge_df_forcast['mean_current'].astype(float)

reference_discharge_df_forcast['mean_voltage'] =
reference_discharge_df_forcast['mean_voltage'].astype(float)
reference_discharge_df_forcast['mean_temperature'] =
reference_discharge_df_forcast['mean_temperature'].astype(float)

reference_discharge_df_forcast =
reference_discharge_df_forcast.reset_index(drop=True).copy()


# In[759]:


from importlib import reload
reload(plt)
```

```
plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.figure(figsize = (8, 6))
plt.ylabel('Mean Voltage',weight='bold')
plt.xlabel('Date',weight='bold')
plt.title('Voltage vs. Date on Reference Discharge Period ',weight='bold')
plt.plot(reference_discharge_df_forcast['date'],
reference_discharge_df_forcast['mean_voltage'], alpha=0.6)
plt.legend()
plt.show()


# In[760]:


reference_discharge_df_forcast[['Month','mean_temperature']].groupby('Month').agg({'mean_temperature
{'max', 'min', 'mean'}})


# In[765]:


df_Holt_refer = reference_discharge_df_forcast.copy().drop(['Month'], axis = 1 )
df_Holt_refer = df_Holt_refer.rename(columns={'date': 'ds', 'mean_voltage': 'y'})

df_Holt_refer['ds'] = df_Holt_refer['ds'].dt.date


df_Holt_refer['ds'] = pd.to_datetime(df_Holt_refer['ds'])
df_Holt_refer.head()


# In[766]:



train = df_Holt_refer[(df_Holt_refer['ds'] < '2014-05-01 00:00:00')]
valid = df_Holt_refer[(df_Holt_refer['ds'] >= '2014-05-01 00:00:00')]
train = sm.add_constant(train)
valid = sm.add_constant(valid)
X_train = train.drop(['ds', 'y'], axis = 1)
y_train = train['y']

X_valid =  valid.drop(['ds', 'y'], axis = 1)
y_valid = valid['y']

train.shape, valid.shape, X_train.shape, y_train.shape, X_valid.shape, y_valid.shape


# In[767]:


# now call the minAIC function on our predictors and response variables
new_train_linear_model, linear_variables = minAIC_OLS_model(X_train, y_train)

# Now fit the linear model on the new predictors and the test data
new_linear_model = sm.OLS(y_valid,X_valid[linear_variables]).fit()
```

```
results = new_linear_model.summary()
results


# In[768]:


X = pd.concat([X_train[linear_variables],X_valid[linear_variables]], axis = 0)
Y = pd.concat([y_train,y_valid], axis = 0)
model = sm.OLS(Y, X).fit()
model.summary()


# In[770]:


residuals = model.resid
d = {"date" : df_Holt_refer['ds'],"residual" : np.abs(residuals)}
df_Holt_refer_Linear = pd.DataFrame(data = d)
df_Holt_refer_Linear.head()
df_Holt_refer_Linear['date'] = df_Holt_refer_Linear['date'].dt.date


# In[772]:


ts_ref = df_Holt_refer_Linear.set_index('date') #dataframe to time series


# In[773]:


#To see if it is stationary time series::

from importlib import reload
reload(plt)

plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.figure(figsize = (8, 6))
plt.ylabel('Residuals(returned by the best OLS model)',weight='bold')
plt.xlabel('Date',weight='bold')
plt.title('Time Series Residuals of Voltage',weight='bold')
plt.plot(ts_ref ,alpha=0.6)
plt.legend()


# In[774]:


#Check ACF and PACF:
from matplotlib import pyplot
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

series = ts_ref
plot_acf(series)
plot_pacf(series)
pyplot.show()
```

```
# In[775]:


# Split the data into training and test
train = df_Holt_refer_Linear[(df_Holt_refer_Linear['date'] < datetime.date(2014, 5,
1))]
valid = df_Holt_refer_Linear[(df_Holt_refer_Linear['date']  >=  datetime.date(2014,
5, 1))]

train.shape, valid.shape


# #### 1. Predict the voltage's residual using Holt-Winters (ExponentialSmoothing)
with  trend and seasonality.

# In[776]:


## Holt Double ExponentialSmoothing on the residual returned by the OLS:
from statsmodels.tsa.holtwinters import Holt, ExponentialSmoothing
n = len(valid)
train_voltage_residual = train['residual']
valid_voltage_residual = valid['residual']


# In[777]:




# voltage_residual = df_HoltLinear['residual']
# fit_Holt = ExponentialSmoothing(train_voltage_residual, trend =
best_parameters['trend:'], seasonal = best_parameters['seasonal:'], seasonal_periods
= best_parameters['seasonal_periods:']).fit(damping_trend =
best_parameters['damping_trend:'], smoothing_level =
best_parameters['smoothing_level:'], smoothing_seasonal =
best_parameters['smoothing_seasonal:'], remove_bias = True, smoothing_trend =
best_parameters['smoothing_trend:']).fit()

fit_Holt = ExponentialSmoothing(train_voltage_residual, trend = 'additive',seasonal =
"additive",seasonal_periods =20).fit()
fore_Holt = fit_Holt.forecast(len(valid_voltage_residual))


# Plot the fit, forecasts, and original data.

# In[778]:


plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.figure(figsize = (8, 6))
plt.scatter(range(len(df_Holt_refer_Linear)), df_Holt_refer_Linear['residual'],
marker='.', color='black', label = 'original data') #original data
plt.plot(range(len(train_voltage_residual)), fit_Holt.fittedvalues, color='red',
label = 'fitted values') #fitted value in training data
```

```
fore_Holt.plot(color='blue', label ='forecasting values') #forecasted value on test
plt.xlabel('Cycle', weight='bold')
plt.ylabel('Residuals(returned by the best OLS model)',weight='bold')
plt.title("Forecasting of Voltage's Residuals on Validation" ,weight='bold')
# plt.ylim((0,.5))
plt.legend()
plt.show()
sum_square_error = fit_Holt.sse
sum_square_error


# In[779]:


from sklearn.metrics import mean_squared_error
mean_squared_error(valid_voltage_residual, fore_Holt)


# Final Prediction for future 80 cycles:

# In[780]:


fore_Holt_next80 = fit_Holt.forecast(valid.shape[0] + 80)

plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.figure(figsize = (8, 6))

plt.scatter(range(len(df_Holt_refer_Linear)), df_Holt_refer_Linear['residual'],
marker='.', color='black', label = 'original data') #original data
plt.plot(range(len(train_voltage_residual)), fit_Holt.fittedvalues, color='red',label
= 'fitted values') #fitted value in training data
fore_Holt_next80.plot(color='blue', label = 'forecasting values(include validation
set)') #forecasted value on test

plt.xlabel('Cycle', weight='bold')
plt.ylabel('Residuals(returned by the best OLS model)',weight='bold')
plt.title("Forecasting of Voltage's Residuals on  next 80 cycles",weight='bold')

plt.legend()
plt.show()


# Therefore, according to the mean temperature's information in a day in future, we
can use the OLS model done before(as the model of the training set above) to get
fitted value by using the formula( fitted_value = β0+β1 * mean_current) and then add
the fitted_value on the residual returned by the forecasting model above, and so get
prediction on the volatge value(fitted_value +- residual) with reference discharge
data in some future days!

# #### 2. Predict the voltage  using Holt's Linear Trend / Exponential Trend / Damped
Trend models’ result:.

# In[781]:


from statsmodels.tsa.statespace.sarimax import SARIMAX
```

```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.metrics import mean_squared_error
from statsmodels.tools.eval_measures import rmse
import warnings


# In[783]:


RD_holt_linear = reference_discharge_df_forcast.copy()


# In[784]:


RD_holt_linear.date = pd.to_datetime(RD_holt_linear.date)
RD_holt_linear = RD_holt_linear.set_index("date")


# In[785]:


ax = RD_holt_linear['mean_voltage'].plot(figsize = (16,5), title = "mean_voltage
respect to time")
ax.set(xlabel='date', ylabel='Mean_Voltage');


# When we look at plot we can sey there is a trend in data.
#
# Therefore, we chose to use the Holt’s linear trend method and Damped trend methods:
#

# In[535]:


RD_holt_linear


# #### 1) Holt’s linear trend

# It's suitable for time series data with a trend component but without a seasonal
component.

# In[786]:


from statsmodels.tsa.api import Holt


# In[787]:


RD_holt_linear = RD_holt_linear['mean_voltage']
RD_holt_linear.head()


# In[788]:


n_past = 58 # number of past cycles we may use to predict future cycles' voltage
```

```
n_predict = 22 # number of cycle we want to predict

train = RD_holt_linear.head(n_past)
valid = RD_holt_linear.tail(n_predict)

Holt_valid = valid
train.shape, valid.shape


# Now, we find the best smoothing_level and smoothing_slope by considering the
smallest mean square error on validation set:

# In[539]:


from sklearn.metrics import mean_squared_error
import math


best_smoothing_level, best_smoothing_slope = None, None
valid.index = np.arange(n_past,n_past + len(valid))
mse1 = float("inf")

for smoothing_level in np.linspace(0,1,11):
    for smoothing_slope in np.linspace(0,1,11):
            fit1 = Holt(train).fit(smoothing_level, smoothing_slope, optimized=False)
            fore1 = fit1.forecast(n_predict).rename("Holt's linear trend")
            if math.sqrt(mean_squared_error(fore1, valid)) < mse1:
                mse1 = math.sqrt(mean_squared_error(fore1, valid))
                best_smoothing_level = smoothing_level
                best_smoothing_slope = smoothing_slope




# In[540]:


best_smoothing_level, best_smoothing_slope, mse1


# In[541]:


fit1 = Holt(train).fit(best_smoothing_level, best_smoothing_slope, optimized=False)
fore1 = fit1.forecast(22).rename("Holt's linear trend")


# Now plot the forecast, fitted values, and orignal sales data. We use

# In[542]:


n = len(RD_holt_linear)
plt.scatter(range(n), RD_holt_linear, marker='.', color='black', label = 'original
data')
plt.plot(range(len(train)), fit1.fittedvalues, color='red', label = 'fitted value on
training')
plt.xlabel('cycle')
plt.ylabel('mean_voltage')
fore1.plot(color = 'blue', marker = 'o', label = 'forecasting on validation')
plt.title('Holt’s Linear trend')
```

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


# In[543]:


plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.figure(figsize = (8, 6))
plt.scatter(range(n), RD_holt_linear, marker='.', color='black', label = 'original
data')
plt.plot(range(len(train)), fit1.fittedvalues, color='red', label = 'fitted value on
training')
fore1.plot(color='blue', marker = 'o', label ='forecasting values') #forecasted value
on test
plt.xlabel('Cycle', weight='bold')
plt.ylabel('Mean Voltage',weight='bold')
plt.title("Forecasting of Holt’s Linear trend" ,weight='bold')
plt.legend()
plt.show()


# Check for mean square error between the training set and our fitted values:

# In[544]:


mse_fittedvalue1 =  math.sqrt(mean_squared_error(fit1.fittedvalues, train))
mse_fittedvalue1


# Check for mean square error between the validation set and our forecasting data:

# In[545]:


mse1 = math.sqrt(mean_squared_error(fore1, valid))
print('On validation set, the Mean Squared Error of Holt''s Linear trend is
{}'.format(mse1))


# #### 2) Holt’s Exponential trend :

# Similarly, we find the best smoothing_level and smoothing_slope by considering the
smallest mean square error:

# In[546]:


best_smoothing_level2, best_smoothing_slope2 = None, None
valid.index = np.arange(n_past,n_past + len(valid))
mse2 = float("inf")

for smoothing_level in np.linspace(0,1,11):
    for smoothing_slope in np.linspace(0,1,11):
            fit2 = Holt(train, exponential=True).fit(smoothing_level,
smoothing_slope, optimized=False)
```

```
            fore2 = fit2.forecast(n_predict).rename("Holt's Exponential trend")
            if math.sqrt(mean_squared_error(fore2, valid)) < mse2:
                mse2 = math.sqrt(mean_squared_error(fore2, valid))
                best_smoothing_level2 = smoothing_level
                best_smoothing_slope2 = smoothing_slope


# In[547]:


best_smoothing_level2, best_smoothing_slope2, mse2


# In[548]:


fit2 = Holt(train, exponential=True).fit(best_smoothing_level2,
best_smoothing_slope2, optimized=False)
fore2 = fit2.forecast(n_predict).rename("Exponential trend")


# Now plot the forecast, fitted values, and orignal sales data. We use

# In[549]:


n = len(RD_holt_linear)

plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.figure(figsize = (8, 6))
plt.scatter(range(n), RD_holt_linear, marker='.', color='black', label = 'original
data')
plt.plot(range(len(train)), fit2.fittedvalues, color='red', label = 'fitted value on
training')
fore2.plot(color='blue', marker = 'o', label ='forecasting values') #forecasted value
on test
plt.xlabel('Cycle', weight='bold')
plt.ylabel('Mean Voltage',weight='bold')
plt.title("Forecasting of Holt’s Exponential trend" ,weight='bold')
plt.legend()
plt.show()


# In[550]:


fit2 = Holt(train, exponential=True).fit(best_smoothing_level2,
best_smoothing_slope2, optimized=False)
fore2 = fit2.forecast(n_predict).rename("Exponential trend")


# Check for mean square error between the training set and our fitted values:

# In[551]:


mse_fittedvalue2 = math.sqrt(mean_squared_error(fit2.fittedvalues, train))
```

```
mse_fittedvalue2


# Check for mean square error between the validation set and our forecasting data:

# In[552]:


mse2 = math.sqrt(mean_squared_error(fore2, valid))
print('On validation set, the Mean Squared Error of Holt''s Exponential trend
{}'.format((mse2)))


# #### 3) Damped trend method
#

# We also add the damping trend in the model:

# In[553]:


best_smoothing_level3, best_smoothing_slope3 = None, None
valid.index = np.arange(55,55 + len(valid))
mse3 = float("inf")

for smoothing_level in np.linspace(0,1,11):
    for smoothing_slope in np.linspace(0,1,11):
        for damping_trend in np.linspace(0,1,11):
            fit3 = Holt(train, damped_trend=True,
initialization_method="estimated").fit(smoothing_level = smoothing_level,
smoothing_slope = smoothing_slope)
            fore3 = fit3.forecast(n_predict).rename("Holt's Damped trend")
            if math.sqrt(mean_squared_error(fore3, valid)) < mse3:
                mse3 = math.sqrt(mean_squared_error(fore2, valid))
                best_smoothing_level3 = smoothing_level
                best_smoothing_slope3 = smoothing_slope


# In[554]:


best_smoothing_level3, best_smoothing_slope3, mse3


# In[555]:


fit3 = Holt(train, damped_trend=True,
initialization_method="estimated").fit(smoothing_level = best_smoothing_level3,
smoothing_slope = best_smoothing_slope3)
fore3 = fit3.forecast(n_predict).rename("Holt's Damped trend")


# Now plot the forecast, fitted values, and orignal sales data. We use

# In[556]:


n = len(RD_holt_linear)

plt.rc('font', family='Arial')
plt.rc('font', size= 16)
```

```
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.figure(figsize = (8, 6))
plt.scatter(range(n), RD_holt_linear, marker='.', color='black', label = 'original
data')
plt.plot(range(len(train)), fit3.fittedvalues, color='red', marker = '*', label =
'fitted value on training')
fore3.plot(color='blue', marker = 'o', label ='forecasting values') #forecasted value
on test
plt.xlabel('Cycle', weight='bold')
plt.ylabel('Mean Voltage',weight='bold')
plt.title("Forecasting of Holt’s Damped trend" ,weight='bold')
plt.legend()
plt.show()


# Check for mean square error between the training set and our fitted values:

# In[557]:


mse_fittedvalue3 = math.sqrt(mean_squared_error(fit3.fittedvalues, train))
mse_fittedvalue3


# Check for mean square error between the validation set and our forecasting data:

# In[558]:


mse3 = math.sqrt(mean_squared_error(fore3, valid))
print('On validation set, the Mean Squared Error of Holt''s Exponential trend
{}'.format((mse3)))


# In[559]:


Damped_trend_mse_error = mse3


# Compare Holt'Linear Trend, Holt's  Exponential Trend and Holt's Damped Trend on
AIC, BIC, AICC:

# In[560]:


fit1.aic, fit2.aic, fit3.aic


# In[561]:


fit1.bic, fit2.bic, fit3.bic


# In[562]:


fit1.aicc, fit2.aicc, fit3.aicc
```

```
# In[563]:


fit6 = Holt(RD_holt_linear, damped_trend=True,
initialization_method="estimated").fit(smoothing_level = best_smoothing_level3,
smoothing_slope = best_smoothing_slope3)
fore6= fit6.forecast(80).rename("Holt's Damped trend")
n = len(RD_holt_linear)
plt.scatter(range(n), RD_holt_linear, marker='.', color='black', label = 'original
data')
plt.plot(range(len(RD_holt_linear)), fit6.fittedvalues, color='red', marker = '*',
label = 'fitted value on training')
plt.xlabel('cycle')
plt.ylabel('mean_voltage')
fore6.plot(color = 'blue', marker = 'o', label = 'forecasting on next 80 cycles')
plt.title('Holt’s Damped trend')
plt.legend()
plt.show()

mse_fittedvalue6 = math.sqrt(mean_squared_error(fit6.fittedvalues, RD_holt_linear))
mse_fittedvalue6


# In[564]:


fit5= Holt(RD_holt_linear, exponential=True).fit(best_smoothing_level2,
best_smoothing_slope2, optimized=False)
fore5 = fit5.forecast(80).rename("Exponential trend")
n = len(RD_holt_linear)
plt.scatter(range(n), RD_holt_linear, marker='.', color='black', label = 'original
data')
plt.plot(range(len(RD_holt_linear)), fit5.fittedvalues, color='red', marker = '*',
label = 'fitted value on training')
plt.xlabel('cycle')
plt.ylabel('mean_voltage')
fore5.plot(color = 'blue', marker = 'o', label = 'forecasting on next 80 cycles')
plt.title('Holt’s Exponential trend')
plt.legend()
plt.show()

mse_fittedvalue5 = math.sqrt(mean_squared_error(fit5.fittedvalues, RD_holt_linear))
mse_fittedvalue5


# In[565]:


fit4 = Holt(RD_holt_linear).fit(best_smoothing_level, best_smoothing_slope,
optimized=False)
fore4 = fit1.forecast(80).rename("Holt's linear trend")
n = len(RD_holt_linear)
plt.scatter(range(n), RD_holt_linear, marker='.', color='black', label = 'original
data')
plt.plot(range(len(RD_holt_linear)), fit4.fittedvalues, color='red', marker = '*',
label = 'fitted value on training')
plt.xlabel('cycle')
plt.ylabel('mean_voltage')
fore4.plot(color = 'blue', marker = 'o', label = 'forecasting on next 80 cycles')
plt.title('Holt’s linear trend')
```

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

mse_fittedvalue4 = math.sqrt(mean_squared_error(fit4.fittedvalues, RD_holt_linear))
mse_fittedvalue4


# In[ ]:


n = len(RD_holt_linear)

plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.figure(figsize = (8, 6))
plt.scatter(range(n), RD_holt_linear, marker='.', color='black', label = 'original
data')
plt.plot(range(len(train)), fit3.fittedvalues, color='red', marker = '*', label =
'fitted value on training')
fore3.plot(color='blue', marker = 'o', label ='forecasting values') #forecasted value
on test
plt.xlabel('Cycle', weight='bold')
plt.ylabel('Mean Voltage',weight='bold')
plt.title("Forecasting of Holt’s Damped trend" ,weight='bold')
plt.legend()
plt.show()


# #### We can see that the Holt's Damped Trend is better than other methods.

# #### Forecasting Prediction using  the Holt's Damped Trend:

# In[566]:


RD_holt_linear = RD_holt_linear.reset_index()

RD_holt_linear = RD_holt_linear['mean_voltage']
RD_holt_linear


# In[573]:


n = len(RD_holt_linear)

plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.figure(figsize = (8, 6))
n_predict = 22 # number of cycle we want to predict

fore_newcycle = fit3.forecast(80+n_predict)
fore_valid = fore_newcycle[0:25]
fore_newcycle  = fore_newcycle[25:]
```

```
n = len(RD_holt_linear)

plt.scatter(range(n), RD_holt_linear, marker='.', color='black', label = 'original
data')
plt.plot(range(len(train)), fit3.fittedvalues, color='red',  marker = '*', label =
'fitted value on training')

fore_newcycle.plot(label = 'forecast for next 80 cycles', marker = 'o', color =
'green')
fore_valid.plot(label = 'forecast on the validation', marker = 'o', color = 'blue')

plt.xlabel('Cycle', weight='bold')
plt.ylabel('Mean Voltage',weight='bold')
plt.title("Forecasting of Holt's Damped trend" ,weight='bold')
plt.legend()
plt.show()


# #### 3) Prophet Forecast

# Prophecy is a process of forecasting time series data based on additional models,
where non-linear trends coincide with annual, weekly, and daily seasonality and
holiday effects. It is most suitable for time series with strong seasonal influence
and historical data of multiple seasons. Prophet has strong robustness to missing
data and trend changes, and can usually handle outliers well.

# In[616]:


reference_discharge_df_forcast


# In[617]:


from fbprophet import Prophet
RD_forecast_Prophet = reference_discharge_df_forcast[['date', 'mean_voltage']]
RD_forecast_Prophet = RD_forecast_Prophet.rename(columns={"date": "ds",
"mean_voltage": "y"})
RD_forecast_Prophet.head()


# In[618]:


train = RD_forecast_Prophet[(RD_forecast_Prophet['ds'] < '2014-05-01 00:00:00')]
valid = RD_forecast_Prophet[(RD_forecast_Prophet['ds'] >= '2014-05-01 00:00:00')]
train.head()
train.shape, valid.shape


# In[655]:


#hyperparameter tuning
parameter = {'seasonality_mode':('multiplicative','additive'),
               'changepoint_prior_scale':[0.001, 0.05, 0.1,0.3,0.5],
              'holidays_prior_scale':[0.001, 0.05, 0.1,0.3,0.5],
              'n_changepoints' : [5,10,20,30, 40, 100]}

#Hyperparameter tuning
seasonality_mode_list = ['multiplicative','additive']
```

```
changepoint_prior_scale_list = [0.05, 0.1,0.3,0.5]
holidays_prior_scale_list = [0.05, 0.1,0.3,0.5]
n_changepoints_list = [10,20,30,40,50]

best_parameters = {'seasonality_mode:' : None, 'changepoint_prior_scale:' : None,
'holidays_prior_scale:' : None, 'n_changepoints:' : None}

mse_valid = float('inf')
for ss in seasonality_mode_list:
    for cps in changepoint_prior_scale_list:
        for hps in holidays_prior_scale_list:
            for nc in n_changepoints_list:
                m = Prophet(seasonality_mode = ss, changepoint_prior_scale = cps,
holidays_prior_scale = hps, n_changepoints = nc)
                m.fit(train)
                future = m.make_future_dataframe(periods=valid.shape[0])
                prophet_pred = m.predict(future)
                prophet_pred = pd.DataFrame({ "Prediction" : prophet_pred[-22:]
["yhat"]})
                if np.sum(np.isnan(prophet_pred['Prediction'])) == 0:
                    if mean_squared_error(valid['y'], prophet_pred['Prediction']) <
mse_valid:
                                        best_parameters['seasonality_mode:'] =  ss
                                        best_parameters['changepoint_prior_scale:'] =
cps
                                        best_parameters['holidays_prior_scale:'] = hps
                                        best_parameters['n_changepoints:'] = nc

                                        mse_valid = mean_squared_error(valid['y'],
prophet_pred['Prediction'])


# In[656]:


best_parameters, mse_valid


# In[ ]:


m = Prophet(seasonality_mode = best_parameters['seasonality_mode:'],
changepoint_prior_scale = best_parameters['changepoint_prior_scale:'],
holidays_prior_scale = best_parameters['holidays_prior_scale:'], n_changepoints =
best_parameters['n_changepoints:'])
m.fit(train)
future = m.make_future_dataframe(periods=22)
prophet_pred = m.predict(future)


# In[ ]:


prophet_pred = pd.DataFrame({ "Prediction" : prophet_pred[-22:]["yhat"]})


# In[663]:


valid["FB_Prophet_Predictions"] = prophet_pred['Prediction'].values

n = len(RD_holt_linear)
```

```
plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.figure(figsize = (8, 6))
plt.ylabel('Mean Voltage',weight='bold')
plt.xlabel('Cycle',weight='bold')
plt.title('Prophet Forecasting on Reference Discharge period', weight='bold')
plt.plot(np.arange(n), RD_holt_linear.values, label = 'actual values')

plt.plot(valid.index, valid["FB_Prophet_Predictions"], alpha=0.6, label =
'forecasting values on validation')
####plt.gca().xaxis.set_major_locator(ticker.MultipleLocator(15))
plt.legend()
plt.show()


# We disregard the ds here since we consider each row as different cycle rather than
specific strictly days/months/years

# In[665]:


fb_prophet_rmse_error = rmse(valid['y'], valid["FB_Prophet_Predictions"])
fb_prophet_mse_error = fb_prophet_rmse_error **2
mean_value = RD_holt_linear.mean()

print(f'MSE Error: {fb_prophet_mse_error}\nRMSE Error: {fb_prophet_rmse_error}\nMean:
{mean_value}')


# ### Model comparison:

# Comparison between the Prophet forecasting and Damped Trend:

# In[666]:


mse_errors = [Damped_trend_mse_error, fb_prophet_mse_error]

errors = pd.DataFrame({"Models" : ["Damped_trend",  "Prophet"], "MSE Errors" :
mse_errors})
errors


# In[680]:


valid["FB_Prophet_Predictions"] = prophet_pred['Prediction'].values

n = len(RD_holt_linear)


plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
```

```
plt.rcParams['axes.linewidth'] = 2
plt.figure(figsize = (8, 6))
plt.ylabel('Mean Voltage',weight='bold')
plt.xlabel('Cycle',weight='bold')
plt.title("Forecasting of Prophet vs. Holt's Damped Trend on validation set",
weight='bold')
plt.plot(valid.index, valid['FB_Prophet_Predictions'], alpha=0.6,  linestyle="-",
label = 'Prophet', color = 'red')
plt.plot(valid.index, fore3.values,alpha=0.6, label = 'Holt Damped trend', color =
'orange')
plt.plot(valid.index, Holt_valid.values, linestyle="-.",alpha=0.6, label = 'actual
values', color = 'blue')

plt.legend()
plt.show()


# ## Code for 3.4 Capacity Forecast

# ### Capacity Forecast on Reference Discharge

# In[ ]:


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error
from sklearn import linear_model
from sklearn.neural_network import MLPRegressor
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM, Flatten
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
#from keras.callbacks import EarlyStopping
from keras.layers import ConvLSTM2D
from mat4py import loadmat

#%%
def pullout_dataset(dataset):
    data = loadmat(dataset)
    #Pulling out all the data from raw dataset

    data2=data['data']

    step=data2['step']

    #Eight features
    comment=pd.Series(step['comment'])
    Type=pd.Series(step['type'])
    current=pd.Series(step['current'])
    time=pd.Series(step['time'])
    relativeTime=pd.Series(step['relativeTime'])
    voltage=pd.Series(step['voltage'])
    temperature=pd.Series(step['temperature'])
    date=pd.Series(step['date'])
    #This df is our whole Dataframe
    df=pd.DataFrame([comment,Type,date,current,time,relativeTime,voltage,temperature])
```

```
    df=df.T

df.columns=['comment','Type','date','current','time','relativeTime','voltage','temperature']
    return df


df_9=pullout_dataset('RW9.mat')

#%%
def hour_interval(startTime, endTime):

    total_seconds = (endTime - startTime).total_seconds()

    hour = total_seconds / 60 /60
    return hour

#%%



df_RDC = np.array(df_9[df_9['comment'] == 'reference discharge'])
df_RDC=pd.DataFrame(df_RDC)
df_RDC.columns=['comment','Type','date','current','time','relativeTime','voltage','temperature']

#Degradation of Measured Capacity
# Stage: Reference Discharge
plt.rc('font', family='Arial')
plt.rc('font', size= 18)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2

plt.figure(figsize = (12, 6))
capacity=[]
for i in range(len(df_RDC)):
    cap=np.trapz(df_RDC.iloc[i,3],df_RDC.iloc[i,5])/3600
    capacity.append(cap)
plt.ylabel('Capacity (Ah)')
plt.title('Degradation of Measured Capacity')
plt.scatter(df_RDC['date'],capacity,s=20,c='green')
plt.gca().xaxis.set_major_locator(ticker.MultipleLocator(5))
plt.xticks(rotation=90)
plt.show()



date=[]
from dateutil.parser import parse

for i in range(len(df_RDC)):
    date.append(parse(df_RDC.iloc[i,2]))


j=0
hour_time=[0]
for i in range(len(df_RDC)-1):

    j+=hour_interval(date[i],date[i+1])
    j=round(j,2)
    hour_time.append(j)
```

```
# capacity_df=pd.concat([pd.DataFrame(date),pd.DataFrame(capacity)],axis=1)
# capacity_df.columns=['time(hour)','capacity']
capacity_df=pd.DataFrame(capacity)



#%%
plt.rc('font', family='Arial')
plt.rc('font', size= 18)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2

plt.figure(figsize = (12, 6))

plt.ylabel('Capacity (Ah)')
plt.xlabel('Time(hour)')
plt.title('Degradation of Measured Capacity')
plt.plot(capacity_df,c='green')

plt.show()



#%%
# load the dataset
# dataframe=capacity_df.set_index('time(hour)',inplace=True)

dataframe=capacity_df


dataset = dataframe.values
dataset = dataset.astype('float32')

scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)


train_size = int(len(dataset) * 0.75)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]




def to_sequences(dataset, seq_size=1):
    x = []
    y = []

    for i in range(len(dataset)-seq_size-1):

        window = dataset[i:(i+seq_size), 0]
        x.append(window)
        y.append(dataset[i+seq_size, 0])

    return np.array(x),np.array(y)


seq_size = 1

trainX, trainY = to_sequences(train, seq_size)
```

```
testX, testY = to_sequences(test, seq_size)


#####################################################

trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

print('Single LSTM with hidden Dense...')
model = Sequential()
model.add(LSTM(64, input_shape=(None, seq_size)))
model.add(Dense(32))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')

model.summary()

history=model.fit(trainX, trainY, validation_data=(testX, testY),
          verbose=2, epochs=200)

plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.ylabel('MSE')
plt.xlabel('Epoch')
plt.legend()



trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
#%%

trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))

testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))


train_r2=r2_score(trainY[0], trainPredict[:,0])
print('Train R_square: %.2f %%' % (train_r2*100))


test_r2=r2_score(testY[0], testPredict[:,0])
print('Test R_square: %.2f %%' % (test_r2*100))

#%%
# shift train predictions for plotting
#we must shift the predictions so that they align on the x-axis with the original
dataset.
trainPredictPlot = np.empty_like(dataset)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[seq_size:len(trainPredict)+seq_size, :] = trainPredict

# shift test predictions for plotting
testPredictPlot = np.empty_like(dataset)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(trainPredict)+(seq_size*1)+1:len(dataset)-2, :] = testPredict
```

```
# plot baseline and predictions


plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2

plt.figure(figsize = (18, 6))

plt.ylabel('Capacity',weight='bold')
plt.xlabel('Numbers of Cycle',weight='bold')
plt.title('Capacity Forecast Last 20 Cycles on Reference Discharge',weight='bold')
plt.plot(scaler.inverse_transform(dataset),label='Baseline',linewidth=2)
# plt.plot(trainPredictPlot,label='train-set')
plt.plot(testPredictPlot,label='Forecast',linewidth=2)
plt.legend()
plt.vlines(60, 0.8, 2, linestyles ="dashed", colors ="brown",linewidth=2)
# plt.gca().xaxis.set_major_locator(ticker.MultipleLocator(18))
# plt.xticks(rotation=30)
plt.show()



#%%
def
R2plot(result_train_true,result_train_predict,result_test_true,result_test_predict,mse_test,r2_train
    plt.rc('font', family='Arial')
    plt.rc('font', size= 16)
    plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
    plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
    plt.rcParams['ytick.labelright'] = True
    plt.rcParams['axes.linewidth'] = 2

    plt.figure(figsize = (6, 6))
    fig =plt.gcf()
    ax=plt.gca()
    #ax.set_xticklabels(tick_label)
    # plt.scatter(result_train_true,result_train_predict,s=2,label='Training set of
RW, 80%')
    # plt.scatter(result_test_true,result_test_predict,s=2,label='Test set of RW,
20%')
    plt.scatter(result_train_true,result_train_predict,s=20,label='Train')
    plt.scatter(result_test_true,result_test_predict,s=20,label='Forecast')


    plt.xlim([0.5,2.5])
    plt.ylim([0.5,2.5])
    plt.legend(loc='upper left',frameon=False)
    ax.plot(ax.get_xlim(), ax.get_ylim(), ls="--", c=".5",linewidth=2)

    plt.xlabel(r'Ture', va='top')
    plt.ylabel(r'Predicted', va='bottom')

    plt.title('Predicted vs. True Value on RD',y=1.04,weight='bold')
    plt.text(0.6, 0.27, ' $RMSE $= %.2f' % (mse_test),ha='left',va='center',
transform=fig.transFigure)
    # plt.text(0.5, 0.27, ' $Train: R^{2}$ = %.5f' %
(r2_train),ha='left',va='center', transform=fig.transFigure)
```

```
    plt.text(0.6, 0.22, ' $R^{2}$ = %.2f%%' % (r2_test*100),ha='left',va='center',
transform=fig.transFigure)

    ax.xaxis.set_major_locator(ticker.LinearLocator(6))
    ax.yaxis.set_major_locator(ticker.LinearLocator(6))
    ax.xaxis.set_minor_locator(ticker.AutoMinorLocator(4))
    ax.yaxis.set_minor_locator(ticker.AutoMinorLocator(4))

    ax.tick_params(which='both', direction='out',top=True,labelright=True)
    ax.tick_params(which='major', length=15,width=2)
    ax.tick_params(which='minor', length=8,width=2)
    ax.tick_params(axis='y', labelright=False)
    ax.set_aspect(aspect=1, anchor=None)

    plt.tight_layout(pad=0.4)
    plt.show()
R2plot(result_train_true=trainY,
           result_train_predict=trainPredict,
           result_test_true=testY,
           result_test_predict=testPredict,
           mse_test=testScore,
           r2_train=train_r2,
           r2_test=test_r2)


# ### Capacity Forecast on Random Walk

# In[ ]:


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from sklearn.metrics import mean_squared_error
from sklearn import linear_model
from sklearn.neural_network import MLPRegressor
import math
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
#from keras.callbacks import EarlyStopping
from keras.layers import ConvLSTM2D
from mat4py import loadmat
import statsmodels.api as sm
import warnings
from pylab import rcParams
warnings.filterwarnings("ignore")

#%%
def pullout_dataset(dataset):
    data = loadmat(dataset)
    #Pulling out all the data from raw dataset

    data2=data['data']

    step=data2['step']

    #Eight features
    comment=pd.Series(step['comment'])
    Type=pd.Series(step['type'])
    current=pd.Series(step['current'])
    time=pd.Series(step['time'])
```

```
    relativeTime=pd.Series(step['relativeTime'])
    voltage=pd.Series(step['voltage'])
    temperature=pd.Series(step['temperature'])
    date=pd.Series(step['date'])
    #This df is our whole Dataframe
    df=pd.DataFrame([comment,Type,date,current,time,relativeTime,voltage,temperature])
    df=df.T

df.columns=['comment','Type','date','current','time','relativeTime','voltage','temperature']
    return df


df_9=pullout_dataset('RW9.mat')

#%%
def hour_interval(startTime, endTime):

    total_seconds = (endTime - startTime).total_seconds()

    hour = total_seconds / 60 /60
    return hour

#%%

df_RDC = np.array(df_9[  (df_9['comment'] == 'discharge (random walk)') ])
# df_RDC= np.array(df_9[(df_9['comment'] == 'pulsed load (discharge)') ])
# df_RDC = np.array(df_9[df_9['comment'] == 'reference discharge'])
df_RDC=pd.DataFrame(df_RDC)
df_RDC.columns=['comment','Type','date','current','time','relativeTime','voltage','temperature']
#%%
#Degradation of Measured Capacity
# Stage: Reference Discharge


capacity=[]
for i in range(len(df_RDC)):
    if type(df_RDC.iloc[i,3])!=float:
        cap=np.trapz(df_RDC.iloc[i,3],df_RDC.iloc[i,5])/3600
        capacity.append(cap)
date=[]
from dateutil.parser import parse

for i in range(len(df_RDC)):
    date.append(parse(df_RDC.iloc[i,2]))

date_only = pd.to_datetime(date).date

j=0
hour_time=[0]
for i in range(len(df_RDC)-1):

    j+=hour_interval(date[i],date[i+1])
    j=round(j,2)
    hour_time.append(j)

capacity_df=pd.concat([pd.DataFrame(date_only),pd.DataFrame(capacity)],axis=1)
#HOur time
capacity_df.columns=['date','capacity']

capacity_df['date'] = pd.to_datetime(capacity_df['date'])
capacity_df.set_index('date',inplace=True)
# capacity_df.dropndf_RDC = np.array(df_9[  (df_9['comment'] == 'discharge (random
```

```
walk)') ])a(inplace=True)
plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.figure(figsize = (18, 6))
plt.ylabel('Capacity',weight='bold')
plt.xlabel('Numbers of Cycle',weight='bold')
plt.title('Capacity vs. No.Cycles on Random Walk Period',weight='bold')#for Frist 100
Steps
plt.plot(capacity,alpha=0.6)
# plt.plot(capacity[-1000:],c='green')
plt.gca().xaxis.set_major_locator(ticker.MultipleLocator(3000))

plt.show()

# capacity_df_selected = capacity_df[~capacity_df.index.duplicated(keep='first')]
capacity_df_selected =capacity_df.resample('d').mean().dropna()

#%%

capacity_df_selected.dropna(inplace=True)
plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.figure(figsize = (18, 6))
plt.title('Capacity vs. No.Cycles on Random Walk Period',weight='bold')
plt.ylabel('Capacity',weight='bold')
# plt.title('Time Series Capacity')
plt.scatter(date,capacity_df,c='gray',s=5,label='RW Capacity')
plt.plot(capacity_df_selected,c='blue',label='RW Capacity Mean of Each Date')
plt.gca().xaxis.set_major_locator(ticker.MultipleLocator(20))
# plt.xticks(rotation=90)
plt.legend()
plt.show()


#%%Model


dataframe=capacity_df_selected
dataframe.sort_index(inplace=True)

rcParams['figure.figsize'] = 18, 12
decomposition = sm.tsa.seasonal_decompose(dataframe, model='additive',period =
int(len(dataframe)/10))
fig = decomposition.plot()
plt.show()
#%%

import itertools
p = d = q = range(0, 4)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 0.5) for x in list(itertools.product(p, d, q))]

for param in pdq:
    for param_seasonal in seasonal_pdq:
```

```
        try:
            model =
sm.tsa.statespace.SARIMAX(dataframe,order=param,seasonal_order=param_seasonal,

enforce_stationarity=False,enforce_invertibility=False)

            results = model.fit()
            print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, results.aic))
        except:
            continue
#%%
model = sm.tsa.statespace.SARIMAX(dataframe,
                                order=(3, 3, 3),
                                seasonal_order=(0, 0, 0, 12),
                             )
history = model.fit()
print(history.summary().tables[1])
history.plot_diagnostics(figsize=(16, 8))
plt.show()


#%%
pred_train=history.get_prediction(start=pd.to_datetime('2014-01-07'),end=pd.to_datetime('2014-03-17'
pred = history.get_prediction(start=pd.to_datetime('2014-03-17'), dynamic=False)
pred_ci = pred.conf_int()
ax = dataframe['2014':].plot( figsize=(18, 6),linewidth=2)
pred.predicted_mean.plot(ax=ax, label='Forecast', alpha=.7, figsize=(18,
6),linewidth=2)
# ax.fill_between(pred_ci.index,
#                 pred_ci.iloc[:, 0],
#                 pred_ci.iloc[:, 1], color='k', alpha=.1)

plt.vlines('2014-03-17', 0.02, 0.15, linestyles ="dashed", colors
="brown",linewidth=2)
plt.ylabel('Capacity',weight='bold')
plt.xlabel('Date',weight='bold')
plt.title('Capacity Forecast 2 months on Random Walk',weight='bold')
plt.gca().xaxis.set_major_locator(ticker.MultipleLocator(15))
ax.legend()
plt.show()
#%%

trainPredict = np.array(pred_train.predicted_mean)
trainY = dataframe['2014-01-07':'2014-03-17']

testPredict = np.array(pred.predicted_mean)
testY = dataframe['2014-03-17':]

# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY, trainPredict))
print('Train Score: %.2f RMSE' % (trainScore))

testScore = math.sqrt(mean_squared_error(testY, testPredict))
print('Test Score: %.2f RMSE' % (testScore))


train_r2=r2_score(trainY, trainPredict)
print('Train R_square: %.2f %%' % (train_r2*100))


test_r2=r2_score(testY, testPredict)
print('Test R_square: %.2f %%' % (test_r2*100))
```

```
#%%
def
R2plot(result_train_true,result_train_predict,result_test_true,result_test_predict,mse_test,r2_train
    plt.rc('font', family='Arial')
    plt.rc('font', size= 16)
    plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
    plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
    plt.rcParams['ytick.labelright'] = True
    plt.rcParams['axes.linewidth'] = 2

    plt.figure(figsize = (6, 6))
    fig =plt.gcf()
    ax=plt.gca()
    #ax.set_xticklabels(tick_label)
    # plt.scatter(result_train_true,result_train_predict,s=2,label='Training set of
RW, 80%')
    # plt.scatter(result_test_true,result_test_predict,s=2,label='Test set of RW,
20%')
    plt.scatter(result_train_true,result_train_predict,s=20,label='Train')
    plt.scatter(result_test_true,result_test_predict,s=20,label='Forecast')


    plt.xlim([0.0,0.2])
    plt.ylim([0.0,0.2])
    plt.legend(loc='upper left',frameon=False)
    ax.plot(ax.get_xlim(), ax.get_ylim(), ls="--", c=".5",linewidth=2)

    plt.xlabel(r'Ture', va='top')
    plt.ylabel(r'Predicted', va='bottom')
    plt.title('Predicted vs. True Value on RW',y=1.04,weight='bold')

    plt.text(0.6, 0.27, ' $RMSE $= %.2f' % (mse_test),ha='left',va='center',
transform=fig.transFigure)
    # plt.text(0.5, 0.27, ' $Train: R^{2}$ = %.5f' %
(r2_train),ha='left',va='center', transform=fig.transFigure)
    plt.text(0.6, 0.22, ' $R^{2}$ = %.2f%%' % (r2_test*100),ha='left',va='center',
transform=fig.transFigure)

    ax.xaxis.set_major_locator(ticker.LinearLocator(6))
    ax.yaxis.set_major_locator(ticker.LinearLocator(6))
    ax.xaxis.set_minor_locator(ticker.AutoMinorLocator(4))
    ax.yaxis.set_minor_locator(ticker.AutoMinorLocator(4))

    ax.tick_params(which='both', direction='out',top=True,labelright=True)
    ax.tick_params(which='major', length=15,width=2)
    ax.tick_params(which='minor', length=8,width=2)
    ax.tick_params(axis='y', labelright=False)
    ax.set_aspect(aspect=1, anchor=None)

    plt.tight_layout(pad=0.4)
    plt.show()
R2plot(result_train_true=trainY,
            result_train_predict=trainPredict,
            result_test_true=testY,
            result_test_predict=testPredict,
            mse_test=testScore,
            r2_train=train_r2,
            r2_test=test_r2)
```

```
# ## Code for 3.5 Cycle Duration Forecast

# In[ ]:


# In[1]:


import pandas as pd
import numpy as np
from mat4py import loadmat
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.linear_model import LinearRegression
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.linear_model import Lasso
from sklearn import model_selection
from sklearn.model_selection import cross_val_score
from sklearn.metrics import r2_score
import statsmodels.api as sm
import tensorflow as tf
from sklearn.metrics import mean_squared_error


# In[2]:


data = loadmat('RW9.mat')
data2=data['data']

step=data2['step']

#Eight features
comment=pd.Series(step['comment'])
Type=pd.Series(step['type'])
current=pd.Series(step['current'])
time=pd.Series(step['time'])
relativeTime=pd.Series(step['relativeTime'])
voltage=pd.Series(step['voltage'])
temperature=pd.Series(step['temperature'])
date=pd.Series(step['date'])

#%%This df is our whole Dataframe
df=pd.DataFrame([comment,Type,date,current,time,relativeTime,voltage,temperature])
df=df.T
df.columns=['comment','Type','date','current','time','relativeTime','voltage','temperature']

#%%There are 15 different comment types
comment_type = []
for i in comment:
    comment_type.append(i)
comment_type = set(comment_type)
comment_type=list(comment_type)



df_RW = np.array(df[(df['comment'] == 'rest (random walk)') | (df['comment'] ==
'discharge (random walk)') | (df['comment'] == 'charge (random walk)')])

df_RW=pd.DataFrame(df_RW)
```

```
df_RW.columns=['comment','Type','date','current','time','relativeTime','voltage','temperature']


# To prevent the out of memory stuff comes out, I select the df_RW with indexs $100 *
1^i$ where i $\in$ $[0, 1126]$.

# In[3]:


index = [i for i in range(1,len(df_RW), 100)]
df_RW_selected = df_RW[df_RW.index.isin(index)]


# In[4]:



for i in range(len(df_RW_selected)):
        if  isinstance(df_RW_selected.iloc[i,3], float) or
isinstance(df_RW_selected.iloc[i,3], int): #check for if current is float or int
            df_RW_selected.iloc[i,3] = [df_RW_selected.iloc[i,3]]
        if  isinstance(df_RW_selected.iloc[i,4], float) or
isinstance(df_RW_selected.iloc[i,4], int):  #check for if time is float or int
            df_RW_selected.iloc[i,4] = [df_RW_selected.iloc[i,4]]
        if  isinstance(df_RW_selected.iloc[i,5], float) or
isinstance(df_RW_selected.iloc[i,5], int): #check for if relativeTime is float
            df_RW_selected.iloc[i,5] = [df_RW_selected.iloc[i,5]]
        if  isinstance(df_RW_selected.iloc[i,6], float) or
isinstance(df_RW_selected.iloc[i,6], int):  #check for if voltage is float  or int
            df_RW_selected.iloc[i,6] = [df_RW_selected.iloc[i,6]]
        if  isinstance(df_RW_selected.iloc[i,7], float) or
isinstance(df_RW_selected.iloc[i,7], int):  #check for if temperature is float or int
            df_RW_selected.iloc[i,7] = [df_RW_selected.iloc[i,7]]
df_RW_selected.head()  #one thing to check if the comment is in the order in its
original data set (df)
df_RW_selected = df_RW_selected.reset_index()


# We want to know the time duration for each cycle and what  the time duration in the
next n cycles will be.

# For each cycle:
#
# Duration = Time(s) (Cycle ends)  - Time(s) (Cycle starts).

# In[5]:


reference_discharge_df_forcast_duration = df.copy()
reference_discharge_df_forcast_duration =  reference_discharge_df_forcast_duration
[reference_discharge_df_forcast_duration['comment'] == 'reference discharge']
reference_discharge_df_forcast_duration['date'] =
pd.to_datetime(reference_discharge_df_forcast_duration['date'], format = '%d-%b-%Y %H:
%M:%S', errors = 'ignore')
reference_discharge_df_forcast_duration =
reference_discharge_df_forcast_duration[['date',  'time']]

reference_discharge_df_forcast_duration =
reference_discharge_df_forcast_duration.sort_values(by=['date']).copy()
date_index = reference_discharge_df_forcast_duration.index

reference_discharge_df_forcast_duration
```

```
# In[6]:


reference_discharge_df_forcast_duration =
reference_discharge_df_forcast_duration.reset_index().drop(columns ='index', axis = 1)
reference_discharge_df_forcast_duration


# In[7]:


reference_discharge_df_forcast_duration


# In[8]:


duration = list()
for i in range(len(reference_discharge_df_forcast_duration)):
    duration.append(reference_discharge_df_forcast_duration['time'][i][-1] -
reference_discharge_df_forcast_duration['time'][i][0])

reference_discharge_df_forcast_duration['duration'] = duration
reference_discharge_df_forcast_duration =
reference_discharge_df_forcast_duration[['date', 'duration']]
reference_discharge_df_forcast_duration.head()


# In[9]:


sns.lineplot(data=reference_discharge_df_forcast_duration, x="date", y="duration",
label = 'date versus. duration')

reference_discharge_df_forcast_duration =
reference_discharge_df_forcast_duration.rename(columns={'date':'ds', 'duration' :
'y'})
# reference_discharge_df_forcast_duration =
reference_discharge_df_forcast_duration.set_index('ds')


# In[10]:


from statsmodels.tsa.stattools import adfuller

def stationary_test(ts_ref):
    result=adfuller(ts_ref)
    labels = ['Test Statistic','p-value','Numbers of Lags Used','Number of
Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] > 0.05:
        print("not enough evidence against null hypothesis(Ho),showing that it is non-
stationary.")

    else:
        print("strong evidence against the null hypothesis(Ho), reject the null
hypothesis. Data is stationary.")
```

```
# In[11]:


stationary_test(reference_discharge_df_forcast_duration['y'])


# In[12]:


train =
reference_discharge_df_forcast_duration[(reference_discharge_df_forcast_duration['ds']
< '2014-05-01 00:00:00')]
valid =
reference_discharge_df_forcast_duration[(reference_discharge_df_forcast_duration['ds']>=
'2014-05-01 00:00:00')]
train.head()
train.shape, valid.shape


# Hyperparameter tuning:

# In[ ]:


## import itertools
from fbprophet import Prophet

from prophet.diagnostics import cross_validation
from prophet.diagnostics import performance_metrics
import itertools

parameters = {
    'changepoint_prior_scale': [0.1, 0.5],
    'seasonality_prior_scale': [0.1, 1.0],
    'seasonality_mode' : ['additive', 'multiplicative'],
    'holidays_prior_scale' : [0.01, 0.1],
    'n_changepoints' : [10,20],
    'changepoint_range' : [0.01, 0.1, 0.5]

}

cutoffs = pd.to_datetime(['2014-03-01','2014-03-30'])


# Generate all combinations of parameters
all_params = [dict(zip(parameters.keys(), item)) for item in
itertools.product(*parameters.values())]
rmses = []  # Store the RMSEs for each params here

# Use cross validation to evaluate all parameters
for params in all_params:
    m = Prophet(**params).fit(train)  # Fit model with given params
    df_cv = cross_validation(m, cutoffs=cutoffs, horizon='20 days',
parallel="processes")
    df_p = performance_metrics(df_cv, rolling_window=1)
    rmses.append(df_p['rmse'].values[0])

# Find the best parameters
tuning_results = pd.DataFrame(all_params)
```

```
tuning_results['rmse'] = rmses
print(tuning_results)


# In[ ]:


best_params = all_params[np.argmin(rmses)]


# In[13]:


from fbprophet import Prophet

m = Prophet(changepoint_range = 0.001, changepoint_prior_scale = 0.5,
seasonality_prior_scale = 1.0,  seasonality_mode = 'multiplicative',
holidays_prior_scale = 0.1, weekly_seasonality = False, daily_seasonality = True,
n_changepoints = 200) #weekly_seasonality = False, daily_seasonality = True
m.fit(train)


future = m.make_future_dataframe(periods=valid.shape[0])
prophet_pred = m.predict(future)
prophet_pred.tail()


# In[ ]:


prophet_pred = pd.DataFrame({ "Prediction" : prophet_pred[-valid.shape[0]:]["yhat"]})
prophet_pred.head()


# In[17]:


from statsmodels.tools.eval_measures import rmse

valid["FB_Prophet_Predictions"] = prophet_pred['Prediction'].values

n = len(reference_discharge_df_forcast_duration)
plt.rc('font', family='Arial')
plt.rc('font', size= 16)
plt.rcParams['xtick.top'] = plt.rcParams['xtick.bottom'] = True
plt.rcParams['ytick.left'] = plt.rcParams['ytick.right'] = True
plt.rcParams['ytick.labelright'] = True
plt.rcParams['axes.linewidth'] = 2
plt.figure(figsize = (8, 6))
ax = sns.lineplot(x= np.arange(n), y=reference_discharge_df_forcast_duration['y'])
# sns.lineplot(x= valid.index, y=valid["y"])
sns.lineplot(x=valid.index, y = valid["FB_Prophet_Predictions"])
ax.set_xlabel('Cycle')
ax.set_ylabel('Duration(in seconds)')
ax.title.set_text('Forecasting values vs. Actual values using Prophet')
ax.legend(['actual values', 'forecasting values'])

fb_prophet_rmse_error = rmse(valid['y'], valid["FB_Prophet_Predictions"])
fb_prophet_mse_error = mean_squared_error(valid['y'], valid["FB_Prophet_Predictions"])
mean_value = reference_discharge_df_forcast_duration.mean()

print(f'MSE Error: {fb_prophet_mse_error}\nRMSE Error: {fb_prophet_rmse_error}\nMean:
```

```
{mean_value}')
```