final machine learning modelling 2022 carbon_based Supercapacitors without ANN

October 28, 2022

this is the project of 2022 article with 925 rows which have been modified in different shapes to get more efficiency

[1]:
```
import pandas as pd
from category_encoders import *
import matplotlib.pyplot as plt
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
import numpy as np
import pandas as pd
from colorama import Fore, Back, Style
from seaborn import heatmap
import seaborn as sns
```

[2]:
```
def EmptyColumnNames(df):
    EmptyColumns=[]
    for column in df.columns:
        if df[column].isnull().sum()>0:
            EmptyColumns.append(column)
    return EmptyColumns
```

[3]:
```
def TargeEncoding(categorical_features, target):



    X=df[categorical_features]
    y=df[target]


    Encoder=TargetEncoder(min_samples_leaf=20,smoothing=10)
```

```
    EncoderFit=Encoder.fit(X,y)
    dfEncoded=EncoderFit.transform(X)



    df[categorical_features]=dfEncoded[categorical_features]
    dfEncoded=df

    return dfEncoded
```

[4]:
```
def NullColumnsPlot(df):
    for column in df.columns:
        if df[column].isnull().sum()>0:

            df[column].plot(kind='kde')
            plt.show()

            df[column].plot(kind='box')
            plt.xlabel(column,fontsize=15)
            plt.show()

            TotalDataCount=df.shape[0]
            MissingValuesCount=df[column].isnull().sum()
            AvailableValues=TotalDataCount-MissingValuesCount
            UniqueValuesCount=df[column].nunique()
            print('std= %.2f' %df[column].std(),' ','mean= %.2f'%df[column].
 ↪mean())
            print('count of available values= %i' %AvailableValues,' ','count␣
 ↪of missing values= %i'%df[column].isnull().sum(),' ',
                  'count of unique values= %i' %df[column].nunique())
            print('available values/total data = %.3f'%(AvailableValues/
 ↪TotalDataCount),' ',
                  'unique values/available values = %.3f' %(UniqueValuesCount/
 ↪AvailableValues))
            print(80*'#')
            print(2*'\n')
```

[5]:
```
def fillingMissingValues(df):
    EmptyColumnNames(df)

    for column in EmptyColumnNames(df):

        TotalDataCount=df.shape[0]
        MissingValuesCount=df[column].isnull().sum()
        AvailableValues=TotalDataCount-MissingValuesCount
        UniqueValuesCount=df[column].nunique()
        AvailableRatio=AvailableValues/TotalDataCount
```

```
        UniqueRatio=UniqueValuesCount/AvailableValues
        std=df[column].std()
        mean=df[column].mean()
        mode=df[column].mode()
        StdMeanRatio=std/mean


        if AvailableRatio>0.6 and UniqueRatio<(AvailableRatio/2):


            if StdMeanRatio>1:
                df[column].fillna((mean+(std/5)), inplace= True)
                print('column= ',column,' ','available ratio= %.
↪3f'%AvailableRatio,' ',
                      'unique ratio= %.3f'%UniqueRatio)
                print('mean= %.3f'%mean, ' ', 'std= %.3f'%std)
                print(Fore.RED,'missing values filled with %.4f'%(mean+(std/5)))
                print(Style.RESET_ALL)
                print(80*'#')
                print(2*'\n')
                continue


            if StdMeanRatio<=1:
                df[column].fillna((mean-(std/5)), inplace= True)
                print('column= ',column,' ','available ratio= %.
↪3f'%AvailableRatio,' ',
                      'unique ratio= %.3f'%UniqueRatio)
                print('mean= %.3f'%mean, ' ', 'std= %.3f'%std)
                print(Fore.RED,'missing values filled with %.4f'%(mean-(std/5)))
                print(Style.RESET_ALL)
                print(80*'#')
                print(2*'\n')
                continue

            continue



        if AvailableRatio>0.5 and UniqueRatio<0.3:


            if StdMeanRatio>1:
                df[column].fillna((mean+(std/2)), inplace= True)
                print('column= ',column,' ','available ratio= %.
↪3f'%AvailableRatio,' ',
                      'unique ratio= %.3f'%UniqueRatio)
```

```
                print('mean= %.3f'%mean, ' ', 'std= %.3f'%std)
                print(Fore.RED,'missing values filled with %.4f'%(mean+(std/2)))
                print(Style.RESET_ALL)
                print(80*'#')
                print(2*'\n')
                continue



            if StdMeanRatio<=1:
                df[column].fillna((mean-(std/2)), inplace= True)
                print('column= ',column,' ','available ratio= %.
↪3f'%AvailableRatio,' ',
                      'unique ratio= %.3f'%UniqueRatio)

                print('mean= %.3f'%mean, ' ', 'std= %.3f'%std)
                print(Fore.RED,'missing values filled with %.4f'%(mean-(std/2)))
                print(Style.RESET_ALL)
                print(80*'#')
                print(2*'\n')
                continue


            if AvailableRatio>0.9:
                df[column].fillna(mean,inplace=True)
                print('column= ',column,' ','available ratio= %.
↪3f'%AvailableRatio,' ',
                      'unique ratio= %.3f'%UniqueRatio)

                print('mean= %.3f'%mean,'mode= %.4f'%mode, ' ', 'std= %.3f'%std)
                print(Fore.RED,'missing values filled with %.4f'%mean)
                print(Style.RESET_ALL)
                print(80*'#')
                print(2*'\n')
                continue


            if UniqueRatio<0.1:
                df[column].fillna(mode[0],inplace=True)
                print('column= ',column,' ','available ratio= %.
↪3f'%AvailableRatio,' ',
                      'unique ratio= %.3f'%UniqueRatio)

                print('mean= %.3f'%mean,'mode= %.4f'%mode, ' ', 'std= %.3f'%std)
                print(Fore.RED,'missing values filled with %.4f'%mode)
                print(Style.RESET_ALL)
```

```
                print(80*'#')
                print(2*'\n')
                continue

            continue

        else:
            continue
```

[6]:
```
def GradientBoostRegScore():
    k=np.arange(0.2,0.5,0.02)
    list=[]
    for i in k:
        for z in range(1,21):
            X_train, X_test, y_train, y_test= train_test_split(X,y,␣
 ↪test_size=i, random_state=z)
            etr=GradientBoostingRegressor(random_state=z)
            reg=etr.fit(X_train,y_train)
            cv_score=cross_val_score(etr,X_train, y_train,n_jobs=-1,cv=10)
            Predictions=etr.predict(X_test)


            score= etr.score(X_test, y_test)
            list.append([i,cv_score.mean(),z,score])
    opted= max(list, key=lambda x:x[3])
    return opted



def GradientBoostRegMse():
    k=np.arange(0.2,0.5,0.02)
    list=[]
    for i in k:
        for z in range(1,21):
            X_train, X_test, y_train, y_test= train_test_split(X,y,␣
 ↪test_size=i, random_state=z)
            etr=GradientBoostingRegressor(random_state=z)
            reg=etr.fit(X_train,y_train)
            cv_score=cross_val_score(etr,X_train, y_train,n_jobs=-1,cv=10)
            Predictions=etr.predict(X_test)
            mse=mean_squared_error(y_test,Predictions)

            score= etr.score(X_test, y_test)
            list.append([i,cv_score.mean(),z,score,mse])
```

```
    opted= min(list, key=lambda x:x[4])
    return opted




def ExtraTreesRegressionScore():
    k=np.arange(0.2,0.3,0.02)
    list=[]
    for i in k:
        for z in range(1,11):
            X_train, X_test, y_train, y_test= train_test_split(X,y,␣
 ↪test_size=i, random_state=z)
            etr=ExtraTreesRegressor(n_jobs=-1)
            reg=etr.fit(X_train,y_train)
            cv_score=cross_val_score(etr,X_train, y_train,n_jobs=-1,cv=10)
            Predictions=etr.predict(X_test)



            score= etr.score(X_test, y_test)
            list.append([i,cv_score.mean(),z,score])

    opted= max(list, key=lambda x:x[3])
    return opted



def ExtraTreesRegressionMse():
    k=np.arange(0.2,0.3,0.02)
    list=[]
    for i in k:
        for z in range(1,11):
            X_train, X_test, y_train, y_test= train_test_split(X,y,␣
 ↪test_size=i, random_state=z)
            etr=ExtraTreesRegressor(n_jobs=-1)
            reg=etr.fit(X_train,y_train)


            cv_score=cross_val_score(etr,X_train, y_train,n_jobs=-1,cv=10)

            Predictions=etr.predict(X_test)
            mse=mean_squared_error(y_test,Predictions)

            score= etr.score(X_test, y_test)
            list.append([i,cv_score.mean(),z,score,mse])
```

```
    opted= min(list, key=lambda x:x[4])
    return opted



def ExtraTreesRegressionCVScore():
    k=np.arange(0.2,0.4,0.02)
    list=[]
    for i in k:
        for z in range(1,11):
            X_train, X_test, y_train, y_test= train_test_split(X,y,␣
 ↪test_size=i, random_state=z)
            etr=ExtraTreesRegressor(n_jobs=-1)
            reg=etr.fit(X_train,y_train)

            cv_score=cross_val_score(etr,X_train, y_train,n_jobs=-1,cv=10)
            Predictions=etr.predict(X_test)
            score= etr.score(X_test, y_test)
            list.append([i,cv_score.mean(),z,score])

    opted= max(list, key=lambda x:x[1])
    return opted
```

[7]:
```
file="dataset.csv"
```

[8]:
```
df=pd.read_csv(file)
df.shape
```

[8]:
```
(925, 22)
```

[9]:
```
df.columns
```

[9]:
```
Index(['Paper #', 'Ref.', 'Limits of Potential Window (V)',
       'Lower Limit of Potential Window (V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)', 'Capacitance (F/g)',
       'Specific Surface Area (m^2/g)',
       'Charge Transfer Resistance (Rct) (ohm)',
       'Equivalent Series Resistance (Rs) (ohm)', 'Electrode Material',
       'Pore Size (nm)', 'Pore Volume (cm^3/g)', 'Ratio of ID/IG', 'N at%',
       'C at%', 'O at%', 'Electrolyte Chemical Formula',
       'Electrolyte Ion Mobility Ranking', 'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)'],
      dtype='object')
```

[10]:
```
df=df.drop(['Paper #', 'Ref.'],axis=1)
df.head()
```

[10]:

	Limits of Potential Window (V)	Lower Limit of Potential Window (V)	Upper Limit of Potential Window (V)	Potential Window (V)	Current Density (A/g)	Capacitance (F/g)	Specific Surface Area (m^2/g)	Charge Transfer Resistance (Rct) (ohm)	Equivalent Series Resistance (Rs) (ohm)	Electrode Material	Pore Size (nm)	Pore Volume (cm^3/g)	Ratio of ID/IG	N at%	C at%	O at%	Electrolyte Chemical Formula	Electrolyte Ion Mobility Ranking
0	0 to 0.8	0.0	0.8	0.8	1.0	680.0	186.3	NaN	7.70	CNF/RGO/moOxNy	NaN	NaN	1.45	2.1	NaN	NaN	H2SO4	7.0
1	0 to 1	0.0	1.0	1.0	1.0	367.0	537.0	6.1	1.95	sulfur-doped graphene foam (SGF)	NaN	NaN	1.28	0.0	85.6	9.1	KOH	6.0
2	0 to 1	0.0	1.0	1.0	2.0	338.0	537.0	6.1	1.95	sulfur-doped graphene foam (SGF)	NaN	NaN	1.28	0.0	85.6	9.1	KOH	6.0
3	0 to 1	0.0	1.0	1.0	5.0	283.0	537.0	6.1	1.95	sulfur-doped graphene foam (SGF)	NaN	NaN	1.28	0.0	85.6	9.1	KOH	6.0
4	0 to 1	0.0	1.0	1.0	10.0	246.0	537.0	6.1	1.95	sulfur-doped graphene foam (SGF)	NaN	NaN	1.28	0.0	85.6	9.1		

```
4                          KOH                                  6.0

   Electrolyte Concentration (M)  \
0                            1.0
1                            6.0
2                            6.0
3                            6.0
4                            6.0

  Cell Configuration (three/two electrode system)
0                          three-electrode system
1                            two-electrode system
2                            two-electrode system
3                            two-electrode system
4                            two-electrode system
```

[11]:
```
df.isnull().sum()
```

[11]:
```
Limits of Potential Window (V)                     4
Lower Limit of Potential Window (V)                4
Upper Limit of Potential Window (V)                4
Potential Window (V)                               5
Current Density (A/g)                             16
Capacitance (F/g)                                 17
Specific Surface Area (m^2/g)                    572
Charge Transfer Resistance (Rct) (ohm)           786
Equivalent Series Resistance (Rs) (ohm)          772
Electrode Material                                 0
Pore Size (nm)                                   769
Pore Volume (cm^3/g)                             729
Ratio of ID/IG                                   596
N at%                                            690
C at%                                            699
O at%                                            703
Electrolyte Chemical Formula                      22
Electrolyte Ion Mobility Ranking                  99
Electrolyte Concentration (M)                     62
Cell Configuration (three/two electrode system)   14
dtype: int64
```

[12]:
```
df.shape
```

[12]:
```
(925, 20)
```

[13]:
```
df=df[df['Capacitance (F/g)'].notnull()]
df.shape
```

[13]:
```
(908, 20)
```

[14]:
```
df.dtypes==object
```

[14]:
```
Limits of Potential Window (V)                      True
Lower Limit of Potential Window (V)                False
Upper Limit of Potential Window (V)                False
Potential Window (V)                               False
Current Density (A/g)                              False
Capacitance (F/g)                                  False
Specific Surface Area (m^2/g)                      False
Charge Transfer Resistance (Rct) (ohm)             False
Equivalent Series Resistance (Rs) (ohm)            False
Electrode Material                                  True
Pore Size (nm)                                     False
Pore Volume (cm^3/g)                               False
Ratio of ID/IG                                     False
N at%                                              False
C at%                                              False
O at%                                              False
Electrolyte Chemical Formula                        True
Electrolyte Ion Mobility Ranking                   False
Electrolyte Concentration (M)                      False
Cell Configuration (three/two electrode system)     True
dtype: bool
```

[15]:
```
categorical_features=['Limits of Potential Window (V)', 'Electrode Material',
       'Electrolyte Chemical Formula',
       'Cell Configuration (three/two electrode system)']
target='Capacitance (F/g)'
```

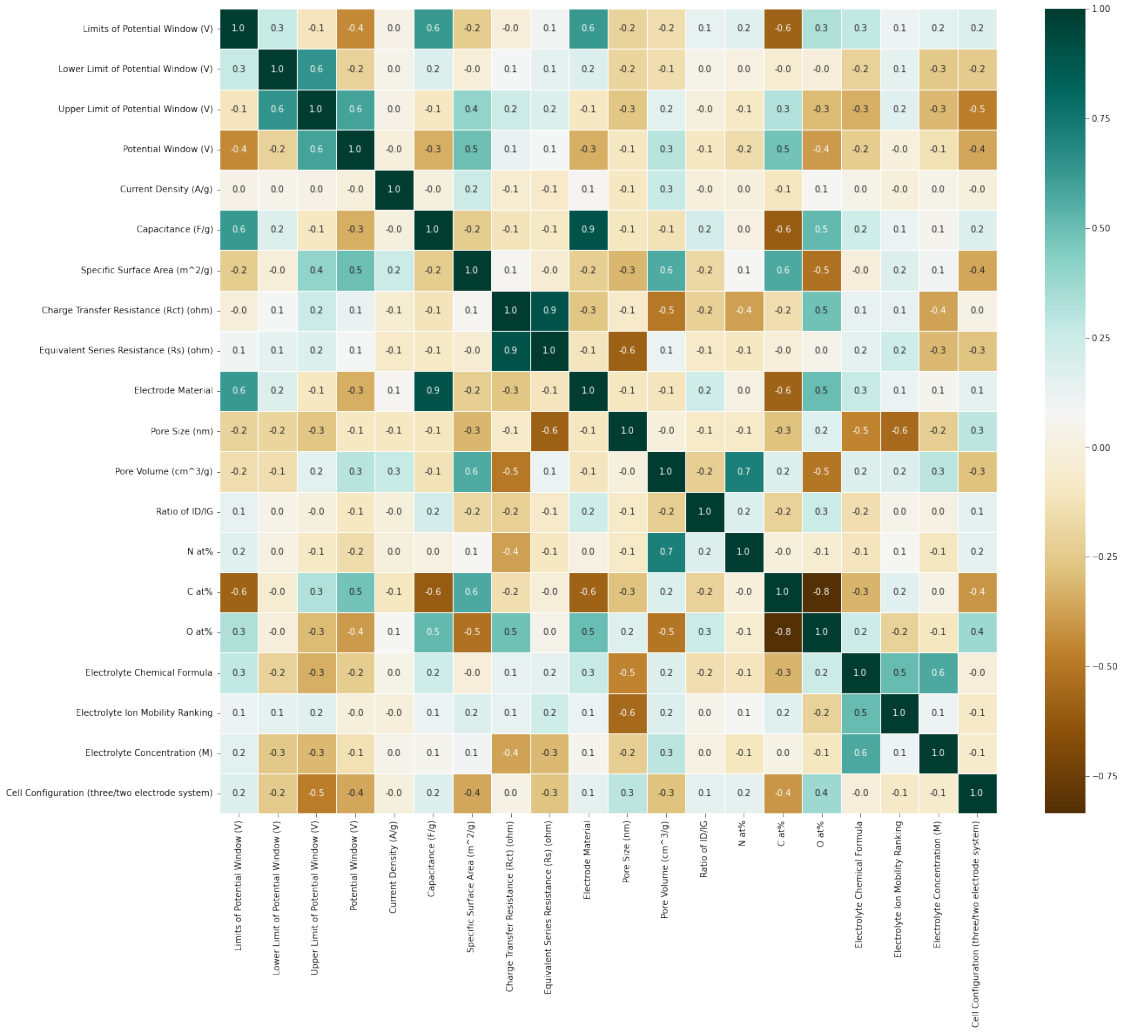
[16]:
```
dfEncoded=TargeEncoding(categorical_features, target)
```

[17]:
```
dfEncoded.shape
```

[17]:
```
(908, 20)
```

[18]:
```
plt.figure(figsize=(20,17))
heatmap(dfEncoded.corr(),annot=True,fmt='.1f',linewidths=0.5,cmap='BrBG')
```

[18]:
```
<AxesSubplot:>
```



[19]:
```
dfEncoded.dtypes
```

[19]:
```
Limits of Potential Window (V)                  float64
Lower Limit of Potential Window (V)             float64
Upper Limit of Potential Window (V)             float64
Potential Window (V)                            float64
Current Density (A/g)                           float64
Capacitance (F/g)                               float64
Specific Surface Area (m^2/g)                   float64
Charge Transfer Resistance (Rct) (ohm)          float64
Equivalent Series Resistance (Rs) (ohm)         float64
Electrode Material                              float64
Pore Size (nm)                                  float64
Pore Volume (cm^3/g)                            float64
Ratio of ID/IG                                  float64
```

```
N at%                                                 float64
C at%                                                 float64
O at%                                                 float64
Electrolyte Chemical Formula                          float64
Electrolyte Ion Mobility Ranking                      float64
Electrolyte Concentration (M)                         float64
Cell Configuration (three/two electrode system)       float64
dtype: object
```

now all the columns are numeric

[20]:
```
dfEncoded.isnull().sum()
```

[20]:
```
Limits of Potential Window (V)                          0
Lower Limit of Potential Window (V)                     1
Upper Limit of Potential Window (V)                     1
Potential Window (V)                                    2
Current Density (A/g)                                   1
Capacitance (F/g)                                       0
Specific Surface Area (m^2/g)                         561
Charge Transfer Resistance (Rct) (ohm)                771
Equivalent Series Resistance (Rs) (ohm)               757
Electrode Material                                      0
Pore Size (nm)                                        755
Pore Volume (cm^3/g)                                  713
Ratio of ID/IG                                        580
N at%                                                 677
C at%                                                 686
O at%                                                 690
Electrolyte Chemical Formula                            0
Electrolyte Ion Mobility Ranking                       97
Electrolyte Concentration (M)                          60
Cell Configuration (three/two electrode system)         0
dtype: int64
```

[21]:
```
dfEncoded.nunique()
```

[21]:
```
Limits of Potential Window (V)                         59
Lower Limit of Potential Window (V)                    15
Upper Limit of Potential Window (V)                    25
Potential Window (V)                                   21
Current Density (A/g)                                  65
Capacitance (F/g)                                     680
Specific Surface Area (m^2/g)                         115
Charge Transfer Resistance (Rct) (ohm)                 38
Equivalent Series Resistance (Rs) (ohm)                35
Electrode Material                                    180
```

```
Pore Size (nm)                                    45
Pore Volume (cm^3/g)                              53
Ratio of ID/IG                                    85
N at%                                             31
C at%                                             67
O at%                                             68
Electrolyte Chemical Formula                      20
Electrolyte Ion Mobility Ranking                   8
Electrolyte Concentration (M)                      8
Cell Configuration (three/two electrode system)    3
dtype: int64
```

[22]:
```
dfEncoded.isnull().mean()
```

[22]:
```
Limits of Potential Window (V)                    0.000000
Lower Limit of Potential Window (V)               0.001101
Upper Limit of Potential Window (V)               0.001101
Potential Window (V)                              0.002203
Current Density (A/g)                             0.001101
Capacitance (F/g)                                 0.000000
Specific Surface Area (m^2/g)                     0.617841
Charge Transfer Resistance (Rct) (ohm)            0.849119
Equivalent Series Resistance (Rs) (ohm)           0.833700
Electrode Material                                0.000000
Pore Size (nm)                                    0.831498
Pore Volume (cm^3/g)                              0.785242
Ratio of ID/IG                                    0.638767
N at%                                             0.745595
C at%                                             0.755507
O at%                                             0.759912
Electrolyte Chemical Formula                      0.000000
Electrolyte Ion Mobility Ranking                  0.106828
Electrolyte Concentration (M)                     0.066079
Cell Configuration (three/two electrode system)   0.000000
dtype: float64
```

[23]:
```
dfEncoded[['Current Density (A/g)','Potential Window (V)','Lower Limit of␣
↪Potential Window (V)'
  ,'Upper Limit of Potential Window (V)']].describe()
```

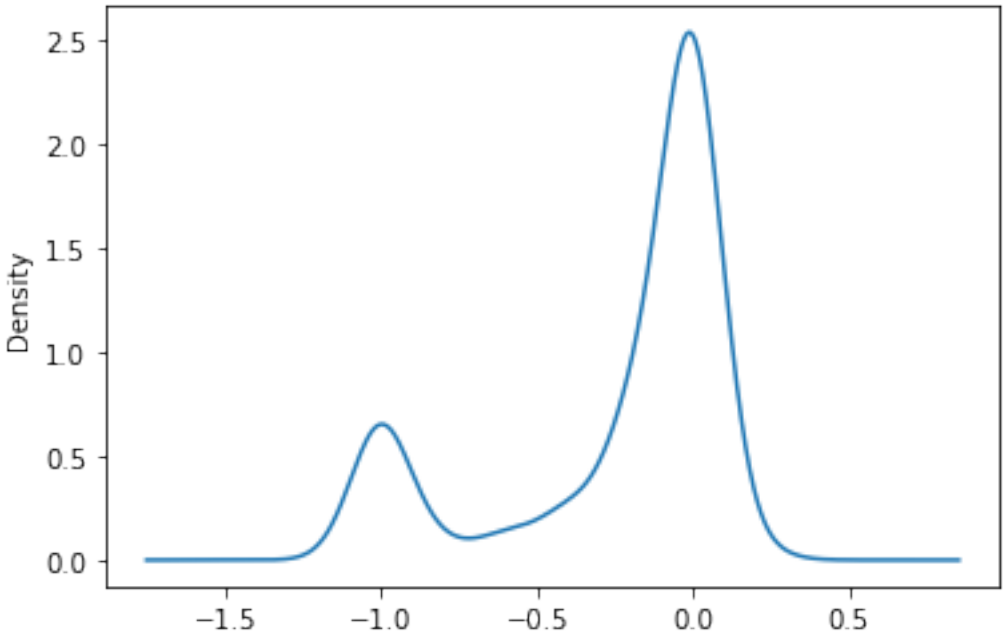
[23]:
```
       Current Density (A/g)  Potential Window (V)  \
count             907.000000            906.000000
mean                5.866494              0.861700
std                13.367131              0.348249
min                 0.050000              0.400000
25%                 1.000000              0.600000
50%                 2.000000              0.825000
```

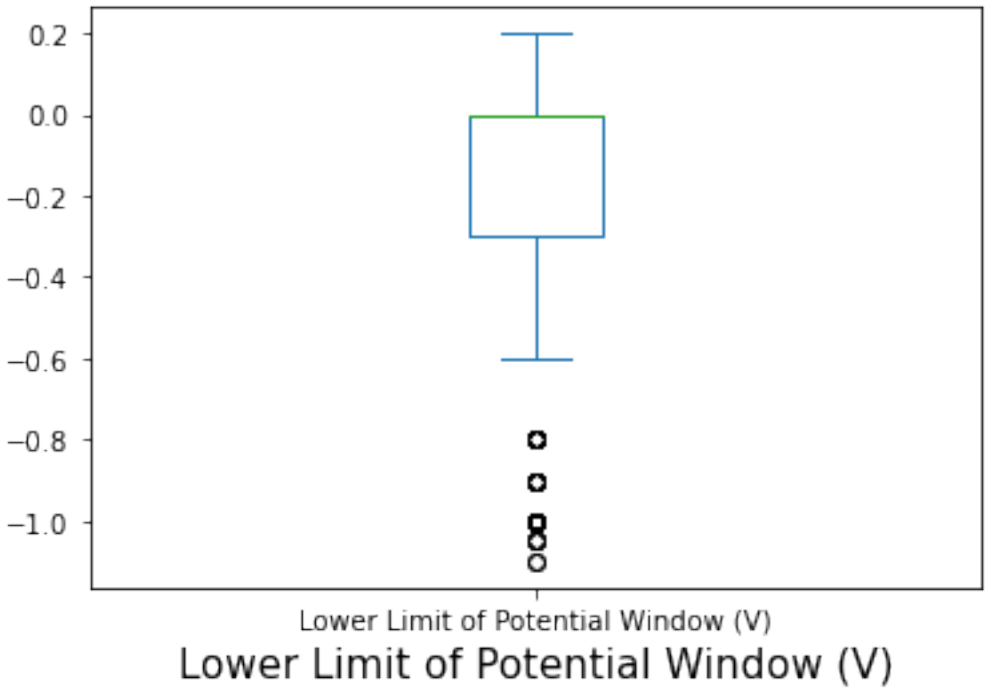
```
75%                  5.000000              1.000000
max                200.000000              3.500000

       Lower Limit of Potential Window (V)  \
count                            907.00000
mean                              -0.23462
std                                0.36850
min                               -1.10000
25%                               -0.30000
50%                                0.00000
75%                                0.00000
max                                0.20000

       Upper Limit of Potential Window (V)
count                           907.000000
mean                              0.627012
std                               0.448844
min                              -0.200000
25%                               0.400000
50%                               0.600000
75%                               0.800000
max                               3.500000
```

[24]:
```
dfEncoded.isnull().sum()<100
```

[24]:
```
Limits of Potential Window (V)                      True
Lower Limit of Potential Window (V)                 True
Upper Limit of Potential Window (V)                 True
Potential Window (V)                                True
Current Density (A/g)                               True
Capacitance (F/g)                                   True
Specific Surface Area (m^2/g)                      False
Charge Transfer Resistance (Rct) (ohm)             False
Equivalent Series Resistance (Rs) (ohm)            False
Electrode Material                                  True
Pore Size (nm)                                     False
Pore Volume (cm^3/g)                               False
Ratio of ID/IG                                     False
N at%                                              False
C at%                                              False
O at%                                              False
Electrolyte Chemical Formula                        True
Electrolyte Ion Mobility Ranking                    True
Electrolyte Concentration (M)                       True
Cell Configuration (three/two electrode system)     True
dtype: bool
```

[25]:
```
NullColumnsPlot(dfEncoded)
```

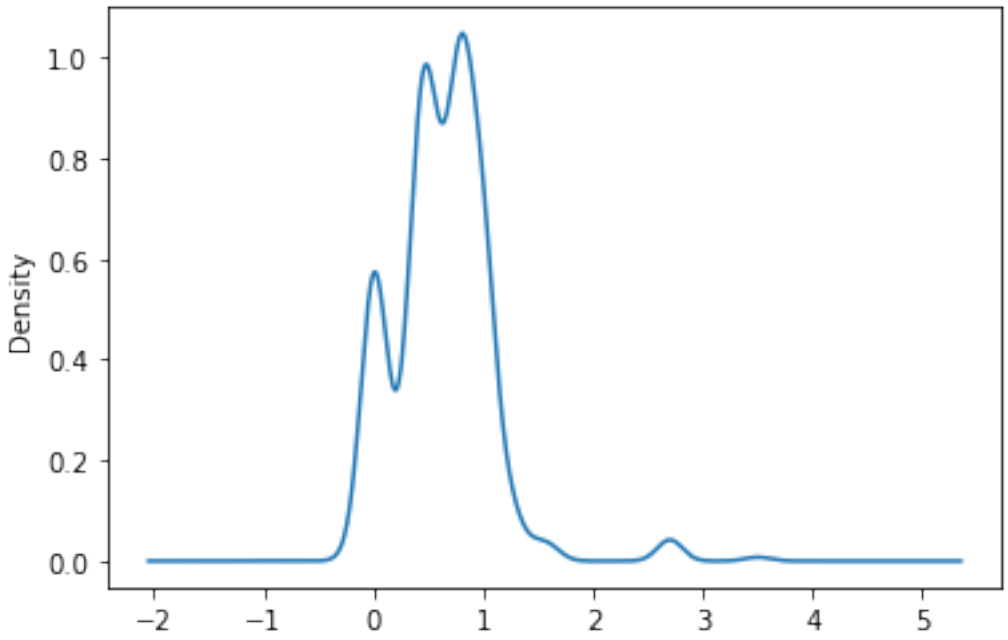


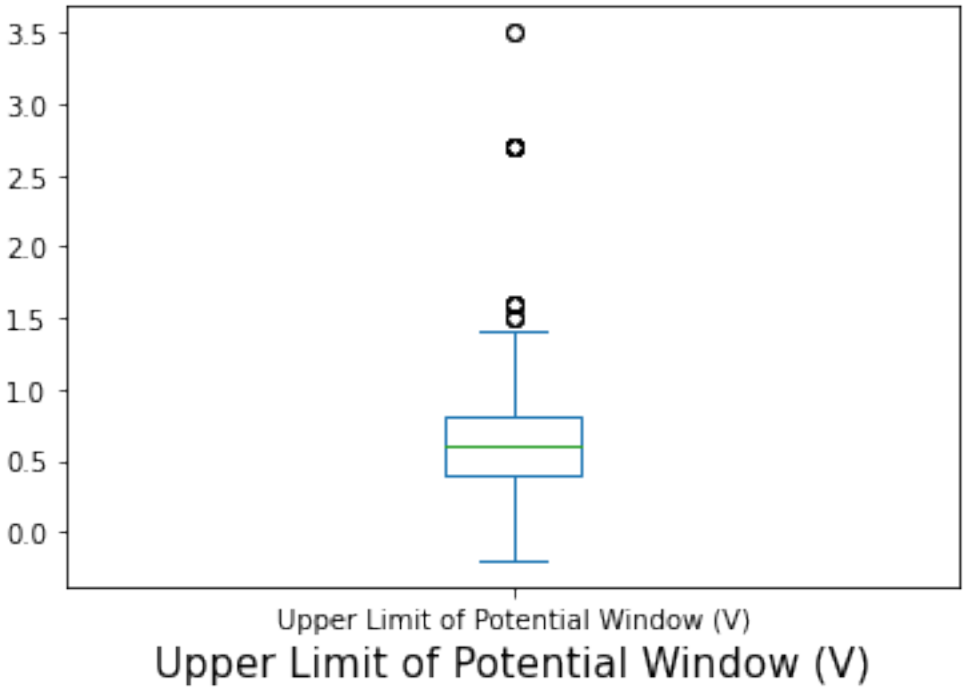


Lower Limit of Potential Window (V)

```
std= 0.37   mean= -0.23
```

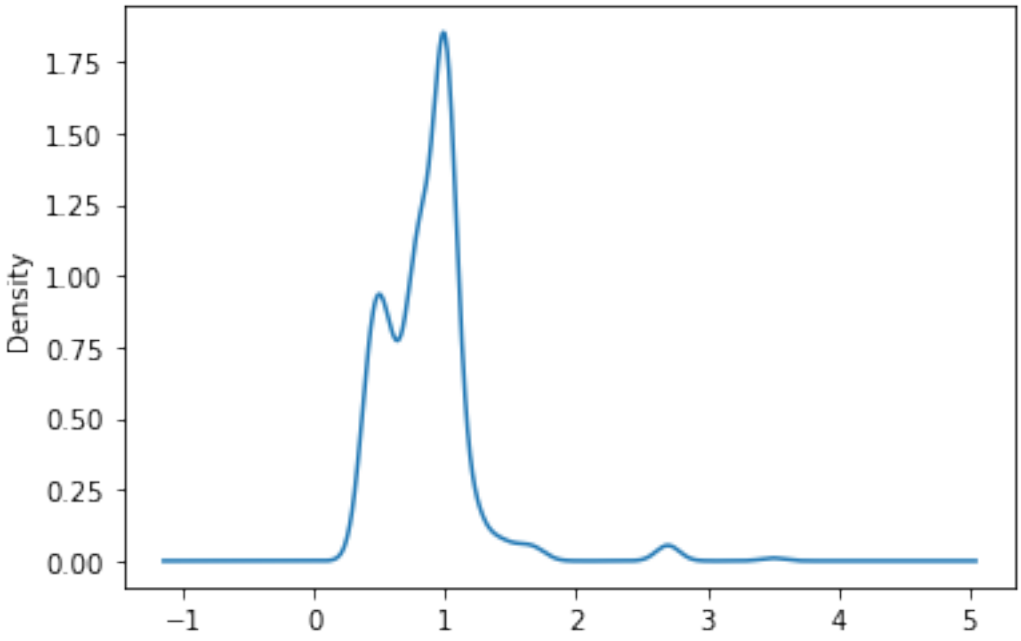
```
count of available values= 907   count of missing values= 1   count of unique
values= 15
available values/total data = 0.999   unique values/available values = 0.017
################################################################################
```

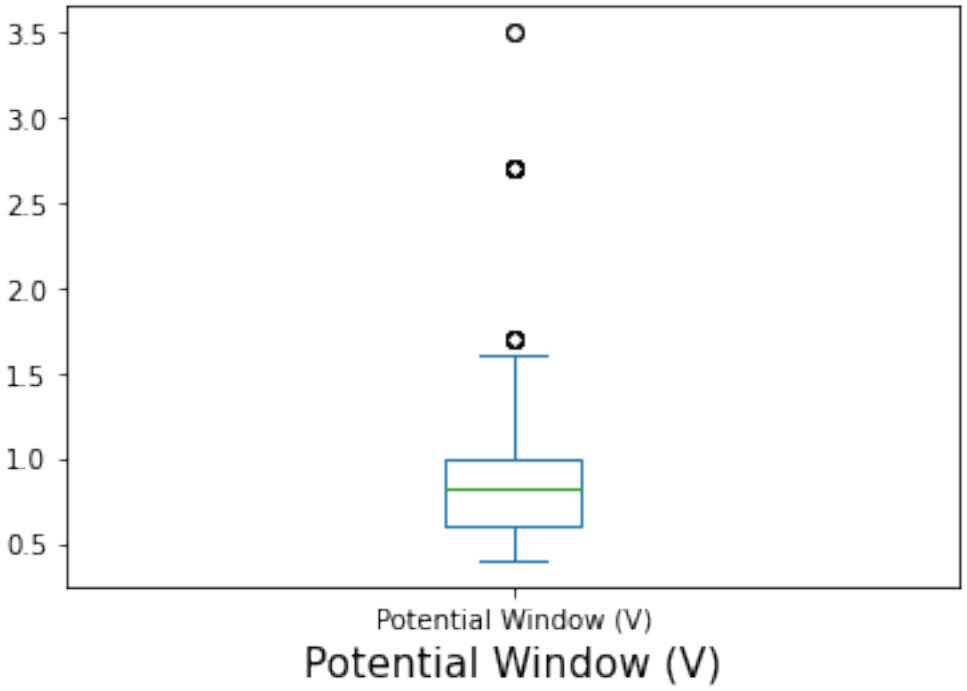




Upper Limit of Potential Window (V)

```
std= 0.45   mean= 0.63
count of available values= 907   count of missing values= 1   count of unique
values= 25
available values/total data = 0.999   unique values/available values = 0.028
################################################################################
```

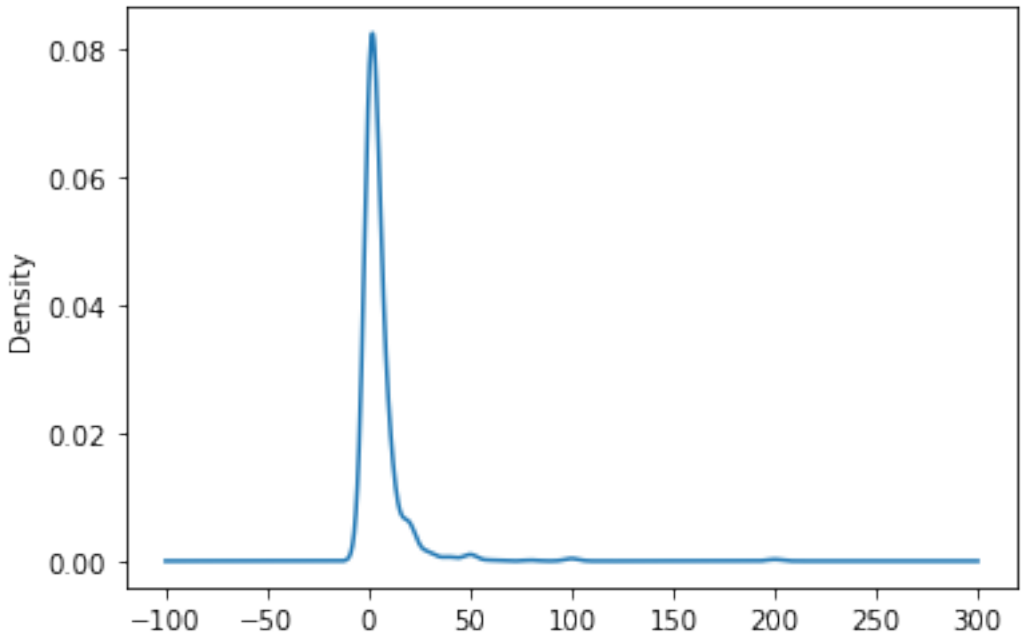


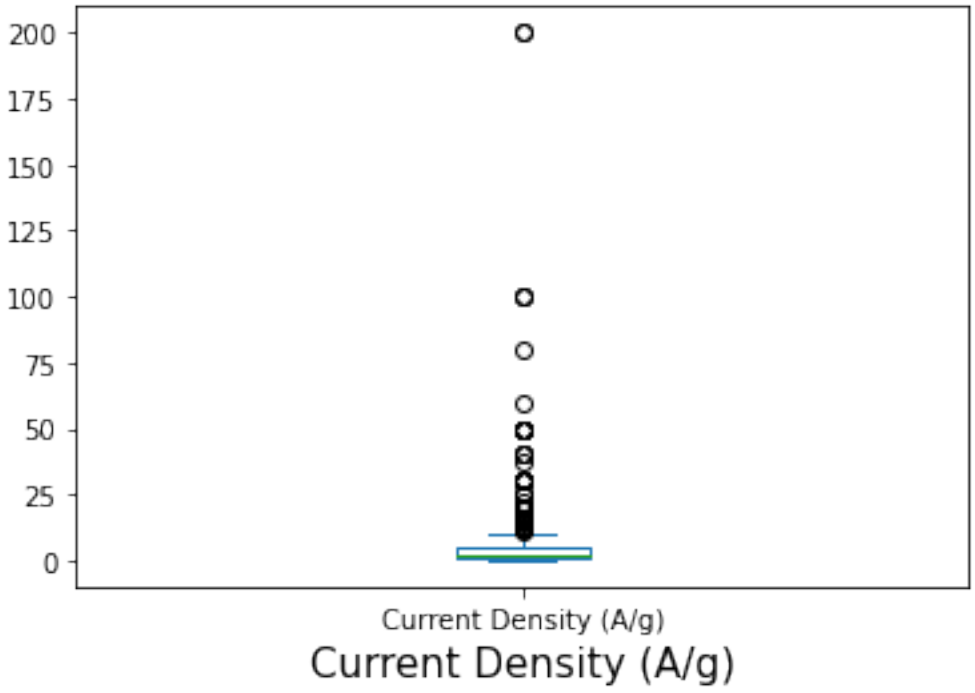


Potential Window (V)

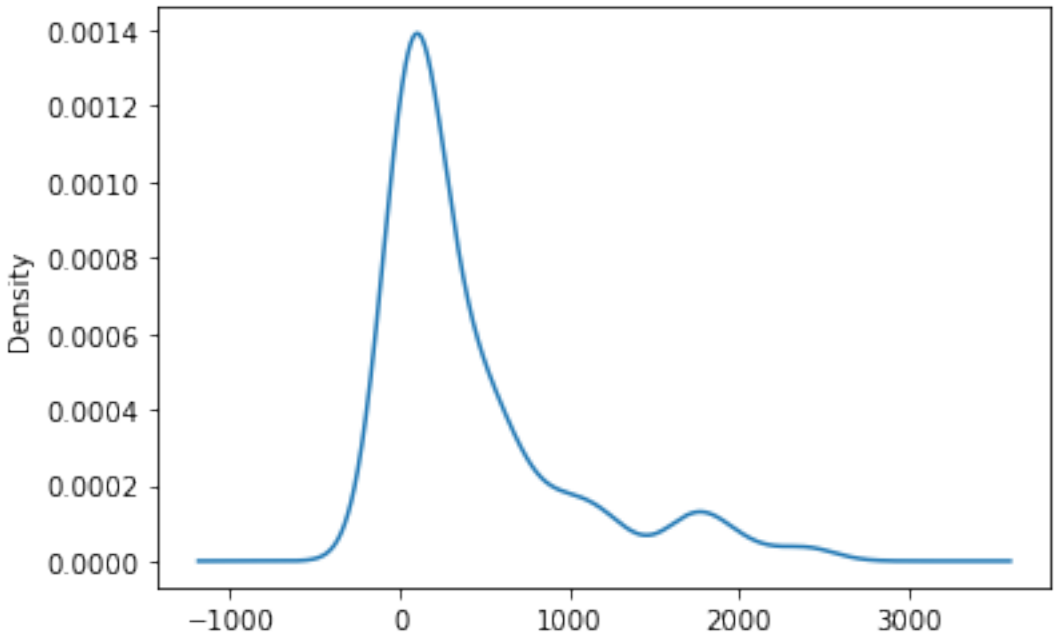
std= 0.35 mean= 0.86
count of available values= 906 count of missing values= 2 count of unique values= 21

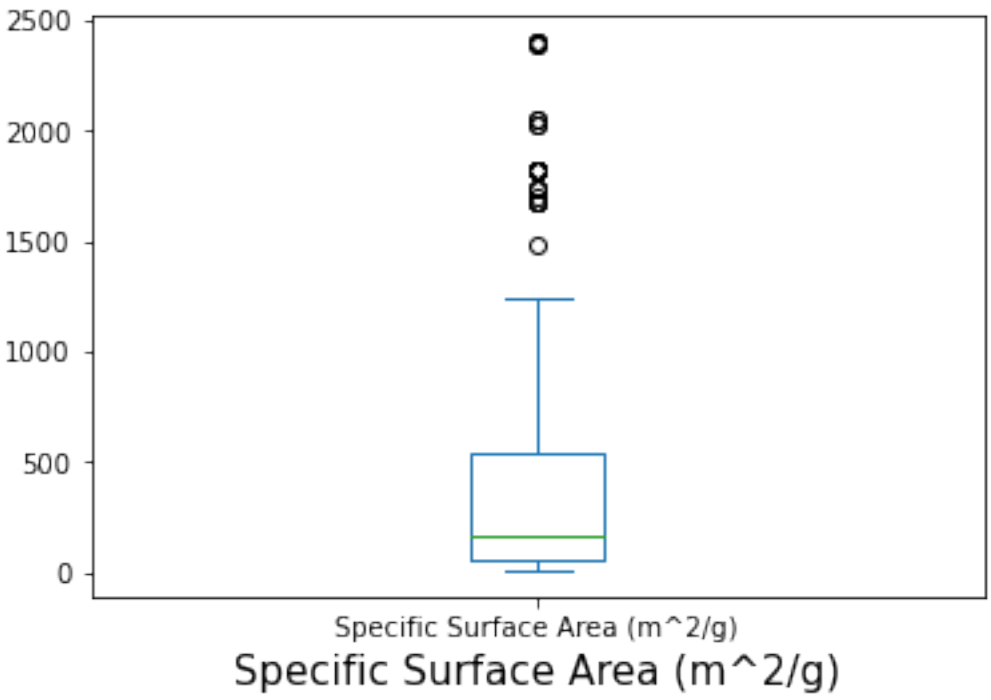
```
available values/total data = 0.998   unique values/available values = 0.023
################################################################################
```



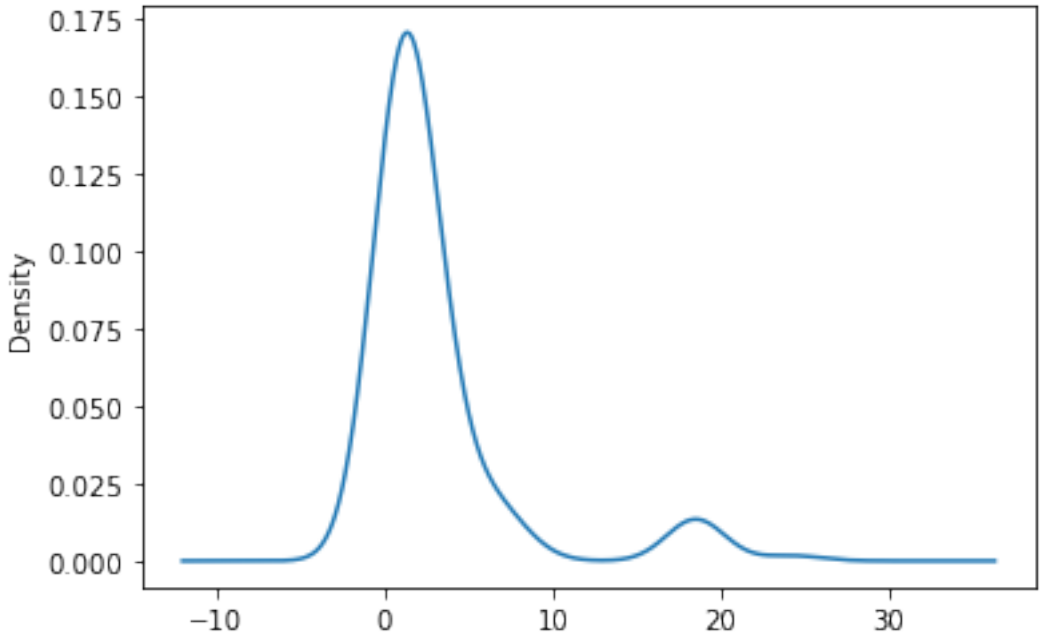


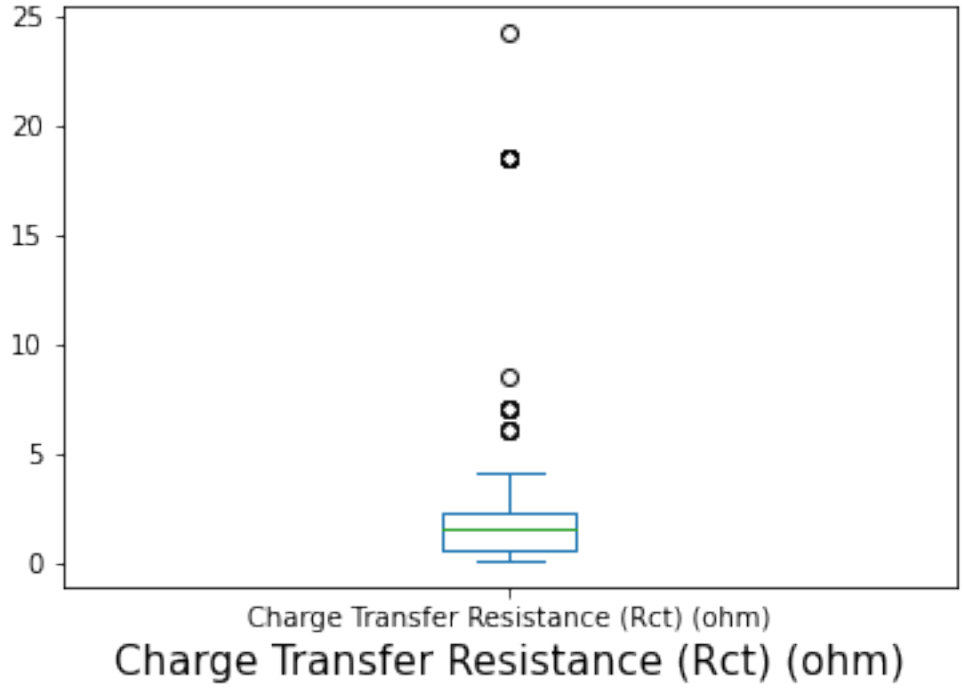
```
std= 13.37   mean= 5.87
count of available values= 907   count of missing values= 1   count of unique
values= 65
available values/total data = 0.999   unique values/available values = 0.072
################################################################################
```






```
std= 547.67   mean= 415.99
count of available values= 347   count of missing values= 561   count of unique
values= 115
available values/total data = 0.382   unique values/available values = 0.331
################################################################################
```

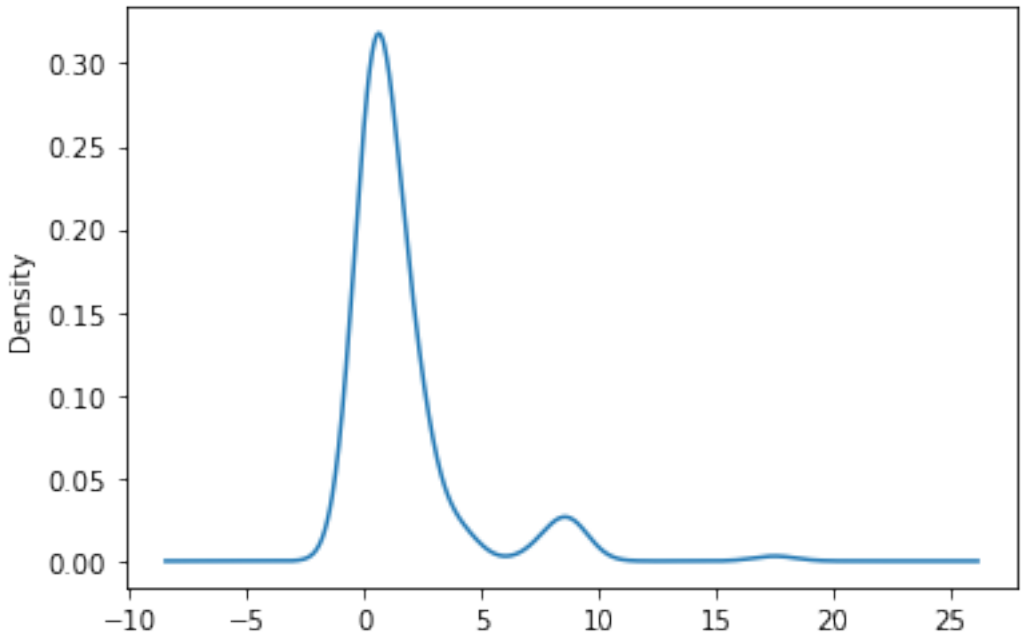


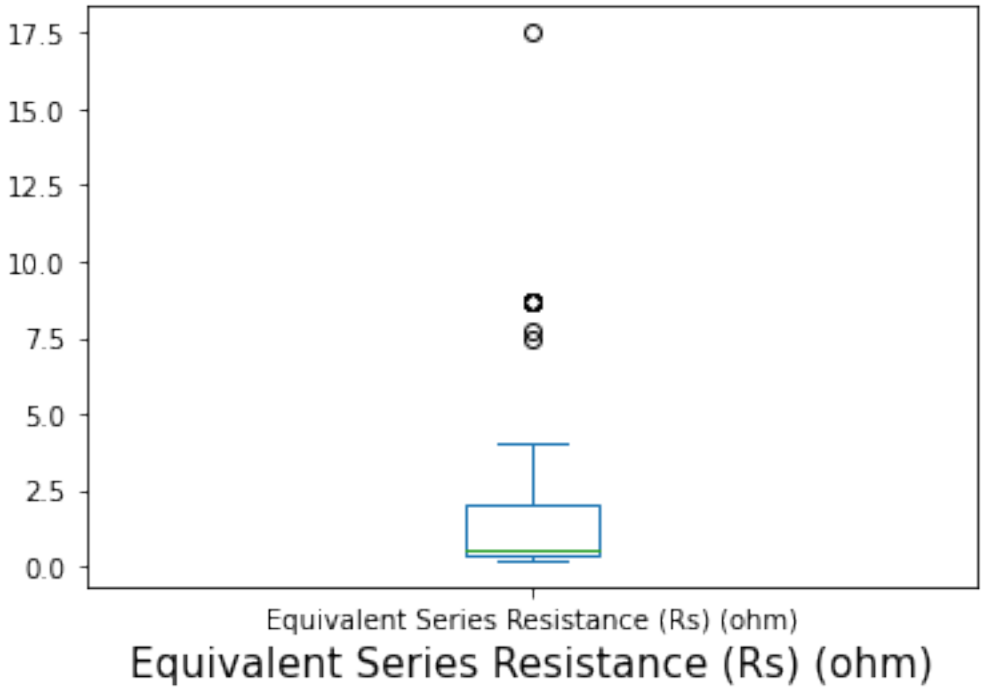


Charge Transfer Resistance (Rct) (ohm)

std= 4.63 mean= 3.03
count of available values= 137 count of missing values= 771 count of unique values= 38

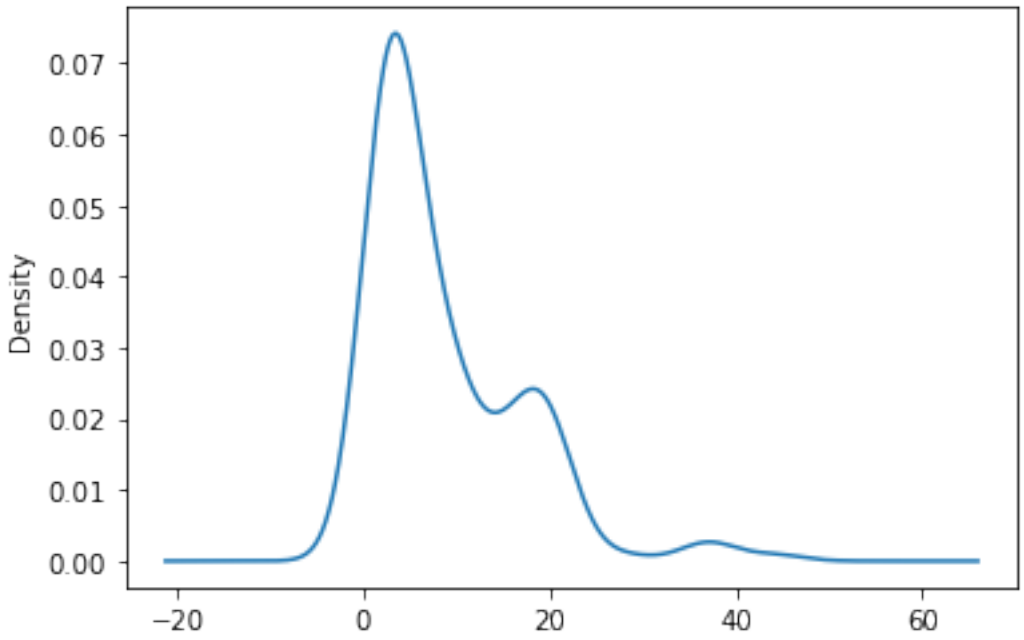
```
available values/total data = 0.151   unique values/available values = 0.277
################################################################################
```

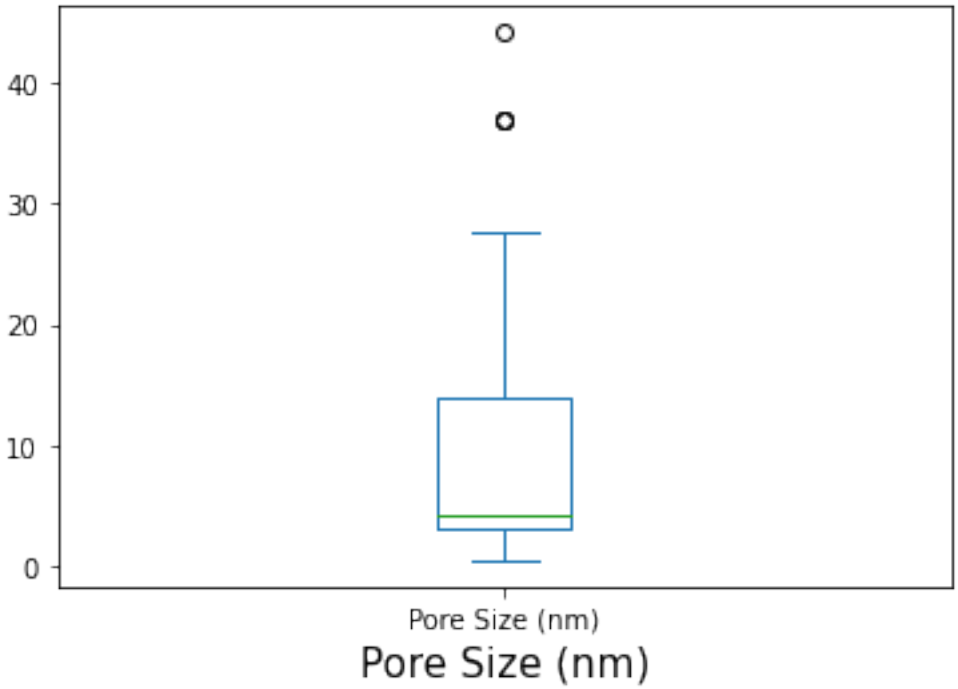




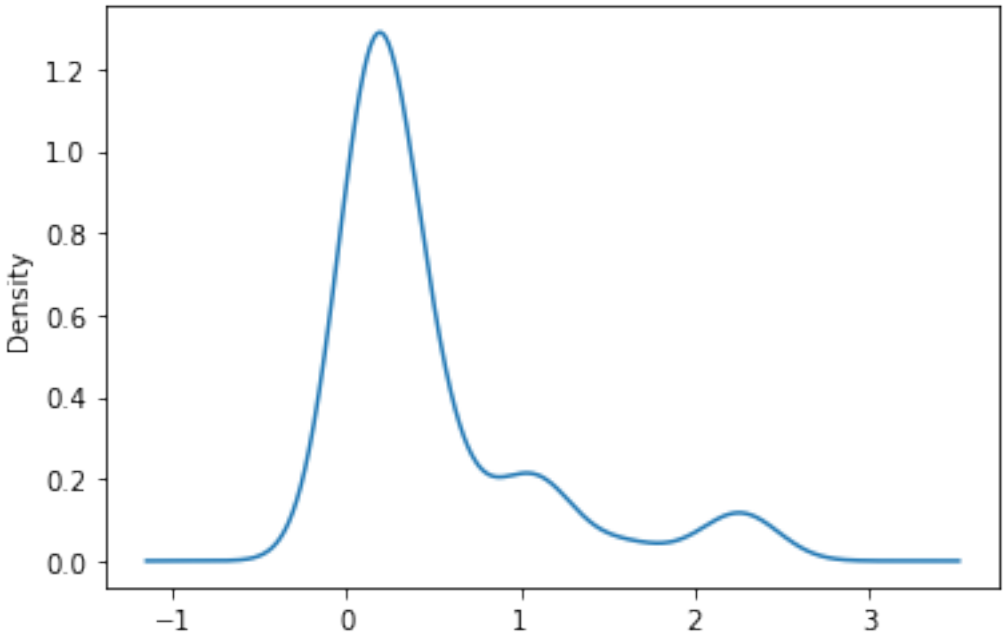
Equivalent Series Resistance (Rs) (ohm)

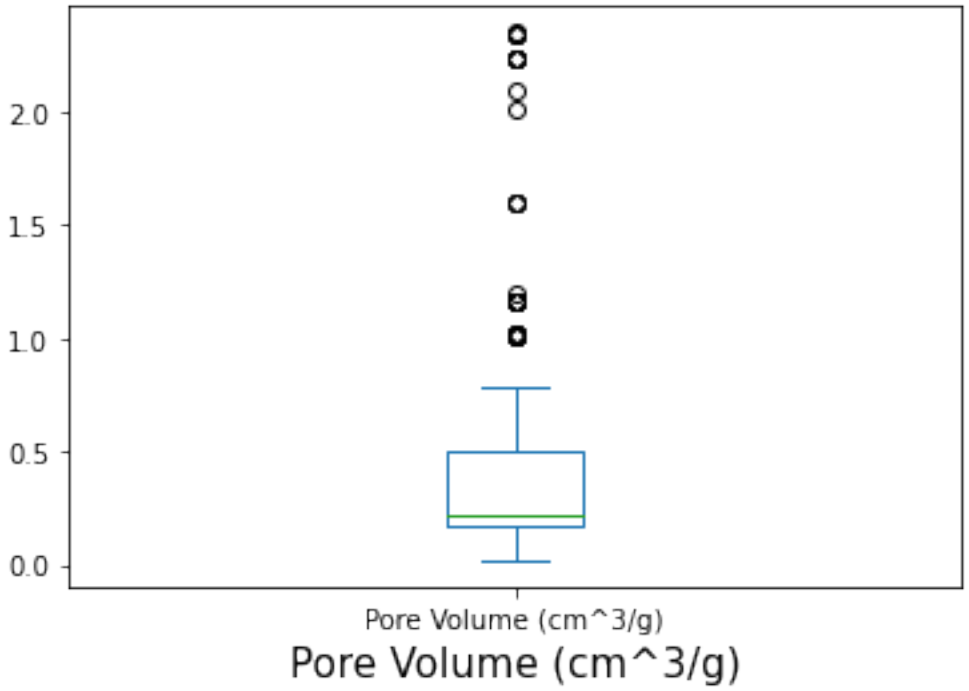
```
std= 2.45   mean= 1.60
count of available values= 151   count of missing values= 757   count of unique
values= 35
available values/total data = 0.166   unique values/available values = 0.232
################################################################################
```






```
std= 8.15   mean= 8.68
count of available values= 153   count of missing values= 755   count of unique
values= 45
available values/total data = 0.169   unique values/available values = 0.294
################################################################################
```

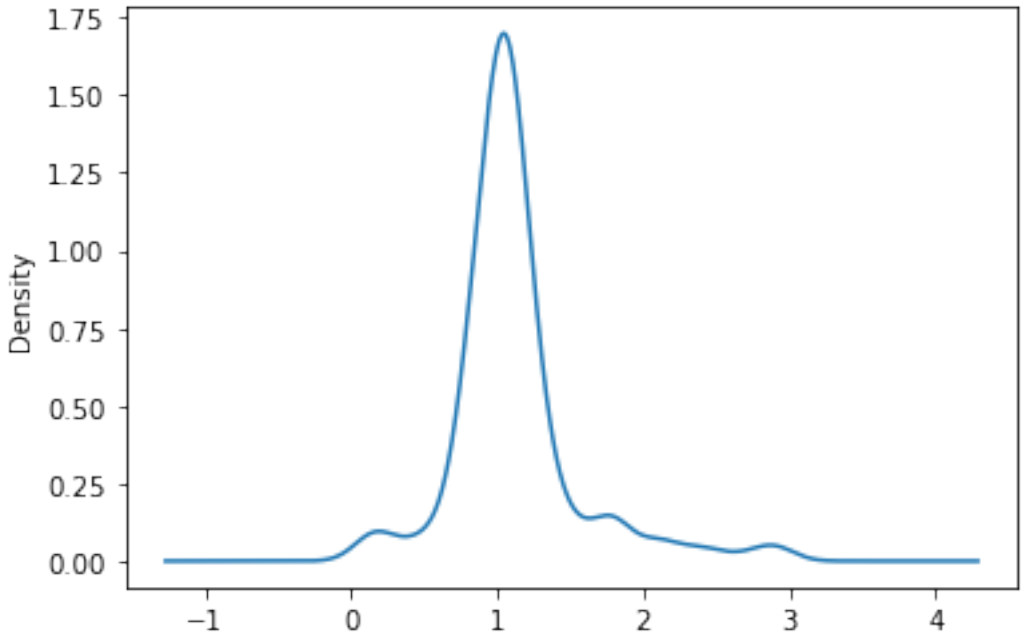


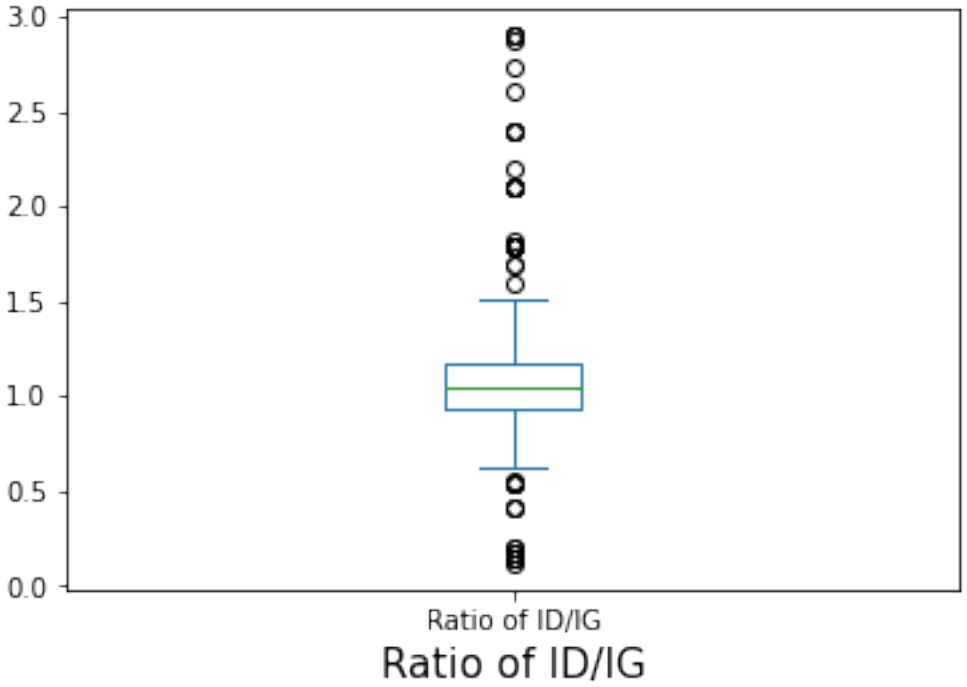


Pore Volume (cm^3/g)

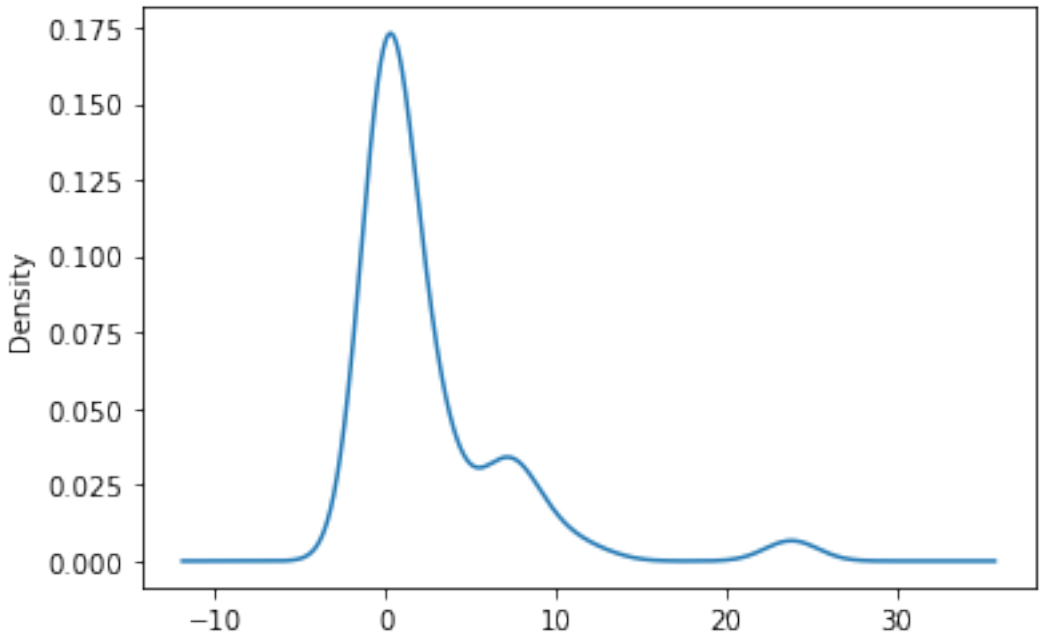
std= 0.59 mean= 0.48
count of available values= 195 count of missing values= 713 count of unique values= 53

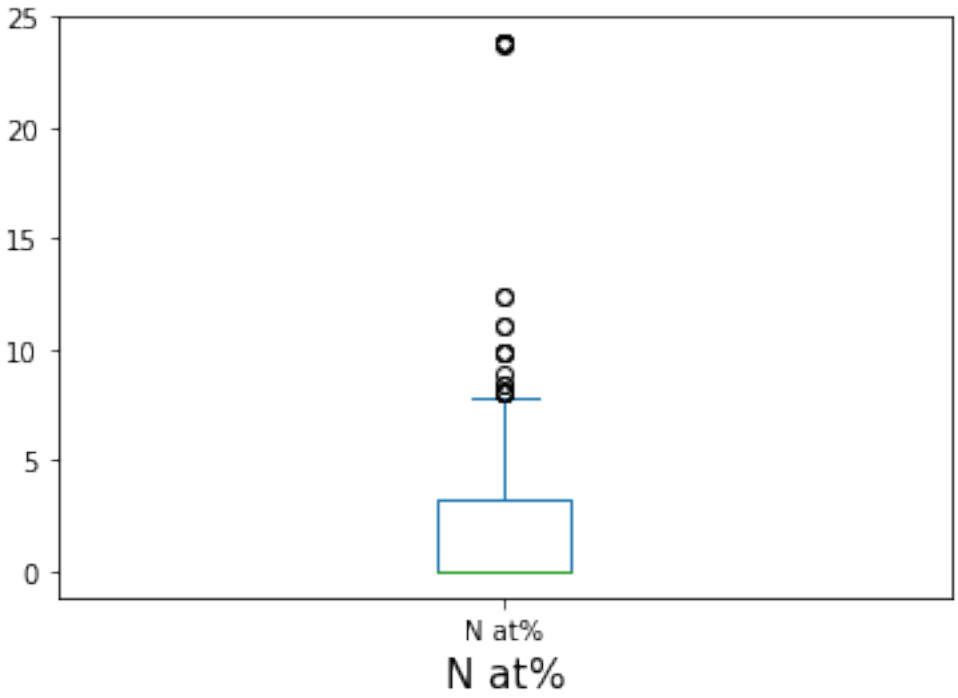
```
available values/total data = 0.215   unique values/available values = 0.272
################################################################################
```



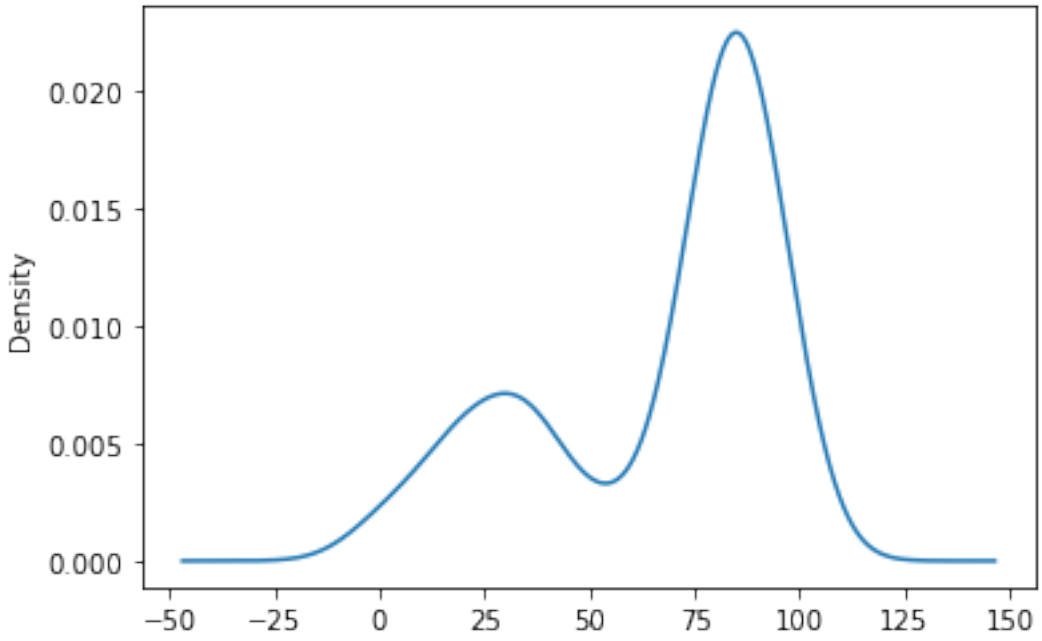


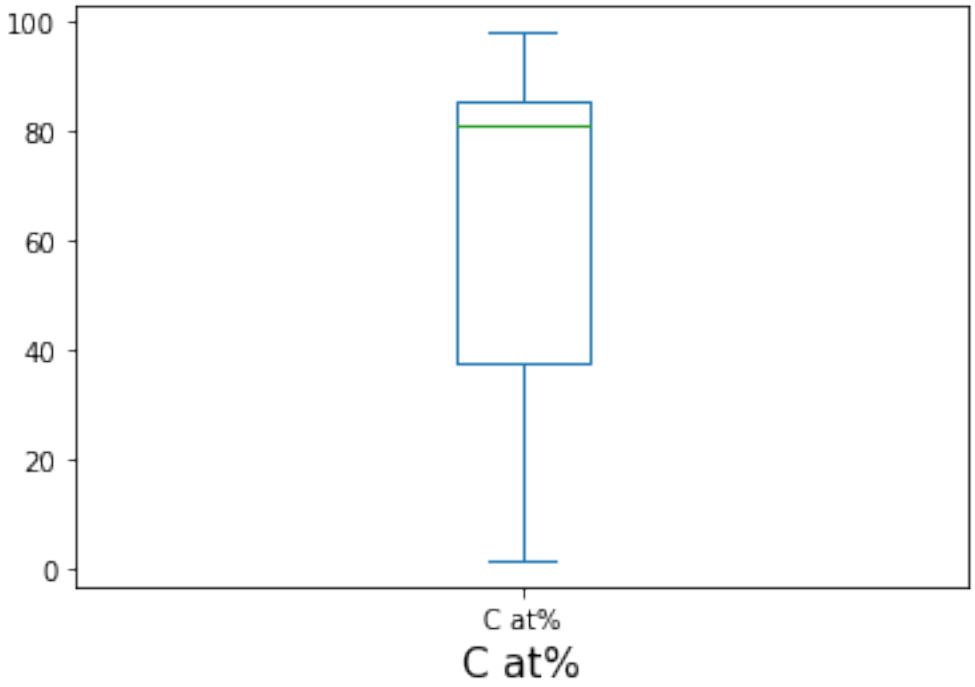
```
std= 0.43   mean= 1.12
count of available values= 328   count of missing values= 580   count of unique
values= 85
available values/total data = 0.361   unique values/available values = 0.259
################################################################################
```





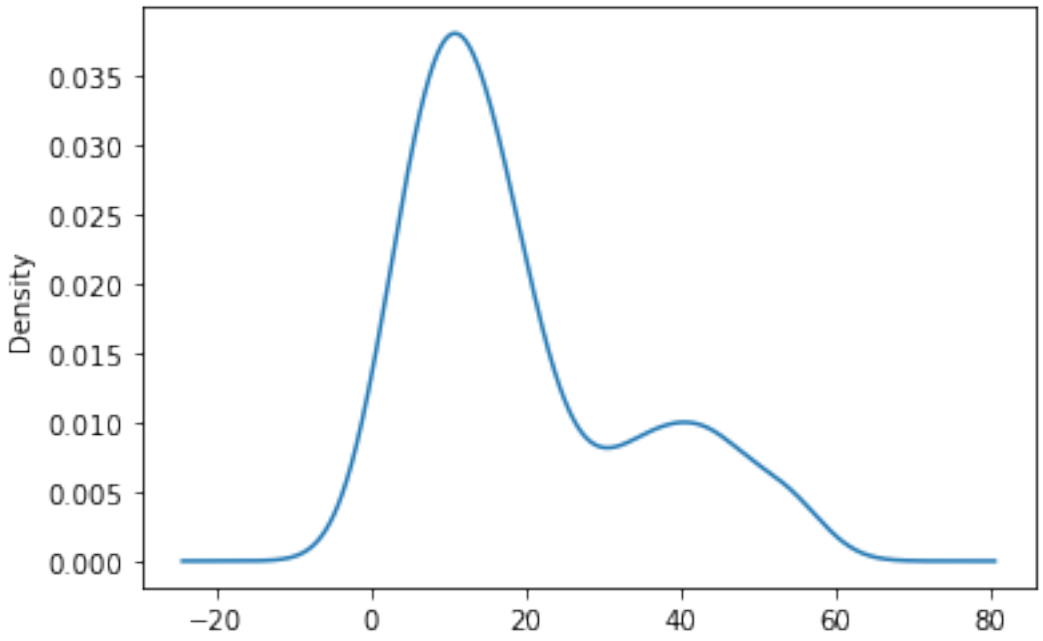

```
std= 4.61   mean= 2.52
count of available values= 231   count of missing values= 677   count of unique
values= 31
available values/total data = 0.254   unique values/available values = 0.134
################################################################################
```

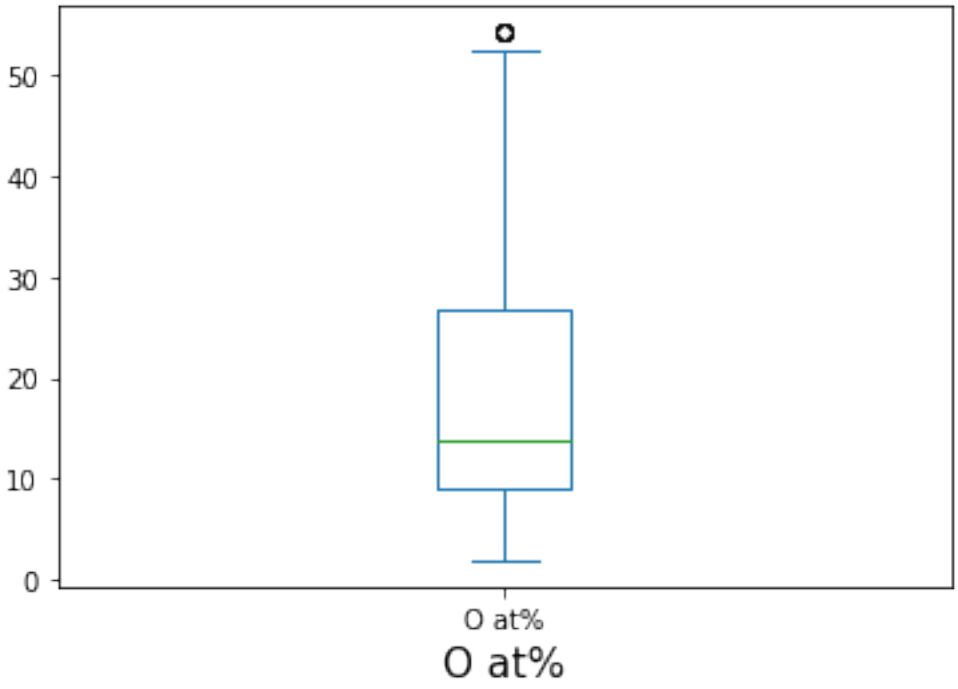




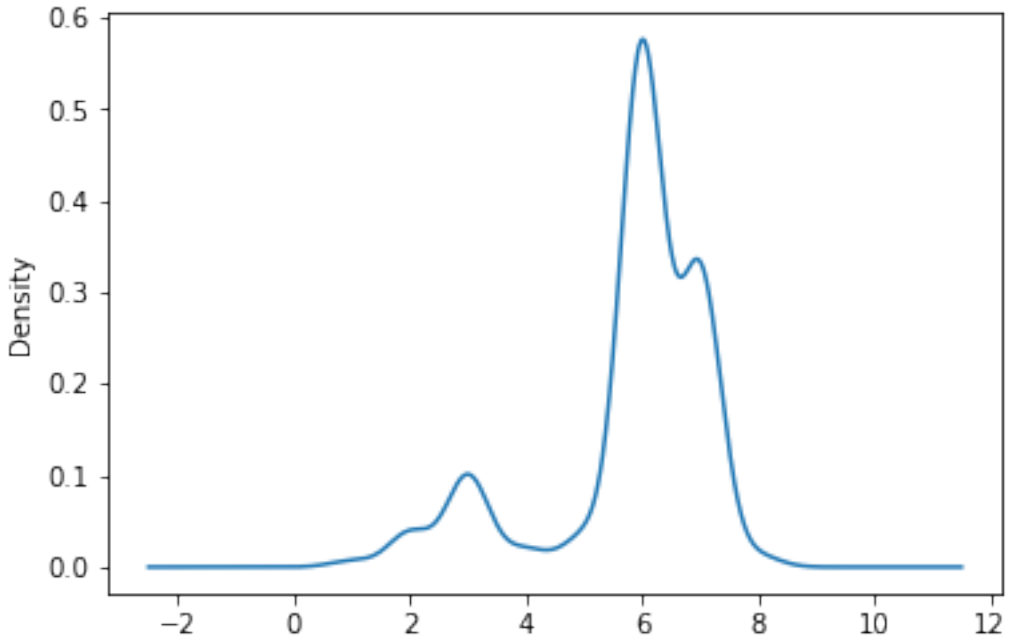
std= 28.53 mean= 66.65
count of available values= 222 count of missing values= 686 count of unique values= 67

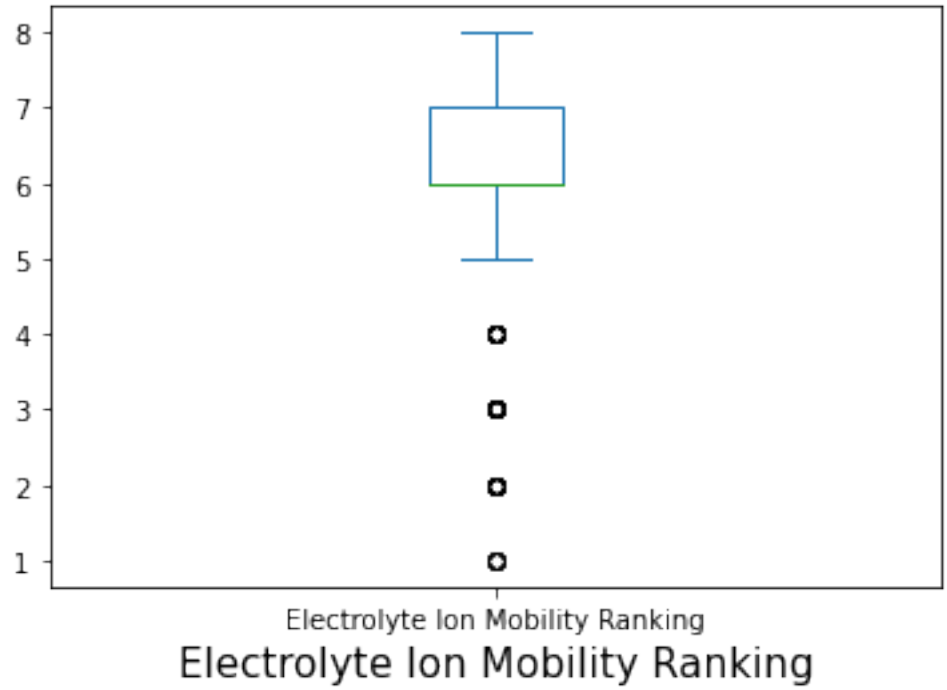
```
available values/total data = 0.244   unique values/available values = 0.302
################################################################################
```





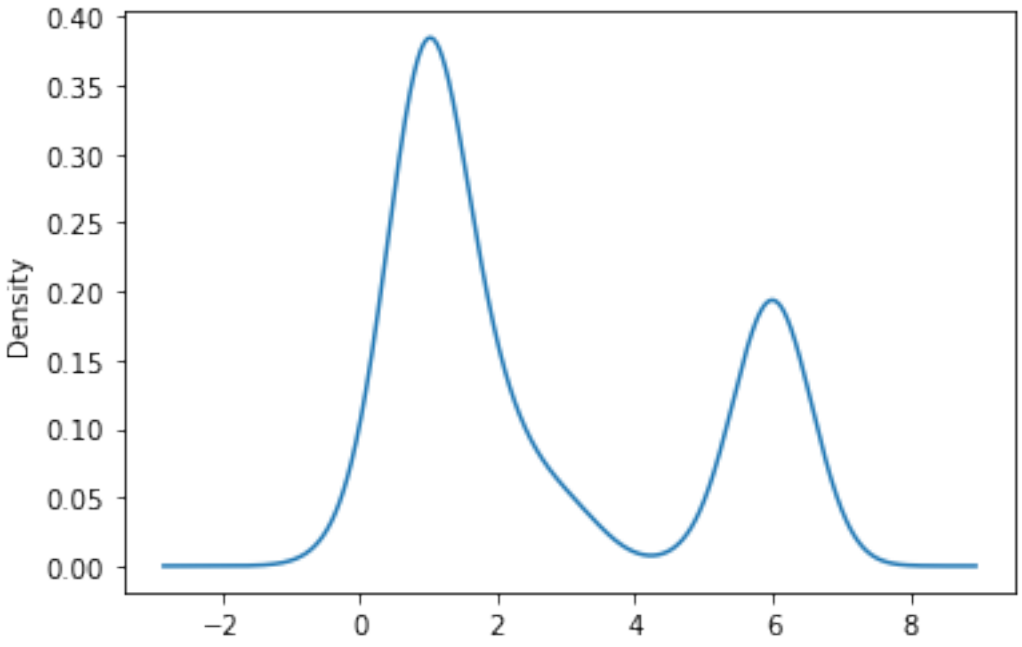
```
std= 14.49   mean= 19.17
count of available values= 218   count of missing values= 690   count of unique
values= 68
available values/total data = 0.240   unique values/available values = 0.312
################################################################################
```

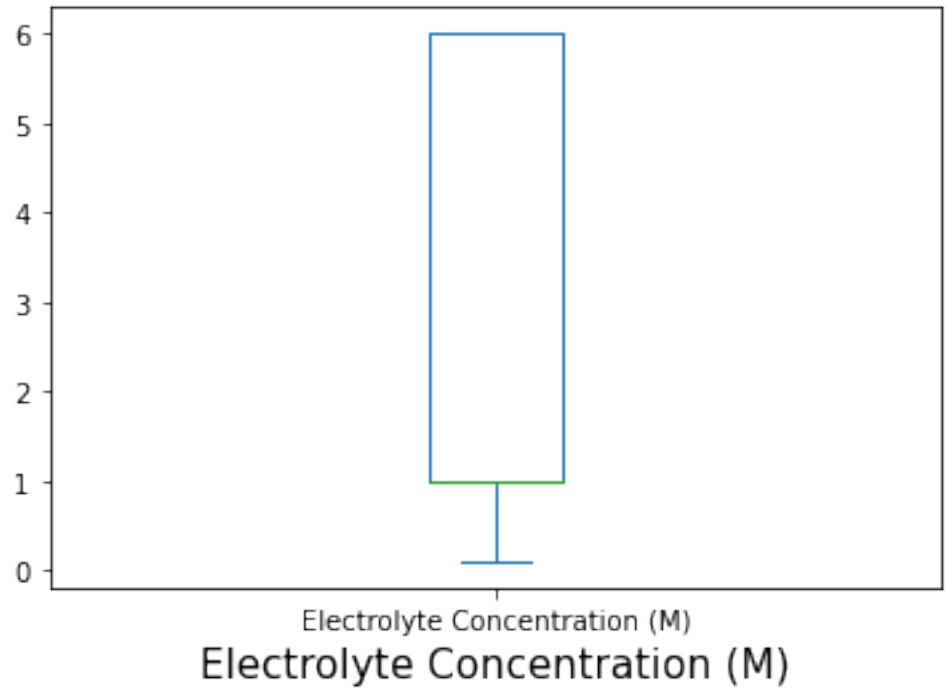




Electrolyte Ion Mobility Ranking

std= 1.39 mean= 5.81
count of available values= 811 count of missing values= 97 count of unique values= 8
available values/total data = 0.893 unique values/available values = 0.010
##





std= 2.19 mean= 2.60
count of available values= 848 count of missing values= 60 count of unique values= 8

```
available values/total data = 0.934   unique values/available values = 0.009
################################################################################
```

[26]:
```
fillingMissingValues(dfEncoded)
```

```
column=  Lower Limit of Potential Window (V)   available ratio= 0.999   unique
ratio= 0.017
mean= -0.235   std= 0.369
 missing values filled with -0.3083

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


column=  Upper Limit of Potential Window (V)   available ratio= 0.999   unique
ratio= 0.028
mean= 0.627   std= 0.449
 missing values filled with 0.5372

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


column=  Potential Window (V)   available ratio= 0.998   unique ratio= 0.023
mean= 0.862   std= 0.348
 missing values filled with 0.7920

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


column=  Current Density (A/g)   available ratio= 0.999   unique ratio= 0.072
mean= 5.866   std= 13.367
 missing values filled with 8.5399

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


column=  Electrolyte Ion Mobility Ranking   available ratio= 0.893   unique
ratio= 0.010
```

```
mean= 5.808   std= 1.389
 missing values filled with 5.5299

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


column=  Electrolyte Concentration (M)   available ratio= 0.934   unique ratio=
0.009
mean= 2.595   std= 2.195
 missing values filled with 2.1564

################################################################################
```

[27]:
```
dfEncoded.isnull().sum()
```

[27]:
```
Limits of Potential Window (V)                    0
Lower Limit of Potential Window (V)               0
Upper Limit of Potential Window (V)               0
Potential Window (V)                              0
Current Density (A/g)                             0
Capacitance (F/g)                                 0
Specific Surface Area (m^2/g)                   561
Charge Transfer Resistance (Rct) (ohm)          771
Equivalent Series Resistance (Rs) (ohm)         757
Electrode Material                                0
Pore Size (nm)                                  755
Pore Volume (cm^3/g)                            713
Ratio of ID/IG                                  580
N at%                                           677
C at%                                           686
O at%                                           690
Electrolyte Chemical Formula                      0
Electrolyte Ion Mobility Ranking                  0
Electrolyte Concentration (M)                     0
Cell Configuration (three/two electrode system)   0
dtype: int64
```

[28]:
```
EmptyColumnNames(dfEncoded)
```

[28]:
```
['Specific Surface Area (m^2/g)',
 'Charge Transfer Resistance (Rct) (ohm)',
 'Equivalent Series Resistance (Rs) (ohm)',
```

```
'Pore Size (nm)',
'Pore Volume (cm^3/g)',
'Ratio of ID/IG',
'N at%',
'C at%',
'O at%']
```

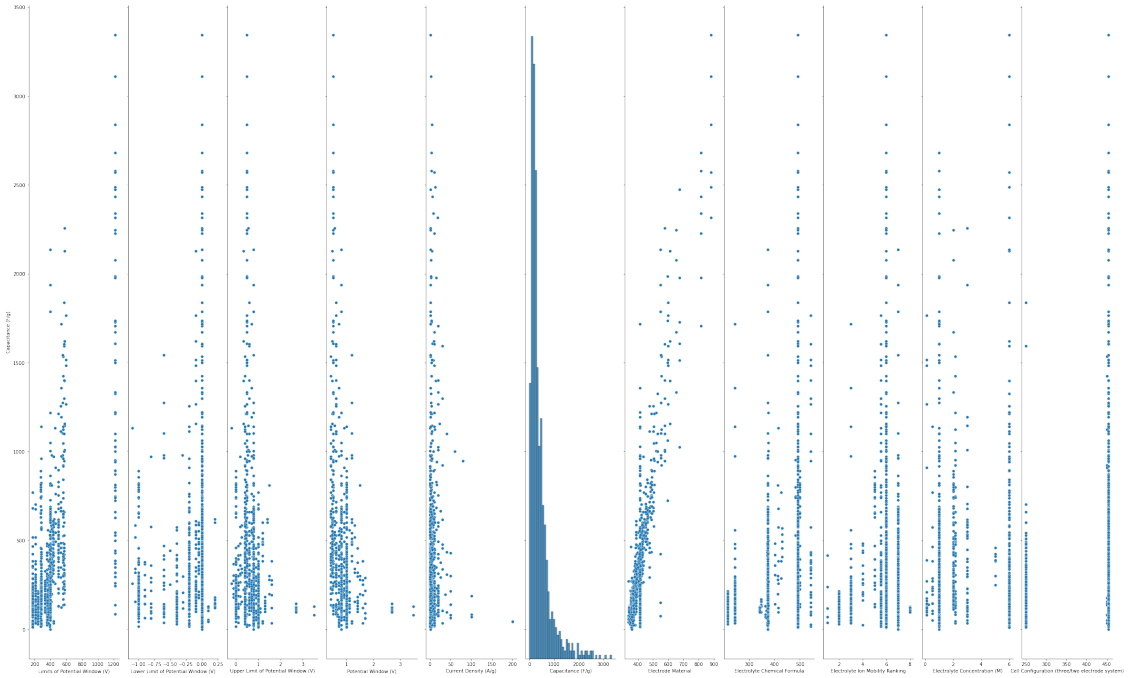
[29]:
```
df=dfEncoded.drop(columns=EmptyColumnNames(dfEncoded),axis=1)
```

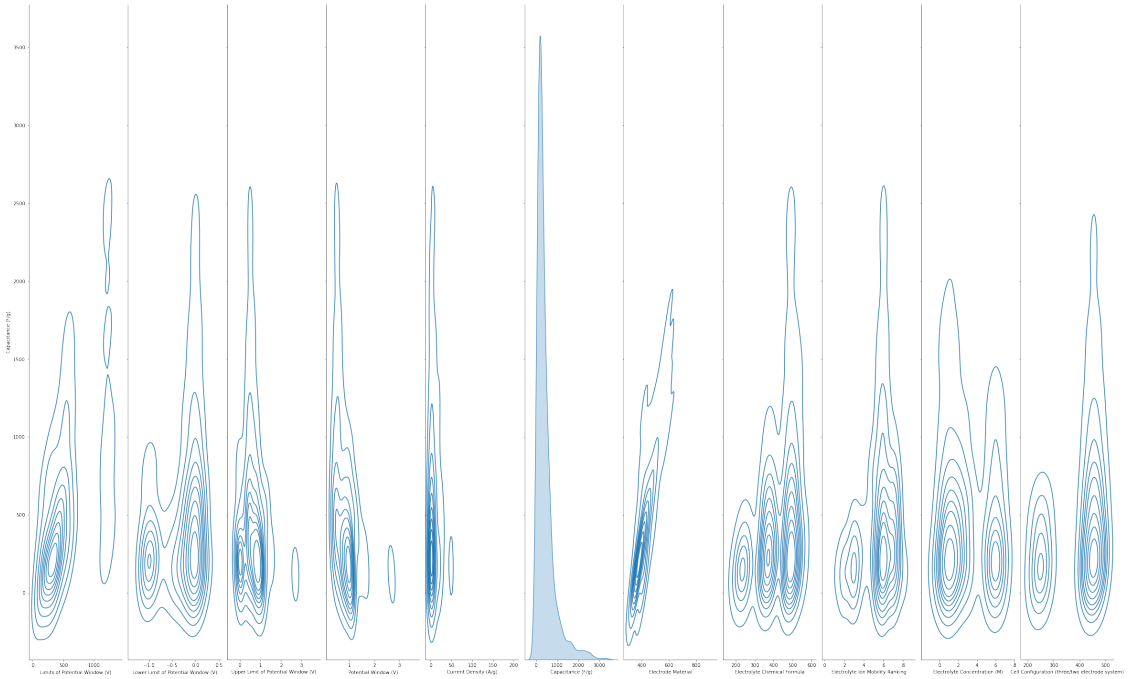
[30]:
```
df.shape
```

[30]:
```
(908, 11)
```

[31]:
```
sns.pairplot(df,y_vars='Capacitance (F/g)',aspect=3/20,height=20)

sns.pairplot(df,kind='kde',y_vars='Capacitance (F/g)',aspect=3/20,height=20)
```

[31]:
```
<seaborn.axisgrid.PairGrid at 0x7f7a077628b0>
```





[32]:
```
df.columns
```

[32]:
```
Index(['Limits of Potential Window (V)', 'Lower Limit of Potential Window (V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)', 'Capacitance (F/g)', 'Electrode Material',
       'Electrolyte Chemical Formula', 'Electrolyte Ion Mobility Ranking',
       'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)'],
      dtype='object')
```

[33]:
```
df.isnull().sum()
```

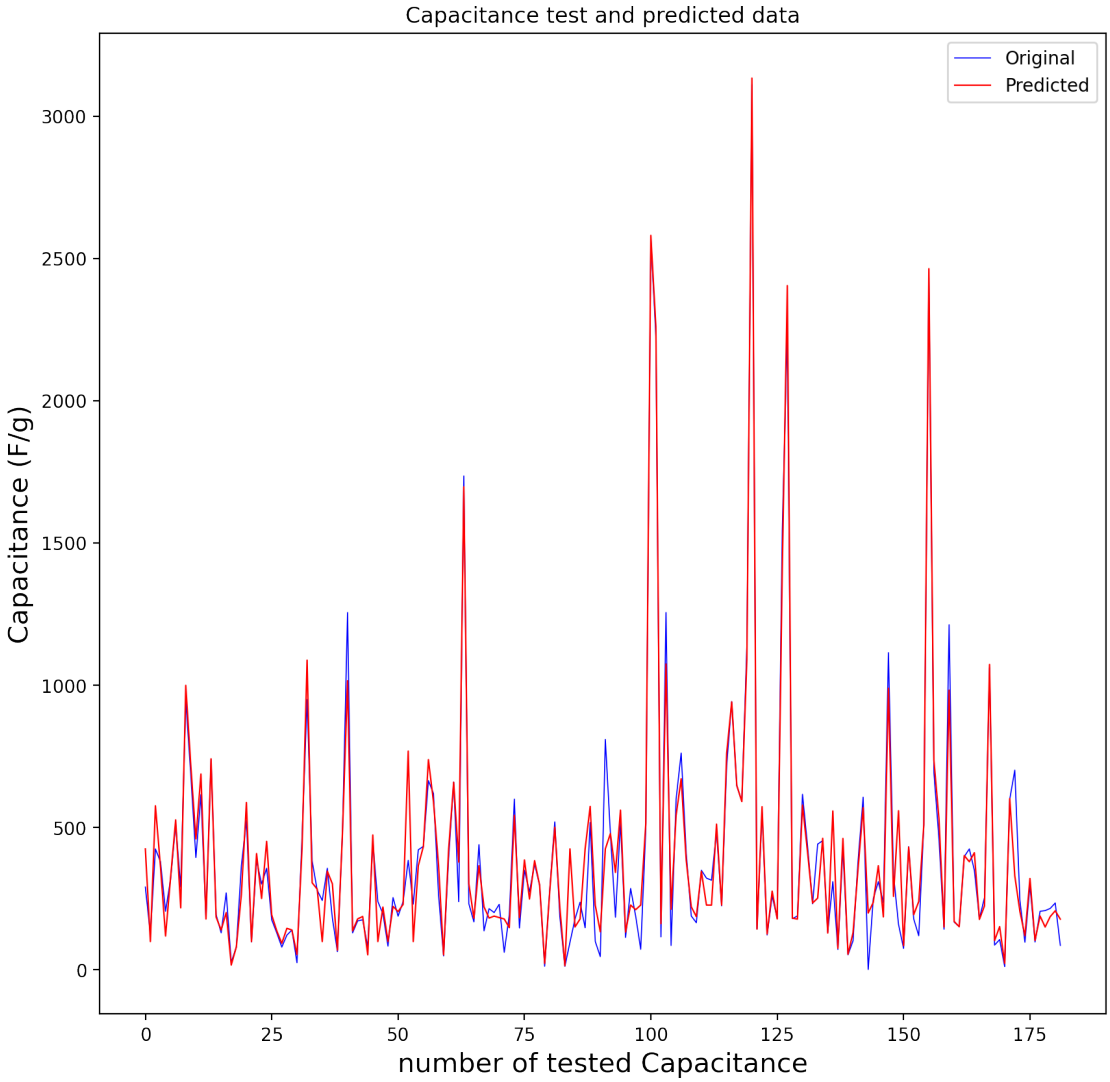
[33]:
```
Limits of Potential Window (V)                  0
Lower Limit of Potential Window (V)             0
Upper Limit of Potential Window (V)             0
Potential Window (V)                            0
Current Density (A/g)                           0
Capacitance (F/g)                               0
Electrode Material                              0
Electrolyte Chemical Formula                    0
Electrolyte Ion Mobility Ranking                0
Electrolyte Concentration (M)                   0
Cell Configuration (three/two electrode system) 0
dtype: int64
```

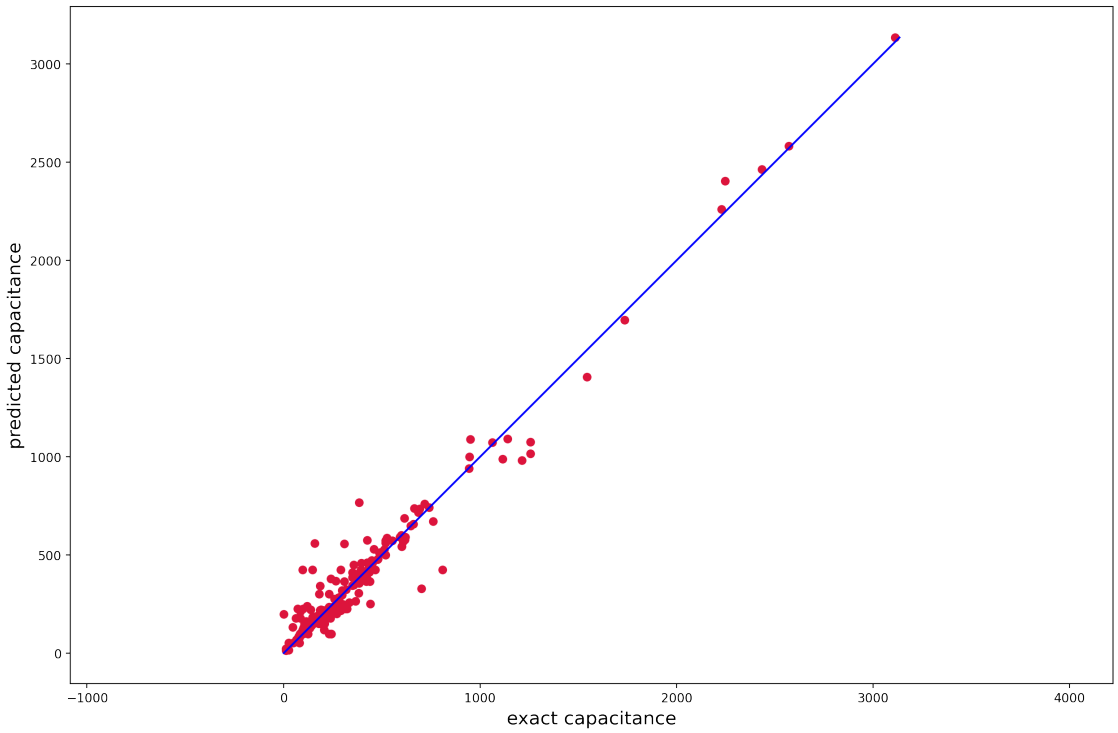
[34]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window␣
 ↪(V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)', 'Electrode Material',
       'Electrolyte Chemical Formula', 'Electrolyte Ion Mobility Ranking',
       'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']



X=df[Predictors].values
y=df[TargetVariable]







PredictorScaler=StandardScaler()


PredictorScalerFit=PredictorScaler.fit(X)


X=PredictorScalerFit.transform(X)
y=y.values.ravel()









opted=ExtraTreesRegressionScore()
```

[35]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],␣
 ↪random_state=opted[2])
etr=ExtraTreesRegressor(n_jobs=-1)
etr.fit(X_train,y_train)


Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)
```

```
mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)


TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions


from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()




plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)

print('\n')
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].head(60))
print(2*'\n')
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
print('number of tested data= %i'%count)
print('test_size"percent"= %.2f'%(opted[0]*100))
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
print('optimized MSE: %.4f ' %mse)
```

```
print('optimized RMSE: %.4f ' %mse**(0.5))
print('optimized random state: %i'%opted[2] )
```





	Capacitance (F/g)	PredictedCapacitance (F/g)
0	291.00	425.166667
1	125.10	99.450000
2	425.00	576.680000
3	380.00	361.173200
4	206.00	119.000000
5	319.24	325.494800
6	512.30	527.165000
7	290.90	218.100000
8	947.00	1000.000000
9	685.00	717.716000
10	395.00	460.332200
11	616.00	688.366000
12	181.00	178.821000
13	740.00	741.806000
14	194.38	185.468800
15	130.10	140.070000
16	270.50	201.224000
17	27.00	17.000000
18	80.00	80.380900
19	366.00	264.811000
20	525.00	588.249600

21	98.00	99.360000
22	384.60	408.680000
23	301.79	250.892200
24	357.00	451.733300
25	176.00	193.048200
26	130.00	137.310800
27	80.00	93.300000
28	122.00	146.000000
29	140.00	140.000000
30	25.00	53.806000
31	466.00	425.166667
32	950.00	1088.610000
33	382.94	306.052200
34	280.00	283.723333
35	244.00	99.450000
36	357.00	347.107200
37	182.00	302.468000
38	64.00	69.403000
39	480.78	480.780000
40	1256.00	1016.404000
41	130.00	137.144900
42	172.00	179.087200
43	176.00	187.669700
44	80.00	52.950800
45	449.00	474.059700
46	240.00	99.450000
47	197.10	220.607000
48	83.40	99.450000
49	254.00	223.077200
50	188.80	204.813000
51	240.00	230.135000
52	385.00	769.000000
53	231.00	99.450000
54	422.00	366.000000
55	434.00	433.400000
56	665.00	739.000000
57	620.00	591.250000
58	265.00	369.370000
59	49.00	52.888400

```
total number of data= 908
number of trained data= 727
number of tested data= 181
test_size"percent"= 20.00
score for xtest and ytest : 0.9578
cross validation score: 0.8594
```

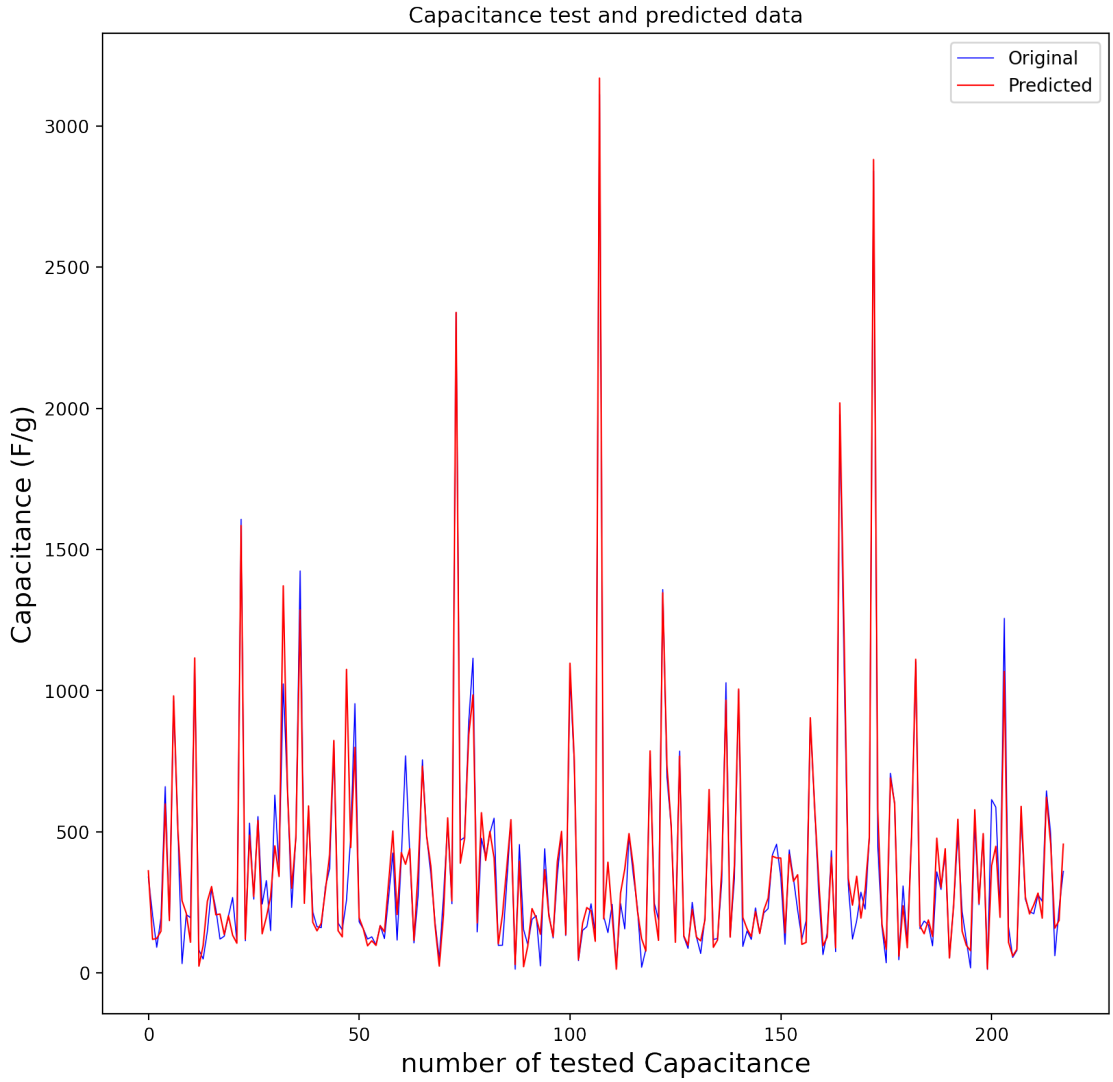
```
optimized MSE: 8901.3019
optimized RMSE: 94.3467
optimized random state: 4
```

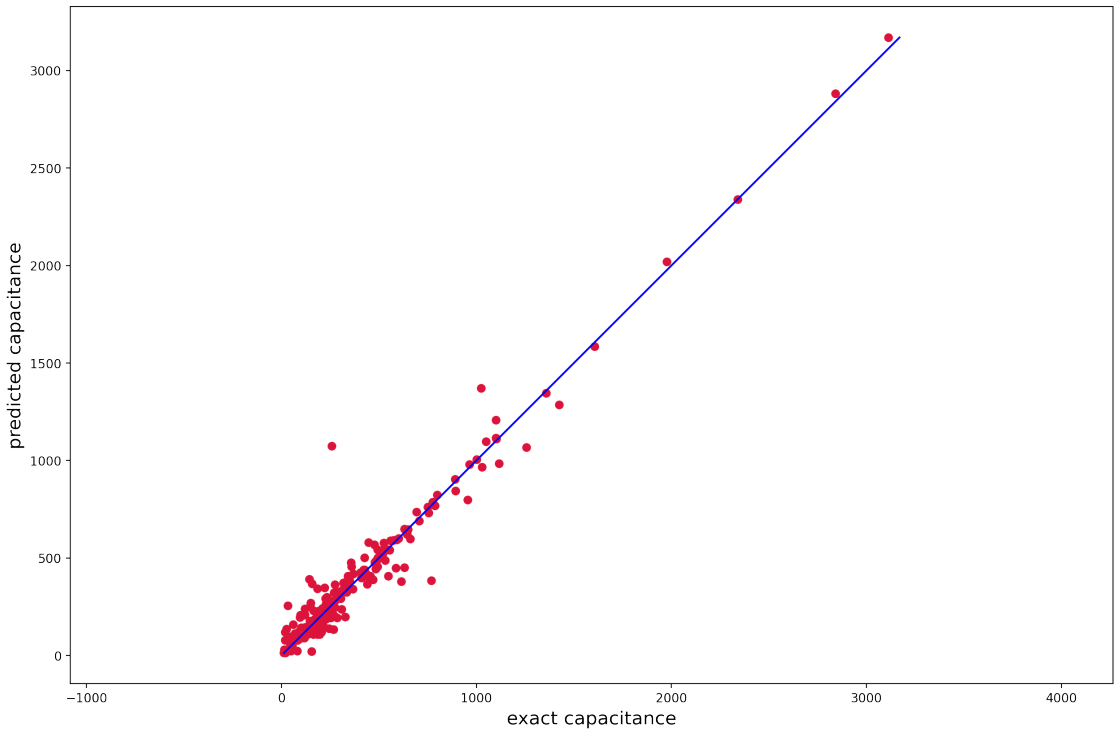
[36]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window␣
 ↪(V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)', 'Electrode Material',
       'Electrolyte Chemical Formula', 'Electrolyte Ion Mobility Ranking',
       'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']



X=df[Predictors].values
y=df[TargetVariable]




PredictorScaler=StandardScaler()
PredictorScalerFit=PredictorScaler.fit(X)


X=PredictorScalerFit.transform(X)
y=y.values.ravel()









opted=ExtraTreesRegressionMse()
```

[37]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],␣
 ↪random_state=opted[2])
etr=ExtraTreesRegressor(n_jobs=-1)
etr.fit(X_train,y_train)


Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)
```

```
mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)


TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions


from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()




plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)

print('\n')
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].head(60))
print(2*'\n')
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
print('number of tested data= %i'%count)
print('test_size"percent"= %.2f'%(opted[0]*100))
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
```

```
print('optimized MSE: %.4f ' %mse)
print('optimized RMSE: %.4f ' %mse**(0.5))
print('optimized random state: %i'%opted[2] )
```





	Capacitance (F/g)	PredictedCapacitance (F/g)
0	339.88	361.8045
1	206.00	119.0000
2	90.90	122.2240
3	195.50	147.8690
4	660.00	598.8000
5	187.00	185.6350
6	964.00	981.3100
7	509.85	509.8500
8	33.00	256.0000
9	207.00	211.3179
10	195.60	108.8000
11	1100.00	1115.7700
12	81.14	24.2920
13	49.40	97.5570
14	147.00	253.7130
15	301.79	306.0680
16	228.00	205.9041
17	120.00	208.6770
18	130.00	136.3200
19	201.50	202.5300
20	267.39	133.5079

21	112.00	105.3510
22	1607.00	1584.8000
23	113.90	119.8710
24	530.70	487.9312
25	261.40	270.8340
26	554.00	539.9020
27	244.00	138.9000
28	327.00	197.7790
29	150.00	270.1700
30	630.00	450.2258
31	367.00	341.8000
32	1024.00	1371.2707
33	650.00	647.3300
34	232.00	300.2038
35	485.00	478.6194
36	1424.00	1285.8200
37	274.00	246.6420
38	575.00	591.5640
39	216.30	179.5274
40	164.00	149.3120
41	160.30	175.8702
42	305.00	292.1660
43	369.00	419.5840
44	799.00	823.2800
45	177.20	149.9640
46	152.00	127.8800
47	258.00	1075.3178
48	483.00	444.2270
49	954.00	799.2000
50	181.00	195.5680
51	156.00	155.0900
52	120.00	96.1600
53	128.00	114.1390
54	100.00	97.5820
55	168.00	167.3200
56	122.00	146.0000
57	270.00	322.3260
58	425.00	502.6596
59	116.40	207.0000

```
total number of data= 908
number of trained data= 691
number of tested data= 217
test_size"percent"= 24.00
score for xtest and ytest : 0.9529
cross validation score: 0.8621
```

```
optimized MSE: 8270.5400
optimized RMSE: 90.9425
optimized random state: 10
```

[38]:
```
df.shape
```

[38]:
```
(908, 11)
```

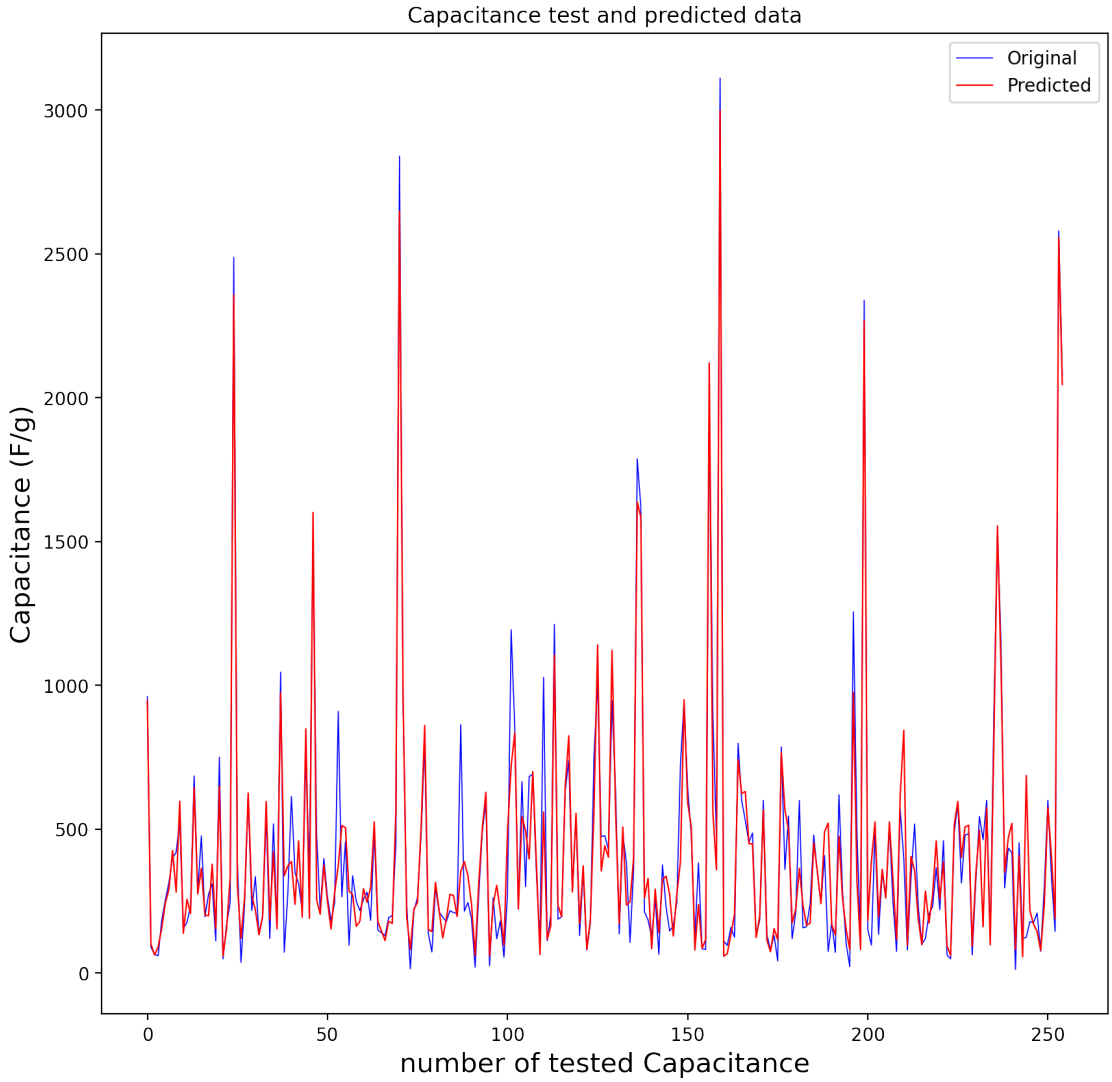
the following model is GBR on the latter df; df6

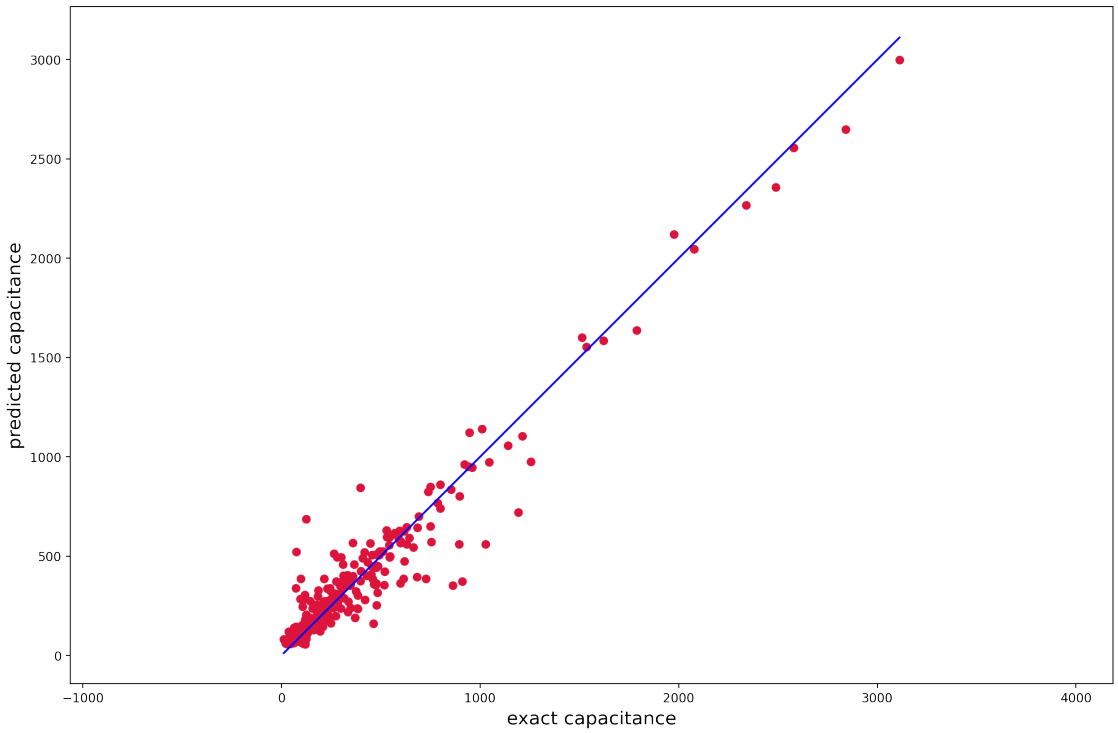
[39]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window␣
 ↪(V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)', 'Electrode Material',
       'Electrolyte Chemical Formula', 'Electrolyte Ion Mobility Ranking',
       'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']



X=df[Predictors].values
y=df[TargetVariable]




PredictorScaler=StandardScaler()
PredictorScalerFit=PredictorScaler.fit(X)



X=PredictorScalerFit.transform(X)
y=y.values.ravel()






opted=GradientBoostRegScore()
```

[40]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],␣
 ↪random_state=opted[2])
etr=GradientBoostingRegressor(random_state=opted[2])
etr.fit(X_train,y_train)
```

```
Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)


mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)

TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions


from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()




plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)
print('\n')
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].head(60))
print(2*'\n')
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
print('number of tested data= %i'%count)
print('test_size"percent"= %.2f'%(opted[0]*100))
```

```
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
print('optimized MSE: %.4f ' %mse)
print('optimized RMSE: %.4f ' %mse**(0.5))
print('optimized random state: %.4f'%opted[2] )
```





	Capacitance (F/g)	PredictedCapacitance (F/g)
0	961.00	945.886207
1	90.00	101.304775
2	65.00	62.782737
3	60.00	89.979144
4	184.00	155.807954
5	253.80	242.695677
6	315.00	290.279744
7	400.00	425.817949
8	420.00	281.436803
9	530.70	597.390440
10	154.40	138.212338
11	176.00	256.253538
12	232.10	206.694071
13	685.00	643.736335
14	286.00	275.295734
15	477.00	364.015595
16	196.00	201.687249
17	274.00	199.147762
18	310.00	378.631222
19	112.00	153.945992
20	750.00	649.890017

21	49.00	59.456875
22	168.70	154.460386
23	245.00	332.254522
24	2488.00	2358.211373
25	384.60	303.881305
26	37.00	120.520701
27	237.60	260.654265
28	616.00	625.713406
29	217.11	270.334656
30	335.00	218.719389
31	134.00	133.094344
32	192.00	198.128562
33	545.00	596.323580
34	120.00	184.496916
35	519.00	421.808680
36	166.00	153.945992
37	1046.00	974.698934
38	72.30	338.215226
39	275.00	372.608561
40	614.00	387.720492
41	349.00	238.975042
42	310.00	459.283850
43	215.80	194.057923
44	750.00	849.290215
45	370.00	190.115594
46	1514.00	1601.267652
47	480.00	254.251775
48	218.10	204.786117
49	398.00	375.085264
50	265.00	249.201573
51	181.00	153.125488
52	245.00	280.268976
53	910.00	373.592333
54	265.00	512.592717
55	457.00	505.454021
56	96.16	284.623395
57	338.00	272.512522
58	250.00	162.372613
59	217.20	179.471272

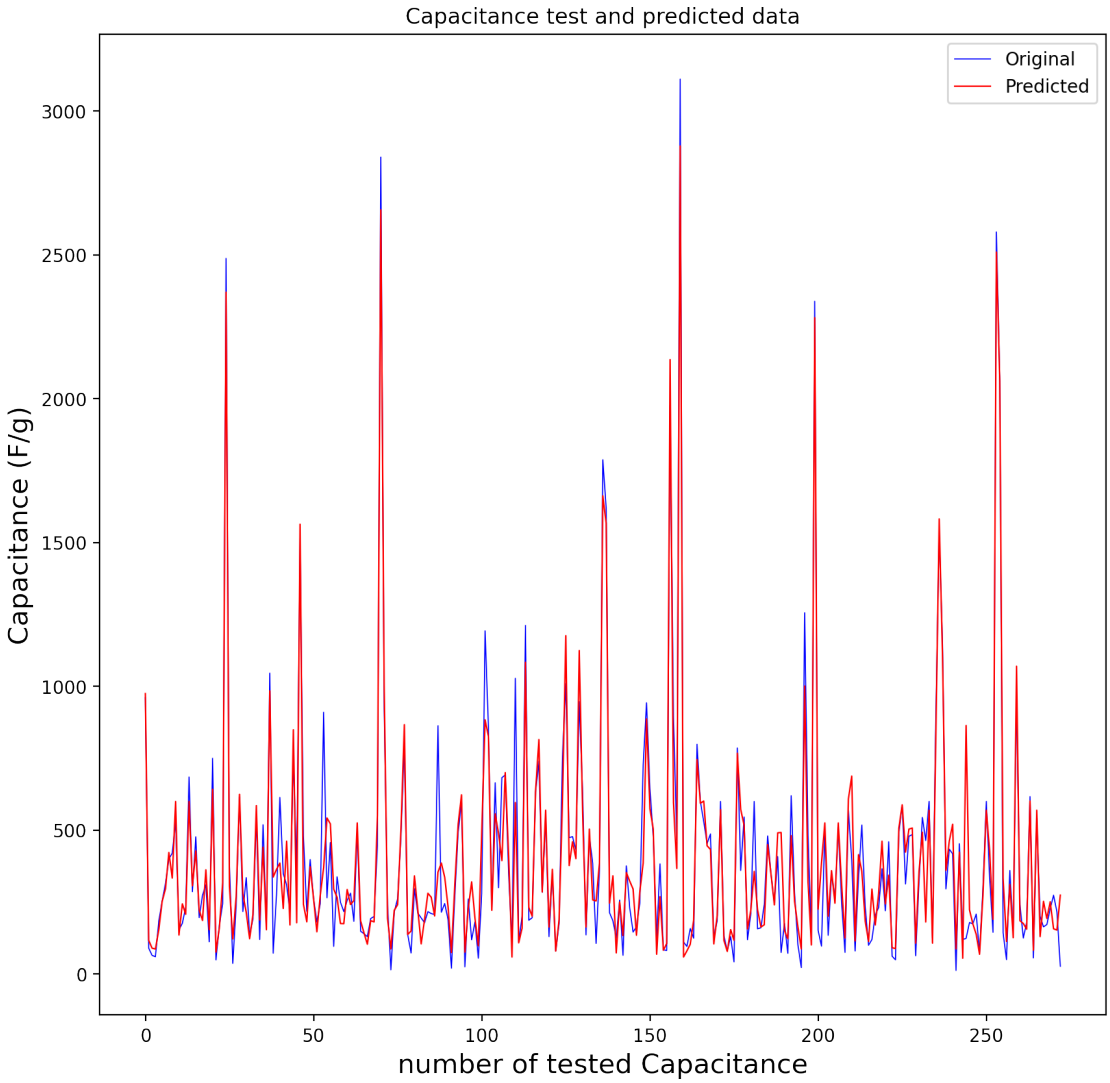
```
total number of data= 908
number of trained data= 654
number of tested data= 254
test_size"percent"= 28.00
score for xtest and ytest : 0.9296
cross validation score: 0.8328
```

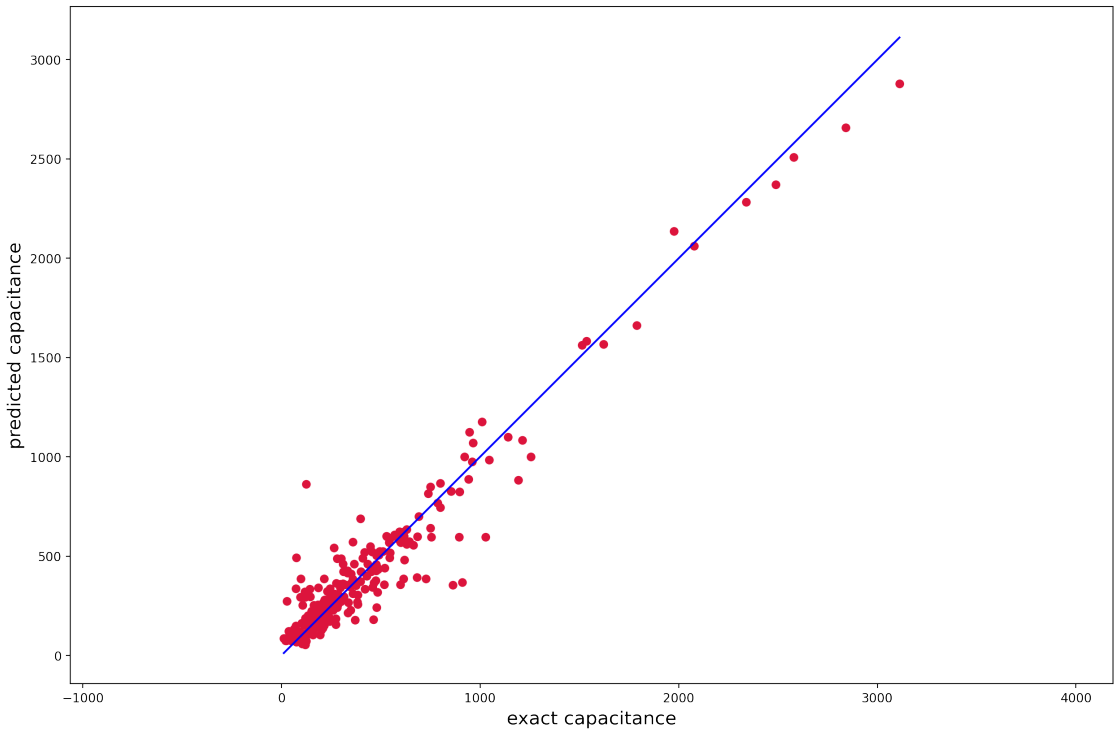
```
optimized MSE: 14970.5532
optimized RMSE: 122.3542
optimized random state: 9.0000
```

[41]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window␣
 ↪(V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)', 'Electrode Material',
       'Electrolyte Chemical Formula', 'Electrolyte Ion Mobility Ranking',
       'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']



X=df[Predictors].values
y=df[TargetVariable]




PredictorScaler=StandardScaler()
PredictorScalerFit=PredictorScaler.fit(X)


X=PredictorScalerFit.transform(X)
y=y.values.ravel()




opted=GradientBoostRegMse()
```

[42]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],␣
 ↪random_state=opted[2])
etr=GradientBoostingRegressor(random_state=opted[2])
etr.fit(X_train,y_train)


Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)


mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)
```

```
TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions


from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()




plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)
print('\n')
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].head(60))
print(2*'\n')
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
print('number of tested data= %i'%count)
print('test_size"percent"= %.2f'%(opted[0]*100))
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
print('optimized MSE: %.4f ' %mse)
print('optimized RMSE: %.4f ' %mse**(0.5))
print('optimized random state: %.4f'%opted[2] )
```


Capacitance test and predicted data
Original
Predicted
Capacitance (F/g)
number of tested Capacitance
0
500
1000
1500
2000
2500
3000
0
50
100
150
200
250



	Capacitance (F/g)	PredictedCapacitance (F/g)
0	961.00	975.003499
1	90.00	117.383456
2	65.00	89.870296
3	60.00	86.951263
4	184.00	153.457808
5	253.80	252.415733
6	315.00	296.593161
7	400.00	422.557864
8	420.00	333.810465
9	530.70	599.844036
10	154.40	135.575399
11	176.00	243.908922
12	232.10	207.761151
13	685.00	599.669243
14	286.00	306.706766
15	477.00	428.094277
16	196.00	223.005441
17	274.00	186.245018
18	310.00	362.330529
19	112.00	154.057886
20	750.00	642.214907

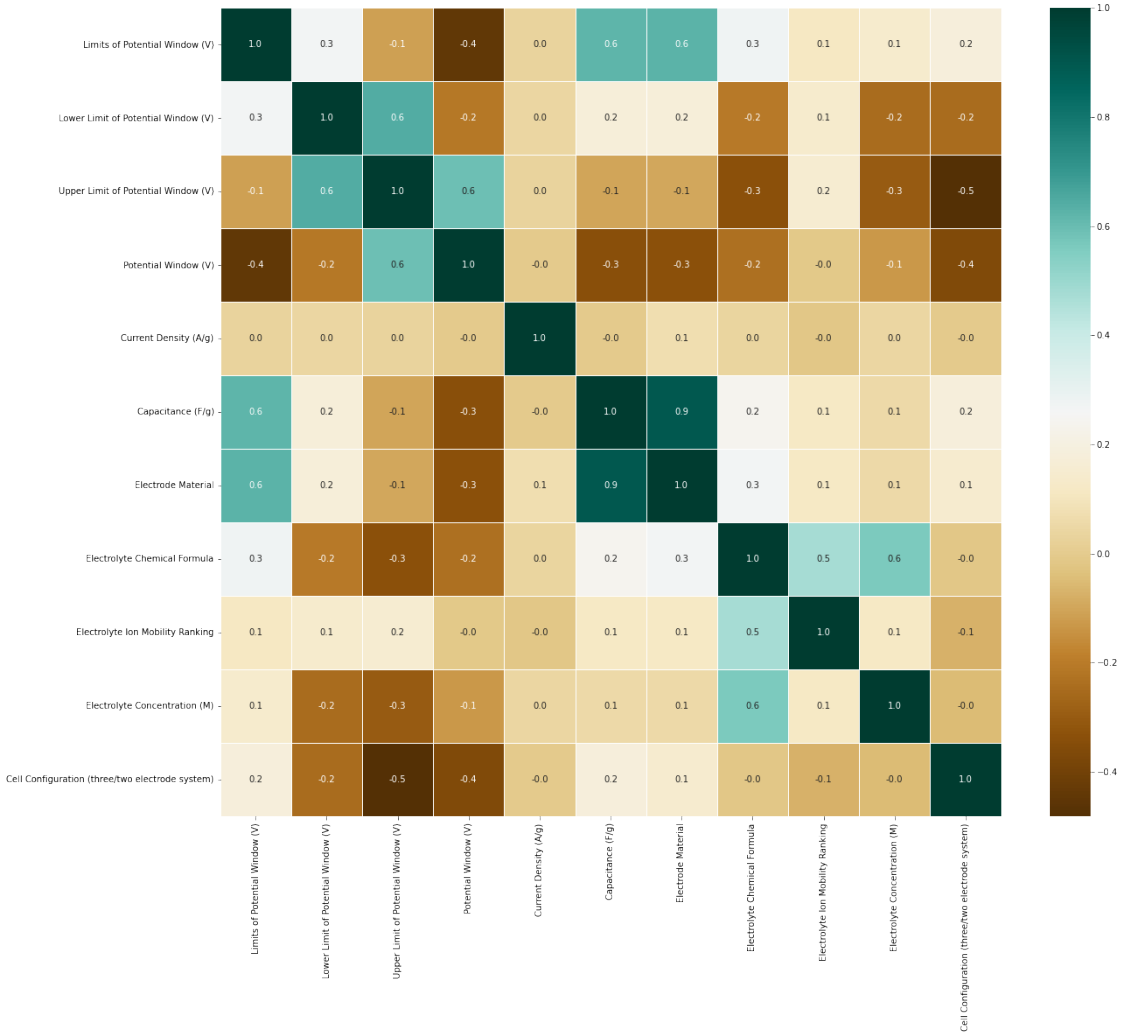
21	49.00	76.139889
22	168.70	160.086903
23	245.00	317.160291
24	2488.00	2370.608921
25	384.60	305.239230
26	37.00	122.003914
27	237.60	268.785383
28	616.00	624.768366
29	217.11	278.611801
30	335.00	214.244925
31	134.00	122.786893
32	192.00	206.846292
33	545.00	585.578162
34	120.00	187.701002
35	519.00	440.655037
36	166.00	154.057886
37	1046.00	984.570229
38	72.30	336.936774
39	275.00	363.967816
40	614.00	385.783100
41	349.00	228.050069
42	310.00	461.816782
43	215.80	170.549849
44	750.00	849.214010
45	370.00	178.233664
46	1514.00	1563.880943
47	480.00	242.298776
48	218.10	181.591720
49	398.00	372.877027
50	265.00	262.990158
51	181.00	146.815755
52	245.00	273.512180
53	910.00	369.327423
54	265.00	542.605633
55	457.00	522.170019
56	96.16	295.264184
57	338.00	266.723786
58	250.00	175.658619
59	217.20	175.042956

```
total number of data= 908
number of trained data= 636
number of tested data= 272
test_size"percent"= 30.00
score for xtest and ytest : 0.9295
cross validation score: 0.8397
```

```
optimized MSE: 14351.4027
optimized RMSE: 119.7973
optimized random state: 9.0000
```

[43]:
```
plt.figure(figsize=(20,17))
heatmap(df.corr(),annot=True,fmt='.1f',linewidths=0.5,cmap='BrBG')
```

[43]: <AxesSubplot:>



[44]:
```
dfEncoded.shape
```

[44]: (908, 20)

[45]:
```
LargeNanColumns=[]
for column in dfEncoded.columns:
    if dfEncoded[column].isnull().sum()>0:
```

```
        LargeNanColumns.append(column)
```

[46]:
```
LargeNanColumns
```

[46]:
```
['Specific Surface Area (m^2/g)',
 'Charge Transfer Resistance (Rct) (ohm)',
 'Equivalent Series Resistance (Rs) (ohm)',
 'Pore Size (nm)',
 'Pore Volume (cm^3/g)',
 'Ratio of ID/IG',
 'N at%',
 'C at%',
 'O at%']
```

[47]:
```
dframe={}
for i in range(1,10):
    df=dfEncoded
    df=df[dfEncoded.isnull().sum(axis=1)<i]
    dframe[i]=df
    print(i,dframe[i].shape)
```

```
1 (0, 20)
2 (16, 20)
3 (51, 20)
4 (77, 20)
5 (197, 20)
6 (304, 20)
7 (340, 20)
8 (414, 20)
9 (583, 20)
```

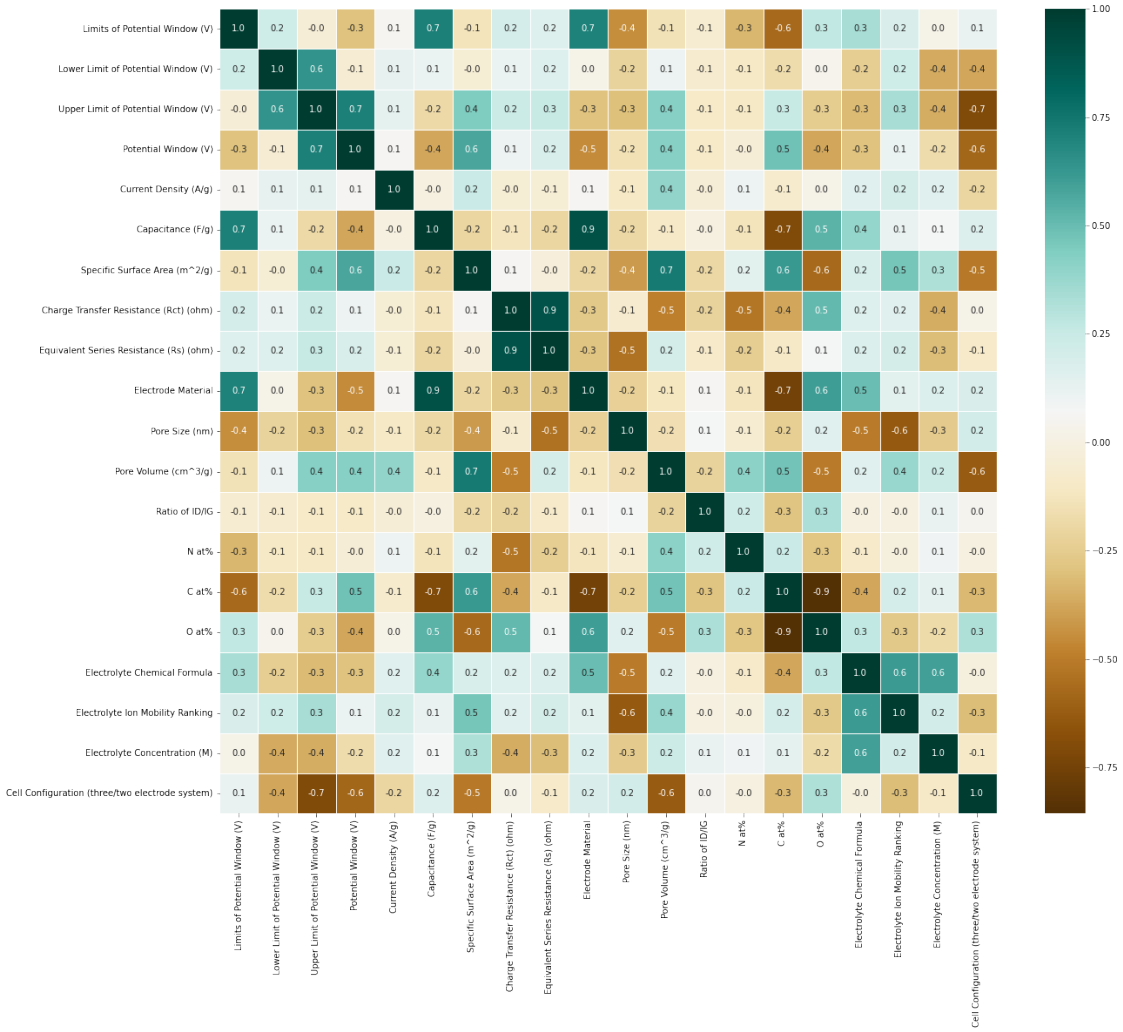
rows with lessthan 5 missing values

[48]:
```
df5=dframe[5]
df5.shape
```

[48]:
```
(197, 20)
```

[49]:
```
plt.figure(figsize=(20,17))
heatmap(df5.corr(),annot=True,fmt='.1f',linewidths=0.5,cmap='BrBG')
```

[49]:
```
<AxesSubplot:>
```



[50]:
```
df5.head()
```

[50]:

	Limits of Potential Window (V)	Lower Limit of Potential Window (V)	Upper Limit of Potential Window (V)	Potential Window (V)
1	207.895441	0.0	1.0	1.0
2	207.895441	0.0	1.0	1.0
3	207.895441	0.0	1.0	1.0
4	207.895441	0.0	1.0	1.0
68	394.311962	0.0	0.8	0.8

	Current Density (A/g)	Capacitance (F/g)	Specific Surface Area (m^2/g)
1	1.0	367.0	537.0
2	2.0	338.0	537.0
3	5.0	283.0	537.0
4	10.0	246.0	537.0
68	0.1	237.6	NaN

	Charge Transfer Resistance (Rct) (ohm)
1	6.10
2	6.10
3	6.10
4	6.10
68	4.11

	Equivalent Series Resistance (Rs) (ohm)	Electrode Material
1	1.95	397.524840
2	1.95	397.524840
3	1.95	397.524840
4	1.95	397.524840
68	NaN	415.498645

	Pore Size (nm)	Pore Volume (cm^3/g)	Ratio of ID/IG	N at%	C at%	O at%
1	NaN	NaN	1.28	0.0	85.60	9.10
2	NaN	NaN	1.28	0.0	85.60	9.10
3	NaN	NaN	1.28	0.0	85.60	9.10
4	NaN	NaN	1.28	0.0	85.60	9.10
68	NaN	NaN	0.17	0.0	84.35	14.56

	Electrolyte Chemical Formula	Electrolyte Ion Mobility Ranking
1	491.038310	6.0
2	491.038310	6.0
3	491.038310	6.0
4	491.038310	6.0
68	372.518745	7.0

	Electrolyte Concentration (M)
1	6.0
2	6.0
3	6.0
4	6.0
68	1.0

	Cell Configuration (three/two electrode system)
1	249.745539
2	249.745539
3	249.745539

```
4                                    249.745539
68                                   454.153769
```

[51]:
```
df5.isnull().sum()
```

[51]:
```
Limits of Potential Window (V)                    0
Lower Limit of Potential Window (V)               0
Upper Limit of Potential Window (V)               0
Potential Window (V)                              0
Current Density (A/g)                             0
Capacitance (F/g)                                 0
Specific Surface Area (m^2/g)                    45
Charge Transfer Resistance (Rct) (ohm)           73
Equivalent Series Resistance (Rs) (ohm)          62
Electrode Material                                0
Pore Size (nm)                                   90
Pore Volume (cm^3/g)                             79
Ratio of ID/IG                                   67
N at%                                            76
C at%                                            76
O at%                                            76
Electrolyte Chemical Formula                      0
Electrolyte Ion Mobility Ranking                  0
Electrolyte Concentration (M)                     0
Cell Configuration (three/two electrode system)   0
dtype: int64
```

[52]:
```
df5.dropna(axis=0,inplace=True,subset=['O at%'])
```

```
/tmp/ipykernel_270313/462258733.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df5.dropna(axis=0,inplace=True,subset=['O at%'])
```

[53]:
```
df5.isnull().sum()
```

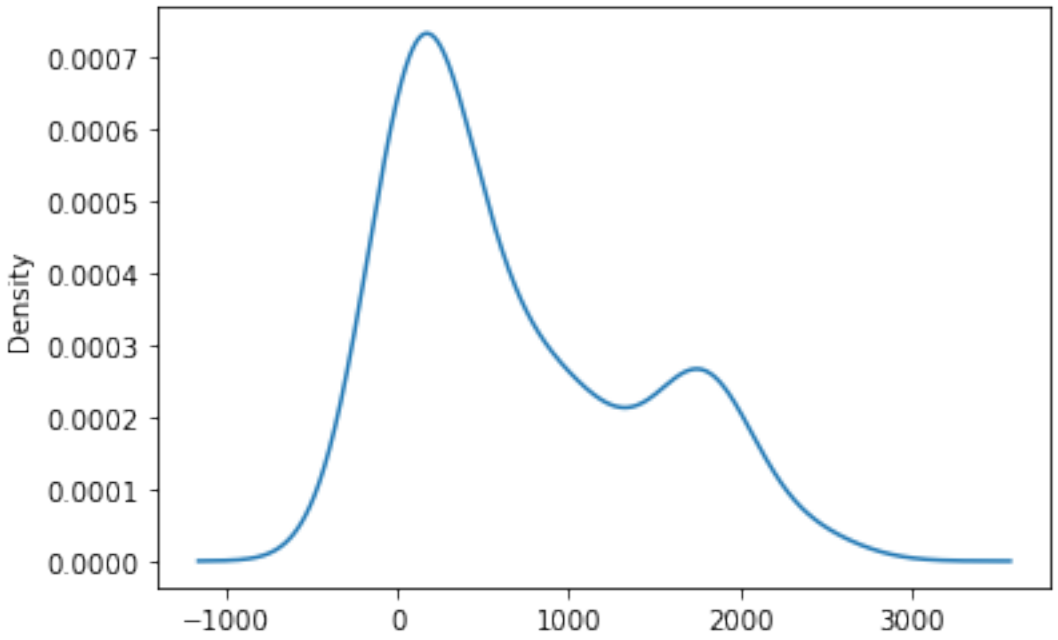
[53]:
```
Limits of Potential Window (V)                    0
Lower Limit of Potential Window (V)               0
Upper Limit of Potential Window (V)               0
Potential Window (V)                              0
Current Density (A/g)                             0
Capacitance (F/g)                                 0
Specific Surface Area (m^2/g)                    45
Charge Transfer Resistance (Rct) (ohm)           69
Equivalent Series Resistance (Rs) (ohm)          42
```

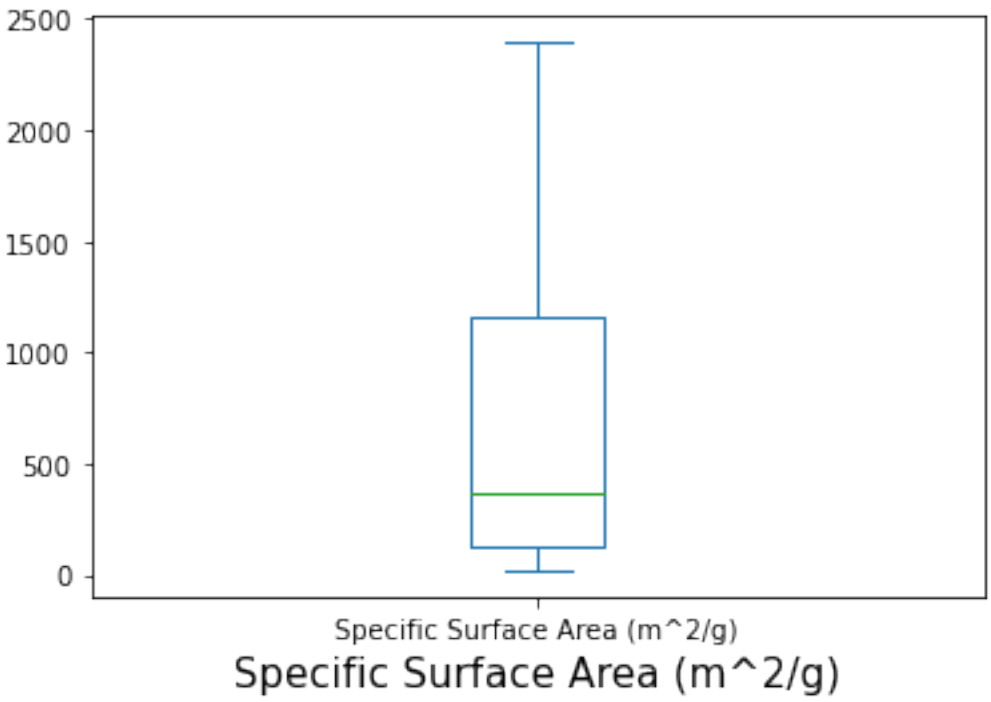
```
Electrode Material                                    0
Pore Size (nm)                                       83
Pore Volume (cm^3/g)                                 72
Ratio of ID/IG                                       41
N at%                                                 0
C at%                                                 0
O at%                                                 0
Electrolyte Chemical Formula                          0
Electrolyte Ion Mobility Ranking                      0
Electrolyte Concentration (M)                         0
Cell Configuration (three/two electrode system)       0
dtype: int64
```

[54]:
```
df5.shape
```

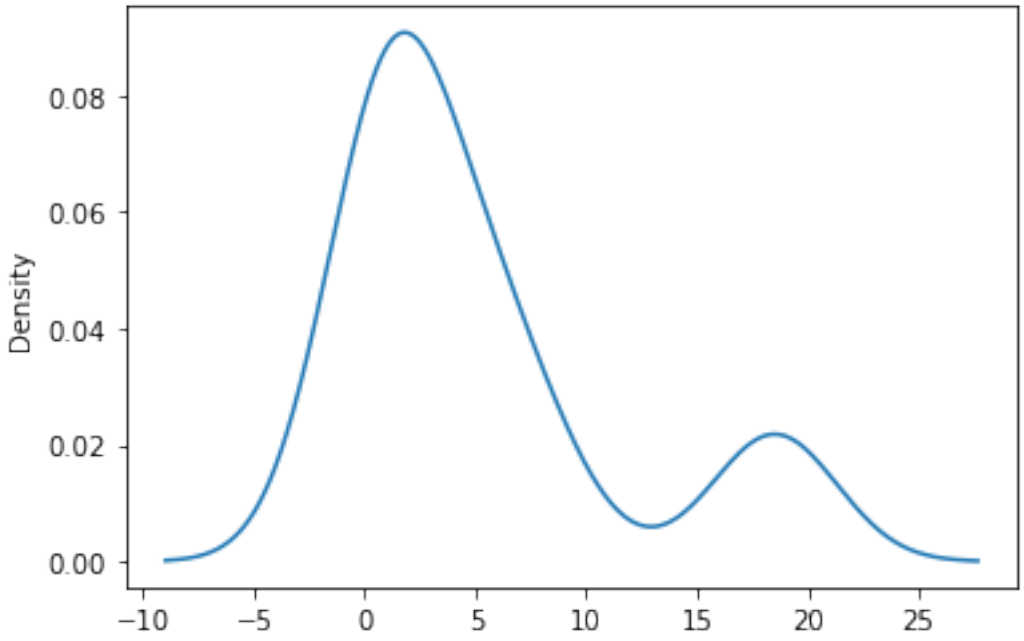
[54]:
```
(121, 20)
```

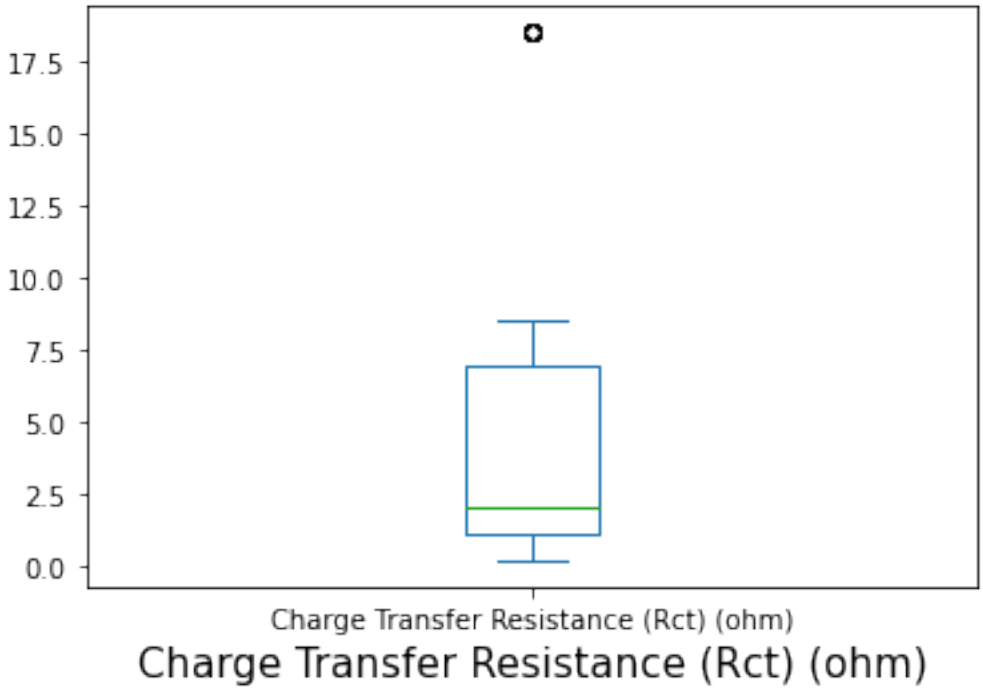
[55]:
```
NullColumnsPlot(df5)
```






```
std= 704.29   mean= 686.44
count of available values= 76   count of missing values= 45   count of unique
values= 25
available values/total data = 0.628   unique values/available values = 0.329
################################################################################
```

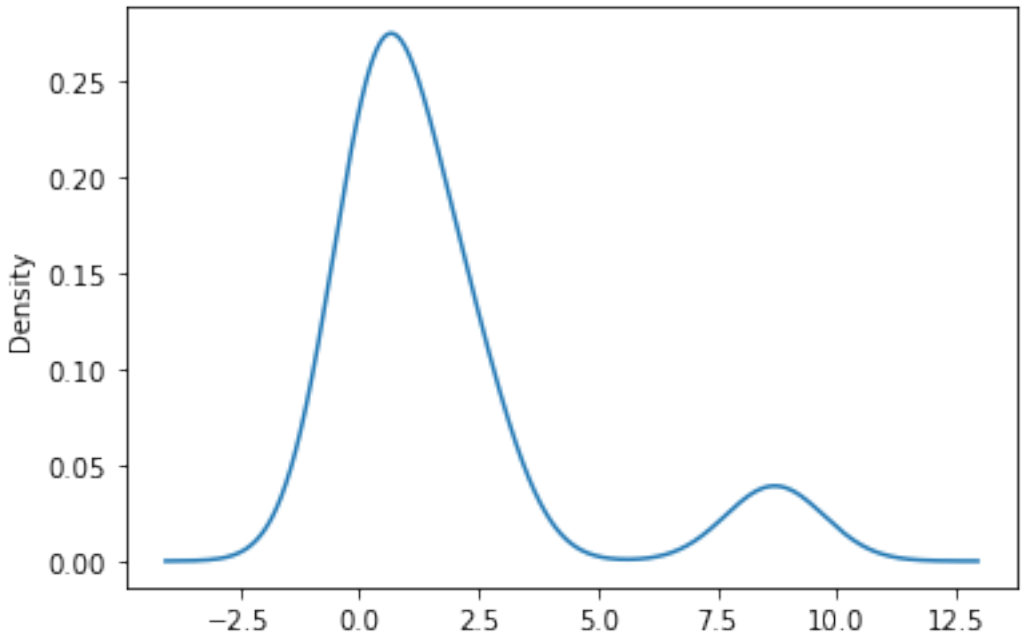


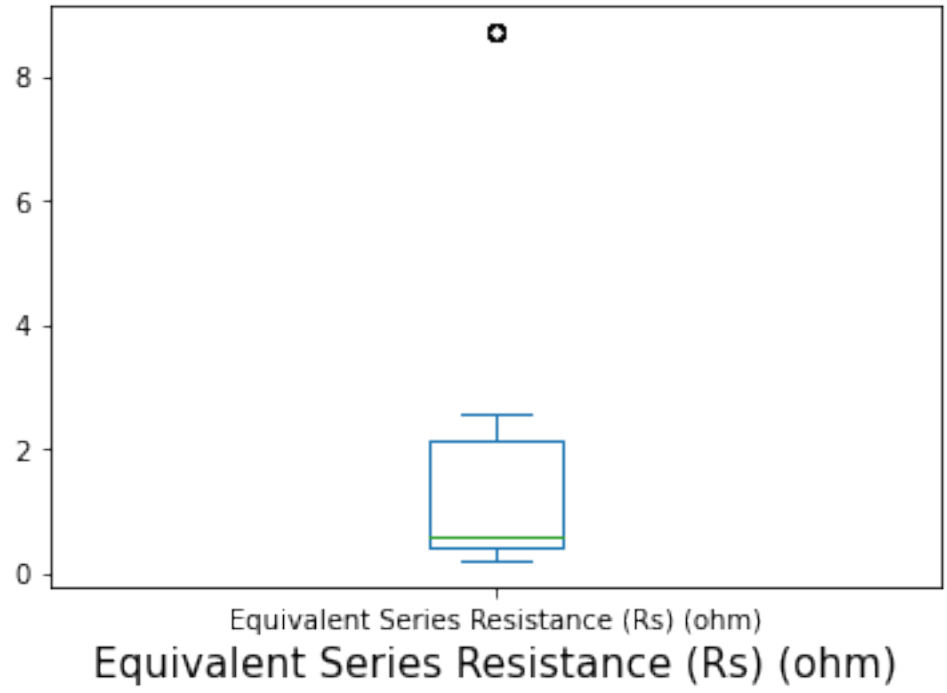


Charge Transfer Resistance (Rct) (ohm)

```
std= 6.19   mean= 5.14
count of available values= 52   count of missing values= 69   count of unique
values= 16
```

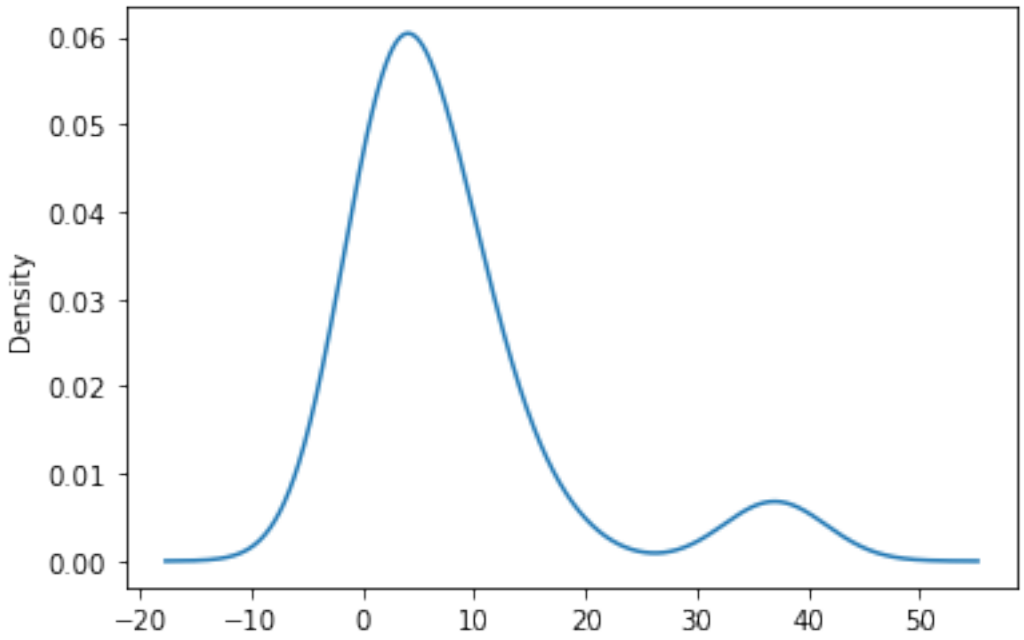
```
available values/total data = 0.430   unique values/available values = 0.308
################################################################################
```

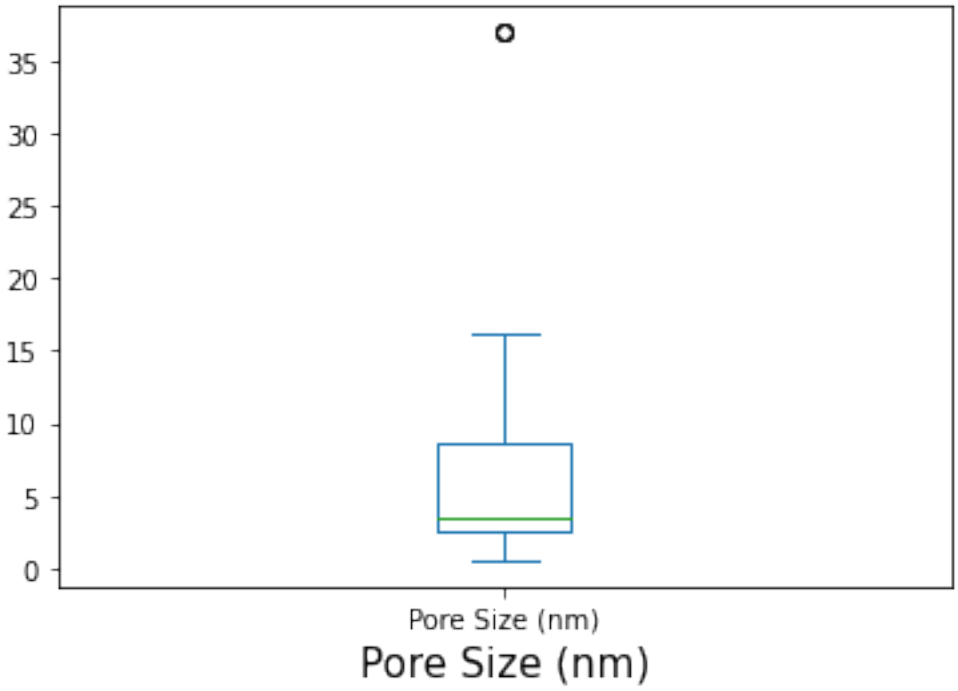




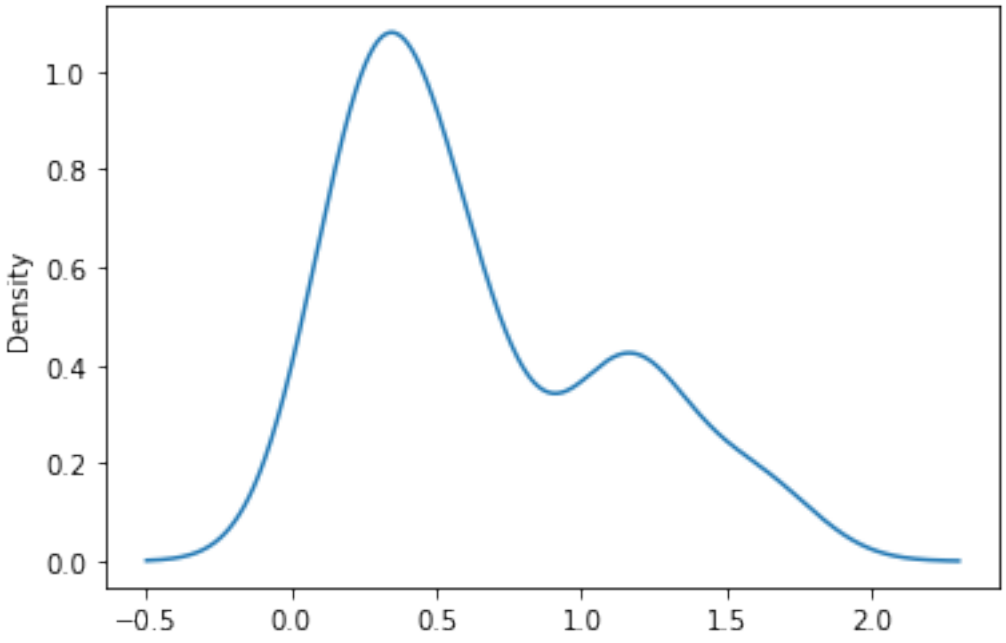
Equivalent Series Resistance (Rs) (ohm)

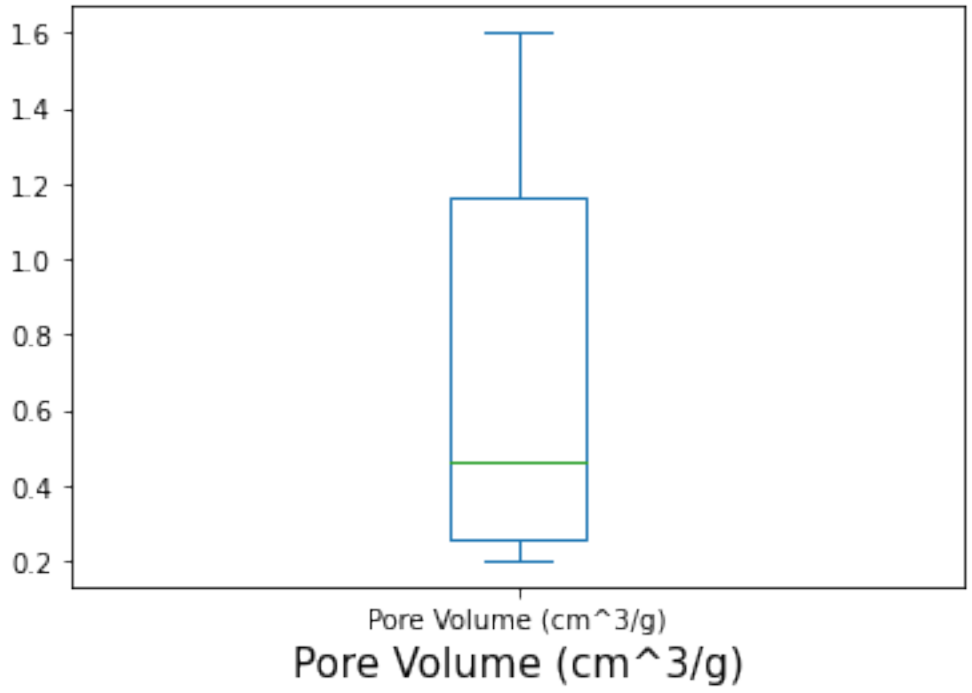
```
std= 2.47   mean= 1.73
count of available values= 79   count of missing values= 42   count of unique
values= 18
available values/total data = 0.653   unique values/available values = 0.228
################################################################################
```





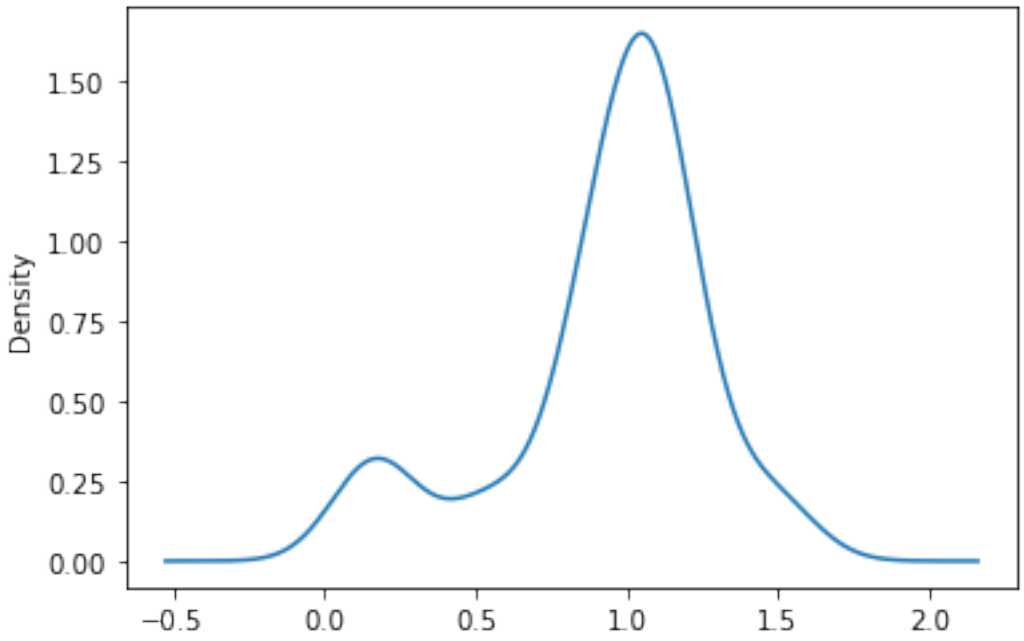

```
std= 9.56   mean= 7.78
count of available values= 38   count of missing values= 83   count of unique
values= 13
available values/total data = 0.314   unique values/available values = 0.342
################################################################################
```

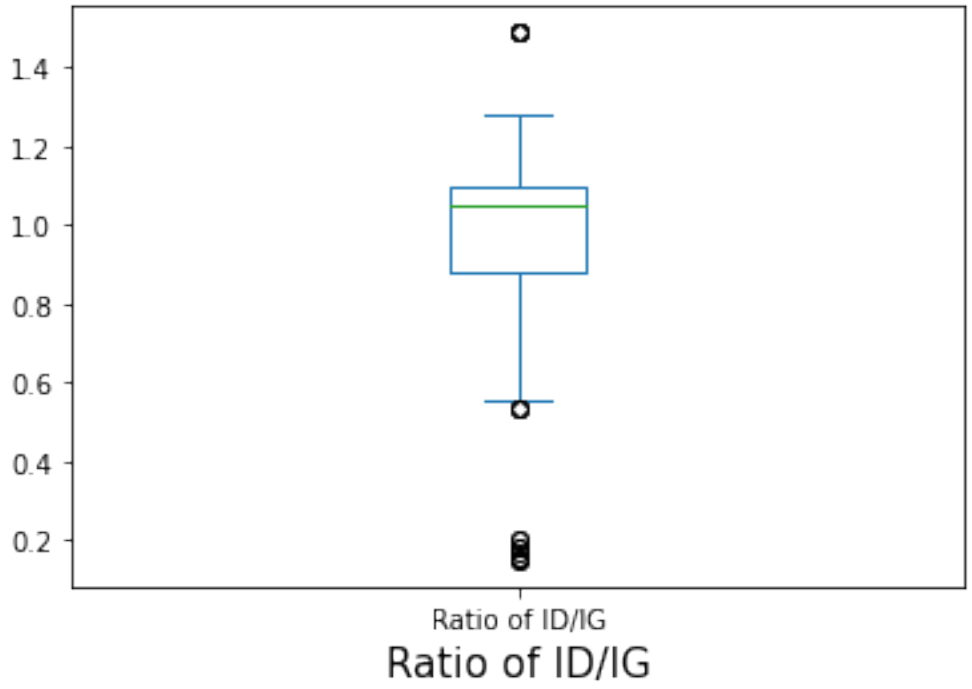





```
std= 0.45   mean= 0.64
count of available values= 49   count of missing values= 72   count of unique
values= 13
```

```
available values/total data = 0.405   unique values/available values = 0.265
################################################################################
```





Ratio of ID/IG

```
std= 0.34   mean= 0.94
count of available values= 80   count of missing values= 41   count of unique
values= 24
available values/total data = 0.661   unique values/available values = 0.300
################################################################################
```

[56]:
```
fillingMissingValues(df5)
```

```
column=  Equivalent Series Resistance (Rs) (ohm)   available ratio= 0.653
unique ratio= 0.228
mean= 1.728   std= 2.472
 missing values filled with 2.2226

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


column=  Ratio of ID/IG   available ratio= 0.661   unique ratio= 0.300
mean= 0.936   std= 0.338
 missing values filled with 0.8682

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


/tmp/ipykernel_270313/1691008062.py:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[column].fillna((mean+(std/5)), inplace= True)
/tmp/ipykernel_270313/1691008062.py:34: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[column].fillna((mean-(std/5)), inplace= True)
```

[57]:
```
print(df5.isnull().sum())
print(df5.shape)
```

```
Limits of Potential Window (V)                      0
Lower Limit of Potential Window (V)                 0
Upper Limit of Potential Window (V)                 0
Potential Window (V)                                0
Current Density (A/g)                               0
Capacitance (F/g)                                   0
Specific Surface Area (m^2/g)                      45
Charge Transfer Resistance (Rct) (ohm)             69
Equivalent Series Resistance (Rs) (ohm)             0
Electrode Material                                  0
Pore Size (nm)                                     83
Pore Volume (cm^3/g)                               72
Ratio of ID/IG                                      0
N at%                                               0
C at%                                               0
O at%                                               0
Electrolyte Chemical Formula                        0
Electrolyte Ion Mobility Ranking                    0
Electrolyte Concentration (M)                       0
Cell Configuration (three/two electrode system)     0
dtype: int64
(121, 20)
```

[58]:
```
df5[EmptyColumnNames(df5)].describe()
```

[58]:
```
       Specific Surface Area (m^2/g)  Charge Transfer Resistance (Rct) (ohm)  \
count                      76.000000                               52.000000
mean                      686.436789                                5.142692
std                       704.290266                                6.191272
min                        20.131000                                0.180000
25%                       131.900000                                1.100000
50%                       369.000000                                2.000000
75%                      1162.000000                                6.980000
max                      2393.000000                               18.500000

       Pore Size (nm)  Pore Volume (cm^3/g)
count       38.000000             49.000000
mean         7.784842              0.639490
std          9.558796              0.449505
min          0.530000              0.200900
25%          2.490000              0.259000
50%          3.550000              0.461000
75%          8.700000              1.160000
max         37.000000              1.600000
```

[59]:
```
df=df5.drop(axis=1,columns=EmptyColumnNames(df5))
```

[60]:
```
df.shape
```

[60]:
```
(121, 16)
```

[61]:
```
df.describe()
```

[61]:

	Limits of Potential Window (V)	Lower Limit of Potential Window (V)	\
count	121.000000	121.000000	
mean	361.784007	-0.516529	
std	187.945381	0.428437	
min	175.819120	-1.000000	
25%	274.376243	-1.000000	
50%	329.643156	-0.400000	
75%	380.690950	-0.200000	
max	1220.633937	0.000000	

	Upper Limit of Potential Window (V)	Potential Window (V)	\
count	121.000000	121.000000	
mean	0.589256	1.089256	
std	0.714936	0.522462	
min	-0.100000	0.500000	
25%	0.000000	0.900000	
50%	0.400000	1.000000	
75%	0.900000	1.000000	
max	2.700000	2.700000	

	Current Density (A/g)	Capacitance (F/g)	\
count	121.000000	121.000000	
mean	6.099752	401.338843	
std	8.246454	383.830743	
min	0.100000	15.000000	
25%	1.000000	160.600000	
50%	3.000000	257.000000	
75%	10.000000	458.700000	
max	50.000000	2246.630000	

	Equivalent Series Resistance (Rs) (ohm)	Electrode Material	\
count	121.000000	121.000000	
mean	1.899826	416.545002	
std	2.006821	65.571697	
min	0.200000	341.234664	
25%	0.480000	372.617261	
50%	2.150000	397.524840	
75%	2.222595	477.403946	
max	8.700000	653.457009	

	Ratio of ID/IG	N at%	C at%	O at%	\

```
count      121.000000  121.000000  121.000000  121.000000
mean         0.912961    1.524959   63.261240   21.090992
std          0.276367    2.601716   30.466178   15.725154
min          0.147000    0.000000    1.400000    2.200000
25%          0.868227    0.000000   32.370000    6.930000
50%          0.880000    0.000000   77.830000   16.400000
75%          1.100000    2.300000   86.210000   35.720000
max          1.490000    8.910000   97.300000   52.480000

       Electrolyte Chemical Formula  Electrolyte Ion Mobility Ranking  \
count                    121.000000                        121.000000
mean                     439.750913                          5.842975
std                       83.076697                          1.264966
min                      243.102425                          1.000000
25%                      372.518745                          6.000000
50%                      491.038310                          6.000000
75%                      491.038310                          6.000000
max                      499.830600                          8.000000

       Electrolyte Concentration (M)  \
count                     121.000000
mean                        2.891736
std                         2.300854
min                         0.100000
25%                         1.000000
50%                         2.000000
75%                         6.000000
max                         6.000000

       Cell Configuration (three/two electrode system)
count                                       121.000000
mean                                        428.785441
std                                          67.631534
min                                         249.745539
25%                                         454.153769
50%                                         454.153769
75%                                         454.153769
max                                         454.153769
```

[62]:
```
df.isnull().sum()
```

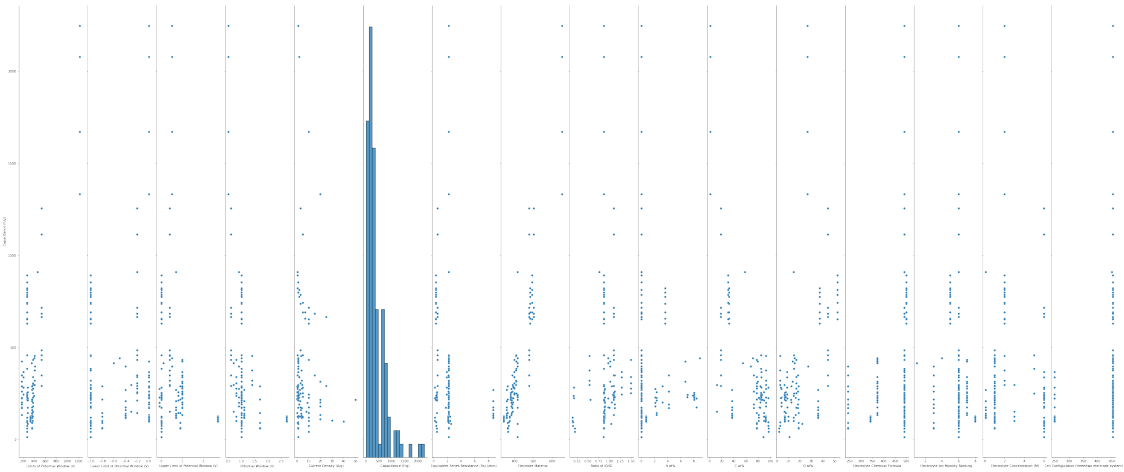
[62]:
```
Limits of Potential Window (V)                  0
Lower Limit of Potential Window (V)             0
Upper Limit of Potential Window (V)             0
Potential Window (V)                            0
Current Density (A/g)                           0
Capacitance (F/g)                               0
```

```
Equivalent Series Resistance (Rs) (ohm)            0
Electrode Material                                 0
Ratio of ID/IG                                     0
N at%                                              0
C at%                                              0
O at%                                              0
Electrolyte Chemical Formula                       0
Electrolyte Ion Mobility Ranking                   0
Electrolyte Concentration (M)                      0
Cell Configuration (three/two electrode system)    0
dtype: int64
```

[63]:
```
import seaborn as sns
```

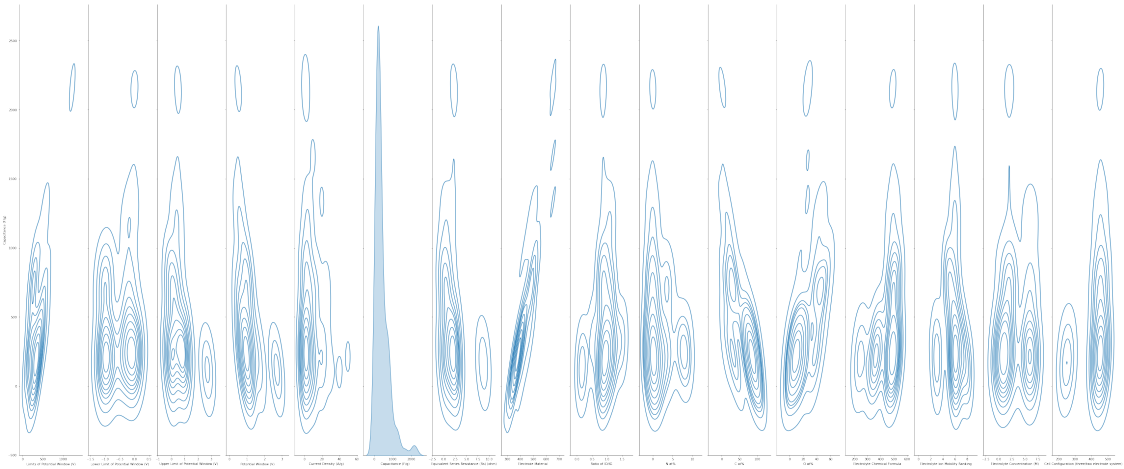
[64]:
```
sns.pairplot(df,y_vars='Capacitance (F/g)',aspect=3/20,height=20)
```

[64]:
```
<seaborn.axisgrid.PairGrid at 0x7f7a0fbbe8e0>
```



[65]:
```
sns.pairplot(df,kind='kde',y_vars='Capacitance (F/g)',aspect=3/20,height=20)
```

[65]:
```
<seaborn.axisgrid.PairGrid at 0x7f7a0fdb9580>
```



[66]:
```
df.columns
```

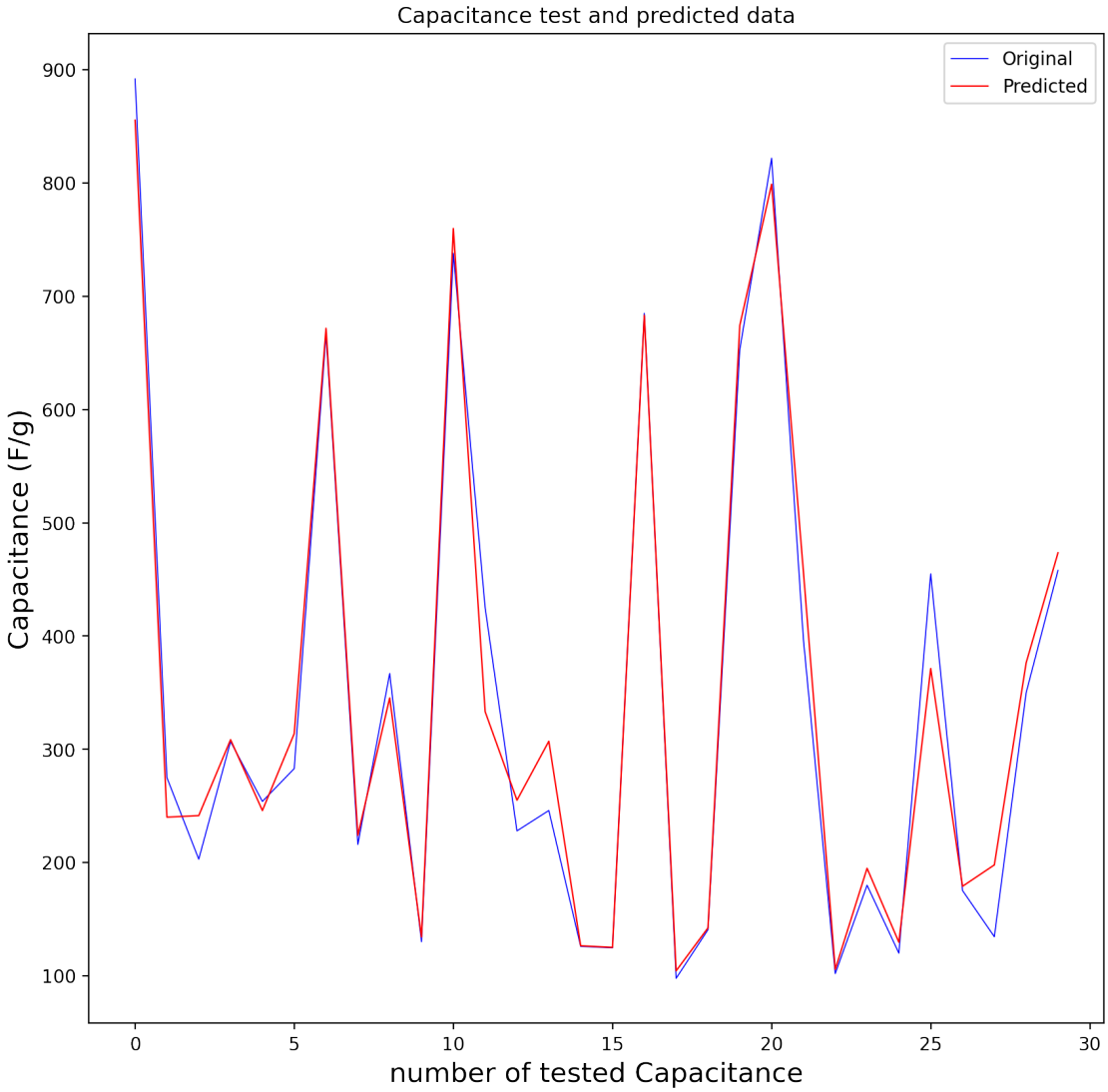
[66]:
```
Index(['Limits of Potential Window (V)', 'Lower Limit of Potential Window (V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)', 'Capacitance (F/g)',
       'Equivalent Series Resistance (Rs) (ohm)', 'Electrode Material',
       'Ratio of ID/IG', 'N at%', 'C at%', 'O at%',
       'Electrolyte Chemical Formula', 'Electrolyte Ion Mobility Ranking',
       'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)'],
      dtype='object')
```

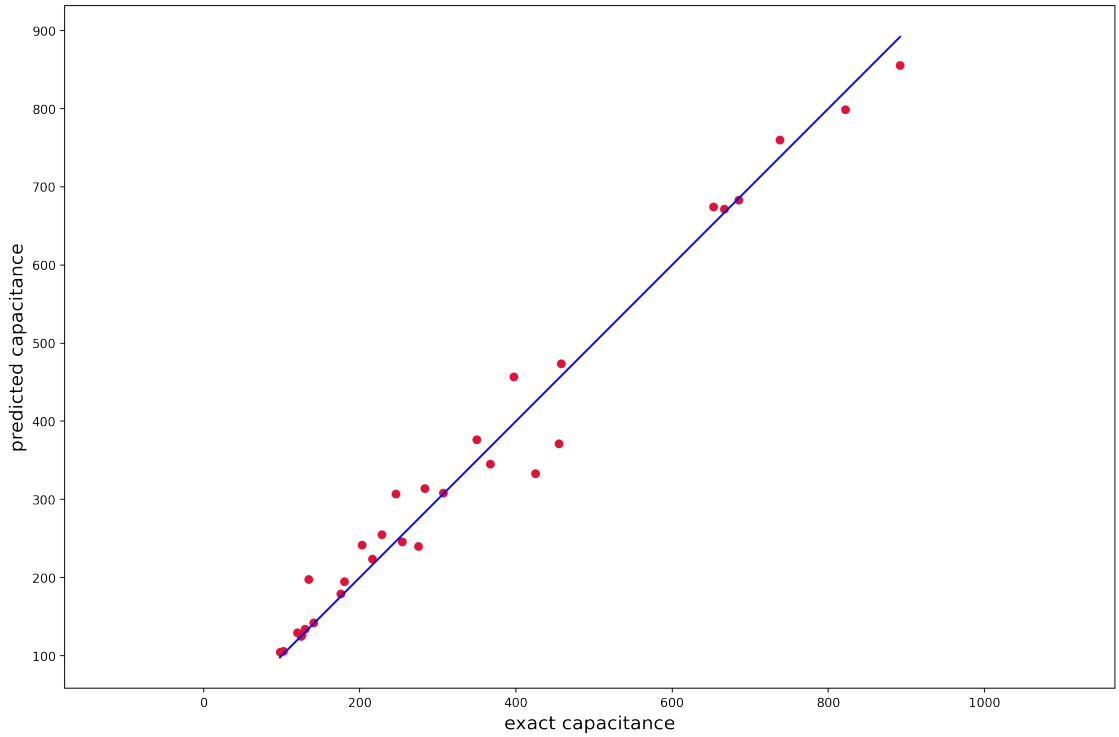
[67]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window␣
 ↪(V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)',
       'Equivalent Series Resistance (Rs) (ohm)', 'Electrode Material',
       'Ratio of ID/IG', 'N at%', 'C at%', 'O at%',
       'Electrolyte Chemical Formula', 'Electrolyte Ion Mobility Ranking',
       'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']



X=df[Predictors].values
y=df[TargetVariable]



from sklearn.preprocessing import StandardScaler
```

```
PredictorScaler=StandardScaler()
PredictorScalerFit=PredictorScaler.fit(X)


X=PredictorScalerFit.transform(X)
y=y.values.ravel()



from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error , r2_score




opted=ExtraTreesRegressionCVScore()
```

[68]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],␣
  ↪random_state=opted[2])
etr=ExtraTreesRegressor(n_jobs=-1)
etr.fit(X_train,y_train)


Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)



mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)



TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions



from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
```

```
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()




plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)
print('\n')
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].
 ↪head(count))
print(2*'\n')
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
print('number of tested data= %i'%count)
print('test_size"percent"= %.2f'%(opted[0]*100))
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
print('optimized MSE: %.4f ' %mse)
print('optimized RMSE: %.4f ' %mse**(0.5))
```


Capacitance test and predicted data
Original
Predicted
Capacitance (F/g)
number of tested Capacitance
900
800
700
600
500
400
300
200
100
0
5
10
15
20
25
30



	Capacitance (F/g)	PredictedCapacitance (F/g)
0	892.00	855.5500
1	275.00	240.1070
2	203.00	241.4730
3	306.87	308.5312
4	254.00	245.9600
5	283.00	313.8740
6	666.60	671.7520
7	216.00	224.0400
8	367.00	345.3905
9	130.10	134.3110
10	738.00	759.8900
11	425.00	333.2575
12	228.00	255.0710
13	246.00	307.1450
14	126.00	126.5490
15	124.70	125.1060
16	685.00	683.1640
17	97.80	104.4000
18	140.70	142.3120
19	653.00	674.3100
20	822.00	799.0000

```
21              397.00                  456.8215
22              102.00                  105.7420
23              180.00                  194.9680
24              120.00                  129.6480
25              455.00                  371.3736
26              175.30                  179.0680
27              134.52                  197.9490
28              350.00                  376.4400
```

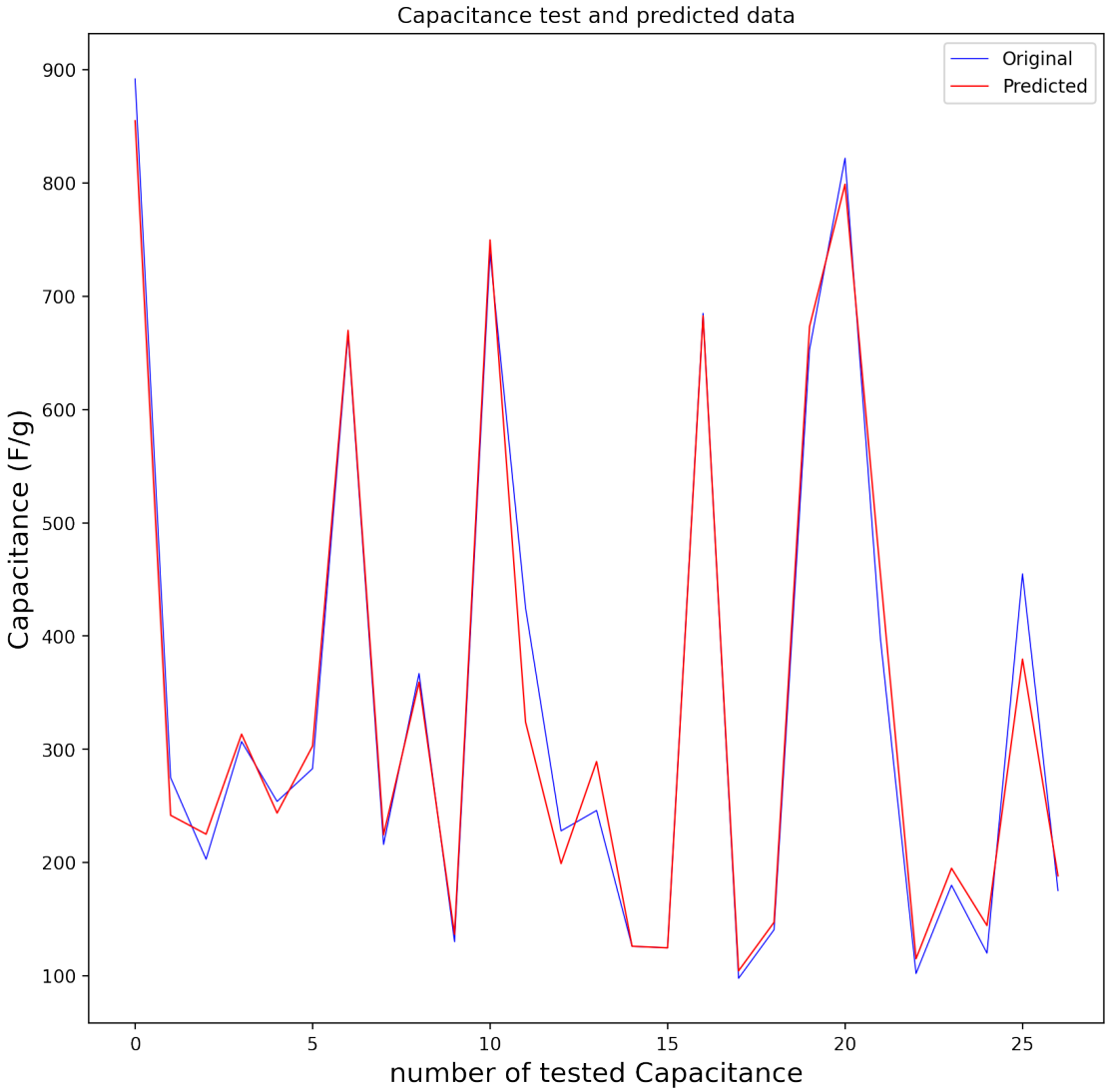
```
total number of data= 121
number of trained data= 92
number of tested data= 29
test_size"percent"= 24.00
score for xtest and ytest : 0.9770
cross validation score: 0.9389
optimized MSE: 1196.8349
optimized RMSE: 34.5953
```

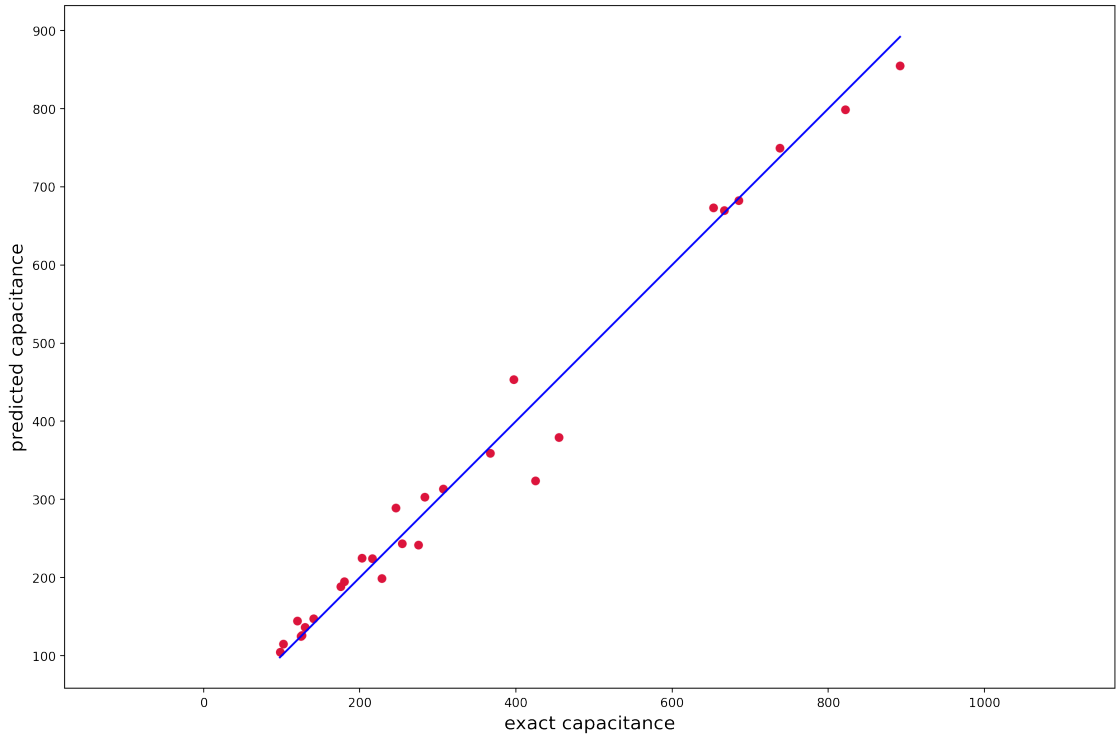
[69]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window␣
 ↪(V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)',
       'Equivalent Series Resistance (Rs) (ohm)', 'Electrode Material',
       'Ratio of ID/IG', 'N at%', 'C at%', 'O at%',
       'Electrolyte Chemical Formula', 'Electrolyte Ion Mobility Ranking',
       'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']


X=df[Predictors].values
y=df[TargetVariable]



from sklearn.preprocessing import StandardScaler

PredictorScaler=StandardScaler()
PredictorScalerFit=PredictorScaler.fit(X)




X=PredictorScalerFit.transform(X)
y=y.values.ravel()
```

```
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error , r2_score




opted=ExtraTreesRegressionMse()
```

[70]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],␣
 ↪random_state=opted[2])
etr=ExtraTreesRegressor(n_jobs=-1)
etr.fit(X_train,y_train)


Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)


mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)


TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions


from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()
```

```
plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)
print('\n')
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].
 ↪head(count))
print(2*'\n')
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
print('number of tested data= %i'%count)
print('test_size"percent"= %.2f'%(opted[0]*100))
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
print('optimized MSE: %.4f ' %mse)
print('optimized RMSE: %.4f ' %mse**(0.5))
```


Capacitance test and predicted data
Original
Predicted
Capacitance (F/g)
number of tested Capacitance
900
800
700
600
500
400
300
200
100
0
5
10
15
20
25



	Capacitance (F/g)	PredictedCapacitance (F/g)
0	892.00	855.0000
1	275.00	241.7870
2	203.00	225.0962
3	306.87	313.3936
4	254.00	243.7300
5	283.00	302.9320
6	666.60	669.9120
7	216.00	224.5200
8	367.00	359.2545
9	130.10	136.5970
10	738.00	749.8200
11	425.00	324.1140
12	228.00	199.1164
13	246.00	289.2130
14	126.00	126.1420
15	124.70	124.6870
16	685.00	682.6880
17	97.80	104.4000
18	140.70	147.2650
19	653.00	673.3360
20	822.00	799.0000

```
21              397.00                  453.7090
22              102.00                  115.0840
23              180.00                  194.9824
24              120.00                  144.4138
25              455.00                  379.7000



total number of data= 121
number of trained data= 95
number of tested data= 26
test_size"percent"= 22.00
score for xtest and ytest : 0.9816
cross validation score: 0.9328
optimized MSE: 1029.3361
optimized RMSE: 32.0833
```

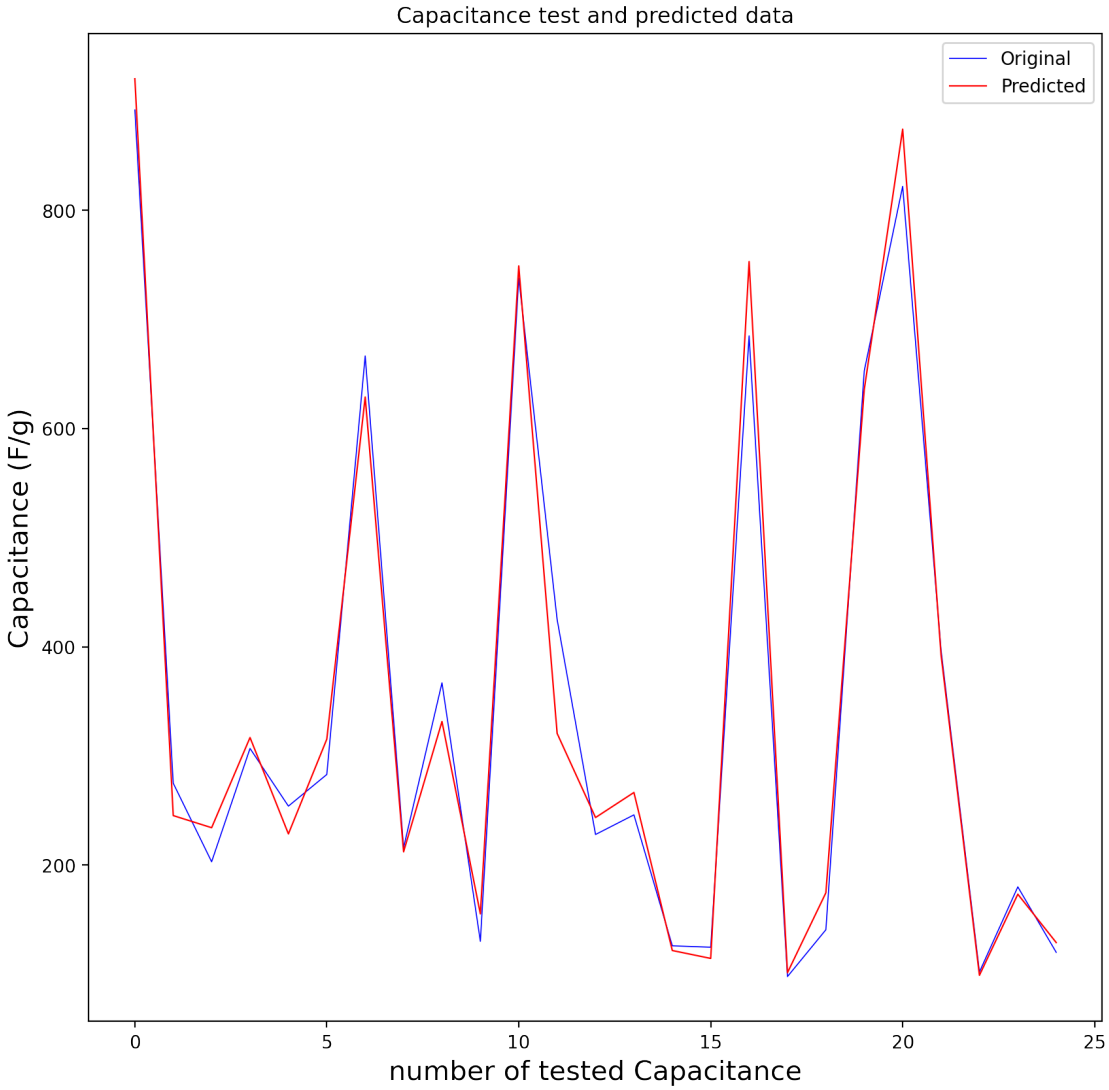
Gradient boosting regressor

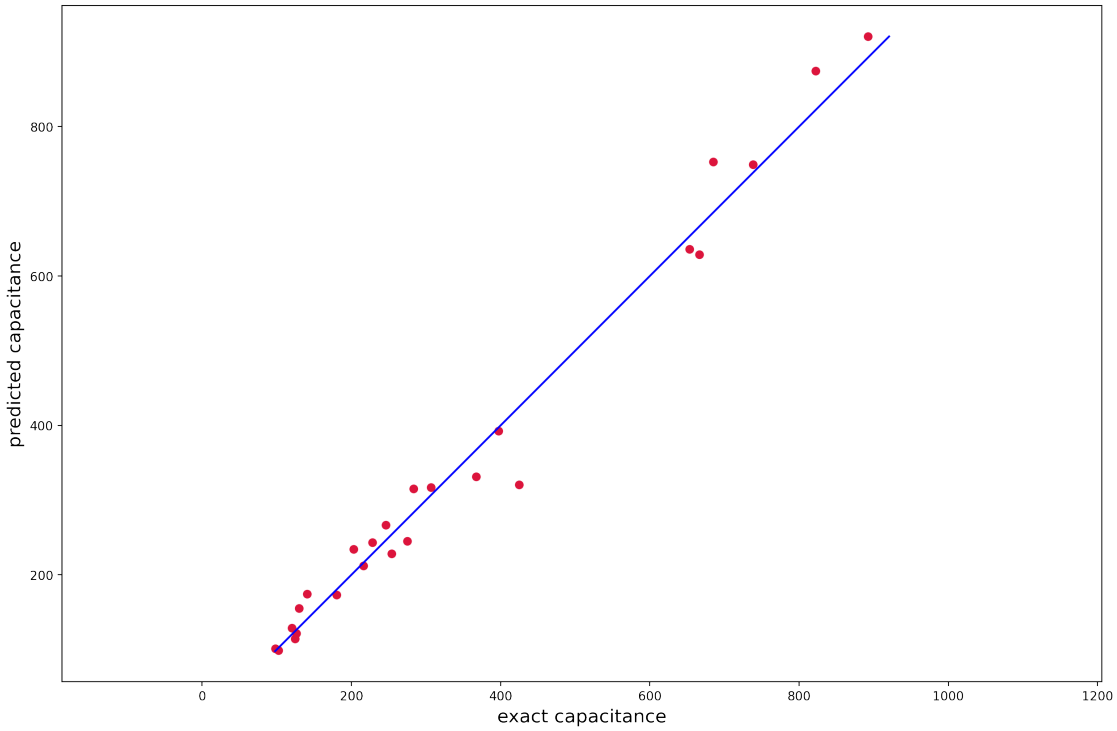
[71]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window␣
 ↪(V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)',
       'Equivalent Series Resistance (Rs) (ohm)', 'Electrode Material',
       'Ratio of ID/IG', 'N at%', 'C at%', 'O at%',
       'Electrolyte Chemical Formula', 'Electrolyte Ion Mobility Ranking',
       'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']



X=df[Predictors].values
y=df[TargetVariable]


from sklearn.preprocessing import StandardScaler


PredictorScaler=StandardScaler()
PredictorScalerFit=PredictorScaler.fit(X)



X=PredictorScalerFit.transform(X)
y=y.values.ravel()
```

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error , r2_score



opted=GradientBoostRegScore()
```

[72]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],␣
 ↪random_state=opted[2])
etr=GradientBoostingRegressor(random_state=opted[2])
etr.fit(X_train,y_train)


Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)



mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)



TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions


from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()
```

```
plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)
print('\n')
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].
 ↪head(count))
print(2*'\n')
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
print('number of tested data= %i'%count)
print('test_size"percent"= %.2f'%(opted[0]*100))
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
print('optimized MSE: %.4f ' %mse)
print('optimized RMSE: %.4f ' %mse**(0.5))
print('optimized random state: %.4f'%opted[2] )
```


Capacitance test and predicted data
Original
Predicted
800
600
400
200
Capacitance (F/g)
0
5
10
15
20
25
number of tested Capacitance



	Capacitance (F/g)	PredictedCapacitance (F/g)
0	892.00	920.635956
1	275.00	245.331810
2	203.00	234.225915
3	306.87	316.867978
4	254.00	228.544737
5	283.00	315.383181
6	666.60	628.915350
7	216.00	212.148015
8	367.00	331.542386
9	130.10	155.270697
10	738.00	749.076828
11	425.00	320.627014
12	228.00	243.548236
13	246.00	266.510822
14	126.00	121.668425
15	124.70	114.434229
16	685.00	753.075659
17	97.80	101.360570
18	140.70	174.568202
19	653.00	635.866917
20	822.00	874.407751

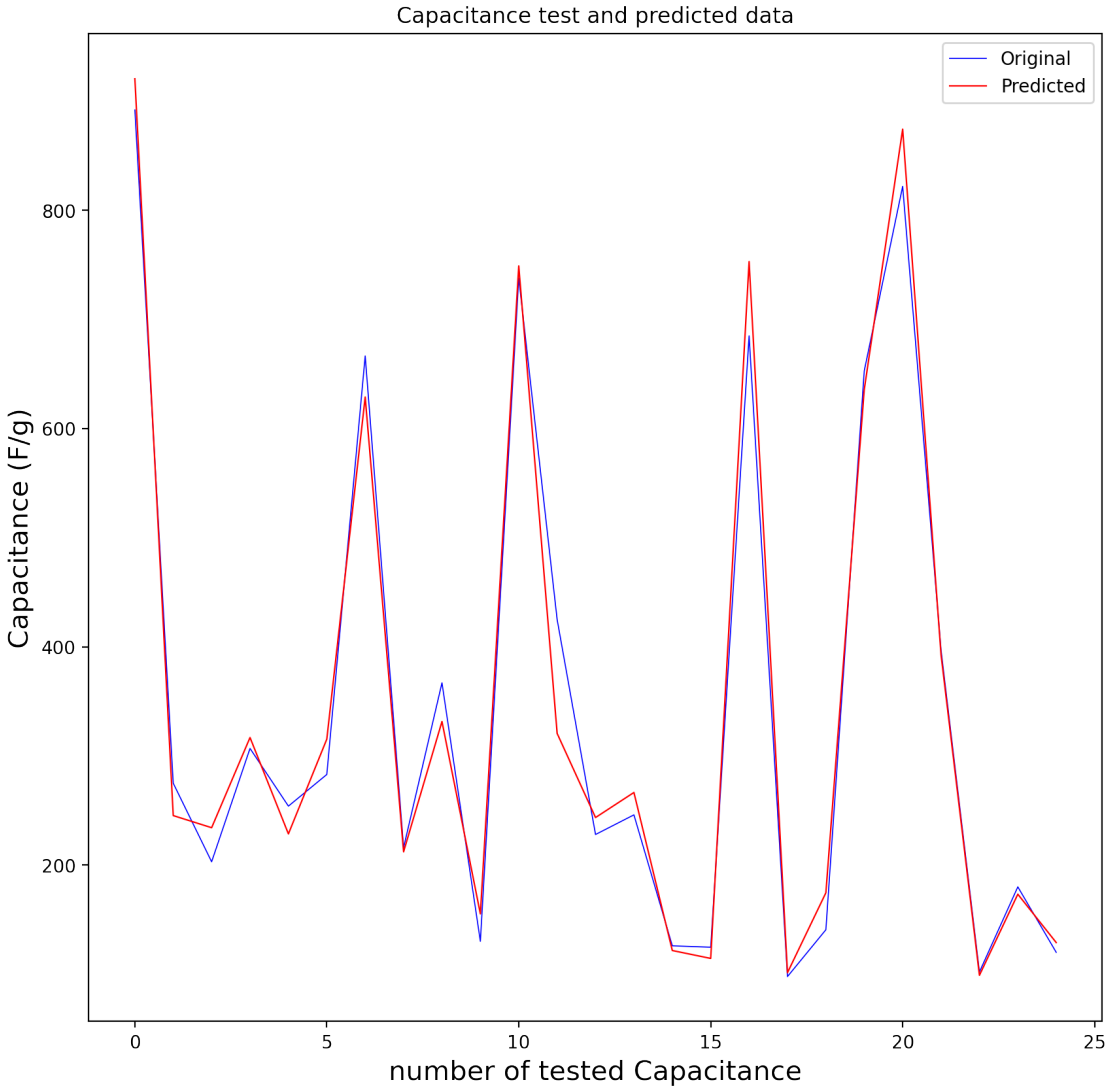
```
21              397.00                  392.658197
22              102.00                   98.958679
23              180.00                  173.281060



total number of data= 121
number of trained data= 97
number of tested data= 24
test_size"percent"= 20.00
score for xtest and ytest : 0.9805
cross validation score: 0.8678
optimized MSE: 1143.5425
optimized RMSE: 33.8163
optimized random state: 2.0000
```

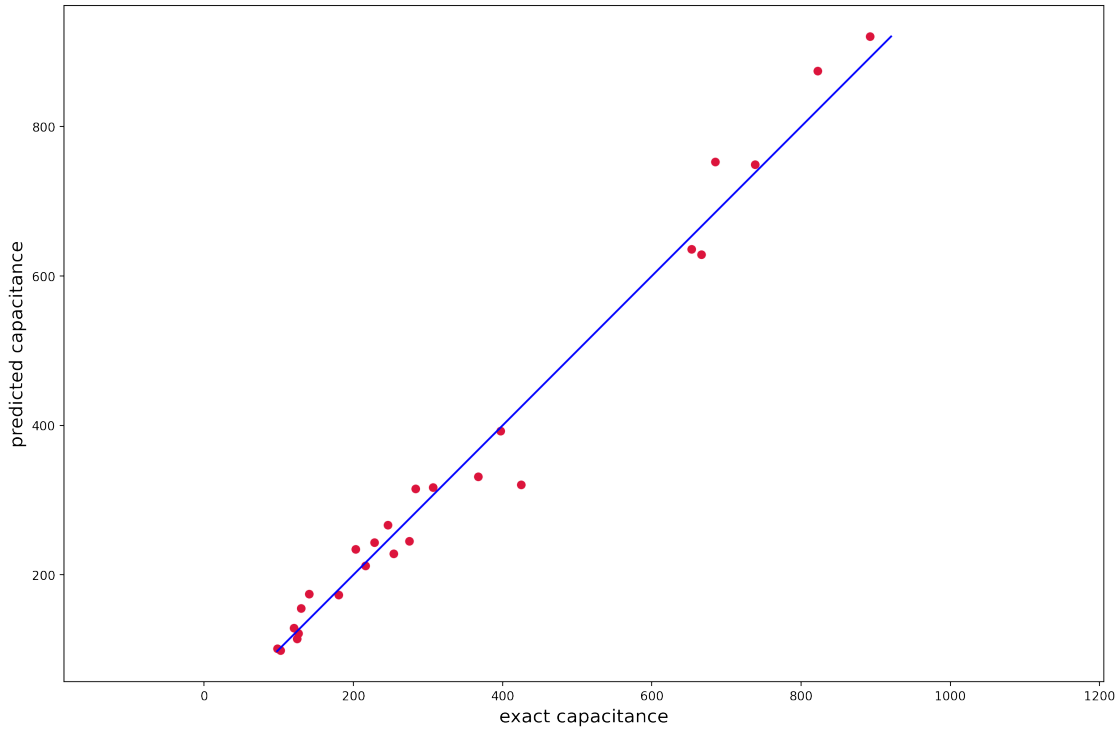
[73]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window␣
 ↪(V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)',
       'Equivalent Series Resistance (Rs) (ohm)', 'Electrode Material',
       'Ratio of ID/IG', 'N at%', 'C at%', 'O at%',
       'Electrolyte Chemical Formula', 'Electrolyte Ion Mobility Ranking',
       'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']



X=df[Predictors].values
y=df[TargetVariable]



from sklearn.preprocessing import StandardScaler
PredictorScaler=StandardScaler()



PredictorScalerFit=PredictorScaler.fit(X)
X=PredictorScalerFit.transform(X)

y=y.values.ravel()


from sklearn.ensemble import GradientBoostingRegressor
```

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error , r2_score




opted=GradientBoostRegMse()
```

[74]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],␣
 ↪random_state=opted[2])
etr=GradientBoostingRegressor(random_state=opted[2])
etr.fit(X_train,y_train)



Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)




mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)



TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions


from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()
```

```
plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)
print('\n')
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].
 ↪head(count))
print(2*'\n')
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
print('number of tested data= %i'%count)
print('test_size"percent"= %.2f'%(opted[0]*100))
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
print('optimized MSE: %.4f ' %mse)
print('optimized RMSE: %.4f ' %mse**(0.5))
print('optimized random state: %.4f'%opted[2] )
```


Capacitance test and predicted data
Original
Predicted
800
600
400
200
Capacitance (F/g)
0
5
10
15
20
25
number of tested Capacitance



	Capacitance (F/g)	PredictedCapacitance (F/g)
0	892.00	920.635956
1	275.00	245.331810
2	203.00	234.225915
3	306.87	316.867978
4	254.00	228.544737
5	283.00	315.383181
6	666.60	628.915350
7	216.00	212.148015
8	367.00	331.542386
9	130.10	155.270697
10	738.00	749.076828
11	425.00	320.627014
12	228.00	243.548236
13	246.00	266.510822
14	126.00	121.668425
15	124.70	114.434229
16	685.00	753.075659
17	97.80	101.360570
18	140.70	174.568202
19	653.00	635.866917
20	822.00	874.407751

```
21              397.00                  392.658197
22              102.00                   98.958679
23              180.00                  173.281060
```

```
total number of data= 121
number of trained data= 97
number of tested data= 24
test_size"percent"= 20.00
score for xtest and ytest : 0.9805
cross validation score: 0.8678
optimized MSE: 1143.5425
optimized RMSE: 33.8163
optimized random state: 2.0000
```

[75]:
```
df.shape
```

[75]:
```
(121, 16)
```

the above dataset was prepared from main dataset with rows lessthan 5 missing values

rows with lessthan 6 missing values

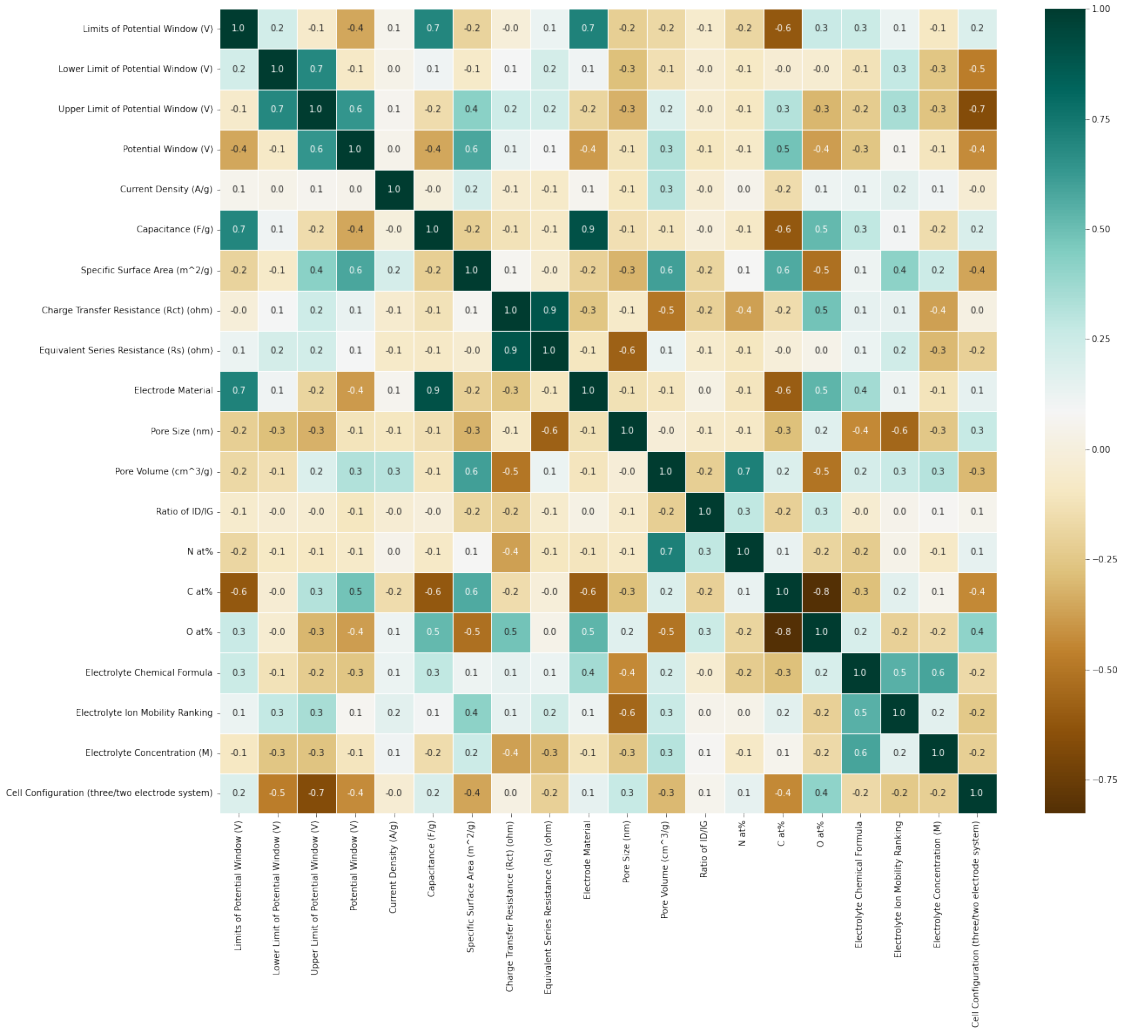
[76]:
```
df6=dframe[6]
```

[77]:
```
df6.shape
```

[77]:
```
(304, 20)
```

[78]:
```
plt.figure(figsize=(20,17))
heatmap(df6.corr(),annot=True,fmt='.1f',linewidths=0.5,cmap='BrBG')
```

[78]:
```
<AxesSubplot:>
```



[79]:
```
df6.isnull().sum()
```

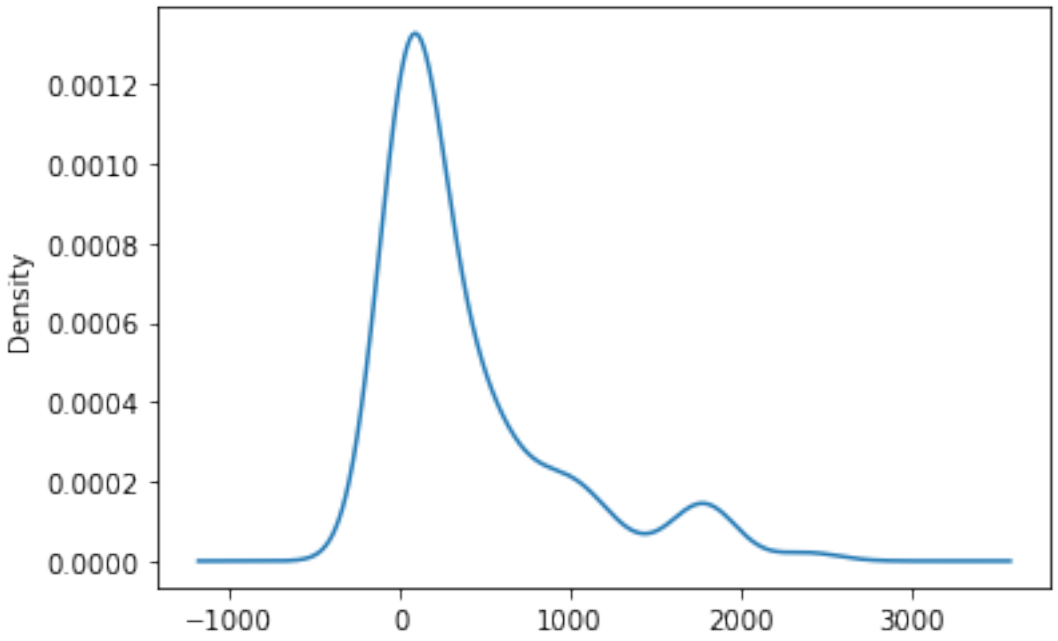
[79]:
```
Limits of Potential Window (V)                    0
Lower Limit of Potential Window (V)               0
Upper Limit of Potential Window (V)               0
Potential Window (V)                              0
Current Density (A/g)                             0
Capacitance (F/g)                                 0
Specific Surface Area (m^2/g)                    96
Charge Transfer Resistance (Rct) (ohm)          167
Equivalent Series Resistance (Rs) (ohm)         160
Electrode Material                                0
Pore Size (nm)                                  167
Pore Volume (cm^3/g)                            151
Ratio of ID/IG                                  115
```

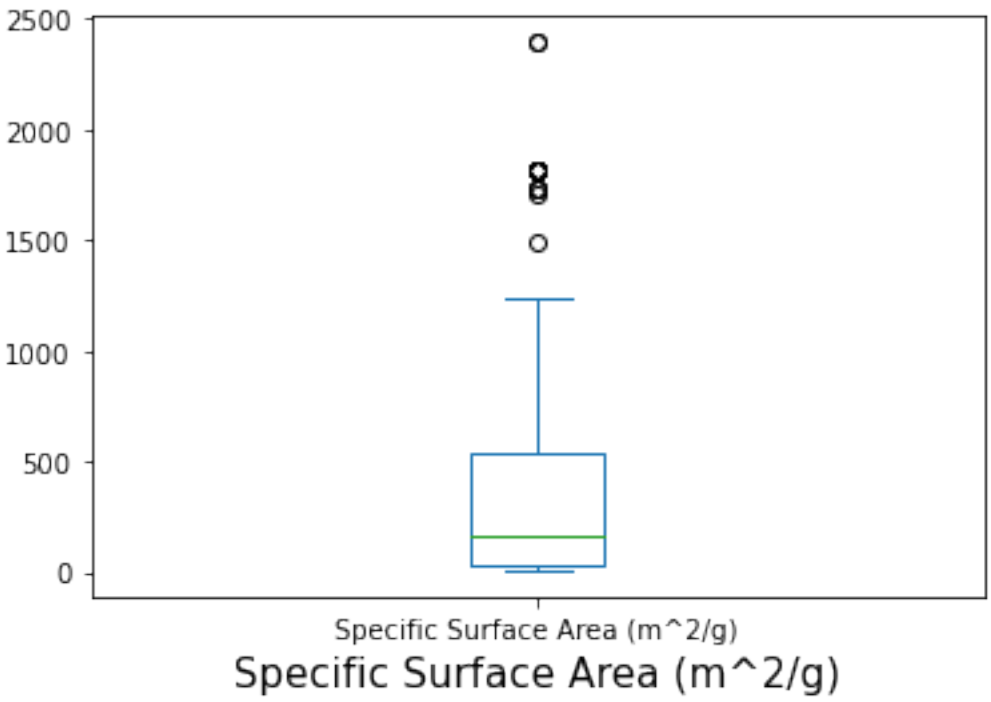
```
N at%                                                105
C at%                                                106
O at%                                                112
Electrolyte Chemical Formula                           0
Electrolyte Ion Mobility Ranking                       0
Electrolyte Concentration (M)                          0
Cell Configuration (three/two electrode system)        0
dtype: int64
```

[80]:
```
EmptyColumnNames(df6)
```

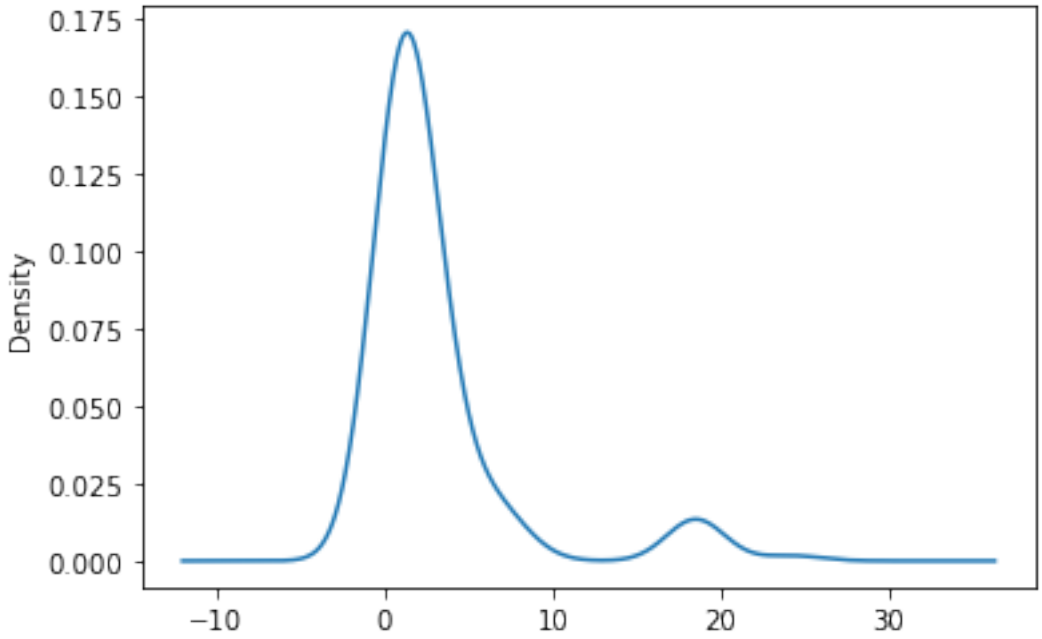
[80]:
```
['Specific Surface Area (m^2/g)',
 'Charge Transfer Resistance (Rct) (ohm)',
 'Equivalent Series Resistance (Rs) (ohm)',
 'Pore Size (nm)',
 'Pore Volume (cm^3/g)',
 'Ratio of ID/IG',
 'N at%',
 'C at%',
 'O at%']
```

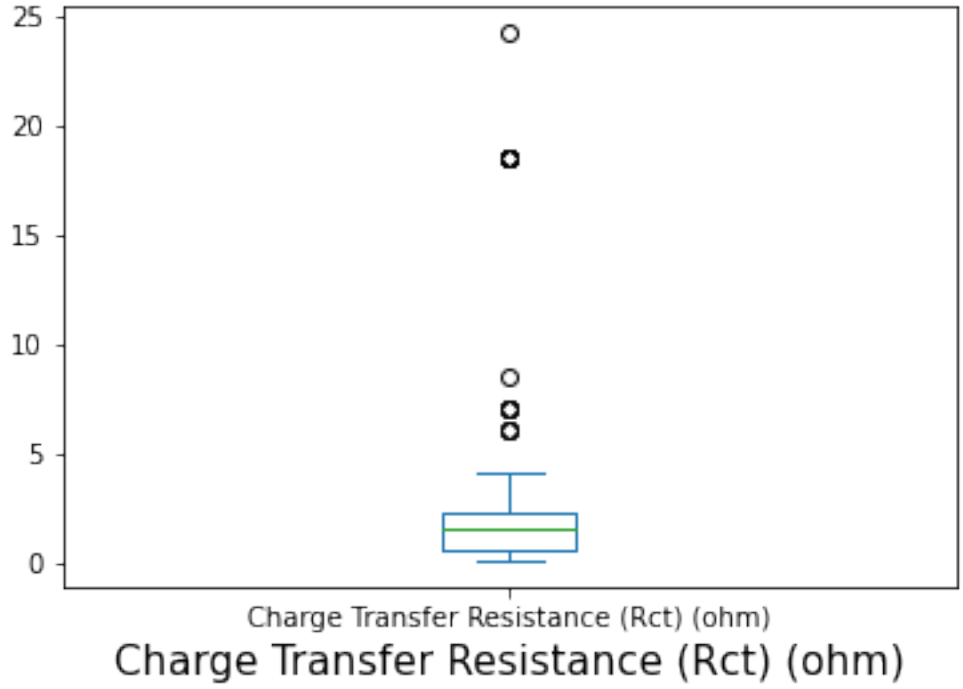
[81]:
```
NullColumnsPlot(df6)
```






```
std= 536.74   mean= 399.33
count of available values= 208   count of missing values= 96   count of unique
values= 64
available values/total data = 0.684   unique values/available values = 0.308
################################################################################
```

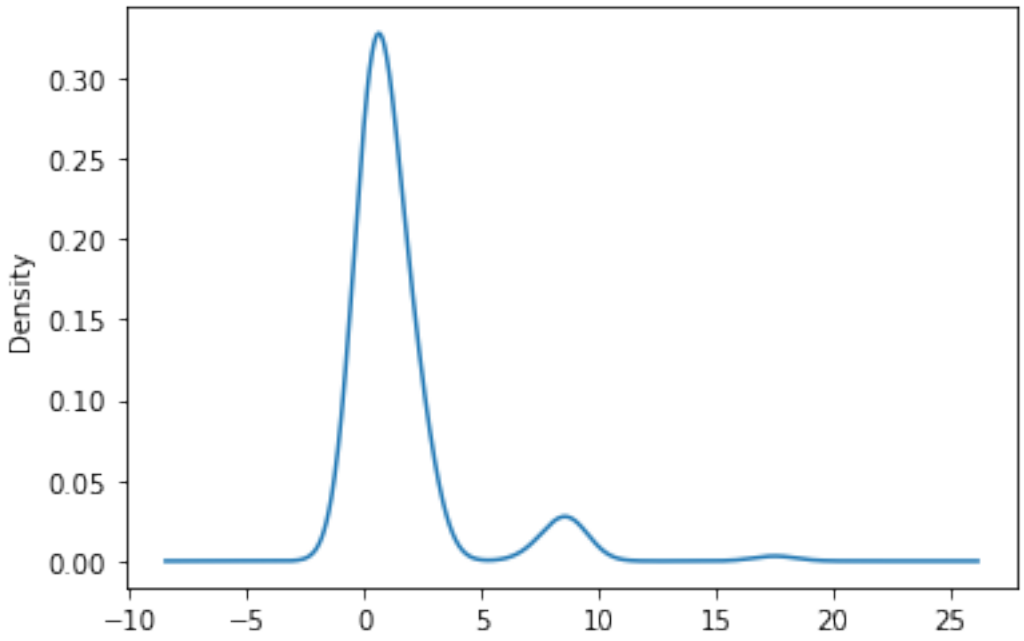


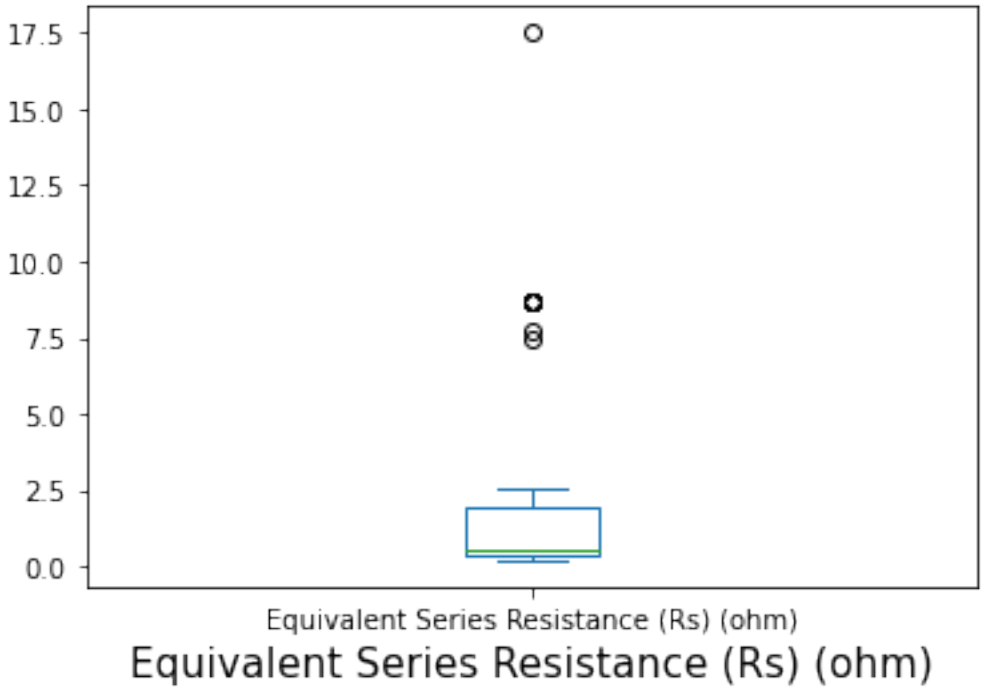


Charge Transfer Resistance (Rct) (ohm)

std= 4.63 mean= 3.03
count of available values= 137 count of missing values= 167 count of unique values= 38

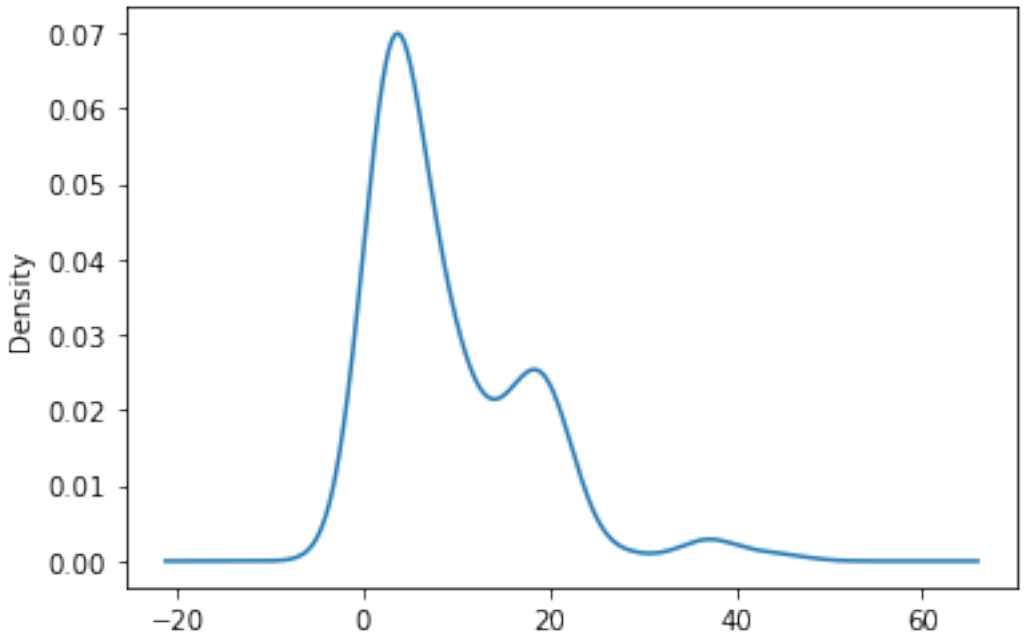
```
available values/total data = 0.451   unique values/available values = 0.277
################################################################################
```

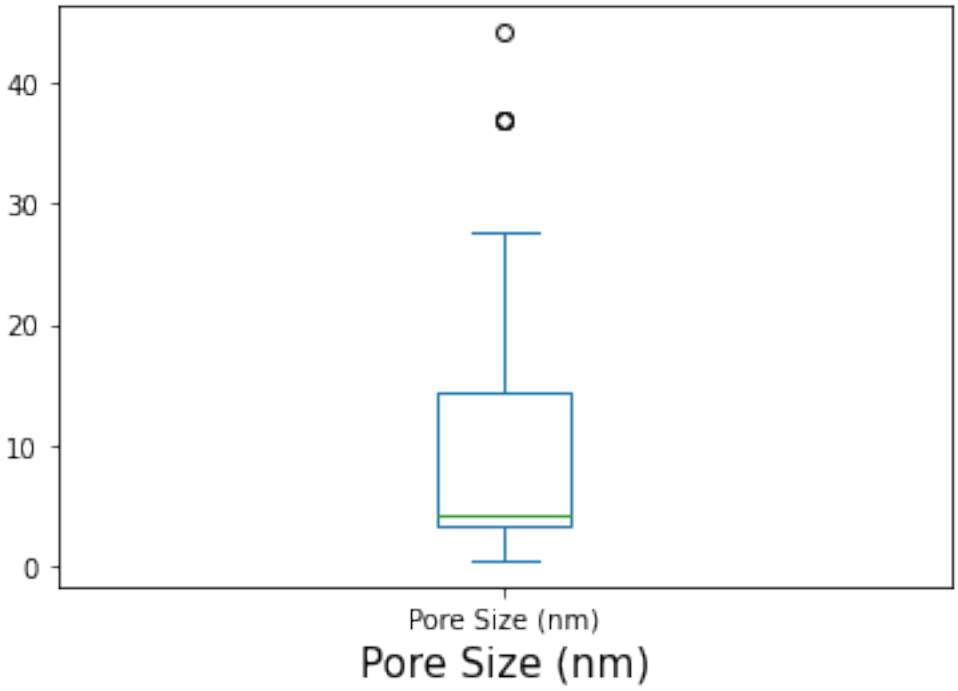




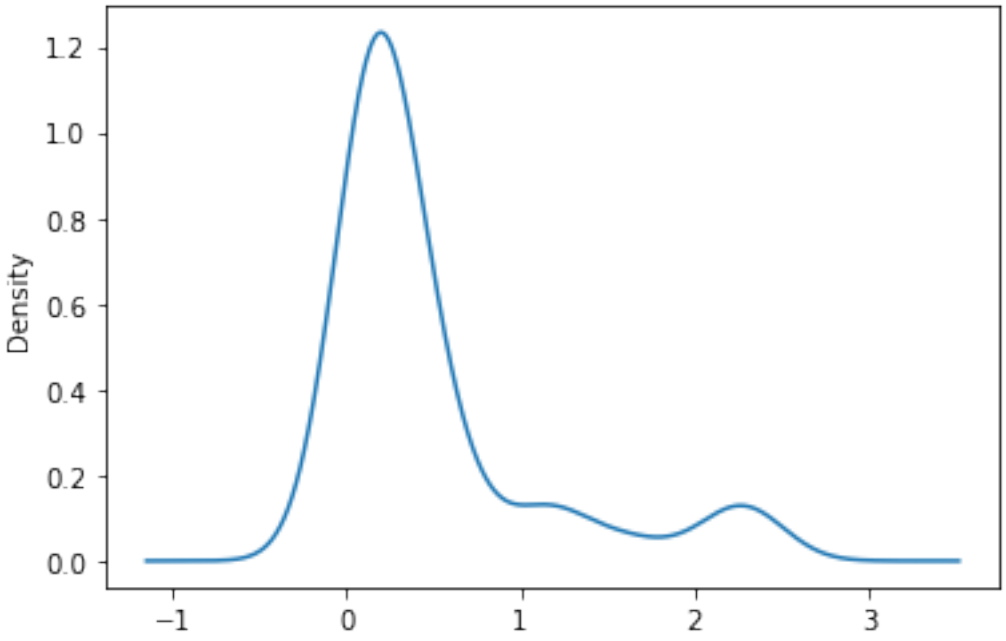
Equivalent Series Resistance (Rs) (ohm)

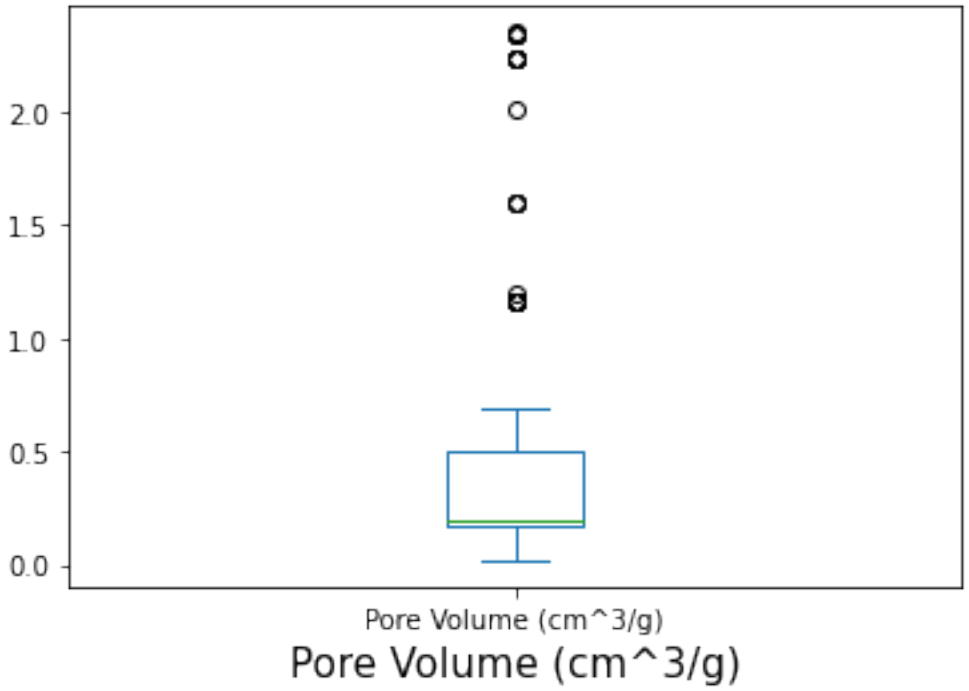
```
std= 2.46   mean= 1.51
count of available values= 144   count of missing values= 160   count of unique
values= 33
available values/total data = 0.474   unique values/available values = 0.229
################################################################################
```






```
std= 8.37   mean= 9.11
count of available values= 137   count of missing values= 167   count of unique
values= 39
available values/total data = 0.451   unique values/available values = 0.285
################################################################################
```

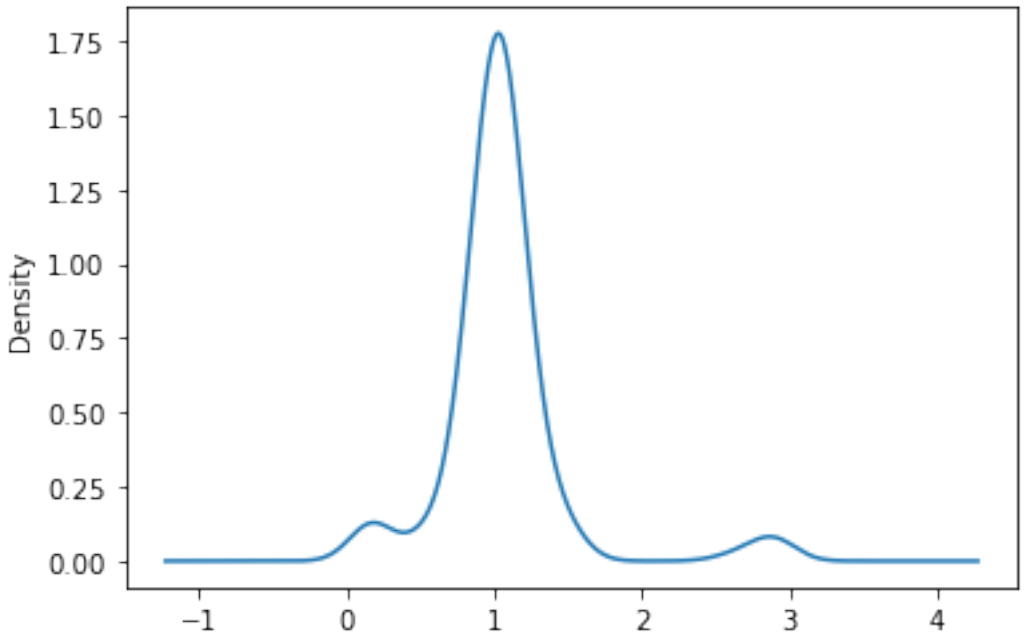


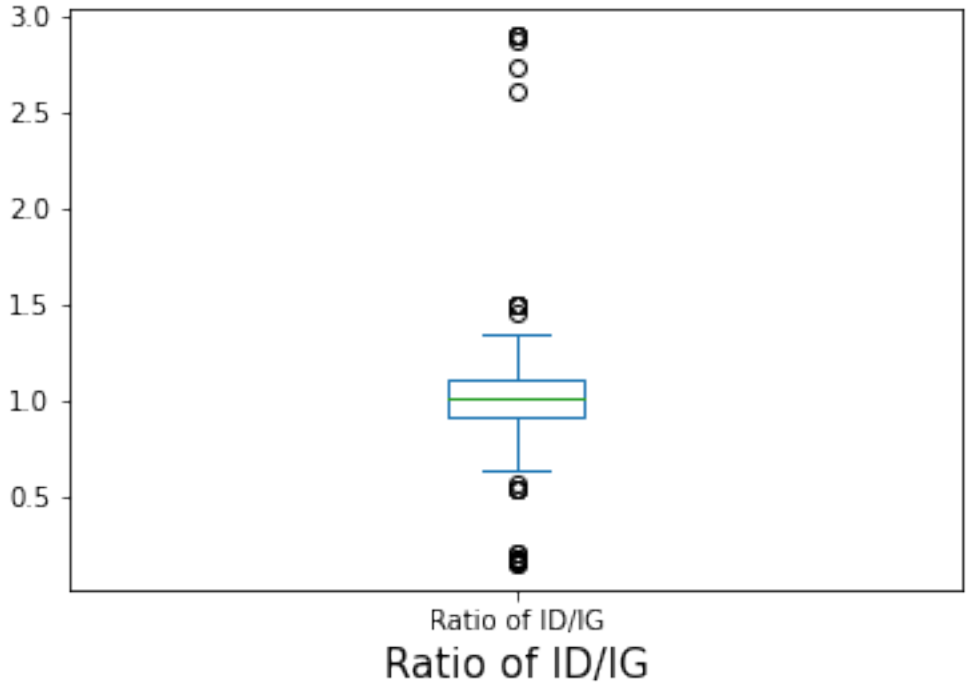


Pore Volume (cm^3/g)

std= 0.62 mean= 0.49
count of available values= 153 count of missing values= 151 count of unique values= 38

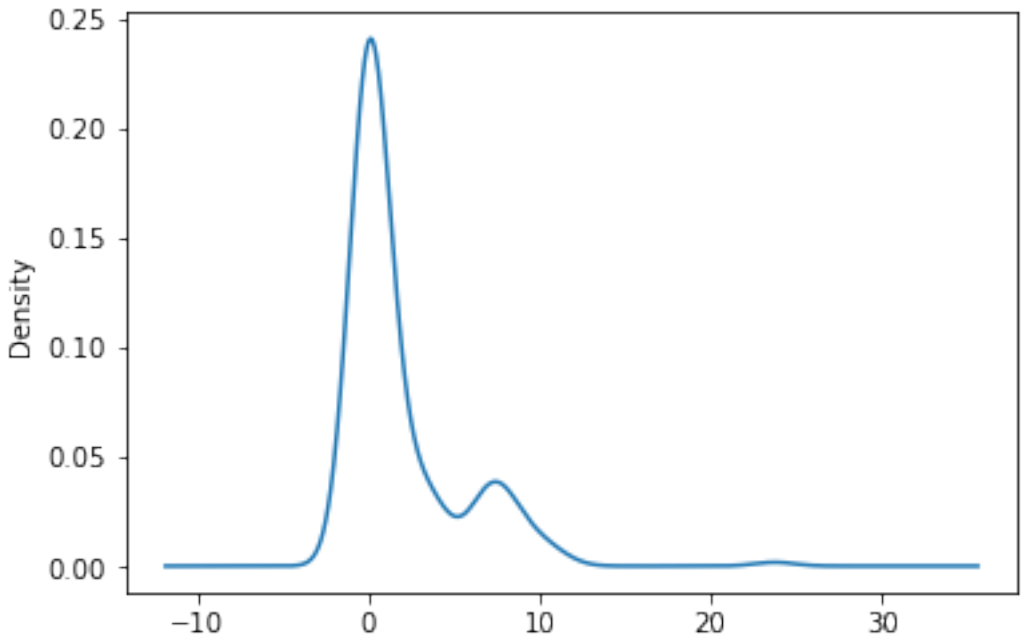
```
available values/total data = 0.503   unique values/available values = 0.248
################################################################################
```

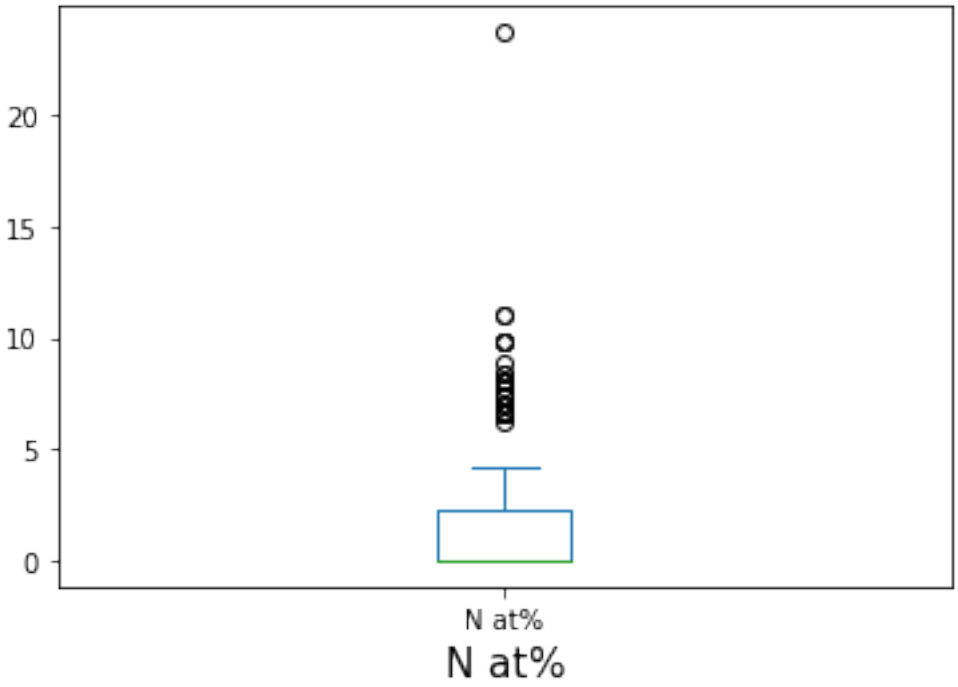




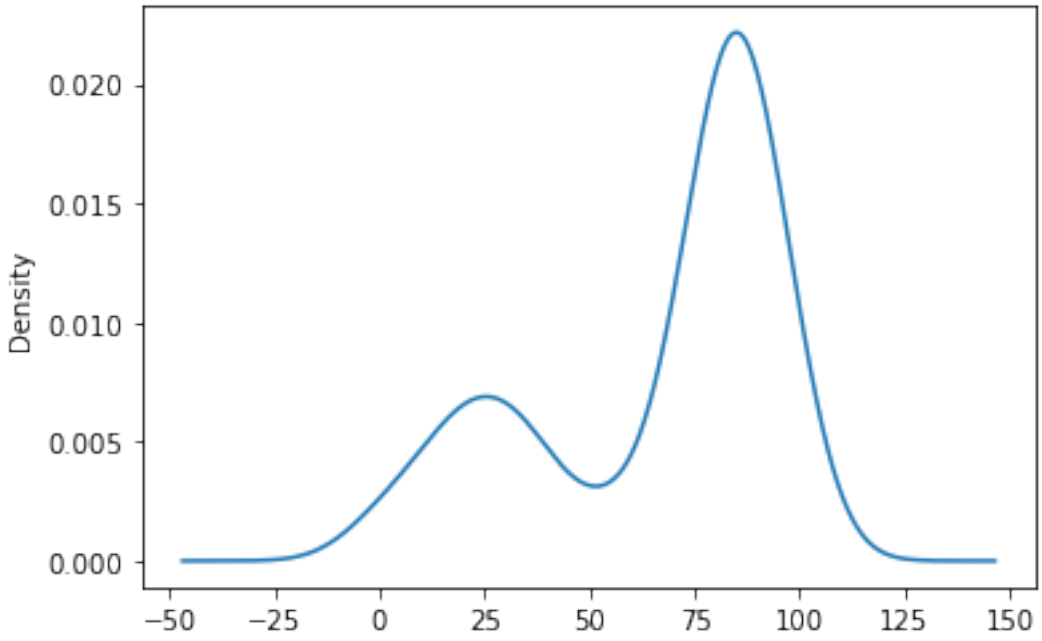
Ratio of ID/IG

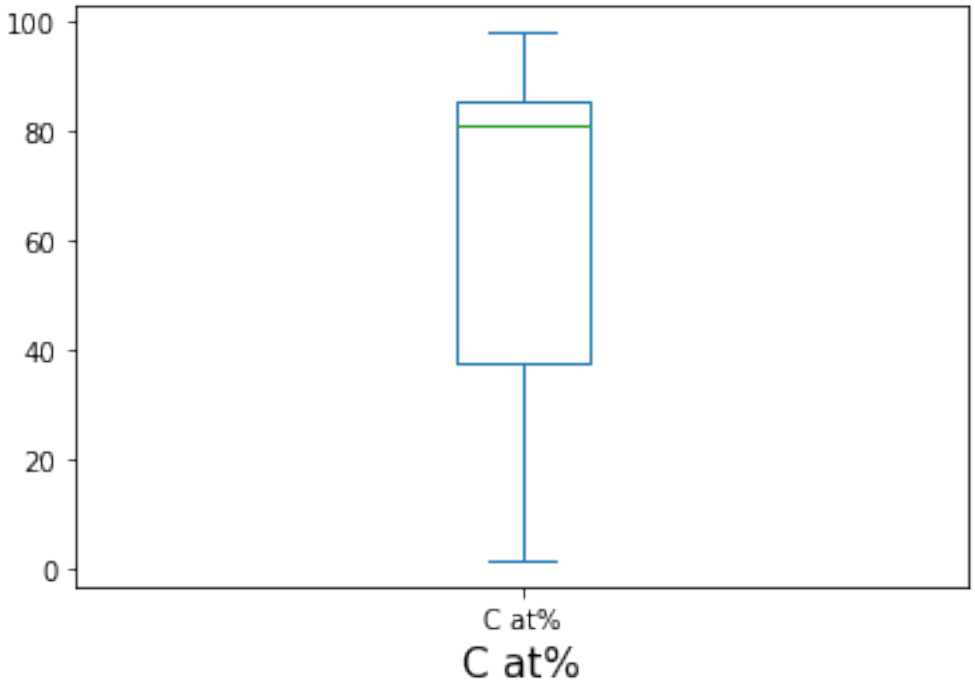
```
std= 0.43   mean= 1.05
count of available values= 189   count of missing values= 115   count of unique
values= 51
available values/total data = 0.622   unique values/available values = 0.270
################################################################################
```





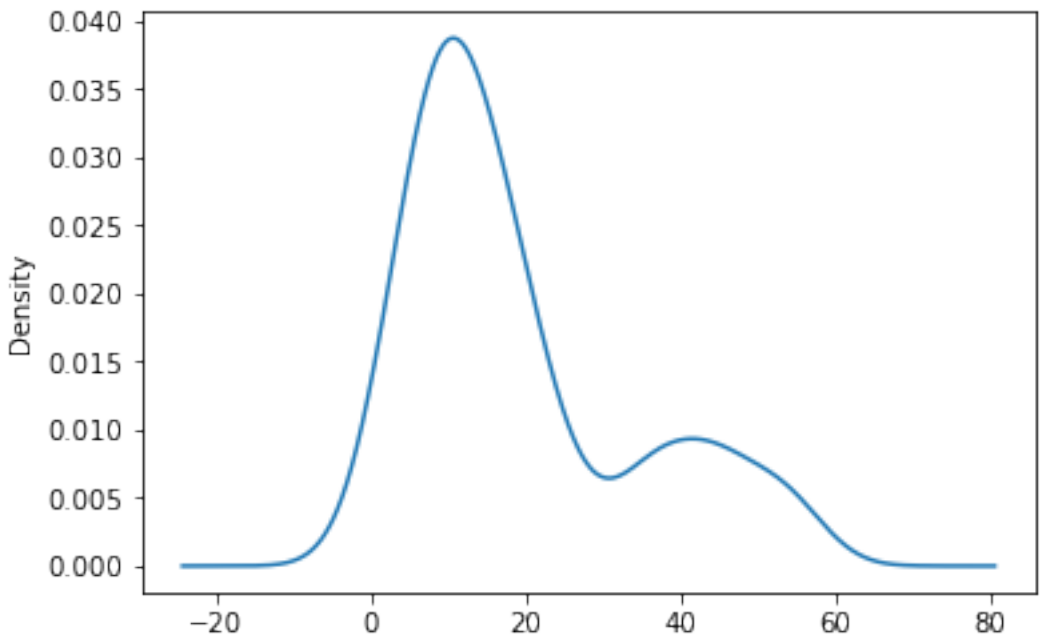

```
std= 3.34   mean= 1.86
count of available values= 199   count of missing values= 105   count of unique
values= 25
available values/total data = 0.655   unique values/available values = 0.126
################################################################################
```

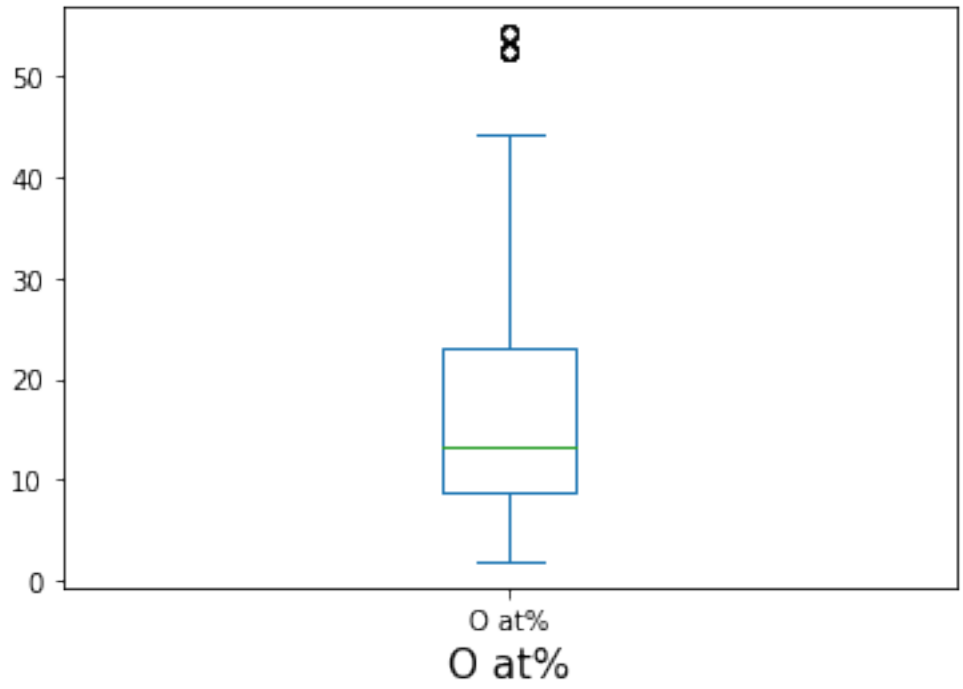




std= 29.11 mean= 66.56
count of available values= 198 count of missing values= 106 count of unique values= 59

```
available values/total data = 0.651   unique values/available values = 0.298
################################################################################
```





```
std= 14.68   mean= 18.96
count of available values= 192   count of missing values= 112   count of unique
values= 58
available values/total data = 0.632   unique values/available values = 0.302
################################################################################
```

[82]:
```
fillingMissingValues(df6)
```

```
column=  Specific Surface Area (m^2/g)   available ratio= 0.684   unique ratio=
0.308
mean= 399.334   std= 536.740
 missing values filled with 506.6816

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


column=  Pore Volume (cm^3/g)   available ratio= 0.503   unique ratio= 0.248
mean= 0.490   std= 0.620
 missing values filled with 0.8001

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


column=  Ratio of ID/IG   available ratio= 0.622   unique ratio= 0.270
mean= 1.054   std= 0.428
 missing values filled with 0.9685

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


column=  N at%   available ratio= 0.655   unique ratio= 0.126
mean= 1.860   std= 3.337
 missing values filled with 2.5277

################################################################################
```

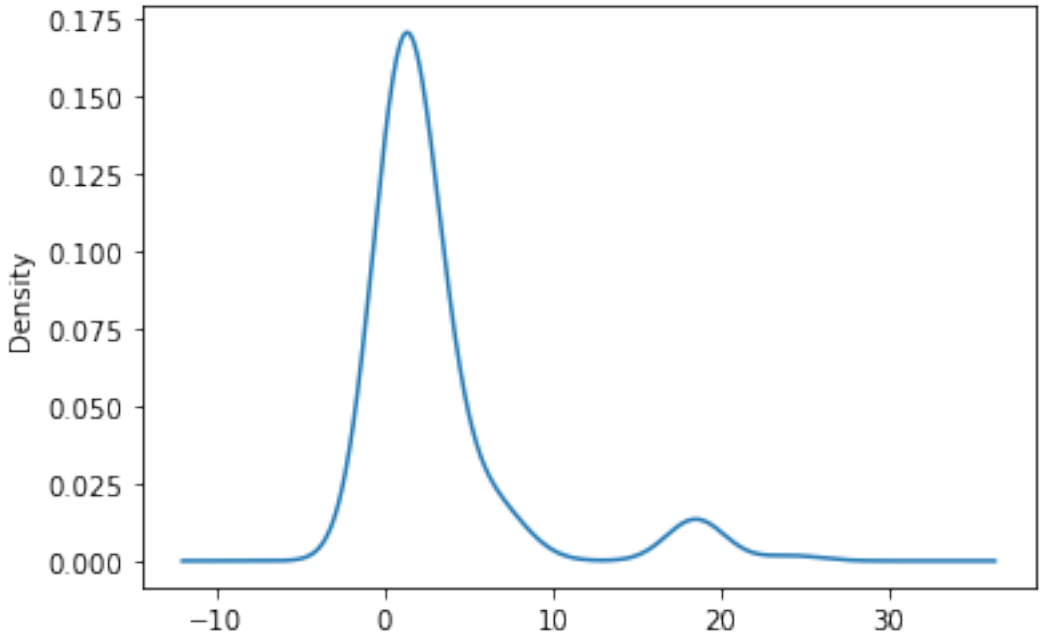
```
column=  C at%   available ratio= 0.651   unique ratio= 0.298
mean= 66.559   std= 29.111
 missing values filled with 60.7372

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


column=  O at%   available ratio= 0.632   unique ratio= 0.302
mean= 18.958   std= 14.684
 missing values filled with 16.0213

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



/tmp/ipykernel_270313/1691008062.py:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[column].fillna((mean+(std/5)), inplace= True)
/tmp/ipykernel_270313/1691008062.py:52: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[column].fillna((mean+(std/2)), inplace= True)
/tmp/ipykernel_270313/1691008062.py:34: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[column].fillna((mean-(std/5)), inplace= True)
/tmp/ipykernel_270313/1691008062.py:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[column].fillna((mean+(std/5)), inplace= True)
/tmp/ipykernel_270313/1691008062.py:34: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

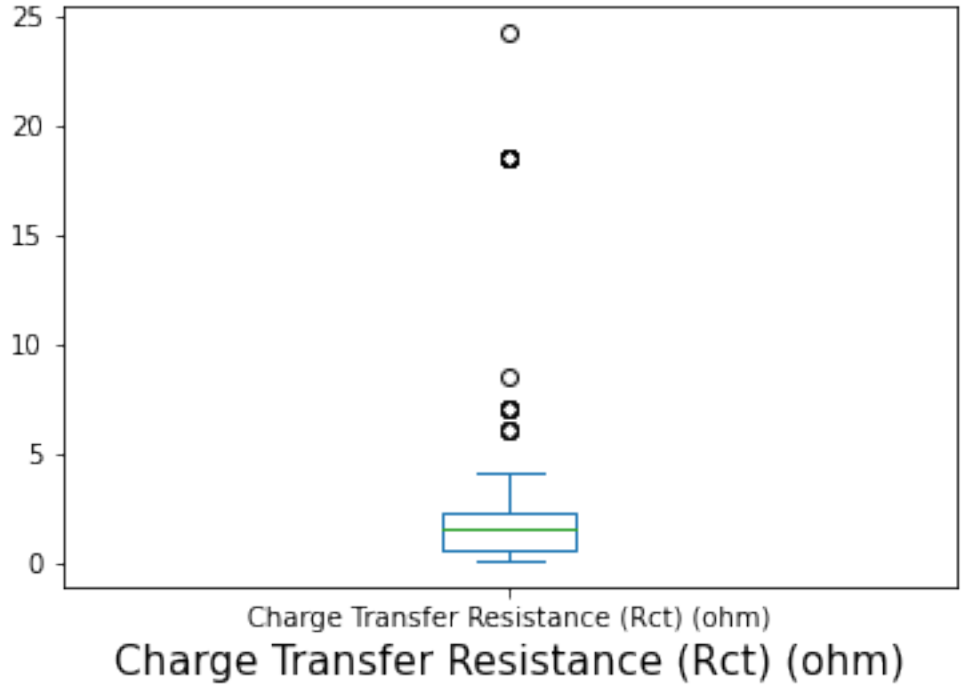
```
  df[column].fillna((mean-(std/5)), inplace= True)
/tmp/ipykernel_270313/1691008062.py:34: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[column].fillna((mean-(std/5)), inplace= True)
```

[83]:
```
df6.isnull().sum()
```

[83]:
```
Limits of Potential Window (V)                      0
Lower Limit of Potential Window (V)                 0
Upper Limit of Potential Window (V)                 0
Potential Window (V)                                0
Current Density (A/g)                               0
Capacitance (F/g)                                   0
Specific Surface Area (m^2/g)                       0
Charge Transfer Resistance (Rct) (ohm)            167
Equivalent Series Resistance (Rs) (ohm)           160
Electrode Material                                  0
Pore Size (nm)                                    167
Pore Volume (cm^3/g)                                0
Ratio of ID/IG                                      0
N at%                                               0
C at%                                               0
O at%                                               0
Electrolyte Chemical Formula                        0
Electrolyte Ion Mobility Ranking                    0
Electrolyte Concentration (M)                       0
Cell Configuration (three/two electrode system)     0
dtype: int64
```

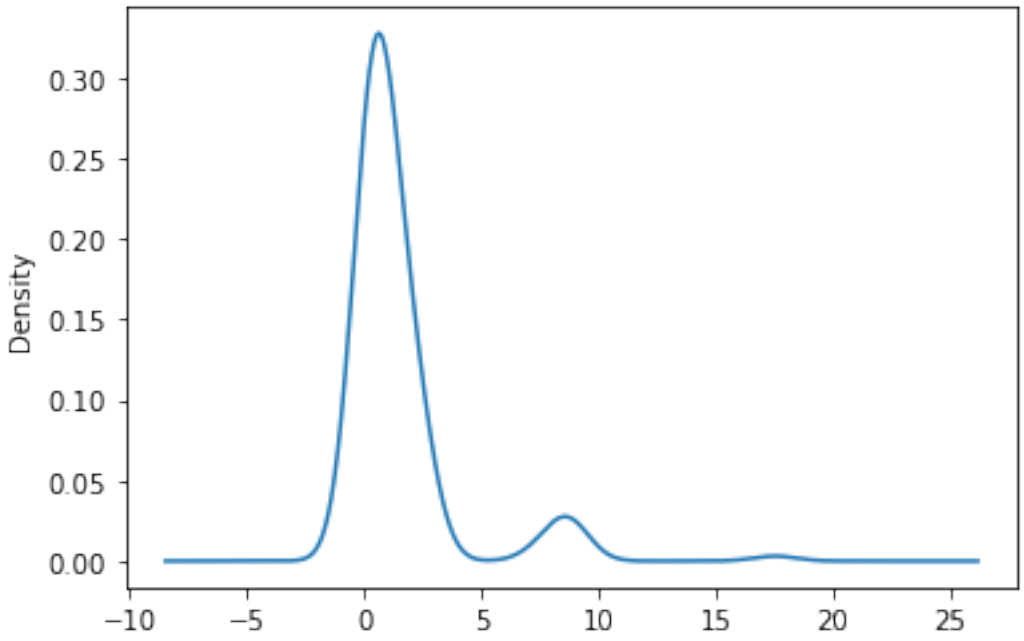
[84]:
```
NullColumnsPlot(df6)
```

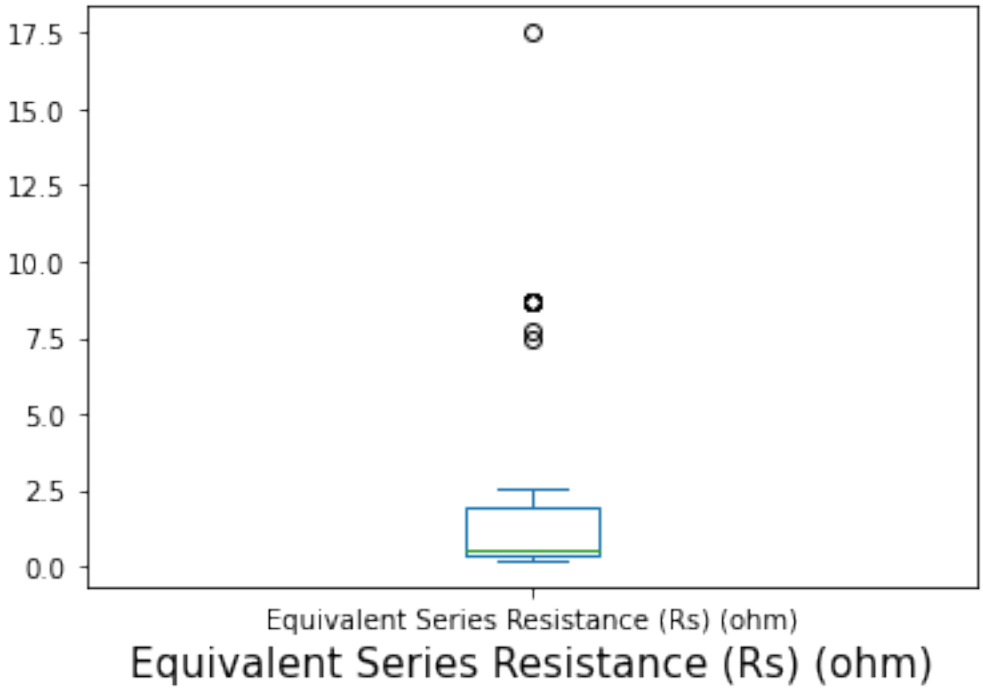




std= 4.63 mean= 3.03
count of available values= 137 count of missing values= 167 count of unique values= 38

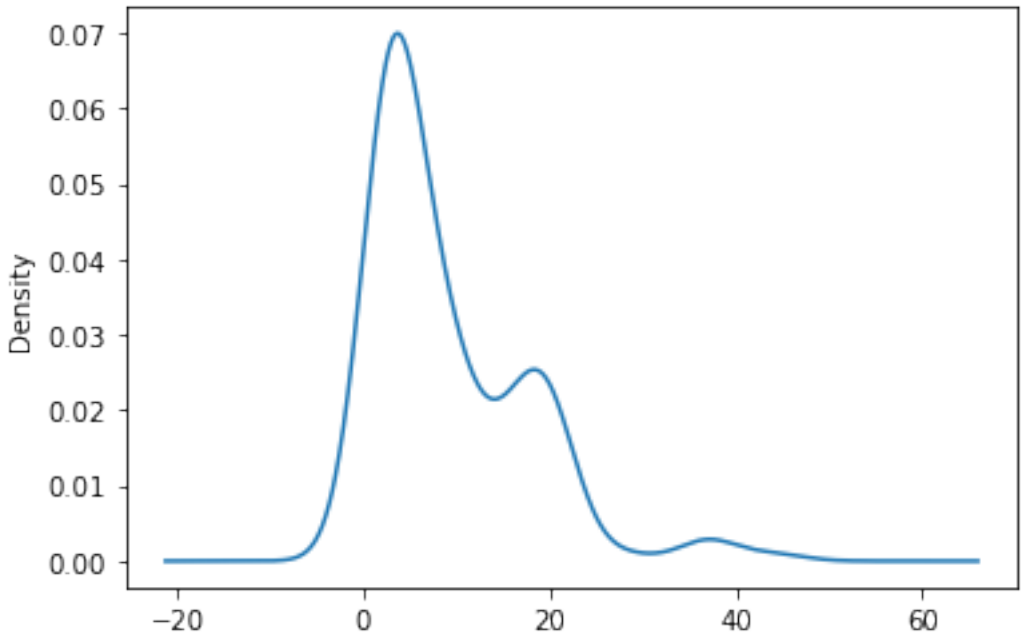
```
available values/total data = 0.451   unique values/available values = 0.277
################################################################################
```

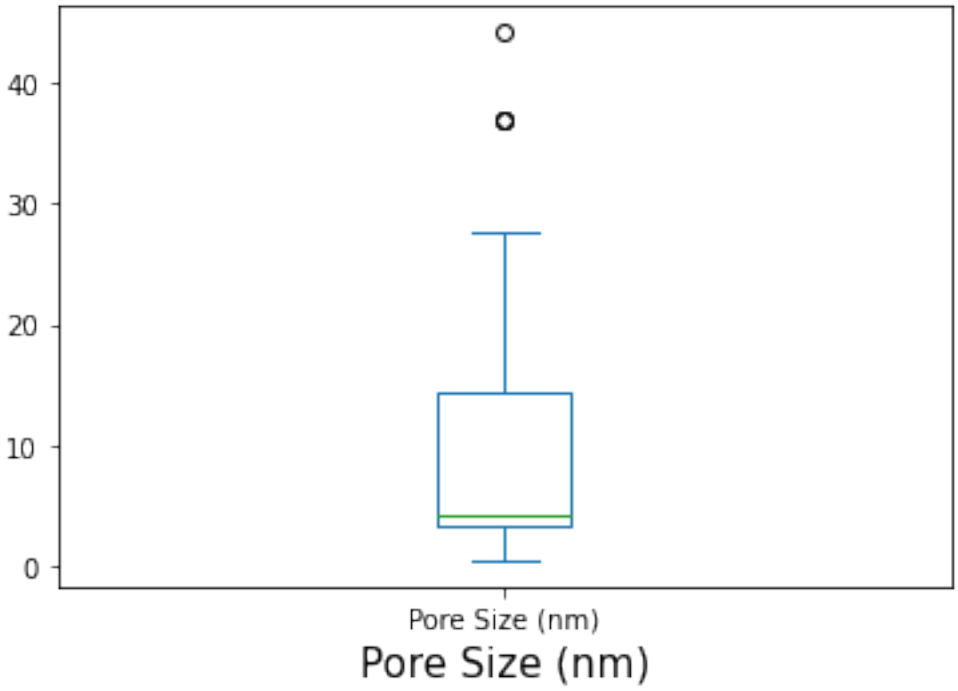




Equivalent Series Resistance (Rs) (ohm)

```
std= 2.46   mean= 1.51
count of available values= 144   count of missing values= 160   count of unique
values= 33
available values/total data = 0.474   unique values/available values = 0.229
################################################################################
```






```
std= 8.37   mean= 9.11
count of available values= 137   count of missing values= 167   count of unique
values= 39
available values/total data = 0.451   unique values/available values = 0.285
################################################################################
```

[85]:
```
EmptyColumnNames(df6)
```

[85]:
```
['Charge Transfer Resistance (Rct) (ohm)',
 'Equivalent Series Resistance (Rs) (ohm)',
 'Pore Size (nm)']
```

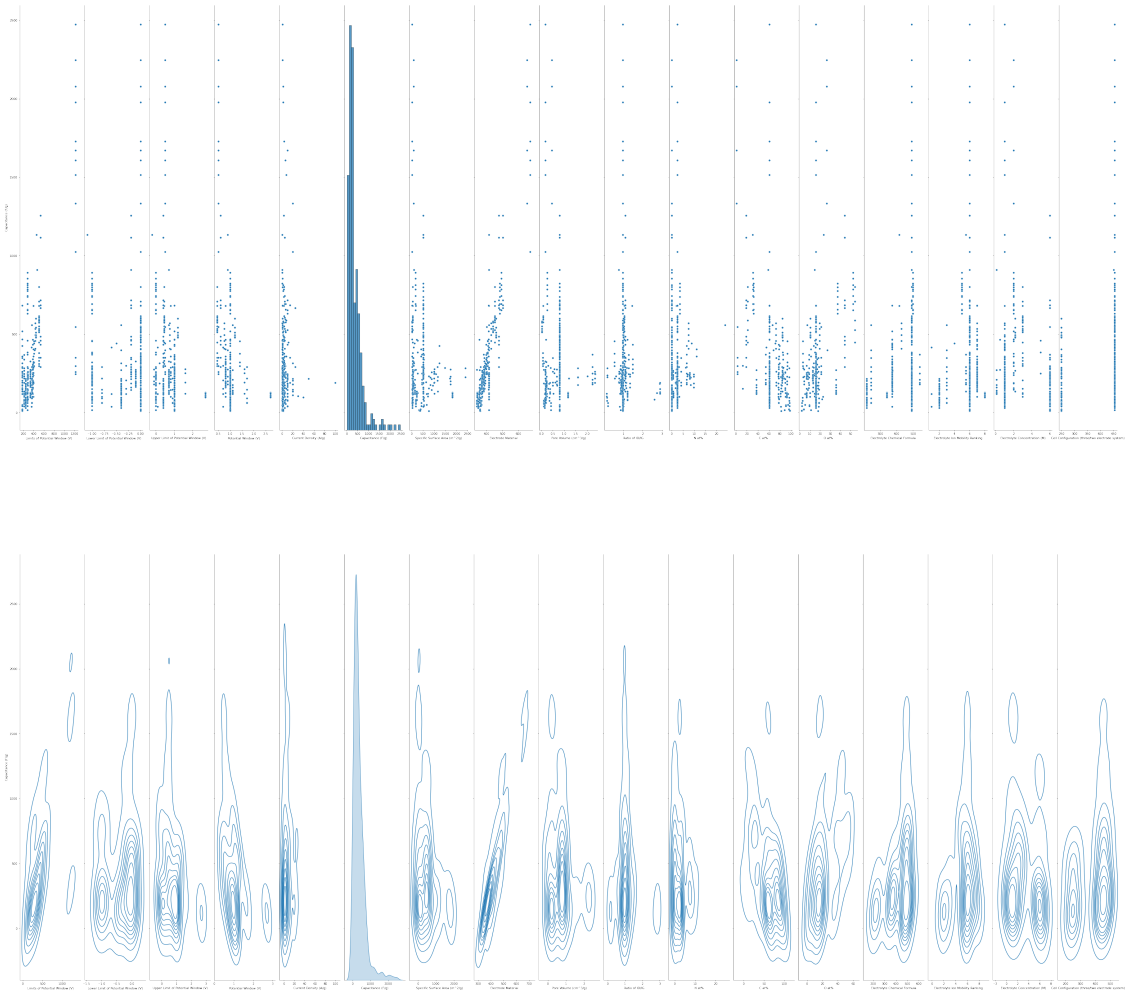
[86]:
```
df=df6.drop(columns=EmptyColumnNames(df6),axis=1)
```

[87]:
```
df.shape
```

[87]:
```
(304, 17)
```

[88]:
```
sns.pairplot(df,y_vars='Capacitance (F/g)',aspect=3/20,height=20)

sns.pairplot(df,kind='kde',y_vars='Capacitance (F/g)',aspect=3/20,height=20)
```

[88]: <seaborn.axisgrid.PairGrid at 0x7f7a047c2d90>



[89]:
```
df.shape
```

[89]: (304, 17)

[90]:
```
df.columns
```

[90]:
```
Index(['Limits of Potential Window (V)', 'Lower Limit of Potential Window (V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)', 'Capacitance (F/g)',
       'Specific Surface Area (m^2/g)', 'Electrode Material',
       'Pore Volume (cm^3/g)', 'Ratio of ID/IG', 'N at%', 'C at%', 'O at%',
       'Electrolyte Chemical Formula', 'Electrolyte Ion Mobility Ranking',
       'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)'],
      dtype='object')
```

[91]:
```
df.isnull().sum()
```

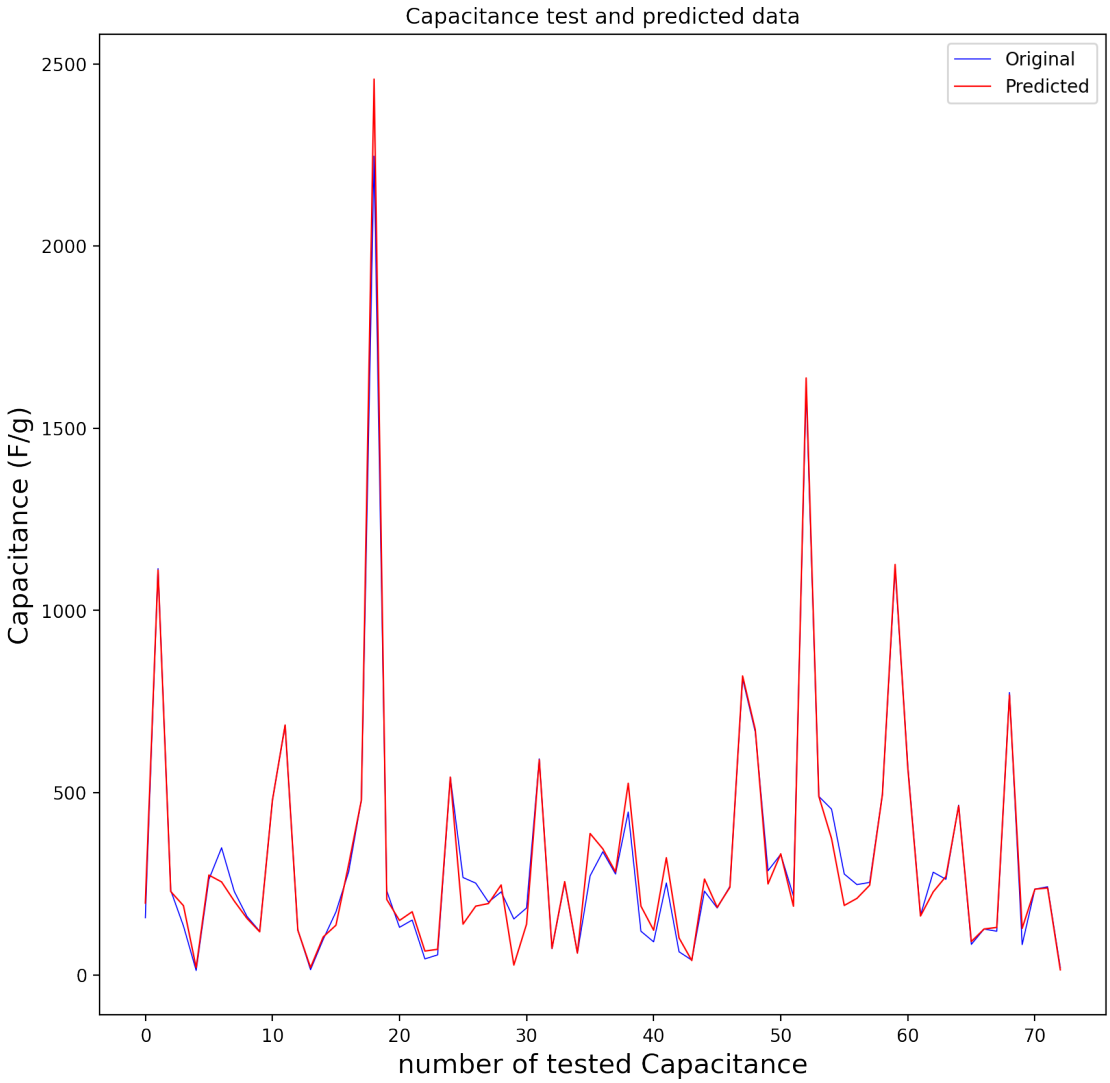
[91]:
```
Limits of Potential Window (V)                    0
Lower Limit of Potential Window (V)               0
Upper Limit of Potential Window (V)               0
Potential Window (V)                              0
Current Density (A/g)                             0
Capacitance (F/g)                                 0
Specific Surface Area (m^2/g)                     0
Electrode Material                                0
Pore Volume (cm^3/g)                              0
Ratio of ID/IG                                    0
N at%                                             0
C at%                                             0
O at%                                             0
Electrolyte Chemical Formula                      0
Electrolyte Ion Mobility Ranking                  0
Electrolyte Concentration (M)                     0
Cell Configuration (three/two electrode system)   0
dtype: int64
```

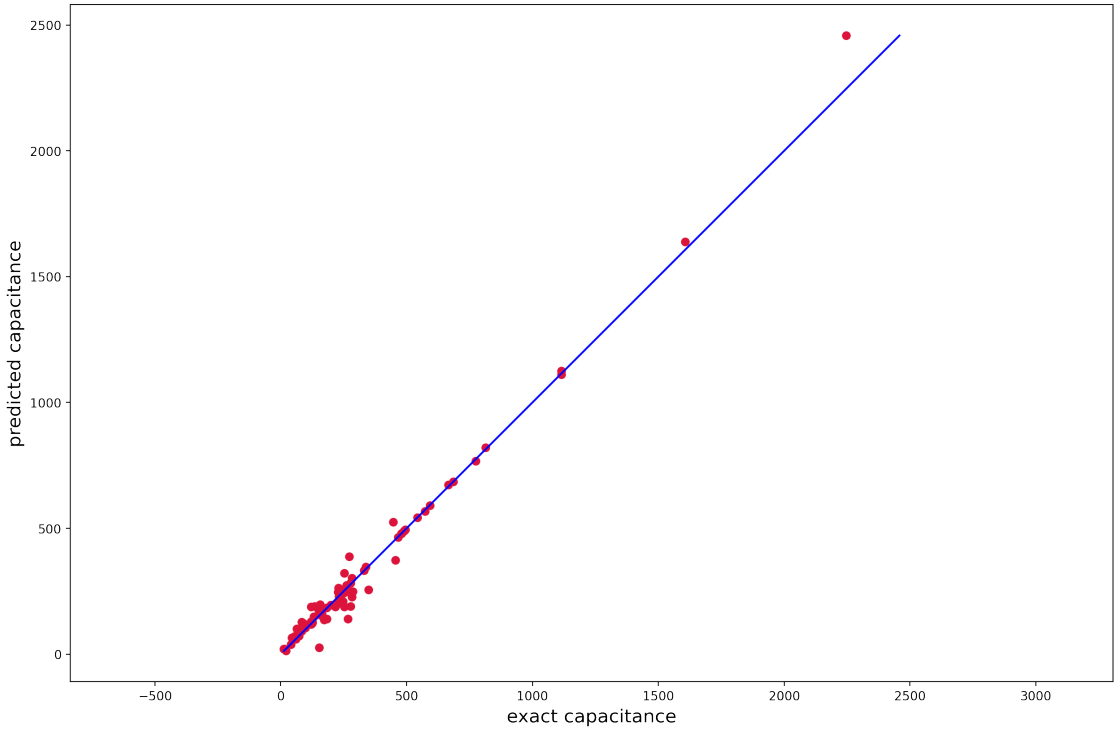
[92]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window␣
 ↪(V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)',
       'Specific Surface Area (m^2/g)', 'Electrode Material',
       'Pore Volume (cm^3/g)', 'Ratio of ID/IG', 'N at%', 'C at%', 'O at%',
       'Electrolyte Chemical Formula', 'Electrolyte Ion Mobility Ranking',
       'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']



X=df[Predictors].values
y=df[TargetVariable]



from sklearn.preprocessing import StandardScaler

PredictorScaler=StandardScaler()


PredictorScalerFit=PredictorScaler.fit(X)
```

```
X=PredictorScalerFit.transform(X)
y=y.values.ravel()


from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error , r2_score









opted=ExtraTreesRegressionScore()
```

[93]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],␣
 ↪random_state=opted[2])
etr=ExtraTreesRegressor(n_jobs=-1)
etr.fit(X_train,y_train)


Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)


mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)


TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions


from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()
```

```
plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)

print('\n')
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].head(60))
print(2*'\n')
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
print('number of tested data= %i'%count)
print('test_size"percent"= %.2f'%(opted[0]*100))
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
print('optimized MSE: %.4f ' %mse)
print('optimized RMSE: %.4f ' %mse**(0.5))
print('optimized random state: %i'%opted[2] )
```


Capacitance test and predicted data
Original
Predicted
Capacitance (F/g)
number of tested Capacitance
0
500
1000
1500
2000
2500
0
10
20
30
40
50
60
70



	Capacitance (F/g)	PredictedCapacitance (F/g)
0	157.00	197.2700
1	1115.00	1110.7020
2	232.00	228.9330
3	134.52	190.2030
4	12.40	20.2200
5	262.00	274.0417
6	349.00	255.3480
7	229.20	202.7640
8	160.60	155.2320
9	120.00	118.4470
10	479.00	479.0000
11	685.00	685.7440
12	123.70	122.8210
13	14.50	20.4600
14	97.80	104.4000
15	173.00	136.4860
16	283.00	303.0720
17	479.83	479.8300
18	2246.63	2458.1400
19	230.30	207.1660
20	130.70	149.5280

21	150.70	173.5140
22	44.10	65.6220
23	55.10	70.4120
24	542.73	542.7300
25	267.39	139.5035
26	252.00	188.8920
27	200.00	195.9481
28	228.00	247.0645
29	153.70	27.1600
30	183.90	139.5600
31	593.00	590.3000
32	72.00	73.2280
33	252.80	256.1087
34	60.00	60.0800
35	272.00	388.1010
36	338.00	346.1256
37	277.00	282.7148
38	447.13	526.0000
39	120.00	188.8920
40	90.90	122.8220
41	252.50	321.6560
42	63.60	101.6220
43	40.80	39.5787
44	230.00	263.2630
45	183.80	185.6810
46	243.00	240.4060
47	813.00	820.5000
48	666.60	672.8560
49	286.00	249.8085
50	331.00	332.4200
51	217.00	188.7067
52	1607.00	1638.1200
53	489.87	489.8700
54	455.00	374.2147
55	277.00	190.7600
56	248.00	210.2057
57	254.00	246.3110
58	494.84	494.8400
59	1115.00	1126.3560

```
total number of data= 304
number of trained data= 232
number of tested data= 72
test_size"percent"= 24.00
score for xtest and ytest : 0.9827
cross validation score: 0.8965
```

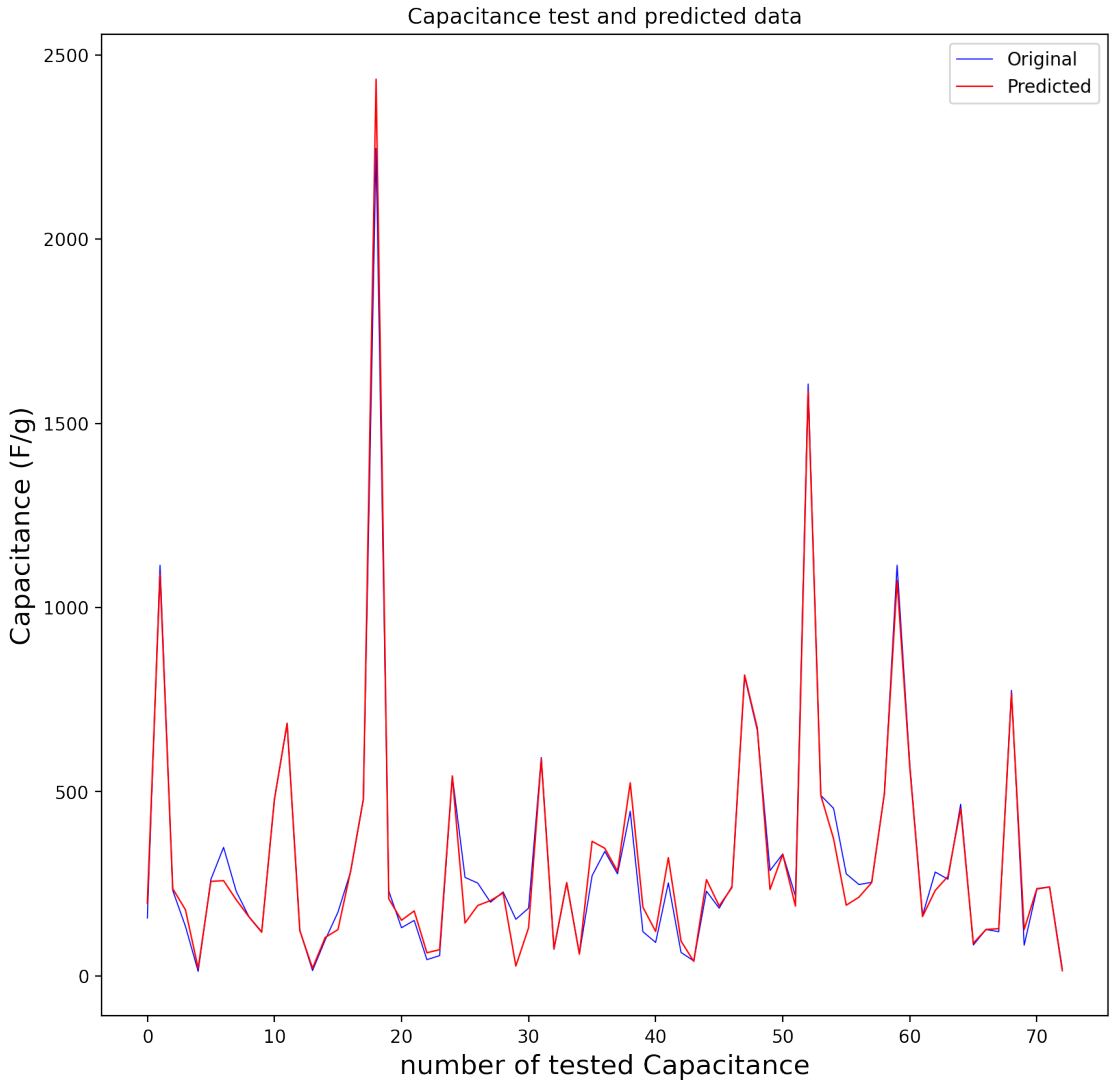
```
optimized MSE: 2175.1783
optimized RMSE: 46.6388
optimized random state: 7
```

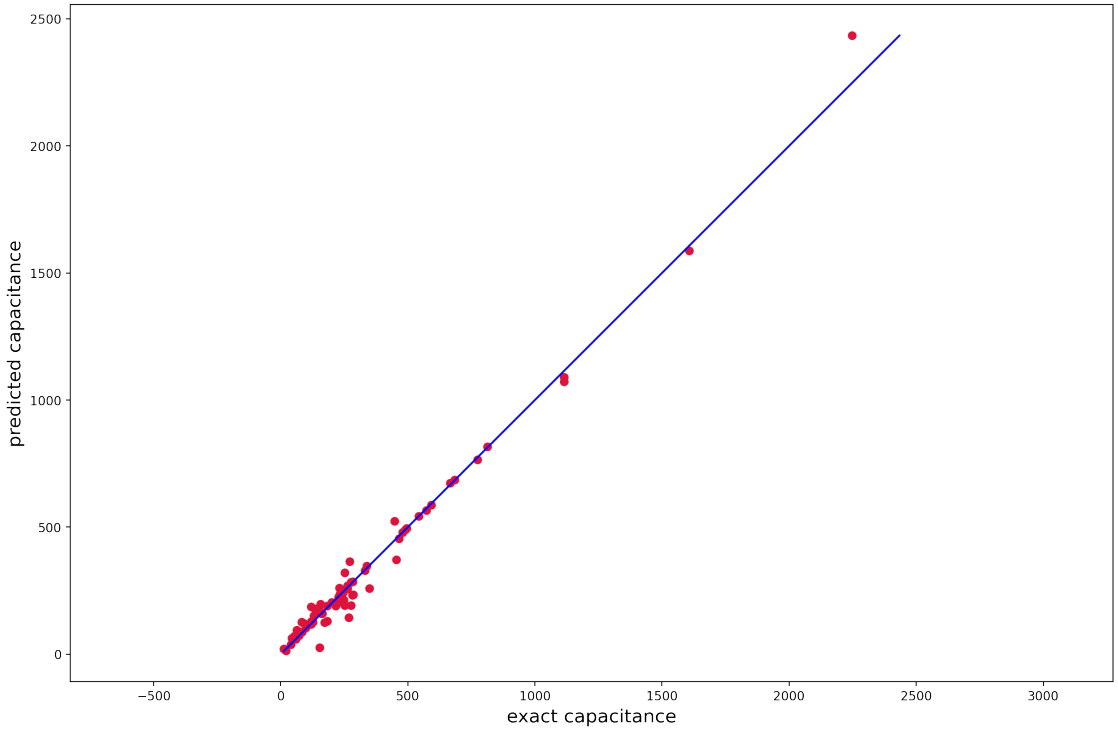
[94]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window␣
 ↪(V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)',
       'Specific Surface Area (m^2/g)', 'Electrode Material',
       'Pore Volume (cm^3/g)', 'Ratio of ID/IG', 'N at%', 'C at%', 'O at%',
       'Electrolyte Chemical Formula', 'Electrolyte Ion Mobility Ranking',
       'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']


X=df[Predictors].values
y=df[TargetVariable]


from sklearn.preprocessing import StandardScaler

PredictorScaler=StandardScaler()
PredictorScalerFit=PredictorScaler.fit(X)


X=PredictorScalerFit.transform(X)
y=y.values.ravel()


from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error , r2_score









opted=ExtraTreesRegressionMse()
```

[95]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],␣
 ↪random_state=opted[2])
etr=ExtraTreesRegressor(n_jobs=-1)
etr.fit(X_train,y_train)
```

```
Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)


mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)


TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions


from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()




plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)

print('\n')
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].head(60))
print(2*'\n')
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
```

```
print('number of tested data= %i'%count)
print('test_size"percent"= %.2f'%(opted[0]*100))
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
print('optimized MSE: %.4f ' %mse)
print('optimized RMSE: %.4f ' %mse**(0.5))
print('optimized random state: %i'%opted[2] )
```





	Capacitance (F/g)	PredictedCapacitance (F/g)
0	157.00	197.4890
1	1115.00	1089.8420
2	232.00	236.8550
3	134.52	179.5947
4	12.40	20.6560
5	262.00	256.6280
6	349.00	258.6580
7	229.20	206.5680
8	160.60	160.0760
9	120.00	118.5280
10	479.00	479.0000
11	685.00	685.7440
12	123.70	123.0680
13	14.50	20.9450
14	97.80	104.4000
15	173.00	125.6630
16	283.00	284.8880
17	479.83	479.8300
18	2246.63	2434.3500
19	230.30	209.4100
20	130.70	151.0090

21	150.70	176.4000
22	44.10	62.8120
23	55.10	71.2210
24	542.73	542.7300
25	267.39	143.5786
26	252.00	191.2370
27	200.00	204.4071
28	228.00	225.0540
29	153.70	26.8330
30	183.90	130.6270
31	593.00	587.9000
32	72.00	74.0400
33	252.80	253.0800
34	60.00	59.2400
35	272.00	365.4470
36	338.00	346.4388
37	277.00	284.0568
38	447.13	524.2400
39	120.00	186.3110
40	90.90	120.8080
41	252.50	321.0400
42	63.60	94.4330
43	40.80	39.3650
44	230.00	261.5000
45	183.80	189.8930
46	243.00	239.9020
47	813.00	817.0500
48	666.60	672.8560
49	286.00	234.7720
50	331.00	330.2600
51	217.00	189.6167
52	1607.00	1586.7600
53	489.87	489.8700
54	455.00	372.4874
55	277.00	191.9670
56	248.00	213.8217
57	254.00	253.2210
58	494.84	494.8400
59	1115.00	1072.8860

```
total number of data= 304
number of trained data= 232
number of tested data= 72
test_size"percent"= 24.00
score for xtest and ytest : 0.9844
cross validation score: 0.8950
```

```
optimized MSE: 1961.6391
optimized RMSE: 44.2904
optimized random state: 7
```

[96]:
```
df.shape
```

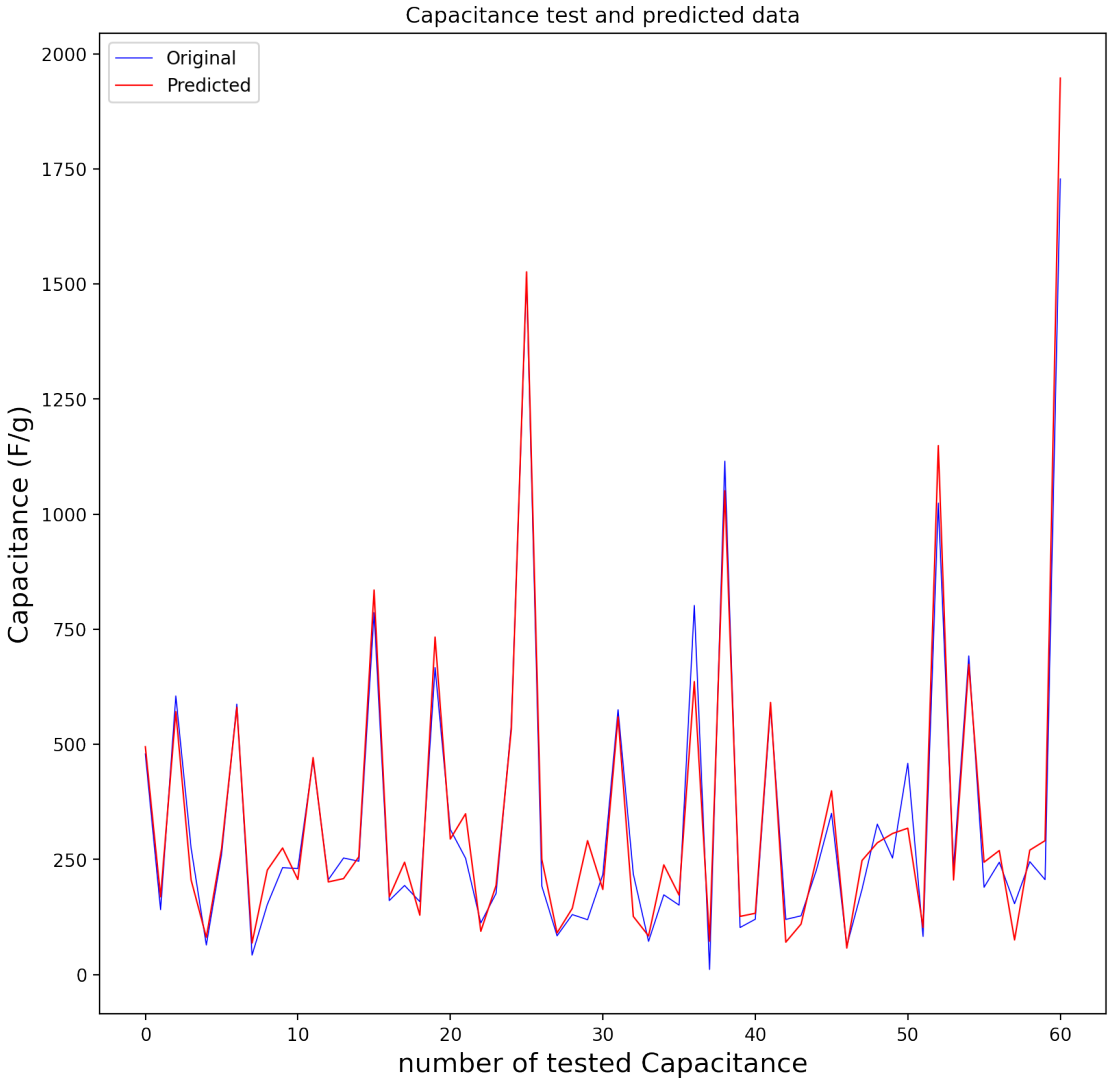
[96]:
```
(304, 17)
```

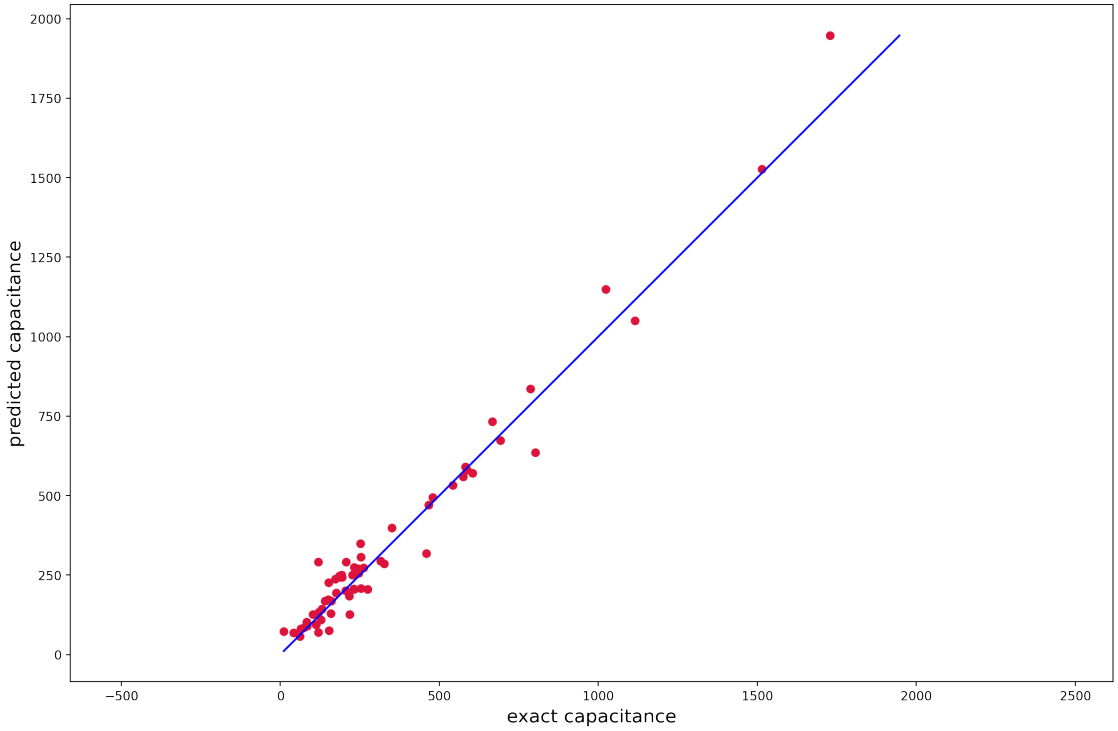
the following model is GBR on the latter df; df6

[97]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window␣
 ↪(V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)',
       'Specific Surface Area (m^2/g)', 'Electrode Material',
       'Pore Volume (cm^3/g)', 'Ratio of ID/IG', 'N at%', 'C at%', 'O at%',
       'Electrolyte Chemical Formula', 'Electrolyte Ion Mobility Ranking',
       'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']


X=df[Predictors].values
y=df[TargetVariable]


from sklearn.preprocessing import StandardScaler


PredictorScaler=StandardScaler()
PredictorScalerFit=PredictorScaler.fit(X)



X=PredictorScalerFit.transform(X)
y=y.values.ravel()



from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error , r2_score


opted=GradientBoostRegScore()
```

[98]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],␣
↪random_state=opted[2])
etr=GradientBoostingRegressor(random_state=opted[2])
etr.fit(X_train,y_train)


Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)


mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)

TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions


from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()




plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)
print('\n')
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].head(60))
print(2*'\n')
```

```
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
print('number of tested data= %i'%count)
print('test_size"percent"= %.2f'%(opted[0]*100))
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
print('optimized MSE: %.4f ' %mse)
print('optimized RMSE: %.4f ' %mse**(0.5))
print('optimized random state: %.4f'%opted[2] )
```





	Capacitance (F/g)	PredictedCapacitance (F/g)
0	479.00	494.704040
1	140.70	168.701307
2	605.00	570.949314
3	275.00	205.536608
4	64.00	80.711567
5	261.00	273.486865
6	587.30	580.283671
7	42.09	68.469587
8	152.00	226.763981
9	232.00	274.711635
10	230.00	206.437076
11	466.00	471.038262
12	205.80	201.020120
13	253.00	208.232629
14	246.00	255.662620
15	786.00	835.049054
16	160.60	168.701307
17	193.30	243.827234
18	158.20	128.871322
19	666.60	732.819910
20	315.00	294.178353

21	252.50	349.067997
22	112.00	93.889528
23	176.00	193.407506
24	542.73	532.778324
25	1514.00	1526.172674
26	192.00	250.172907
27	84.00	90.281174
28	130.10	143.692150
29	119.00	290.646157
30	217.00	184.754390
31	575.00	559.626515
32	218.10	126.000695
33	72.00	83.679075
34	173.00	238.111281
35	150.70	172.336162
36	801.60	636.125610
37	10.90	72.352014
38	1115.00	1050.710194
39	102.00	126.158815
40	120.00	132.940352
41	582.00	590.823664
42	119.60	70.187724
43	127.40	109.514904
44	226.00	249.917868
45	350.00	398.855389
46	62.00	57.384638
47	185.30	247.658873
48	326.60	286.034107
49	252.90	306.231690
50	458.70	317.763823
51	82.60	102.202386
52	1024.00	1148.816632
53	232.00	205.304468
54	692.00	673.567081
55	189.40	243.895628
56	243.80	269.207109
57	153.70	75.100949
58	245.00	270.102889
59	206.00	290.646157

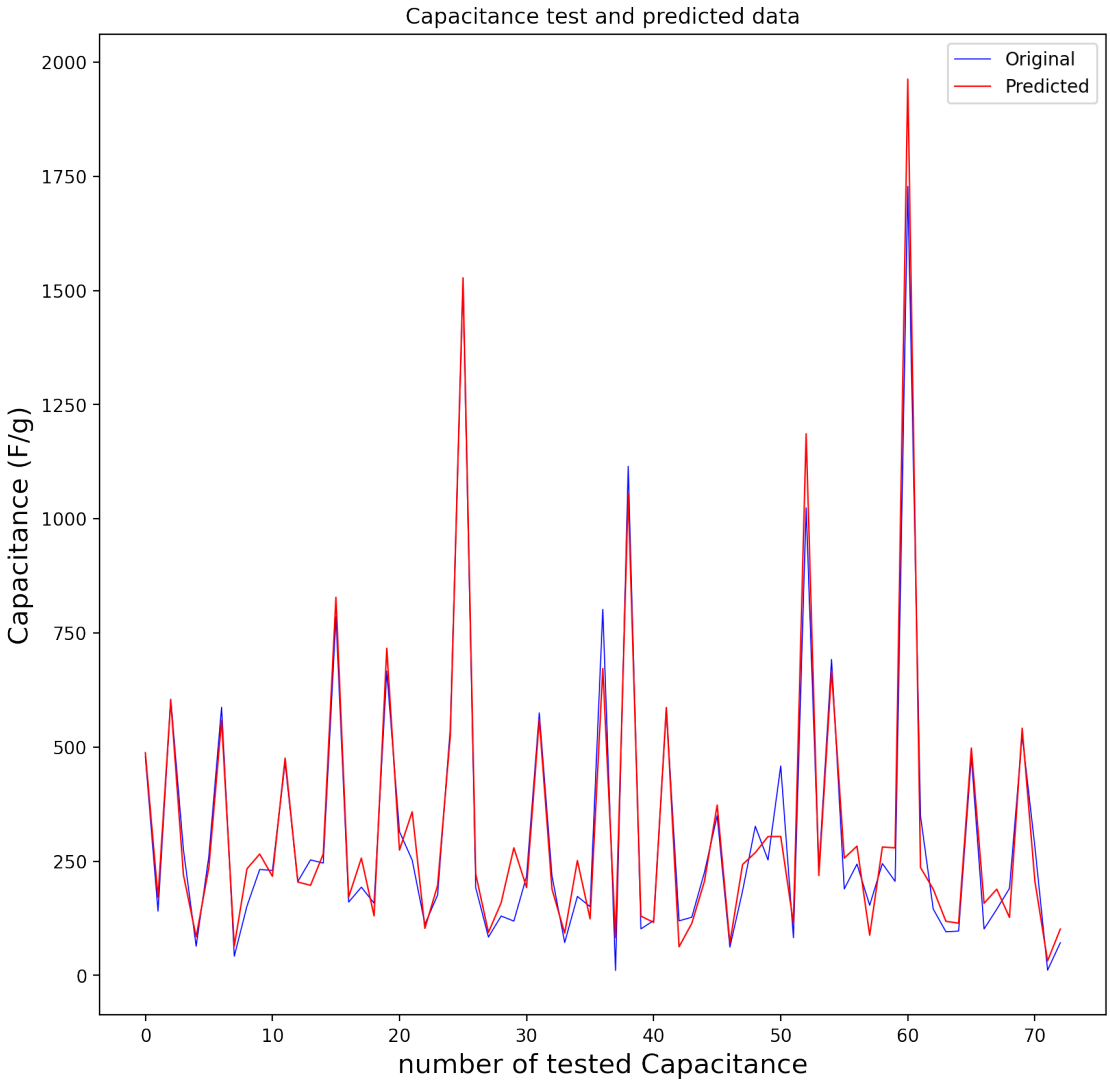
```
total number of data= 304
number of trained data= 244
number of tested data= 60
test_size"percent"= 20.00
score for xtest and ytest : 0.9647
cross validation score: 0.8853
```

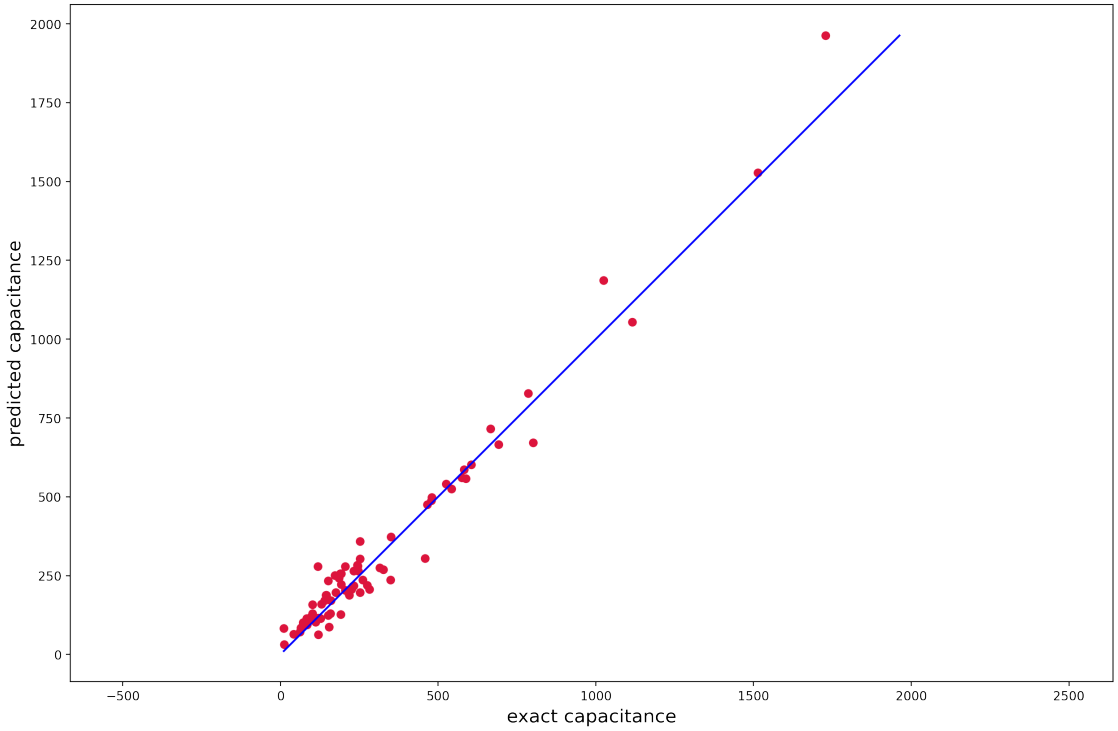
```
optimized MSE: 3927.6800
optimized RMSE: 62.6712
optimized random state: 9.0000
```

[99]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window␣
 ↪(V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)',
       'Specific Surface Area (m^2/g)', 'Electrode Material',
       'Pore Volume (cm^3/g)', 'Ratio of ID/IG', 'N at%', 'C at%', 'O at%',
       'Electrolyte Chemical Formula', 'Electrolyte Ion Mobility Ranking',
       'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']


X=df[Predictors].values
y=df[TargetVariable]


from sklearn.preprocessing import StandardScaler


PredictorScaler=StandardScaler()
PredictorScalerFit=PredictorScaler.fit(X)


X=PredictorScalerFit.transform(X)
y=y.values.ravel()

from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error , r2_score


opted=GradientBoostRegMse()
```

[100]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],␣
 ↪random_state=opted[2])
etr=GradientBoostingRegressor(random_state=opted[2])
etr.fit(X_train,y_train)


Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)
```

```
mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)

TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions


from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()




plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)
print('\n')
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].head(60))
print(2*'\n')
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
print('number of tested data= %i'%count)
print('test_size"percent"= %.2f'%(opted[0]*100))
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
print('optimized MSE: %.4f ' %mse)
print('optimized RMSE: %.4f ' %mse**(0.5))
print('optimized random state: %.4f'%opted[2] )
```


Capacitance test and predicted data
Original
Predicted
Capacitance (F/g)
number of tested Capacitance
0
250
500
750
1000
1250
1500
1750
2000
0
10
20
30
40
50
60
70



	Capacitance (F/g)	PredictedCapacitance (F/g)
0	479.00	487.714531
1	140.70	171.365245
2	605.00	602.210308
3	275.00	219.923255
4	64.00	84.577833
5	261.00	236.305783
6	587.30	558.631483
7	42.09	64.349933
8	152.00	233.554492
9	232.00	265.837656
10	230.00	217.125256
11	466.00	475.850286
12	205.80	204.614972
13	253.00	197.328811
14	246.00	265.815011
15	786.00	828.161241
16	160.60	171.365245
17	193.30	256.912370
18	158.20	130.457304
19	666.60	716.476429
20	315.00	274.574161

21	252.50	358.458845
22	112.00	103.113225
23	176.00	197.069459
24	542.73	525.037707
25	1514.00	1527.682270
26	192.00	221.952494
27	84.00	94.182739
28	130.10	159.235900
29	119.00	279.397856
30	217.00	192.533076
31	575.00	560.472516
32	218.10	189.044132
33	72.00	92.550821
34	173.00	251.474637
35	150.70	123.855462
36	801.60	672.272401
37	10.90	83.195146
38	1115.00	1054.578616
39	102.00	129.841219
40	120.00	116.169349
41	582.00	586.757772
42	119.60	62.742123
43	127.40	113.895789
44	226.00	206.367395
45	350.00	373.140007
46	62.00	71.521708
47	185.30	242.894788
48	326.60	268.931009
49	252.90	304.066398
50	458.70	304.303422
51	82.60	114.530773
52	1024.00	1186.136548
53	232.00	218.823228
54	692.00	666.124055
55	189.40	256.884197
56	243.80	283.180907
57	153.70	88.007652
58	245.00	281.240320
59	206.00	279.397856

```
total number of data= 304
number of trained data= 232
number of tested data= 72
test_size"percent"= 24.00
score for xtest and ytest : 0.9622
cross validation score: 0.8837
```

```
optimized MSE: 3755.2242
optimized RMSE: 61.2799
optimized random state: 9.0000
```

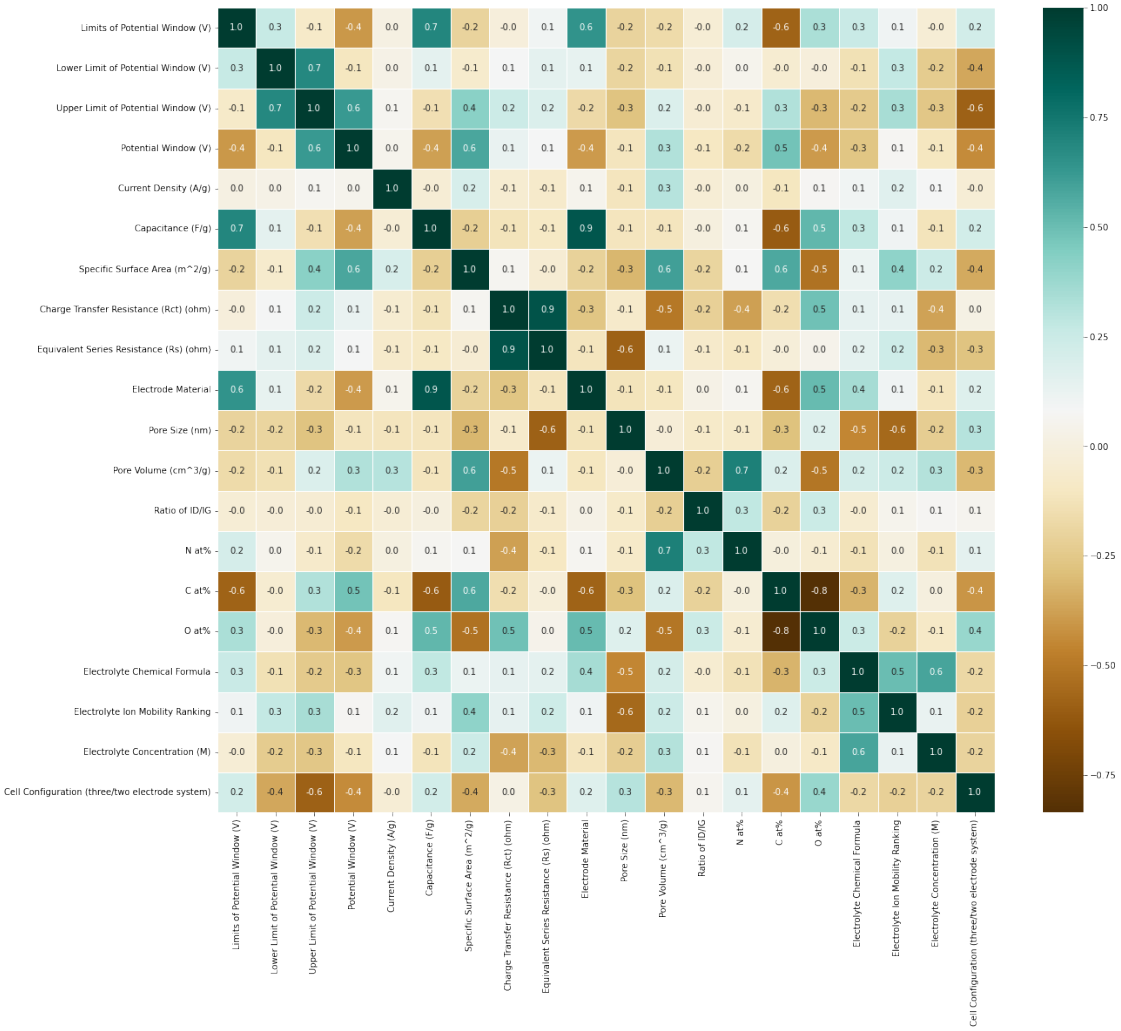
[101]:
```
df7=dframe[7]
```

[102]:
```
df7.shape
```

[102]: (340, 20)

[103]:
```
plt.figure(figsize=(20,17))
heatmap(df7.corr(),annot=True,fmt='.1f',linewidths=0.5,cmap='BrBG')
```

[103]: <AxesSubplot:>



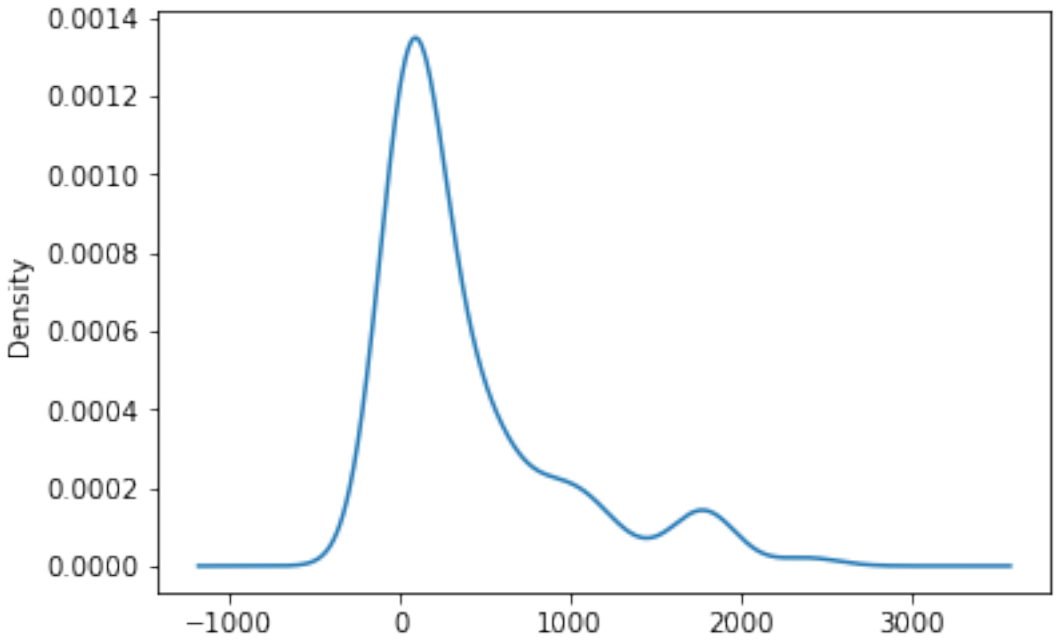
[104]:
```
df7.isnull().sum()
```

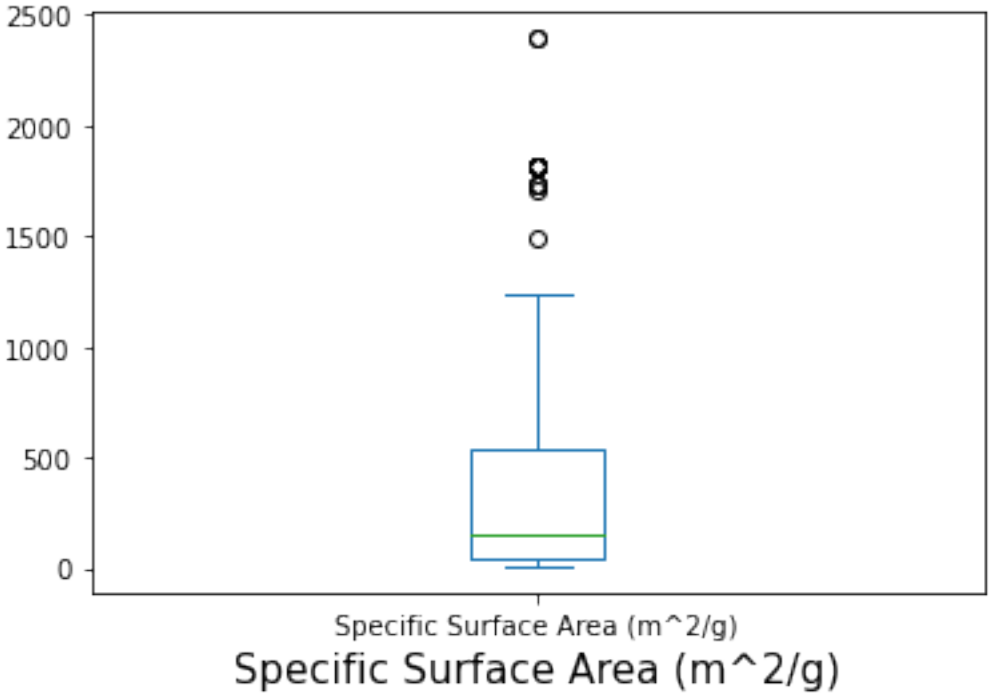
[104]:
```
Limits of Potential Window (V)                      0
Lower Limit of Potential Window (V)                 0
Upper Limit of Potential Window (V)                 0
Potential Window (V)                                0
Current Density (A/g)                               0
Capacitance (F/g)                                   0
Specific Surface Area (m^2/g)                     126
Charge Transfer Resistance (Rct) (ohm)            203
Equivalent Series Resistance (Rs) (ohm)           189
Electrode Material                                  0
Pore Size (nm)                                    195
Pore Volume (cm^3/g)                              182
Ratio of ID/IG                                    141
N at%                                             117
C at%                                             118
O at%                                             124
Electrolyte Chemical Formula                        0
Electrolyte Ion Mobility Ranking                    0
Electrolyte Concentration (M)                       0
Cell Configuration (three/two electrode system)     0
dtype: int64
```

[105]:
```
EmptyColumnNames(df7)
```

[105]:
```
['Specific Surface Area (m^2/g)',
 'Charge Transfer Resistance (Rct) (ohm)',
 'Equivalent Series Resistance (Rs) (ohm)',
 'Pore Size (nm)',
 'Pore Volume (cm^3/g)',
 'Ratio of ID/IG',
 'N at%',
 'C at%',
 'O at%']
```

[106]:
```
NullColumnsPlot(df7)
```

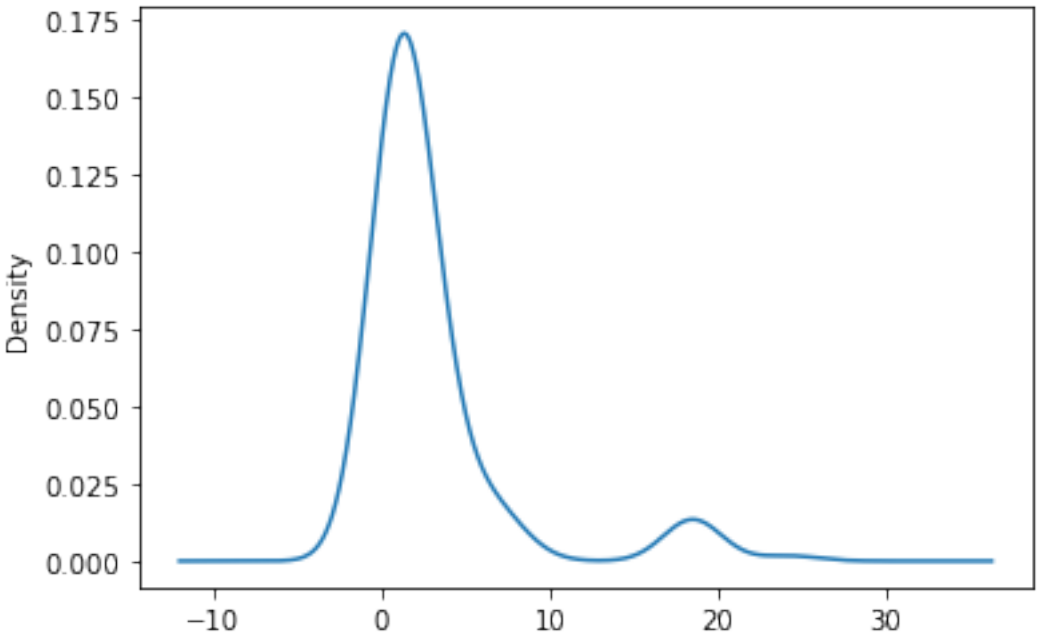


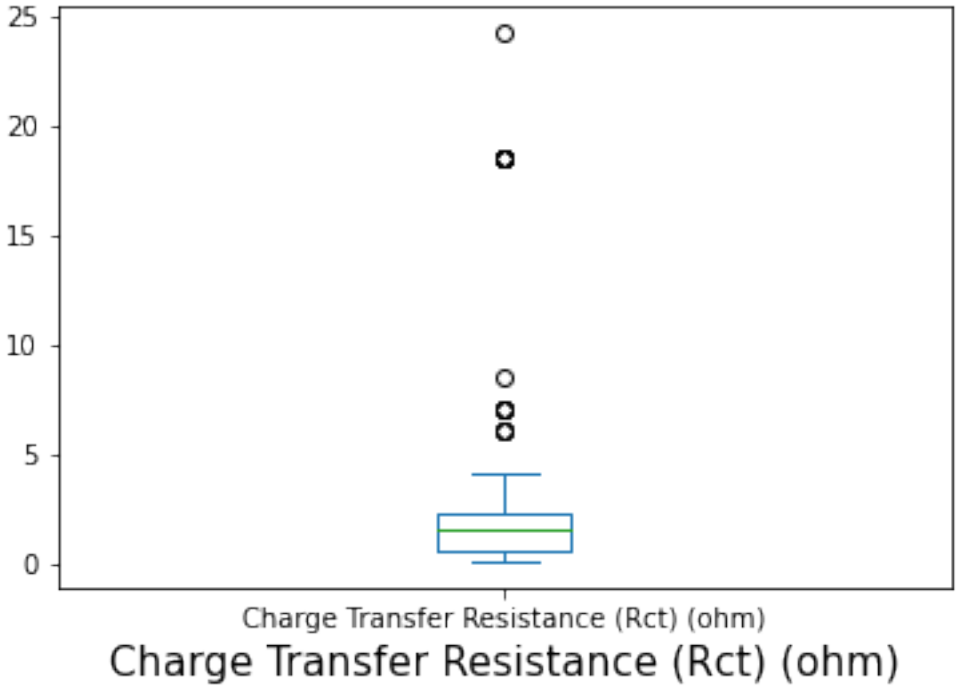


Specific Surface Area (m^2/g)

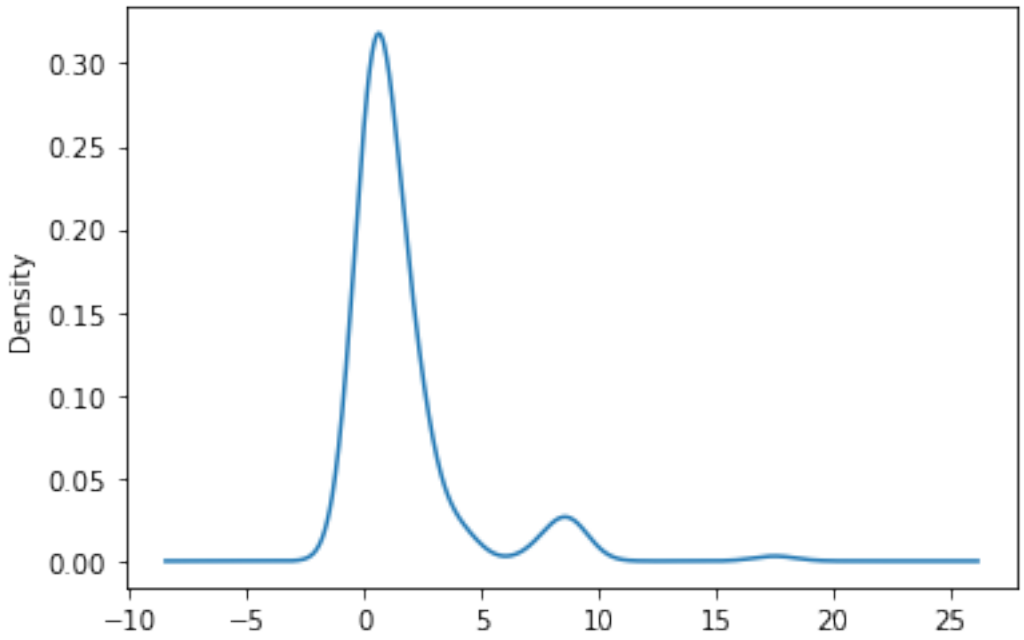
std= 533.76 mean= 397.19
count of available values= 214 count of missing values= 126 count of unique values= 68

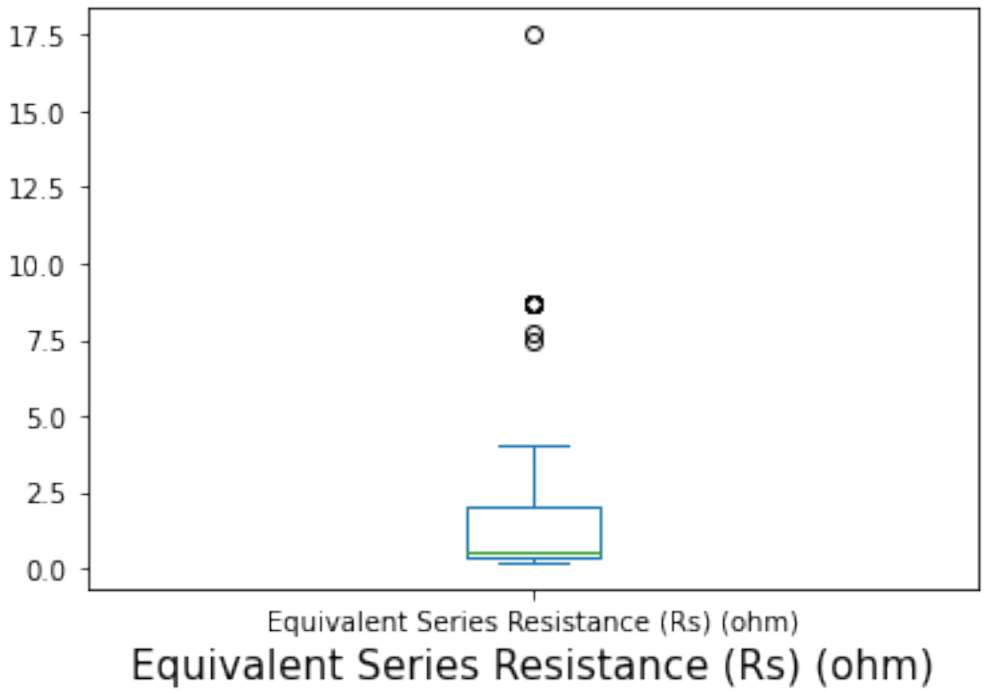
```
available values/total data = 0.629   unique values/available values = 0.318
################################################################################
```



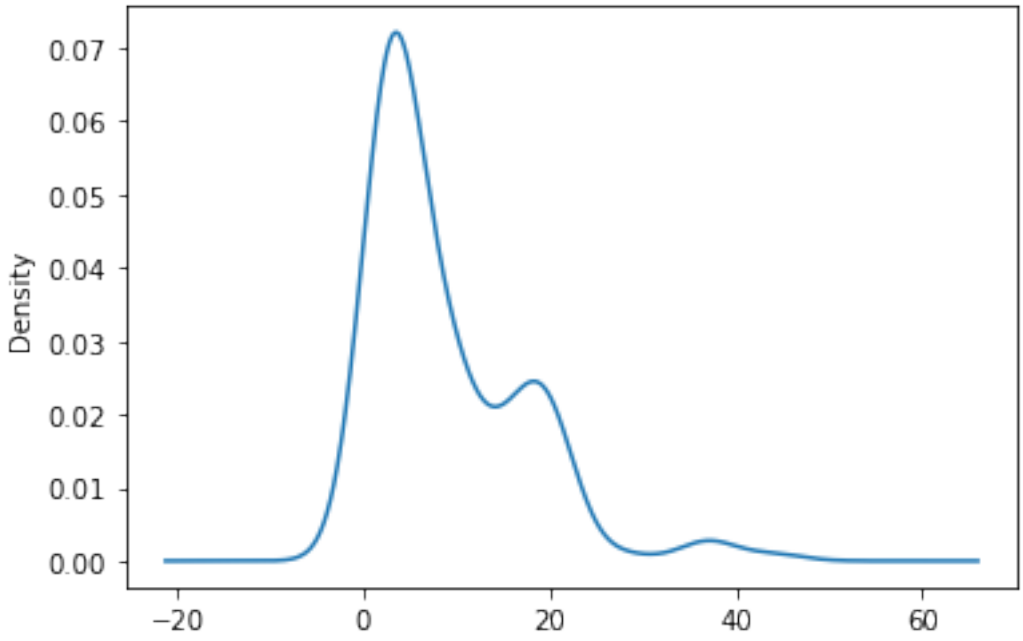


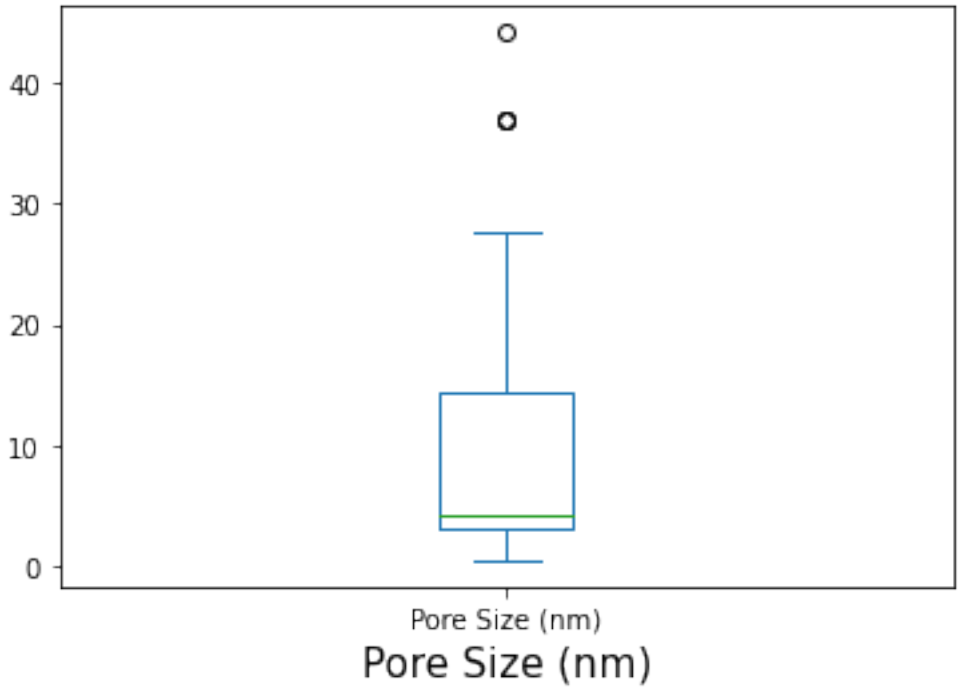
```
std= 4.63   mean= 3.03
count of available values= 137   count of missing values= 203   count of unique
values= 38
available values/total data = 0.403   unique values/available values = 0.277
################################################################################
```





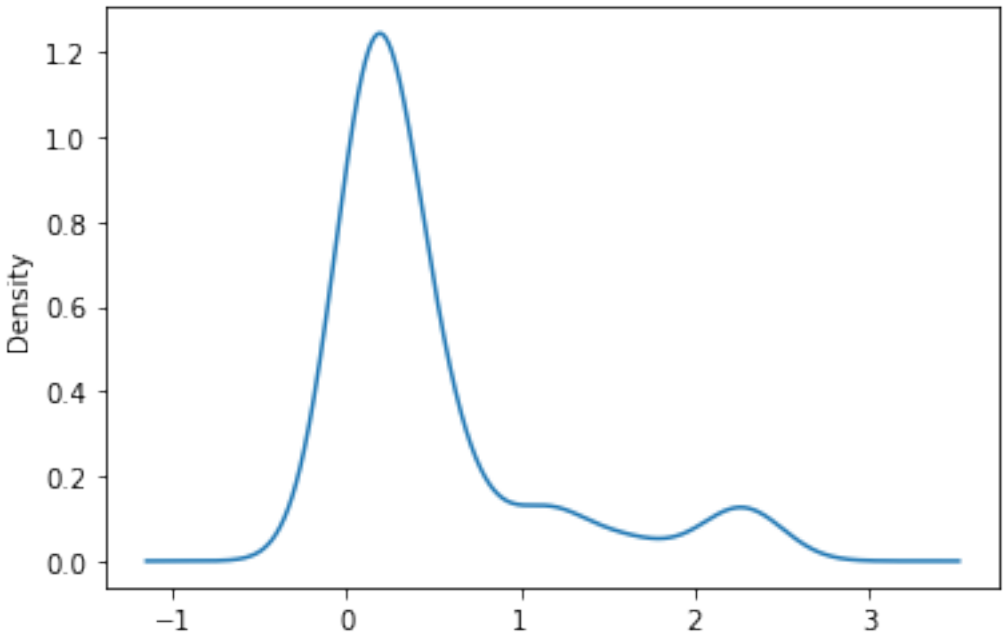

```
std= 2.45   mean= 1.60
count of available values= 151   count of missing values= 189   count of unique
values= 35
available values/total data = 0.444   unique values/available values = 0.232
################################################################################
```

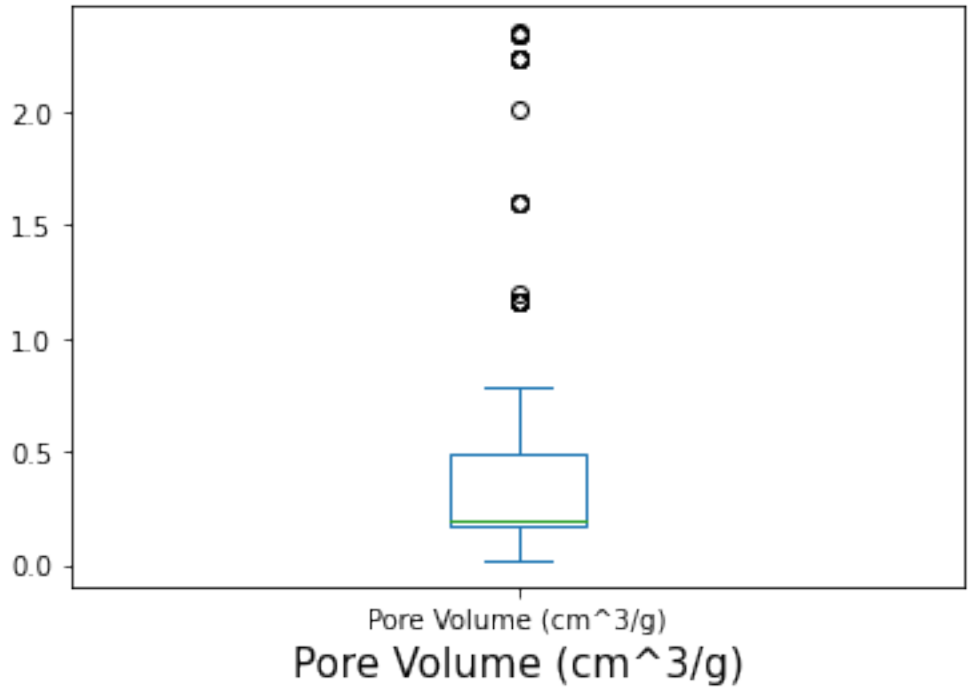




std= 8.27 mean= 8.84
count of available values= 145 count of missing values= 195 count of unique values= 41

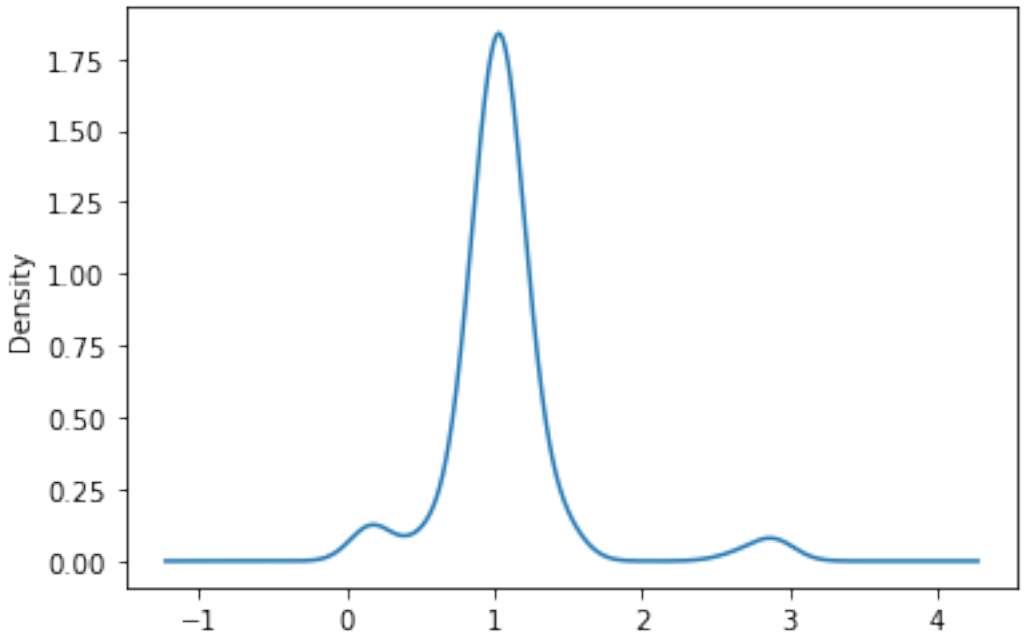
```
available values/total data = 0.426   unique values/available values = 0.283
################################################################################
```

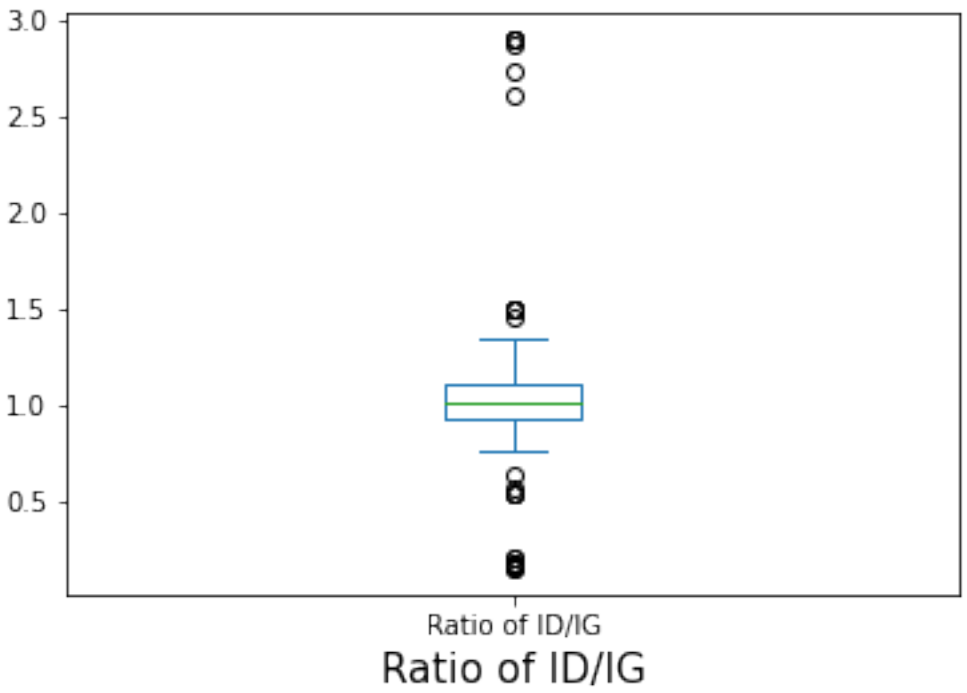




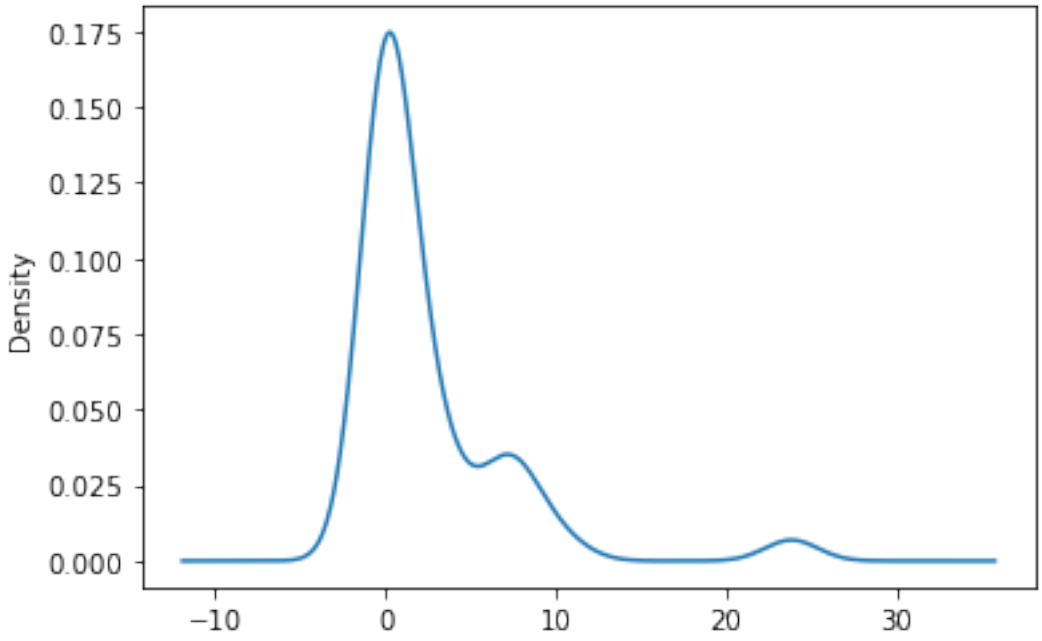
Pore Volume (cm^3/g)

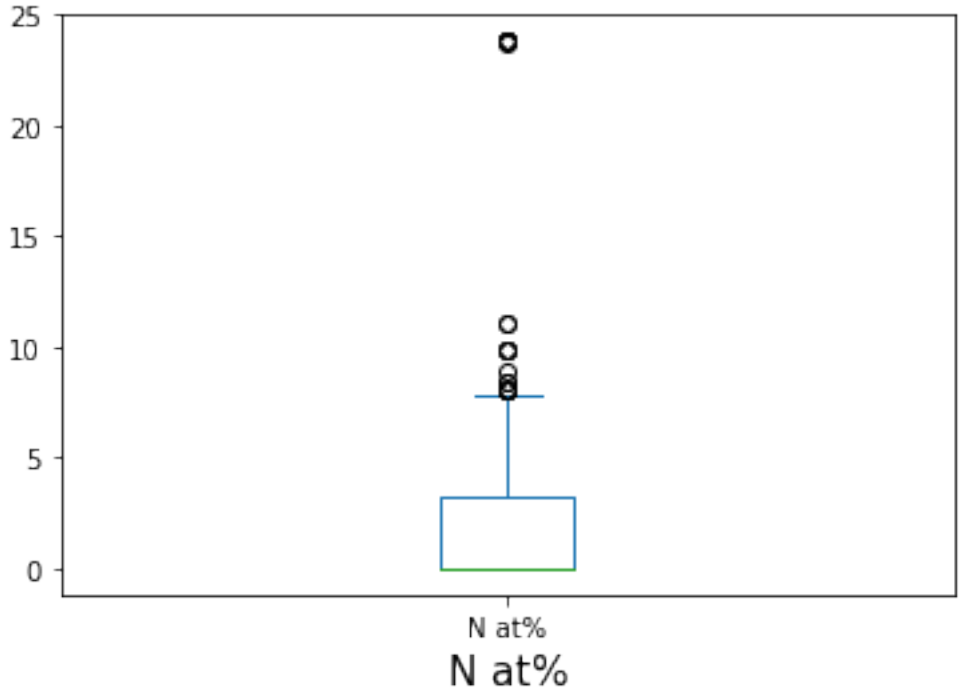
```
std= 0.61   mean= 0.48
count of available values= 158   count of missing values= 182   count of unique
values= 41
available values/total data = 0.465   unique values/available values = 0.259
################################################################################
```





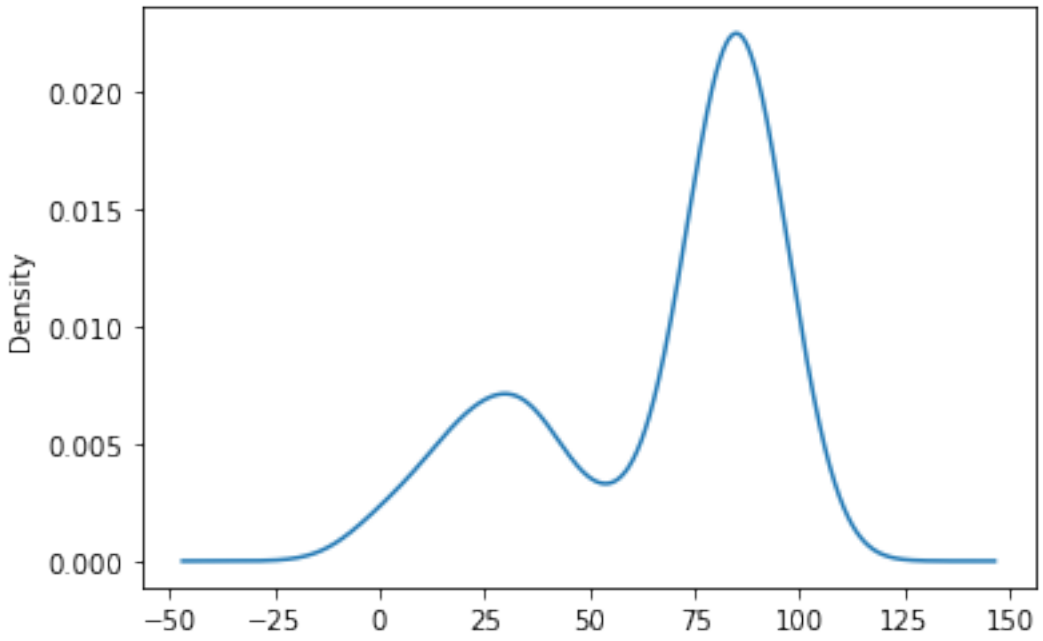

```
std= 0.42   mean= 1.06
count of available values= 199   count of missing values= 141   count of unique
values= 52
available values/total data = 0.585   unique values/available values = 0.261
################################################################################
```

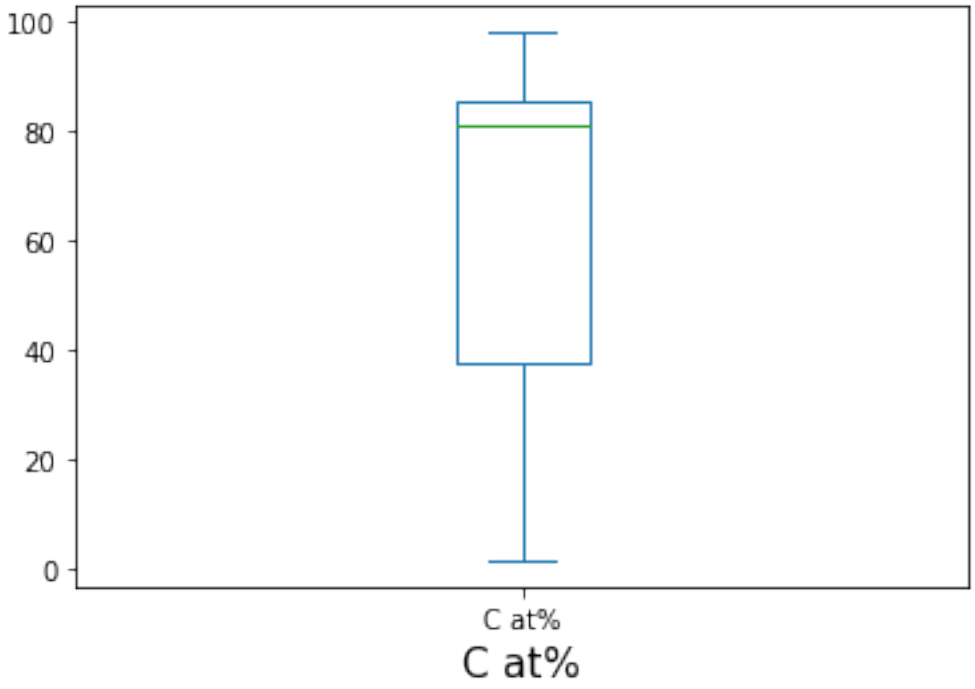




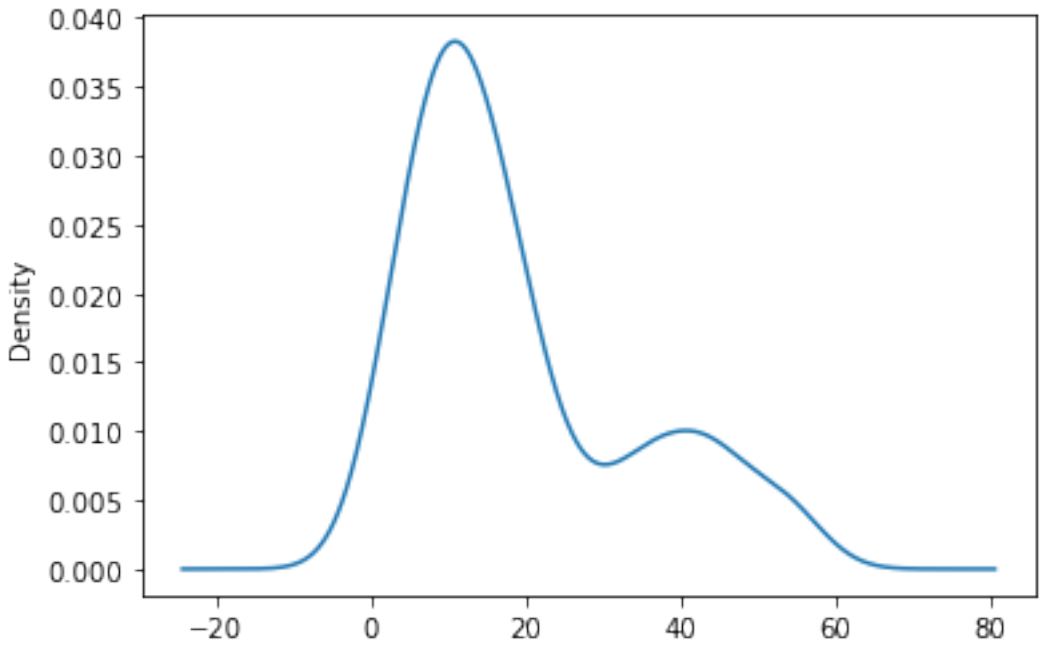
std= 4.59 mean= 2.45
count of available values= 223 count of missing values= 117 count of unique values= 29

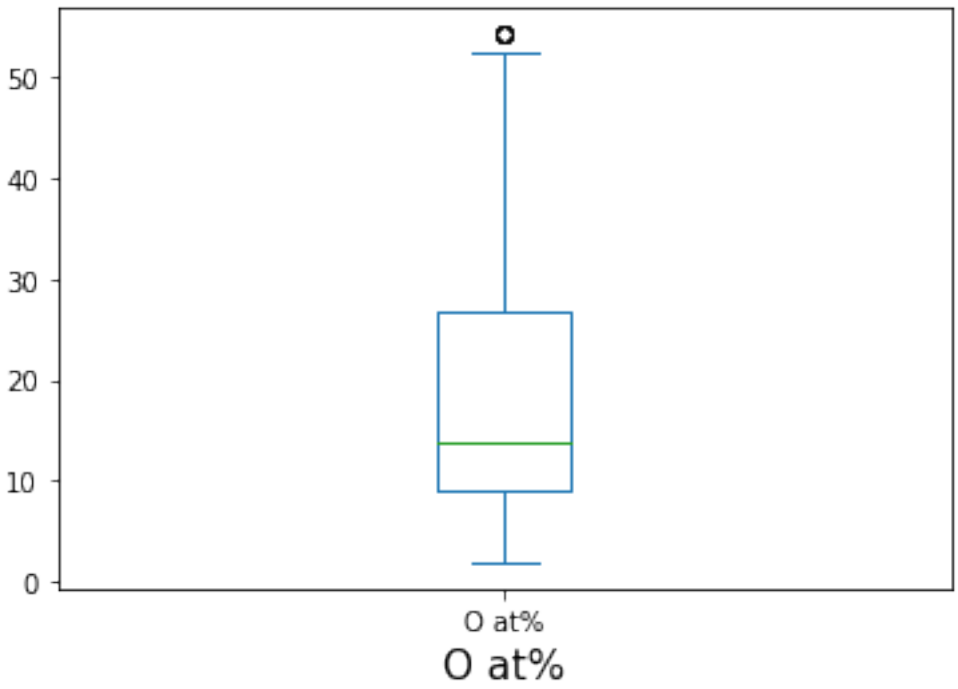
```
available values/total data = 0.656   unique values/available values = 0.130
################################################################################
```





```
std= 28.53   mean= 66.65
count of available values= 222   count of missing values= 118   count of unique
values= 67
available values/total data = 0.653   unique values/available values = 0.302
################################################################################
```






```
std= 14.53   mean= 19.09
count of available values= 216   count of missing values= 124   count of unique
values= 67
available values/total data = 0.635   unique values/available values = 0.310
################################################################################
```

[107]:
```
fillingMissingValues(df7)
```

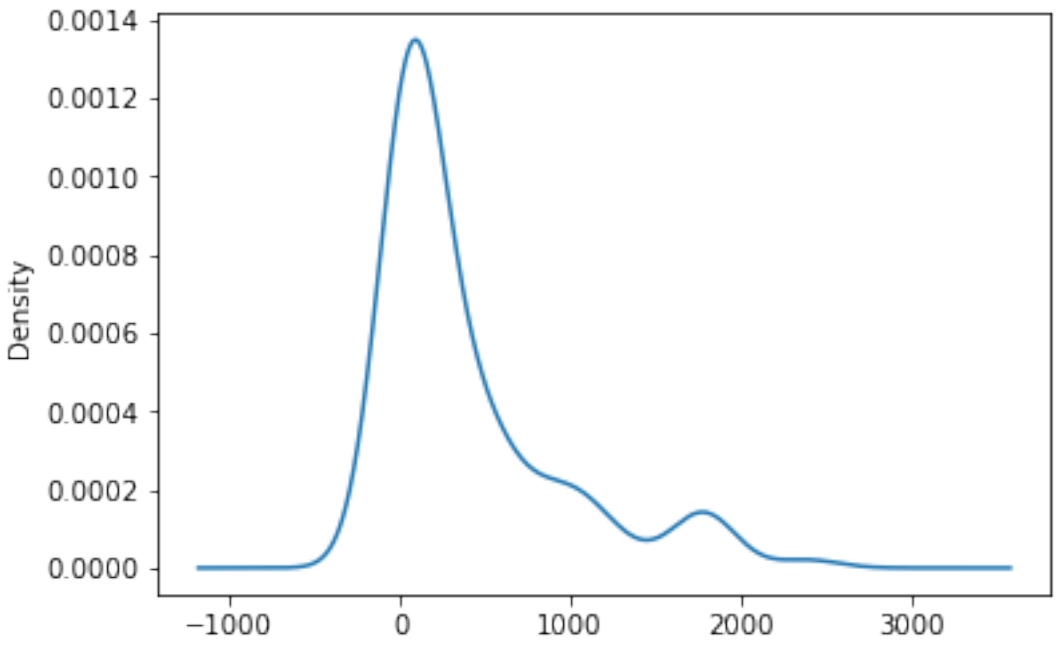
```
column=  Ratio of ID/IG   available ratio= 0.585   unique ratio= 0.261
mean= 1.055   std= 0.417
 missing values filled with 0.8466

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


column=  N at%   available ratio= 0.656   unique ratio= 0.130
mean= 2.450   std= 4.593
 missing values filled with 3.3688

################################################################################
```

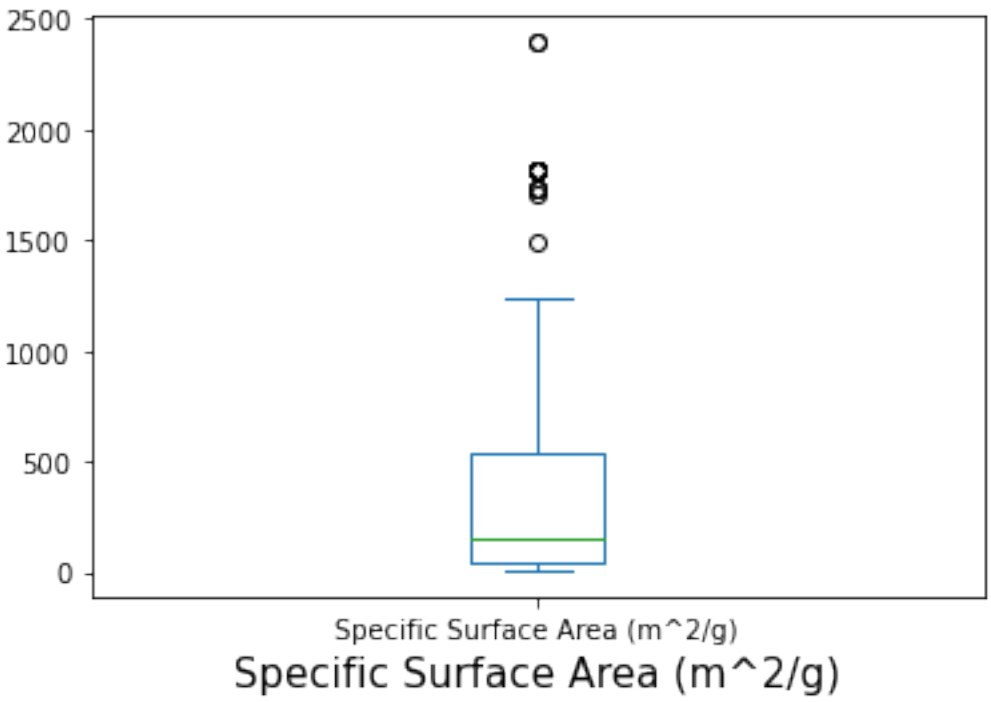
```
column=  C at%   available ratio= 0.653   unique ratio= 0.302
mean= 66.650   std= 28.527
 missing values filled with 60.9449

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



column=  O at%   available ratio= 0.635   unique ratio= 0.310
mean= 19.090   std= 14.533
 missing values filled with 16.1830

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



/tmp/ipykernel_270313/1691008062.py:66: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[column].fillna((mean-(std/2)), inplace= True)
/tmp/ipykernel_270313/1691008062.py:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[column].fillna((mean+(std/5)), inplace= True)
/tmp/ipykernel_270313/1691008062.py:34: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[column].fillna((mean-(std/5)), inplace= True)
/tmp/ipykernel_270313/1691008062.py:34: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[column].fillna((mean-(std/5)), inplace= True)
```

[108]:
```
df7.isnull().sum()
```

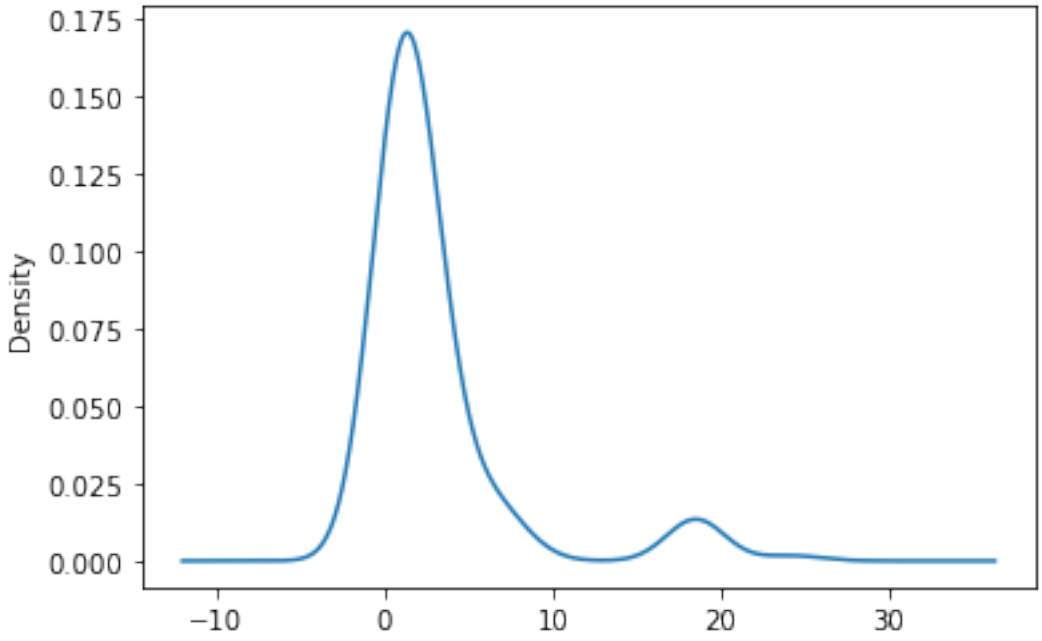
[108]:
```
Limits of Potential Window (V)                      0
Lower Limit of Potential Window (V)                 0
Upper Limit of Potential Window (V)                 0
Potential Window (V)                                0
Current Density (A/g)                               0
Capacitance (F/g)                                   0
Specific Surface Area (m^2/g)                     126
Charge Transfer Resistance (Rct) (ohm)            203
Equivalent Series Resistance (Rs) (ohm)           189
Electrode Material                                  0
Pore Size (nm)                                    195
Pore Volume (cm^3/g)                              182
Ratio of ID/IG                                      0
N at%                                               0
C at%                                               0
O at%                                               0
Electrolyte Chemical Formula                        0
Electrolyte Ion Mobility Ranking                    0
Electrolyte Concentration (M)                       0
Cell Configuration (three/two electrode system)     0
dtype: int64
```

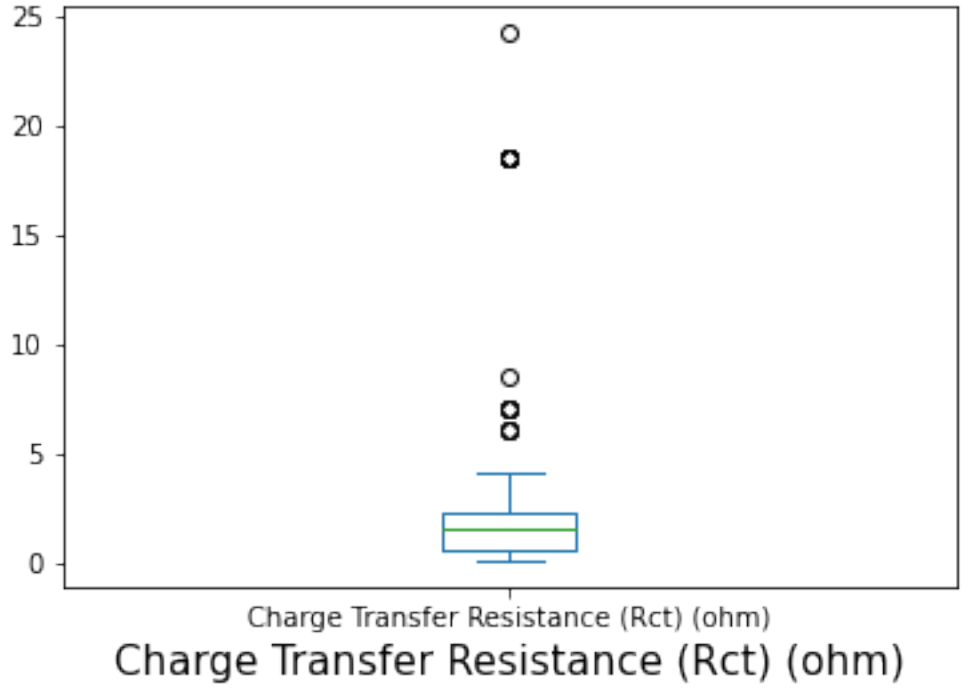
[109]:
```
NullColumnsPlot(df7)
```





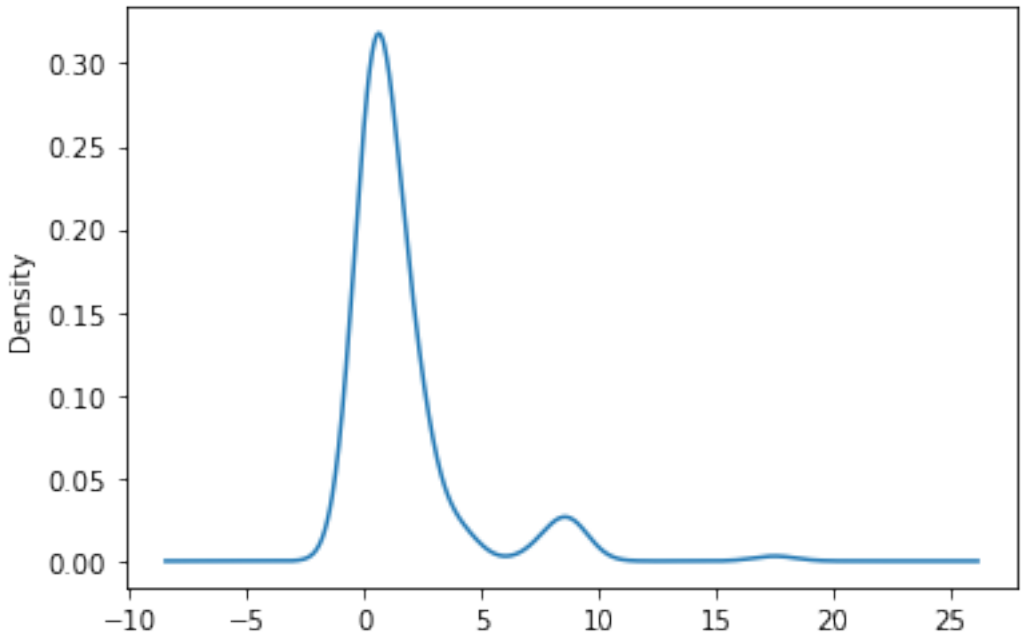

```
std= 533.76   mean= 397.19
count of available values= 214   count of missing values= 126   count of unique
values= 68
available values/total data = 0.629   unique values/available values = 0.318
################################################################################
```

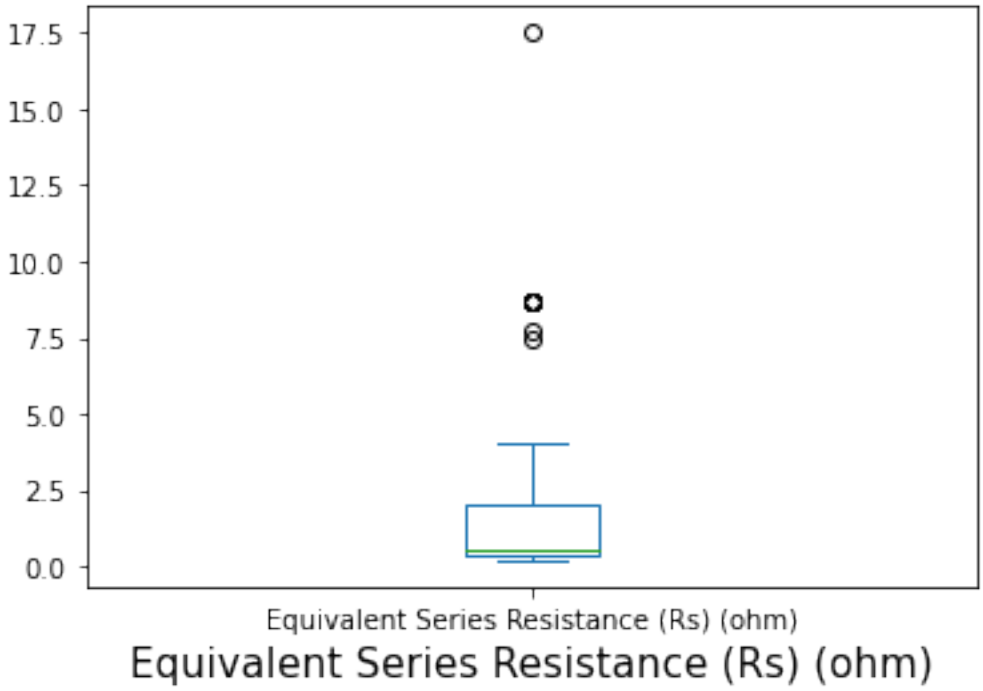




std= 4.63 mean= 3.03
count of available values= 137 count of missing values= 203 count of unique values= 38

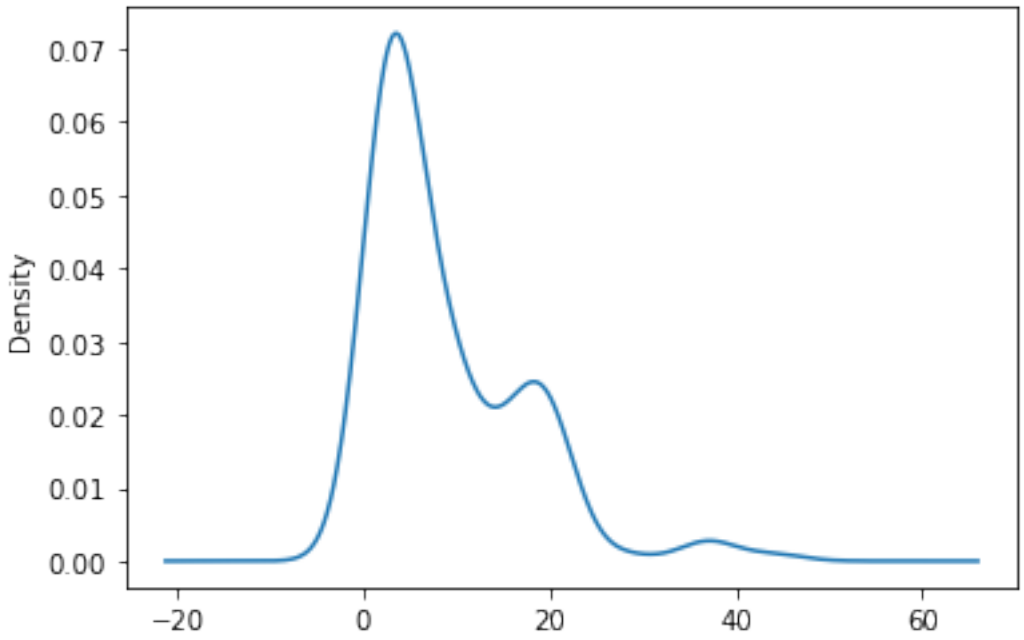
```
available values/total data = 0.403   unique values/available values = 0.277
################################################################################
```

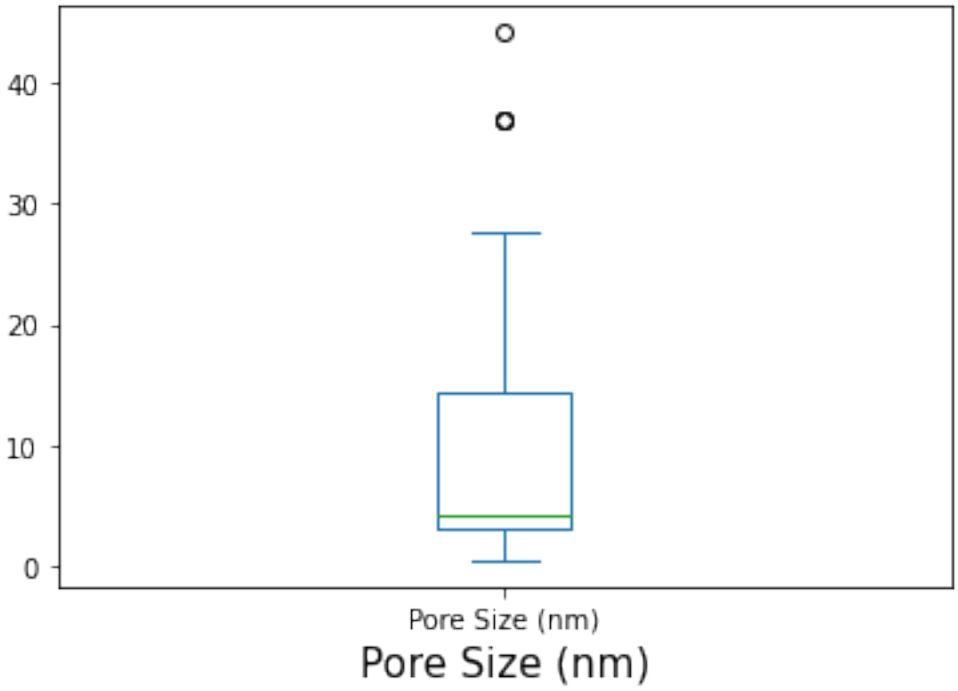




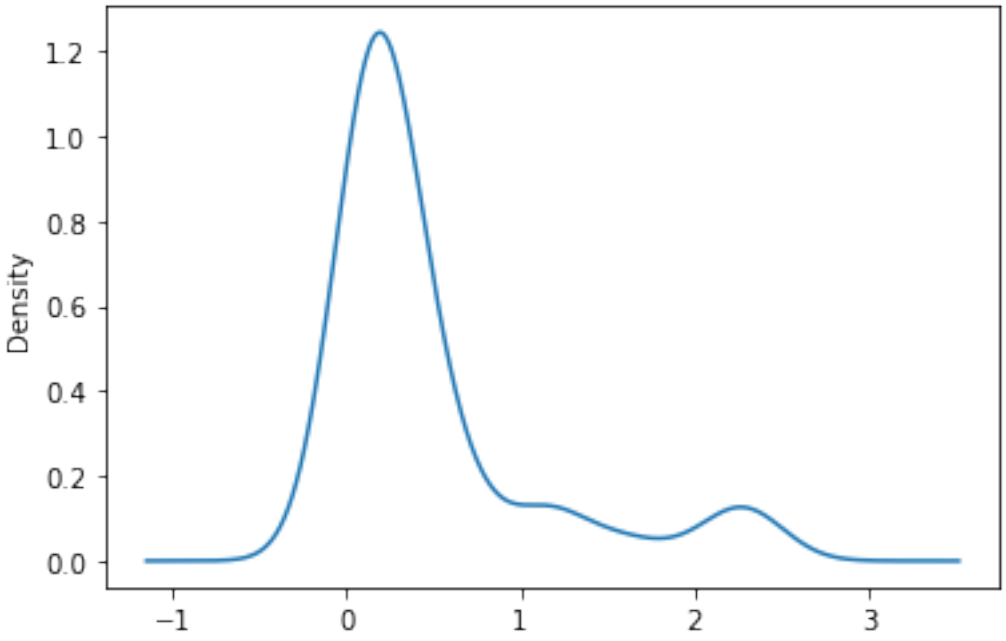
Equivalent Series Resistance (Rs) (ohm)

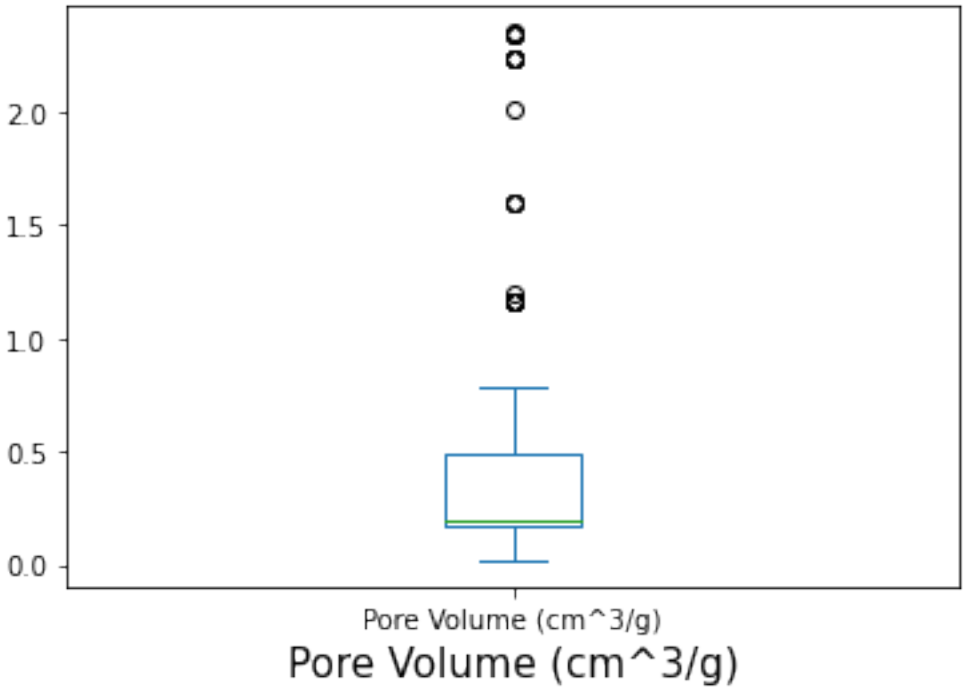
```
std= 2.45   mean= 1.60
count of available values= 151   count of missing values= 189   count of unique
values= 35
available values/total data = 0.444   unique values/available values = 0.232
################################################################################
```






```
std= 8.27   mean= 8.84
count of available values= 145   count of missing values= 195   count of unique
values= 41
available values/total data = 0.426   unique values/available values = 0.283
################################################################################
```





Pore Volume (cm^3/g)

std= 0.61 mean= 0.48
count of available values= 158 count of missing values= 182 count of unique values= 41

```
available values/total data = 0.465   unique values/available values = 0.259
################################################################################
```

[110]:
```
EmptyColumnNames(df7)
```

[110]:
```
['Specific Surface Area (m^2/g)',
 'Charge Transfer Resistance (Rct) (ohm)',
 'Equivalent Series Resistance (Rs) (ohm)',
 'Pore Size (nm)',
 'Pore Volume (cm^3/g)']
```

[111]:
```
df7.shape
```

[111]:
```
(340, 20)
```

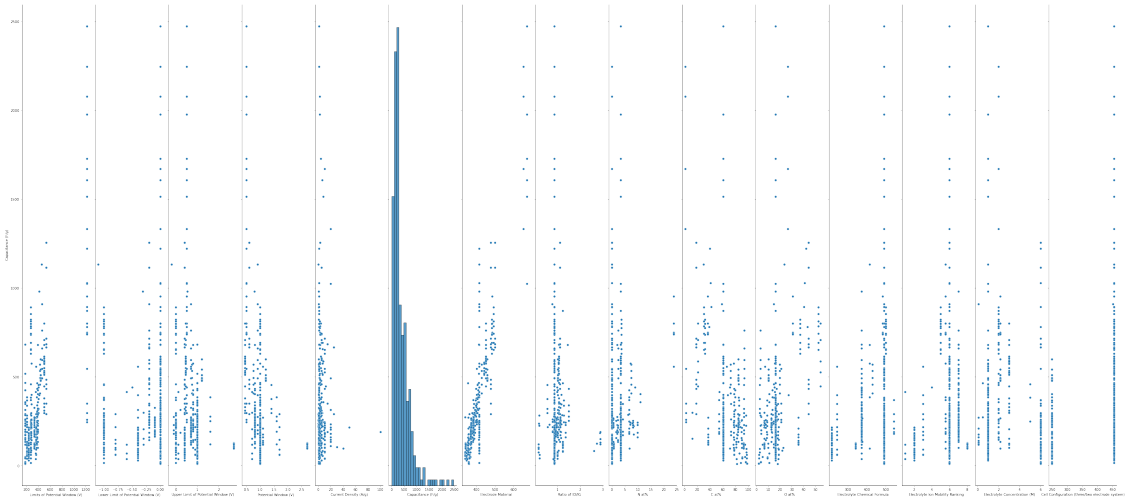
[112]:
```
df=df7.drop(columns=EmptyColumnNames(df7),axis=1)
```

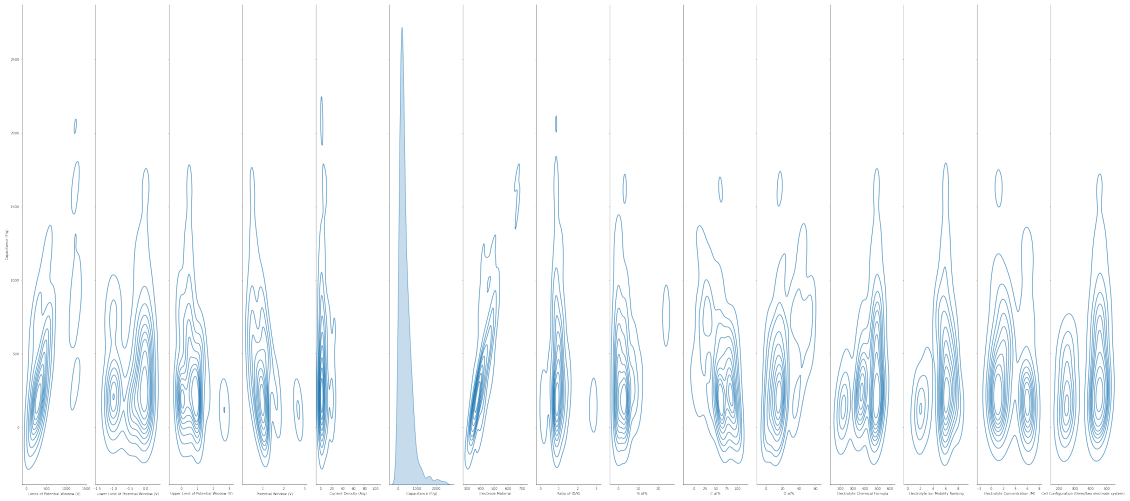
[113]:
```
df.shape
```

[113]:
```
(340, 15)
```

[114]:
```
sns.pairplot(df,y_vars='Capacitance (F/g)',aspect=3/20,height=20)

sns.pairplot(df,kind='kde',y_vars='Capacitance (F/g)',aspect=3/20,height=20)
```

[114]:
```
<seaborn.axisgrid.PairGrid at 0x7f7a023c3f40>
```





[115]:
```
df.shape
```

[115]:
```
(340, 15)
```

[116]:
```
df.columns
```

[116]:
```
Index(['Limits of Potential Window (V)', 'Lower Limit of Potential Window (V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)', 'Capacitance (F/g)', 'Electrode Material',
       'Ratio of ID/IG', 'N at%', 'C at%', 'O at%',
       'Electrolyte Chemical Formula', 'Electrolyte Ion Mobility Ranking',
       'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)'],
      dtype='object')
```

[117]:
```
df.isnull().sum()
```

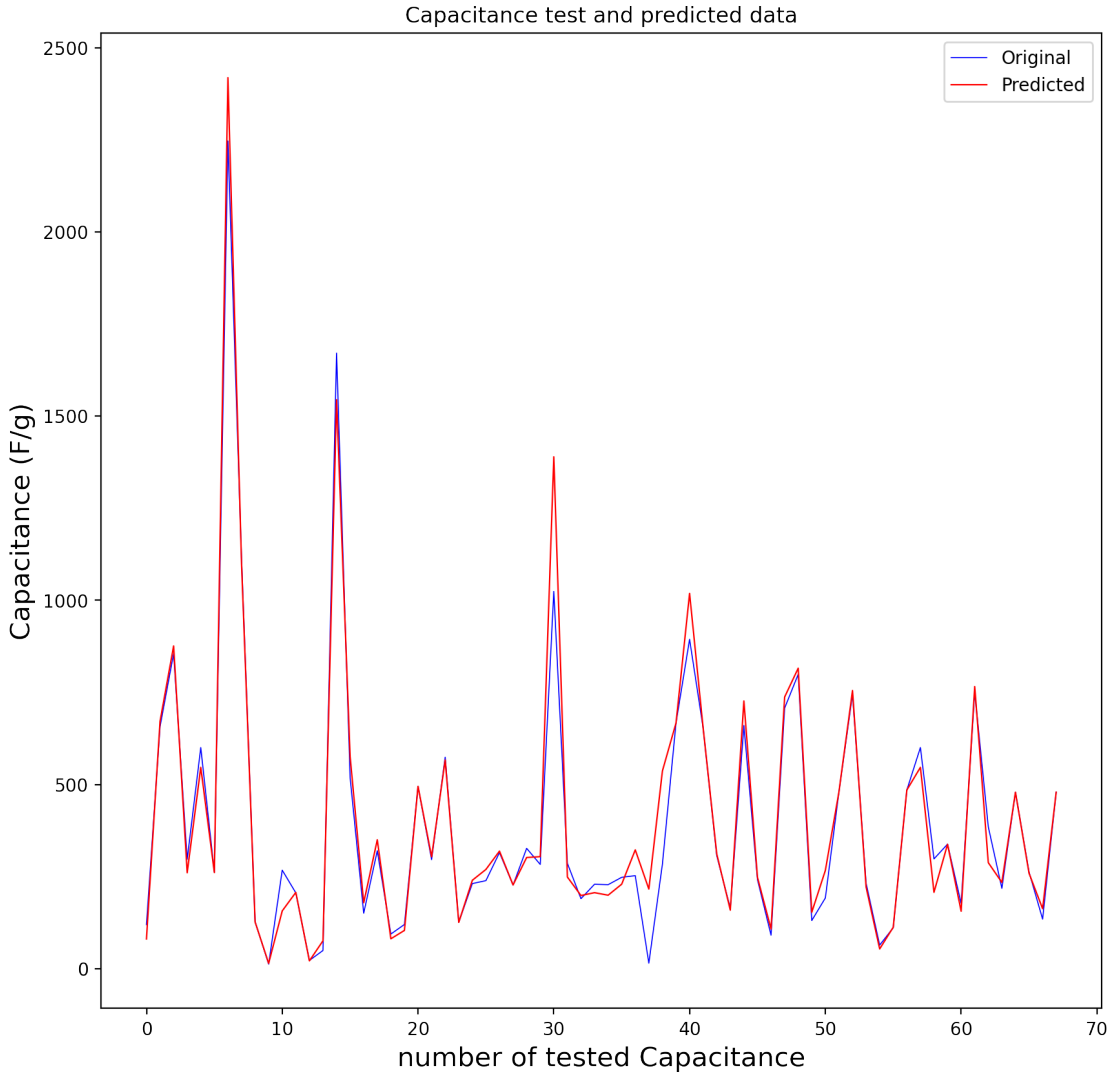
[117]:
```
Limits of Potential Window (V)                  0
Lower Limit of Potential Window (V)             0
Upper Limit of Potential Window (V)             0
Potential Window (V)                            0
Current Density (A/g)                           0
Capacitance (F/g)                               0
Electrode Material                              0
Ratio of ID/IG                                  0
N at%                                           0
C at%                                           0
O at%                                           0
Electrolyte Chemical Formula                    0
Electrolyte Ion Mobility Ranking                0
```

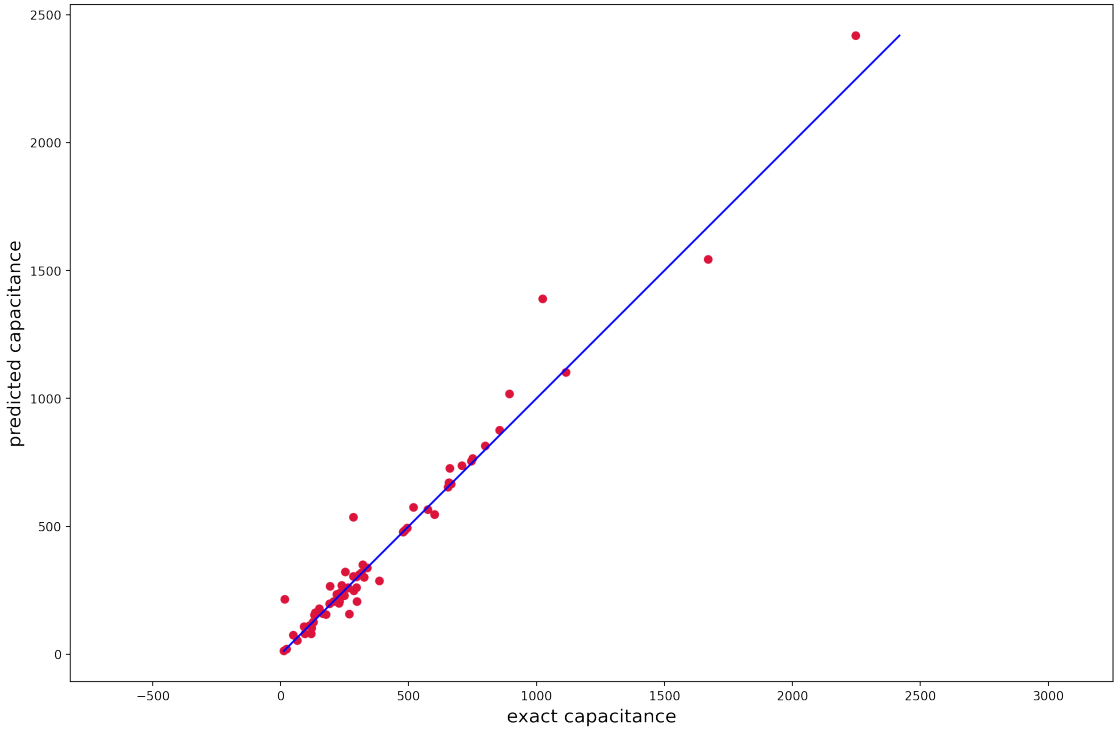
```
Electrolyte Concentration (M)                        0
Cell Configuration (three/two electrode system)      0
dtype: int64
```

[118]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window␣
 ↪(V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)', 'Electrode Material',
       'Ratio of ID/IG', 'N at%', 'C at%', 'O at%',
       'Electrolyte Chemical Formula', 'Electrolyte Ion Mobility Ranking',
       'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']



X=df[Predictors].values
y=df[TargetVariable]



from sklearn.preprocessing import StandardScaler

PredictorScaler=StandardScaler()


PredictorScalerFit=PredictorScaler.fit(X)


X=PredictorScalerFit.transform(X)
y=y.values.ravel()


from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error , r2_score









opted=ExtraTreesRegressionScore()
```

[119]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],␣
↪random_state=opted[2])
etr=ExtraTreesRegressor(n_jobs=-1)
etr.fit(X_train,y_train)


Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)


mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)


TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions


from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()




plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)

print('\n')
```

```
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].head(60))
print(2*'\n')
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
print('number of tested data= %i'%count)
print('test_size"percent"= %.2f'%(opted[0]*100))
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
print('optimized MSE: %.4f ' %mse)
print('optimized RMSE: %.4f ' %mse**(0.5))
print('optimized random state: %i'%opted[2] )
```





	Capacitance (F/g)	PredictedCapacitance (F/g)
0	119.60	80.6192
1	657.00	671.4330
2	855.00	875.9700
3	296.70	260.3430
4	600.19	546.4836
5	262.00	260.8910
6	2246.63	2418.4900
7	1115.00	1101.7220
8	127.40	126.8420
9	12.40	14.4400
10	267.39	156.8218
11	205.80	207.0770
12	22.40	21.2480
13	49.40	75.6871
14	1670.83	1544.3240
15	518.00	575.4088
16	150.70	179.0790
17	320.00	349.8200
18	94.00	81.1540
19	120.00	103.8910
20	494.84	494.8400

21	296.00	303.6600
22	574.00	566.3200
23	126.00	125.8220
24	231.00	239.9780
25	239.00	269.3700
26	315.00	319.1270
27	227.00	227.2609
28	326.60	301.7970
29	283.00	304.0120
30	1024.00	1389.3377
31	286.00	248.7629
32	190.00	198.6000
33	229.20	206.3300
34	228.00	199.4500
35	248.00	229.3220
36	252.50	322.5000
37	15.00	216.0840
38	284.00	536.7940
39	666.60	666.7140
40	894.00	1018.5800
41	653.00	652.7420
42	306.87	311.7136
43	163.80	158.9050
44	660.00	726.6460
45	241.80	248.0230
46	90.90	107.7720
47	707.20	737.2790
48	799.00	815.6200
49	130.70	153.1680
50	192.00	266.7500
51	479.83	479.8300
52	745.00	755.0080
53	232.00	221.3800
54	64.00	53.5810
55	111.10	112.5120
56	484.80	484.8000
57	600.19	546.4836
58	298.00	207.3310
59	338.00	337.5444

```
total number of data= 340
number of trained data= 273
number of tested data= 67
test_size"percent"= 20.00
score for xtest and ytest : 0.9609
cross validation score: 0.8936
```

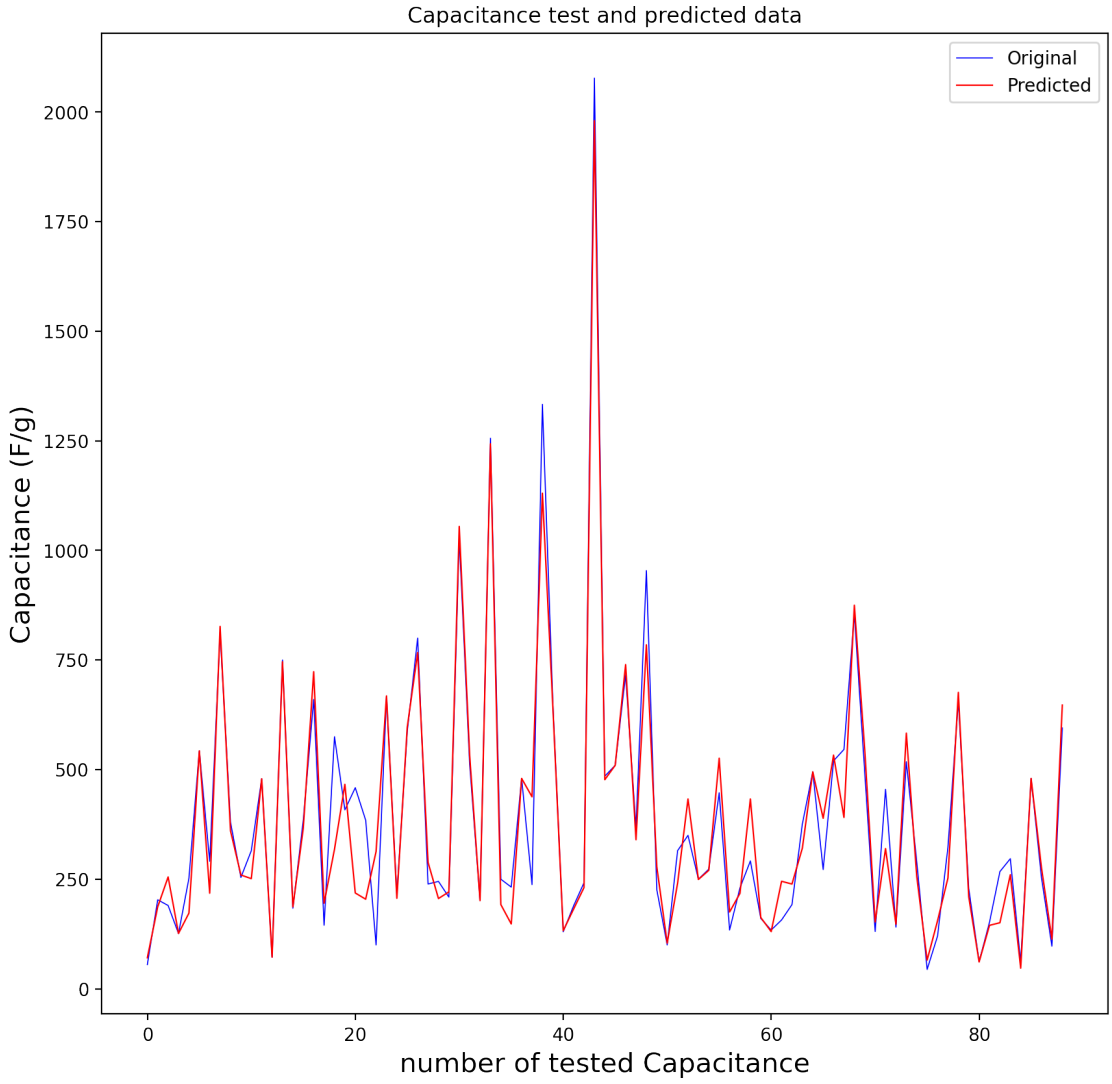
```
optimized MSE: 5422.8287
optimized RMSE: 73.6399
optimized random state: 7
```

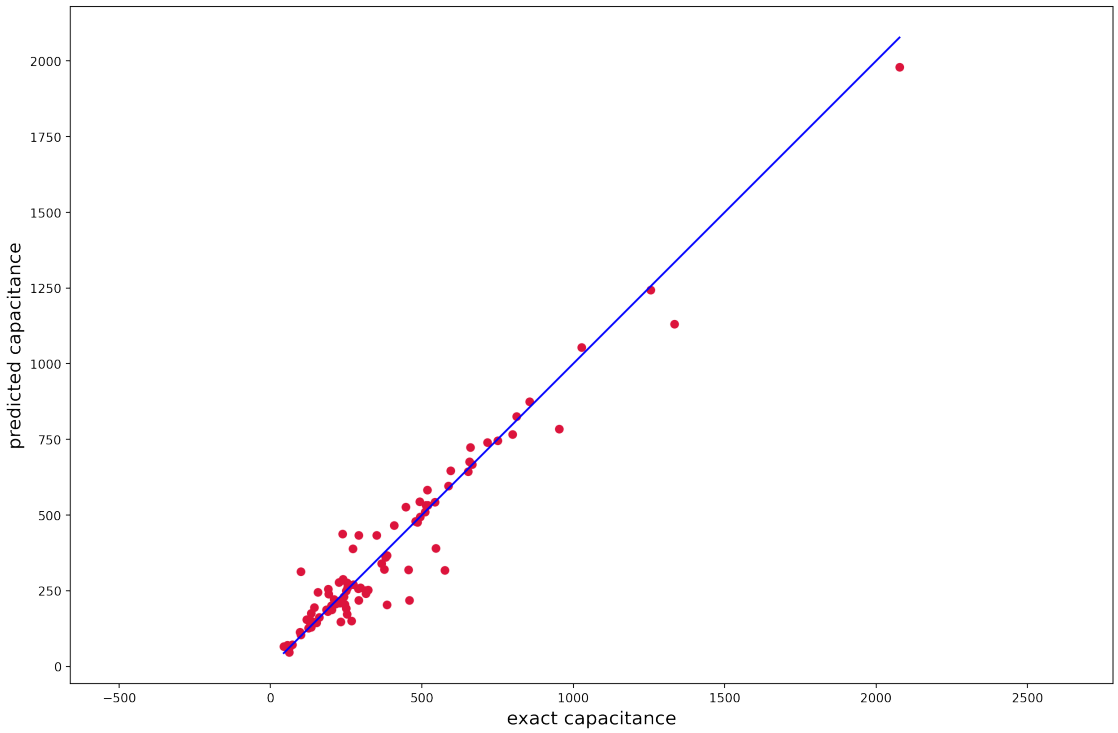
[120]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window␣
 ↪(V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)', 'Electrode Material',
       'Ratio of ID/IG', 'N at%', 'C at%', 'O at%',
       'Electrolyte Chemical Formula', 'Electrolyte Ion Mobility Ranking',
       'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']


X=df[Predictors].values
y=df[TargetVariable]


from sklearn.preprocessing import StandardScaler

PredictorScaler=StandardScaler()
PredictorScalerFit=PredictorScaler.fit(X)


X=PredictorScalerFit.transform(X)
y=y.values.ravel()


from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error , r2_score









opted=ExtraTreesRegressionMse()
```

[121]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],␣
 ↪random_state=opted[2])
etr=ExtraTreesRegressor(n_jobs=-1)
etr.fit(X_train,y_train)
```

```
Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)


mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)


TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions


from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()




plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)

print('\n')
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].head(60))
print(2*'\n')
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
print('number of tested data= %i'%count)
```

```
print('test_size"percent"= %.2f'%(opted[0]*100))
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
print('optimized MSE: %.4f ' %mse)
print('optimized RMSE: %.4f ' %mse**(0.5))
print('optimized random state: %i'%opted[2] )
```





	Capacitance (F/g)	PredictedCapacitance (F/g)
0	55.10	70.8260
1	203.00	187.0179
2	190.00	255.0780
3	126.00	126.2930
4	252.00	172.5600
5	542.73	542.7300
6	290.90	218.1000
7	813.00	826.6400
8	380.00	360.1916
9	254.00	259.6040
10	315.00	251.2244
11	479.00	479.0000
12	72.00	72.2720
13	750.00	745.9200
14	183.80	186.9820
15	385.00	366.5422
16	660.00	723.3620
17	145.00	195.3084
18	575.00	318.2241
19	408.20	466.4108
20	458.70	218.5485

21	384.60	204.3661
22	100.00	313.2048
23	666.60	668.0720
24	215.80	206.4360
25	587.30	596.9940
26	800.00	766.9500
27	239.00	288.3700
28	245.00	205.8620
29	209.20	220.9960
30	1028.00	1054.7200
31	513.00	533.0000
32	201.50	201.2720
33	1256.00	1243.3100
34	250.00	192.1419
35	232.00	147.9080
36	479.83	479.8544
37	237.60	437.9110
38	1333.33	1130.5711
39	653.00	644.2200
40	130.10	132.5320
41	189.40	181.8900
42	242.00	230.9320
43	2077.50	1980.3009
44	485.00	476.9878
45	509.85	509.8500
46	716.60	739.5560
47	367.00	340.0440
48	954.00	784.6400
49	226.00	278.4900
50	100.00	104.1600
51	315.00	240.8130
52	350.00	433.0000
53	250.00	249.3640
54	272.90	270.2030
55	447.13	526.0000
56	134.00	175.1142
57	230.30	217.7840
58	291.60	433.0000
59	160.60	162.6280

```
total number of data= 340
number of trained data= 252
number of tested data= 88
test_size"percent"= 26.00
score for xtest and ytest : 0.9409
cross validation score: 0.8722
```

```
optimized MSE: 5824.2680
optimized RMSE: 76.3169
optimized random state: 4
```

[122]:
```
df.shape
```

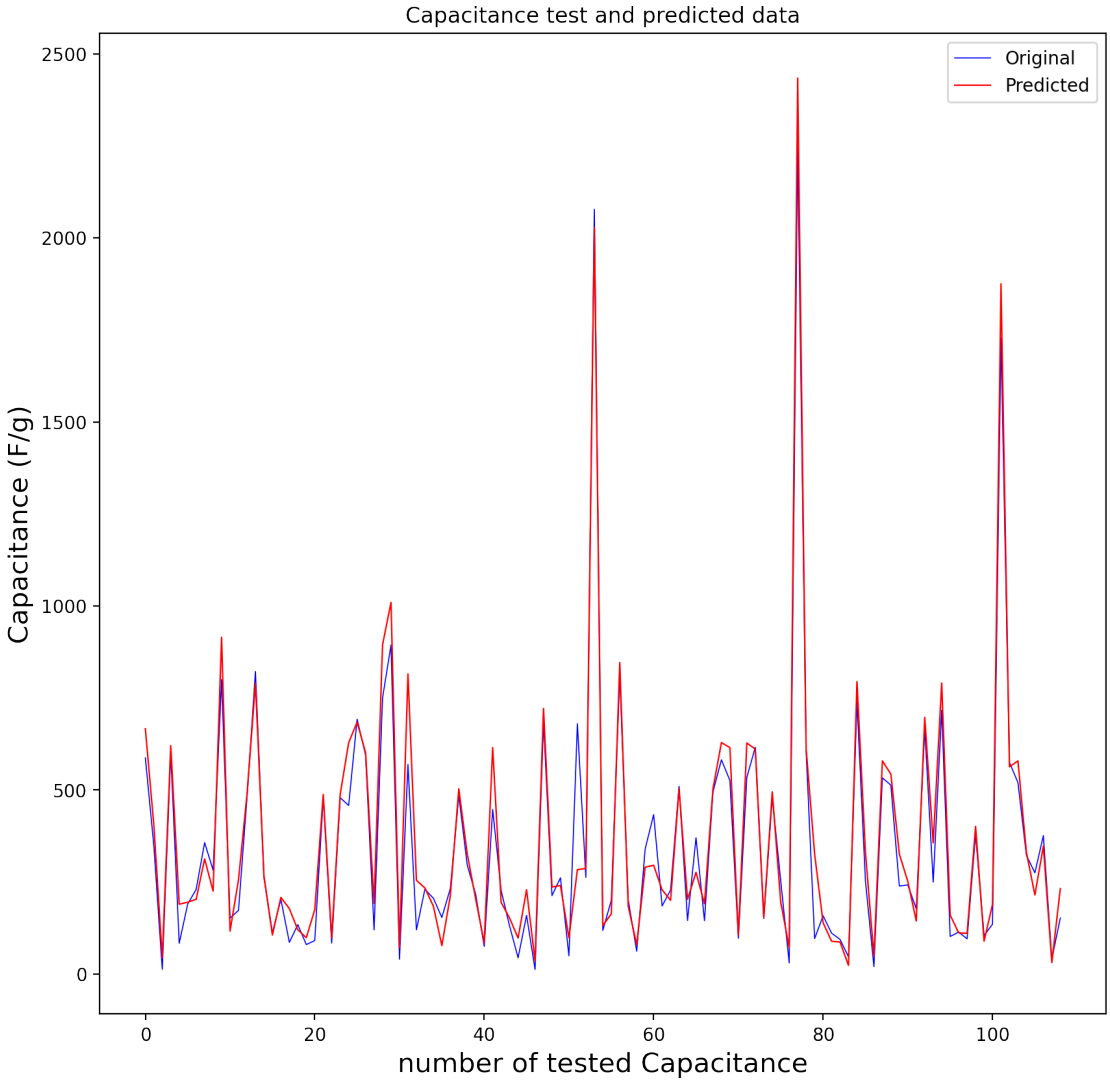
[122]:
```
(340, 15)
```

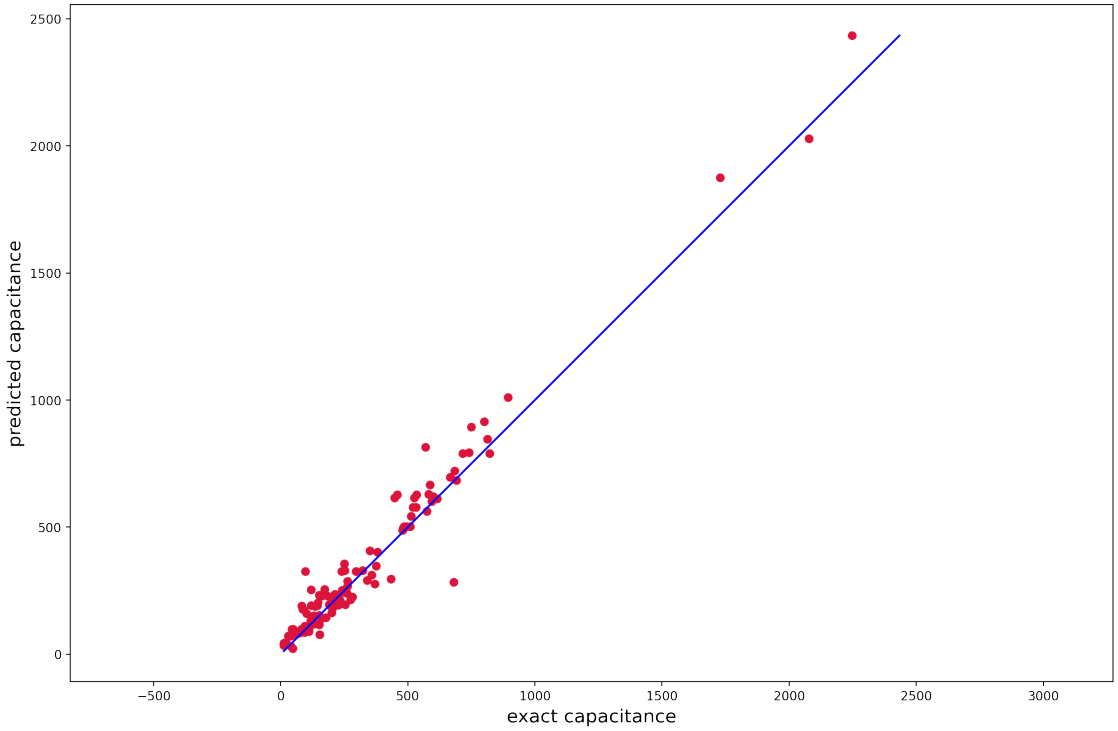
the following model is GBR on the latter df; df6

[123]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window␣
 ↪(V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)', 'Electrode Material',
       'Ratio of ID/IG', 'N at%', 'C at%', 'O at%',
       'Electrolyte Chemical Formula', 'Electrolyte Ion Mobility Ranking',
       'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']


X=df[Predictors].values
y=df[TargetVariable]


from sklearn.preprocessing import StandardScaler


PredictorScaler=StandardScaler()
PredictorScalerFit=PredictorScaler.fit(X)



X=PredictorScalerFit.transform(X)
y=y.values.ravel()



from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error , r2_score


opted=GradientBoostRegScore()
```

[124]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],␣
↪random_state=opted[2])
etr=GradientBoostingRegressor(random_state=opted[2])
etr.fit(X_train,y_train)


Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)


mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)

TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions


from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()




plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)
print('\n')
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].head(60))
print(2*'\n')
```

```
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
print('number of tested data= %i'%count)
print('test_size"percent"= %.2f'%(opted[0]*100))
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
print('optimized MSE: %.4f ' %mse)
print('optimized RMSE: %.4f ' %mse**(0.5))
print('optimized random state: %.4f'%opted[2] )
```





	Capacitance (F/g)	PredictedCapacitance (F/g)
0	587.30	666.455820
1	350.00	406.395388
2	12.70	44.075665
3	600.19	620.475716
4	83.40	189.581698
5	190.00	195.178041
6	230.00	203.054508
7	357.00	312.756266
8	282.00	225.218210
9	800.00	914.805723
10	152.00	116.703666
11	173.00	256.253886
12	478.93	487.860058
13	822.00	790.147084
14	262.00	267.279745
15	112.00	106.008636
16	203.00	208.110441
17	86.00	177.737792
18	134.00	120.118945
19	80.00	98.610214
20	90.90	175.115983

21	479.00	487.860058
22	84.00	98.610214
23	479.00	487.860058
24	458.00	628.500815
25	692.00	684.205852
26	595.00	601.416488
27	120.00	192.065618
28	750.00	894.447895
29	894.00	1009.718755
30	40.00	71.734669
31	569.40	815.385086
32	120.00	254.380304
33	230.30	233.865590
34	205.80	184.290423
35	153.70	77.313260
36	232.00	212.290742
37	484.80	503.126887
38	296.00	325.971373
39	216.00	203.164017
40	74.90	84.697010
41	447.13	615.181633
42	227.00	194.642028
43	130.10	151.786777
44	44.10	97.760018
45	159.40	228.969235
46	12.40	35.201616
47	685.00	721.486646
48	213.00	236.292702
49	261.40	240.398218
50	49.40	98.657369
51	680.00	283.773998
52	262.50	286.716394
53	2077.50	2028.041284
54	118.50	131.542038
55	200.00	163.309873
56	813.00	846.131997
57	201.50	184.290423
58	62.00	79.133091
59	339.88	290.047257

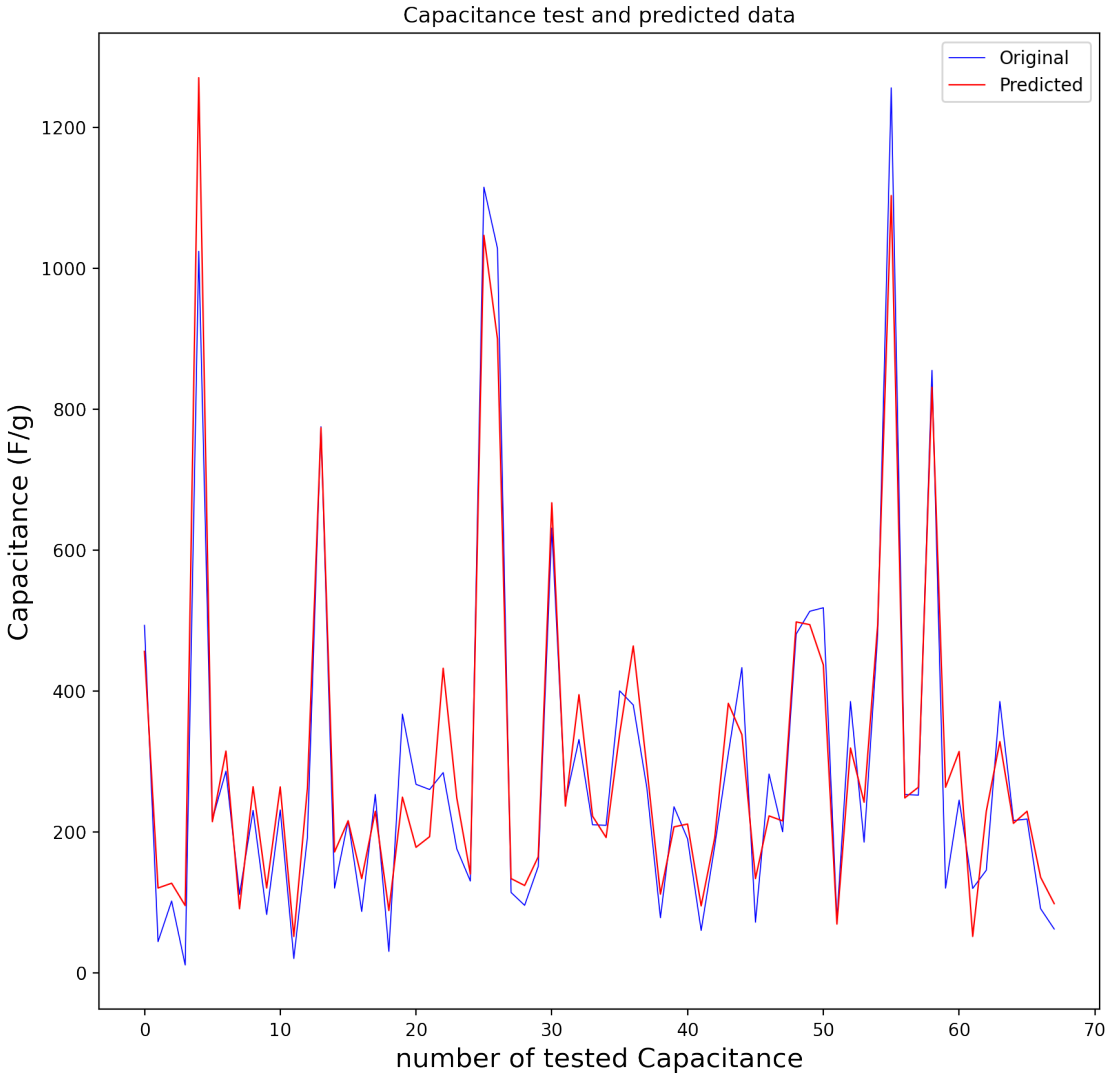
```
total number of data= 340
number of trained data= 232
number of tested data= 108
test_size"percent"= 32.00
score for xtest and ytest : 0.9528
cross validation score: 0.8412
```

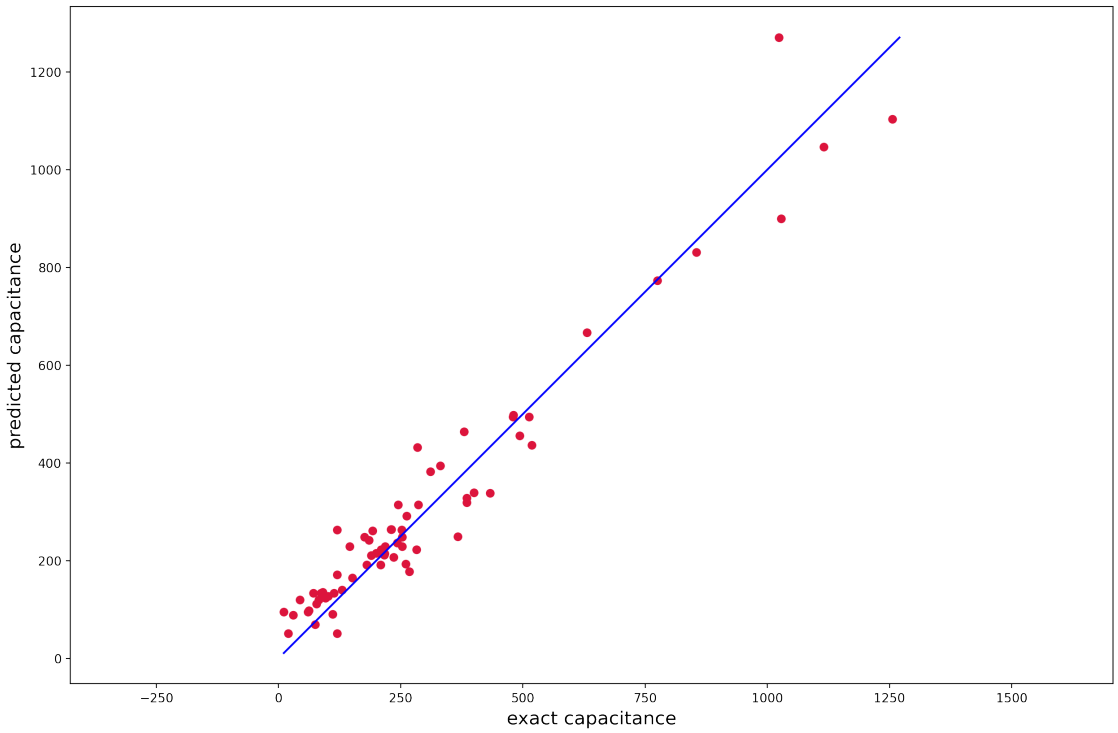
```
optimized MSE: 6095.1335
optimized RMSE: 78.0713
optimized random state: 20.0000
```

[125]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window␣
 ↪(V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)', 'Electrode Material',
       'Ratio of ID/IG', 'N at%', 'C at%', 'O at%',
       'Electrolyte Chemical Formula', 'Electrolyte Ion Mobility Ranking',
       'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']


X=df[Predictors].values
y=df[TargetVariable]


from sklearn.preprocessing import StandardScaler


PredictorScaler=StandardScaler()
PredictorScalerFit=PredictorScaler.fit(X)


X=PredictorScalerFit.transform(X)
y=y.values.ravel()

from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error , r2_score


opted=GradientBoostRegMse()
```

[126]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],␣
 ↪random_state=opted[2])
etr=GradientBoostingRegressor(random_state=opted[2])
etr.fit(X_train,y_train)


Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)
```

```
mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)

TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions


from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()




plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)
print('\n')
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].head(60))
print(2*'\n')
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
print('number of tested data= %i'%count)
print('test_size"percent"= %.2f'%(opted[0]*100))
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
print('optimized MSE: %.4f ' %mse)
print('optimized RMSE: %.4f ' %mse**(0.5))
print('optimized random state: %.4f'%opted[2] )
```


Capacitance test and predicted data
Original
Predicted
Capacitance (F/g)
number of tested Capacitance
0
200
400
600
800
1000
1200
0
10
20
30
40
50
60
70



	Capacitance (F/g)	PredictedCapacitance (F/g)
0	493.00	456.047333
1	44.10	120.216693
2	101.70	126.978751
3	10.90	95.239204
4	1024.00	1270.355549
5	217.20	214.357449
6	286.00	314.554749
7	111.10	90.785378
8	230.30	263.870008
9	82.60	120.216693
10	231.00	263.667834
11	20.10	51.453984
12	192.00	261.437592
13	775.00	773.499622
14	120.00	171.341152
15	215.80	215.644921
16	86.90	133.591341
17	253.00	229.118570
18	30.20	88.271503
19	367.00	249.100233
20	267.39	177.981241

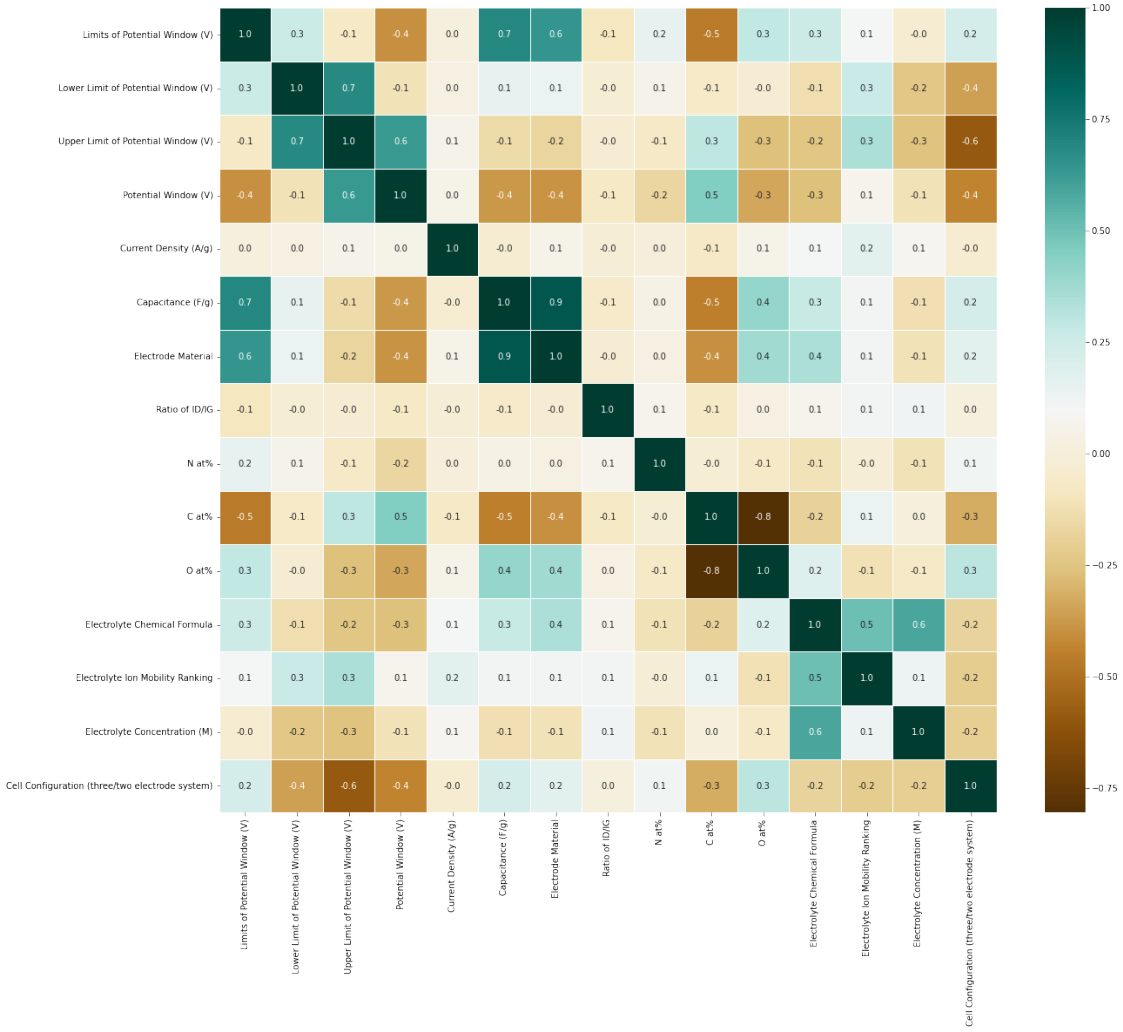
21	260.00	193.077456
22	284.00	432.105584
23	176.00	248.259619
24	130.10	140.125534
25	1115.00	1046.586065
26	1028.00	899.465017
27	113.90	133.591341
28	95.60	123.734778
29	150.70	164.472399
30	631.00	666.992399
31	243.00	236.484716
32	331.00	394.568249
33	210.00	222.228372
34	209.20	191.884433
35	400.00	339.191217
36	380.00	463.741770
37	262.50	291.484712
38	78.00	111.469647
39	235.50	206.963775
40	190.00	210.993919
41	60.00	94.871004
42	180.00	191.471983
43	311.00	382.321619
44	433.05	337.975218
45	71.60	133.591341
46	282.00	222.537446
47	200.00	215.488774
48	480.78	497.859384
49	513.00	494.017560
50	518.00	436.878623
51	75.20	69.043478
52	385.00	318.885555
53	185.30	242.015330
54	479.83	493.833507
55	1256.00	1103.038190
56	252.90	248.098844
57	252.00	263.250648
58	855.00	831.018934
59	120.00	263.250648

```
total number of data= 340
number of trained data= 273
number of tested data= 67
test_size"percent"= 20.00
score for xtest and ytest : 0.9369
cross validation score: 0.8825
```

```
optimized MSE: 4365.1552
optimized RMSE: 66.0693
optimized random state: 10.0000
```

[127]:
```
plt.figure(figsize=(20,17))
heatmap(df.corr(),annot=True,fmt='.1f',linewidths=0.5,cmap='BrBG')
```

[127]: <AxesSubplot:>



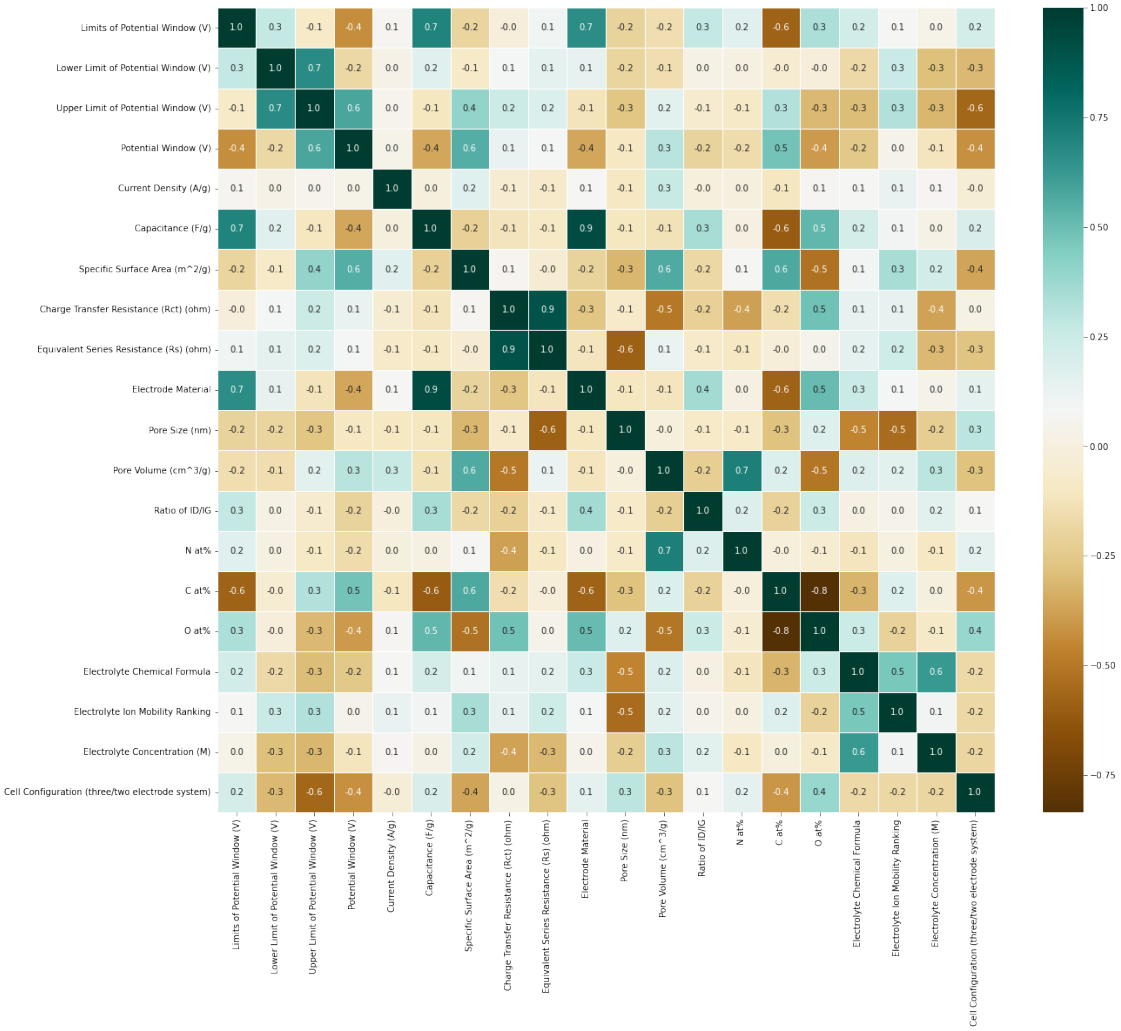
[128]:
```
df8=dframe[8]
```

[129]:
```
df8.shape
```

[129]: (414, 20)

[130]:
```
plt.figure(figsize=(20,17))
heatmap(df8.corr(),annot=True,fmt='.1f',linewidths=0.5,cmap='BrBG')
```

[130]: <AxesSubplot:>



[131]:
```
df8.isnull().sum()
```

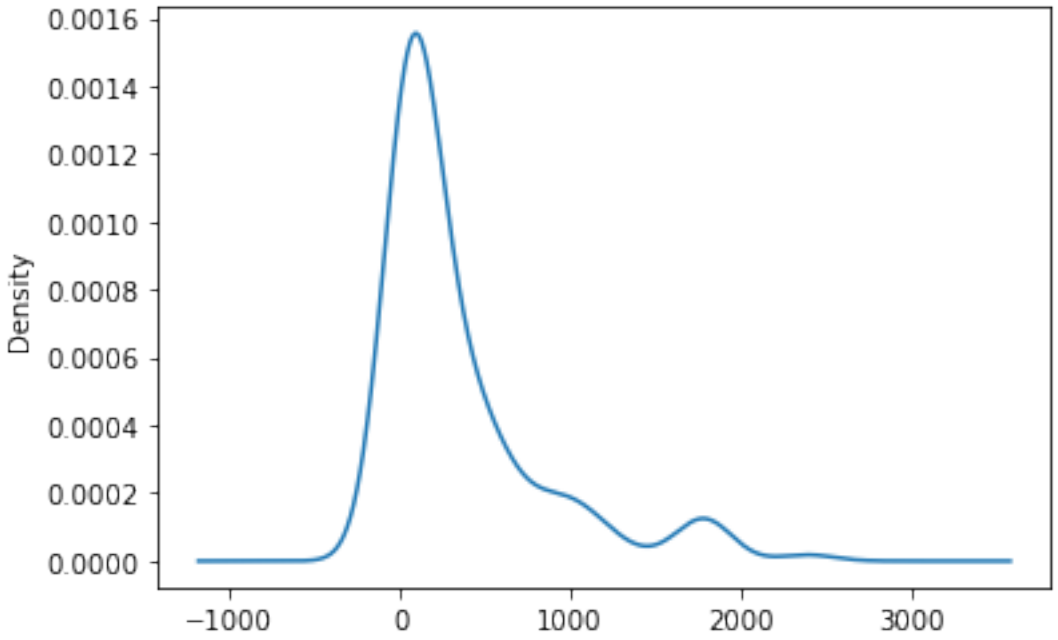
[131]:
```
Limits of Potential Window (V)                  0
Lower Limit of Potential Window (V)             0
Upper Limit of Potential Window (V)             0
Potential Window (V)                            0
Current Density (A/g)                           0
Capacitance (F/g)                               0
Specific Surface Area (m^2/g)                 138
Charge Transfer Resistance (Rct) (ohm)        277
```

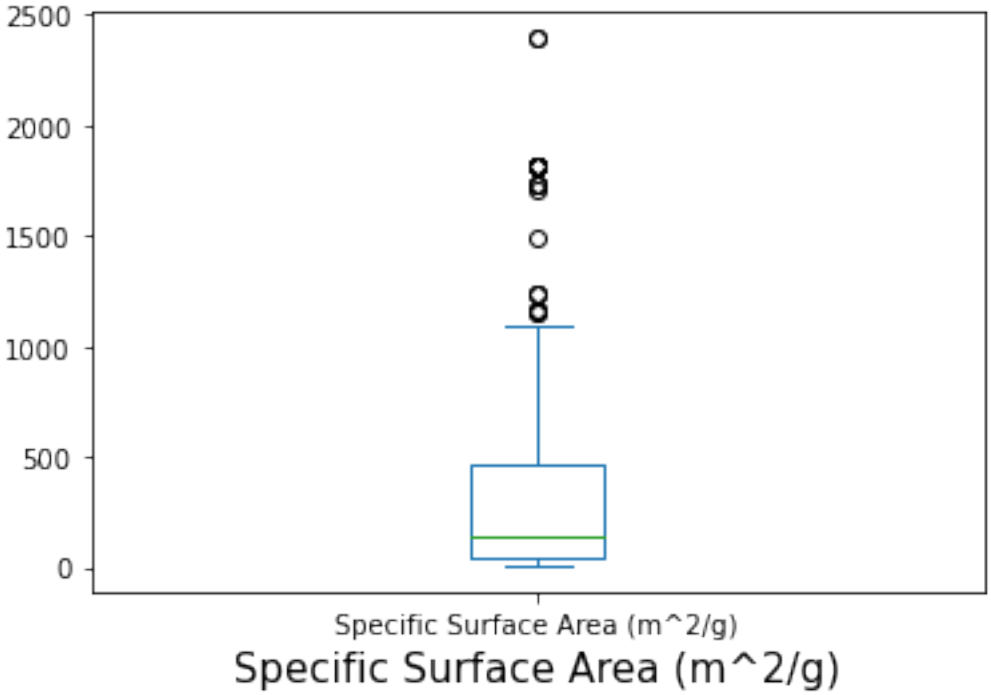
```
Equivalent Series Resistance (Rs) (ohm)            263
Electrode Material                                   0
Pore Size (nm)                                     261
Pore Volume (cm^3/g)                               219
Ratio of ID/IG                                     182
N at%                                              183
C at%                                              192
O at%                                              198
Electrolyte Chemical Formula                         0
Electrolyte Ion Mobility Ranking                     0
Electrolyte Concentration (M)                        0
Cell Configuration (three/two electrode system)      0
dtype: int64
```

[132]:
```
EmptyColumnNames(df8)
```

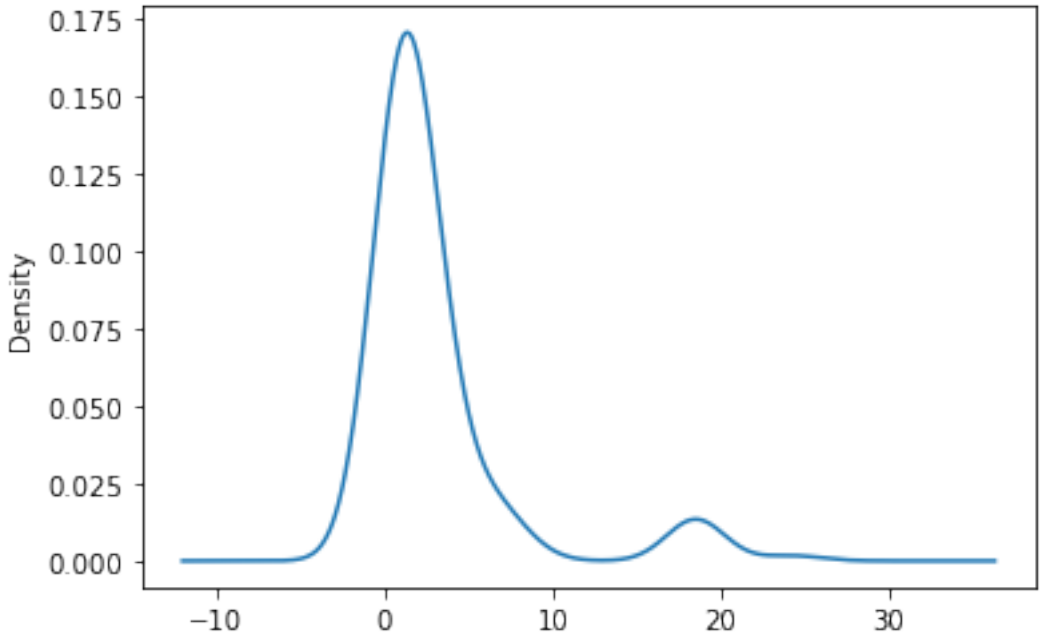
[132]:
```
['Specific Surface Area (m^2/g)',
 'Charge Transfer Resistance (Rct) (ohm)',
 'Equivalent Series Resistance (Rs) (ohm)',
 'Pore Size (nm)',
 'Pore Volume (cm^3/g)',
 'Ratio of ID/IG',
 'N at%',
 'C at%',
 'O at%']
```

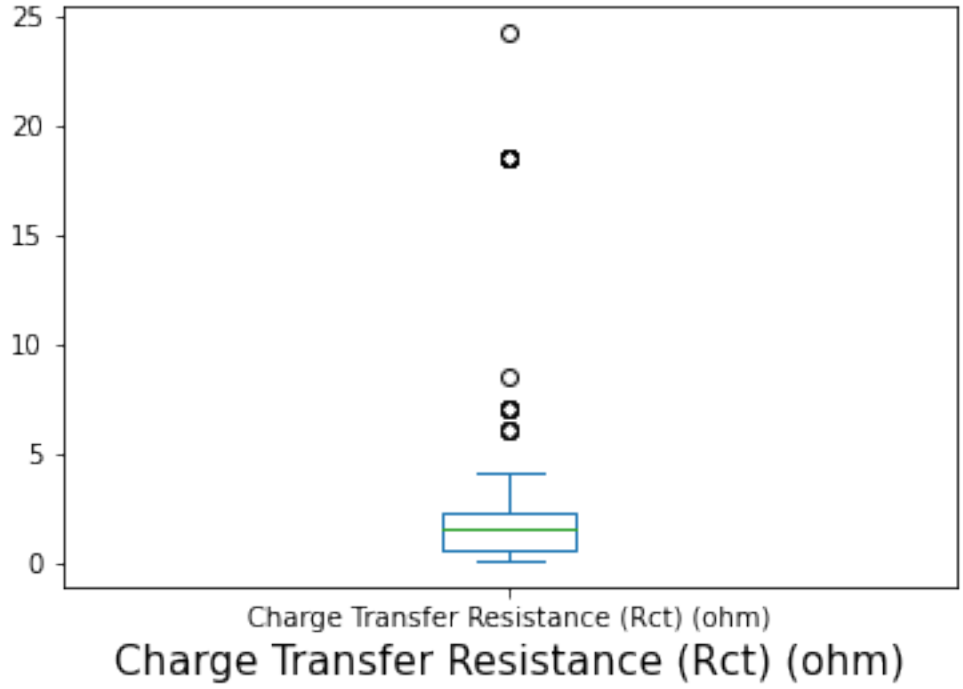
[133]:
```
NullColumnsPlot(df8)
```





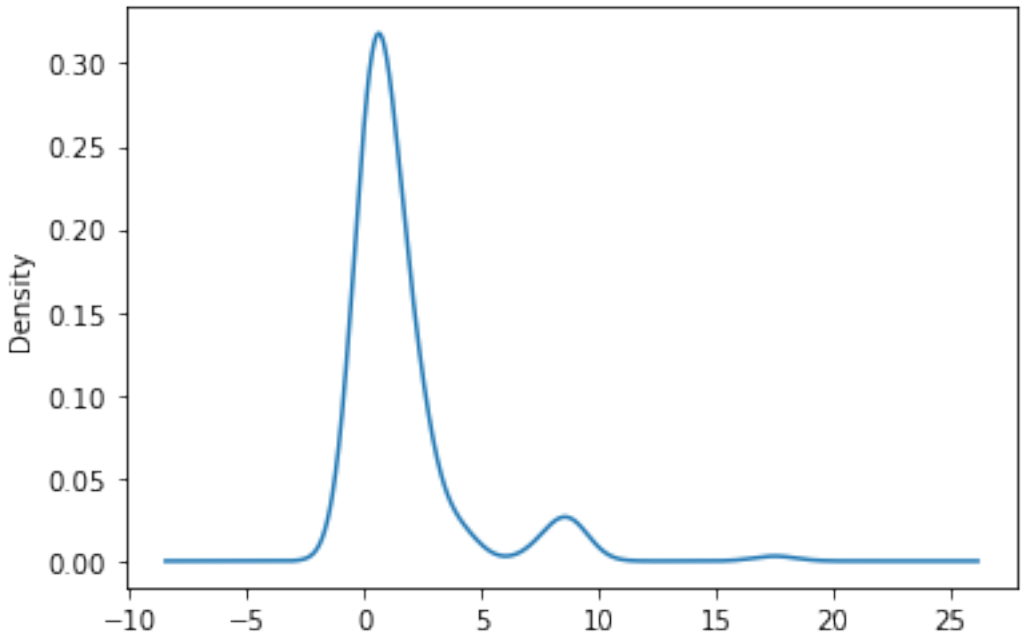

```
std= 485.87   mean= 357.69
count of available values= 276   count of missing values= 138   count of unique
values= 94
available values/total data = 0.667   unique values/available values = 0.341
################################################################################
```

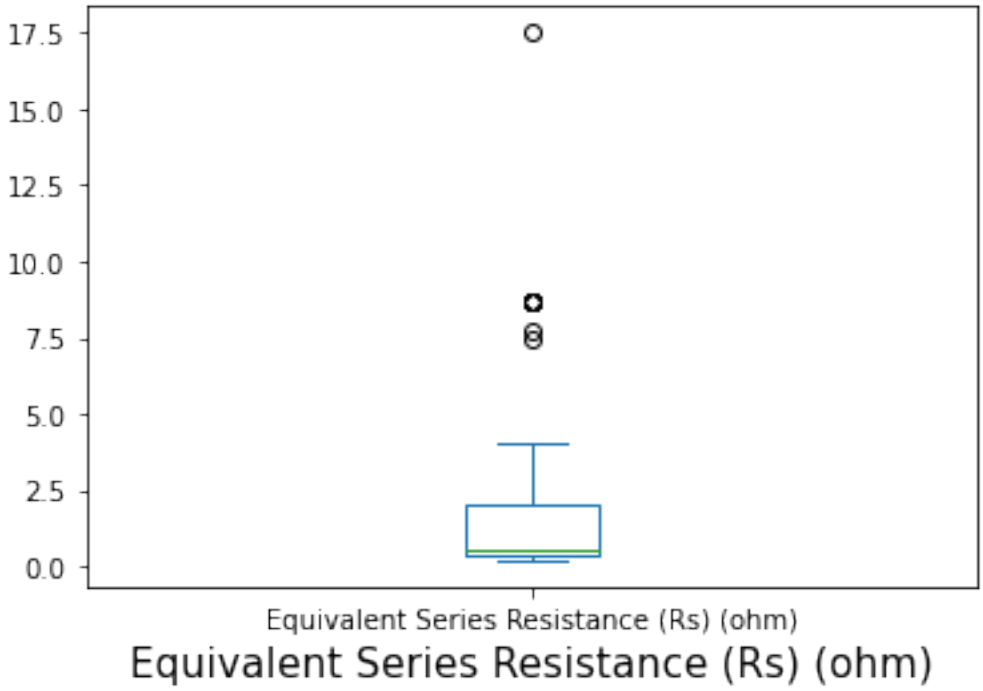





```
std= 4.63   mean= 3.03
count of available values= 137   count of missing values= 277   count of unique
values= 38
```

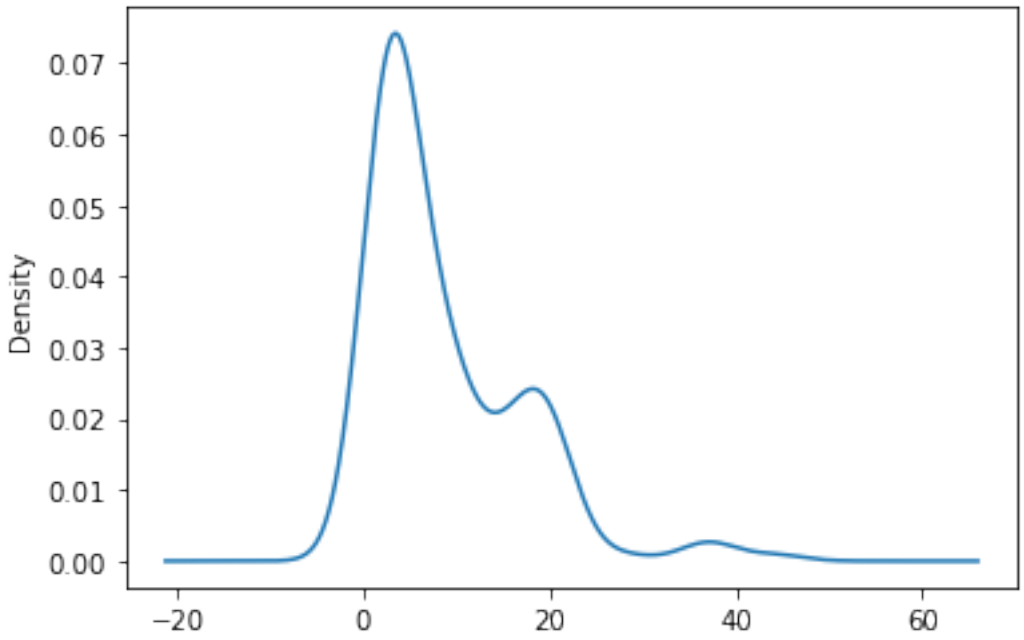
```
available values/total data = 0.331   unique values/available values = 0.277
################################################################################
```

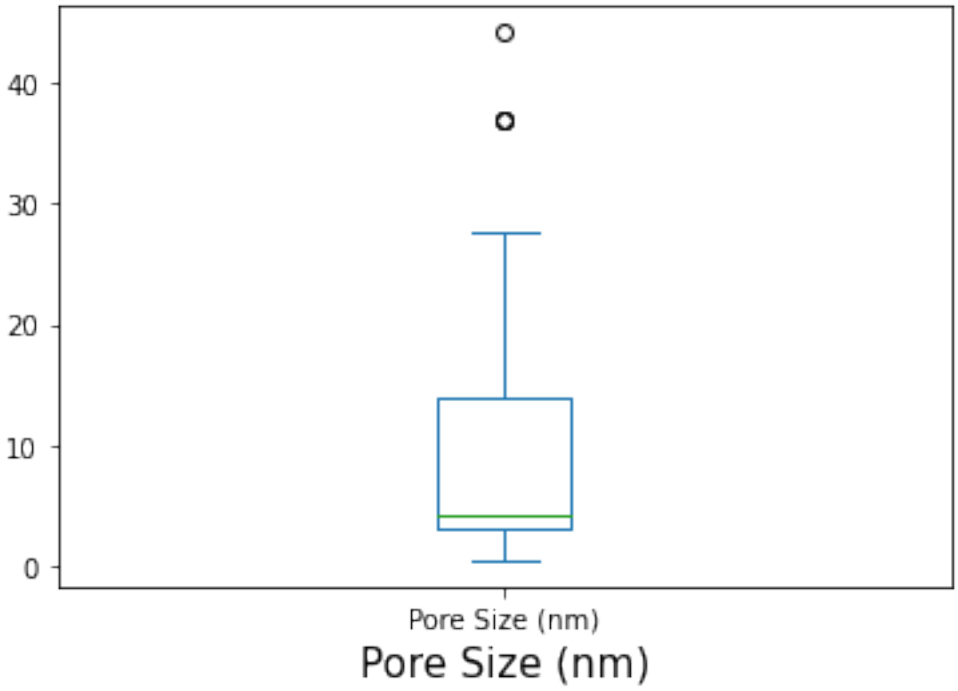




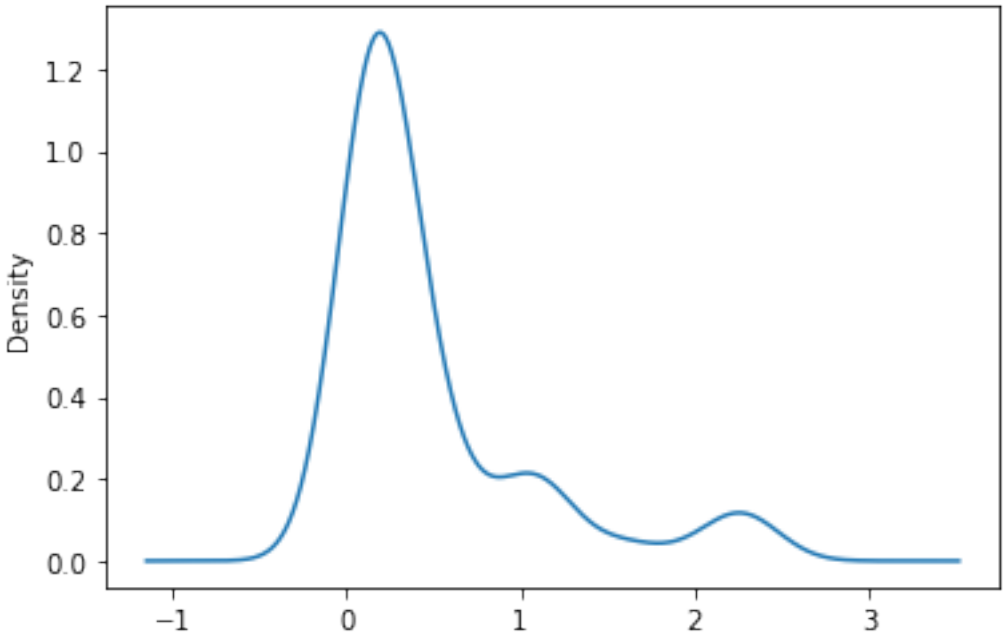
Equivalent Series Resistance (Rs) (ohm)

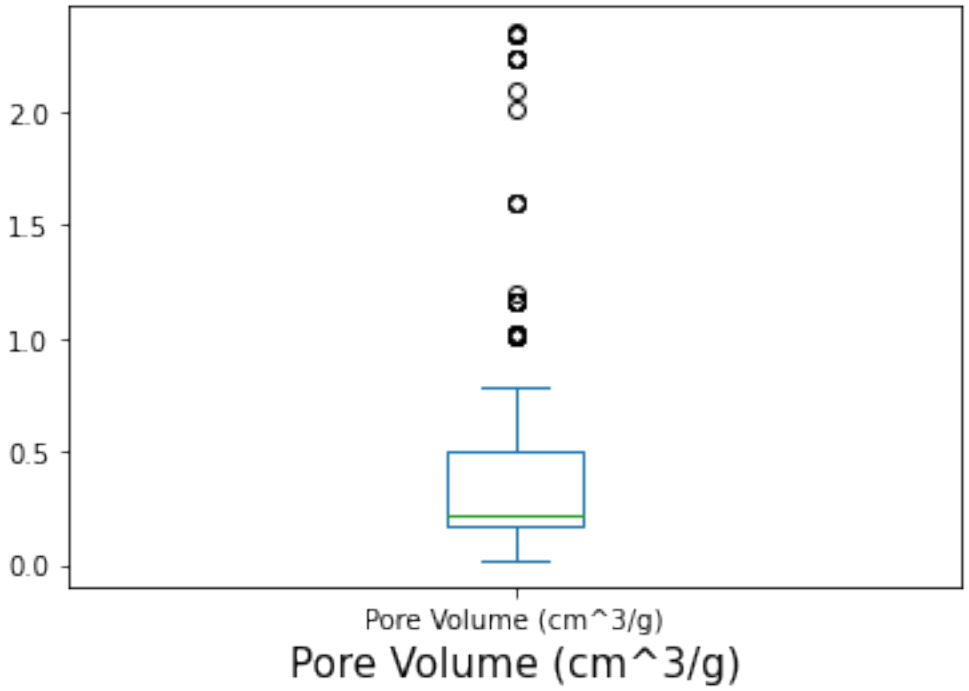
```
std= 2.45   mean= 1.60
count of available values= 151   count of missing values= 263   count of unique
values= 35
available values/total data = 0.365   unique values/available values = 0.232
################################################################################
```






```
std= 8.15   mean= 8.68
count of available values= 153   count of missing values= 261   count of unique
values= 45
available values/total data = 0.370   unique values/available values = 0.294
################################################################################
```

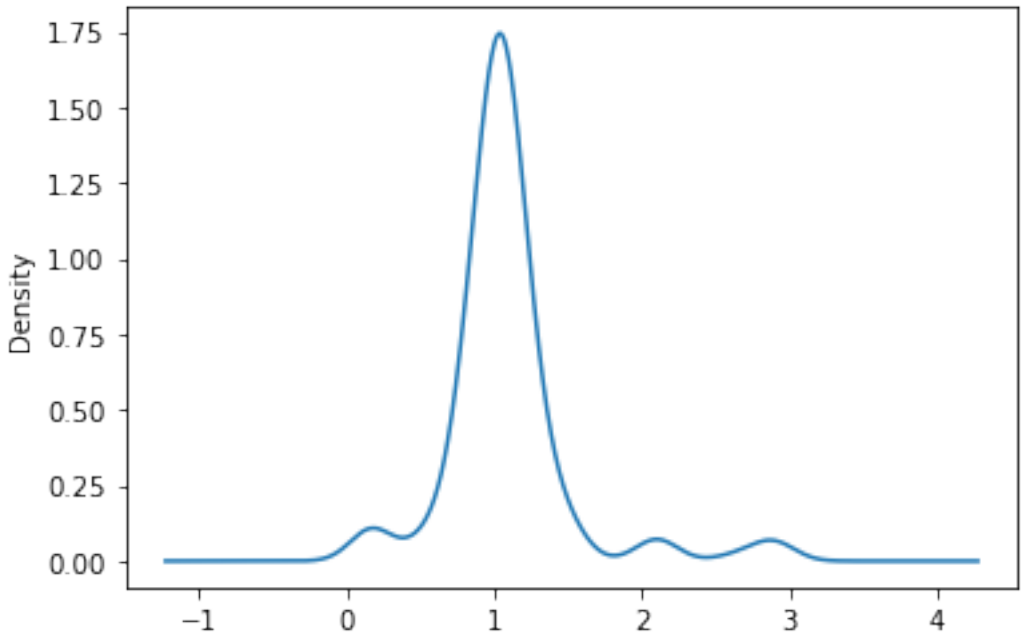


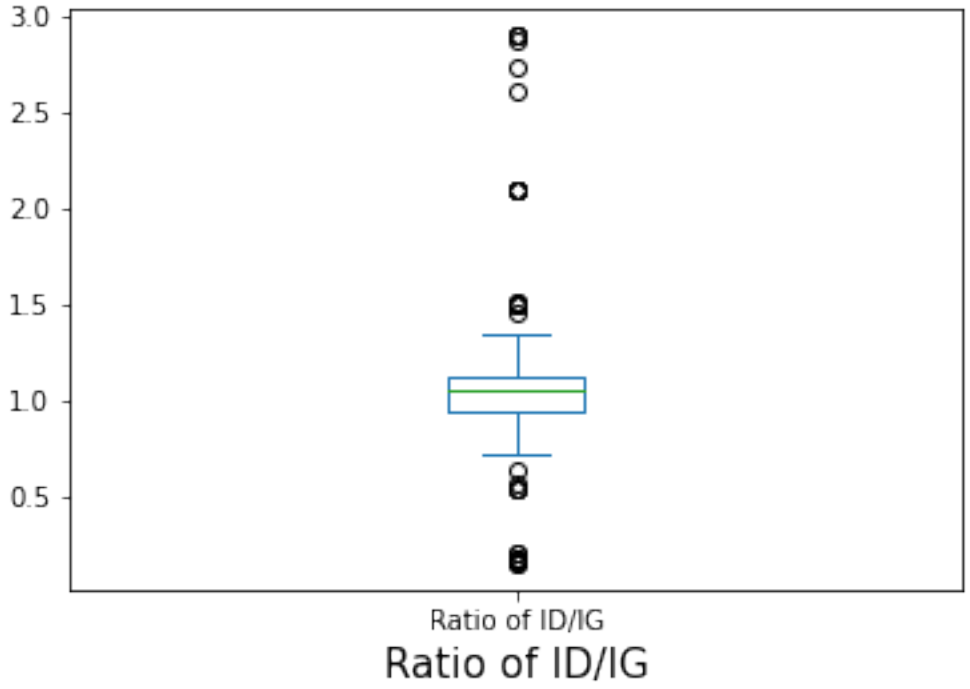


Pore Volume (cm^3/g)

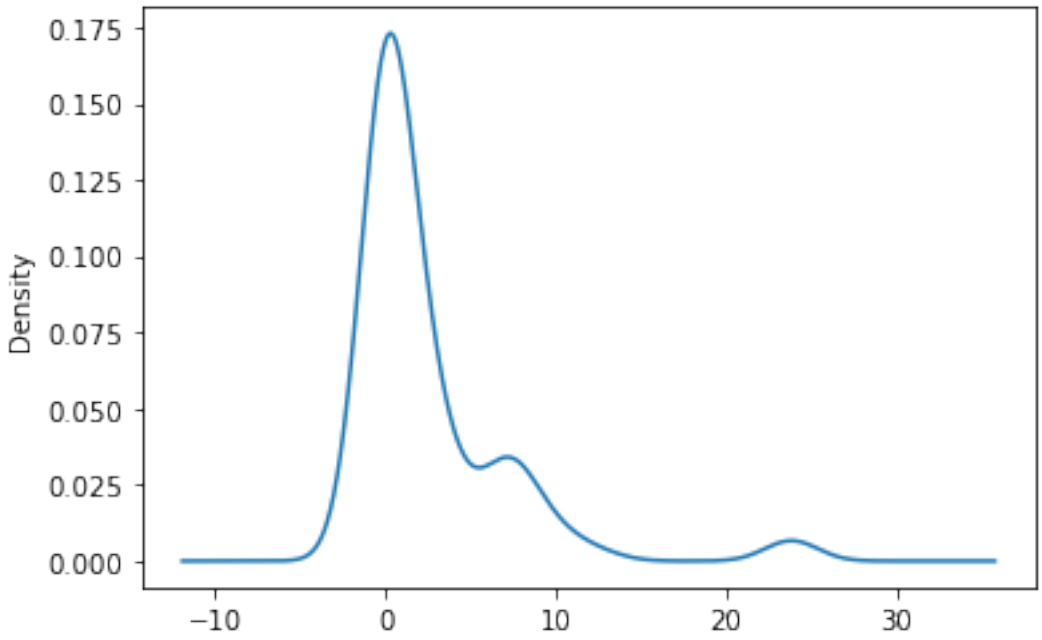
std= 0.59 mean= 0.48
count of available values= 195 count of missing values= 219 count of unique values= 53

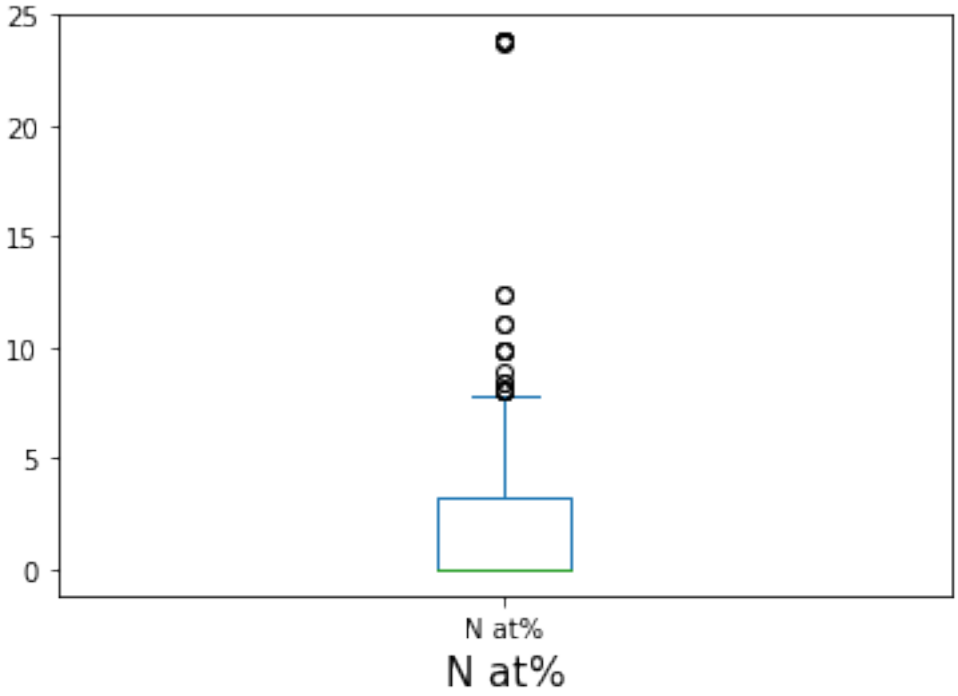
```
available values/total data = 0.471   unique values/available values = 0.272
################################################################################
```



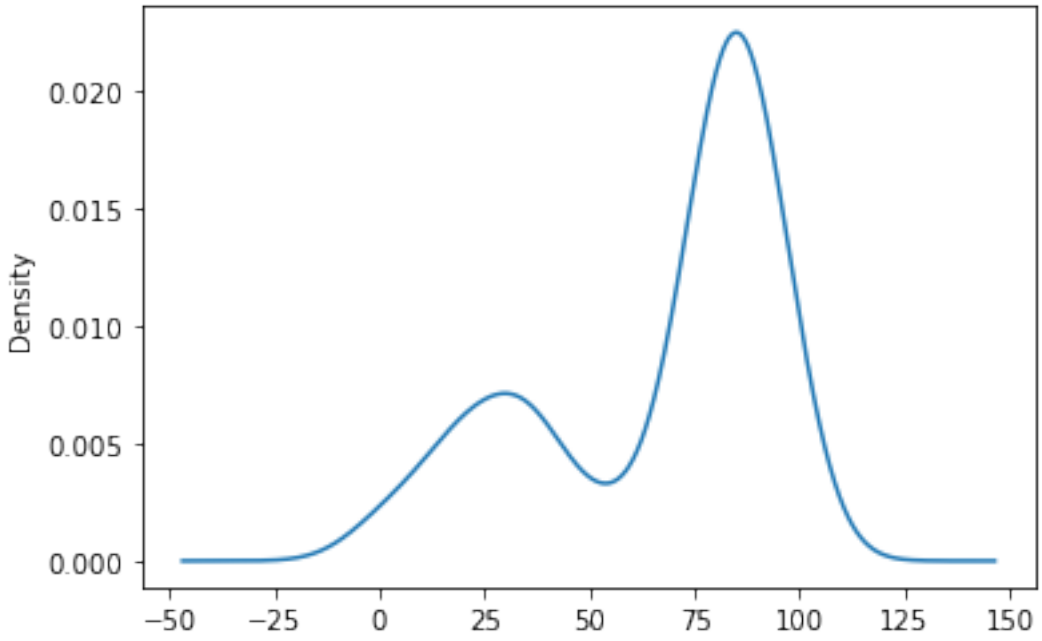


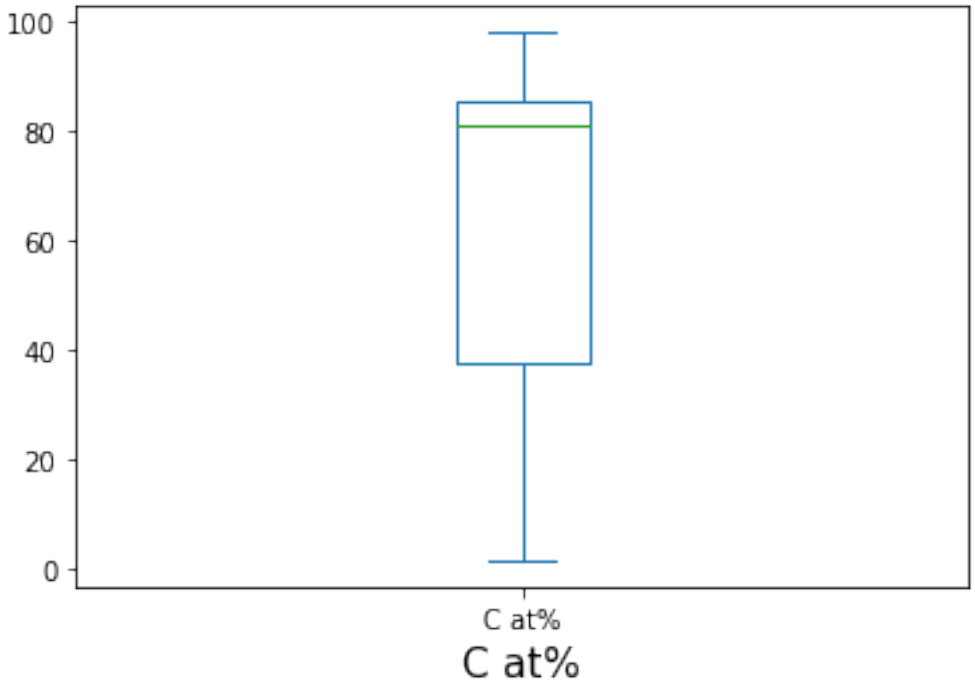
```
std= 0.43   mean= 1.09
count of available values= 232   count of missing values= 182   count of unique
values= 61
available values/total data = 0.560   unique values/available values = 0.263
################################################################################
```





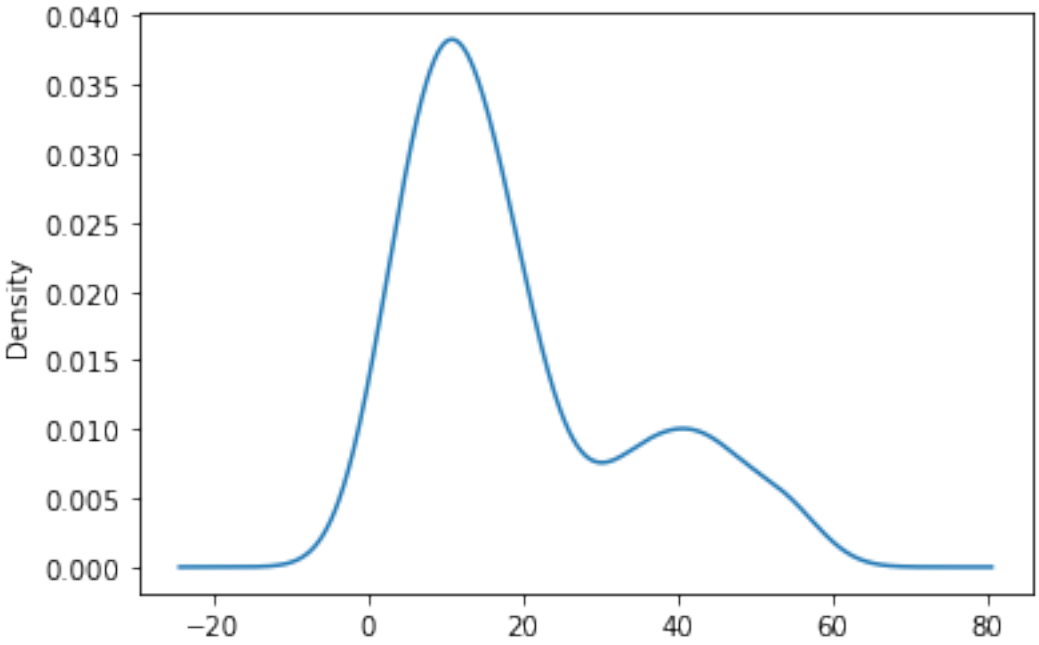

```
std= 4.61   mean= 2.52
count of available values= 231   count of missing values= 183   count of unique
values= 31
available values/total data = 0.558   unique values/available values = 0.134
################################################################################
```

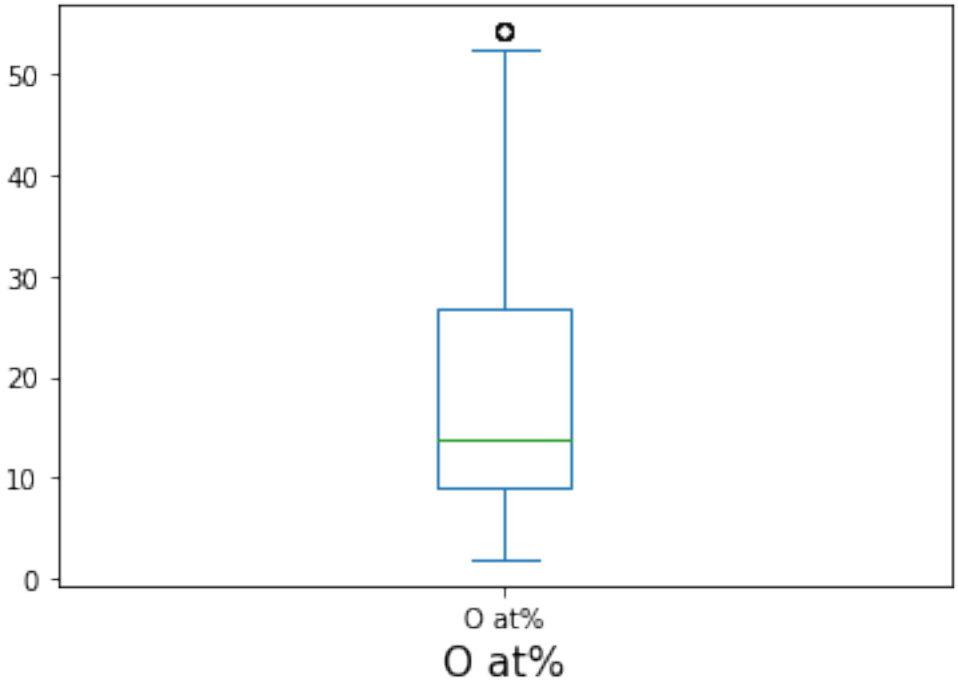




std= 28.53 mean= 66.65
count of available values= 222 count of missing values= 192 count of unique values= 67

```
available values/total data = 0.536   unique values/available values = 0.302
################################################################################
```





```
std= 14.53   mean= 19.09
count of available values= 216   count of missing values= 198   count of unique
values= 67
available values/total data = 0.522   unique values/available values = 0.310
################################################################################
```

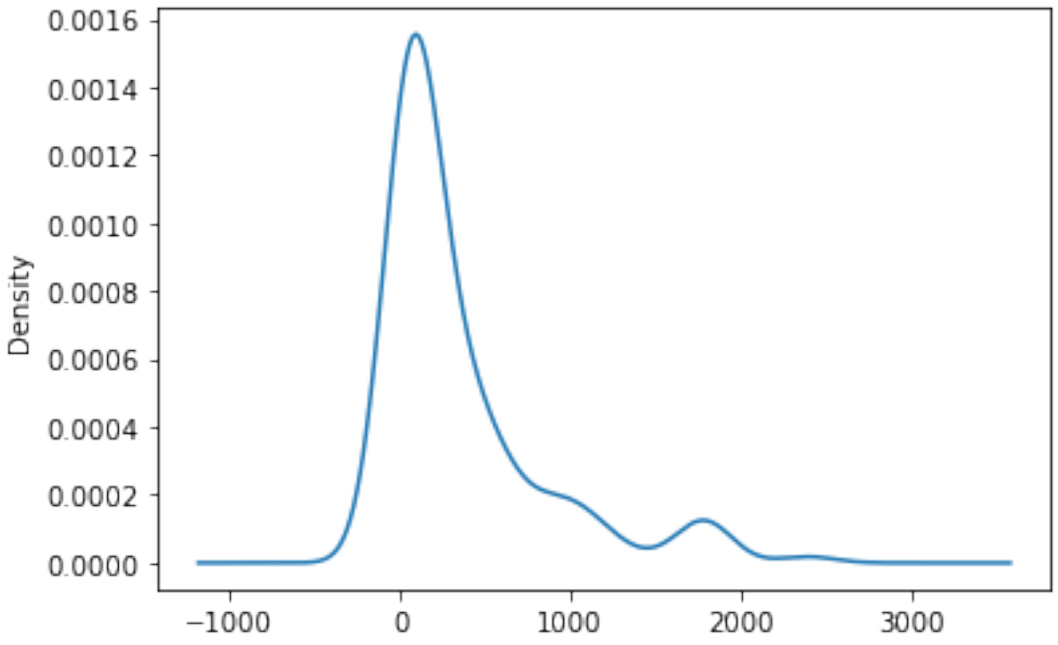
[134]:
```
fillingMissingValues(df8)
```

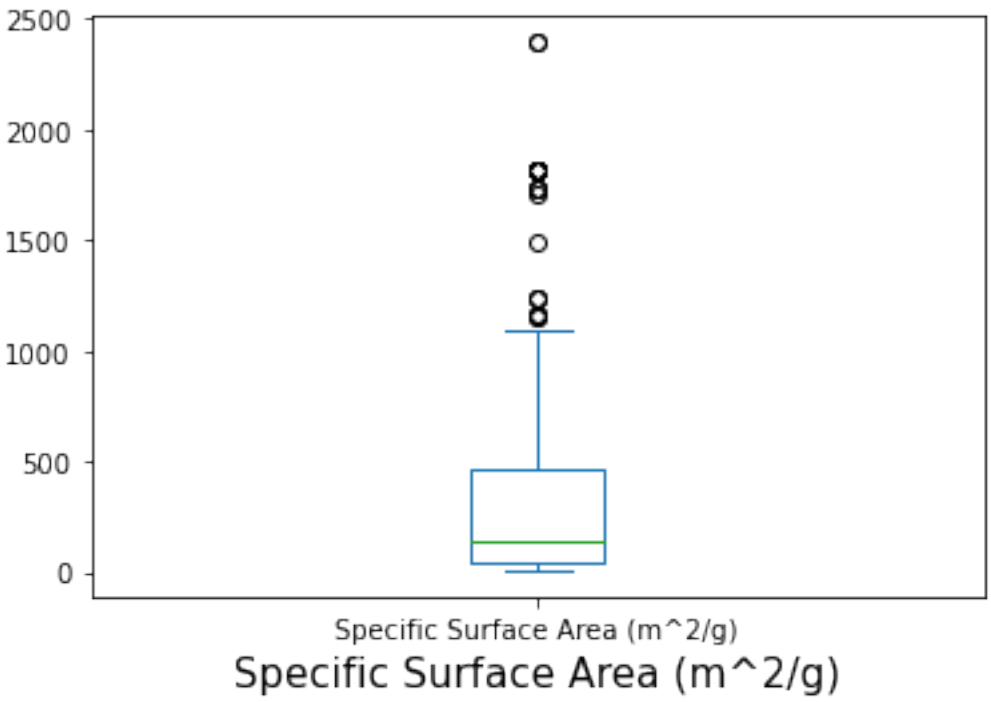
```
column=  Ratio of ID/IG   available ratio= 0.560   unique ratio= 0.263
mean= 1.086   std= 0.427
 missing values filled with 0.8728

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


column=  N at%   available ratio= 0.558   unique ratio= 0.134
mean= 2.522   std= 4.608
 missing values filled with 4.8260

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


/tmp/ipykernel_270313/1691008062.py:66: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[column].fillna((mean-(std/2)), inplace= True)
/tmp/ipykernel_270313/1691008062.py:52: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[column].fillna((mean+(std/2)), inplace= True)
```

[135]:
```
df8.isnull().sum()
```

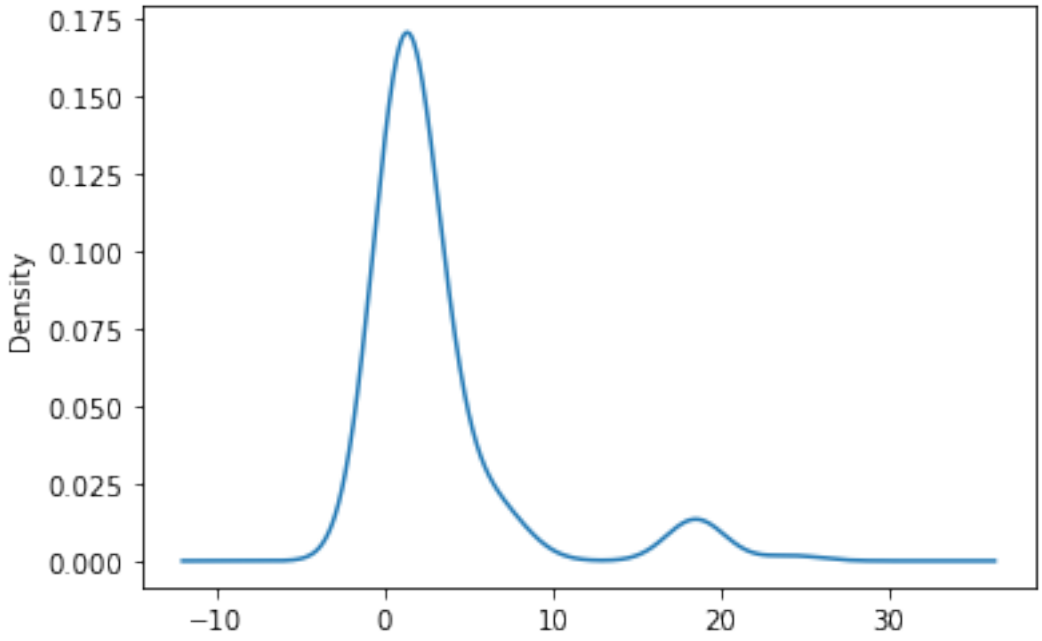
[135]:
```
Limits of Potential Window (V)                        0
Lower Limit of Potential Window (V)                   0
Upper Limit of Potential Window (V)                   0
Potential Window (V)                                  0
Current Density (A/g)                                 0
Capacitance (F/g)                                     0
Specific Surface Area (m^2/g)                       138
Charge Transfer Resistance (Rct) (ohm)              277
Equivalent Series Resistance (Rs) (ohm)             263
Electrode Material                                    0
Pore Size (nm)                                      261
Pore Volume (cm^3/g)                                219
Ratio of ID/IG                                        0
N at%                                                 0
C at%                                               192
O at%                                               198
Electrolyte Chemical Formula                          0
Electrolyte Ion Mobility Ranking                      0
Electrolyte Concentration (M)                         0
Cell Configuration (three/two electrode system)       0
dtype: int64
```

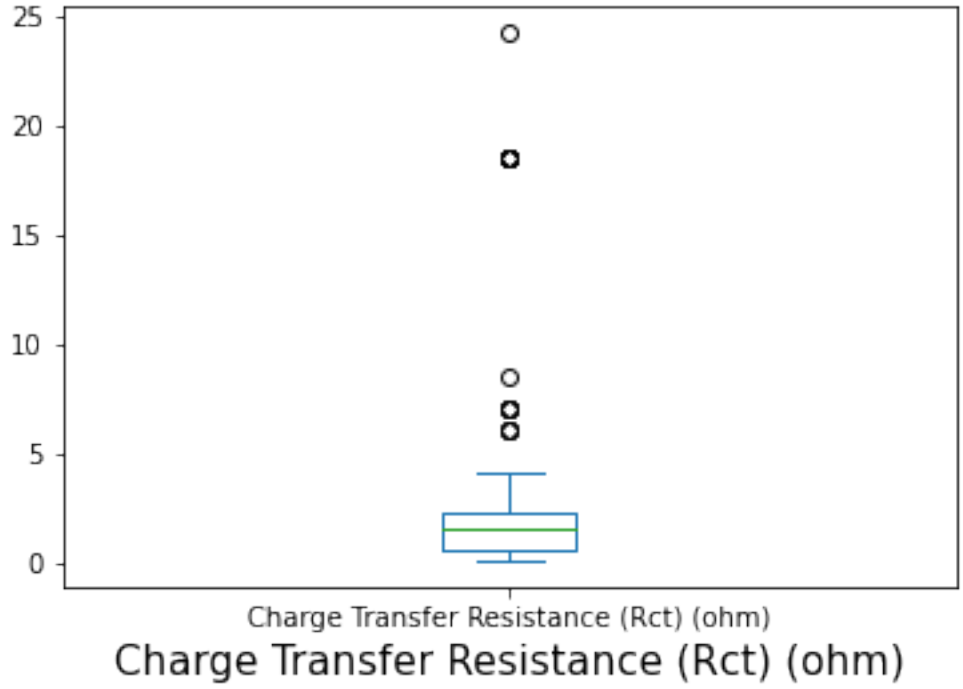
[136]:
```
NullColumnsPlot(df8)
```





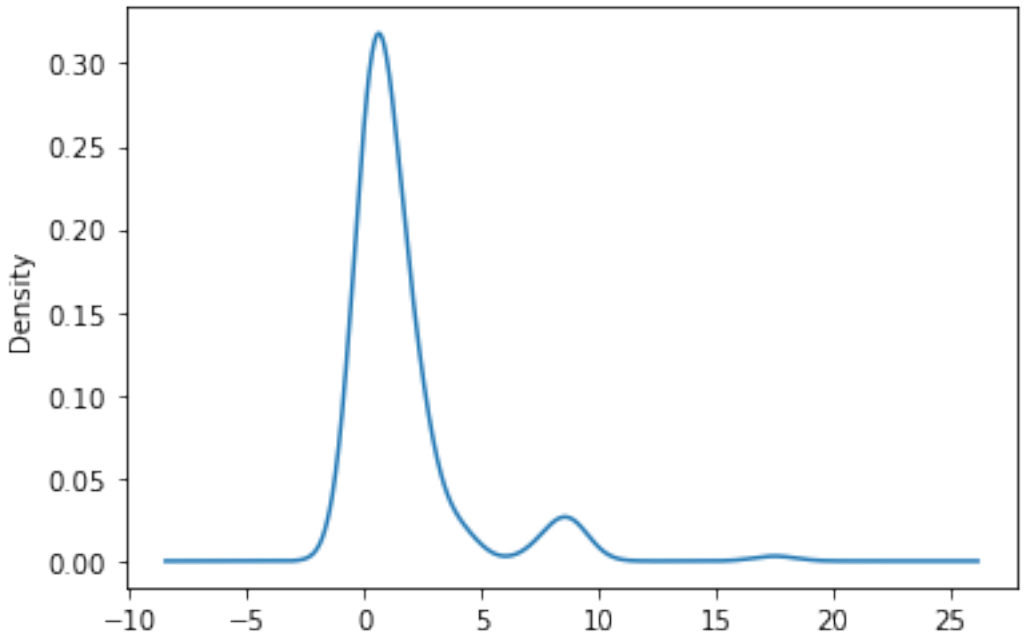

```
std= 485.87   mean= 357.69
count of available values= 276   count of missing values= 138   count of unique
values= 94
available values/total data = 0.667   unique values/available values = 0.341
################################################################################
```

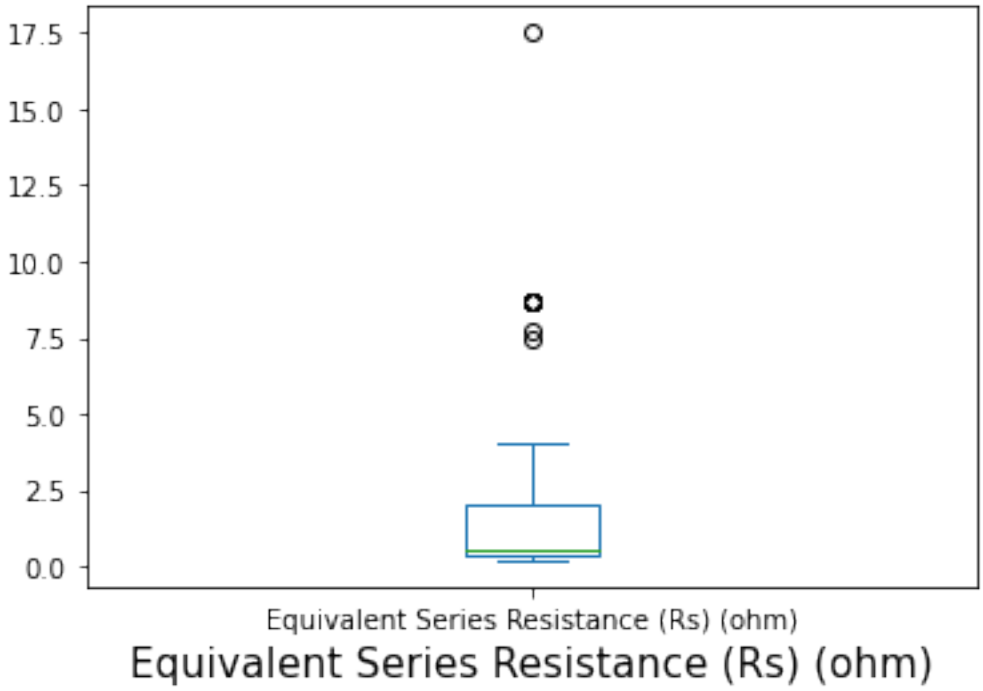




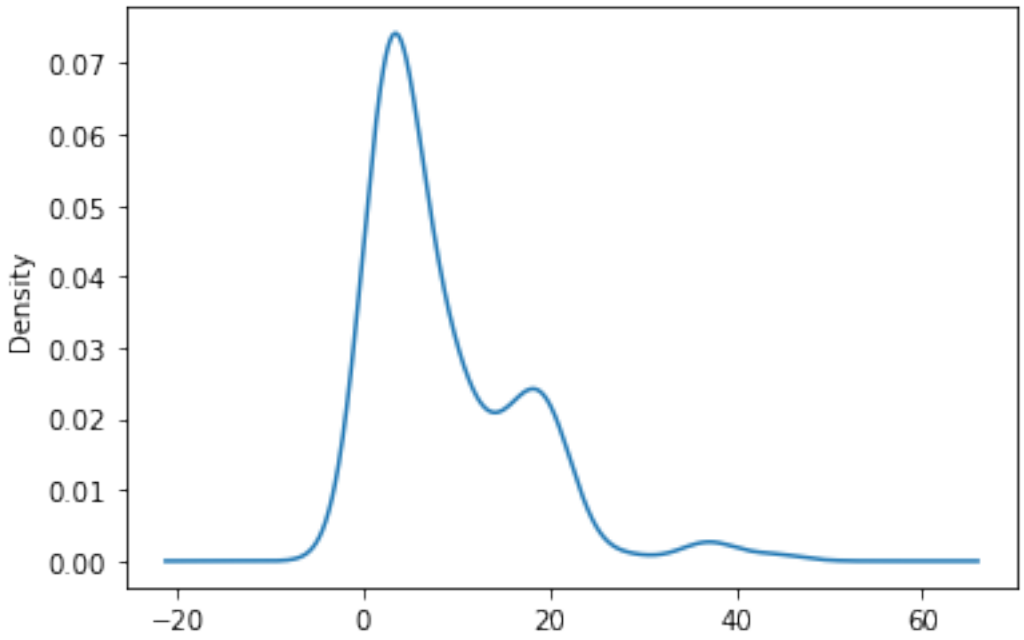
std= 4.63 mean= 3.03
count of available values= 137 count of missing values= 277 count of unique values= 38

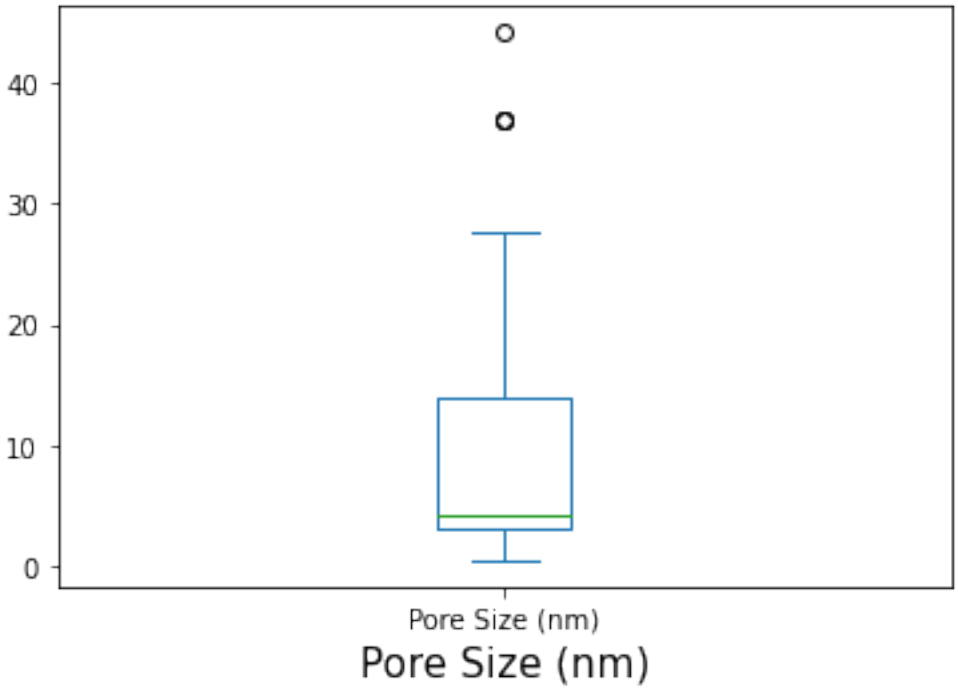
```
available values/total data = 0.331   unique values/available values = 0.277
################################################################################
```



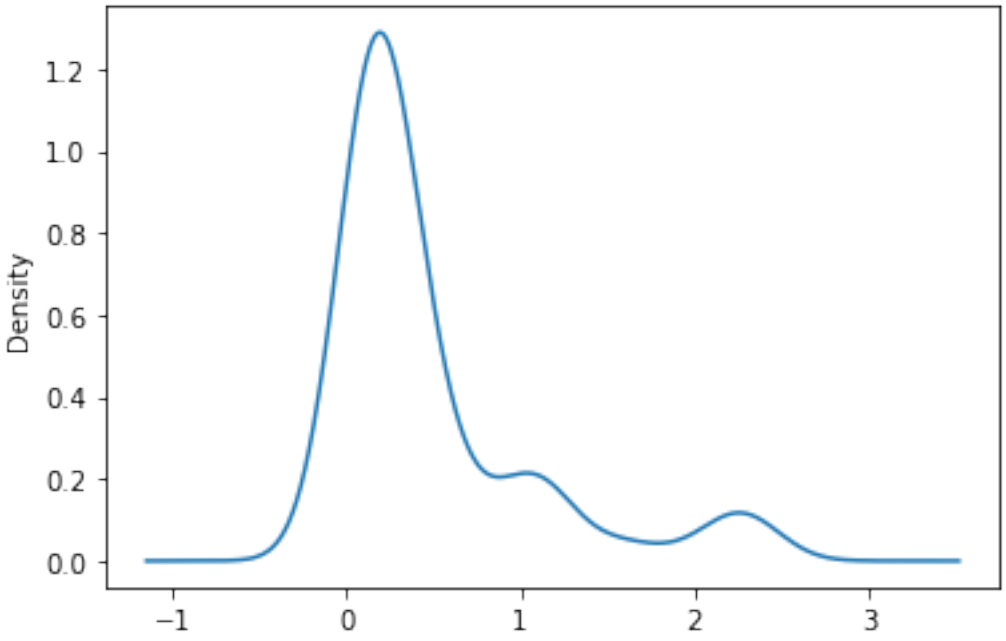


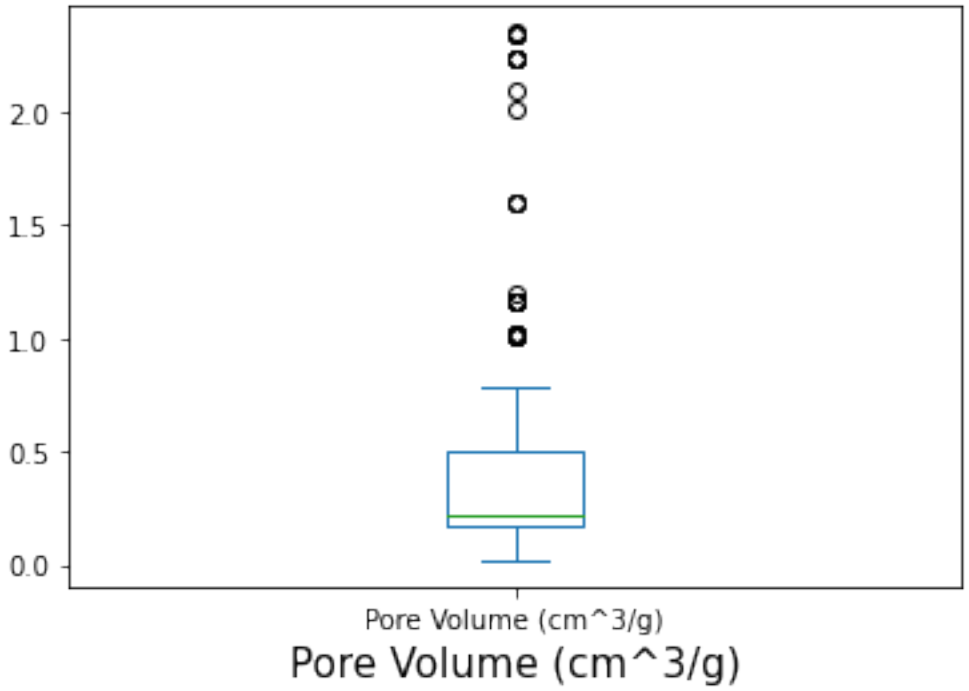
```
std= 2.45   mean= 1.60
count of available values= 151   count of missing values= 263   count of unique
values= 35
available values/total data = 0.365   unique values/available values = 0.232
################################################################################
```






```
std= 8.15   mean= 8.68
count of available values= 153   count of missing values= 261   count of unique
values= 45
available values/total data = 0.370   unique values/available values = 0.294
################################################################################
```

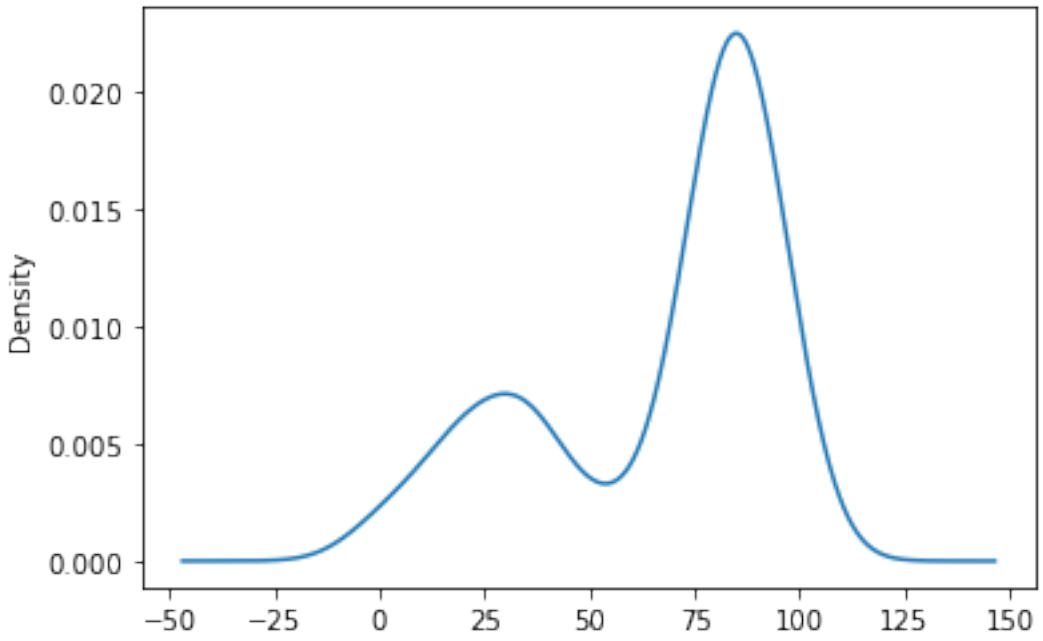


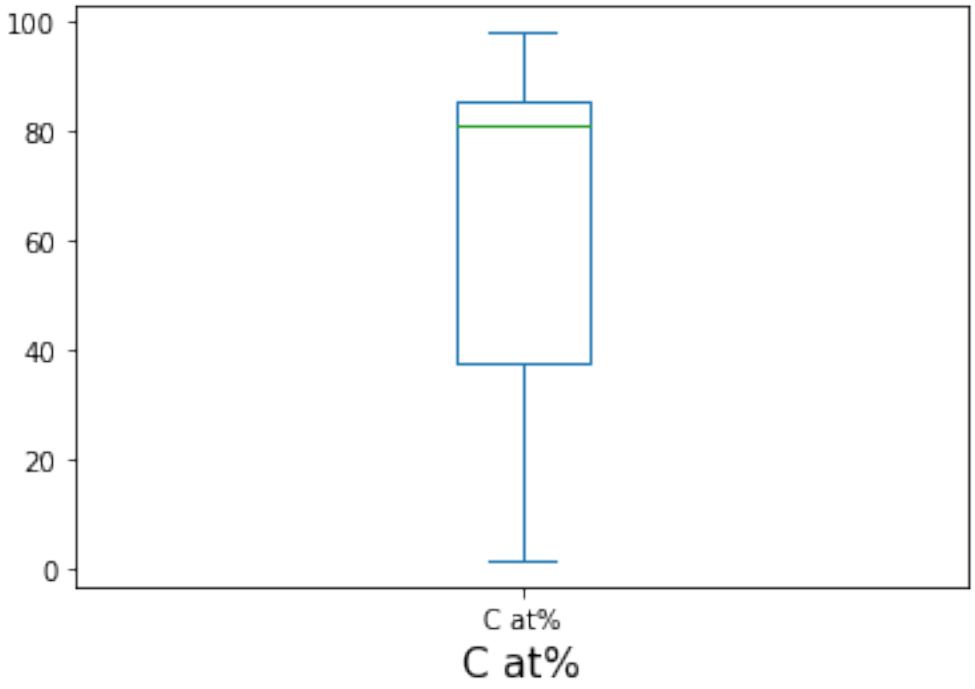


Pore Volume (cm^3/g)

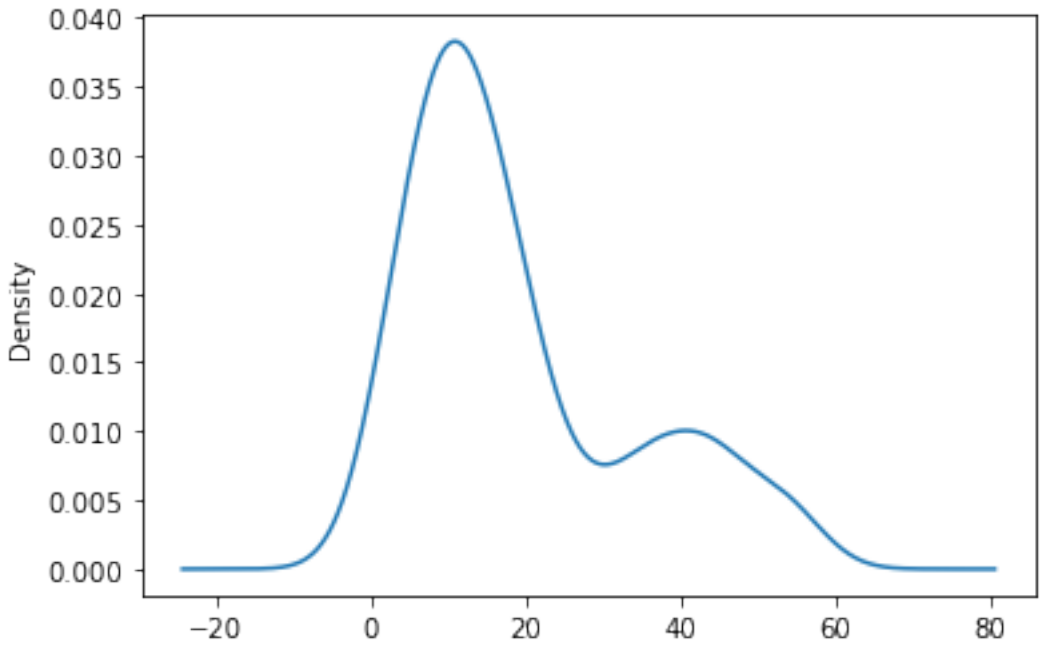
std= 0.59 mean= 0.48
count of available values= 195 count of missing values= 219 count of unique values= 53

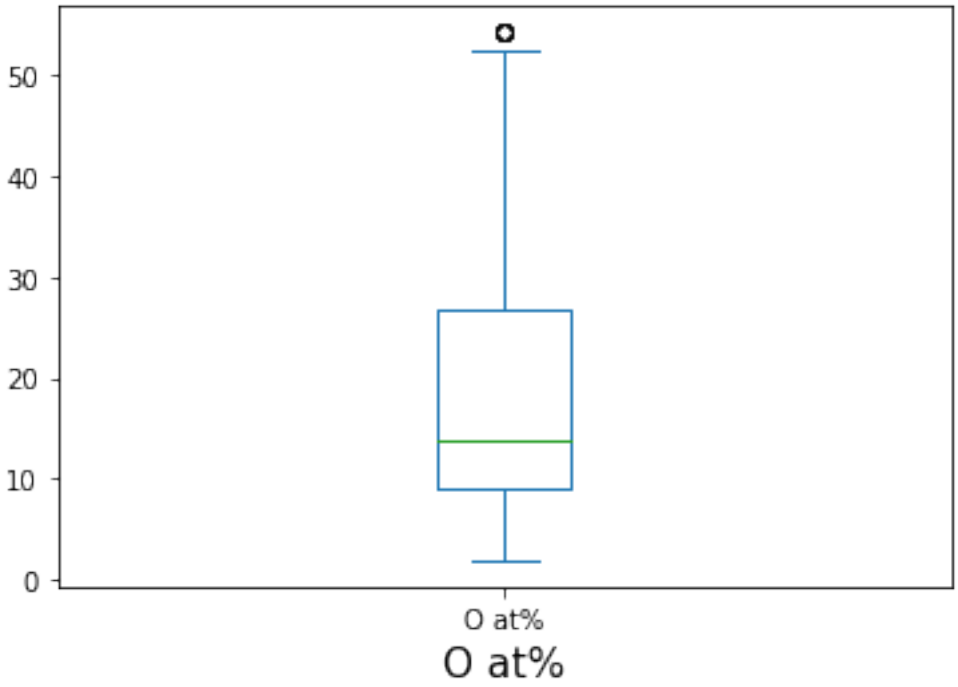
```
available values/total data = 0.471   unique values/available values = 0.272
################################################################################
```





```
std= 28.53   mean= 66.65
count of available values= 222   count of missing values= 192   count of unique
values= 67
available values/total data = 0.536   unique values/available values = 0.302
################################################################################
```






```
std= 14.53   mean= 19.09
count of available values= 216   count of missing values= 198   count of unique
values= 67
available values/total data = 0.522   unique values/available values = 0.310
################################################################################
```

[137]:
```
EmptyColumnNames(df8)
```

[137]:
```
['Specific Surface Area (m^2/g)',
 'Charge Transfer Resistance (Rct) (ohm)',
 'Equivalent Series Resistance (Rs) (ohm)',
 'Pore Size (nm)',
 'Pore Volume (cm^3/g)',
 'C at%',
 'O at%']
```

[138]:
```
df8.shape
```

[138]:
```
(414, 20)
```

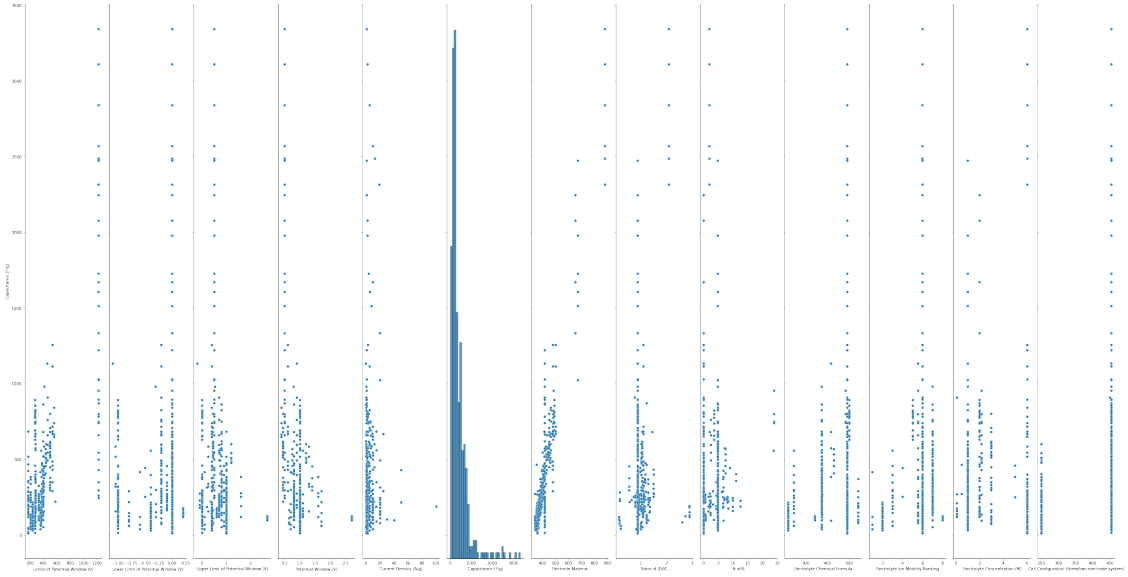
[139]:
```
df=df8.drop(columns=EmptyColumnNames(df8),axis=1)
```

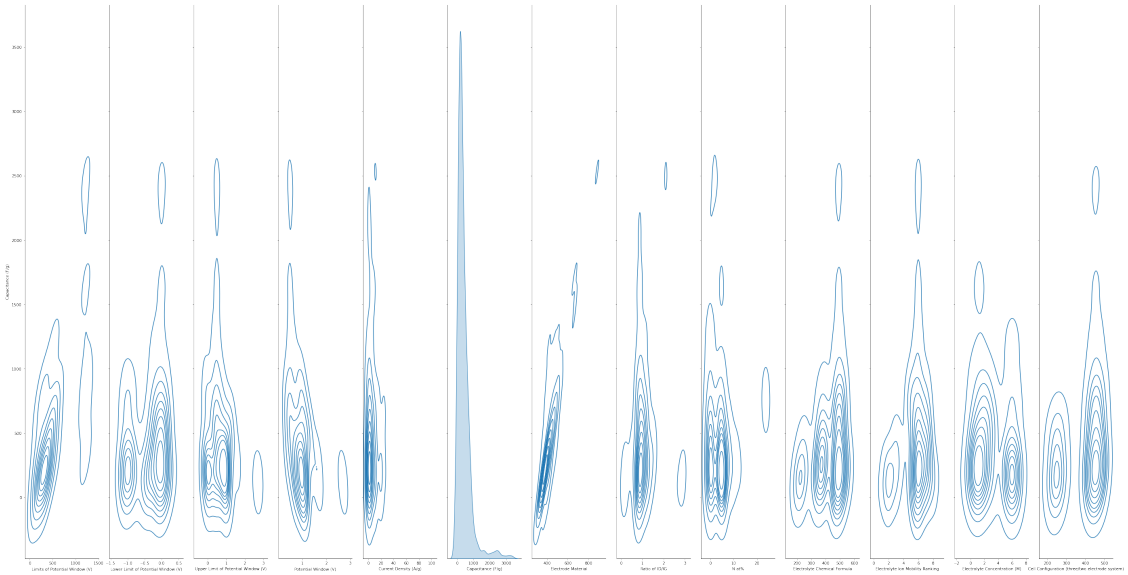
[140]:
```
df.shape
```

[140]: (414, 13)

[141]:
```
sns.pairplot(df,y_vars='Capacitance (F/g)',aspect=3/20,height=20)

sns.pairplot(df,kind='kde',y_vars='Capacitance (F/g)',aspect=3/20,height=20)
```

[141]: <seaborn.axisgrid.PairGrid at 0x7f79ff9f0bb0>





[142]:
```
df.shape
```

[142]:
```
(414, 13)
```

[143]:
```
df.columns
```

[143]:
```
Index(['Limits of Potential Window (V)', 'Lower Limit of Potential Window (V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)', 'Capacitance (F/g)', 'Electrode Material',
       'Ratio of ID/IG', 'N at%', 'Electrolyte Chemical Formula',
       'Electrolyte Ion Mobility Ranking', 'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)'],
      dtype='object')
```

[144]:
```
df.isnull().sum()
```

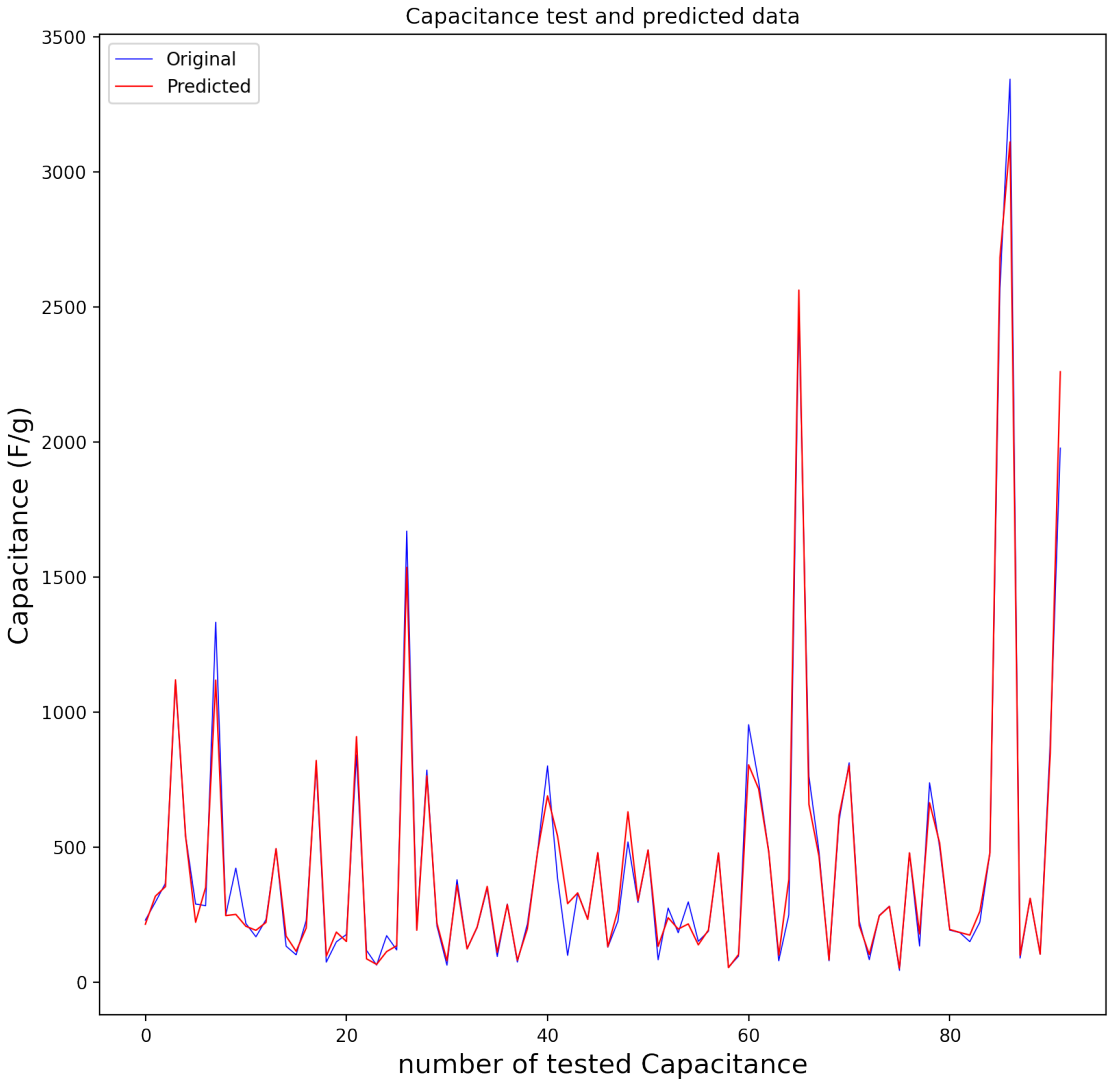
[144]:
```
Limits of Potential Window (V)                    0
Lower Limit of Potential Window (V)               0
Upper Limit of Potential Window (V)               0
Potential Window (V)                              0
Current Density (A/g)                             0
Capacitance (F/g)                                 0
Electrode Material                                0
Ratio of ID/IG                                    0
N at%                                             0
Electrolyte Chemical Formula                      0
Electrolyte Ion Mobility Ranking                  0
Electrolyte Concentration (M)                     0
Cell Configuration (three/two electrode system)   0
dtype: int64
```

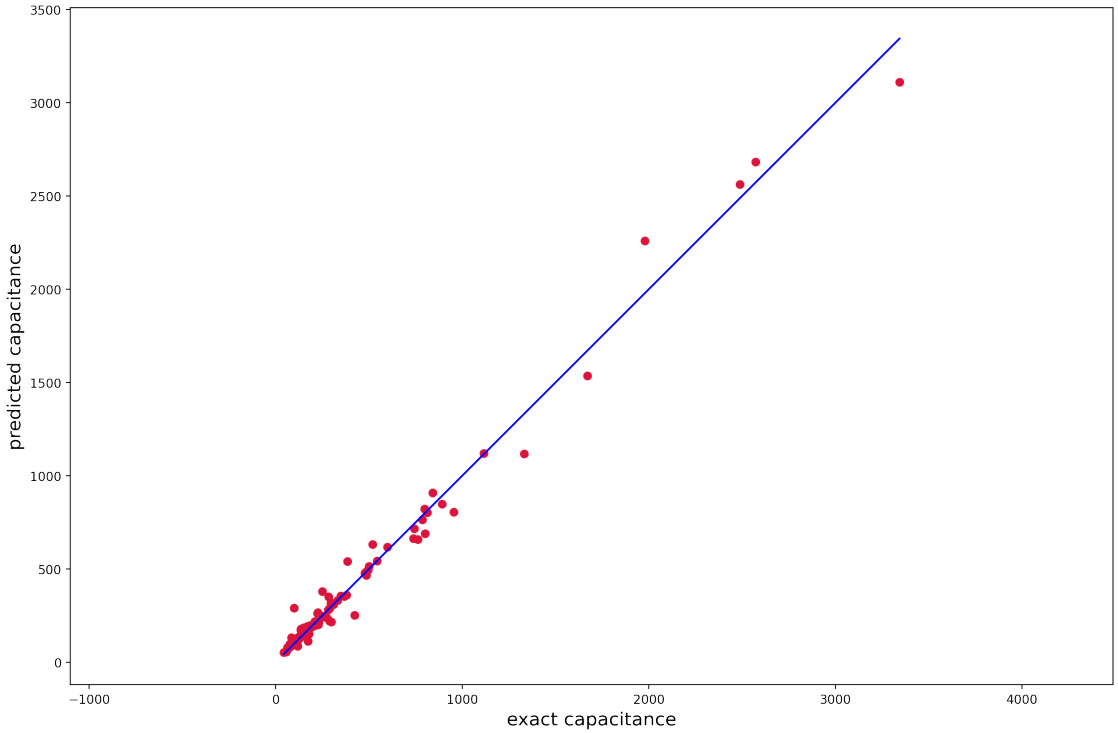
[145]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window␣
 ↪(V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)', 'Electrode Material',
       'Ratio of ID/IG', 'N at%', 'Electrolyte Chemical Formula',
       'Electrolyte Ion Mobility Ranking', 'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']



X=df[Predictors].values
y=df[TargetVariable]
```

```
from sklearn.preprocessing import StandardScaler

PredictorScaler=StandardScaler()


PredictorScalerFit=PredictorScaler.fit(X)


X=PredictorScalerFit.transform(X)
y=y.values.ravel()


from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error , r2_score









opted=ExtraTreesRegressionScore()
```

[146]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],␣
 ↪random_state=opted[2])
etr=ExtraTreesRegressor(n_jobs=-1)
etr.fit(X_train,y_train)


Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)


mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)


TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions


from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
```

```
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()




plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)

print('\n')
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].head(60))
print(2*'\n')
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
print('number of tested data= %i'%count)
print('test_size"percent"= %.2f'%(opted[0]*100))
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
print('optimized MSE: %.4f ' %mse)
print('optimized RMSE: %.4f ' %mse**(0.5))
print('optimized random state: %i'%opted[2] )
```


Capacitance test and predicted data
Original
Predicted
Capacitance (F/g)
number of tested Capacitance
0
500
1000
1500
2000
2500
3000
3500
0
20
40
60
80



	Capacitance (F/g)	PredictedCapacitance (F/g)
0	230.30	215.1620
1	297.00	319.0880
2	367.00	354.3195
3	1115.00	1119.9680
4	542.73	542.7300
5	290.00	222.6396
6	284.00	351.3145
7	1333.33	1118.8150
8	250.00	247.1335
9	423.00	251.7015
10	218.00	207.5750
11	168.70	192.7680
12	232.00	221.8650
13	494.84	494.8400
14	134.00	171.4137
15	102.00	115.0521
16	229.20	201.1430
17	799.00	821.3600
18	74.90	97.1250
19	150.00	186.0900
20	177.20	151.4420

21	842.00	909.6680
22	118.50	87.1680
23	64.00	66.5710
24	173.00	114.0220
25	120.10	135.6000
26	1670.83	1536.2950
27	200.00	193.0480
28	786.00	764.4700
29	209.20	219.0920
30	63.60	80.2950
31	380.00	360.4176
32	124.00	124.0750
33	203.00	204.7300
34	348.00	355.0470
35	95.60	111.4230
36	290.00	287.8660
37	75.20	81.7872
38	216.30	198.7806
39	480.78	480.7800
40	801.60	690.6260
41	385.00	541.6213
42	100.00	291.4750
43	331.00	331.0700
44	235.00	233.2750
45	479.83	479.8300
46	130.10	131.9640
47	226.00	267.6115
48	520.00	631.4420
49	296.00	302.3490
50	489.87	489.8700
51	83.40	133.2020
52	275.00	239.3570
53	183.80	197.3150
54	298.00	216.3765
55	152.00	138.9000
56	189.00	193.5312
57	478.93	478.9300
58	56.00	54.6800
59	97.20	103.3240

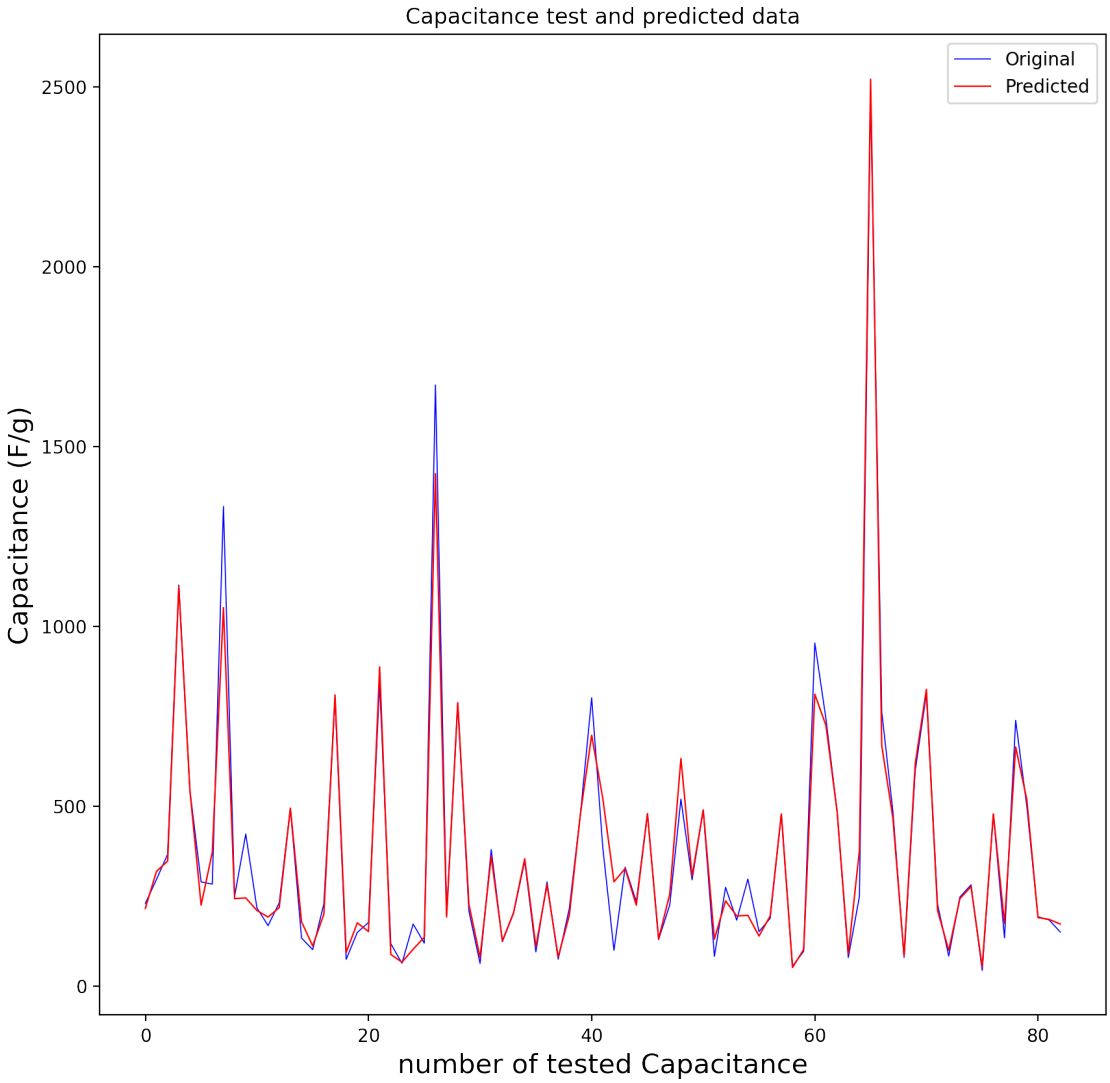
```
total number of data= 414
number of trained data= 323
number of tested data= 91
test_size"percent"= 22.00
score for xtest and ytest : 0.9845
cross validation score: 0.8557
```

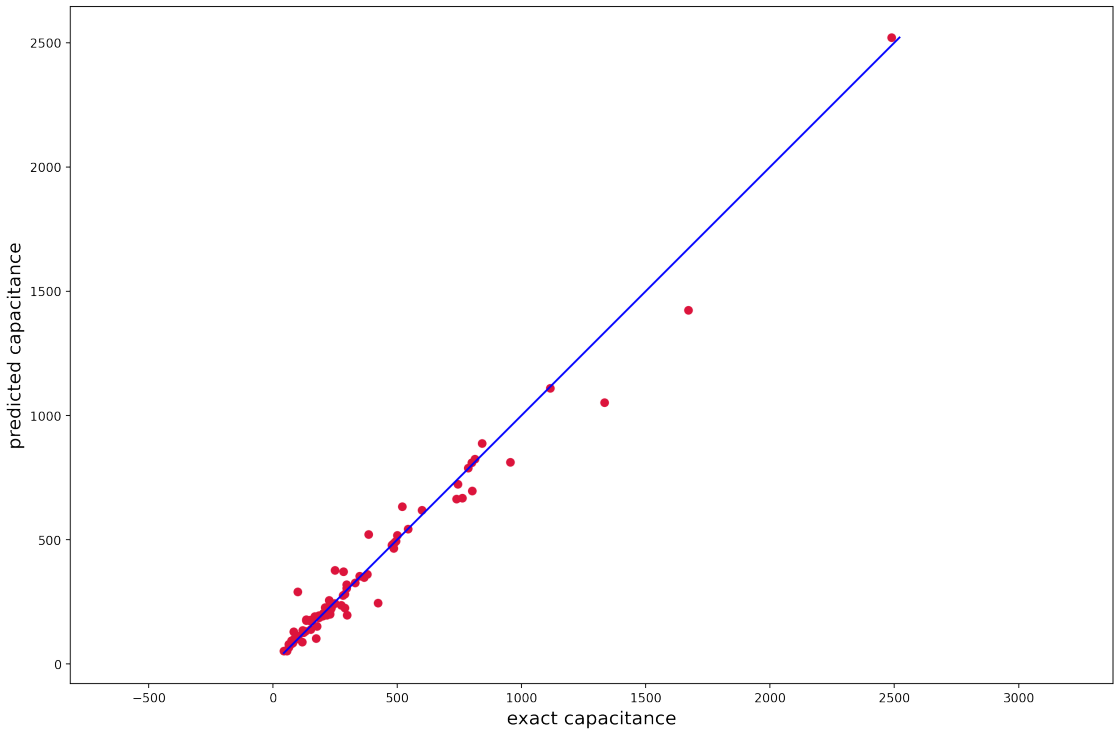
```
optimized MSE: 4739.7316
optimized RMSE: 68.8457
optimized random state: 10
```

[147]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window␣
 ↪(V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)', 'Electrode Material',
       'Ratio of ID/IG', 'N at%', 'Electrolyte Chemical Formula',
       'Electrolyte Ion Mobility Ranking', 'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']


X=df[Predictors].values
y=df[TargetVariable]


PredictorScaler=StandardScaler()
PredictorScalerFit=PredictorScaler.fit(X)


X=PredictorScalerFit.transform(X)
y=y.values.ravel()







opted=ExtraTreesRegressionMse()
```

[148]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],␣
 ↪random_state=opted[2])
etr=ExtraTreesRegressor(n_jobs=-1)
etr.fit(X_train,y_train)


Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)


mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)
```

```
TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions


from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()




plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)

print('\n')
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].head(60))
print(2*'\n')
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
print('number of tested data= %i'%count)
print('test_size"percent"= %.2f'%(opted[0]*100))
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
print('optimized MSE: %.4f ' %mse)
print('optimized RMSE: %.4f ' %mse**(0.5))
print('optimized random state: %i'%opted[2] )
```


Capacitance test and predicted data
Original
Predicted
Capacitance (F/g)
number of tested Capacitance
0
500
1000
1500
2000
2500
0
20
40
60
80



	Capacitance (F/g)	PredictedCapacitance (F/g)
0	230.30	216.3870
1	297.00	319.4390
2	367.00	347.9315
3	1115.00	1110.5900
4	542.73	542.7300
5	290.00	225.7105
6	284.00	372.0888
7	1333.33	1052.2150
8	250.00	243.6050
9	423.00	245.6925
10	218.00	210.6540
11	168.70	192.4460
12	232.00	218.7355
13	494.84	494.8400
14	134.00	179.4360
15	102.00	112.1460
16	229.20	200.3450
17	799.00	809.6400
18	74.90	93.9730
19	150.00	176.4730
20	177.20	152.1150

21	842.00	887.5860
22	118.50	88.5050
23	64.00	67.1470
24	173.00	103.0340
25	120.10	136.3040
26	1670.83	1424.4410
27	200.00	192.8430
28	786.00	788.0900
29	209.20	227.6600
30	63.60	80.2560
31	380.00	360.9748
32	124.00	126.2180
33	203.00	204.7290
34	348.00	354.4300
35	95.60	109.9680
36	290.00	282.1005
37	75.20	81.6975
38	216.30	197.2650
39	480.78	480.7800
40	801.60	697.5760
41	385.00	522.0098
42	100.00	290.4520
43	331.00	327.2200
44	235.00	225.5495
45	479.83	479.8300
46	130.10	131.6000
47	226.00	256.1800
48	520.00	632.8720
49	296.00	306.1400
50	489.87	489.8700
51	83.40	131.2040
52	275.00	237.4230
53	183.80	195.3770
54	298.00	197.1230
55	152.00	139.7510
56	189.00	195.6305
57	478.93	478.9300
58	56.00	52.4000
59	97.20	102.1800

```
total number of data= 414
number of trained data= 332
number of tested data= 82
test_size"percent"= 20.00
score for xtest and ytest : 0.9706
cross validation score: 0.8634
```

```
optimized MSE: 4219.3600
optimized RMSE: 64.9566
optimized random state: 10
```

[149]:
```
df.shape
```

[149]:
```
(414, 13)
```

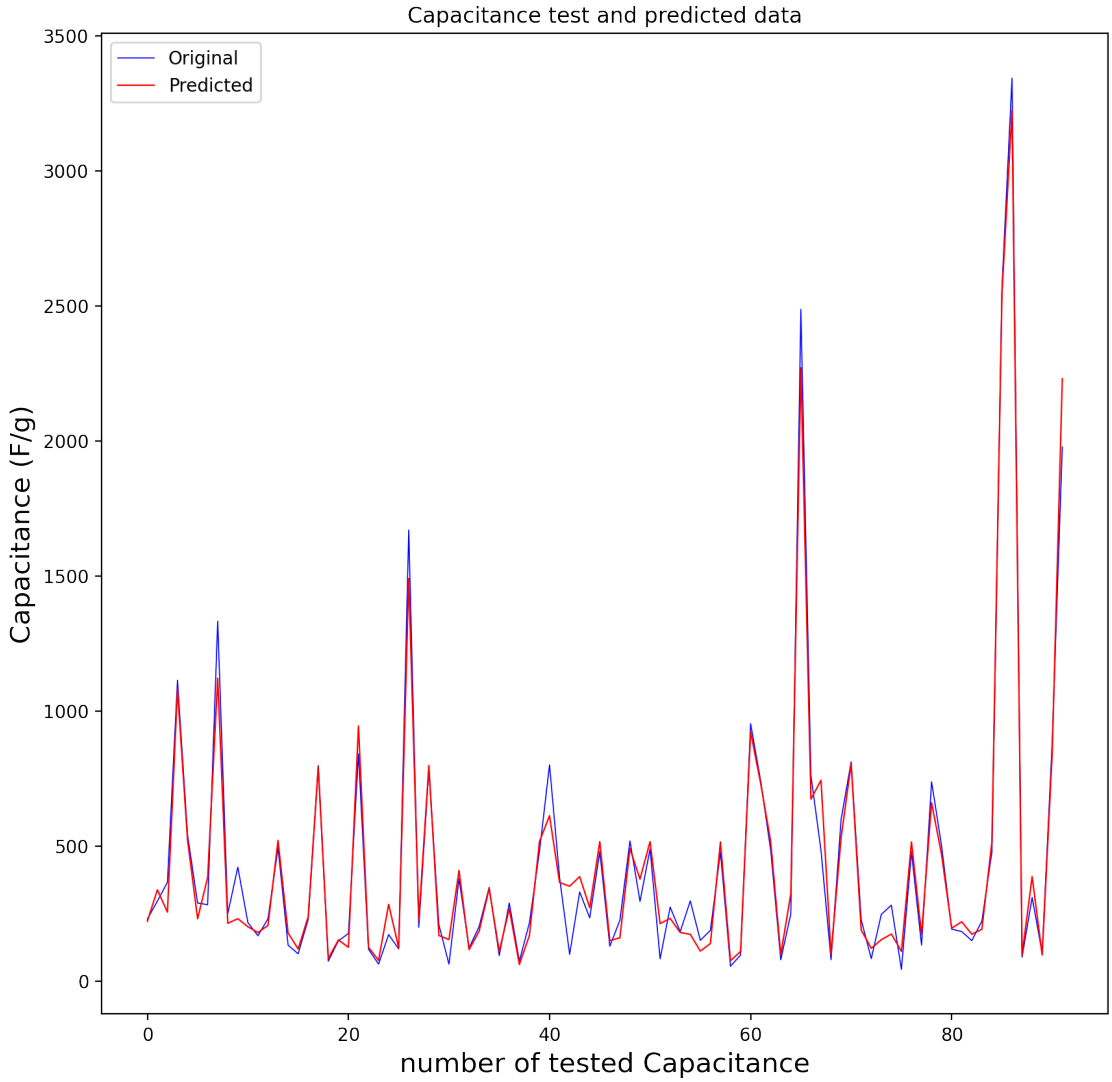
the following model is GBR on the latter df; df6

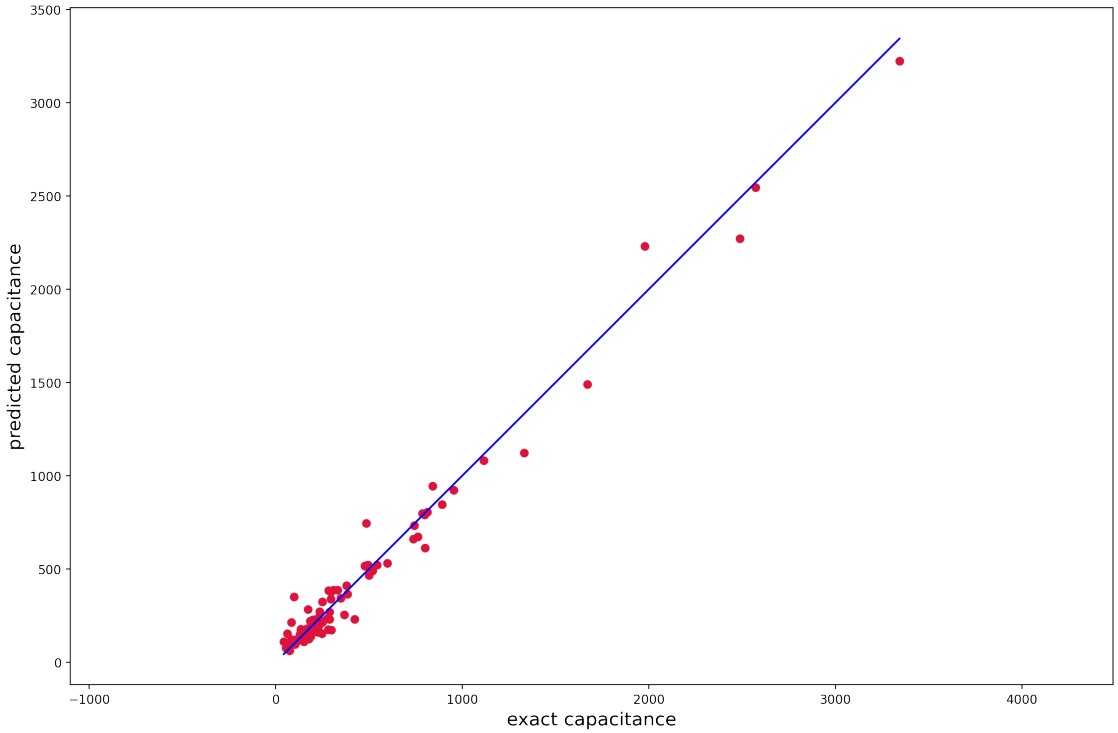
[150]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window␣
 ↪(V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)', 'Electrode Material',
       'Ratio of ID/IG', 'N at%', 'Electrolyte Chemical Formula',
       'Electrolyte Ion Mobility Ranking', 'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']


X=df[Predictors].values
y=df[TargetVariable]




PredictorScaler=StandardScaler()
PredictorScalerFit=PredictorScaler.fit(X)



X=PredictorScalerFit.transform(X)
y=y.values.ravel()




opted=GradientBoostRegScore()
```

[151]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],␣
 ↪random_state=opted[2])
etr=GradientBoostingRegressor(random_state=opted[2])
etr.fit(X_train,y_train)
```

```
Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)


mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)

TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions


from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()




plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)
print('\n')
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].head(60))
print(2*'\n')
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
print('number of tested data= %i'%count)
print('test_size"percent"= %.2f'%(opted[0]*100))
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
```

```
print('optimized MSE: %.4f ' %mse)
print('optimized RMSE: %.4f ' %mse**(0.5))
print('optimized random state: %.4f'%opted[2] )
```





	Capacitance (F/g)	PredictedCapacitance (F/g)
0	230.30	222.183695
1	297.00	338.940048
2	367.00	256.198990
3	1115.00	1081.837734
4	542.73	523.019607
5	290.00	232.104140
6	284.00	384.250204
7	1333.33	1122.355555
8	250.00	214.803433
9	423.00	232.104140
10	218.00	202.243004
11	168.70	180.778901
12	232.00	206.909489
13	494.84	521.733845
14	134.00	179.008929
15	102.00	119.978022
16	229.20	239.367443
17	799.00	792.116716
18	74.90	84.479212
19	150.00	153.348913
20	177.20	126.207146

21	842.00	945.590254
22	118.50	126.207146
23	64.00	77.909052
24	173.00	285.011708
25	120.10	124.505004
26	1670.83	1491.445364
27	200.00	227.891495
28	786.00	799.305678
29	209.20	169.771960
30	63.60	154.772500
31	380.00	410.726125
32	124.00	117.346629
33	203.00	185.505661
34	348.00	343.181579
35	95.60	110.826637
36	290.00	268.981273
37	75.20	62.455343
38	216.30	169.805984
39	480.78	517.256007
40	801.60	613.455107
41	385.00	366.040749
42	100.00	352.136905
43	331.00	387.851556
44	235.00	272.931852
45	479.83	517.256007
46	130.10	150.525477
47	226.00	161.549506
48	520.00	491.510096
49	296.00	378.428631
50	489.87	517.250334
51	83.40	213.887839
52	275.00	233.081358
53	183.80	180.778901
54	298.00	174.086727
55	152.00	112.112399
56	189.00	139.874597
57	478.93	516.313793
58	56.00	76.317964
59	97.20	110.826637

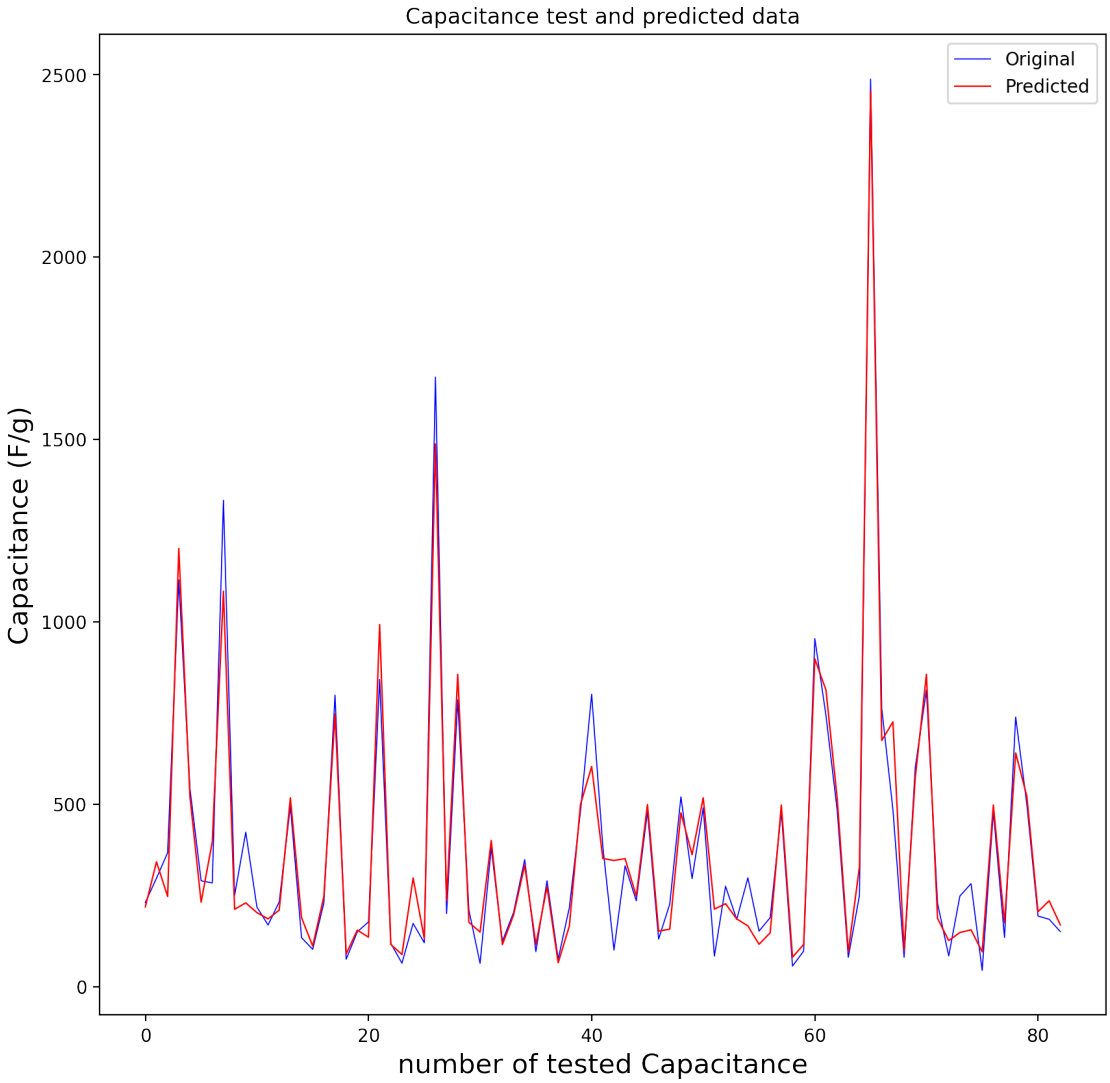
```
total number of data= 414
number of trained data= 323
number of tested data= 91
test_size"percent"= 22.00
score for xtest and ytest : 0.9782
cross validation score: 0.8307
```

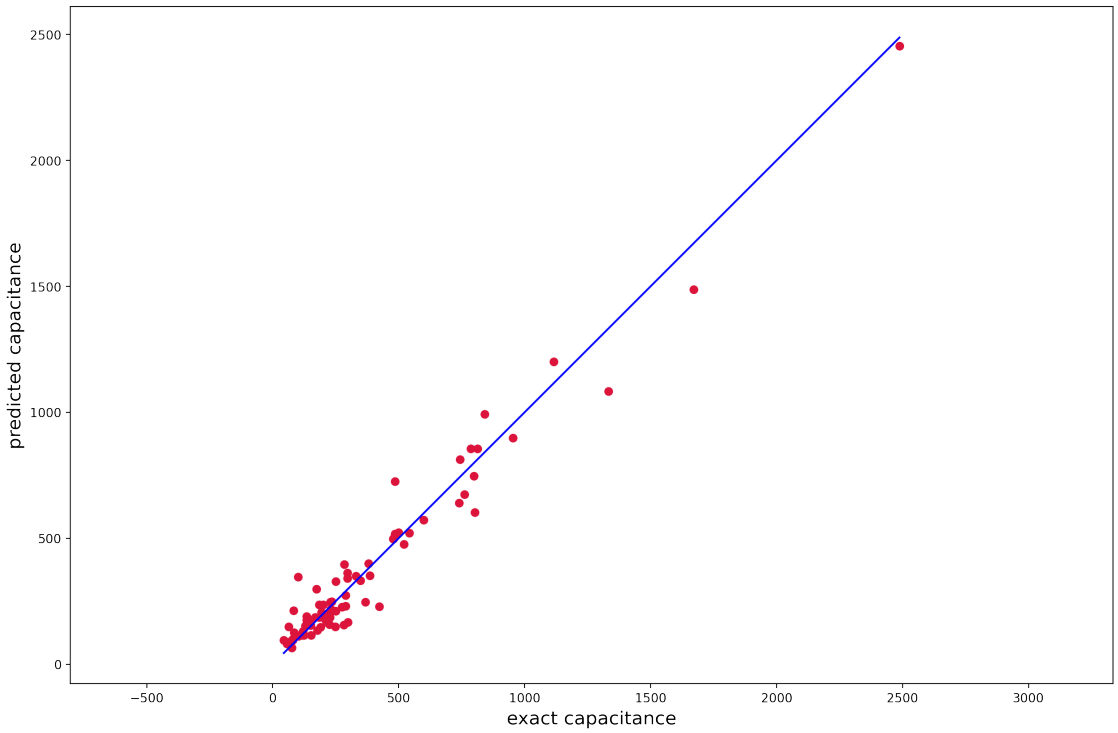
```
optimized MSE: 6659.3253
optimized RMSE: 81.6047
optimized random state: 10.0000
```

[152]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window␣
 ↪(V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)', 'Electrode Material',
       'Ratio of ID/IG', 'N at%', 'Electrolyte Chemical Formula',
       'Electrolyte Ion Mobility Ranking', 'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']


X=df[Predictors].values
y=df[TargetVariable]




PredictorScaler=StandardScaler()
PredictorScalerFit=PredictorScaler.fit(X)


X=PredictorScalerFit.transform(X)
y=y.values.ravel()




opted=GradientBoostRegMse()
```

[153]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],␣
 ↪random_state=opted[2])
etr=GradientBoostingRegressor(random_state=opted[2])
etr.fit(X_train,y_train)


Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)


mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)
```

```
TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions


from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()




plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)
print('\n')
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].head(60))
print(2*'\n')
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
print('number of tested data= %i'%count)
print('test_size"percent"= %.2f'%(opted[0]*100))
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
print('optimized MSE: %.4f ' %mse)
print('optimized RMSE: %.4f ' %mse**(0.5))
print('optimized random state: %.4f'%opted[2] )
```


Capacitance test and predicted data
Original
Predicted
Capacitance (F/g)
number of tested Capacitance
0
500
1000
1500
2000
2500
0
20
40
60
80



	Capacitance (F/g)	PredictedCapacitance (F/g)
0	230.30	218.301771
1	297.00	341.891583
2	367.00	246.909301
3	1115.00	1201.153772
4	542.73	520.217023
5	290.00	231.362668
6	284.00	396.805563
7	1333.33	1083.901066
8	250.00	211.829210
9	423.00	229.385036
10	218.00	202.717491
11	168.70	185.428490
12	232.00	209.566376
13	494.84	517.372545
14	134.00	188.958072
15	102.00	110.931829
16	229.20	246.828053
17	799.00	746.742240
18	74.90	89.483519
19	150.00	154.706772
20	177.20	135.311618

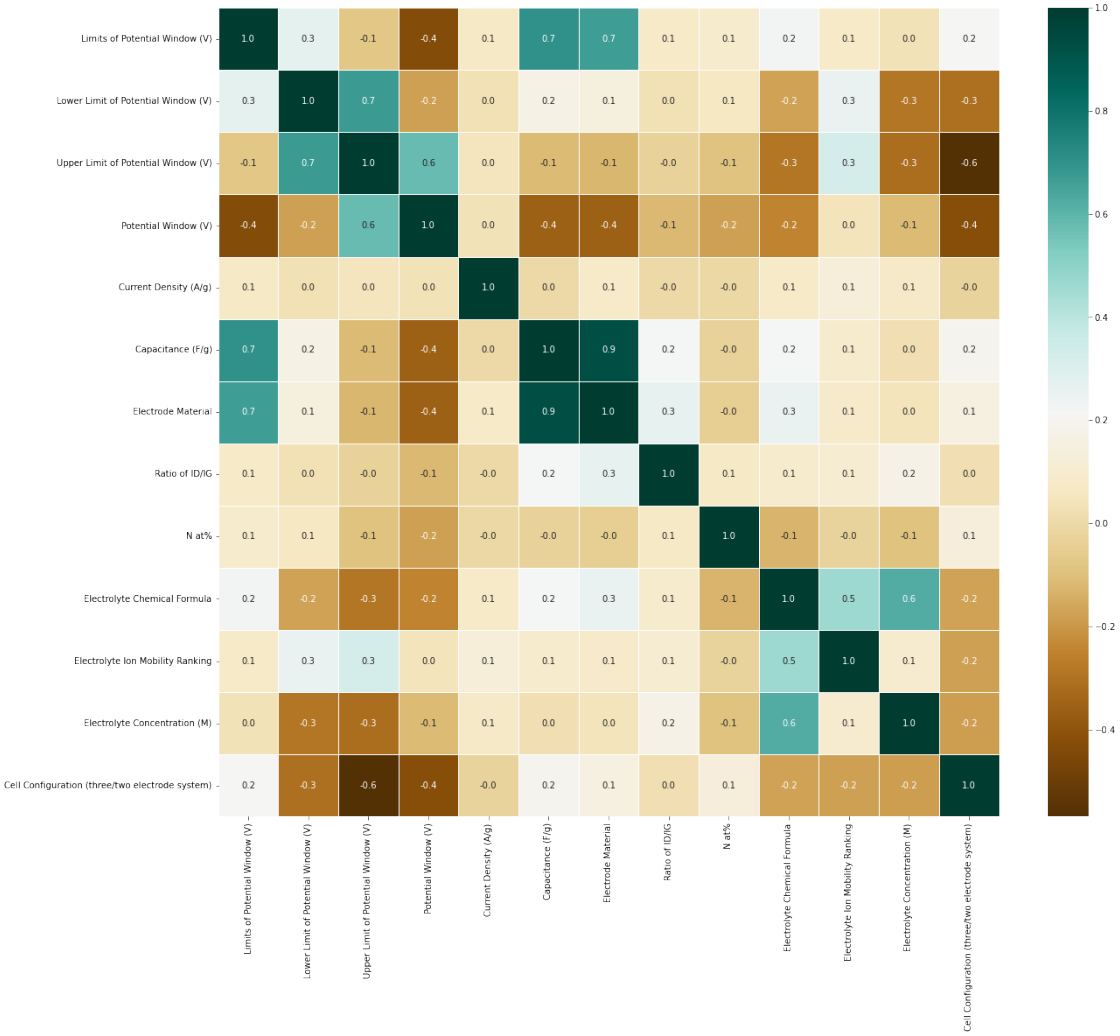
21	842.00	992.551832
22	118.50	115.379535
23	64.00	87.870820
24	173.00	297.715171
25	120.10	131.586133
26	1670.83	1487.626285
27	200.00	235.186199
28	786.00	856.189867
29	209.20	176.038258
30	63.60	149.414130
31	380.00	400.488663
32	124.00	114.879087
33	203.00	195.850600
34	348.00	332.781538
35	95.60	115.271110
36	290.00	273.288668
37	75.20	65.220329
38	216.30	165.586032
39	480.78	499.083045
40	801.60	603.287224
41	385.00	351.144821
42	100.00	345.682640
43	331.00	350.620172
44	235.00	247.735609
45	479.83	499.083045
46	130.10	151.916870
47	226.00	157.318772
48	520.00	476.286643
49	296.00	361.572504
50	489.87	517.372545
51	83.40	212.334344
52	275.00	226.987482
53	183.80	185.428490
54	298.00	166.308773
55	152.00	115.842885
56	189.00	147.340593
57	478.93	497.698481
58	56.00	80.532629
59	97.20	115.271110

```
total number of data= 414
number of trained data= 332
number of tested data= 82
test_size"percent"= 20.00
score for xtest and ytest : 0.9555
cross validation score: 0.8535
```

```
optimized MSE: 6385.3911
optimized RMSE: 79.9086
optimized random state: 10.0000
```

[154]:
```
plt.figure(figsize=(20,17))
heatmap(df.corr(),annot=True,fmt='.1f',linewidths=0.5,cmap='BrBG')
```

[154]: <AxesSubplot:>



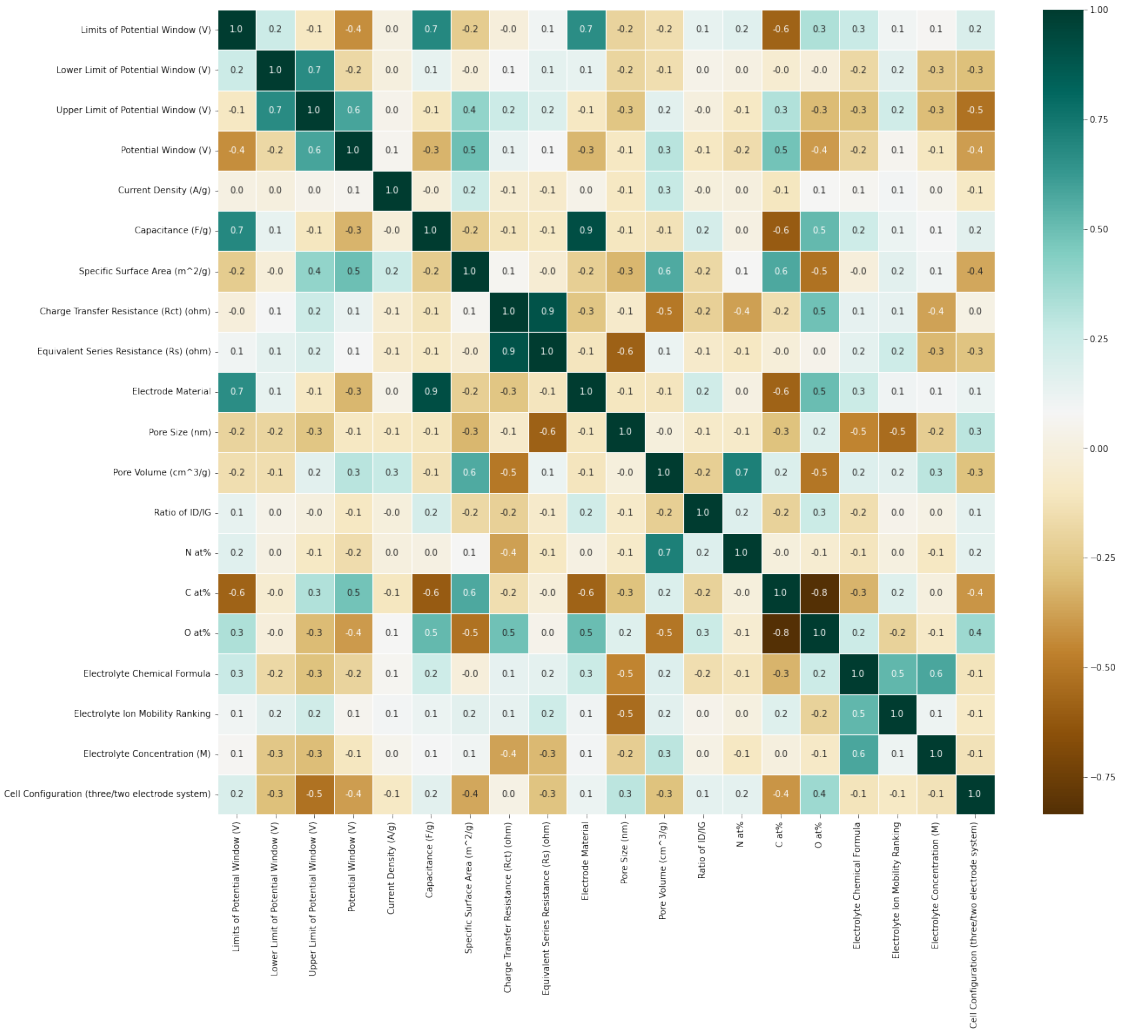
[155]:
```
df9=dframe[9]
```

[156]:
```
df9.shape
```

[156]: (583, 20)

[157]:
```
plt.figure(figsize=(20,17))
heatmap(df9.corr(),annot=True,fmt='.1f',linewidths=0.5,cmap='BrBG')
```

[157]: <AxesSubplot:>



[158]:
```
df9.isnull().sum()
```

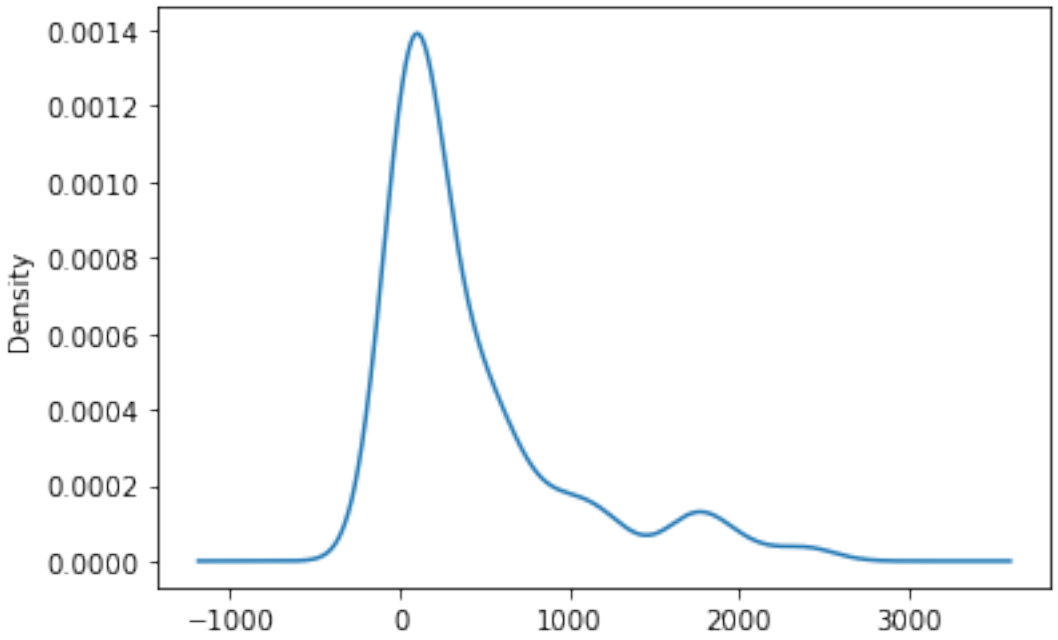
[158]:
```
Limits of Potential Window (V)                  0
Lower Limit of Potential Window (V)             0
Upper Limit of Potential Window (V)             0
Potential Window (V)                            0
Current Density (A/g)                           0
Capacitance (F/g)                               0
Specific Surface Area (m^2/g)                 236
Charge Transfer Resistance (Rct) (ohm)        446
```

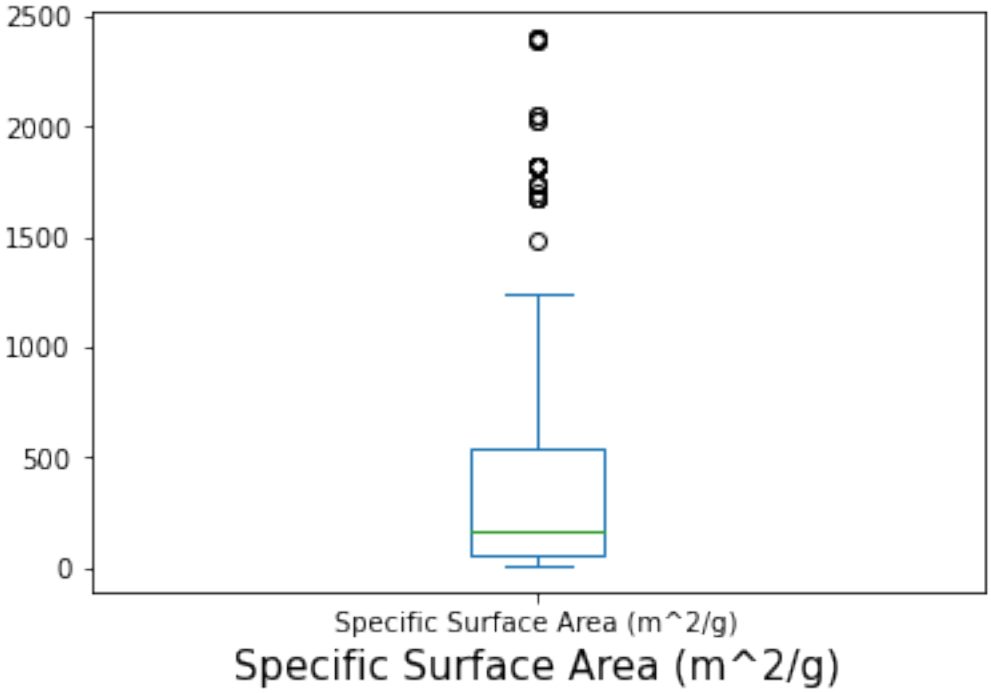
```
Equivalent Series Resistance (Rs) (ohm)            432
Electrode Material                                   0
Pore Size (nm)                                     430
Pore Volume (cm^3/g)                               388
Ratio of ID/IG                                     255
N at%                                              352
C at%                                              361
O at%                                              365
Electrolyte Chemical Formula                         0
Electrolyte Ion Mobility Ranking                     0
Electrolyte Concentration (M)                        0
Cell Configuration (three/two electrode system)      0
dtype: int64
```

[159]:
```
EmptyColumnNames(df9)
```

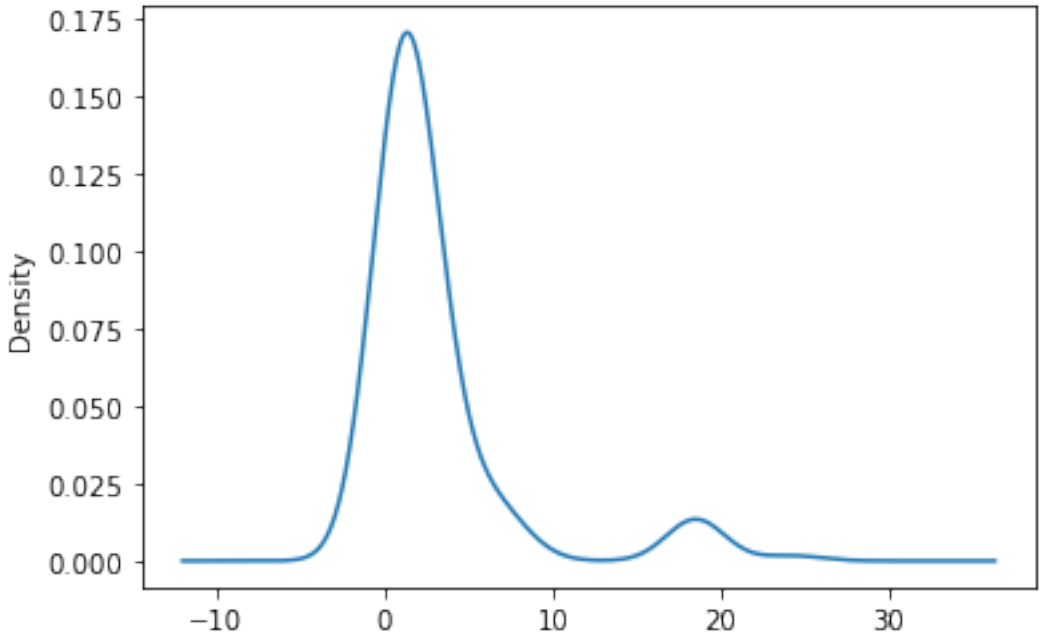
[159]:
```
['Specific Surface Area (m^2/g)',
 'Charge Transfer Resistance (Rct) (ohm)',
 'Equivalent Series Resistance (Rs) (ohm)',
 'Pore Size (nm)',
 'Pore Volume (cm^3/g)',
 'Ratio of ID/IG',
 'N at%',
 'C at%',
 'O at%']
```

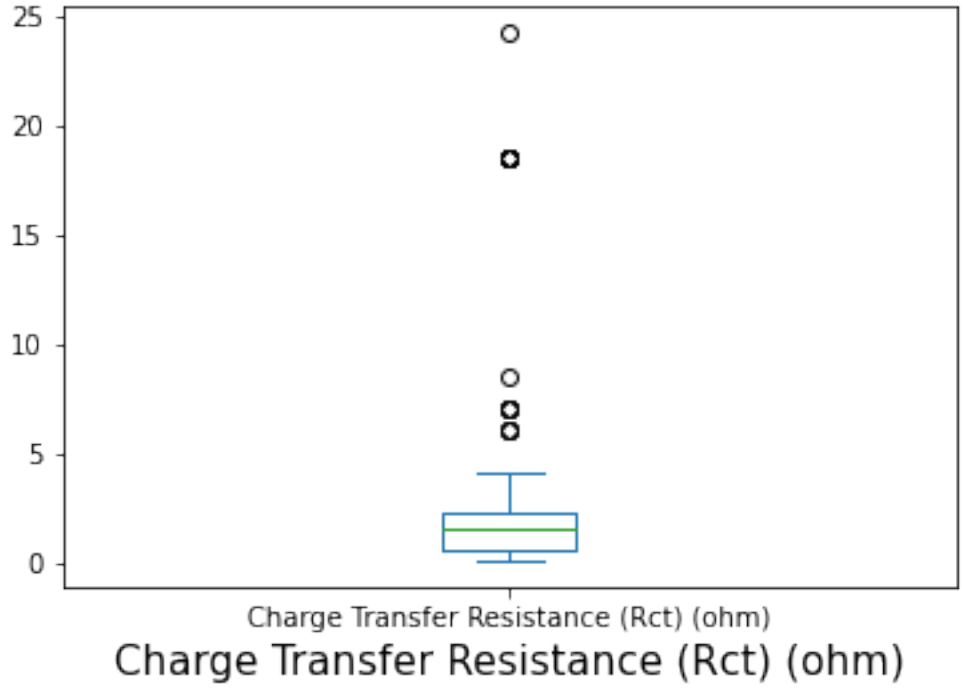
[160]:
```
NullColumnsPlot(df9)
```





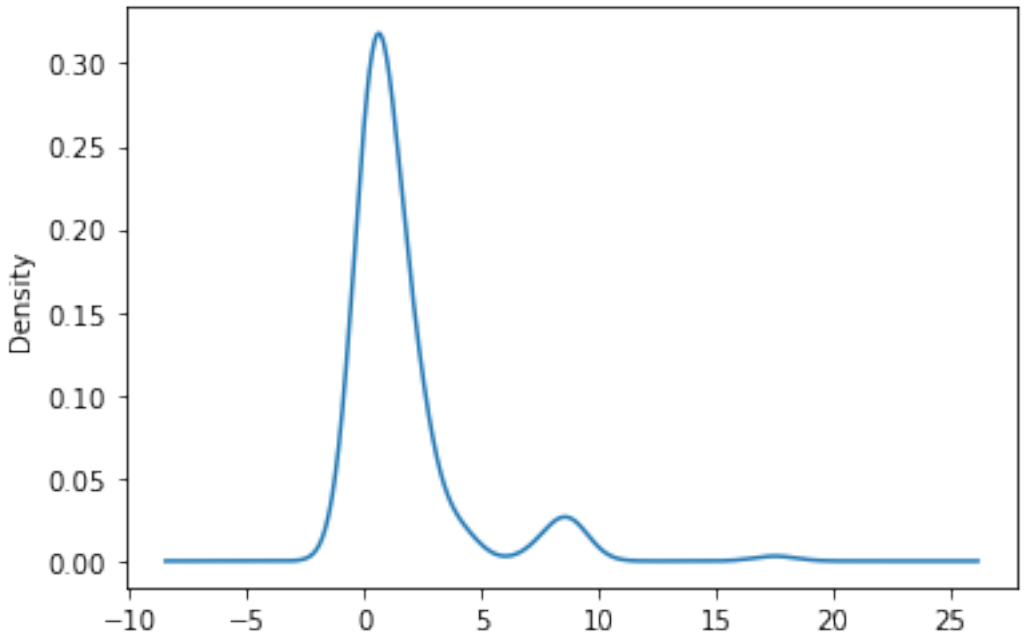

```
std= 547.67   mean= 415.99
count of available values= 347   count of missing values= 236   count of unique
values= 115
available values/total data = 0.595   unique values/available values = 0.331
################################################################################
```

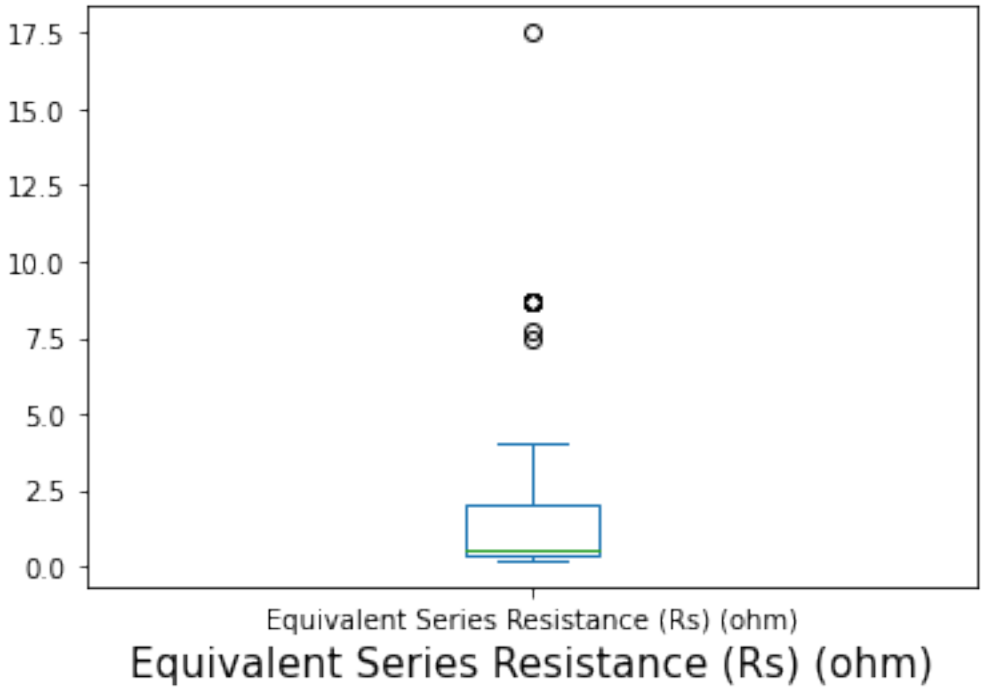




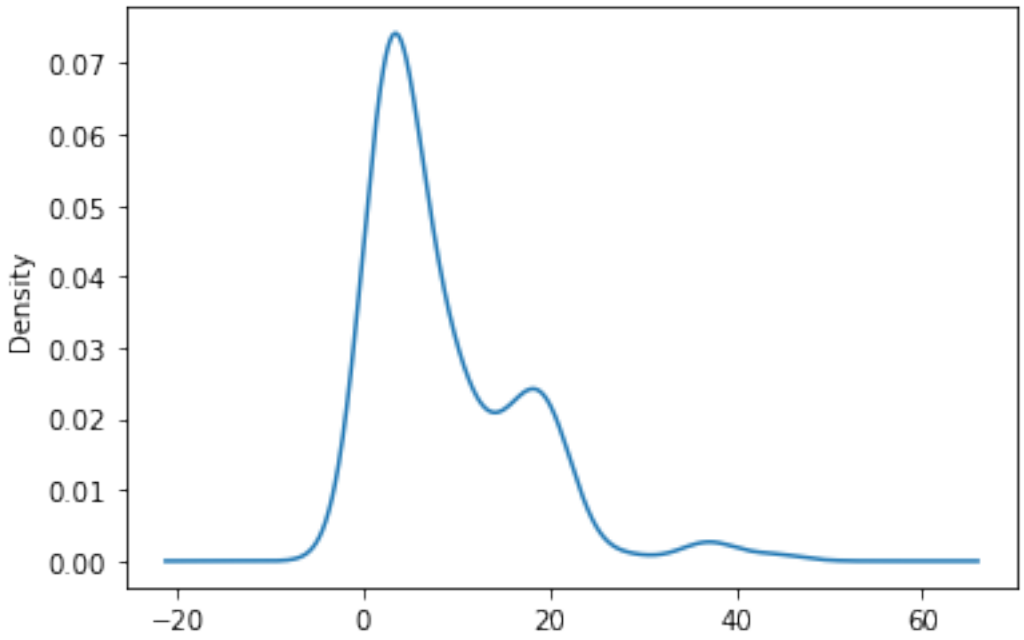

```
std= 4.63   mean= 3.03
count of available values= 137   count of missing values= 446   count of unique
values= 38
```

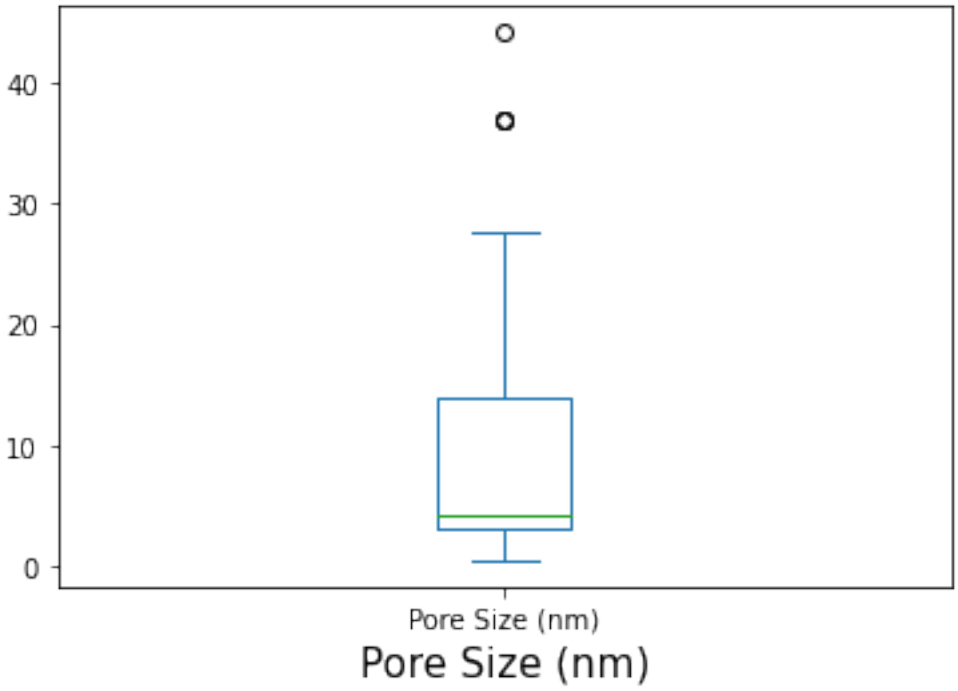
```
available values/total data = 0.235   unique values/available values = 0.277
################################################################################
```



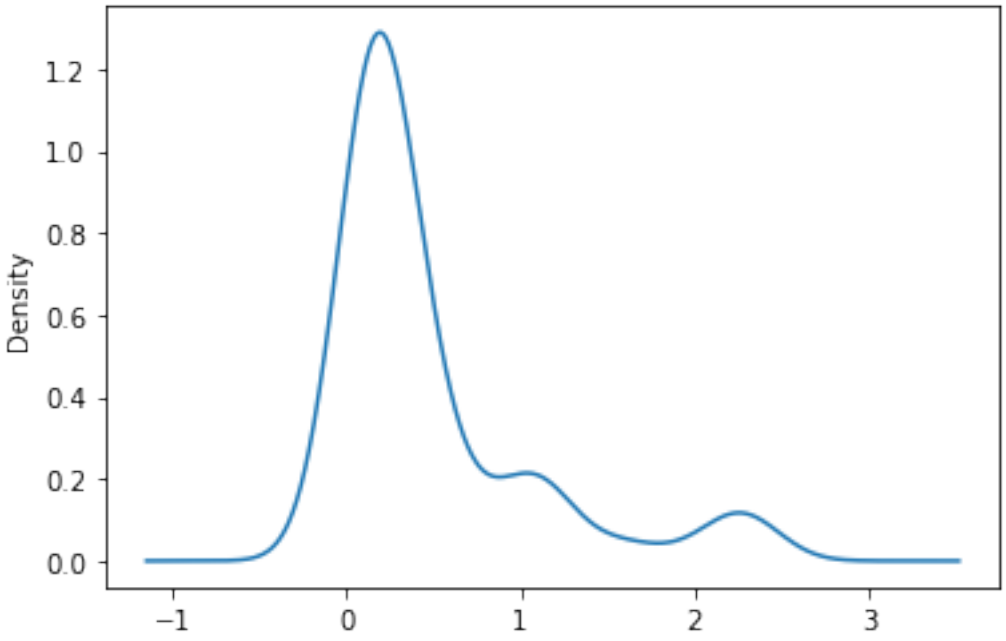


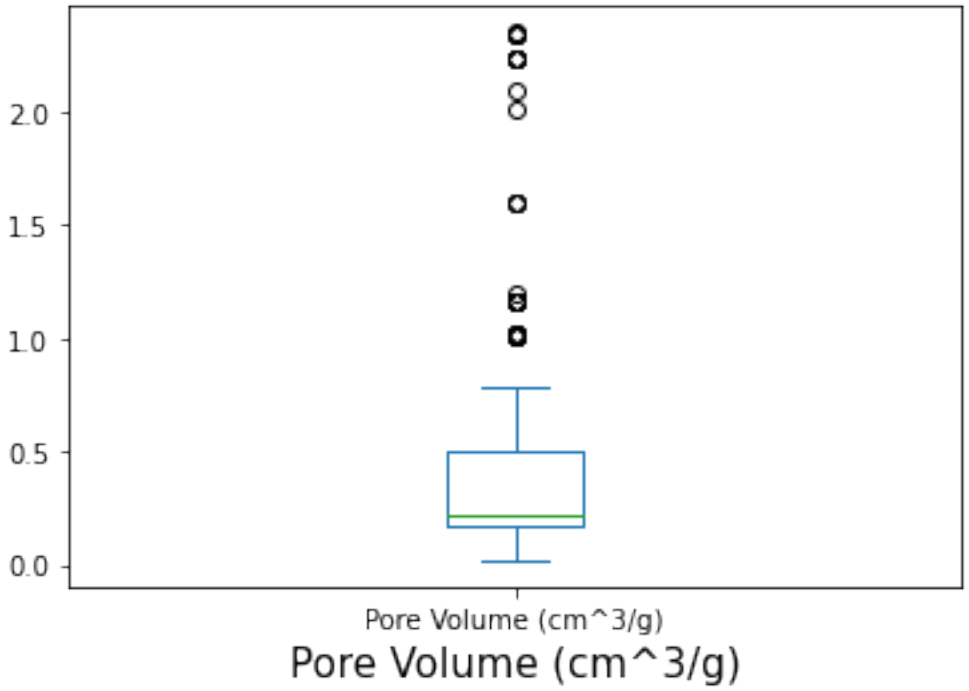
```
std= 2.45   mean= 1.60
count of available values= 151   count of missing values= 432   count of unique
values= 35
available values/total data = 0.259   unique values/available values = 0.232
################################################################################
```






```
std= 8.15   mean= 8.68
count of available values= 153   count of missing values= 430   count of unique
values= 45
available values/total data = 0.262   unique values/available values = 0.294
################################################################################
```

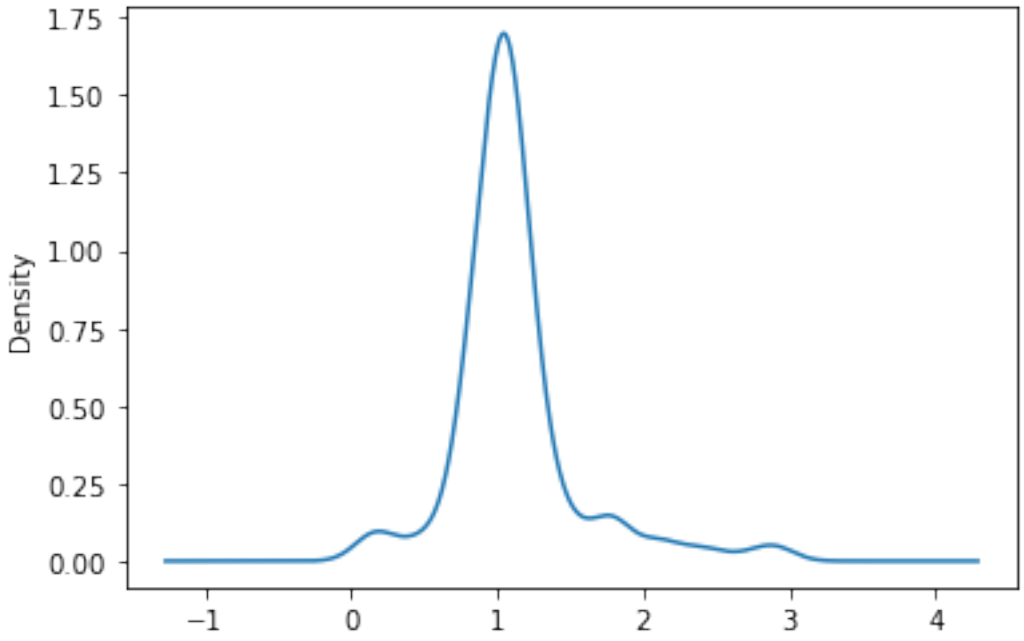


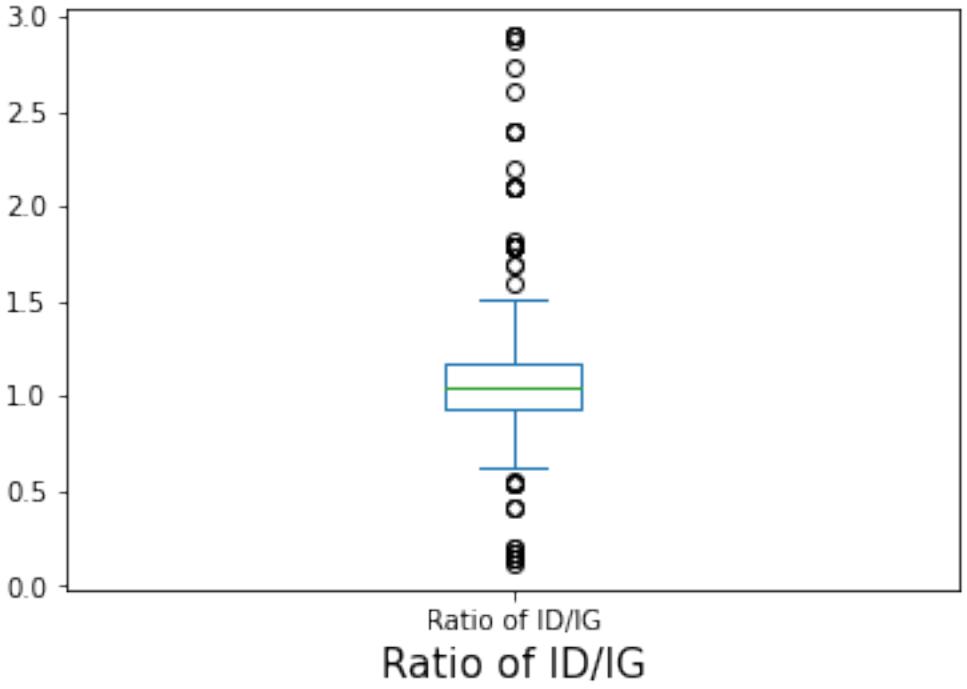


Pore Volume (cm^3/g)

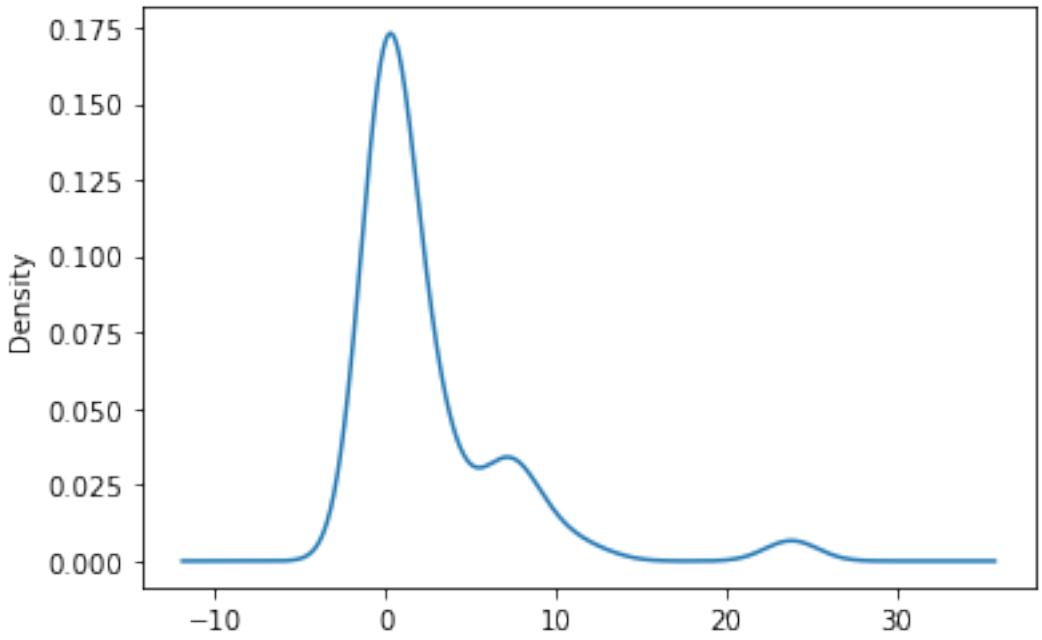
std= 0.59 mean= 0.48
count of available values= 195 count of missing values= 388 count of unique values= 53

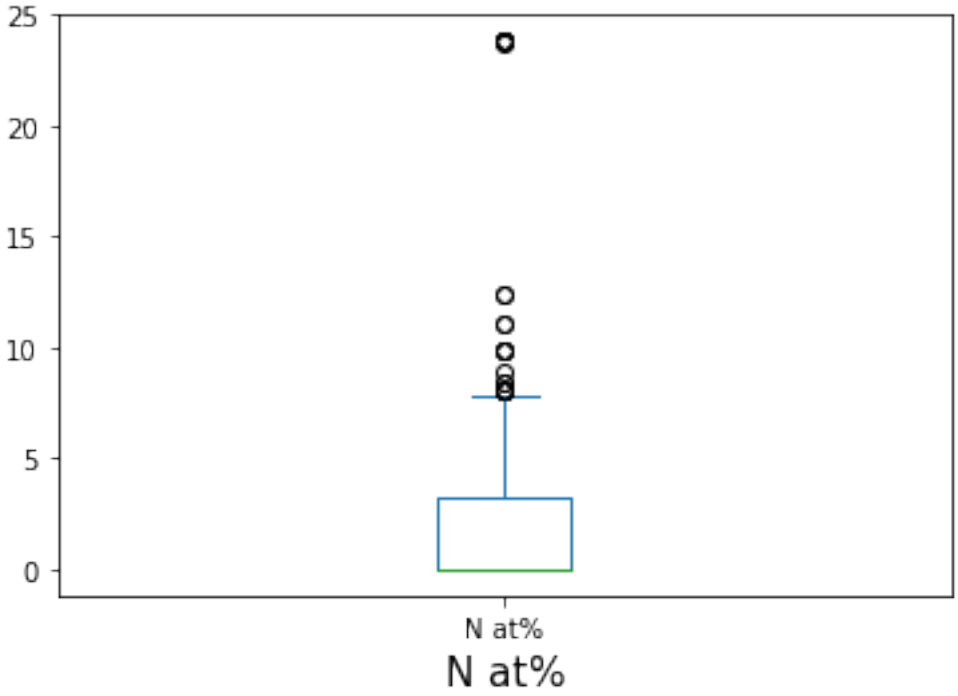
```
available values/total data = 0.334   unique values/available values = 0.272
################################################################################
```



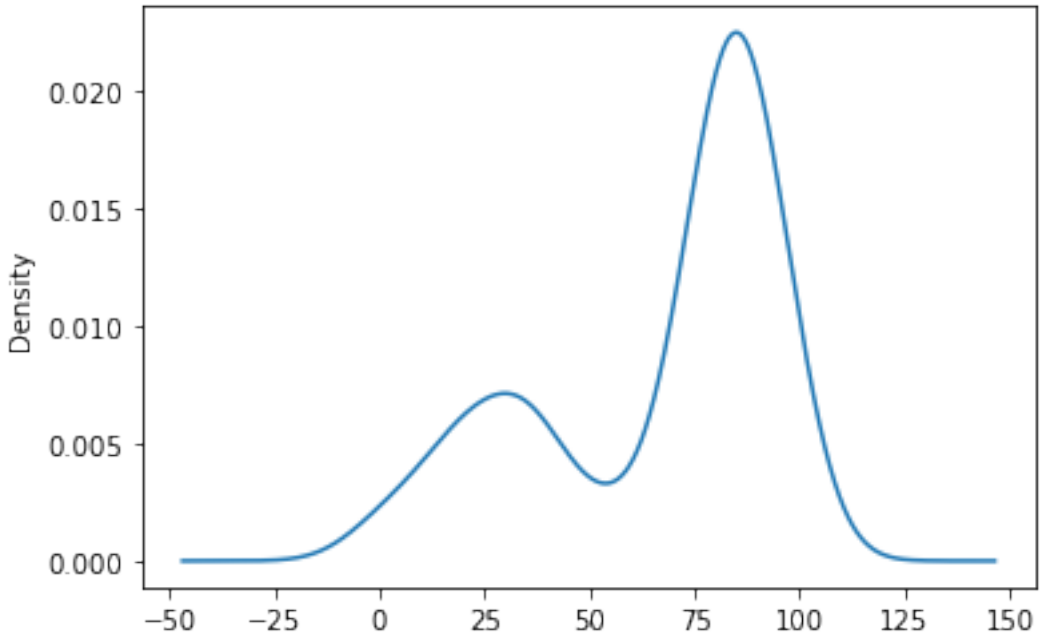


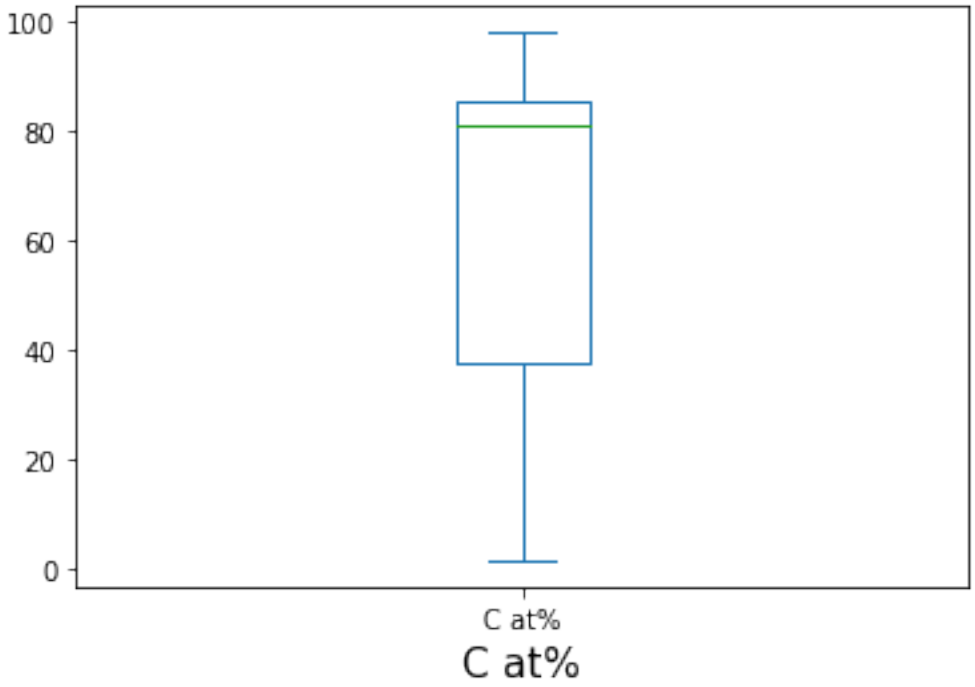
```
std= 0.43   mean= 1.12
count of available values= 328   count of missing values= 255   count of unique
values= 85
available values/total data = 0.563   unique values/available values = 0.259
################################################################################
```





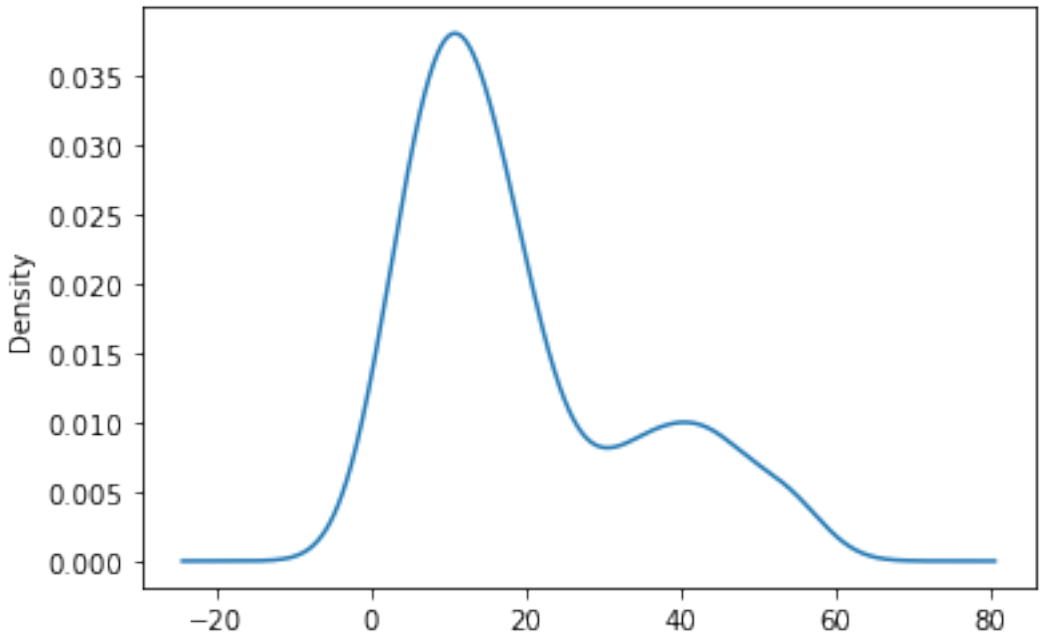

```
std= 4.61   mean= 2.52
count of available values= 231   count of missing values= 352   count of unique
values= 31
available values/total data = 0.396   unique values/available values = 0.134
################################################################################
```

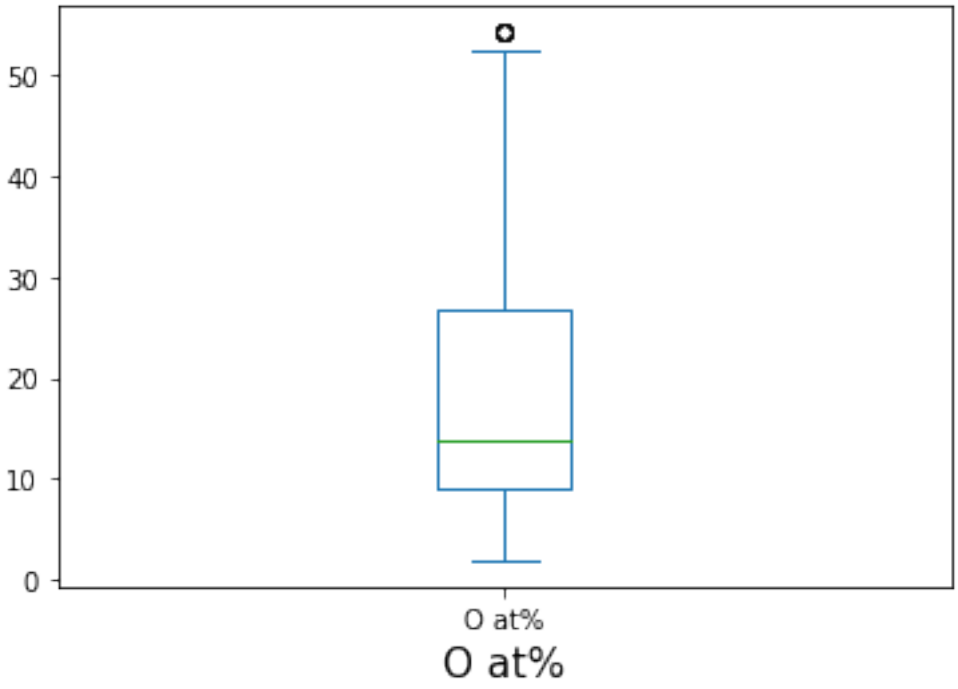




std= 28.53 mean= 66.65
count of available values= 222 count of missing values= 361 count of unique values= 67

```
available values/total data = 0.381   unique values/available values = 0.302
################################################################################
```





```
std= 14.49   mean= 19.17
count of available values= 218   count of missing values= 365   count of unique
values= 68
available values/total data = 0.374   unique values/available values = 0.312
################################################################################
```

[161]:
```
fillingMissingValues(df9)
```

```
column=  Ratio of ID/IG   available ratio= 0.563   unique ratio= 0.259
mean= 1.120   std= 0.433
 missing values filled with 0.9040

################################################################################
```

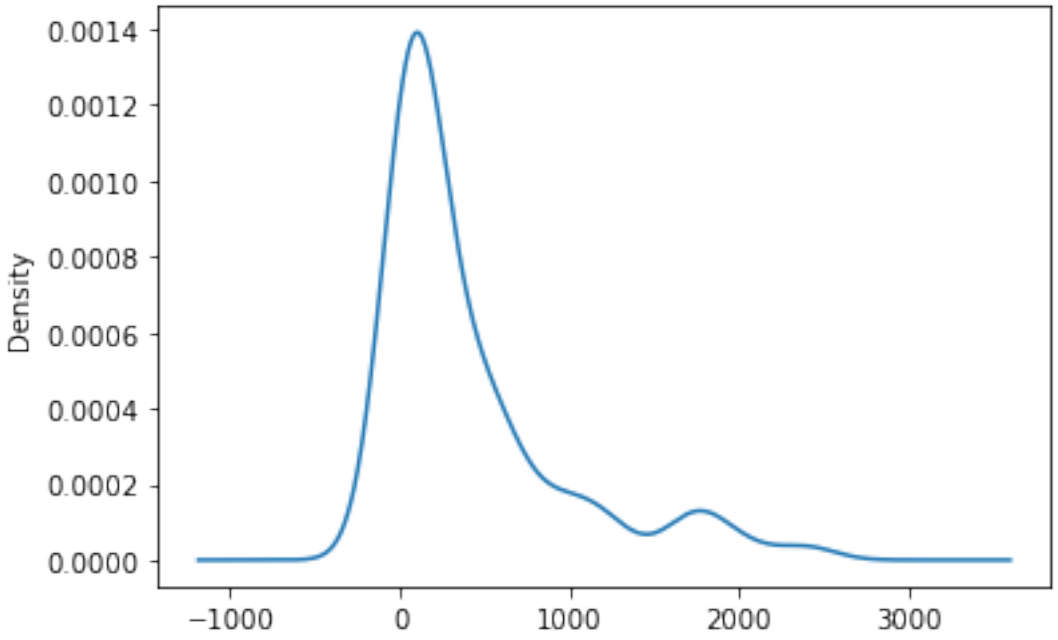
```
/tmp/ipykernel_270313/1691008062.py:66: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[column].fillna((mean-(std/2)), inplace= True)
```

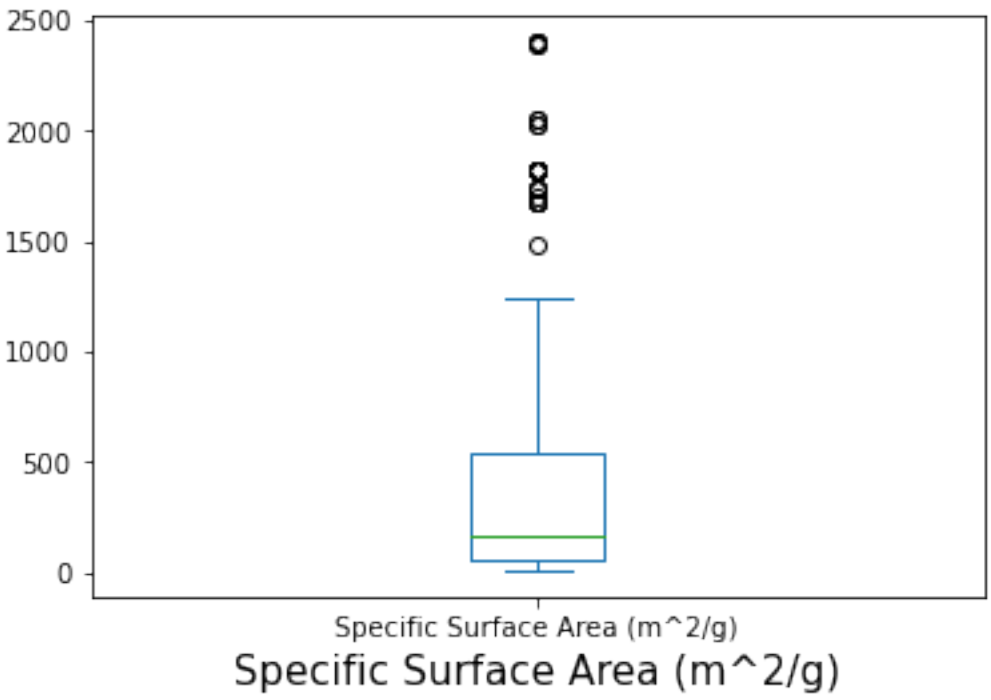
[162]:
```
df9.isnull().sum()
```

[162]:
```
Limits of Potential Window (V)                    0
Lower Limit of Potential Window (V)               0
Upper Limit of Potential Window (V)               0
Potential Window (V)                              0
Current Density (A/g)                             0
Capacitance (F/g)                                 0
Specific Surface Area (m^2/g)                   236
Charge Transfer Resistance (Rct) (ohm)          446
Equivalent Series Resistance (Rs) (ohm)         432
Electrode Material                                0
Pore Size (nm)                                  430
Pore Volume (cm^3/g)                            388
Ratio of ID/IG                                    0
N at%                                           352
C at%                                           361
O at%                                           365
```

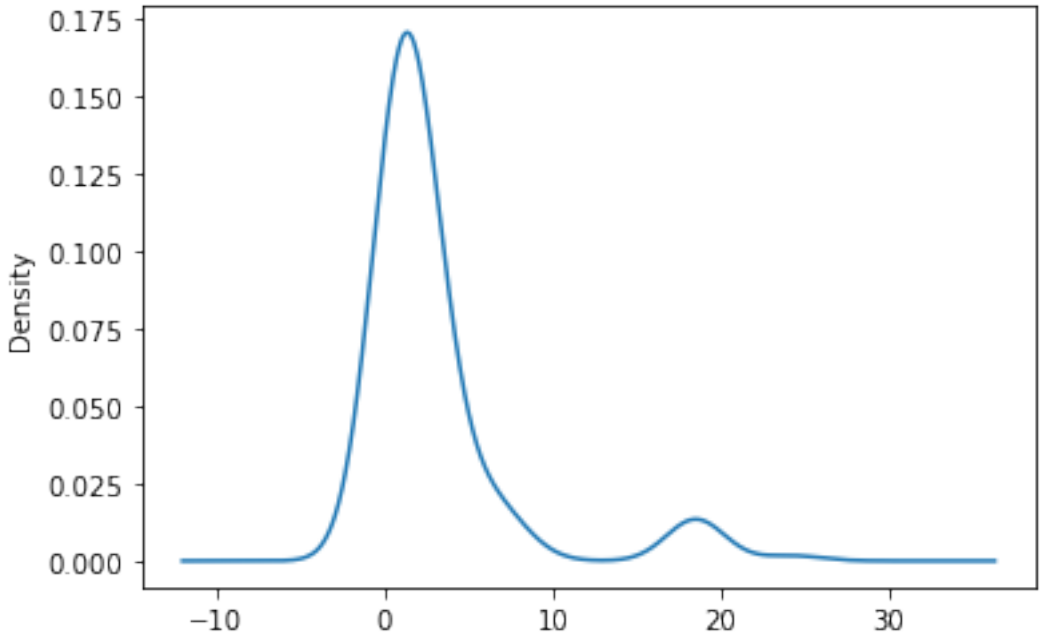
```
Electrolyte Chemical Formula                          0
Electrolyte Ion Mobility Ranking                      0
Electrolyte Concentration (M)                         0
Cell Configuration (three/two electrode system)       0
dtype: int64
```

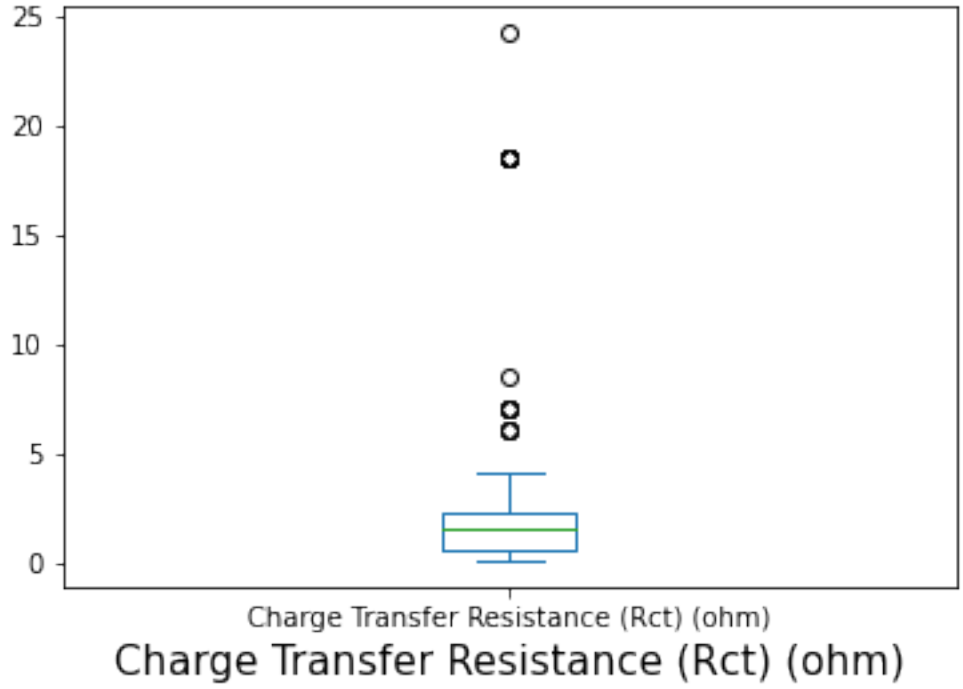
[163]:
```
NullColumnsPlot(df9)
```





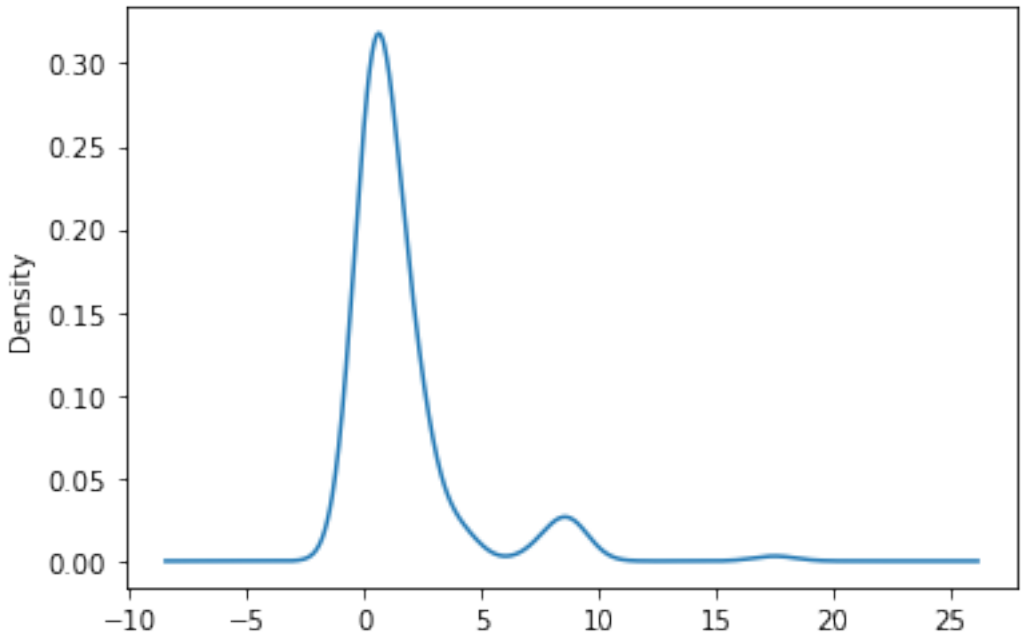

```
std= 547.67   mean= 415.99
count of available values= 347   count of missing values= 236   count of unique
values= 115
available values/total data = 0.595   unique values/available values = 0.331
################################################################################
```

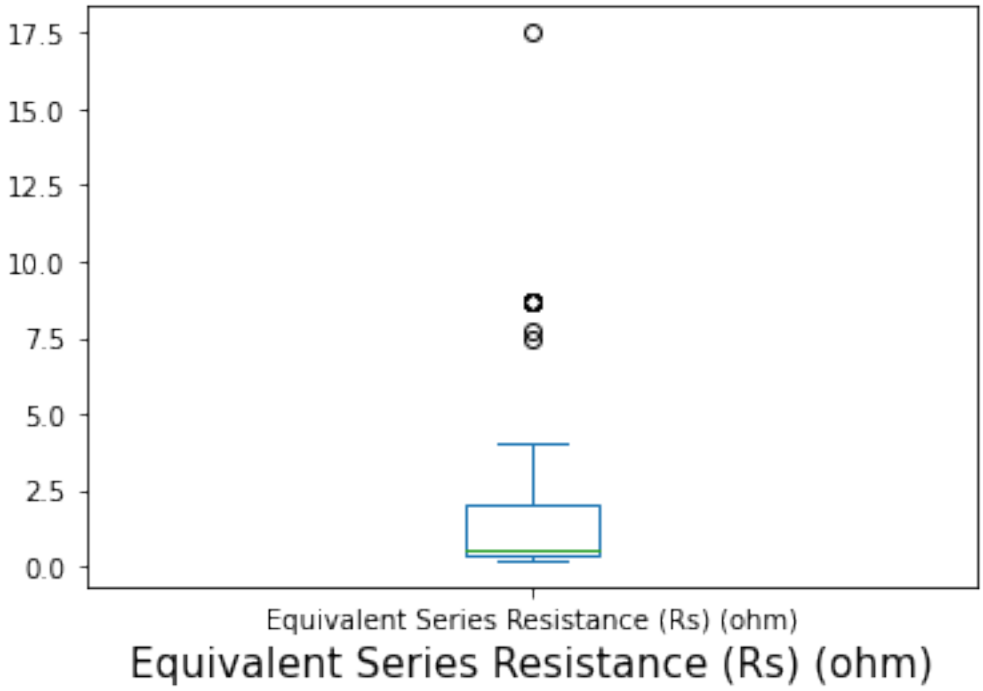





```
std= 4.63   mean= 3.03
count of available values= 137   count of missing values= 446   count of unique
values= 38
```

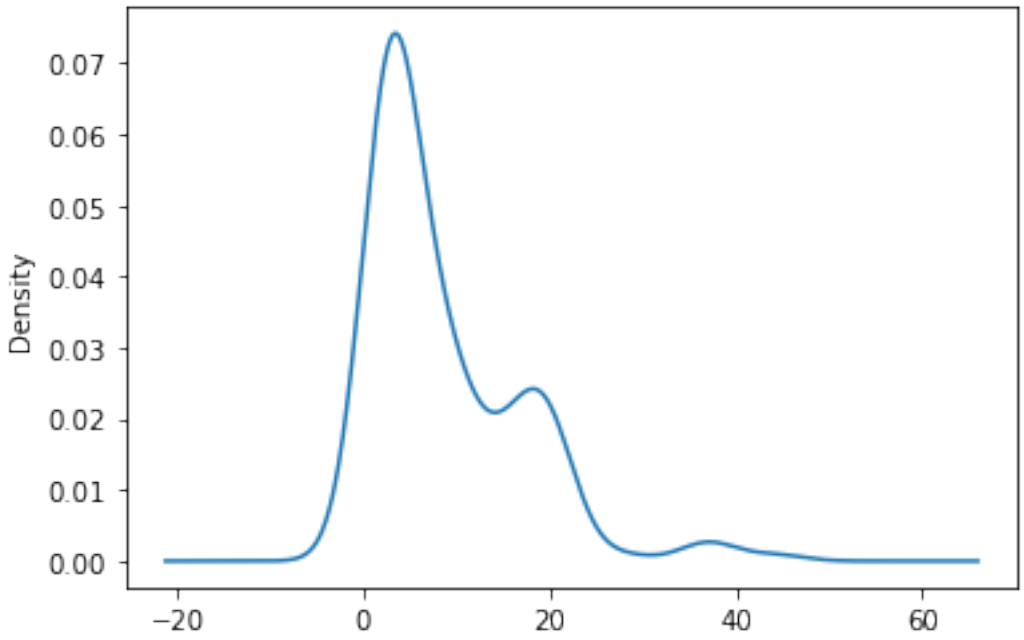
```
available values/total data = 0.235   unique values/available values = 0.277
################################################################################
```

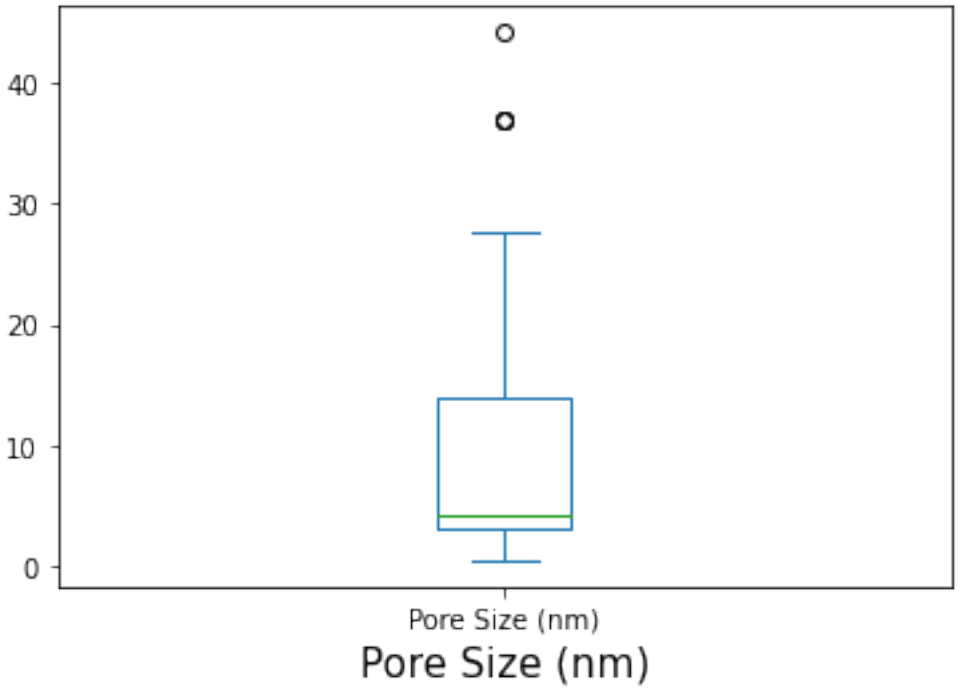




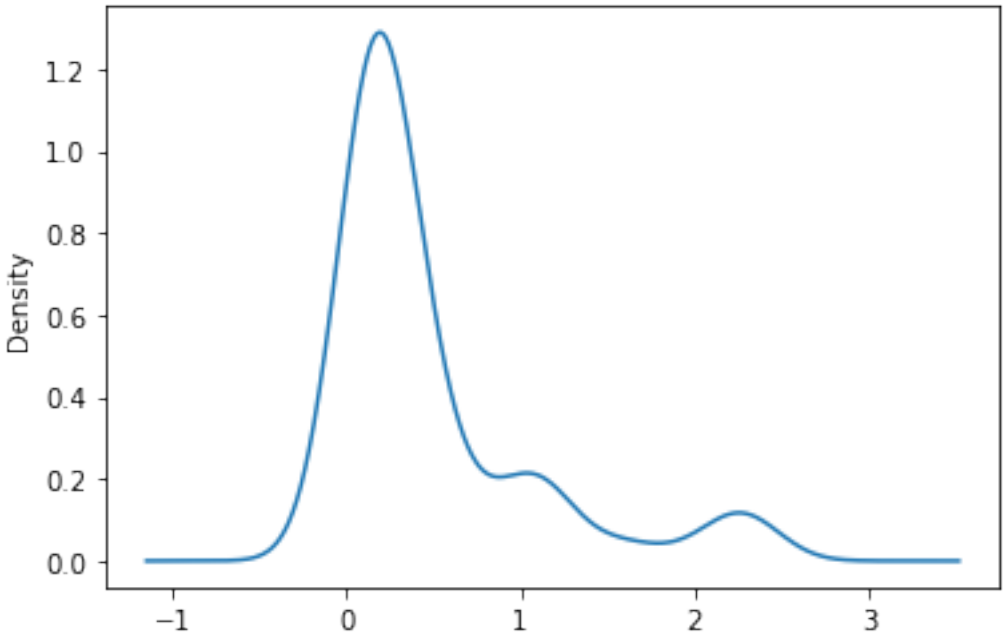
Equivalent Series Resistance (Rs) (ohm)

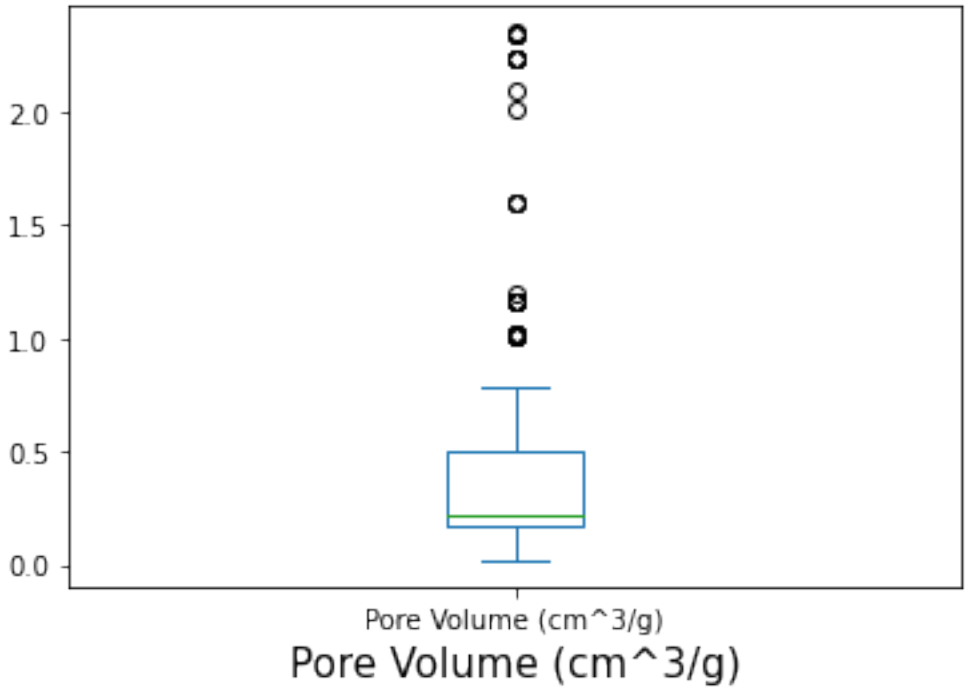
```
std= 2.45   mean= 1.60
count of available values= 151   count of missing values= 432   count of unique
values= 35
available values/total data = 0.259   unique values/available values = 0.232
################################################################################
```






```
std= 8.15   mean= 8.68
count of available values= 153   count of missing values= 430   count of unique
values= 45
available values/total data = 0.262   unique values/available values = 0.294
################################################################################
```

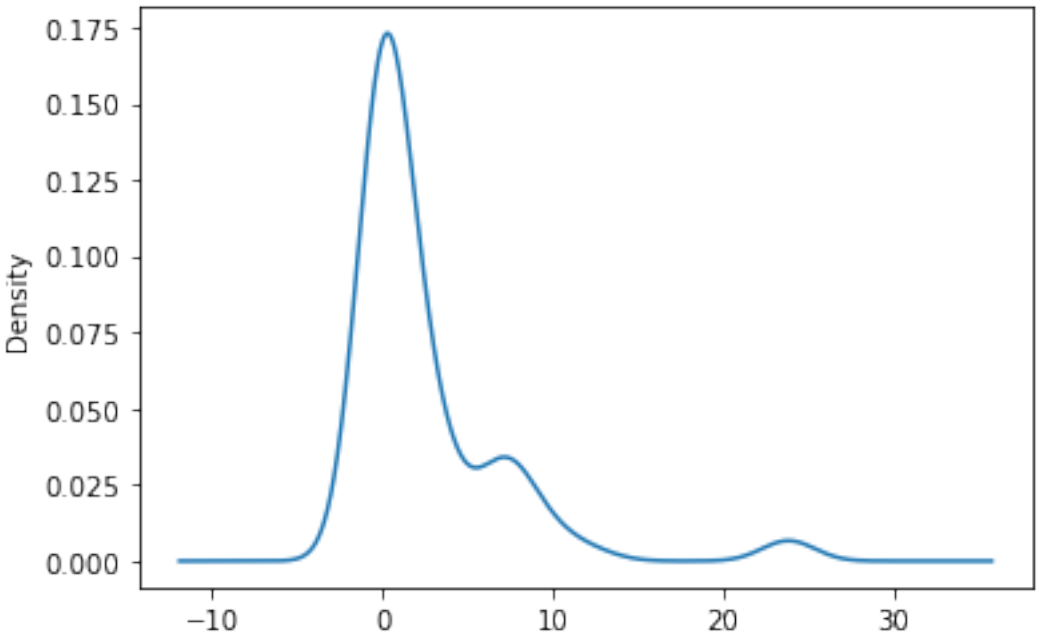


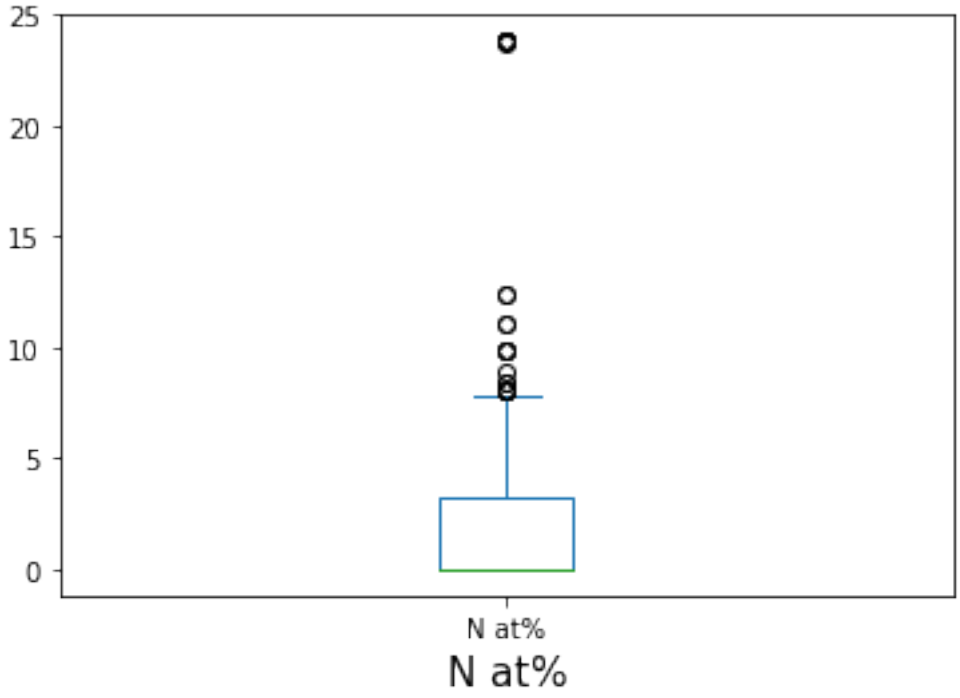


Pore Volume (cm^3/g)

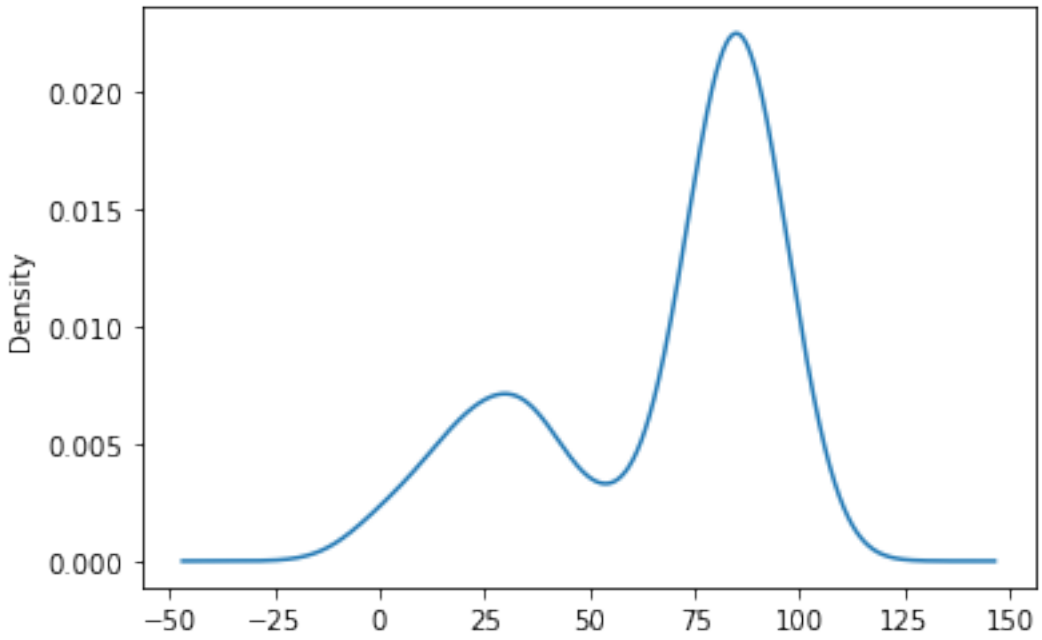
std= 0.59 mean= 0.48
count of available values= 195 count of missing values= 388 count of unique values= 53

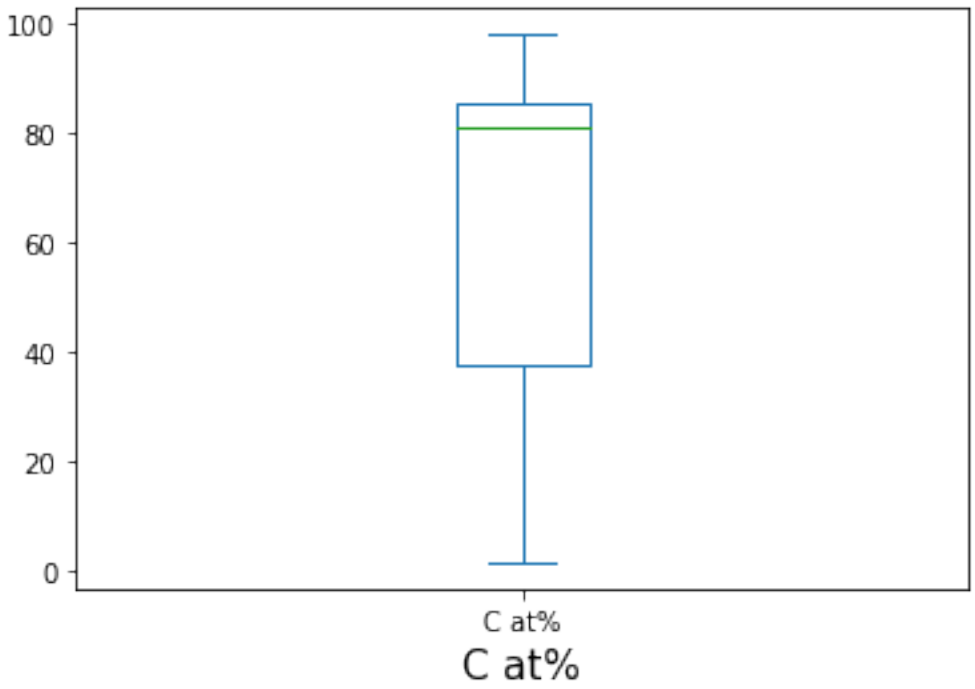
```
available values/total data = 0.334   unique values/available values = 0.272
################################################################################
```



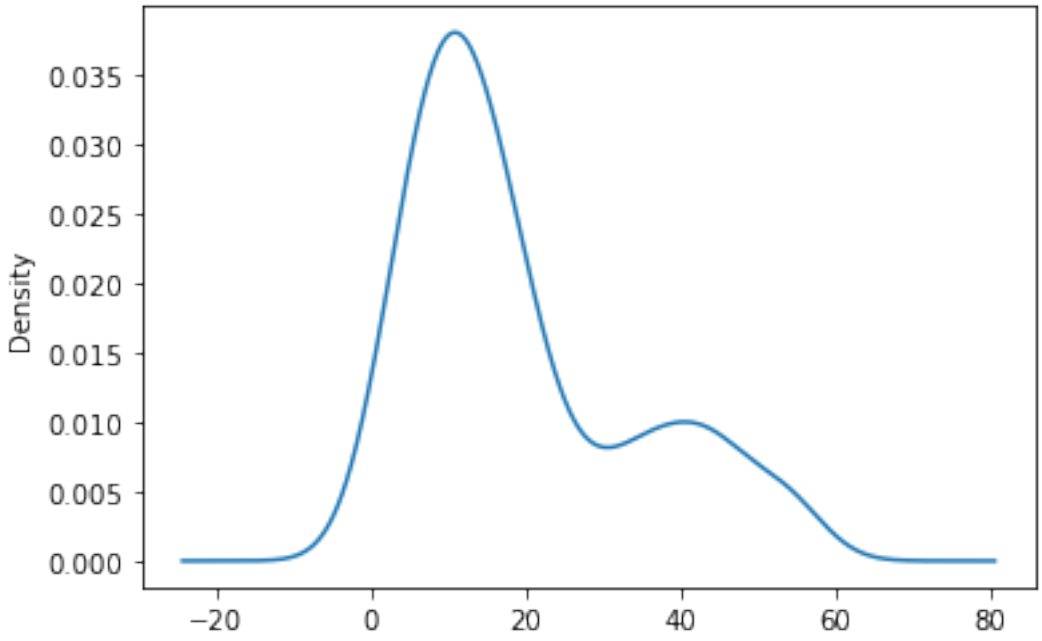


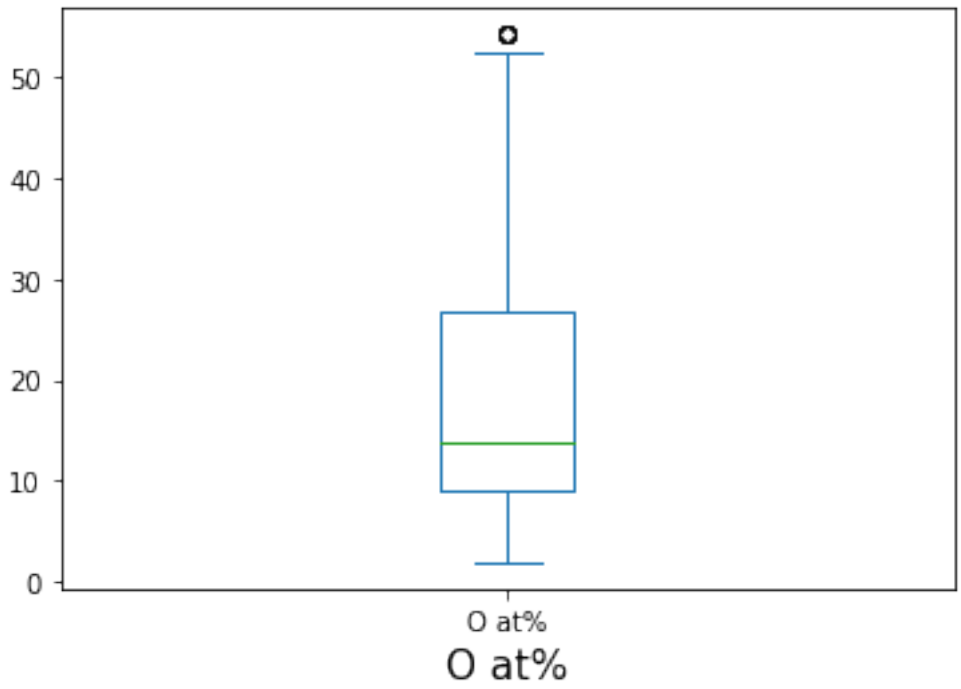
```
std= 4.61   mean= 2.52
count of available values= 231   count of missing values= 352   count of unique
values= 31
available values/total data = 0.396   unique values/available values = 0.134
################################################################################
```






```
std= 28.53   mean= 66.65
count of available values= 222   count of missing values= 361   count of unique
values= 67
available values/total data = 0.381   unique values/available values = 0.302
################################################################################
```





std= 14.49 mean= 19.17
count of available values= 218 count of missing values= 365 count of unique values= 68

```
available values/total data = 0.374   unique values/available values = 0.312
################################################################################
```

[164]:
```
EmptyColumnNames(df9)
```

[164]:
```
['Specific Surface Area (m^2/g)',
 'Charge Transfer Resistance (Rct) (ohm)',
 'Equivalent Series Resistance (Rs) (ohm)',
 'Pore Size (nm)',
 'Pore Volume (cm^3/g)',
 'N at%',
 'C at%',
 'O at%']
```

[165]:
```
df9.shape
```

[165]:
```
(583, 20)
```

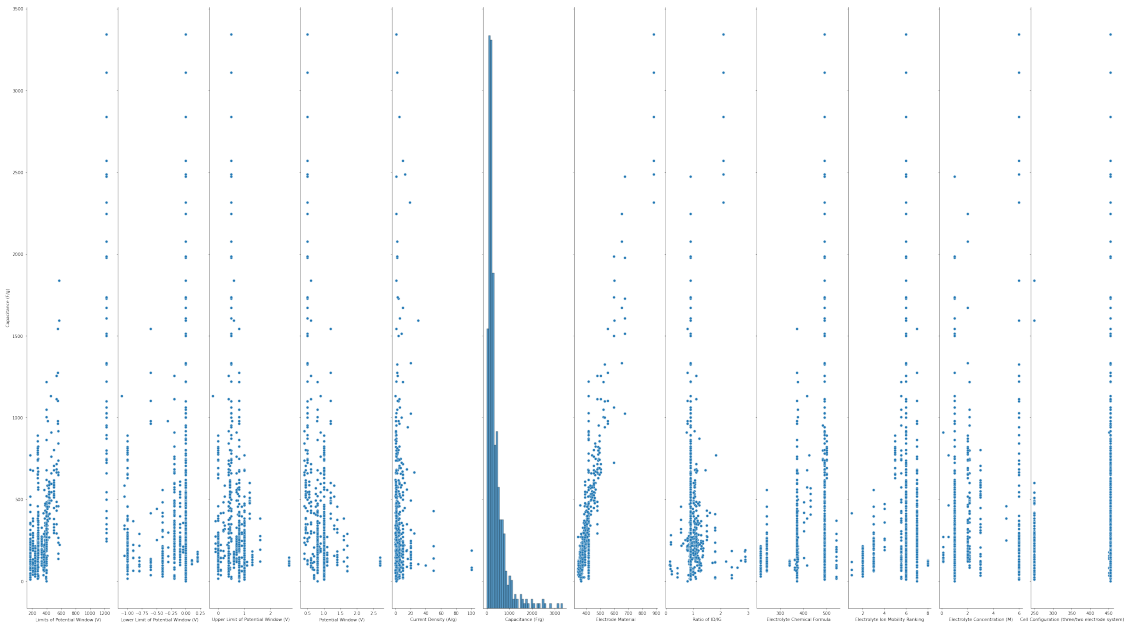
[166]:
```
df=df9.drop(columns=EmptyColumnNames(df9),axis=1)
```

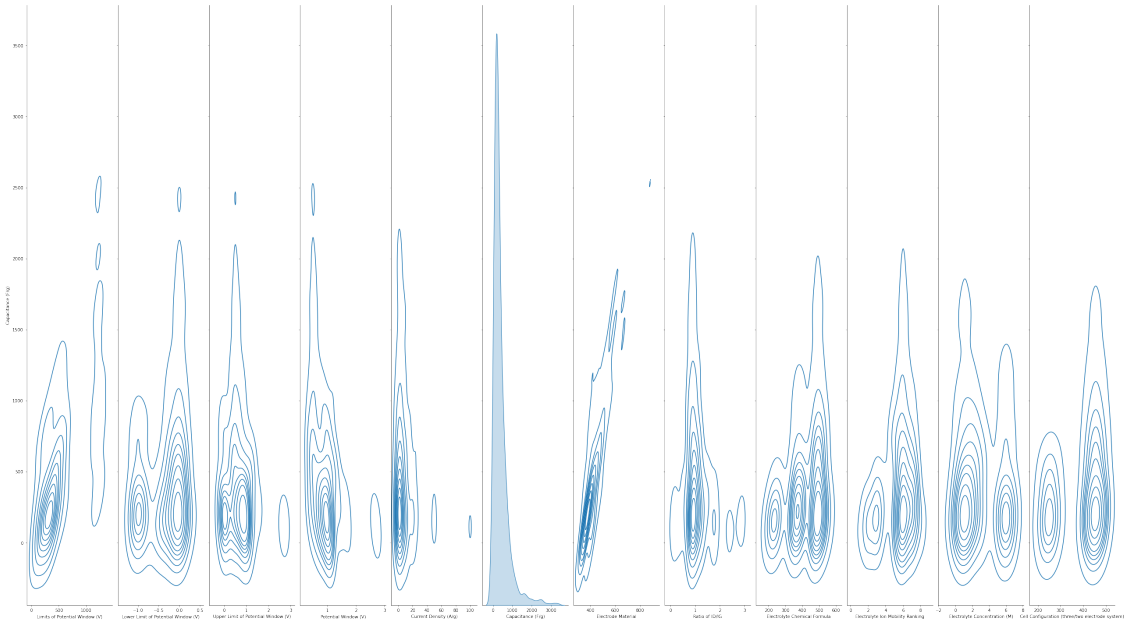
[167]:
```
df.shape
```

[167]:
```
(583, 12)
```

[168]:
```
sns.pairplot(df,y_vars='Capacitance (F/g)',aspect=3/20,height=20)

sns.pairplot(df,kind='kde',y_vars='Capacitance (F/g)',aspect=3/20,height=20)
```

[168]:
```
<seaborn.axisgrid.PairGrid at 0x7f79f814a9a0>
```





[169]:
```
df.shape
```

[169]: (583, 12)

[170]:
```
df.columns
```

[170]:
```
Index(['Limits of Potential Window (V)', 'Lower Limit of Potential Window (V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)', 'Capacitance (F/g)', 'Electrode Material',
       'Ratio of ID/IG', 'Electrolyte Chemical Formula',
       'Electrolyte Ion Mobility Ranking', 'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)'],
      dtype='object')
```

[171]:
```
df.isnull().sum()
```

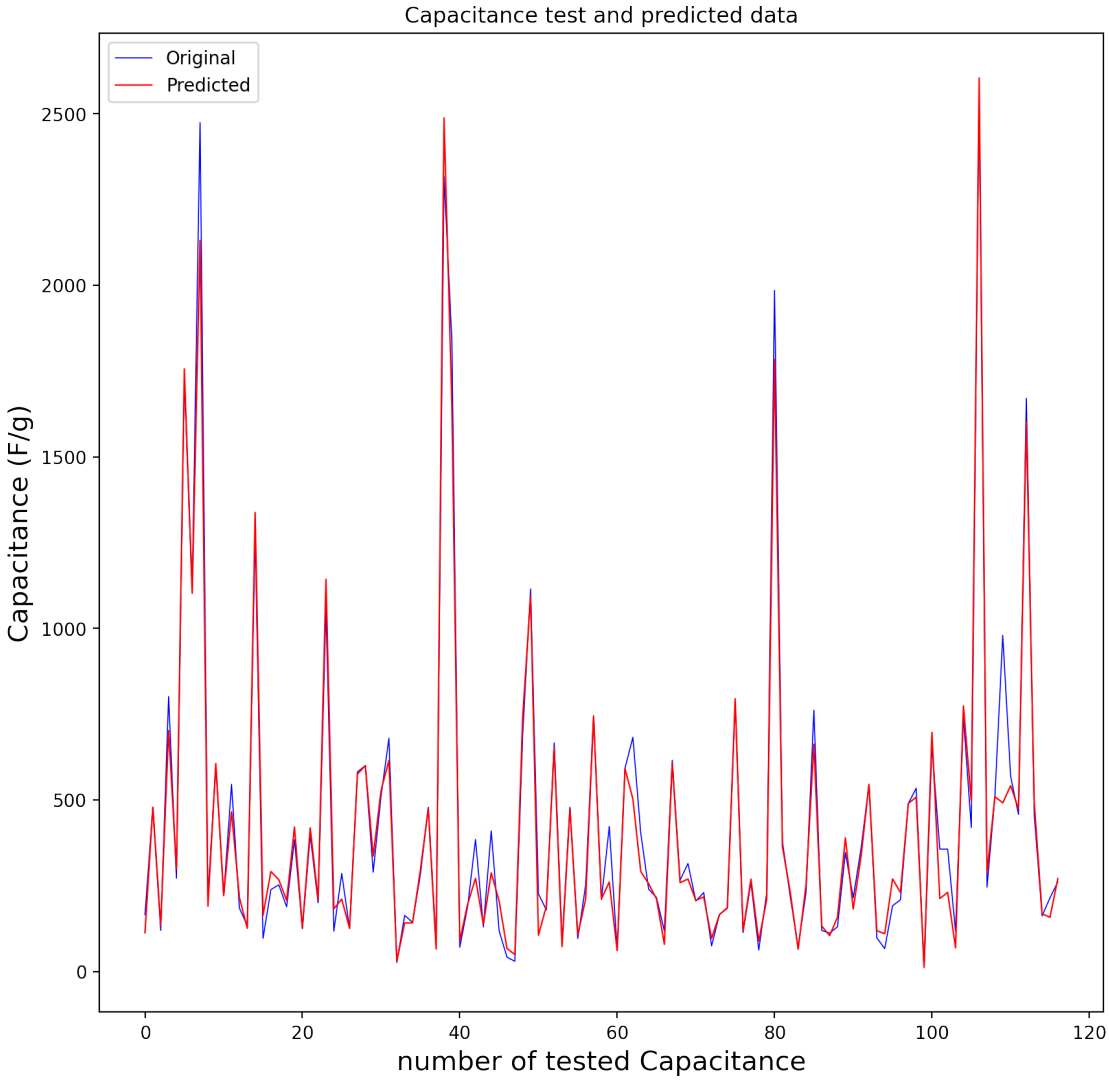
[171]:
```
Limits of Potential Window (V)                    0
Lower Limit of Potential Window (V)               0
Upper Limit of Potential Window (V)               0
Potential Window (V)                              0
Current Density (A/g)                             0
Capacitance (F/g)                                 0
Electrode Material                                0
Ratio of ID/IG                                    0
Electrolyte Chemical Formula                      0
Electrolyte Ion Mobility Ranking                  0
Electrolyte Concentration (M)                     0
Cell Configuration (three/two electrode system)   0
dtype: int64
```

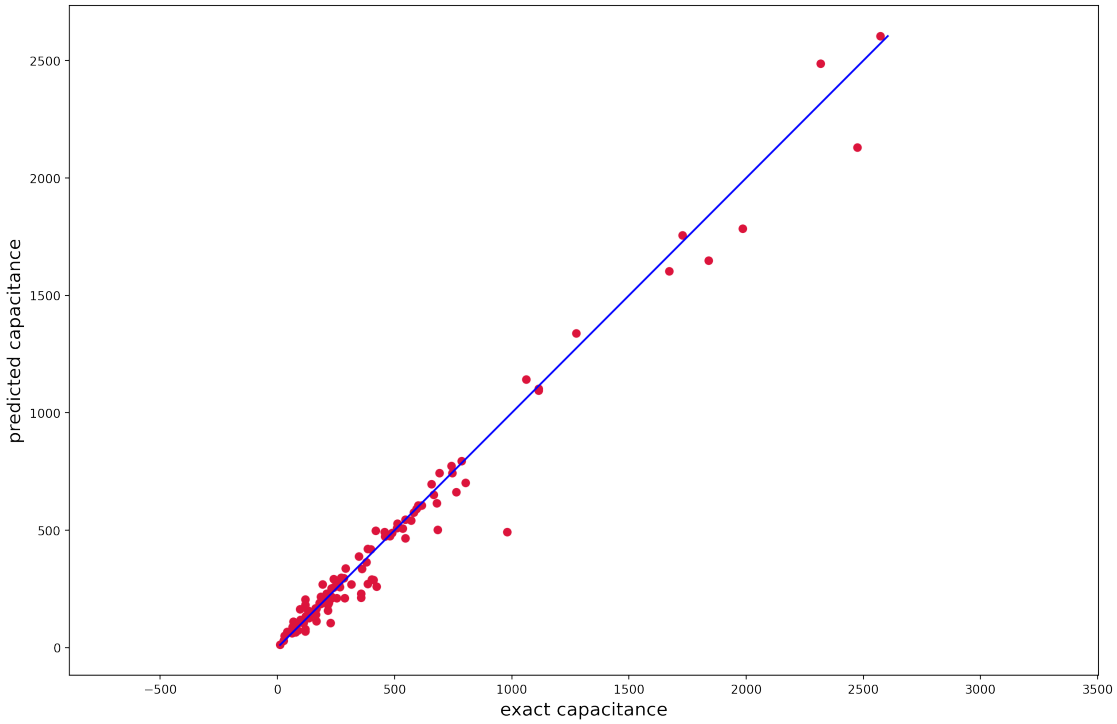
[172]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window␣
 ↪(V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)', 'Electrode Material',
       'Ratio of ID/IG', 'Electrolyte Chemical Formula',
       'Electrolyte Ion Mobility Ranking', 'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']



X=df[Predictors].values
y=df[TargetVariable]




PredictorScaler=StandardScaler()


PredictorScalerFit=PredictorScaler.fit(X)
```

```
X=PredictorScalerFit.transform(X)
y=y.values.ravel()









opted=ExtraTreesRegressionScore()
```

[173]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],␣
 ↪random_state=opted[2])
etr=ExtraTreesRegressor(n_jobs=-1)
etr.fit(X_train,y_train)


Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)


mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)


TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions


from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()
```

```
plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)

print('\n')
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].head(60))
print(2*'\n')
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
print('number of tested data= %i'%count)
print('test_size"percent"= %.2f'%(opted[0]*100))
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
print('optimized MSE: %.4f ' %mse)
print('optimized RMSE: %.4f ' %mse**(0.5))
print('optimized random state: %i'%opted[2] )
```


Capacitance test and predicted data
Original
Predicted
Capacitance (F/g)
number of tested Capacitance
0
500
1000
1500
2000
2500
0
20
40
60
80
100
120



	Capacitance (F/g)	PredictedCapacitance (F/g)
0	166.00	113.3040
1	479.00	479.0000
2	120.10	134.6150
3	801.60	702.4160
4	272.00	298.1160
5	1728.00	1756.4600
6	1115.00	1102.2920
7	2474.00	2131.1191
8	218.00	190.6804
9	600.00	606.3500
10	221.41	221.8280
11	546.00	465.5660
12	185.00	216.0900
13	134.00	126.3440
14	1274.00	1338.0700
15	97.00	164.3340
16	239.00	291.6900
17	252.90	268.0540
18	188.80	207.7050
19	384.60	421.2480
20	125.10	128.6250

21	398.00	418.9976
22	200.60	209.8450
23	1062.00	1143.2300
24	118.00	183.1330
25	286.00	211.2500
26	126.00	126.1010
27	582.00	575.5940
28	600.19	600.1900
29	290.00	337.0300
30	512.30	528.1210
31	680.00	615.5600
32	27.00	30.2950
33	164.00	141.8520
34	143.50	142.0560
35	283.00	296.6575
36	478.93	475.3380
37	78.00	65.7060
38	2316.75	2488.0000
39	1839.40	1649.5130
40	71.00	90.5774
41	189.00	196.3902
42	385.00	272.0938
43	130.00	137.2721
44	410.00	287.8685
45	119.00	206.1100
46	42.09	67.2430
47	30.20	49.9400
48	692.00	744.3200
49	1115.00	1094.4480
50	226.60	106.0980
51	180.00	190.2330
52	666.60	651.2760
53	86.90	72.7700
54	478.93	475.3380
55	96.40	107.6381
56	252.50	211.2500
57	745.00	744.9000
58	210.00	211.7630
59	423.00	261.2295

```
total number of data= 583
number of trained data= 467
number of tested data= 116
test_size"percent"= 20.00
score for xtest and ytest : 0.9738
cross validation score: 0.9068
```

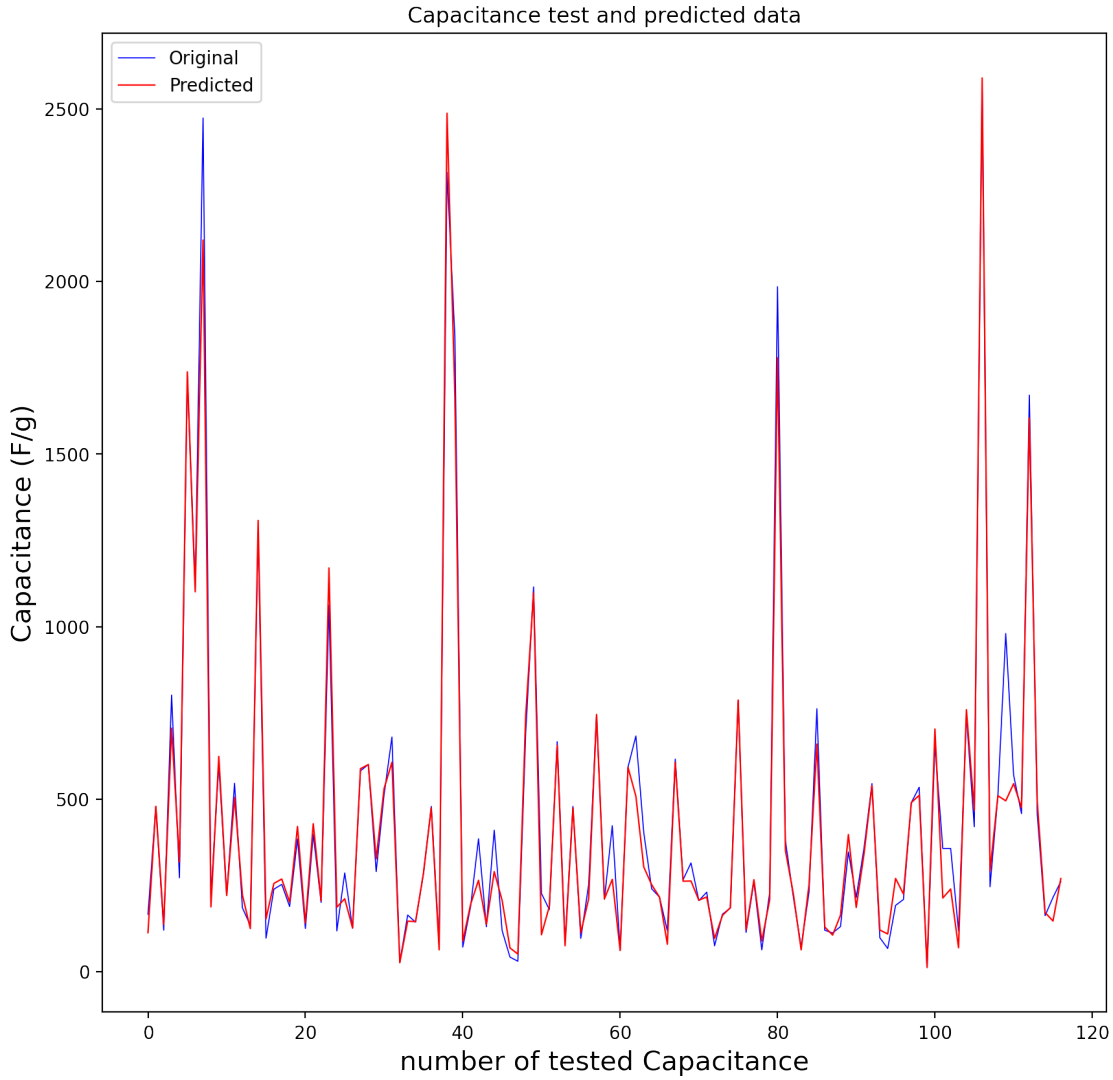
```
optimized MSE: 6338.3397
optimized RMSE: 79.6137
optimized random state: 7
```

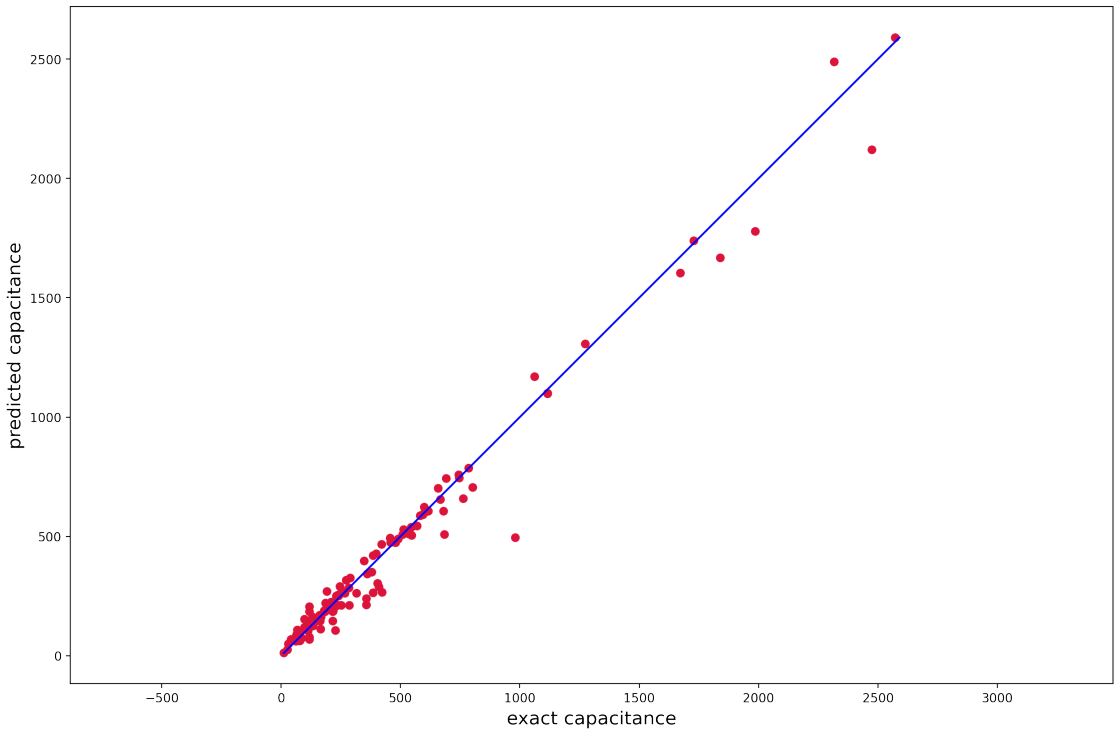
[174]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window (V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)', 'Electrode Material',
       'Ratio of ID/IG', 'Electrolyte Chemical Formula',
       'Electrolyte Ion Mobility Ranking', 'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']



X=df[Predictors].values
y=df[TargetVariable]




PredictorScaler=StandardScaler()
PredictorScalerFit=PredictorScaler.fit(X)


X=PredictorScalerFit.transform(X)
y=y.values.ravel()









opted=ExtraTreesRegressionMse()
```

[175]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],random_state=opted[2])
etr=ExtraTreesRegressor(n_jobs=-1)
etr.fit(X_train,y_train)


Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)
```

```
mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)


TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions


from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()




plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)

print('\n')
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].head(60))
print(2*'\n')
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
print('number of tested data= %i'%count)
print('test_size"percent"= %.2f'%(opted[0]*100))
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
```

```
print('optimized MSE: %.4f ' %mse)
print('optimized RMSE: %.4f ' %mse**(0.5))
print('optimized random state: %i'%opted[2] )
```





	Capacitance (F/g)	PredictedCapacitance (F/g)
0	166.00	113.017000
1	479.00	479.000000
2	120.10	136.736000
3	801.60	705.716000
4	272.00	317.489000
5	1728.00	1738.440000
6	1115.00	1100.892000
7	2474.00	2120.373900
8	218.00	187.621800
9	600.00	623.880000
10	221.41	220.830000
11	546.00	505.265000
12	185.00	222.329167
13	134.00	124.830000
14	1274.00	1307.730000
15	97.00	153.254000
16	239.00	255.750000
17	252.90	268.472000
18	188.80	201.596000
19	384.60	420.679000
20	125.10	140.685000

```
21              398.00                 428.782800
22              200.60                 205.263000
23             1062.00                1170.000000
24              118.00                 186.775000
25              286.00                 211.250000
26              126.00                 126.554000
27              582.00                 588.132000
28              600.19                 600.190000
29              290.00                 327.030000
30              512.30                 530.311000
31              680.00                 606.820000
32               27.00                  26.010000
33              164.00                 146.335000
34              143.50                 145.056000
35              283.00                 284.600000
36              478.93                 474.604000
37               78.00                  63.280000
38             2316.75                2488.000000
39             1839.40                1668.768200
40               71.00                  88.412000
41              189.00                 195.262800
42              385.00                 264.838367
43              130.00                 137.022300
44              410.00                 289.815000
45              119.00                 206.710000
46               42.09                  68.722000
47               30.20                  50.990000
48              692.00                 743.790000
49             1115.00                1097.972000
50              226.60                 106.979000
51              180.00                 188.963000
52              666.60                 655.946000
53               86.90                  75.110000
54              478.93                 474.604000
55               96.40                 112.788300
56              252.50                 211.250000
57              745.00                 745.000000
58              210.00                 210.816000
59              423.00                 267.450000



total number of data= 583
number of trained data= 467
number of tested data= 116
test_size"percent"= 20.00
score for xtest and ytest : 0.9743
cross validation score: 0.9061
```

```
optimized MSE: 6217.6536
optimized RMSE: 78.8521
optimized random state: 7
```

[176]:
```
df.shape
```

[176]:
```
(583, 12)
```

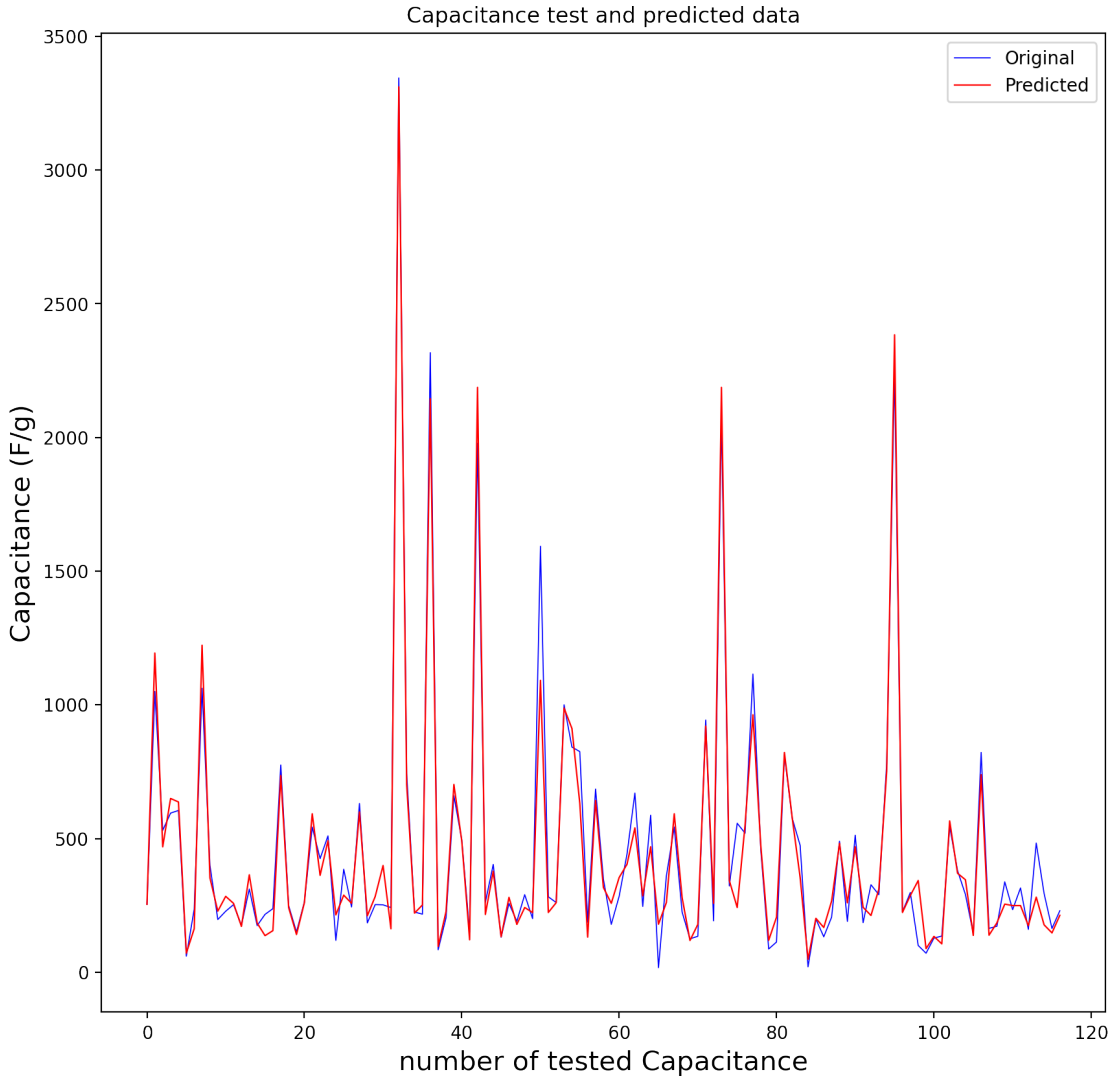
the following model is GBR on the latter df; df6

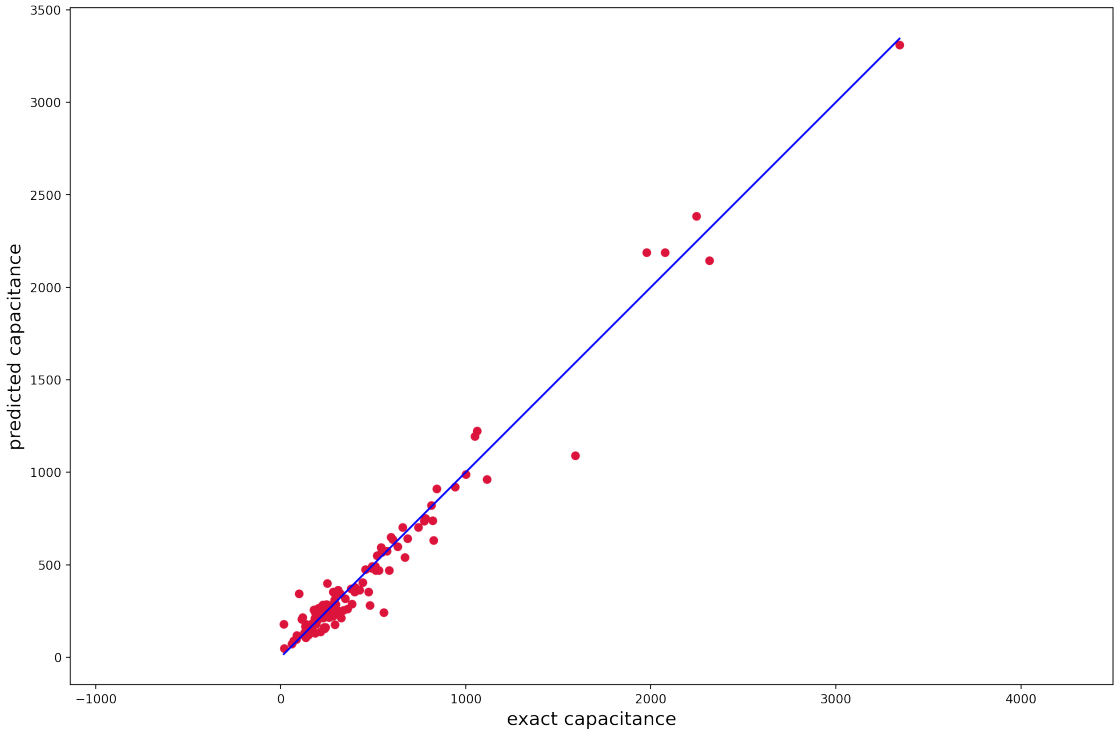
[177]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window␣
 ↪(V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)', 'Electrode Material',
       'Ratio of ID/IG', 'Electrolyte Chemical Formula',
       'Electrolyte Ion Mobility Ranking', 'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']



X=df[Predictors].values
y=df[TargetVariable]




PredictorScaler=StandardScaler()
PredictorScalerFit=PredictorScaler.fit(X)



X=PredictorScalerFit.transform(X)
y=y.values.ravel()







opted=GradientBoostRegScore()
```

[178]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],␣
 ↪random_state=opted[2])
etr=GradientBoostingRegressor(random_state=opted[2])
etr.fit(X_train,y_train)
```

```
Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)


mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)

TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions


from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()




plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)
print('\n')
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].head(60))
print(2*'\n')
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
print('number of tested data= %i'%count)
print('test_size"percent"= %.2f'%(opted[0]*100))
```

```
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
print('optimized MSE: %.4f ' %mse)
print('optimized RMSE: %.4f ' %mse**(0.5))
print('optimized random state: %.4f'%opted[2] )
```





	Capacitance (F/g)	PredictedCapacitance (F/g)
0	253.80	253.949661
1	1050.00	1193.547938
2	530.70	468.958101
3	595.00	649.723330
4	605.00	636.623547
5	60.00	71.129205
6	235.50	162.523718
7	1062.00	1222.682204
8	400.00	352.649867
9	197.10	228.450772
10	228.00	283.720451
11	252.80	257.504626
12	175.30	171.403088
13	311.00	364.264258
14	174.40	183.851680
15	216.30	136.785691
16	237.60	155.478692
17	775.00	735.364173
18	249.00	241.802130
19	150.70	141.042981
20	262.50	259.684268

21	542.73	592.482095
22	425.00	362.372026
23	509.85	491.523414
24	119.10	214.388853
25	385.00	288.112980
26	244.00	257.563427
27	631.00	599.221310
28	184.40	210.860257
29	252.90	283.070885
30	252.00	399.002874
31	241.80	162.523718
32	3344.08	3310.893114
33	743.00	701.571333
34	225.20	221.070085
35	217.00	251.382618
36	2316.75	2144.450273
37	84.00	96.580061
38	203.00	226.432286
39	660.00	702.011808
40	494.84	491.523414
41	150.00	121.297167
42	1978.00	2186.996074
43	261.40	216.011351
44	403.00	377.570430
45	130.70	132.908254
46	257.00	279.841130
47	190.60	178.858798
48	290.00	241.970959
49	200.60	223.053027
50	1593.00	1091.360608
51	282.00	223.468223
52	260.50	260.002192
53	1000.00	987.722198
54	842.00	911.449010
55	825.00	631.771195
56	187.00	131.332639
57	685.00	641.932672
58	348.00	316.715249
59	179.00	257.568106

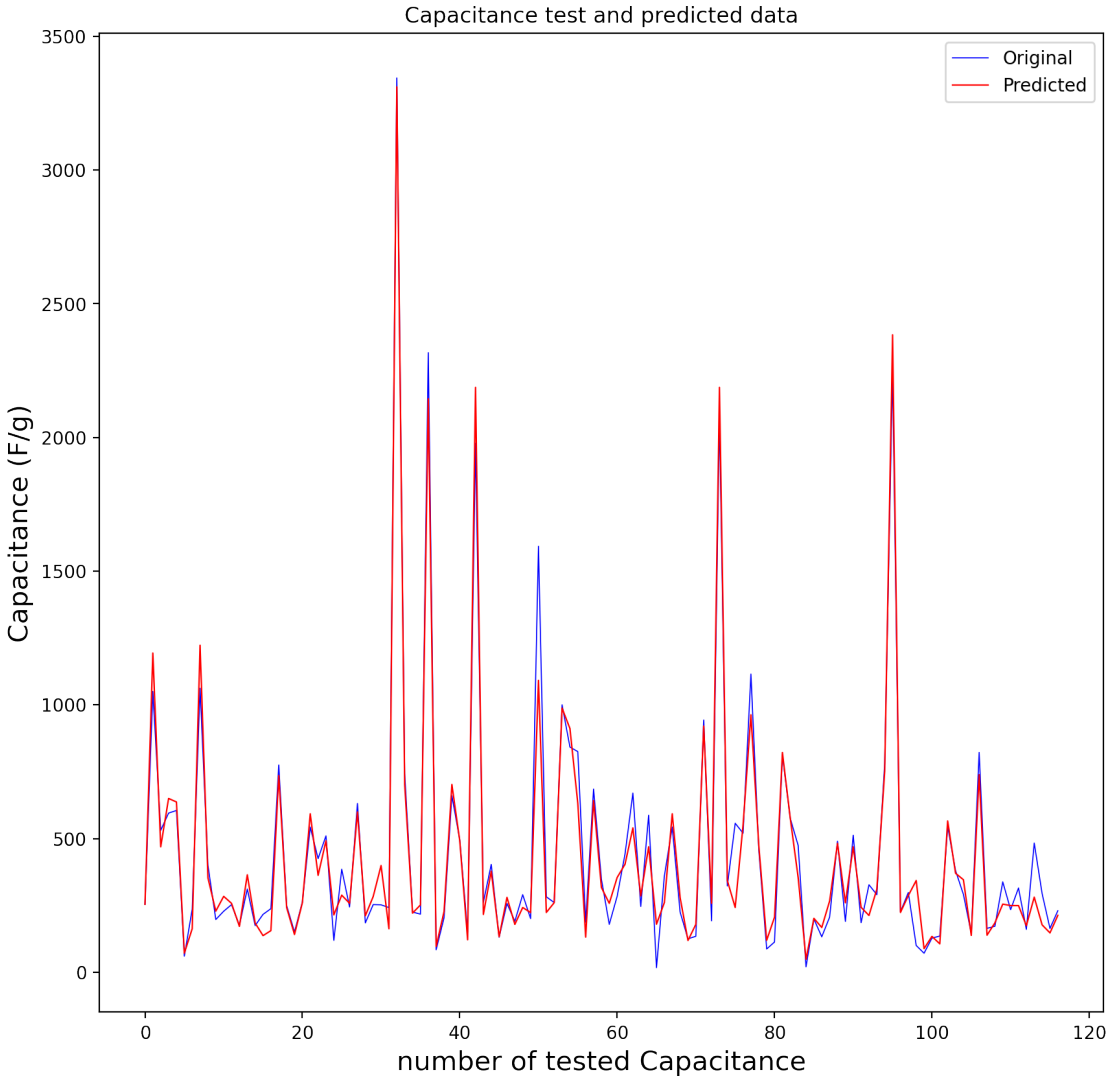
```
total number of data= 583
number of trained data= 467
number of tested data= 116
test_size"percent"= 20.00
score for xtest and ytest : 0.9665
cross validation score: 0.8711
```

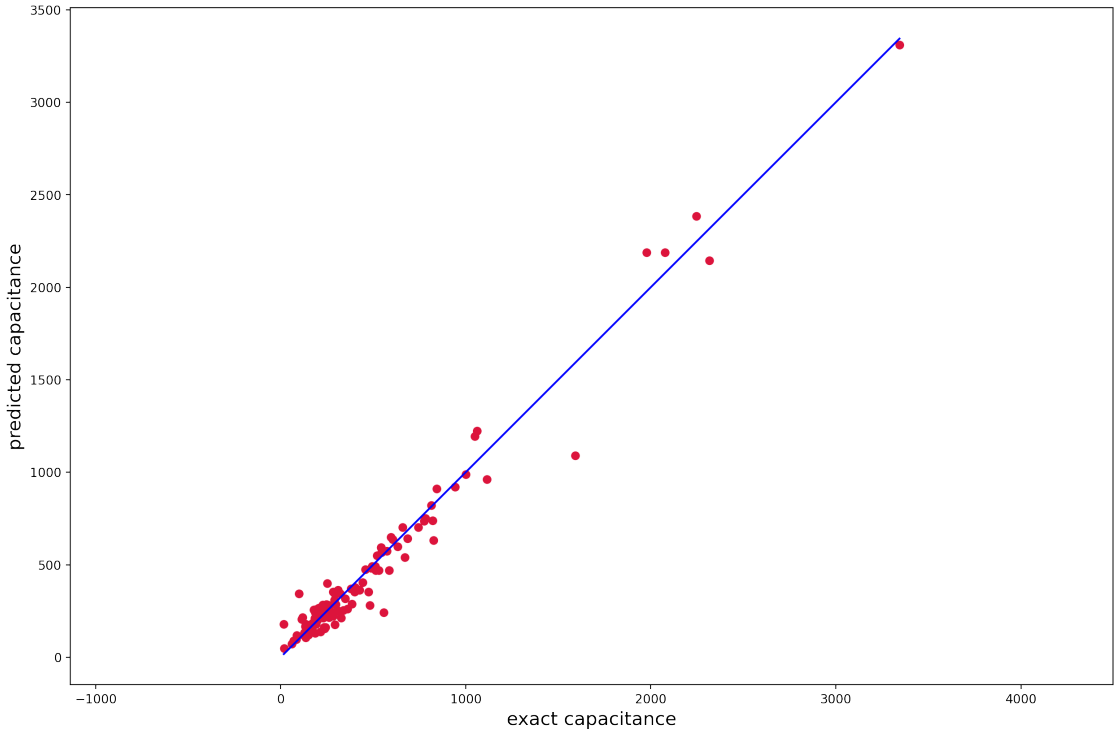
```
optimized MSE: 8300.3769
optimized RMSE: 91.1064
optimized random state: 15.0000
```

[179]:
```
TargetVariable=['Capacitance (F/g)']
Predictors=['Limits of Potential Window (V)', 'Lower Limit of Potential Window␣
 ↪(V)',
       'Upper Limit of Potential Window (V)', 'Potential Window (V)',
       'Current Density (A/g)', 'Electrode Material',
       'Ratio of ID/IG', 'Electrolyte Chemical Formula',
       'Electrolyte Ion Mobility Ranking', 'Electrolyte Concentration (M)',
       'Cell Configuration (three/two electrode system)']



X=df[Predictors].values
y=df[TargetVariable]




PredictorScaler=StandardScaler()
PredictorScalerFit=PredictorScaler.fit(X)


X=PredictorScalerFit.transform(X)
y=y.values.ravel()




opted=GradientBoostRegMse()
```

[180]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=opted[0],␣
 ↪random_state=opted[2])
etr=GradientBoostingRegressor(random_state=opted[2])
etr.fit(X_train,y_train)


Predictions=etr.predict(X_test)
Test_Data=PredictorScalerFit.inverse_transform(X_test)


mse=mean_squared_error(y_test,Predictions)
score=etr.score(X_test,y_test)
cv_score=cross_val_score(etr,X_train,y_train,cv=10,n_jobs=-1)
```

```
TestingData=pd.DataFrame(data=Test_Data, columns=Predictors)
TestingData['Capacitance (F/g)']=y_test
TestingData['PredictedCapacitance (F/g)']=Predictions


from matplotlib.pyplot import figure
figure(figsize=(10,10),dpi=200)
x_ax=range(len(y_test))
plt.plot(x_ax, y_test, lw=0.6, color='b', label='Original')
plt.plot(x_ax,Predictions, lw=0.8, color='r', label='Predicted')
plt.title('Capacitance test and predicted data')
plt.xlabel('number of tested Capacitance',fontsize=15)
plt.ylabel('Capacitance (F/g)',fontsize=15)
plt.legend()
plt.show()




plt.figure(figsize=(15,10),dpi=200)
plt.scatter(y_test,Predictions, c='crimson')
p1=max(max(Predictions),max(y_test))
p2=min(min(Predictions),min(y_test))
plt.plot([p1,p2],[p1,p2],'b-')
plt.xlabel('exact capacitance',fontsize=15)
plt.ylabel('predicted capacitance',fontsize=15)
plt.axis('equal')
plt.show()
count=max(x_ax)
print('\n')
print(TestingData[['Capacitance (F/g)','PredictedCapacitance (F/g)']].head(60))
print(2*'\n')
print('total number of data= %i'%(df.shape[0]))
print('number of trained data= %i'% (df.shape[0]-count))
print('number of tested data= %i'%count)
print('test_size"percent"= %.2f'%(opted[0]*100))
print('score for xtest and ytest : %.4f '%score)
print('cross validation score: %.4f' %cv_score.mean())
print('optimized MSE: %.4f ' %mse)
print('optimized RMSE: %.4f ' %mse**(0.5))
print('optimized random state: %.4f'%opted[2] )
```


Capacitance test and predicted data
Original
Predicted
Capacitance (F/g)
number of tested Capacitance
0
500
1000
1500
2000
2500
3000
3500
0
20
40
60
80
100
120



	Capacitance (F/g)	PredictedCapacitance (F/g)
0	253.80	253.949661
1	1050.00	1193.547938
2	530.70	468.958101
3	595.00	649.723330
4	605.00	636.623547
5	60.00	71.129205
6	235.50	162.523718
7	1062.00	1222.682204
8	400.00	352.649867
9	197.10	228.450772
10	228.00	283.720451
11	252.80	257.504626
12	175.30	171.403088
13	311.00	364.264258
14	174.40	183.851680
15	216.30	136.785691
16	237.60	155.478692
17	775.00	735.364173
18	249.00	241.802130
19	150.70	141.042981
20	262.50	259.684268

```
21              542.73                592.482095
22              425.00                362.372026
23              509.85                491.523414
24              119.10                214.388853
25              385.00                288.112980
26              244.00                257.563427
27              631.00                599.221310
28              184.40                210.860257
29              252.90                283.070885
30              252.00                399.002874
31              241.80                162.523718
32             3344.08               3310.893114
33              743.00                701.571333
34              225.20                221.070085
35              217.00                251.382618
36             2316.75               2144.450273
37               84.00                 96.580061
38              203.00                226.432286
39              660.00                702.011808
40              494.84                491.523414
41              150.00                121.297167
42             1978.00               2186.996074
43              261.40                216.011351
44              403.00                377.570430
45              130.70                132.908254
46              257.00                279.841130
47              190.60                178.858798
48              290.00                241.970959
49              200.60                223.053027
50             1593.00               1091.360608
51              282.00                223.468223
52              260.50                260.002192
53             1000.00                987.722198
54              842.00                911.449010
55              825.00                631.771195
56              187.00                131.332639
57              685.00                641.932672
58              348.00                316.715249
59              179.00                257.568106



total number of data= 583
number of trained data= 467
number of tested data= 116
test_size"percent"= 20.00
score for xtest and ytest : 0.9665
cross validation score: 0.8711
```

```
optimized MSE: 8300.3769
optimized RMSE: 91.1064
optimized random state: 15.0000
```

[181]:
```
plt.figure(figsize=(20,17))
heatmap(df.corr(),annot=True,fmt='.1f',linewidths=0.5,cmap='BrBG')
```

[181]: <AxesSubplot:>