Title: 07 Apr 2024 Analysis

Author ~ Tabish Ali Ansari

In [2]:
```
#importing required libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statistics as stats
```

In [3]:
```
#reading the csv file and porting it into a pandas dataframe
data = pd.read_csv(r"D:\Project - Forecasting Power Consumption\04-04-2024 to 18
                   sep = ";",
                   na_values=["NA", "NaN", "Missing", "Null", "NULL"])
#column names contain IOITSecure447> which is not useful at all and hence is dro
#Changing column names as per our preferences
new_column_names = [col.split(">", 1)[-1].strip() for col in data.columns]
data.columns = new_column_names
data.head()
```

Out[3]:

	TIME	R Ph Voltage	Y Ph Voltage	B Ph Voltage	Average Phase Voltage	RY Line Voltage	YB Line Voltage	BR Line Voltage	R Phase Line current
0	07/04/2024 00:00:01	235.20	236.40	238.05	236.51	409.69	408.17	411.05	35.09
1	07/04/2024 00:00:11	234.92	236.20	237.85	236.35	409.46	407.89	410.84	35.13
2	07/04/2024 00:00:21	234.92	236.20	237.85	236.35	409.46	407.89	410.84	35.13
3	07/04/2024 00:00:31	234.80	235.99	237.71	236.18	409.12	407.61	410.48	35.22
4	07/04/2024 00:00:41	234.80	235.99	237.71	236.18	409.12	407.61	410.48	35.22

5 rows × 30 columns

In [4]:
```
print("The dataset contains (rows, column):", data.shape)
```

```
The dataset contains (rows, column): (8640, 30)
```

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8640 entries, 0 to 8639
Data columns (total 30 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   TIME                      8640 non-null   object 
 1   R Ph Voltage              8640 non-null   float64
 2   Y Ph Voltage              8640 non-null   float64
 3   B Ph Voltage              8640 non-null   float64
 4   Average Phase Voltage     8640 non-null   float64
 5   RY Line Voltage           8640 non-null   float64
 6   YB Line Voltage           8640 non-null   float64
 7   BR Line Voltage           8640 non-null   float64
 8   R Phase Line current      8640 non-null   float64
 9   Y Phase Line current      8640 non-null   float64
 10  B Phase Line current      8640 non-null   float64
 11  Neutral Line current      8640 non-null   float64
 12  R Phase Active Current    8640 non-null   float64
 13  Y Phase Active Current    8640 non-null   float64
 14  B Phase Active Current    8640 non-null   float64
 15  R Phase Reactive Current  8640 non-null   float64
 16  Y Phase Reactive Current  8640 non-null   float64
 17  B Phase Reactive Current  8640 non-null   float64
 18  R- Phase Active Power     8640 non-null   float64
 19  Y- Phase Active Power     8640 non-null   float64
 20  B- Phase Active Power     8640 non-null   float64
 21  3 Phase Active Power      8640 non-null   float64
 22  R- Phase Reactive Power   8640 non-null   float64
 23  Y- Phase Reactive Power   8640 non-null   float64
 24  B- Phase Reactive Power   8640 non-null   float64
 25  3 Phase Reactive Power    8640 non-null   float64
 26  R- Phase Apparent Power   8640 non-null   float64
 27  Y- Phase Apparent Power   8640 non-null   float64
 28  B- Phase Apparent Power   8640 non-null   float64
 29  3 Phase Apparent  Power   8640 non-null   float64
dtypes: float64(29), object(1)
memory usage: 2.0+ MB
```

In [6]:
```
data.describe().T
```

Out[6]:

	count	mean	std	min	25%	50%	75%	max
R Ph Voltage	8640.0	233.803370	3.178845	222.74	231.9575	234.030	235.8900	242.58
Y Ph Voltage	8640.0	232.069130	4.434858	221.16	228.8100	232.840	235.4100	241.88
B Ph Voltage	8640.0	233.211355	3.953158	221.69	230.9900	233.580	235.6600	241.43
Average Phase Voltage	8640.0	233.028390	3.713481	222.04	230.8900	233.325	235.6300	242.00
RY Line Voltage	8640.0	404.218610	6.397829	385.89	400.4800	404.735	408.8700	420.00
YB Line Voltage	8640.0	401.888191	6.479168	383.18	398.0500	402.435	406.3700	417.19
BR Line Voltage	8640.0	404.724051	6.437221	384.70	401.1900	405.300	409.0400	420.41
R Phase Line current	8640.0	26.974626	4.758228	18.00	23.7100	26.140	30.4000	44.62
Y Phase Line current	8640.0	33.126499	11.441572	18.57	21.3000	32.370	44.3900	59.82
B Phase Line current	8640.0	26.372053	6.203605	18.06	22.2000	23.060	29.9025	52.69
Neutral Line current	8640.0	10.956307	5.673764	0.06	6.3700	10.915	15.2800	27.30
R Phase Active Current	8640.0	25.755684	3.965661	17.95	23.3100	25.840	27.4500	43.74
Y Phase Active Current	8640.0	31.950946	11.230858	18.54	21.2800	28.830	43.7800	58.23
B Phase Active Current	8640.0	26.302705	6.164824	18.07	22.1800	23.030	29.8700	52.77
R Phase Reactive Current	8640.0	6.753064	5.085304	0.51	2.8500	4.390	11.0025	18.21
Y Phase Reactive Current	8640.0	7.156251	5.477900	0.07	1.0600	7.390	11.3800	17.41
B Phase Reactive Current	8640.0	-0.220495	1.980352	-2.81	-1.1300	-0.870	-0.4600	11.73

	count	mean	std	min	25%	50%	75%	max
R- Phase Active Power	8640.0	5.946020	0.887718	4.12	5.4600	5.950	6.3200	9.84
Y- Phase Active Power	8640.0	7.332985	2.480592	4.27	4.9700	6.690	9.8000	13.08
B- Phase Active Power	8640.0	6.060556	1.346955	4.19	5.1400	5.390	6.8600	11.67
3 Phase Active Power	8640.0	19.338353	4.082006	13.16	15.6300	18.270	22.9500	31.84
R- Phase Reactive Power	8640.0	1.579657	1.191298	0.12	0.6600	1.040	2.5800	4.28
Y- Phase Reactive Power	8640.0	1.659164	1.282620	0.05	0.2500	1.660	2.5400	4.34
B- Phase Reactive Power	8640.0	-0.053623	0.456576	-0.66	-0.2700	-0.200	-0.1100	2.75
3 Phase Reactive Power	8640.0	3.185850	2.468542	0.00	1.1200	2.240	5.2600	9.62
R- Phase Apparent Power	8640.0	6.302558	1.085053	4.20	5.6200	6.070	7.1200	10.20
Y- Phase Apparent Power	8640.0	7.653997	2.533460	4.32	5.0200	7.560	9.9700	13.47
B- Phase Apparent Power	8640.0	6.130370	1.350076	4.27	5.2100	5.450	6.9000	11.70
3 Phase Apparent Power	8640.0	19.926172	4.176518	13.38	15.8400	19.630	23.2700	49.14

In [7]:
```
data['TIME'] = pd.to_datetime(data['TIME'])  # Convert 'TIME' column to datetime
data.set_index('TIME', inplace=True)  # Set 'TIME' column as index

# Resample the data into one-minute intervals and calculate the mean for each mi
data_minute_avg = data.resample('1Min').mean()

# Drop any NaN values that might have resulted from resampling
data_minute_avg.dropna(inplace=True)

# Display the new DataFrame
data_minute_avg.head()
```

Out[7]:

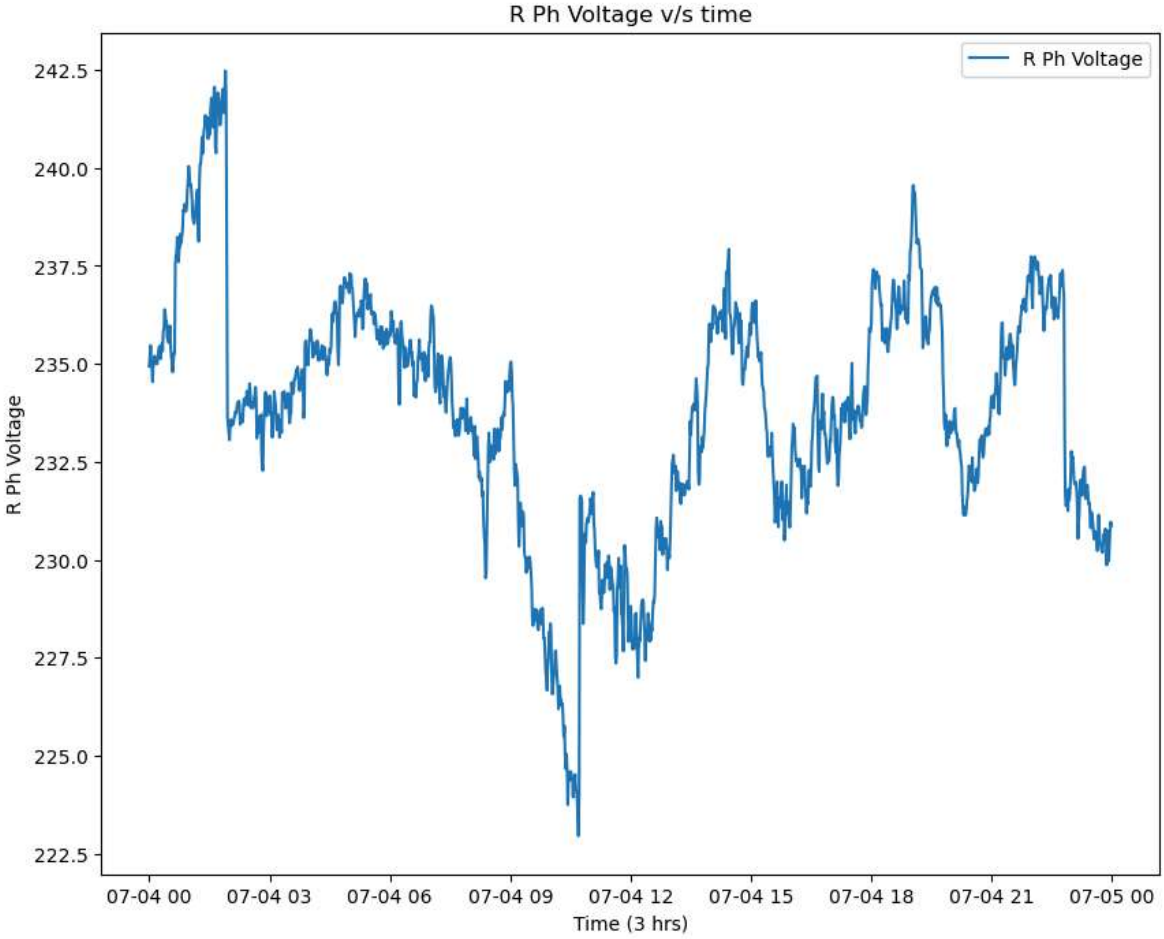
TIME	R Ph Voltage	Y Ph Voltage	B Ph Voltage	Average Phase Voltage	RY Line Voltage	YB Line Voltage	Bl Vc
2024-07-04 00:00:00	234.933333	236.176667	237.870000	236.335000	409.380000	407.878333	410.7
2024-07-04 00:01:00	235.153333	236.486667	238.176667	236.596667	409.703333	408.286667	410.9
2024-07-04 00:02:00	235.466667	237.121667	238.525000	237.081667	410.848333	409.205000	411.8
2024-07-04 00:03:00	235.101667	236.830000	238.266667	236.751667	410.181667	408.826667	411.2
2024-07-04 00:04:00	234.948333	236.295000	237.940000	236.363333	409.643333	408.113333	410.8

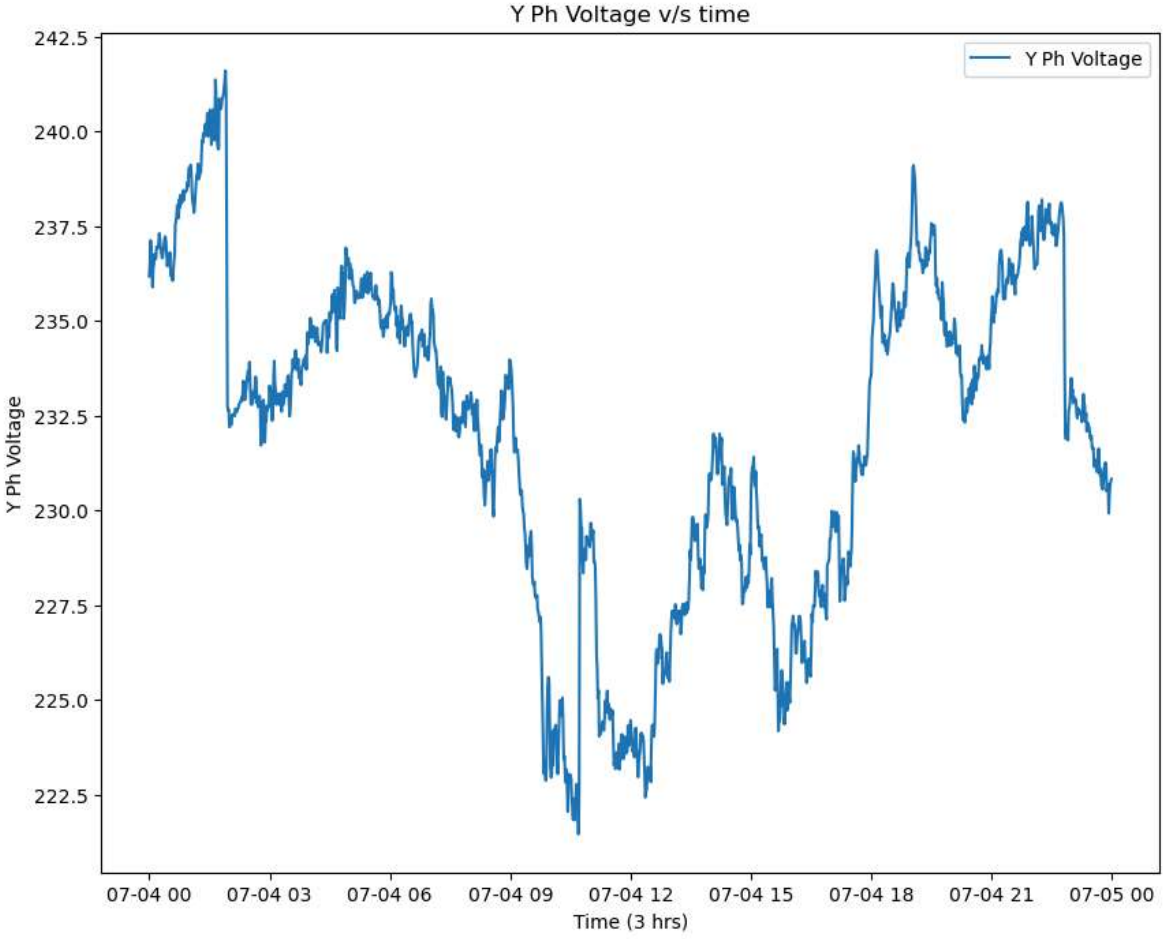
5 rows × 29 columns

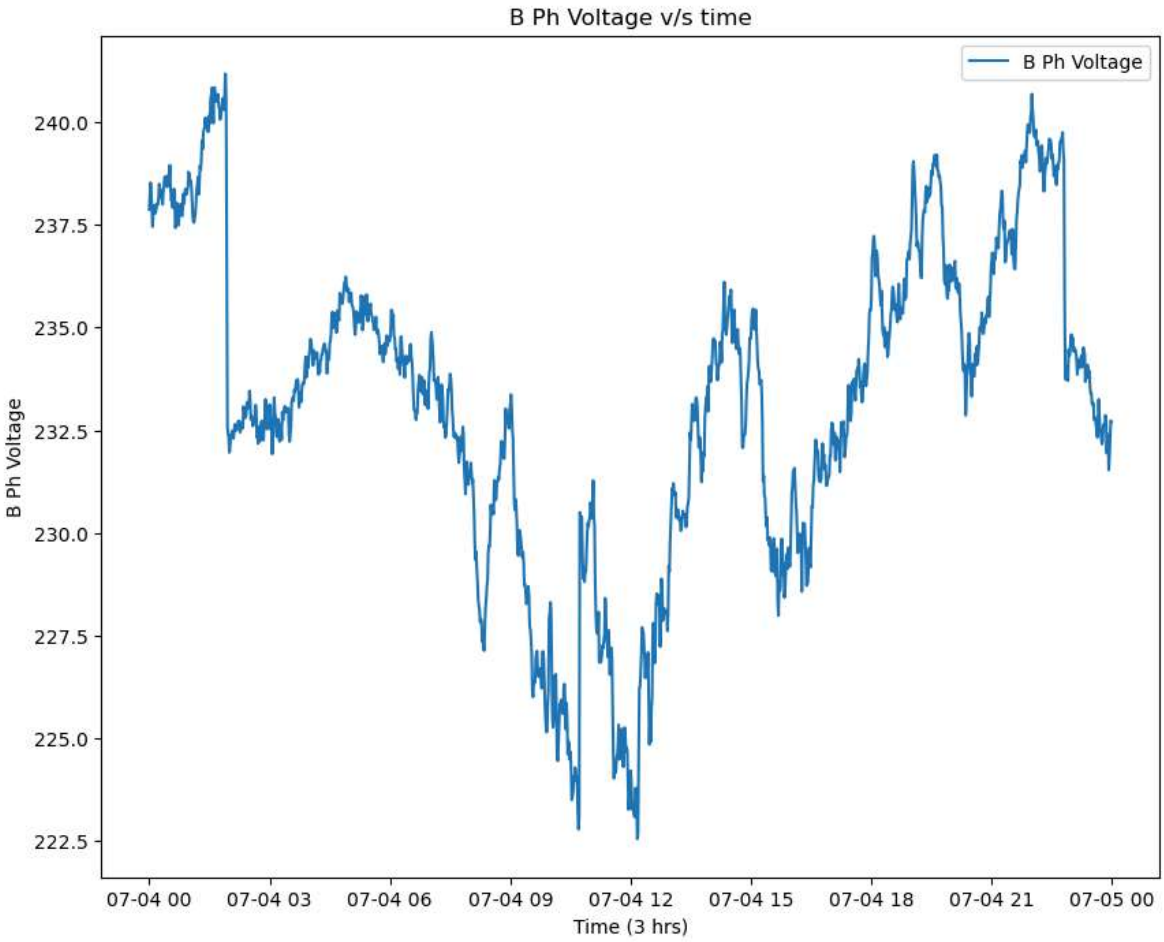
In [8]:
```
data_minute_avg.shape
```

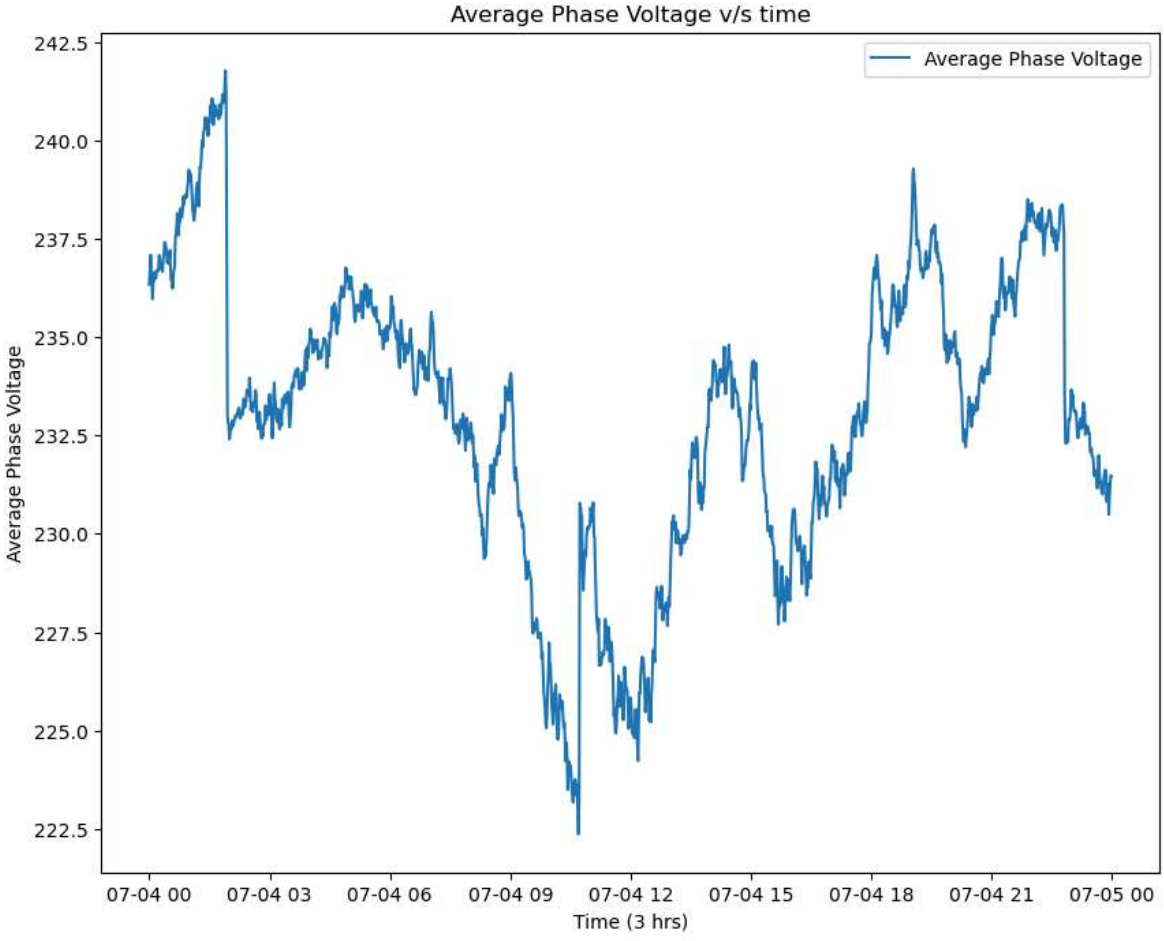
Out[8]:
```
(1440, 29)
```

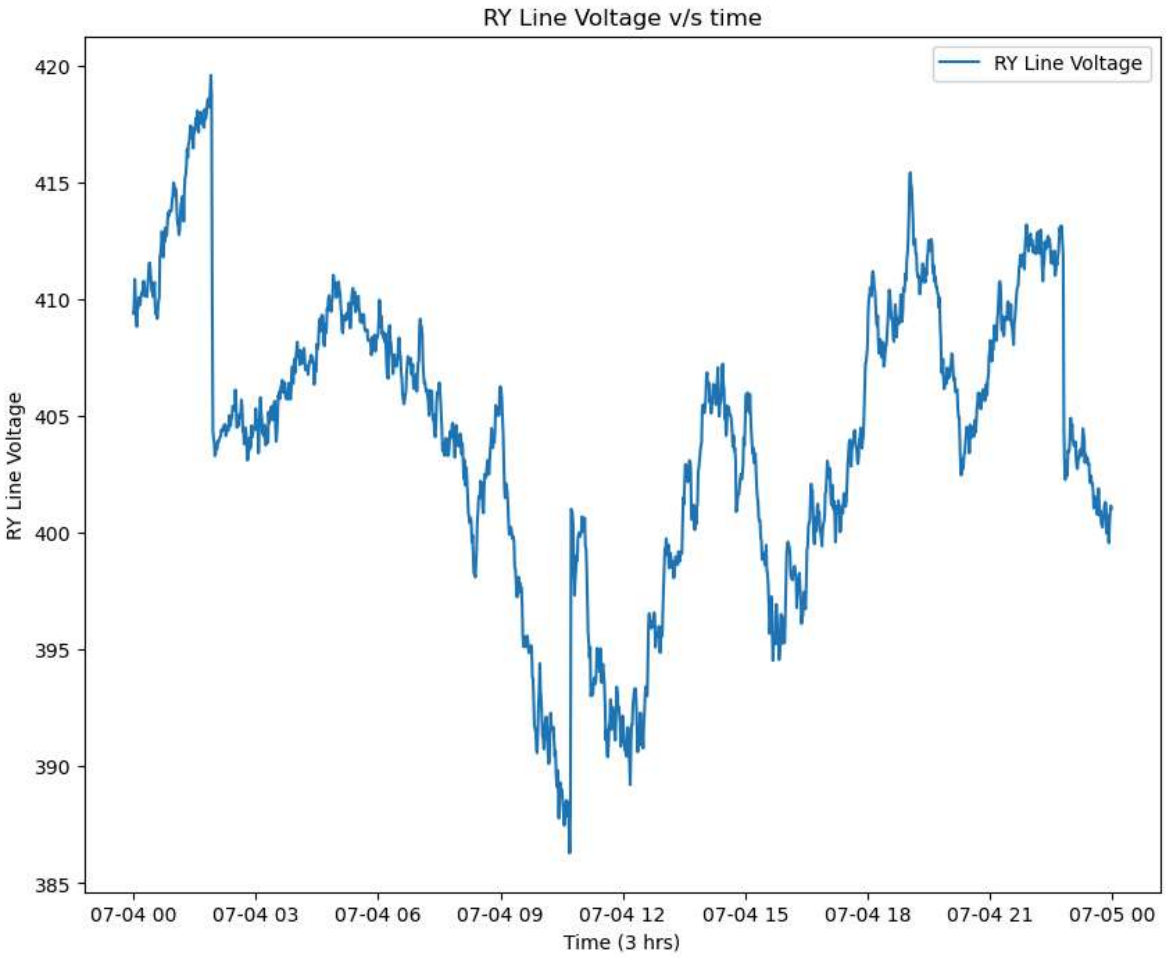
In [9]:
```
for col in data_minute_avg.columns:
    plt.figure(figsize=(10, 8))
    plt.plot(data_minute_avg.index, data_minute_avg[col], label=col)
    tit = col + " v/s time"
    plt.title(tit)
    plt.xlabel("Time (3 hrs)")
    plt.ylabel(col)
    plt.legend()
    plt.show()
```

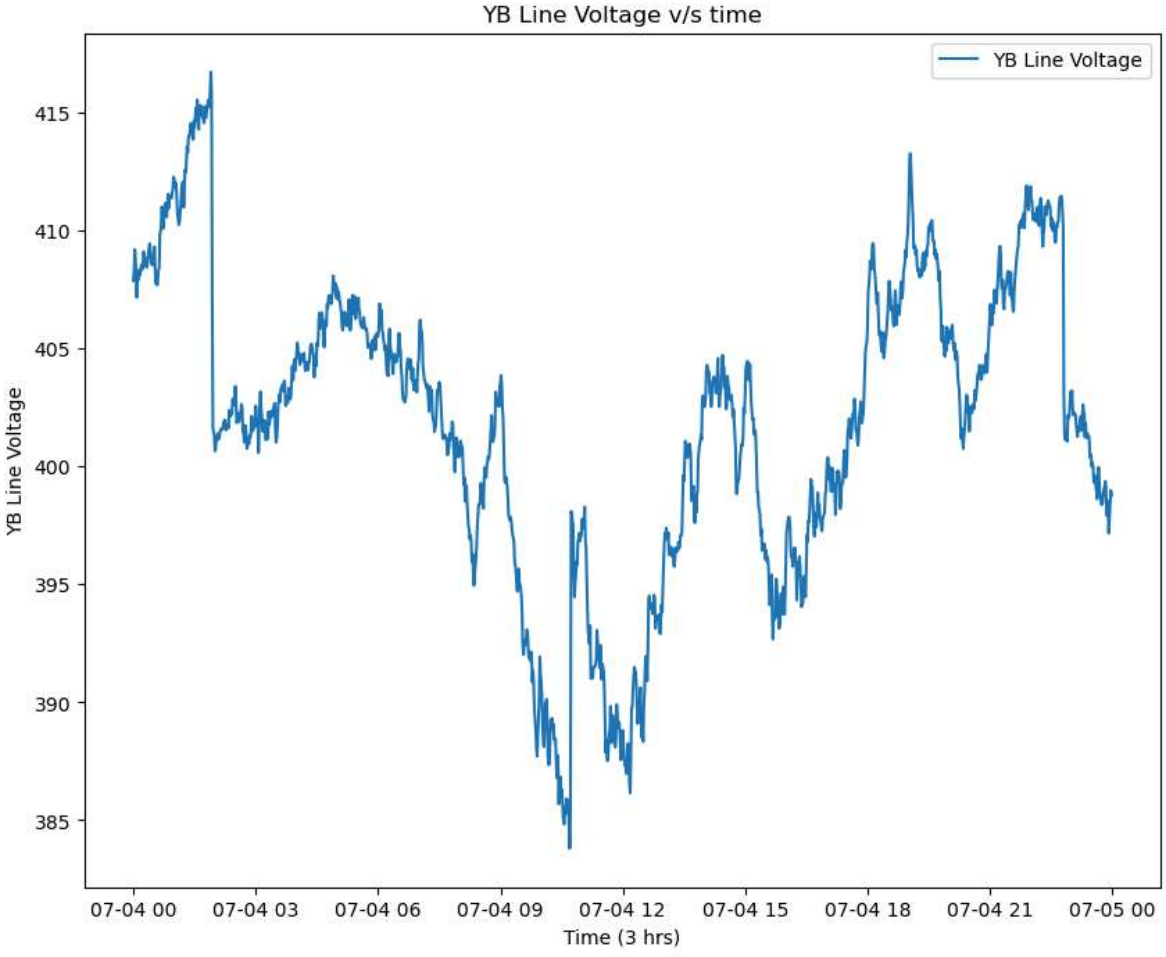

R Ph Voltage v/s time
R Ph Voltage
242.5
240.0
237.5
235.0
232.5
230.0
227.5
225.0
222.5
R Ph Voltage
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

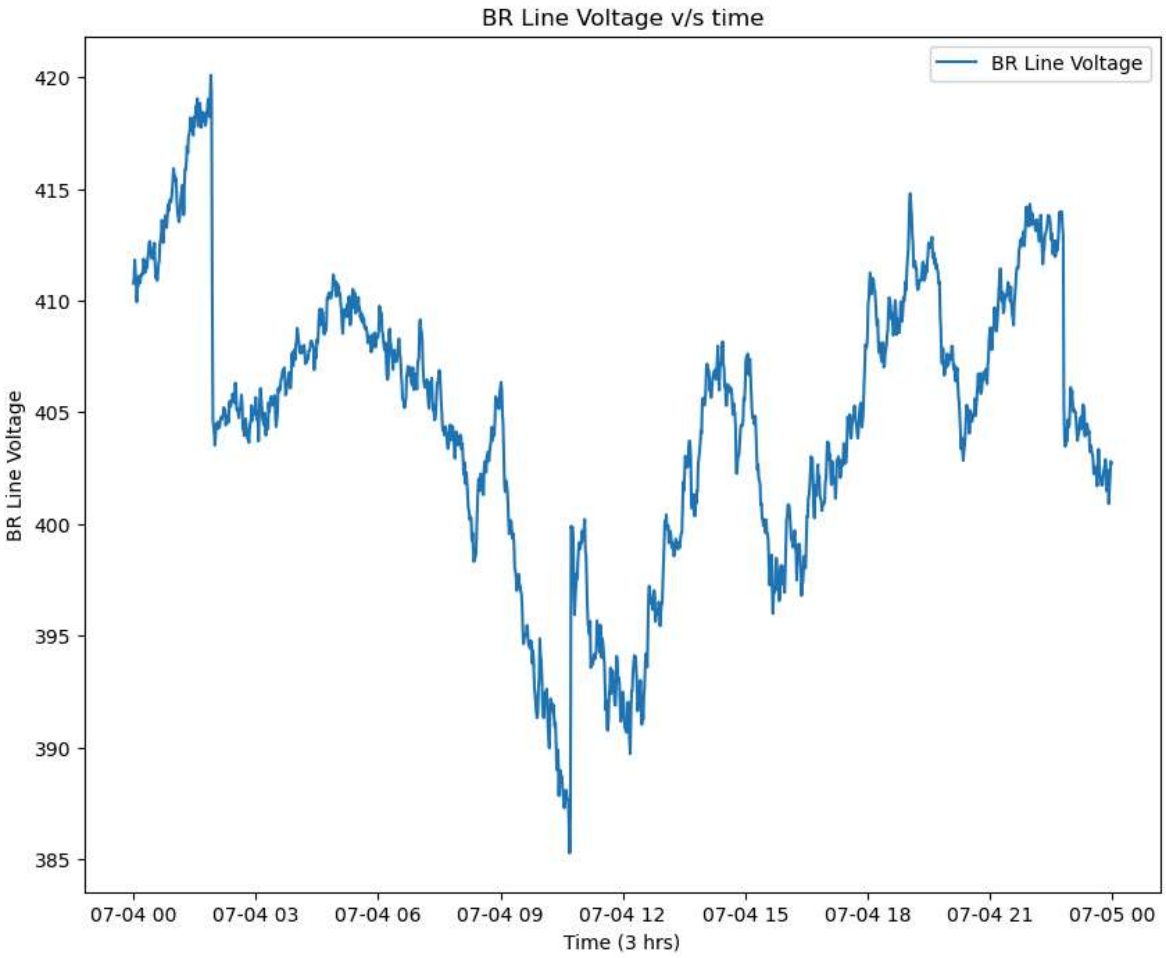

Y Ph Voltage v/s time
Y Ph Voltage
242.5
240.0
237.5
235.0
232.5
230.0
227.5
225.0
222.5
Y Ph Voltage
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

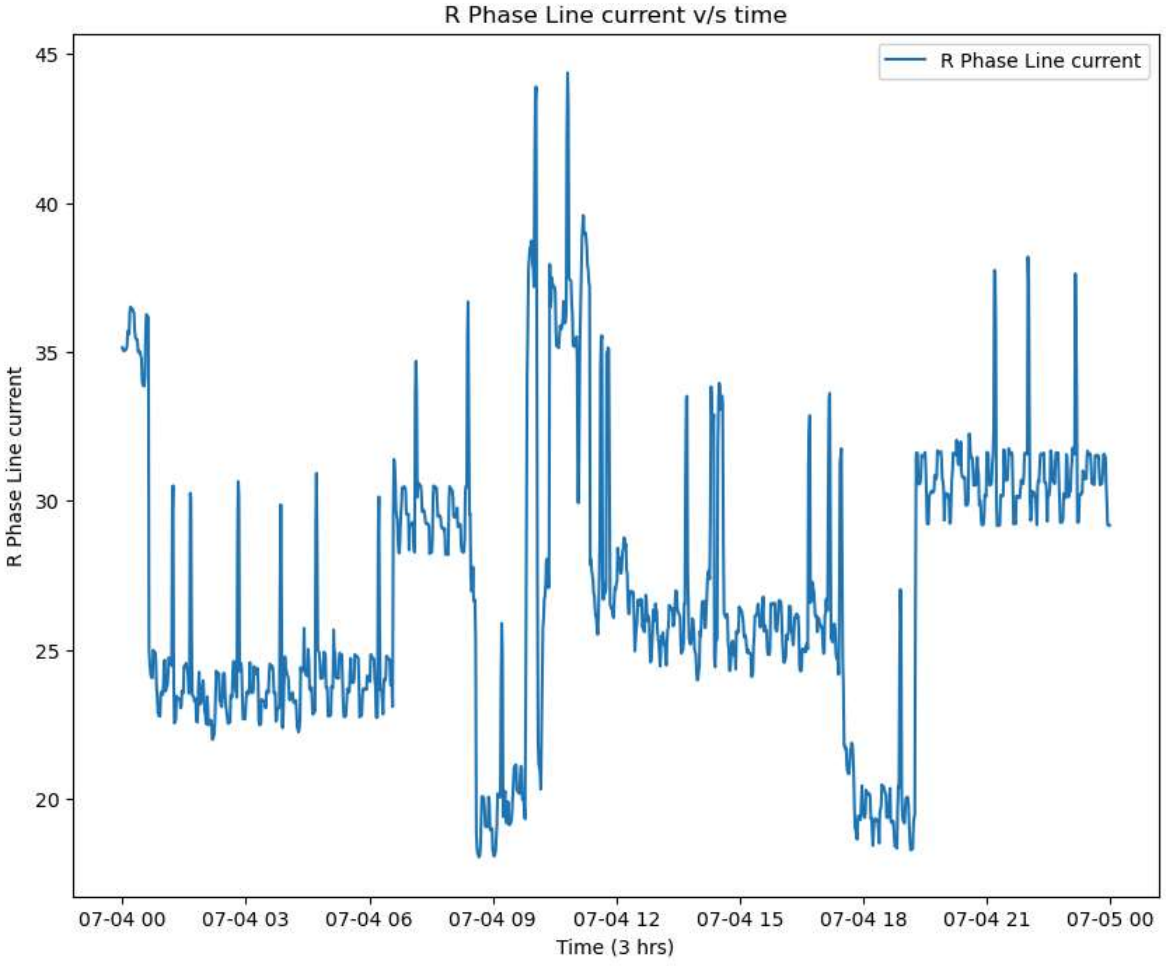

B Ph Voltage v/s time
B Ph Voltage
240.0
237.5
235.0
232.5
230.0
227.5
225.0
222.5
B Ph Voltage
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

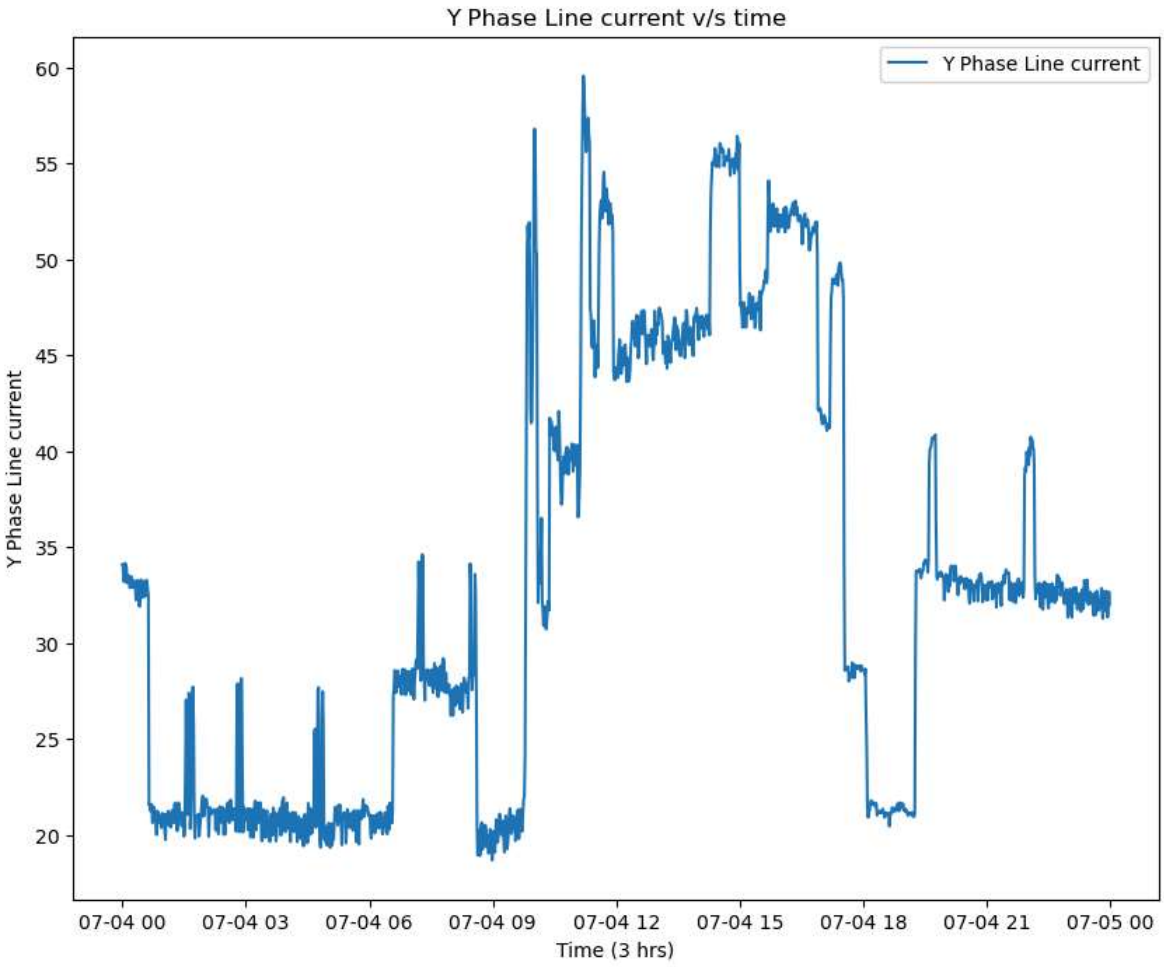

Average Phase Voltage v/s time
Average Phase Voltage
242.5
240.0
237.5
235.0
232.5
230.0
227.5
225.0
222.5
Average Phase Voltage
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

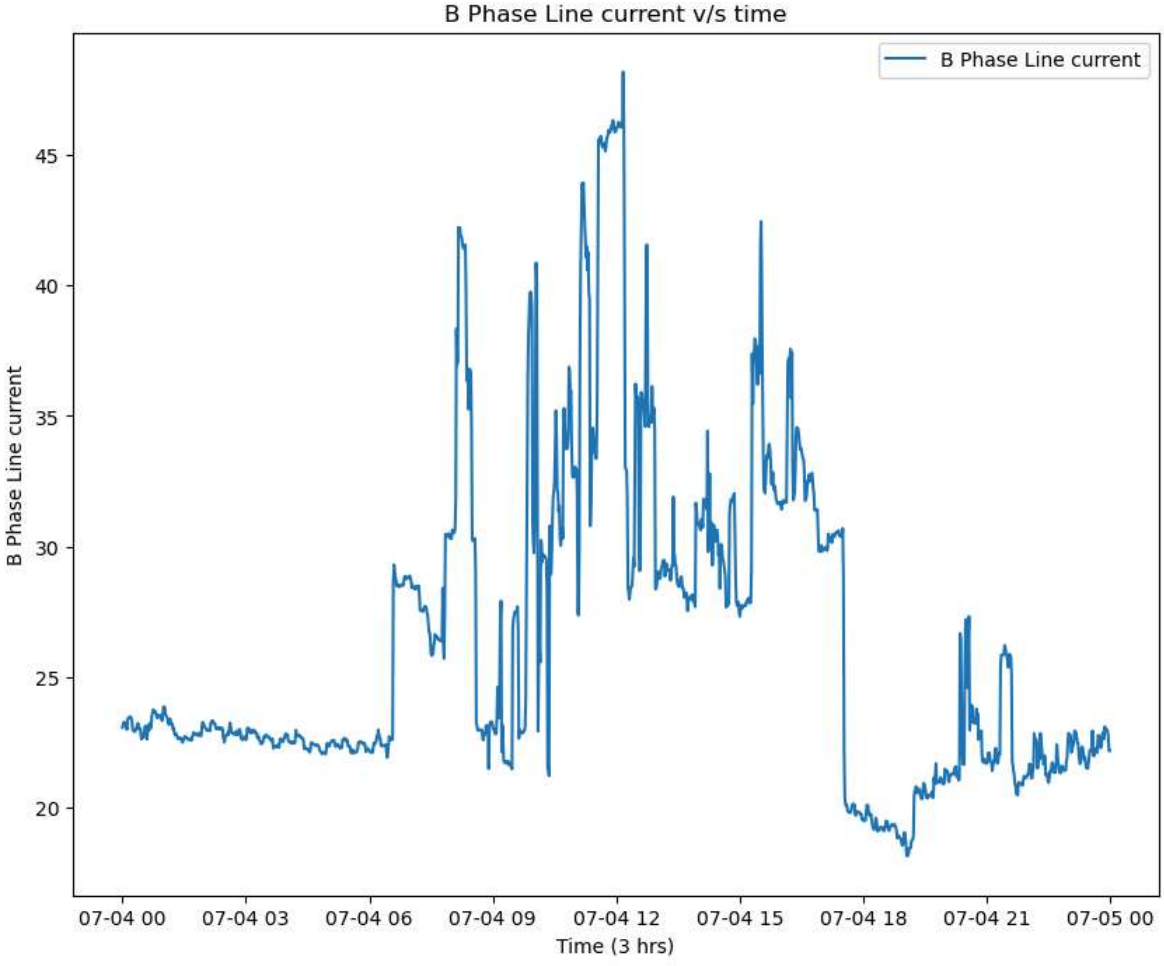

RY Line Voltage v/s time
RY Line Voltage
RY Line Voltage
420
415
410
405
400
395
390
385
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

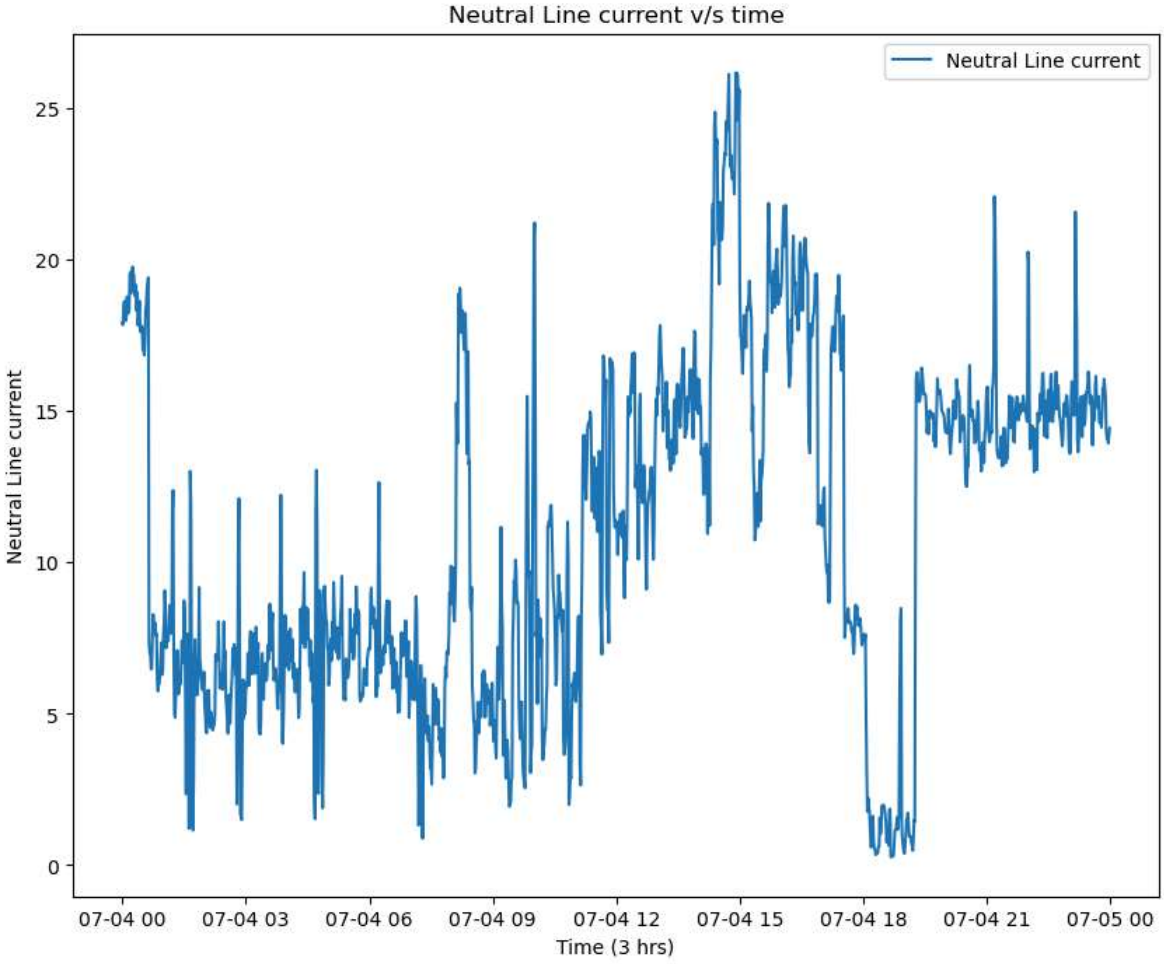

YB Line Voltage v/s time
YB Line Voltage
YB Line Voltage
415
410
405
400
395
390
385
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

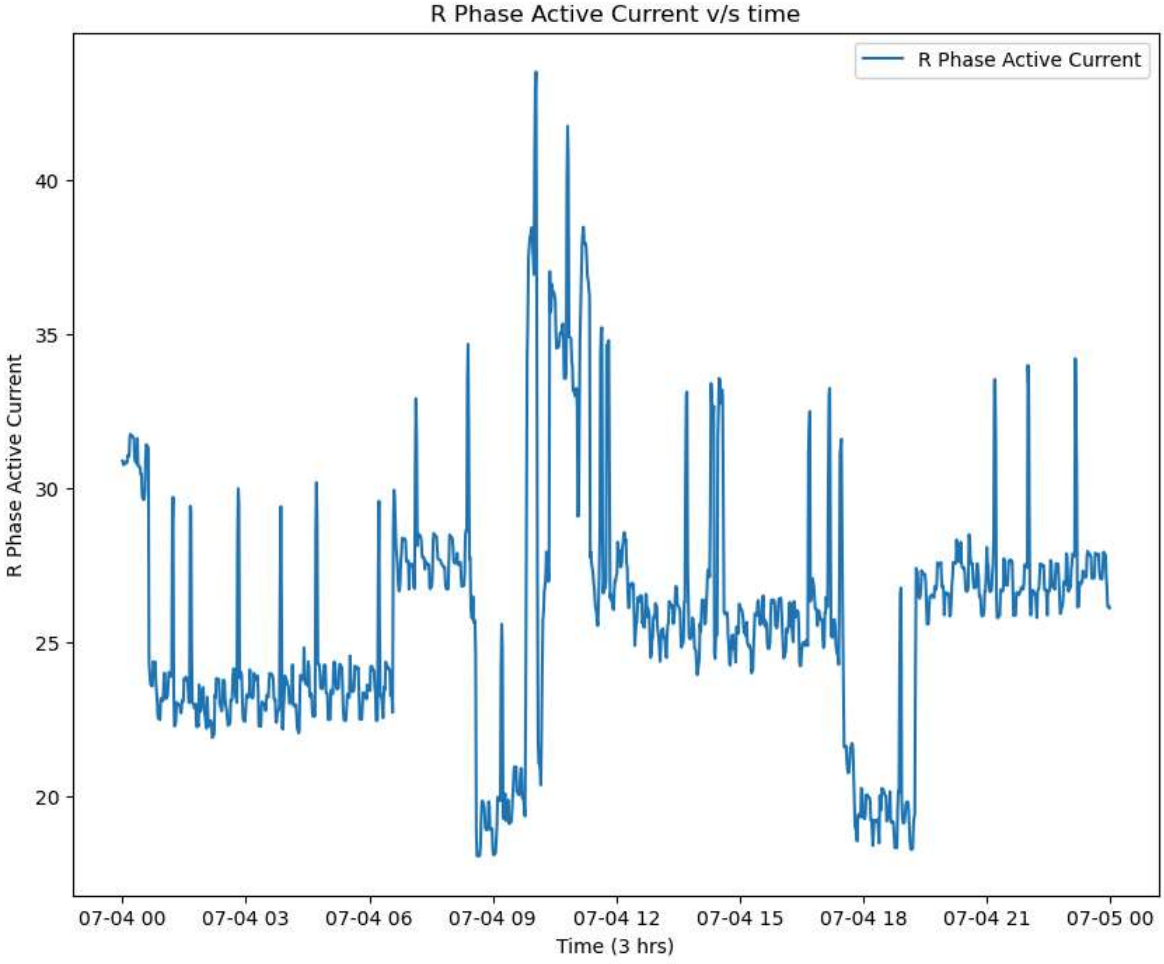

BR Line Voltage v/s time
BR Line Voltage
BR Line Voltage
420
415
410
405
400
395
390
385
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

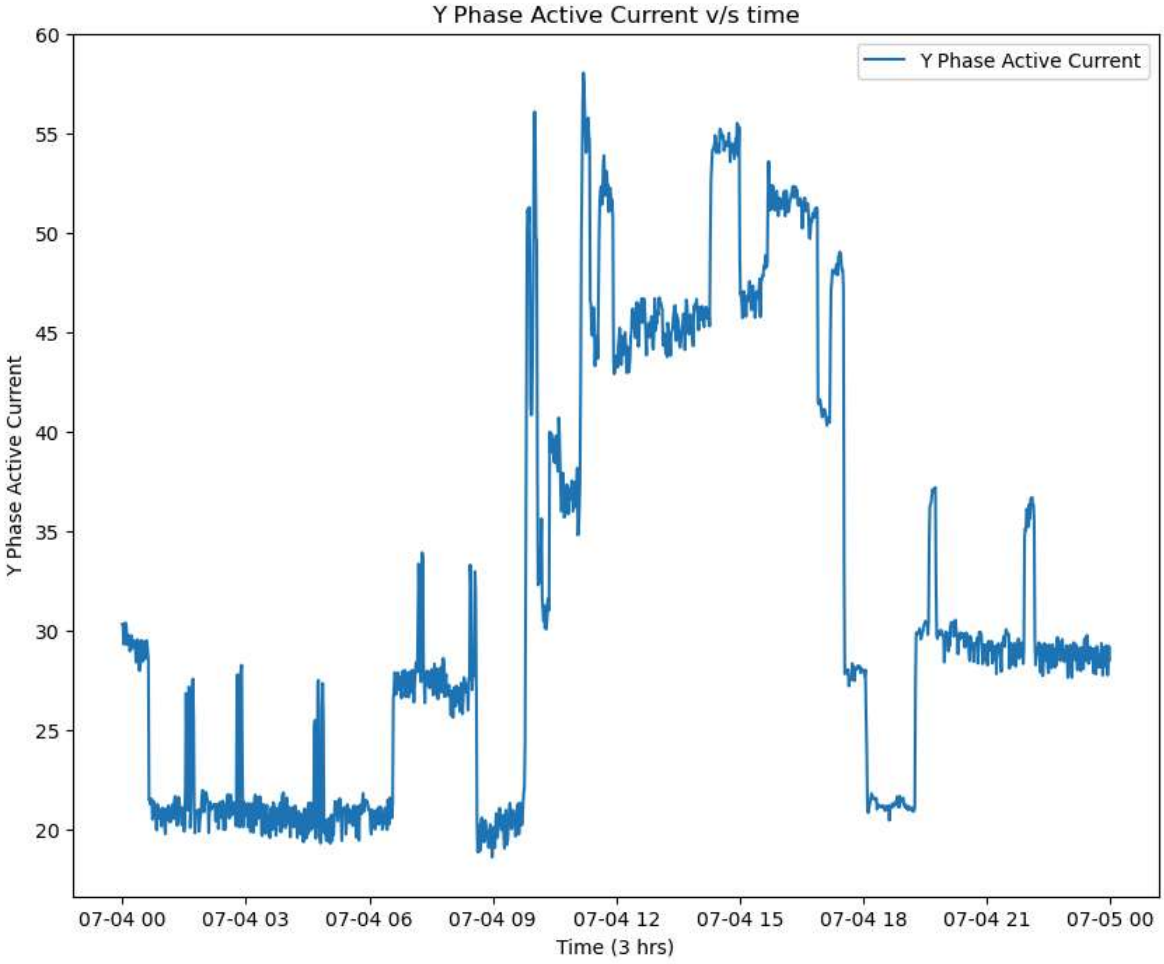

R Phase Line current v/s time
R Phase Line current
R Phase Line current
45
40
35
30
25
20
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

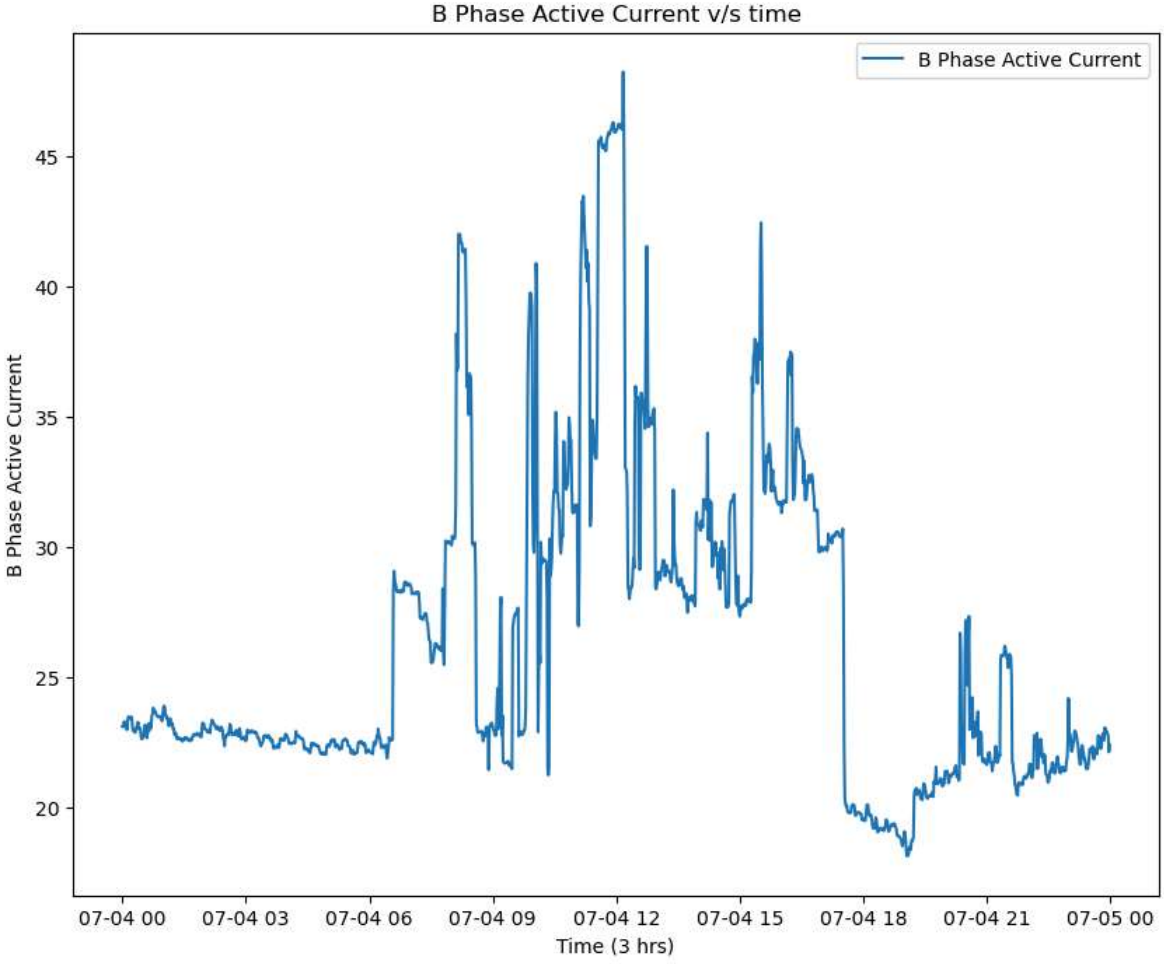

Y Phase Line current v/s time
Y Phase Line current
Y Phase Line current
60
55
50
45
40
35
30
25
20
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

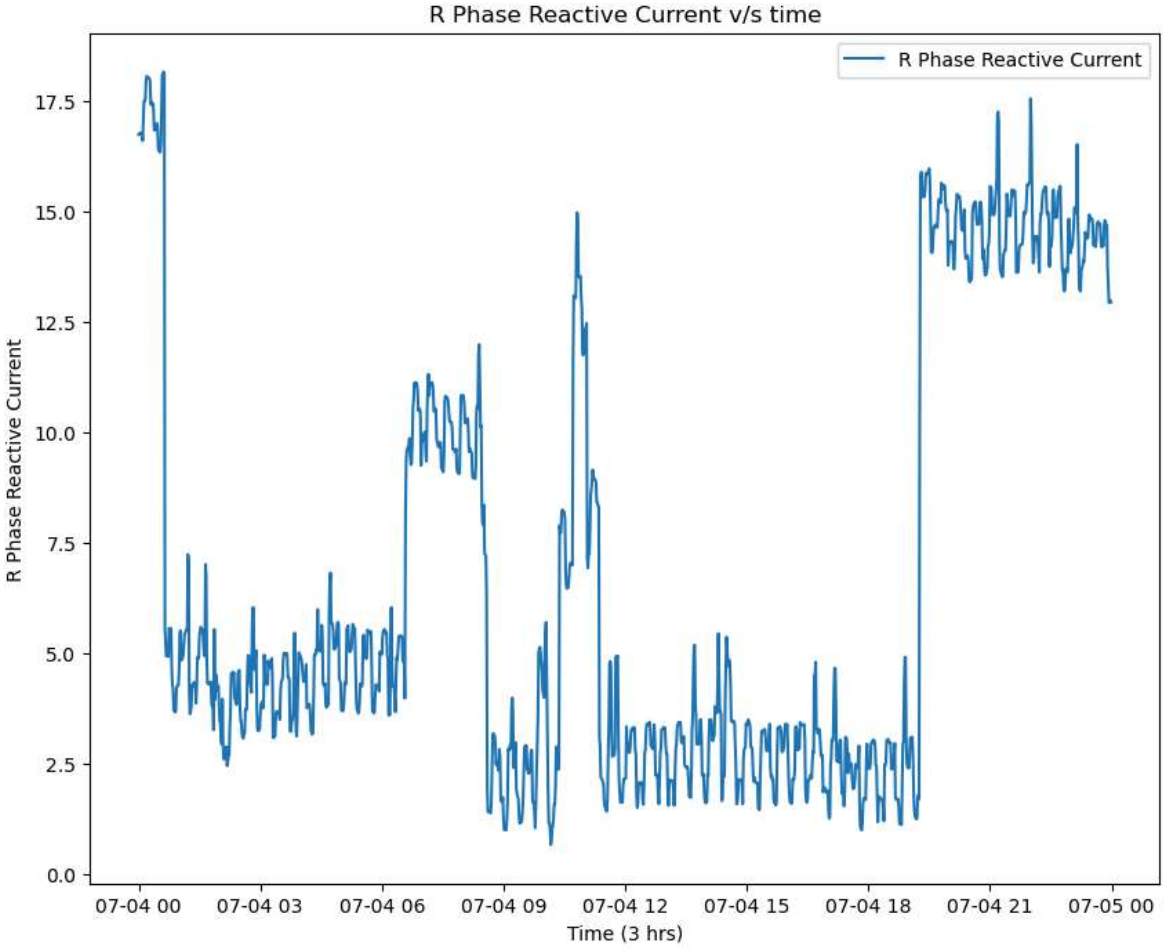

B Phase Line current v/s time
B Phase Line current
B Phase Line current
45
40
35
30
25
20
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

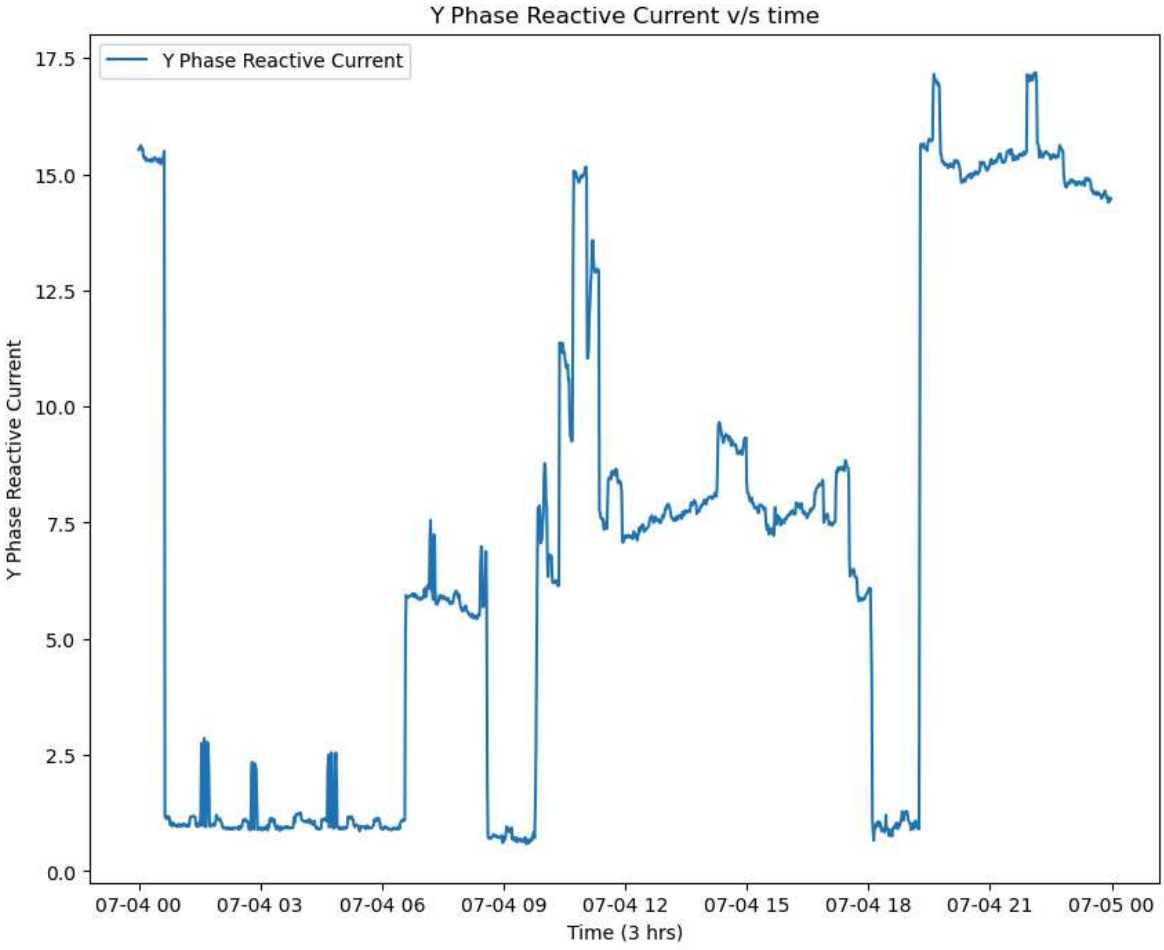

Neutral Line current v/s time
Neutral Line current
Neutral Line current
0
5
10
15
20
25
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

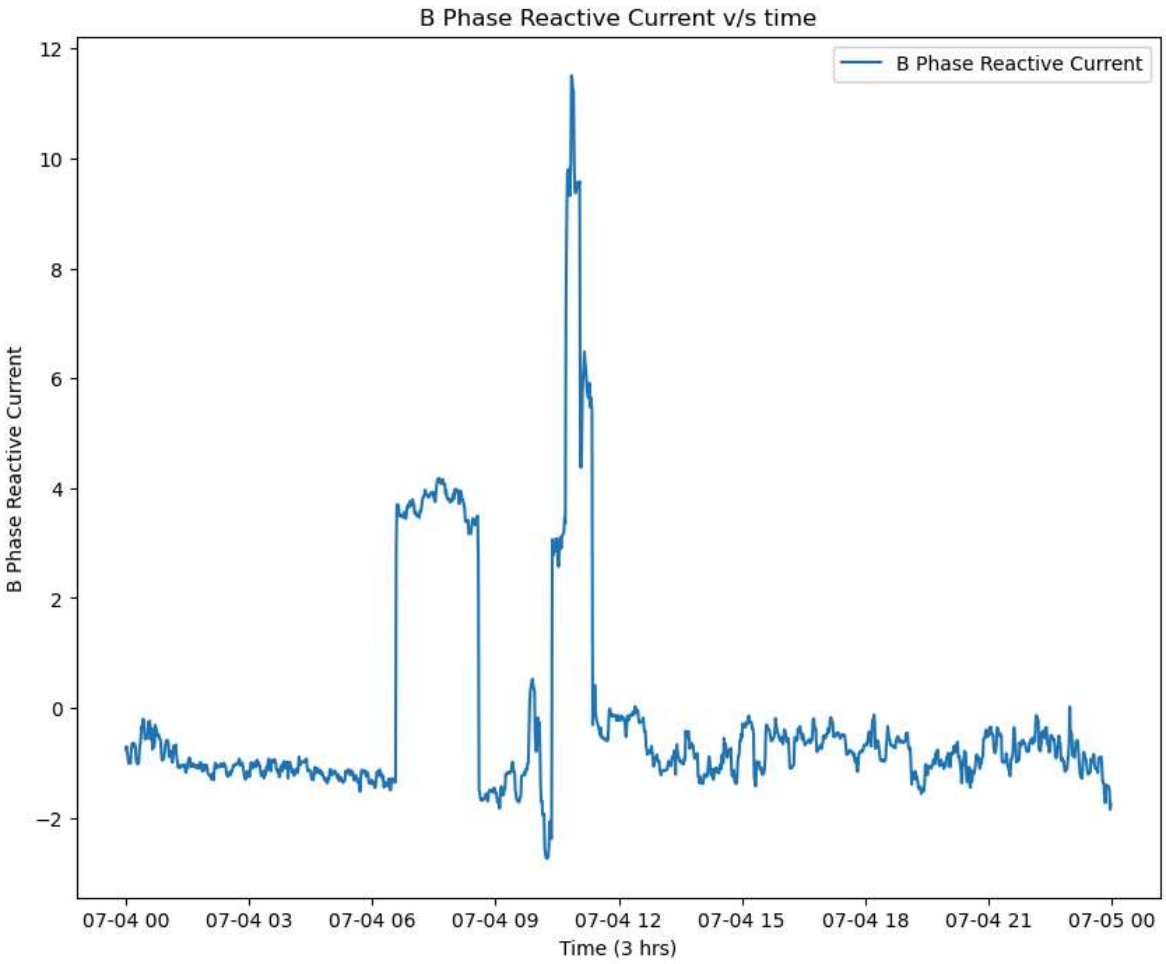

R Phase Active Current v/s time
R Phase Active Current
R Phase Active Current
20
25
30
35
40
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

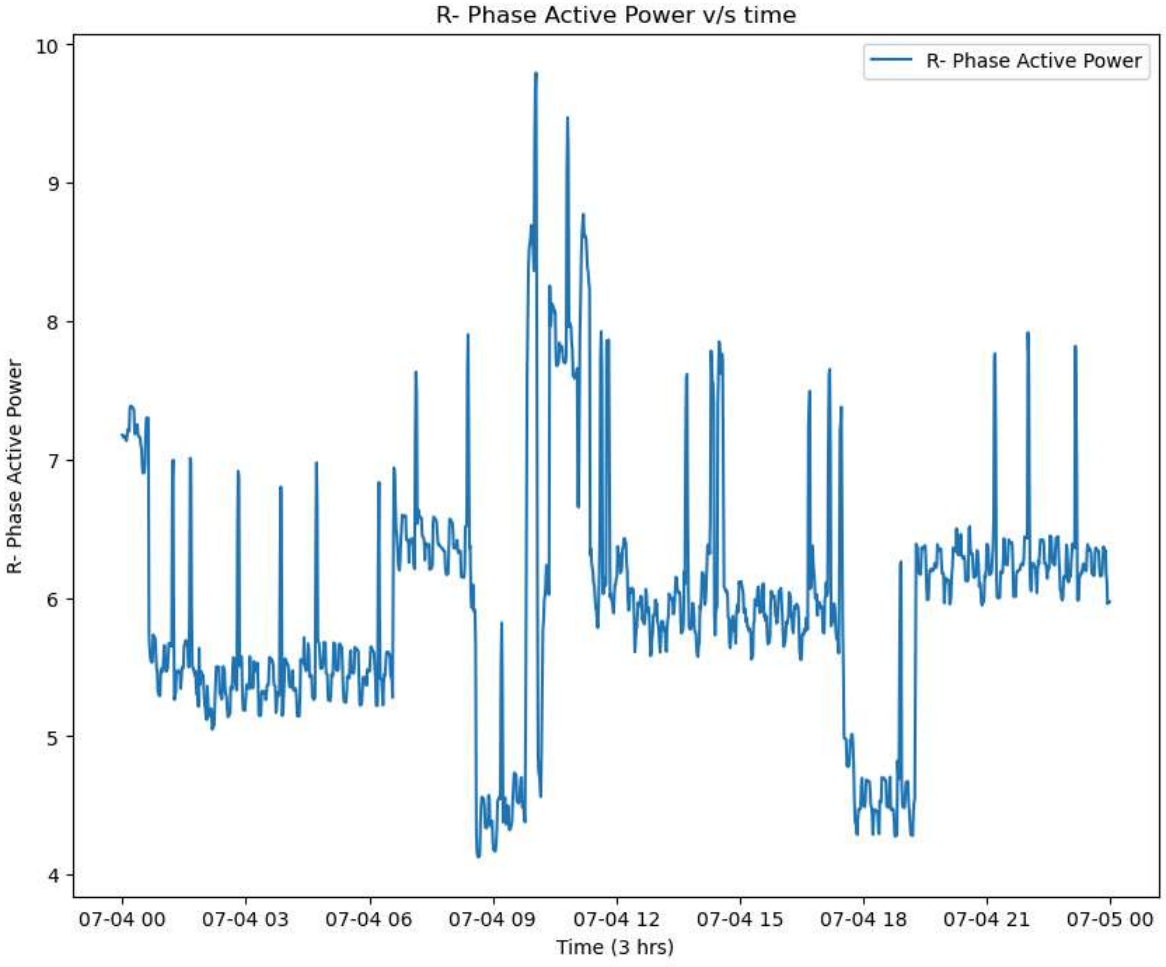

Y Phase Active Current v/s time
Y Phase Active Current
Y Phase Active Current
60
55
50
45
40
35
30
25
20
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

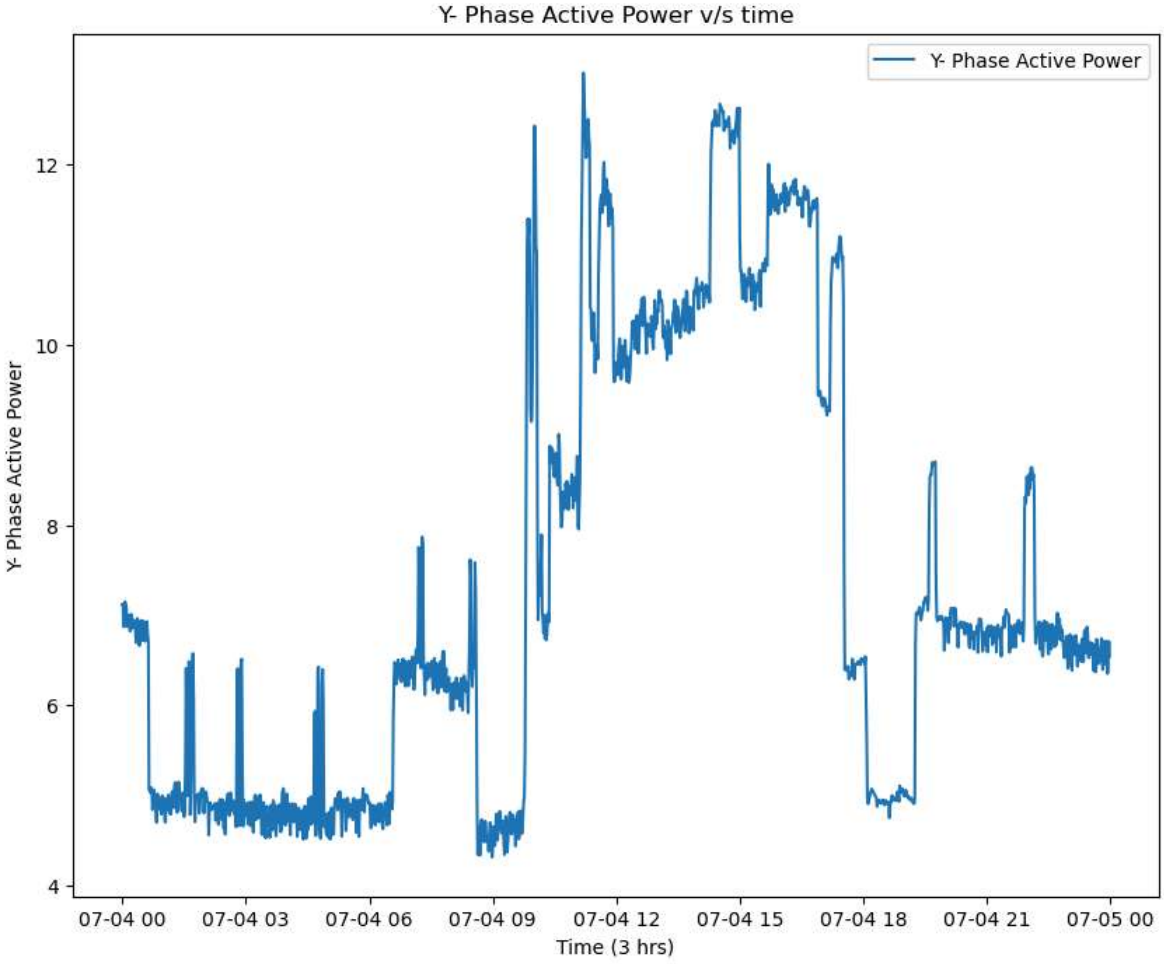

B Phase Active Current v/s time
B Phase Active Current
B Phase Active Current
45
40
35
30
25
20
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

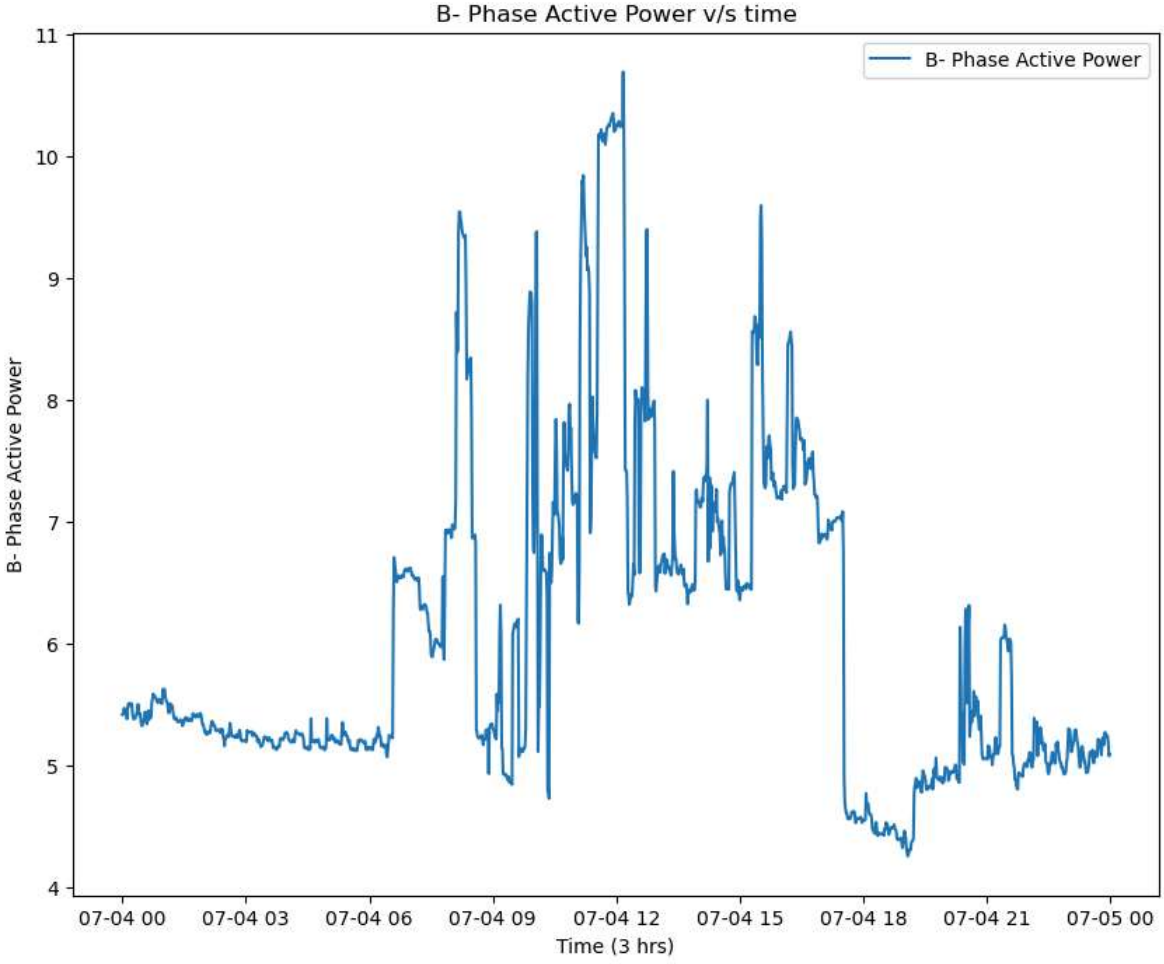

R Phase Reactive Current v/s time
R Phase Reactive Current
R Phase Reactive Current
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

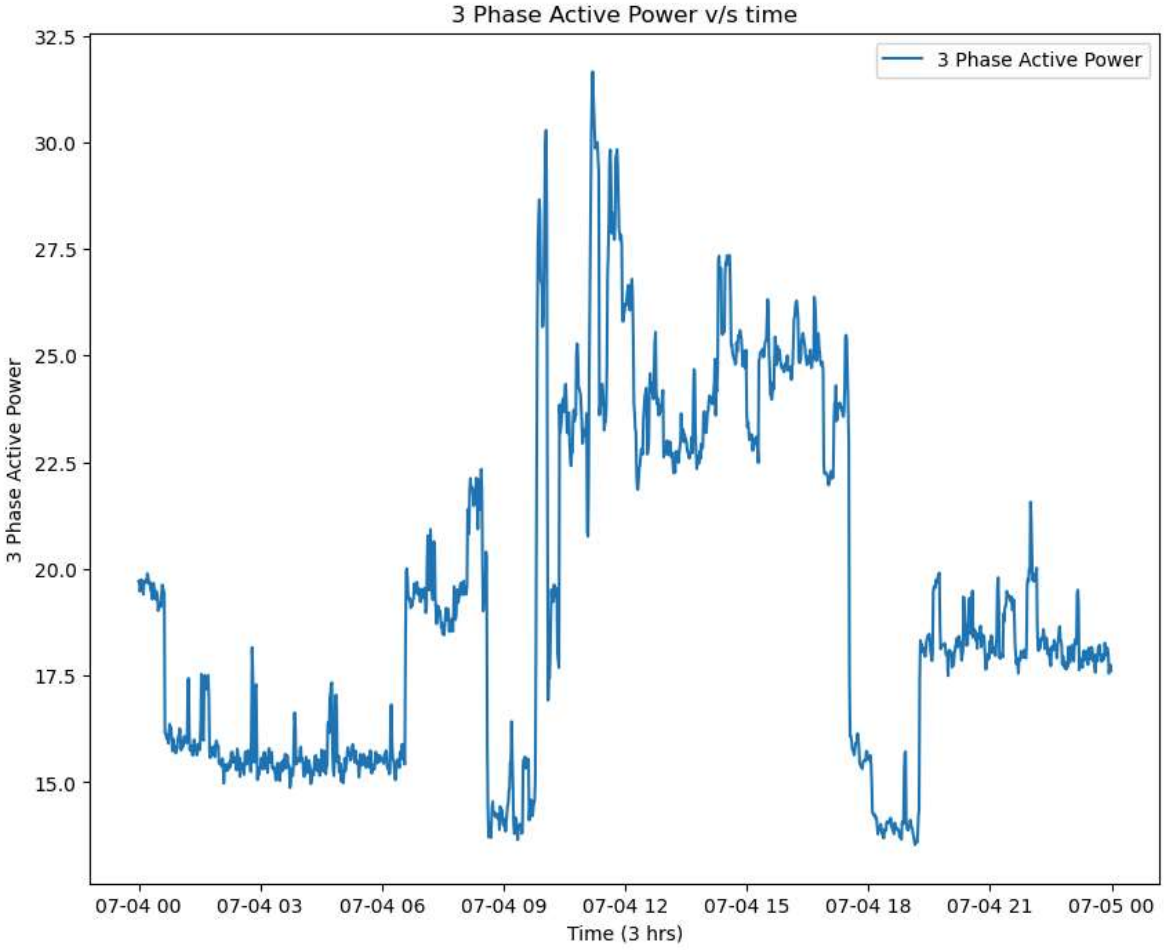

Y Phase Reactive Current v/s time
Y Phase Reactive Current
Y Phase Reactive Current
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

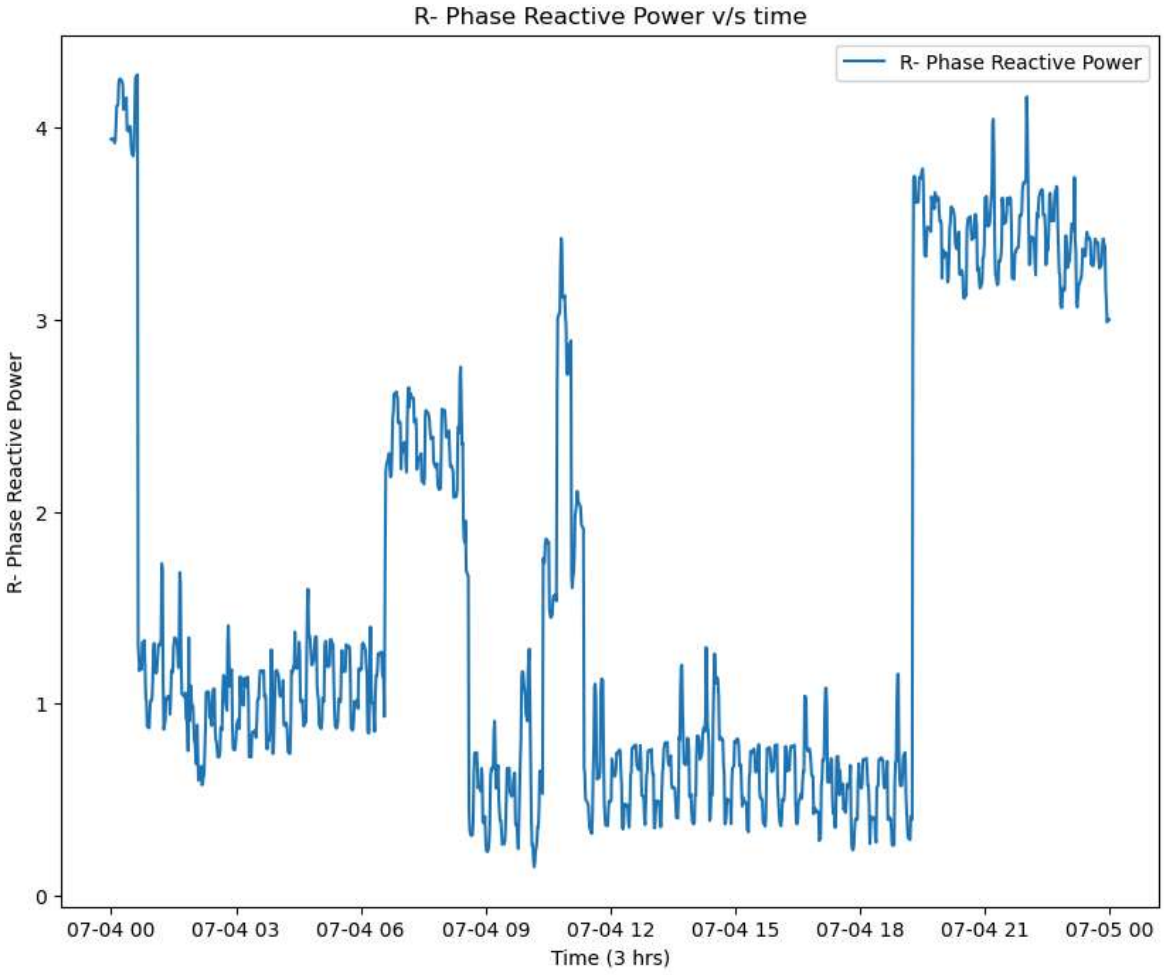

B Phase Reactive Current v/s time
B Phase Reactive Current
B Phase Reactive Current
12
10
8
6
4
2
0
−2
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

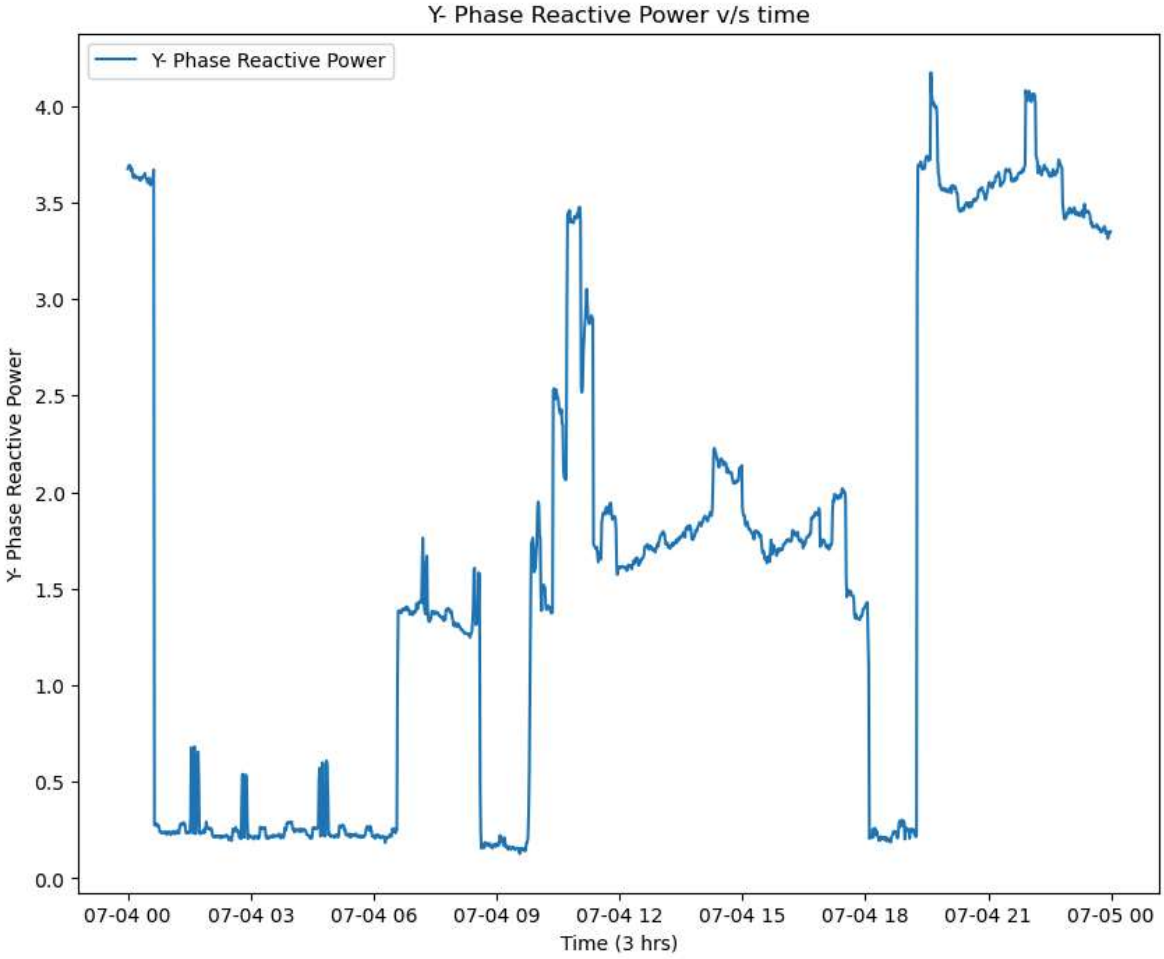

R- Phase Active Power v/s time
R- Phase Active Power
R- Phase Active Power
10
9
8
7
6
5
4
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

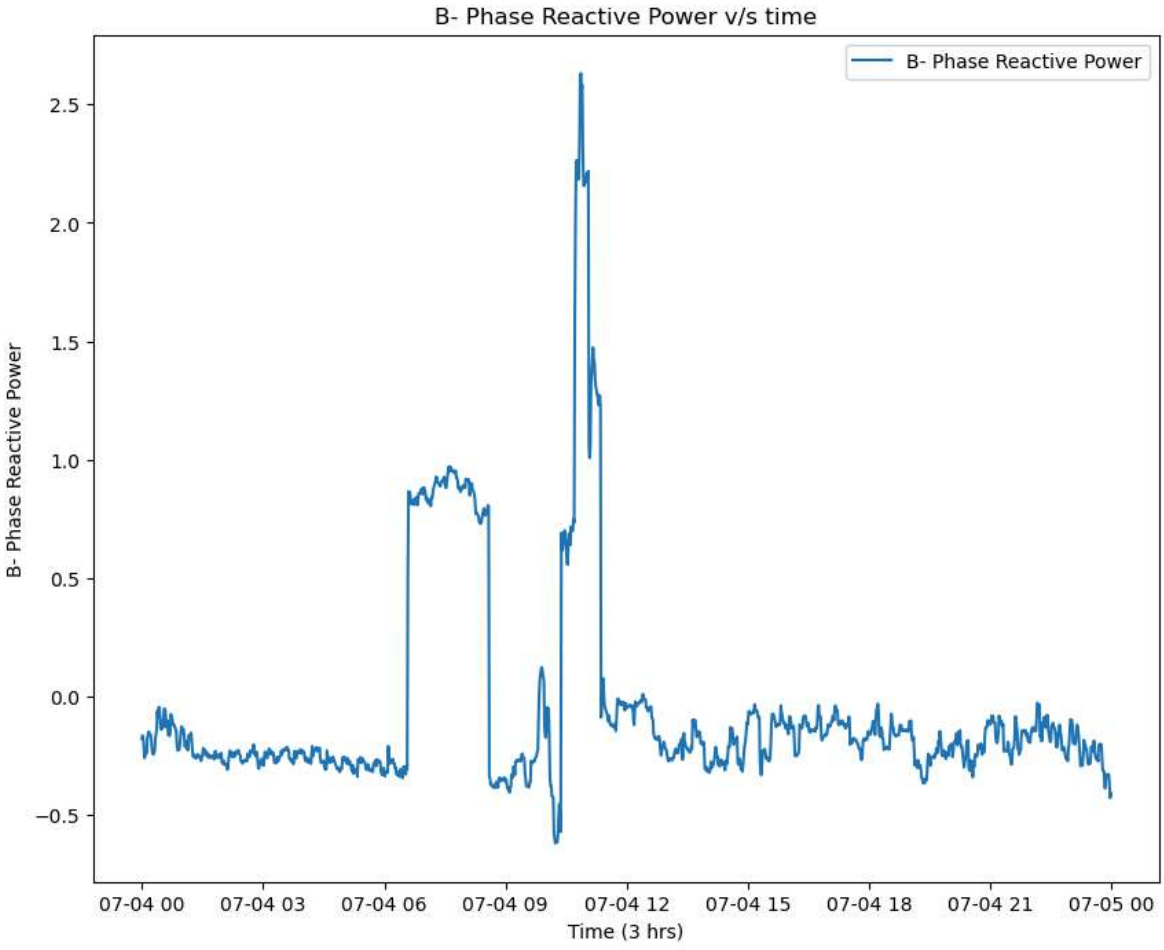

Y- Phase Active Power v/s time
Y- Phase Active Power
Y- Phase Active Power
12
10
8
6
4
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

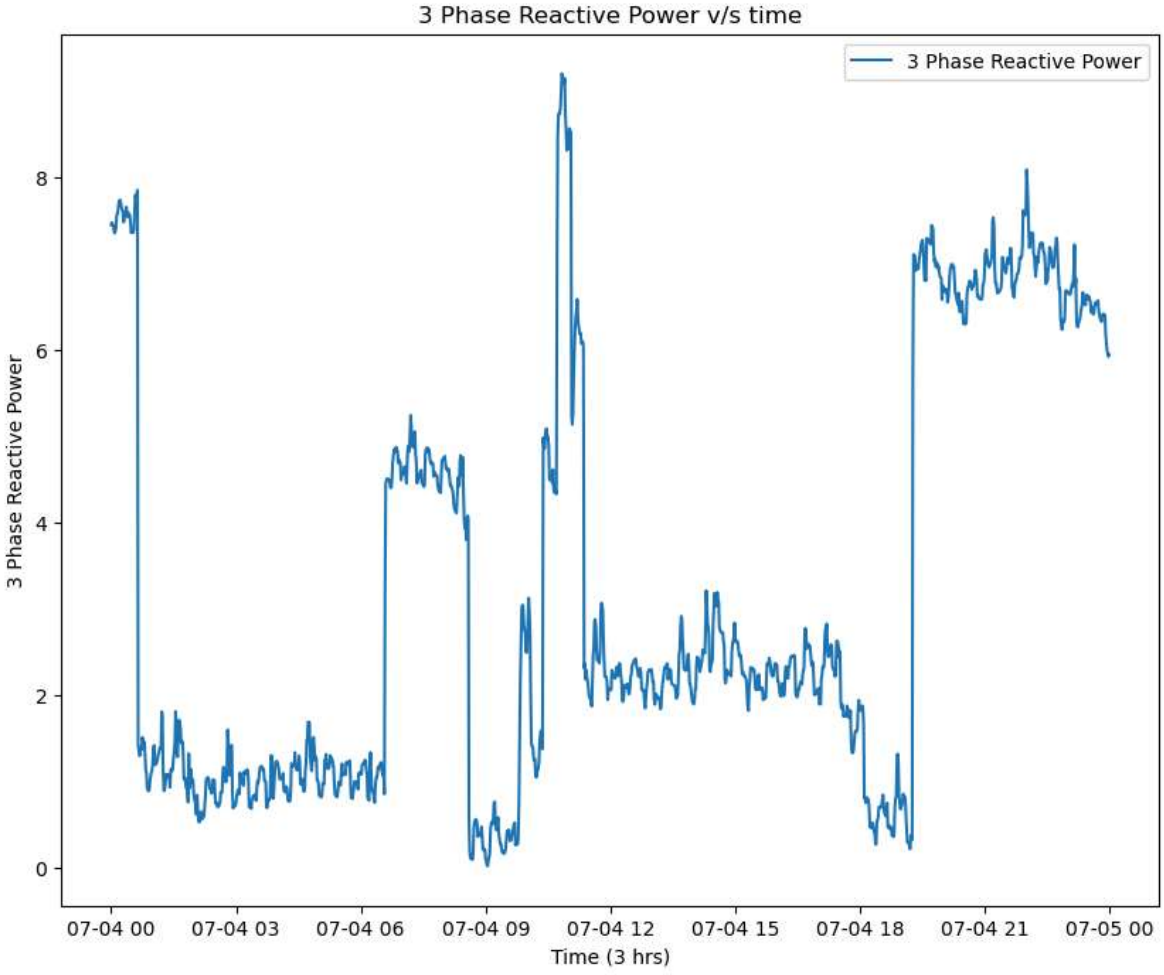

B- Phase Active Power v/s time
B- Phase Active Power
B- Phase Active Power
11
10
9
8
7
6
5
4
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

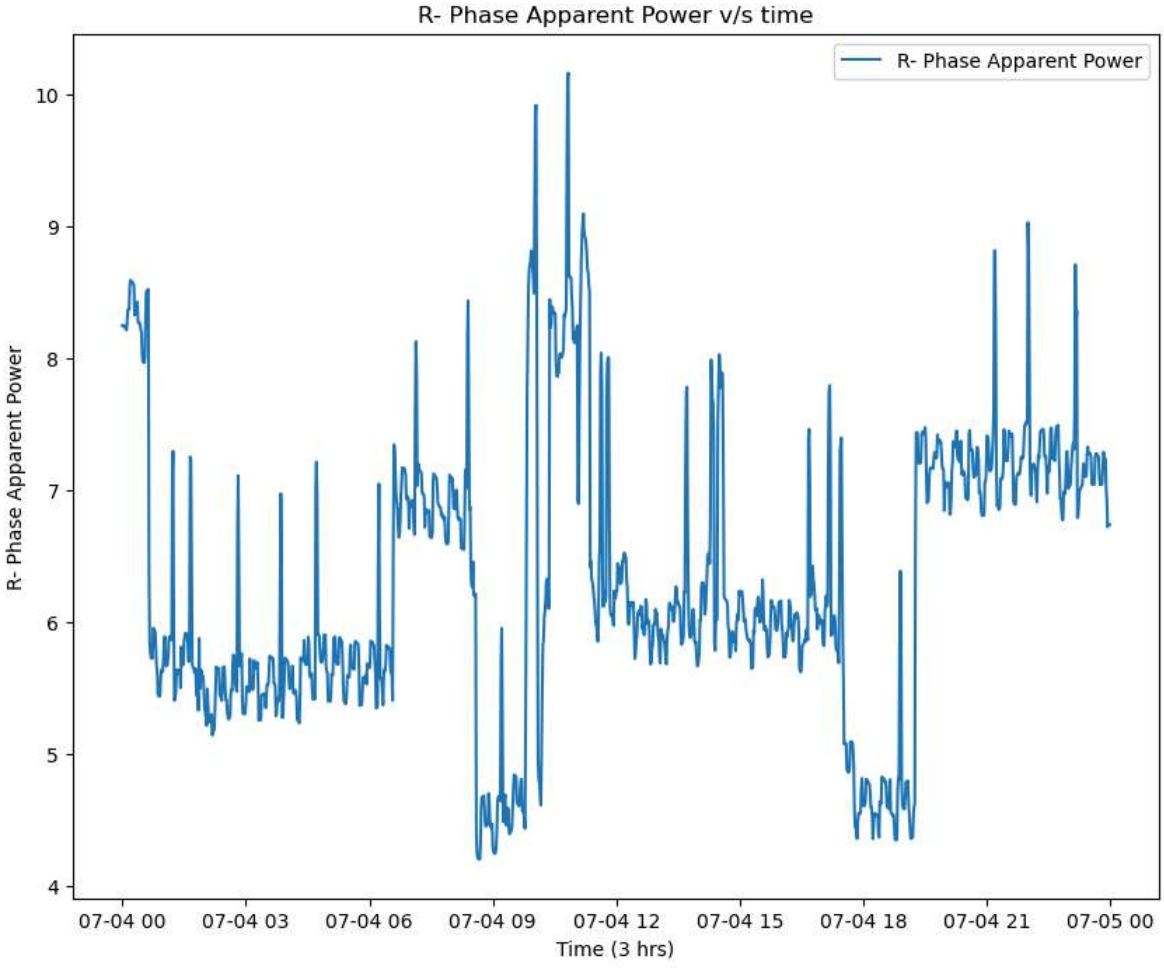

3 Phase Active Power v/s time
3 Phase Active Power
32.5
30.0
27.5
25.0
22.5
20.0
17.5
15.0
3 Phase Active Power
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

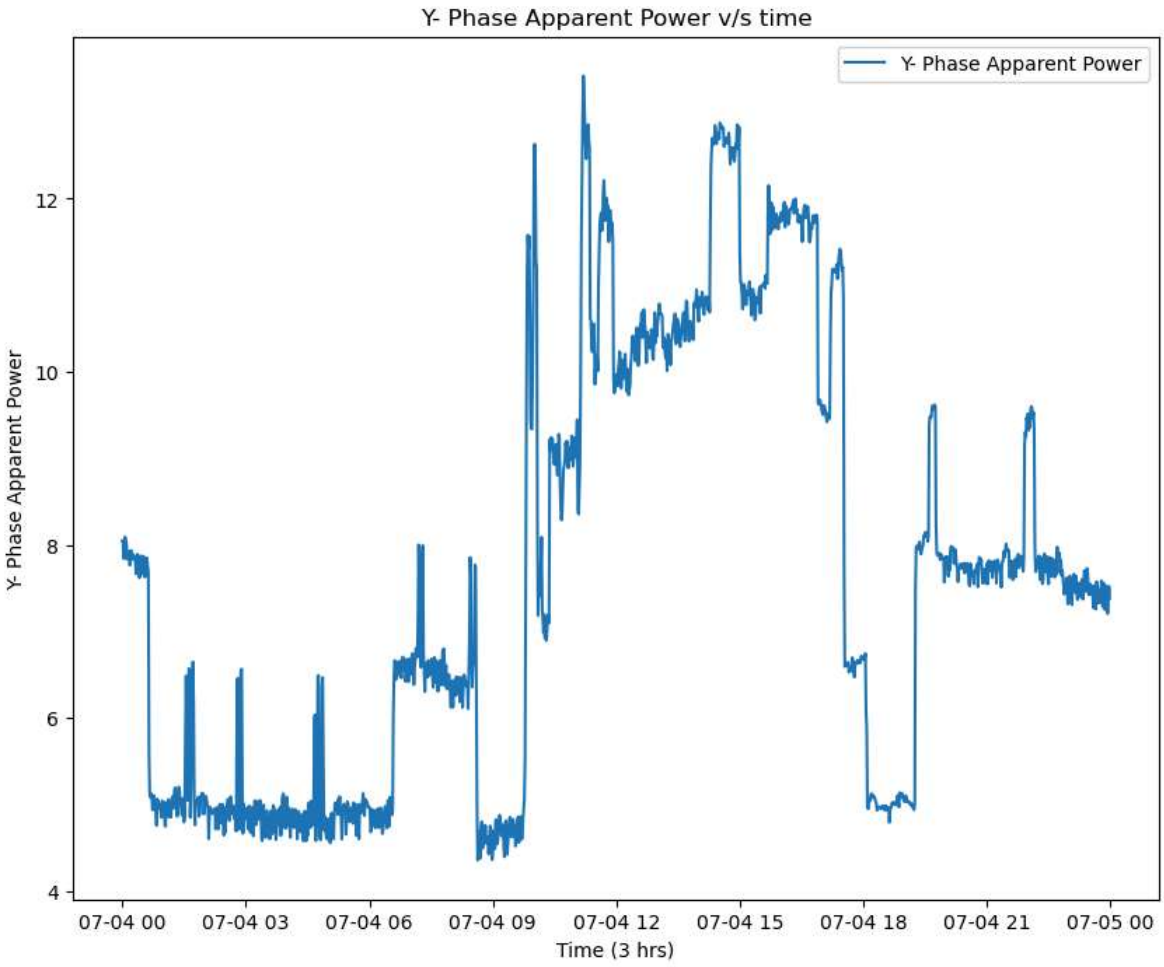

R- Phase Reactive Power v/s time
R- Phase Reactive Power
4
3
2
1
0
R- Phase Reactive Power
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

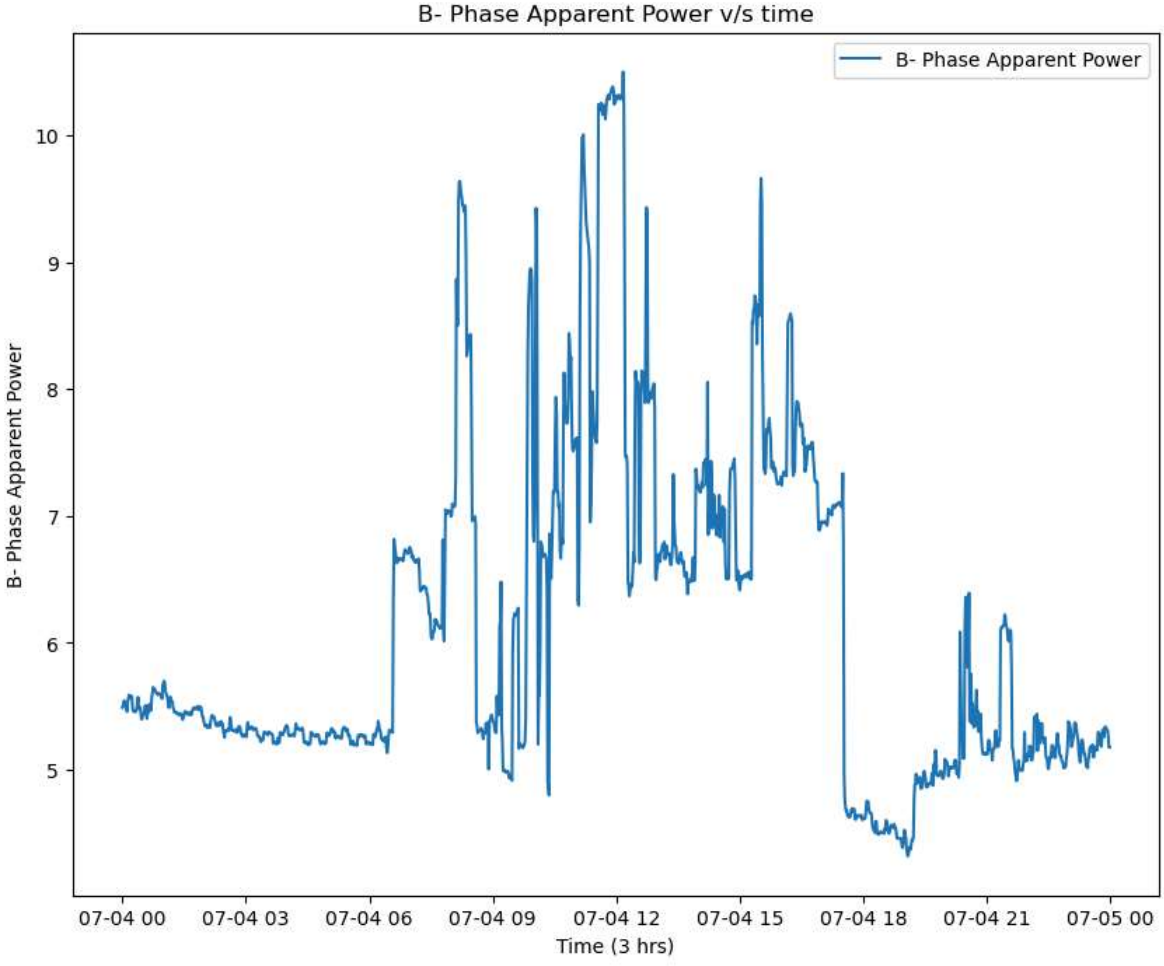

Y- Phase Reactive Power v/s time
Y- Phase Reactive Power
Y- Phase Reactive Power
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

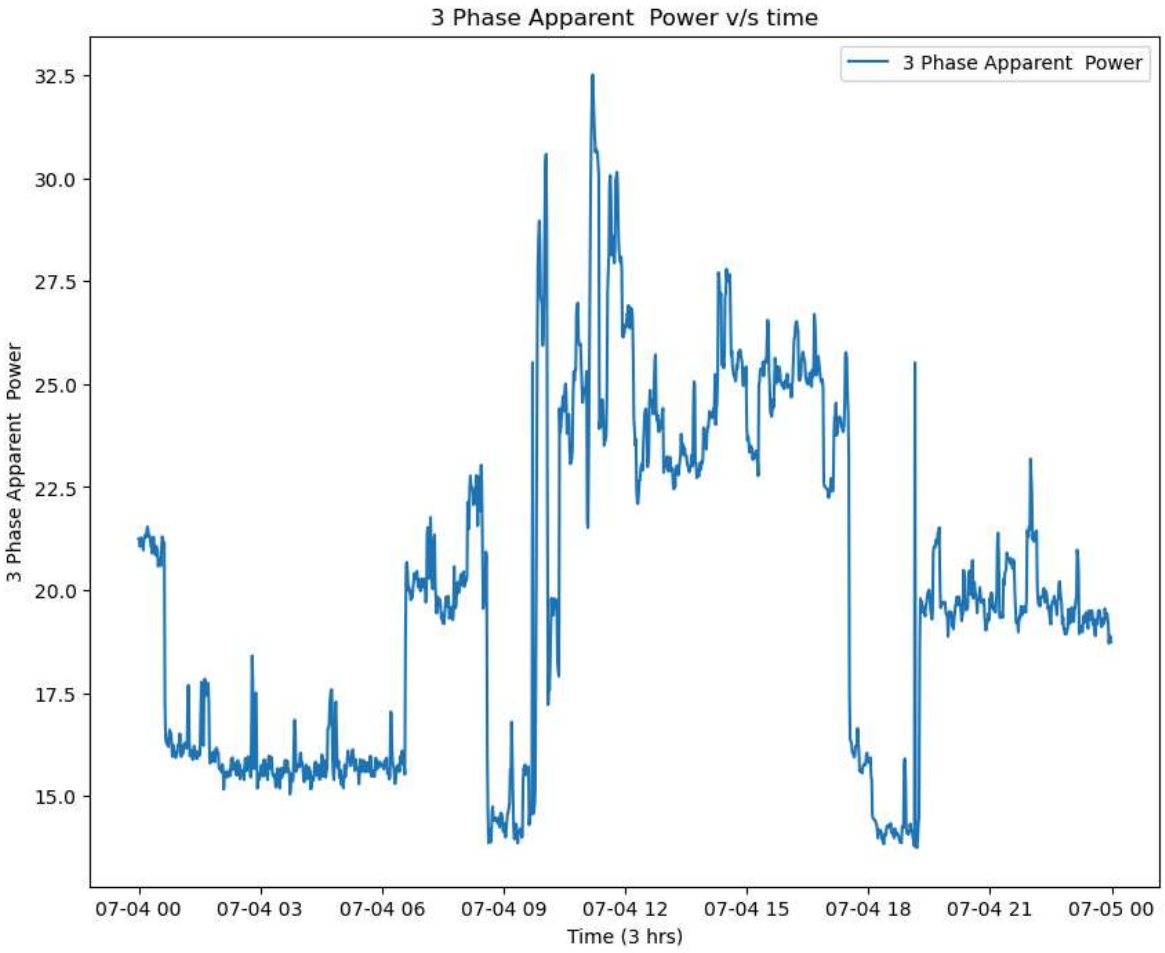

B- Phase Reactive Power v/s time
B- Phase Reactive Power
B- Phase Reactive Power
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)


3 Phase Reactive Power v/s time
3 Phase Reactive Power
3 Phase Reactive Power
0
2
4
6
8
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)


R- Phase Apparent Power v/s time
R- Phase Apparent Power
R- Phase Apparent Power
4
5
6
7
8
9
10
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)


Y- Phase Apparent Power v/s time
Y- Phase Apparent Power
Y- Phase Apparent Power
12
10
8
6
4
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)


B- Phase Apparent Power v/s time
B- Phase Apparent Power
B- Phase Apparent Power
10
9
8
7
6
5
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00
Time (3 hrs)

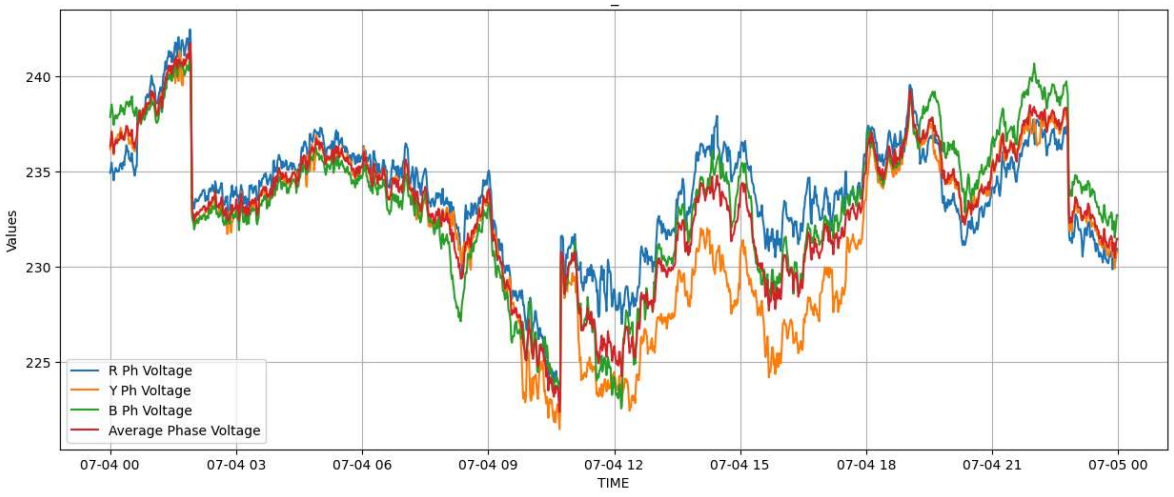


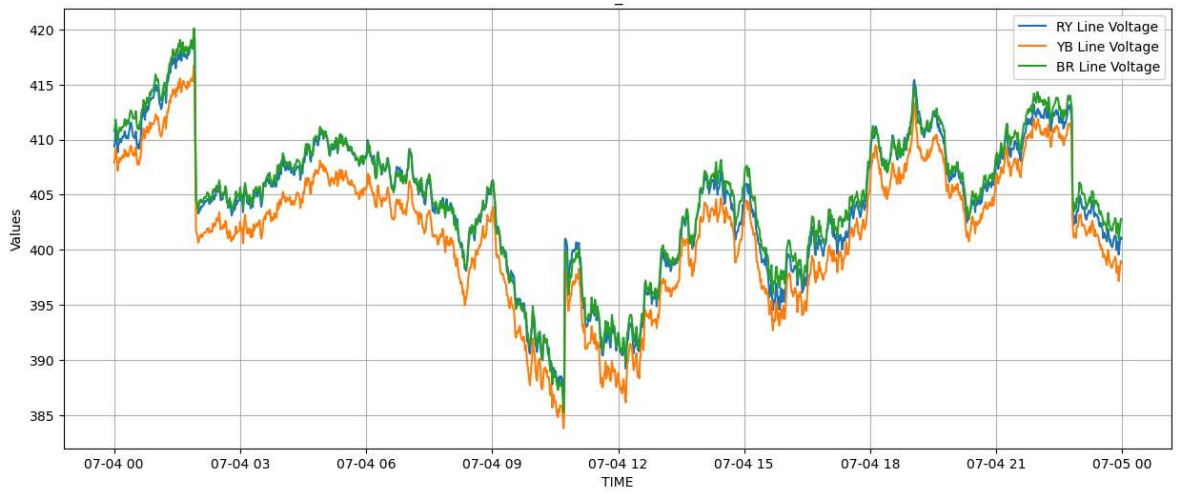
In [23]:
```
columns = data.columns.to_list()
grouped_lists = []
for i in range(0, 3, 3):
    sublist = columns[i:i + 4]
    grouped_lists.append(sublist)
for i in range(4,7,3):
    sublist = columns[i:i + 3]
    grouped_lists.append(sublist)
for i in range(7, 10, 3):
    sublist = columns[i:i + 4]
    grouped_lists.append(sublist)
for i in range(11,17,3):
    sublist = columns[i:i + 3]
    grouped_lists.append(sublist)
for i in range(17, len(columns), 4):
    sublist = columns[i:i + 4]
    grouped_lists.append(sublist)
grouped_lists
```

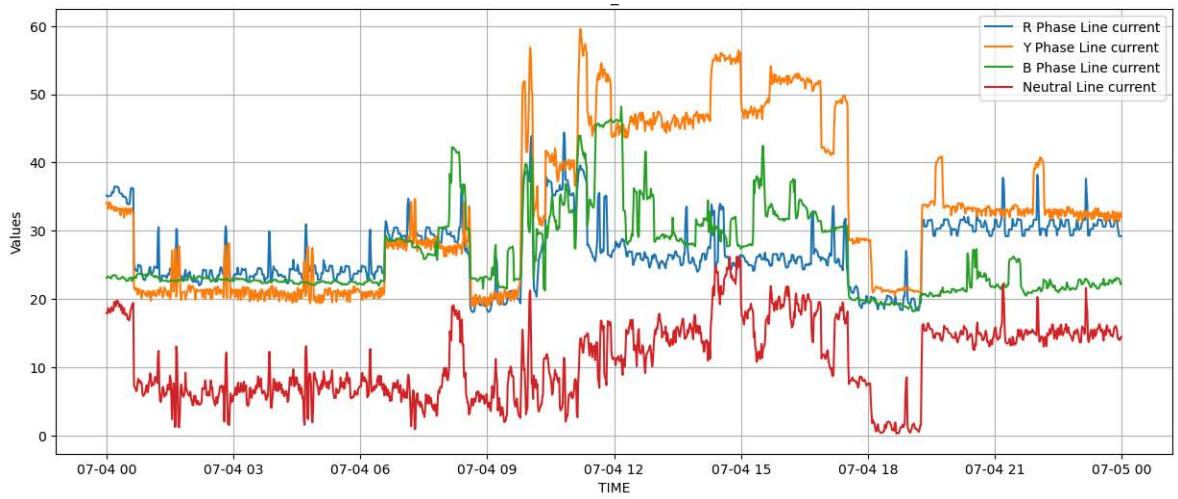
Out[23]:
```
[['R Ph Voltage', 'Y Ph Voltage', 'B Ph Voltage', 'Average Phase Voltage'],
 ['RY Line Voltage', 'YB Line Voltage', 'BR Line Voltage'],
 ['R Phase Line current',
  'Y Phase Line current',
  'B Phase Line current',
  'Neutral Line current'],
 ['R Phase Active Current',
  'Y Phase Active Current',
  'B Phase Active Current'],
 ['R Phase Reactive Current',
  'Y Phase Reactive Current',
  'B Phase Reactive Current'],
 ['R- Phase Active Power',
  'Y- Phase Active Power',
  'B- Phase Active Power',
  '3 Phase Active Power'],
 ['R- Phase Reactive Power',
  'Y- Phase Reactive Power',
  'B- Phase Reactive Power',
  '3 Phase Reactive Power'],
 ['R- Phase Apparent Power',
  'Y- Phase Apparent Power',
  'B- Phase Apparent Power',
  '3 Phase Apparent  Power']]
```

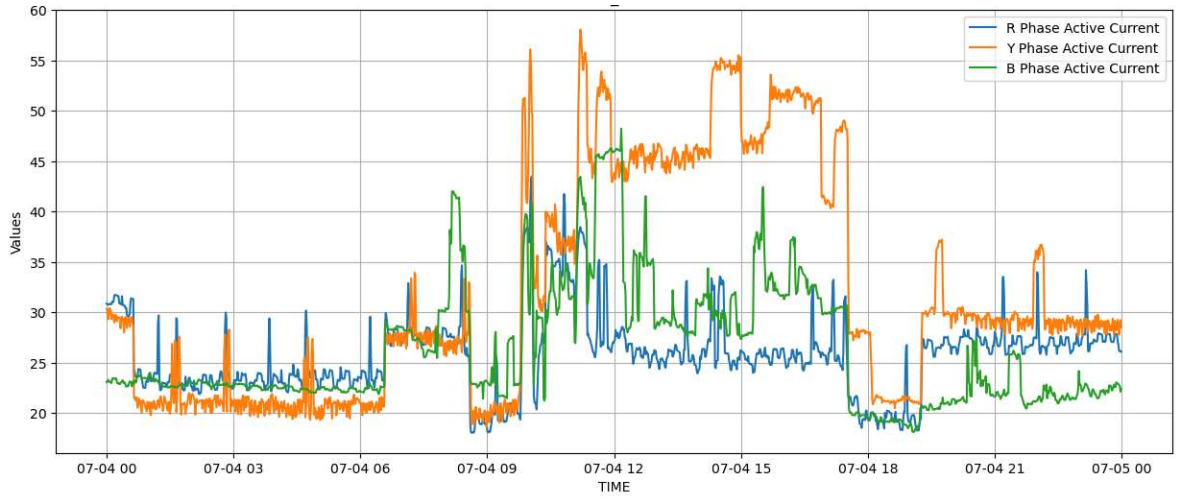
In [27]:
```
#Plotting multiple line charts
plt.figure(figsize=(10, 8))
for group in grouped_lists:
    plt.figure(figsize=(15,6),facecolor='white')
    plt.grid(True)
    plt.plot(data_minute_avg.index,data_minute_avg[group],label=group)
    plt.xlabel('TIME')
    plt.ylabel('Values')
    plt.title('_')
    plt.legend()
    plt.show()
```

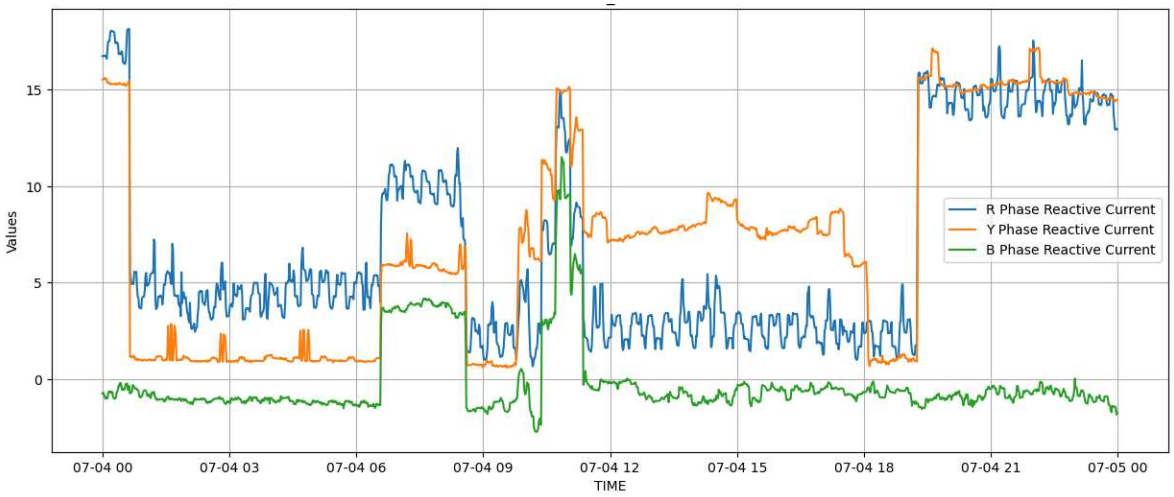
```
<Figure size 1000x800 with 0 Axes>
```

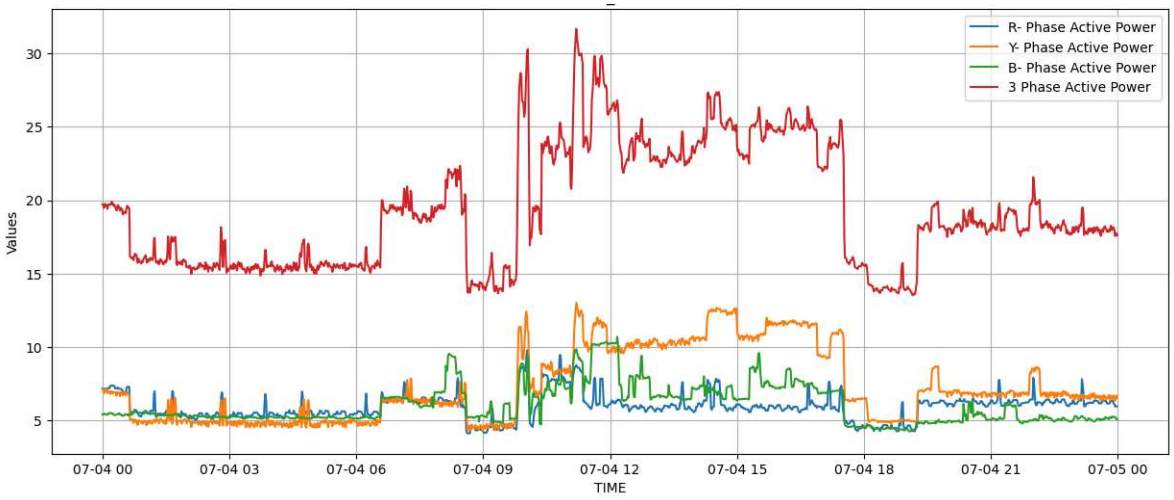


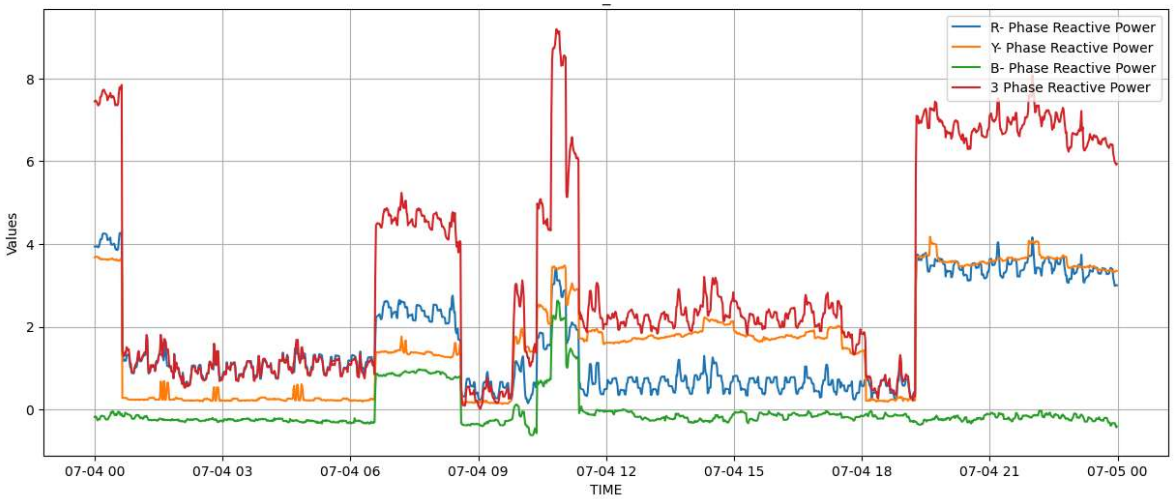

RY Line Voltage
YB Line Voltage
BR Line Voltage
Values
TIME
420
415
410
405
400
395
390
385
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00

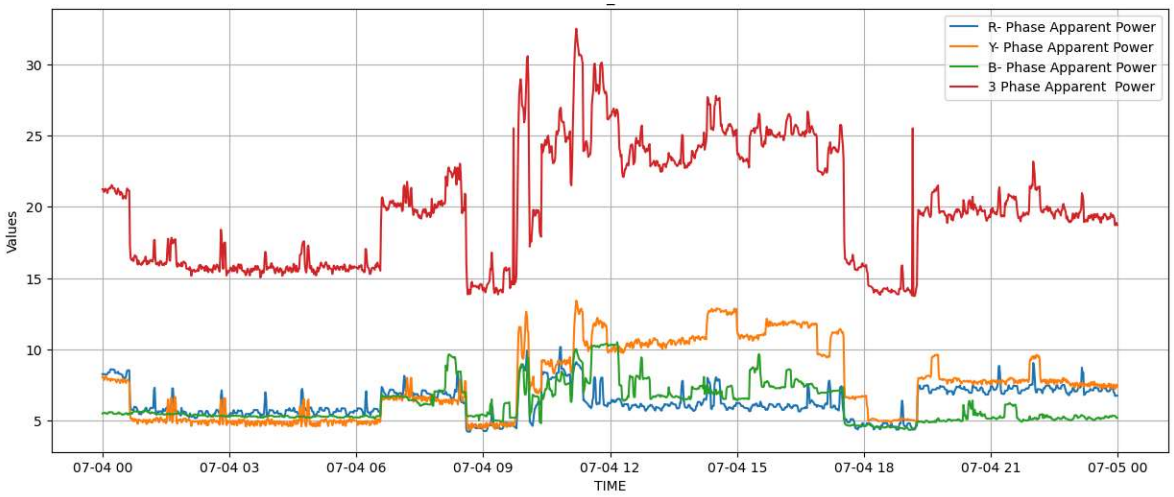

R Phase Line current
Y Phase Line current
B Phase Line current
Neutral Line current
Values
TIME
60
50
40
30
20
10
0
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00


R Phase Active Current
Y Phase Active Current
B Phase Active Current
Values
TIME
60
55
50
45
40
35
30
25
20
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00

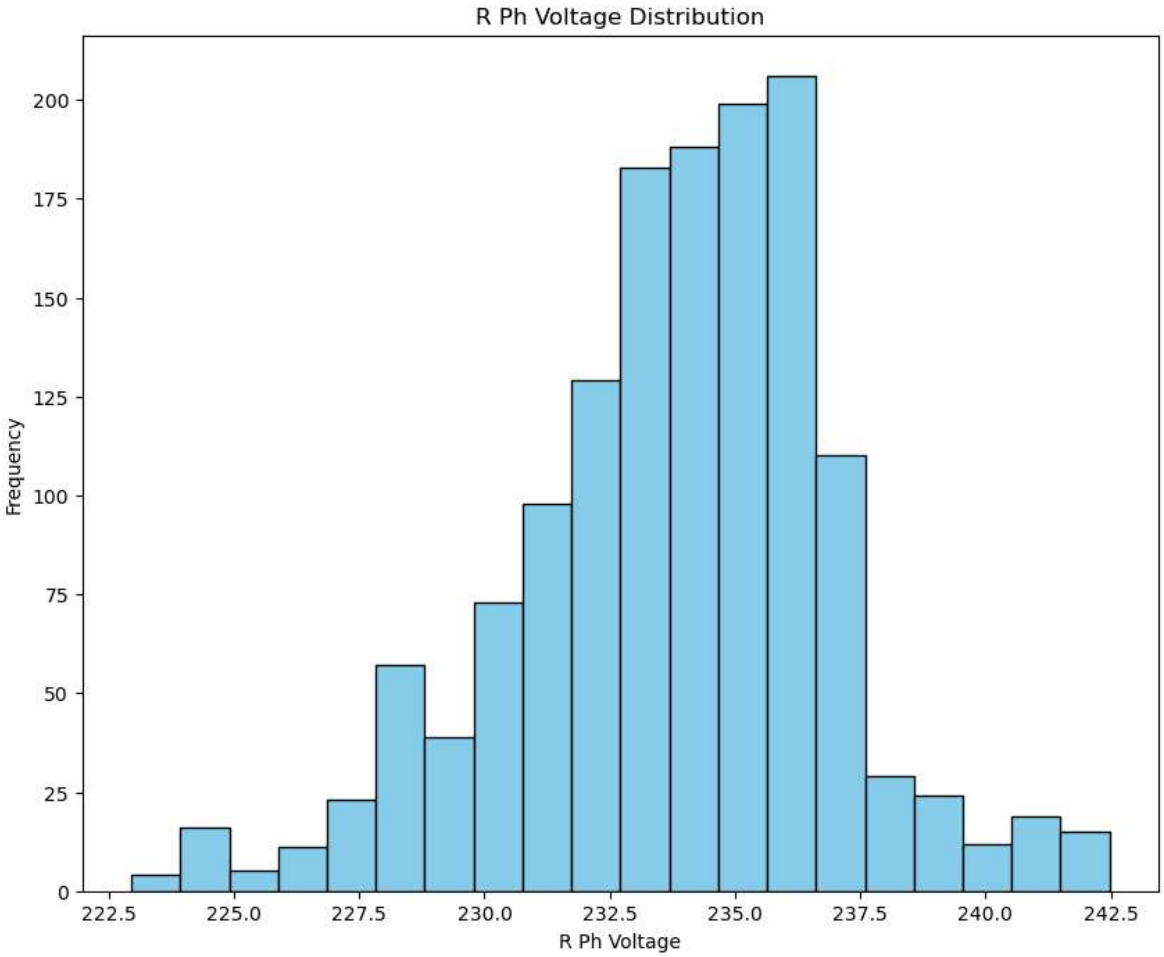

R Phase Reactive Current
Y Phase Reactive Current
B Phase Reactive Current
Values
TIME
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00

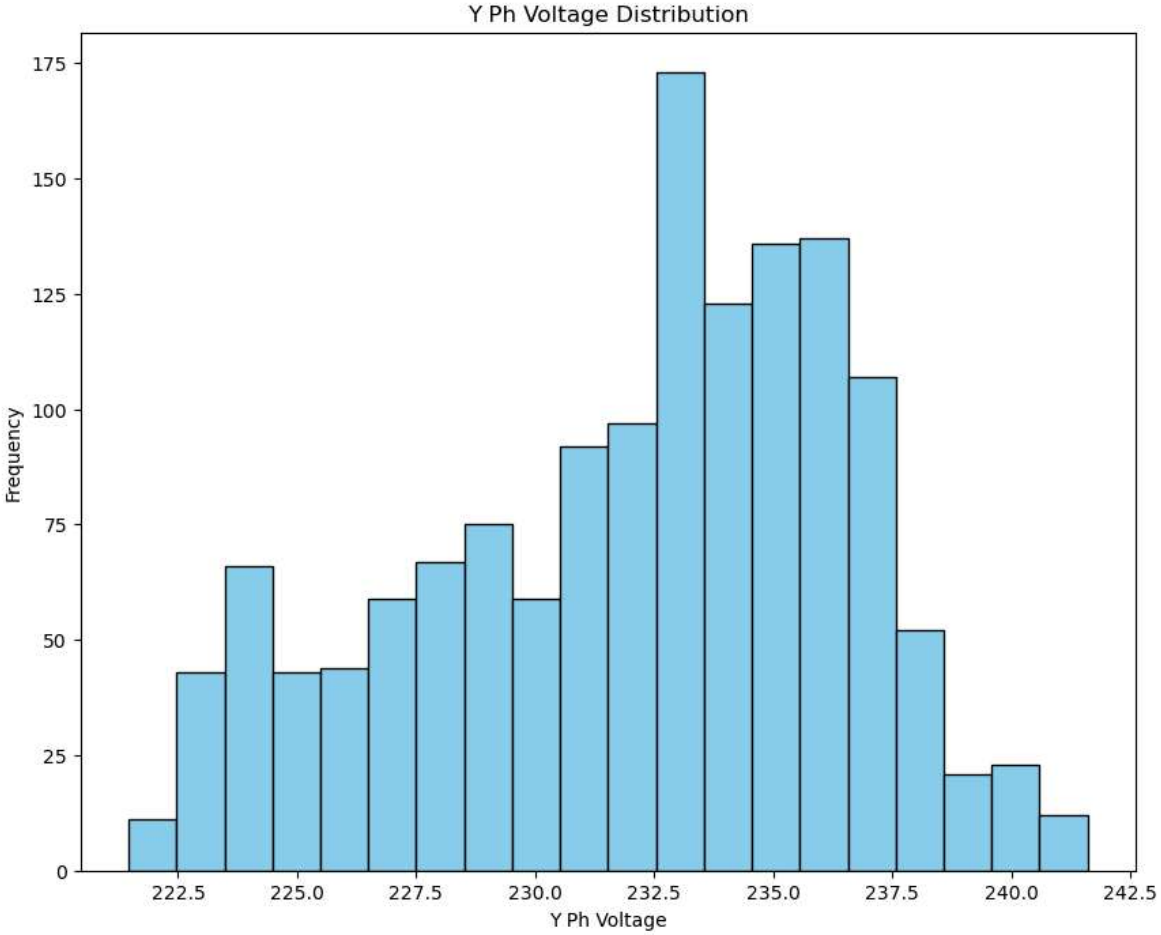

R- Phase Active Power
Y- Phase Active Power
B- Phase Active Power
3 Phase Active Power
Values
TIME
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00

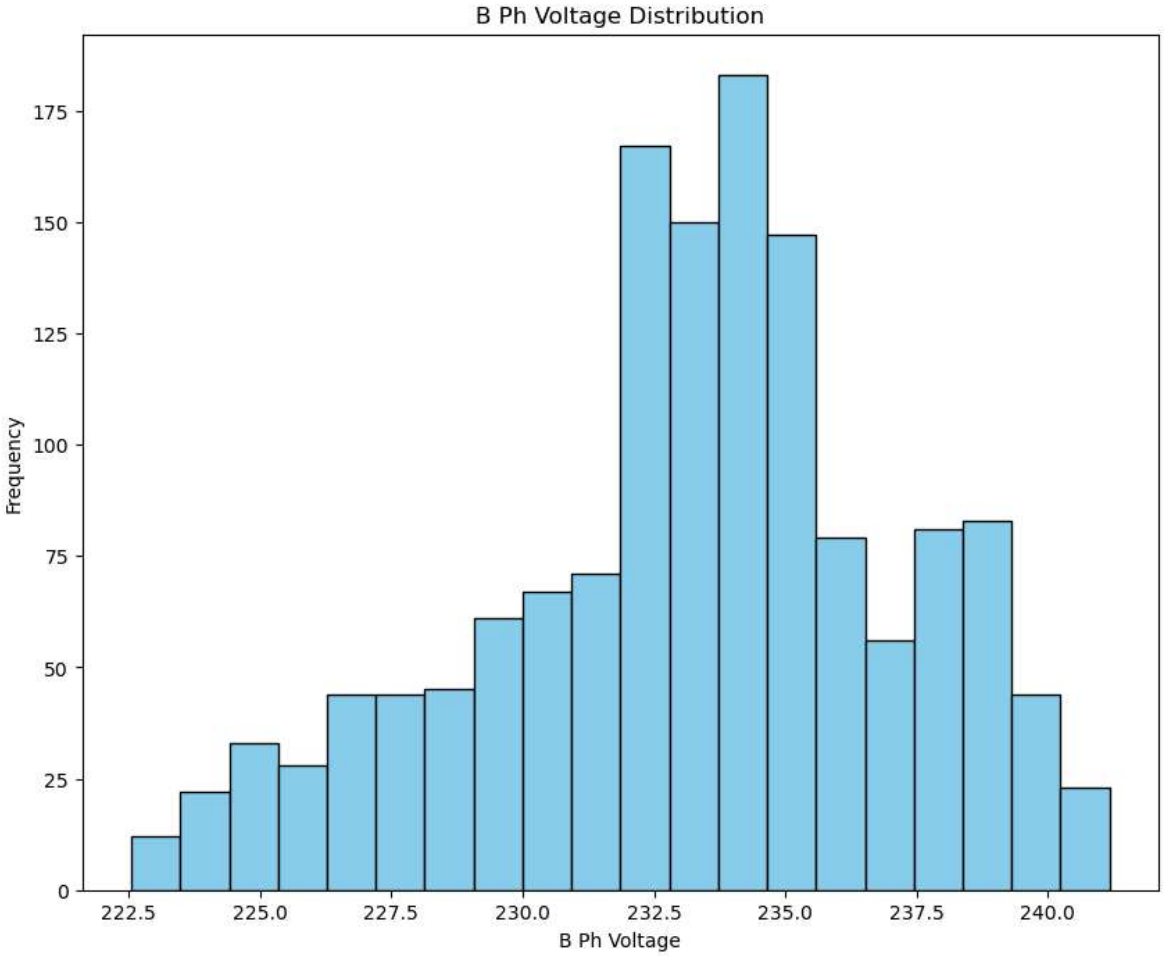

R- Phase Reactive Power
Y- Phase Reactive Power
B- Phase Reactive Power
3 Phase Reactive Power
Values
TIME
07-04 00
07-04 03
07-04 06
07-04 09
07-04 12
07-04 15
07-04 18
07-04 21
07-05 00

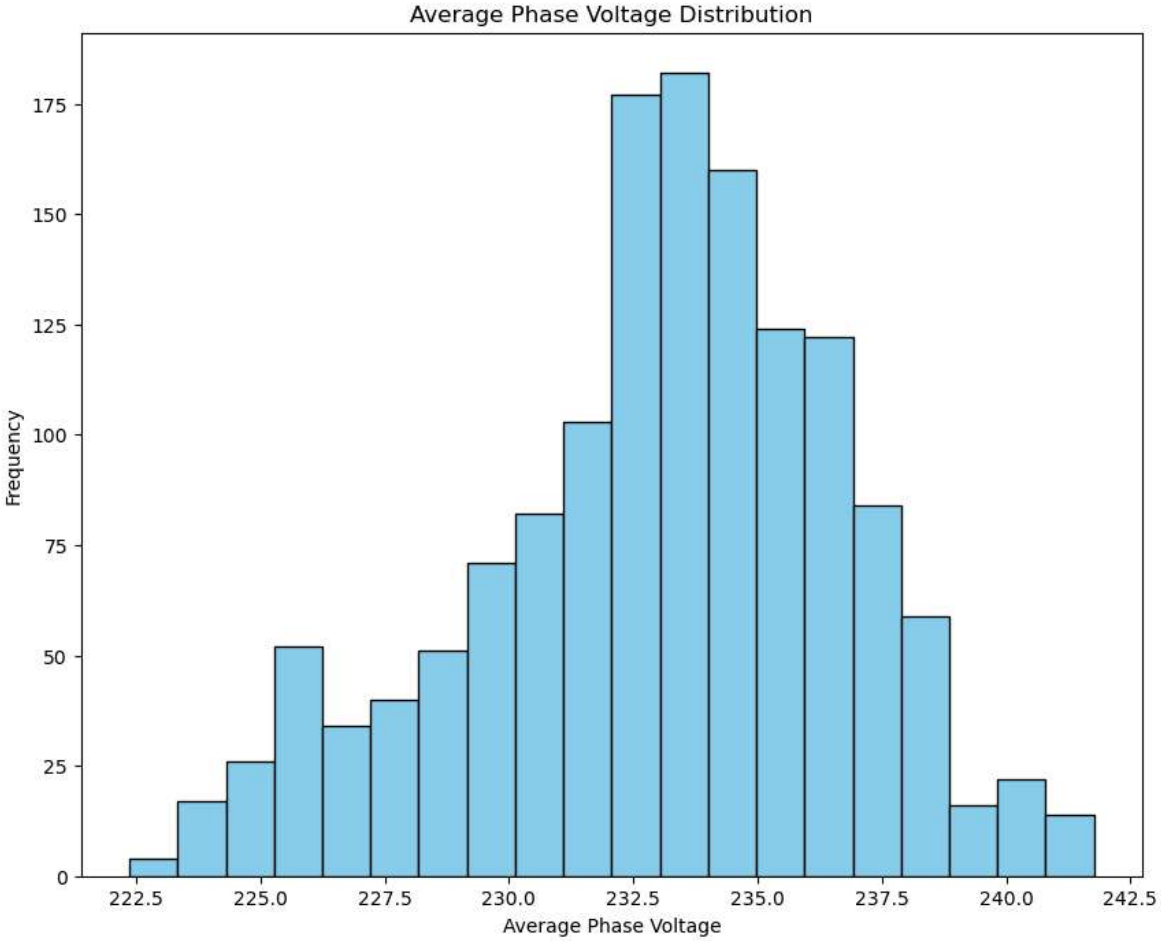


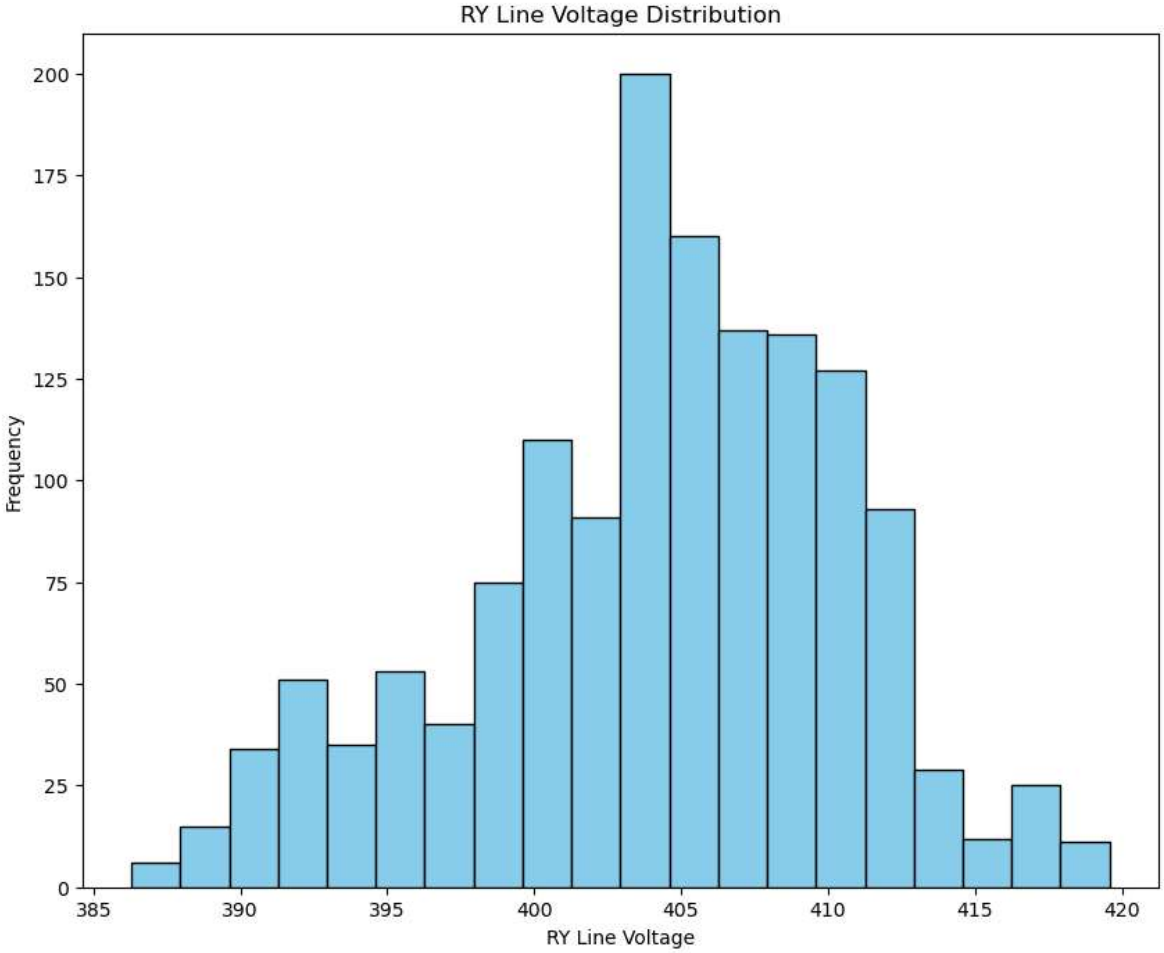
In [10]:
```
# Iterate over each column of the data_minute_avg DataFrame
for col in data_minute_avg.columns:
    plt.figure(figsize=(10, 8))
    plt.hist(data_minute_avg[col], bins=20, color='skyblue', edgecolor='black')
    tit = col + " Distribution"
    plt.title(tit)
    plt.xlabel(col)
    plt.ylabel("Frequency")
    plt.show()
```

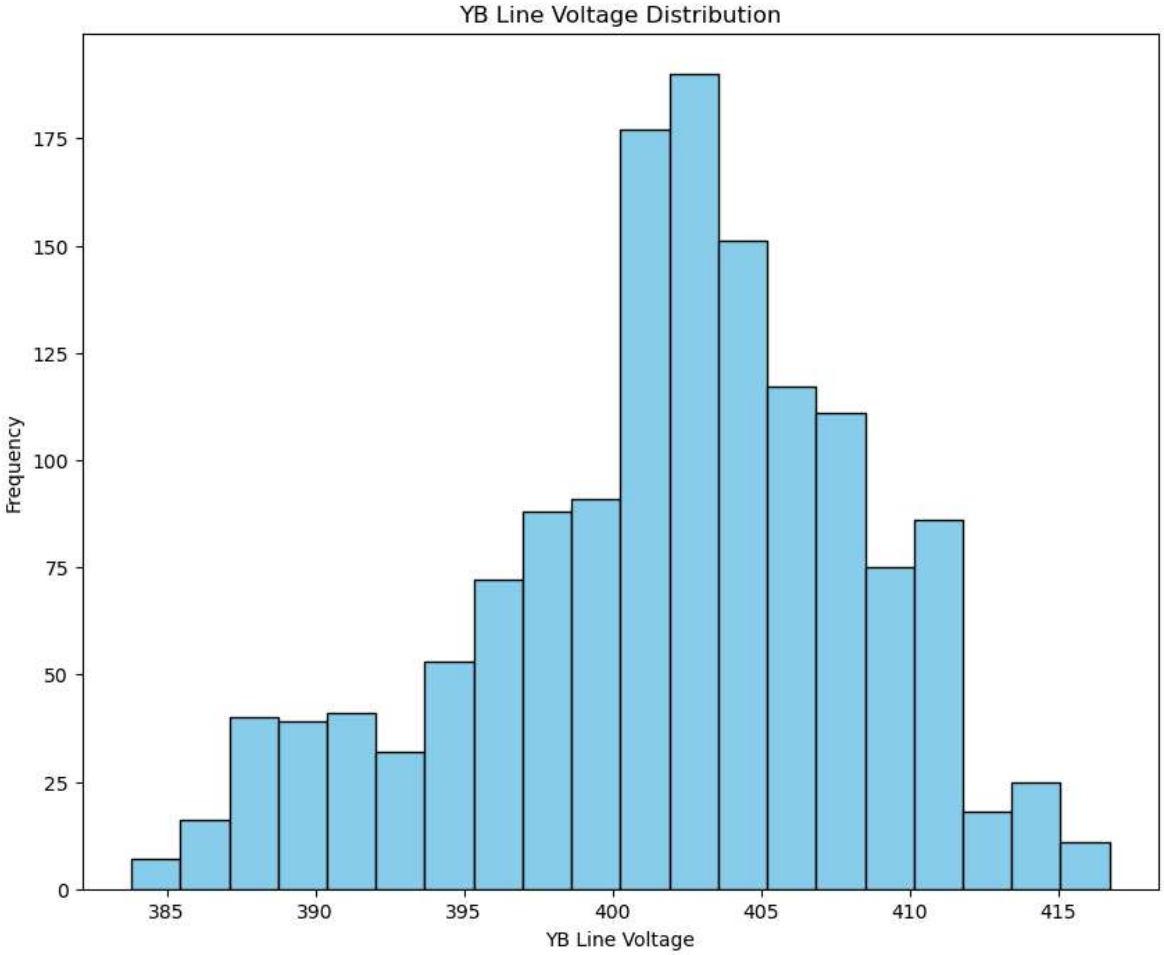


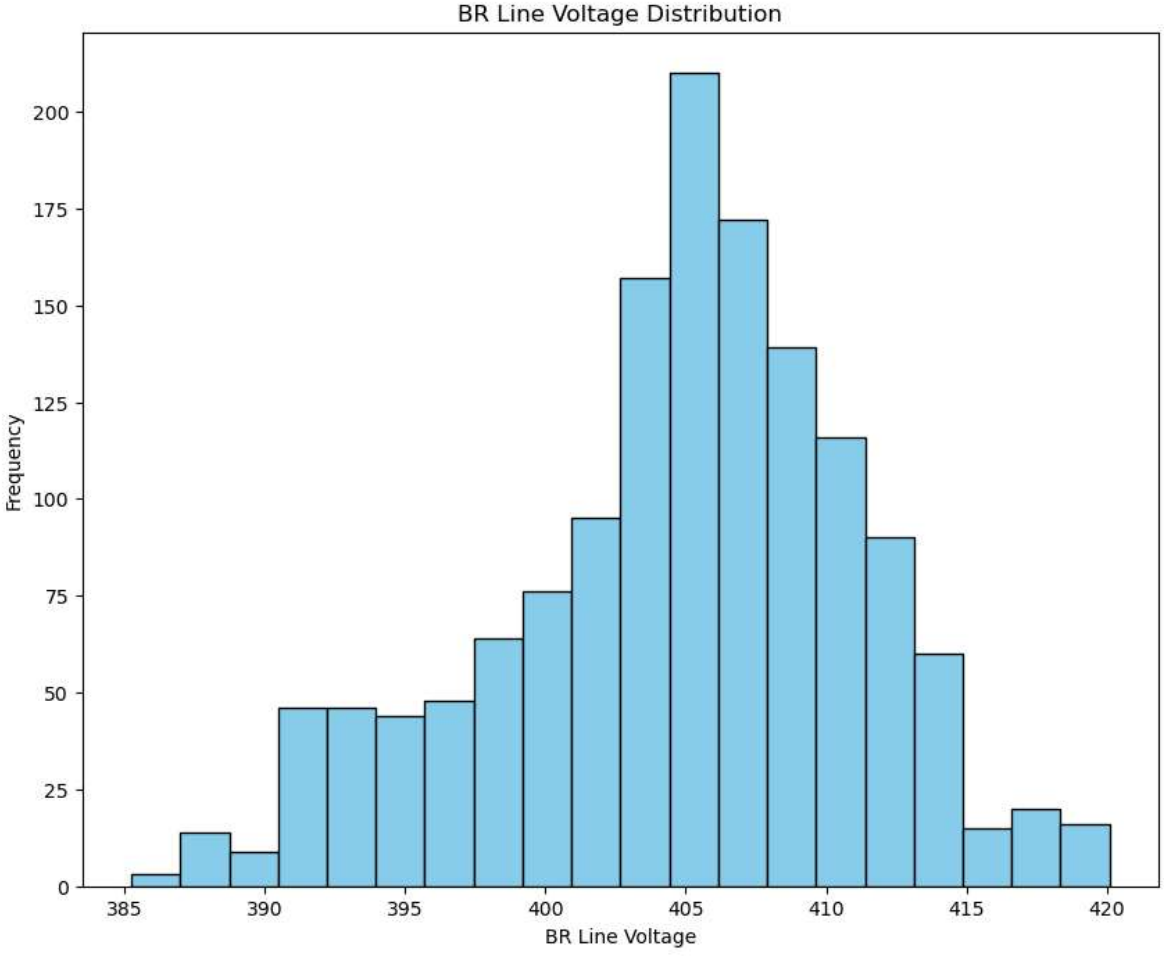

Y Ph Voltage Distribution
Frequency
Y Ph Voltage
0
25
50
75
100
125
150
175
222.5
225.0
227.5
230.0
232.5
235.0
237.5
240.0
242.5

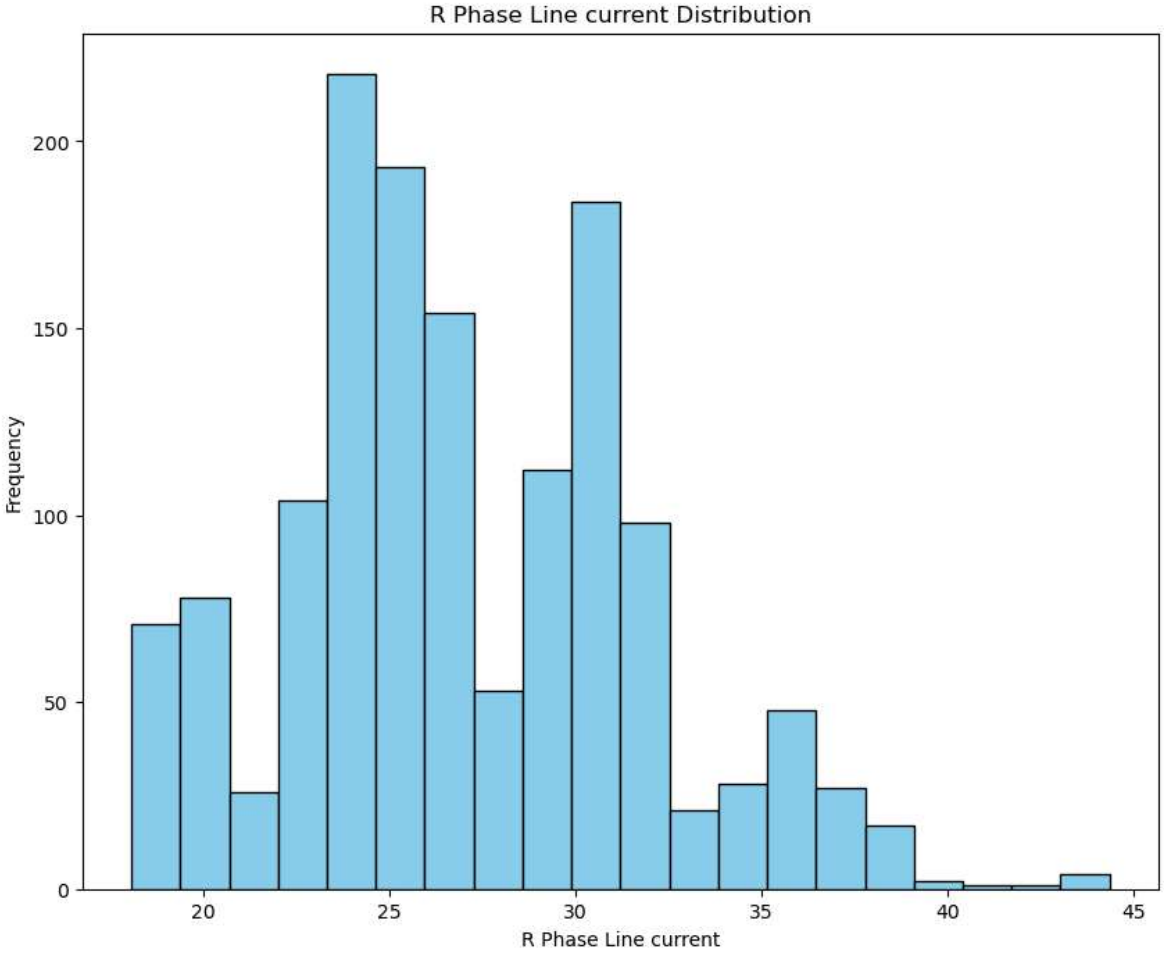

B Ph Voltage Distribution
Frequency
B Ph Voltage
0
25
50
75
100
125
150
175
222.5
225.0
227.5
230.0
232.5
235.0
237.5
240.0

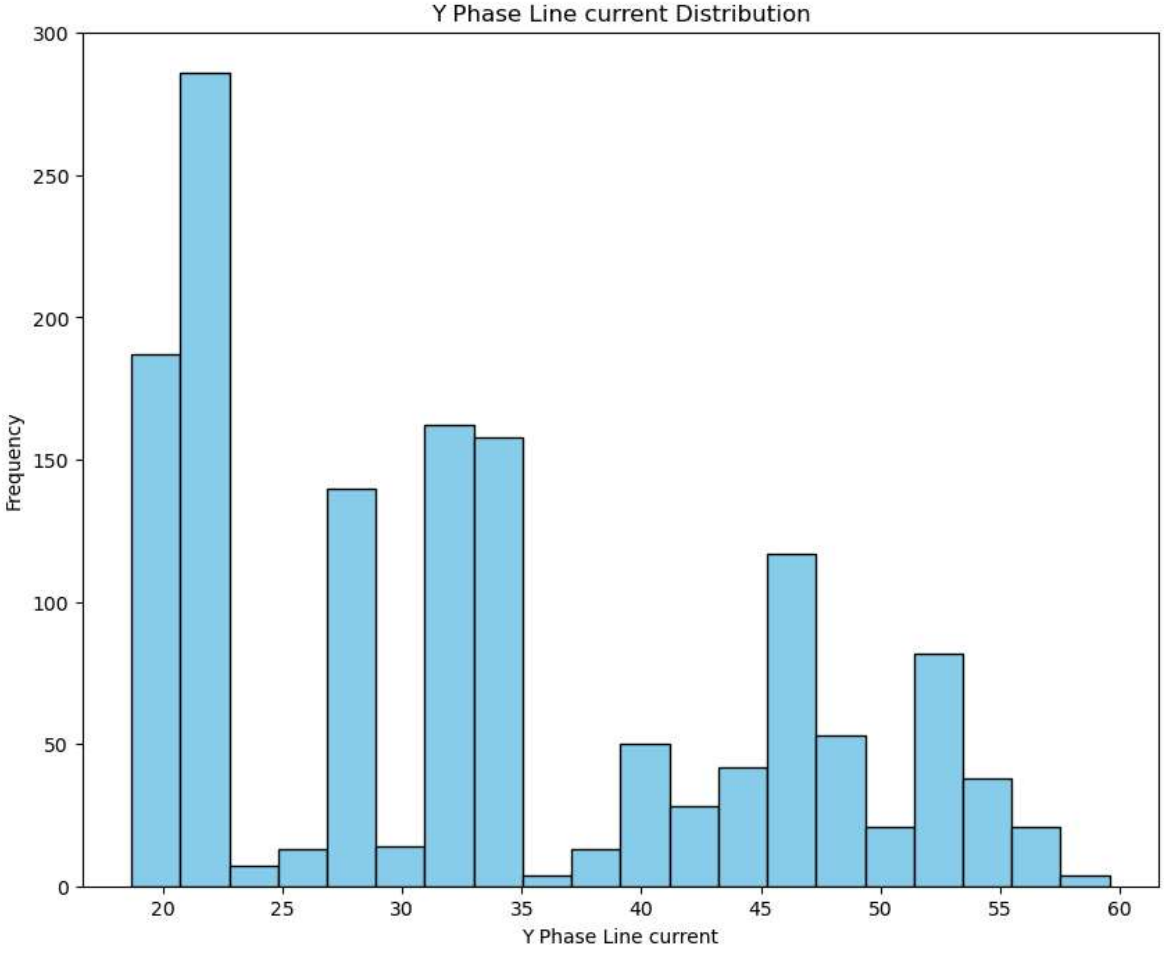

Average Phase Voltage Distribution
Frequency
Average Phase Voltage
0
25
50
75
100
125
150
175
222.5
225.0
227.5
230.0
232.5
235.0
237.5
240.0
242.5

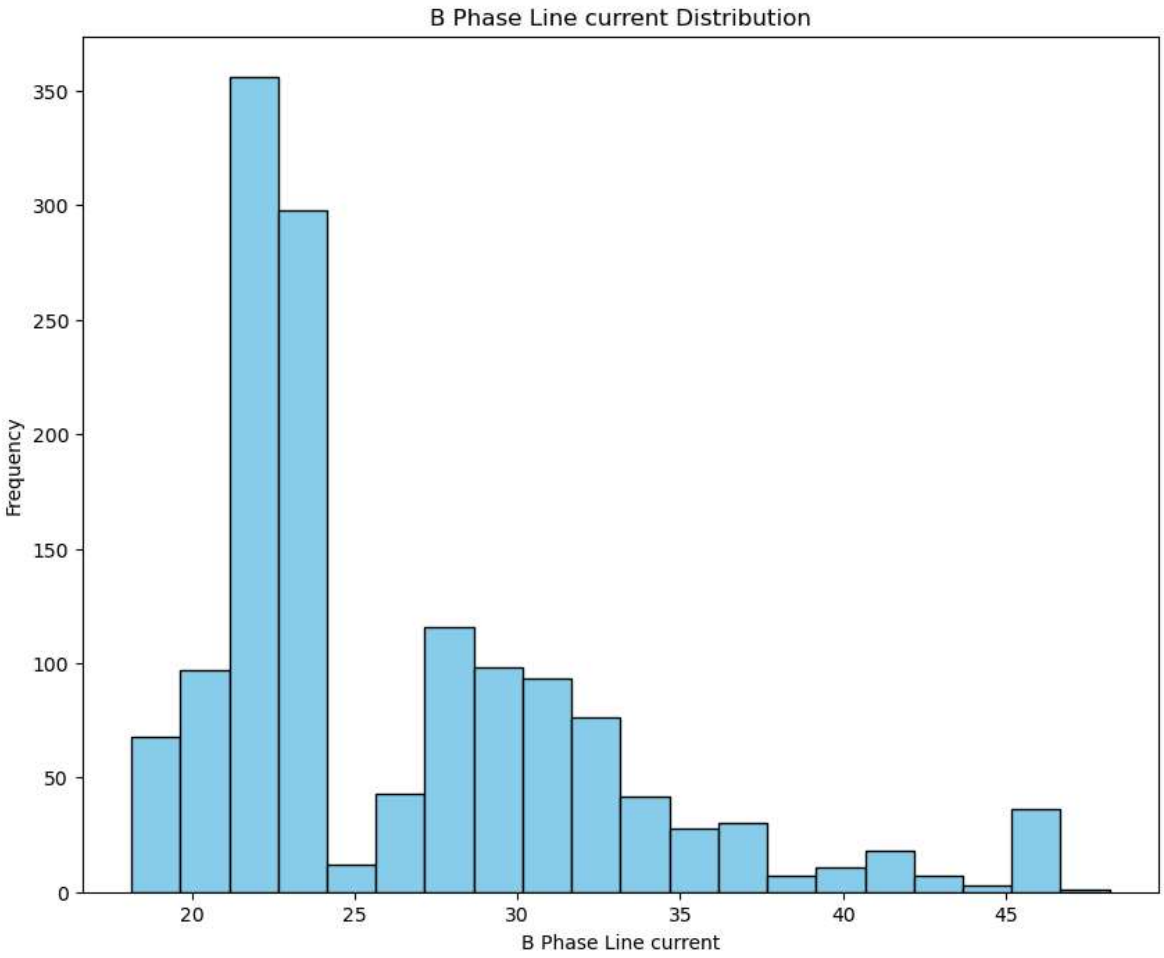

RY Line Voltage Distribution
Frequency
RY Line Voltage
0
25
50
75
100
125
150
175
200
385
390
395
400
405
410
415
420

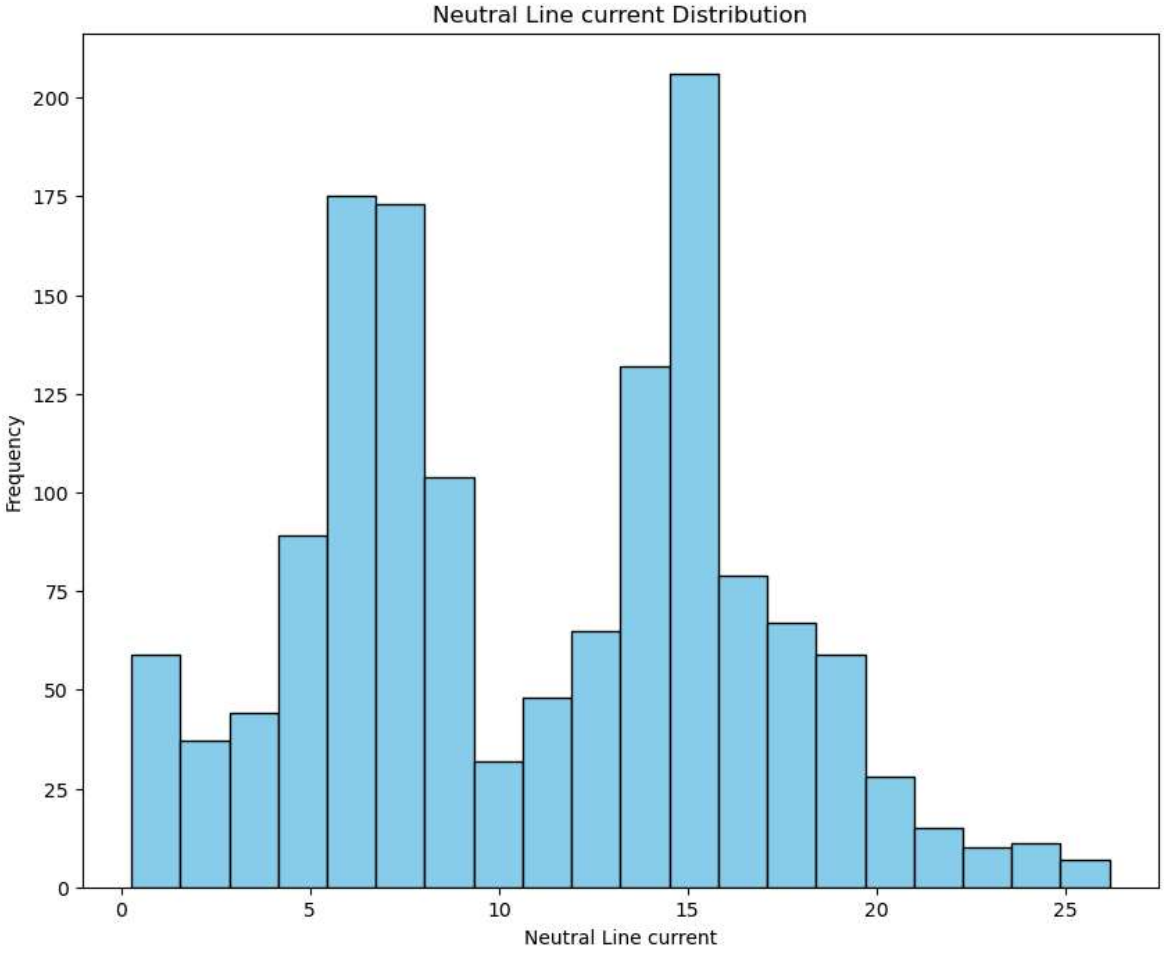

YB Line Voltage Distribution
Frequency
YB Line Voltage
0
25
50
75
100
125
150
175
385
390
395
400
405
410
415

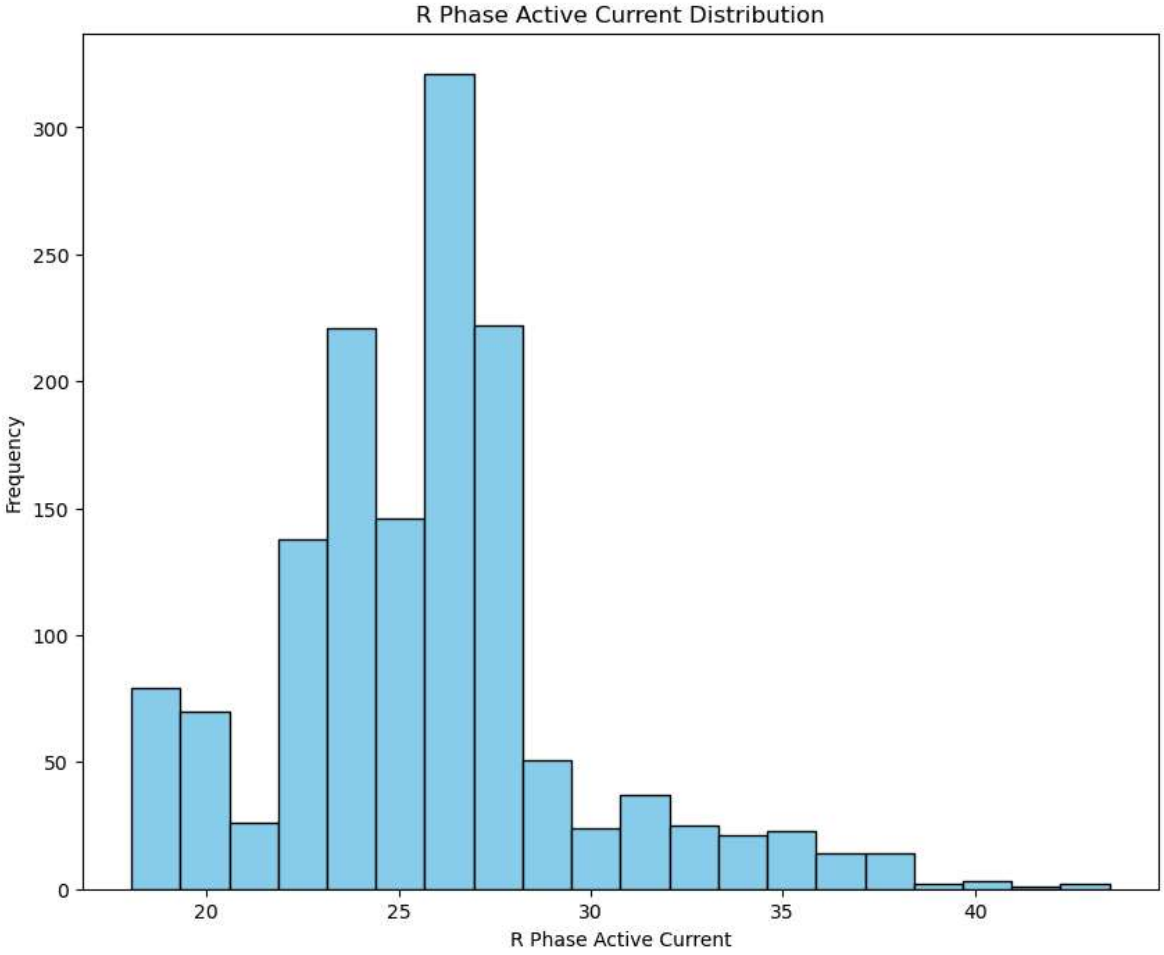

BR Line Voltage Distribution
Frequency
BR Line Voltage
0
25
50
75
100
125
150
175
200
385
390
395
400
405
410
415
420

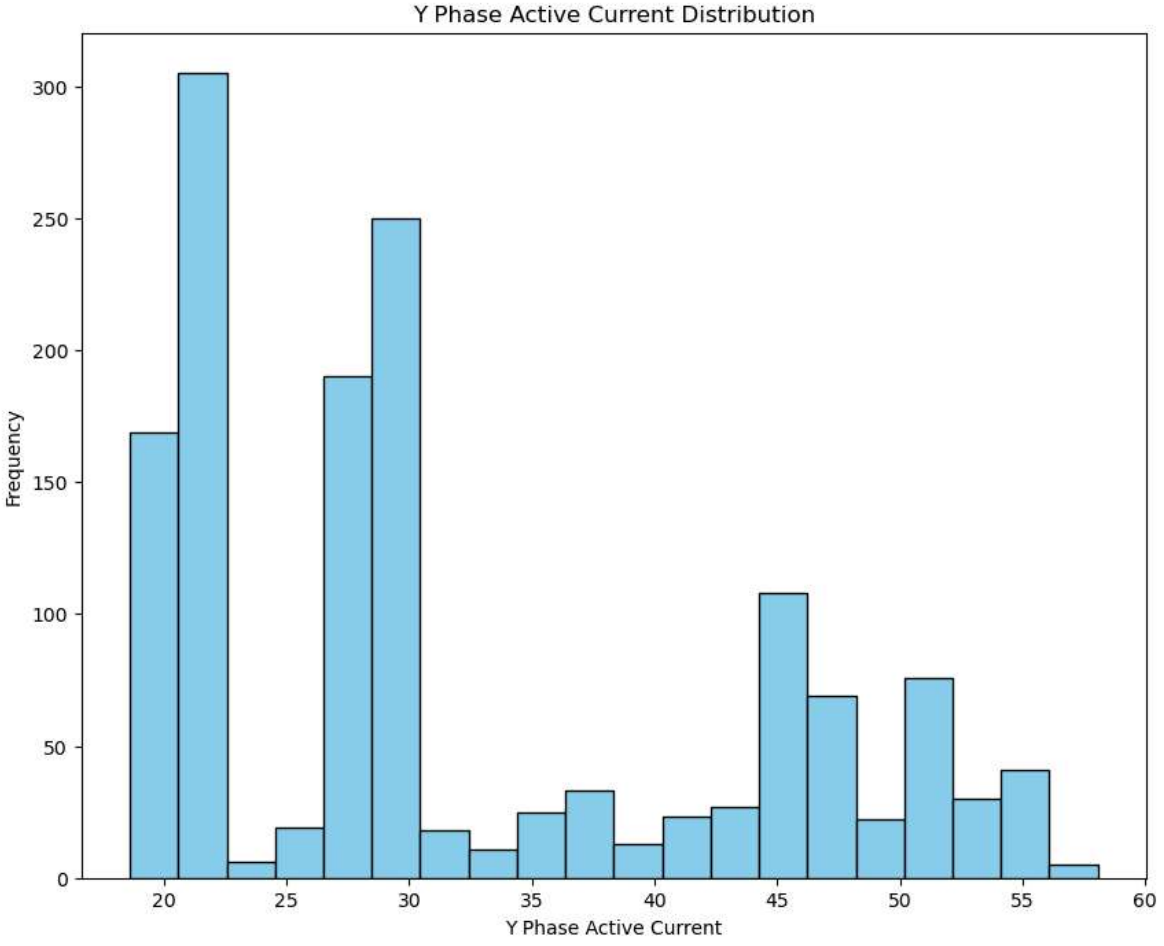

R Phase Line current Distribution
Frequency
R Phase Line current
0
50
100
150
200
20
25
30
35
40
45

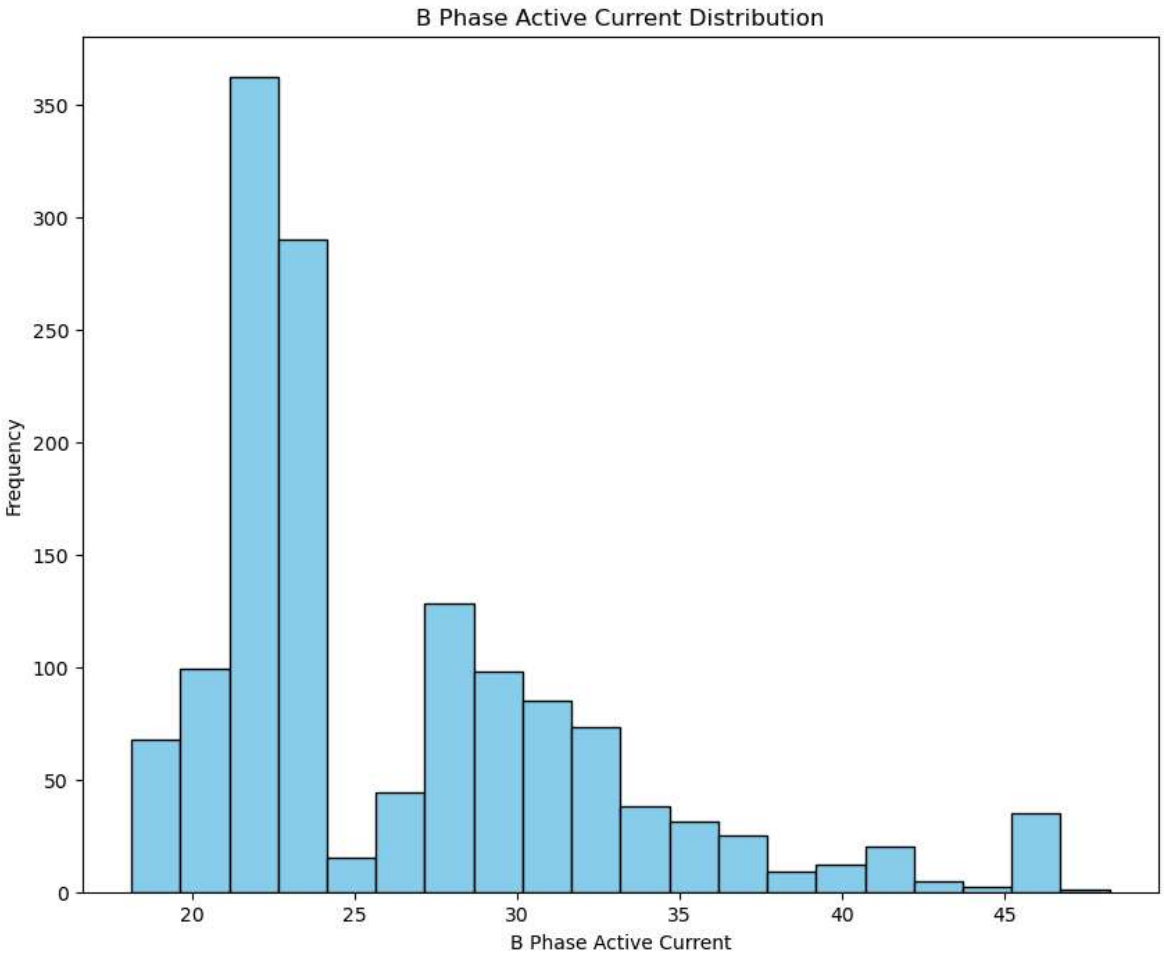

Y Phase Line current Distribution
Frequency
Y Phase Line current
0
50
100
150
200
250
300
20
25
30
35
40
45
50
55
60

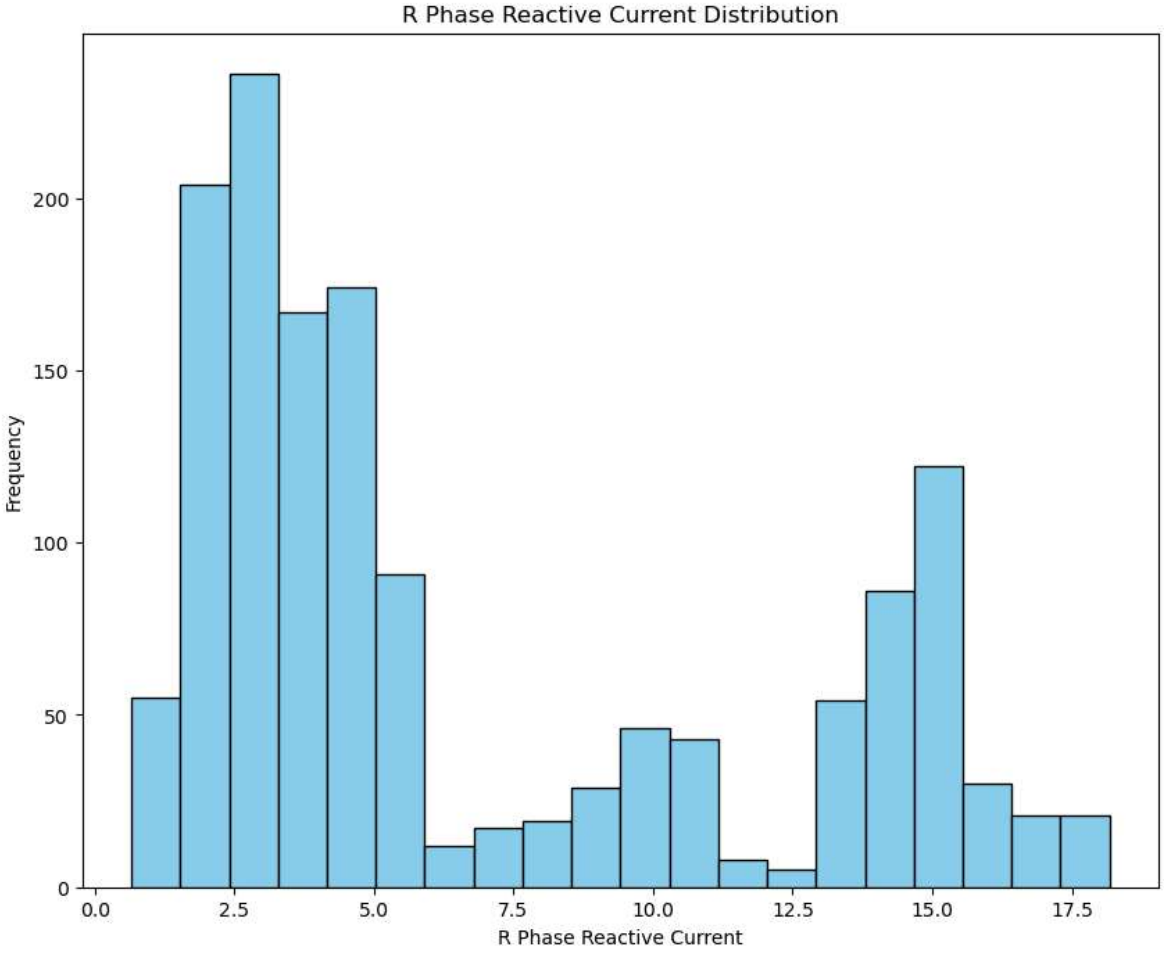

B Phase Line current Distribution
Frequency
B Phase Line current
0
50
100
150
200
250
300
350
20
25
30
35
40
45

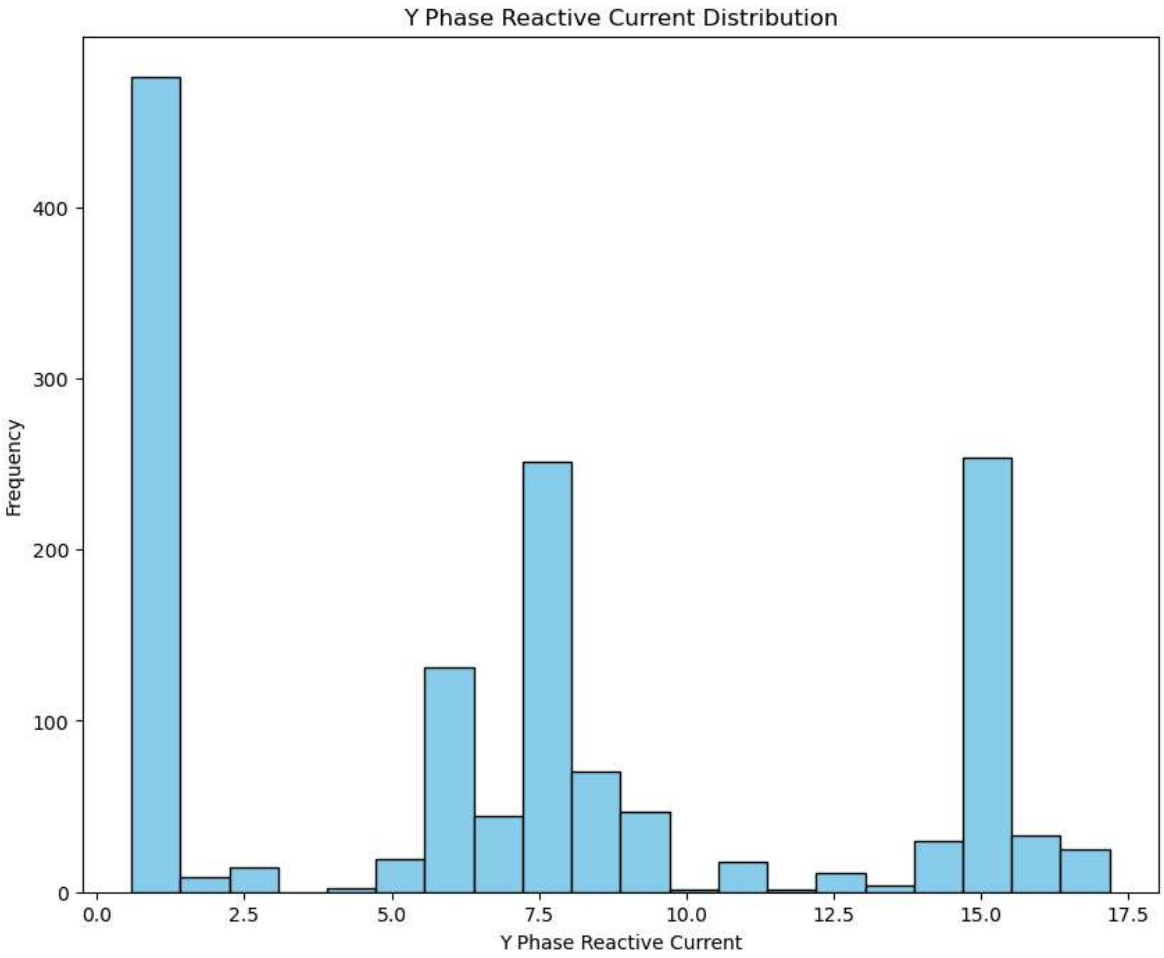

Neutral Line current Distribution
Frequency
Neutral Line current
0
25
50
75
100
125
150
175
200
0
5
10
15
20
25

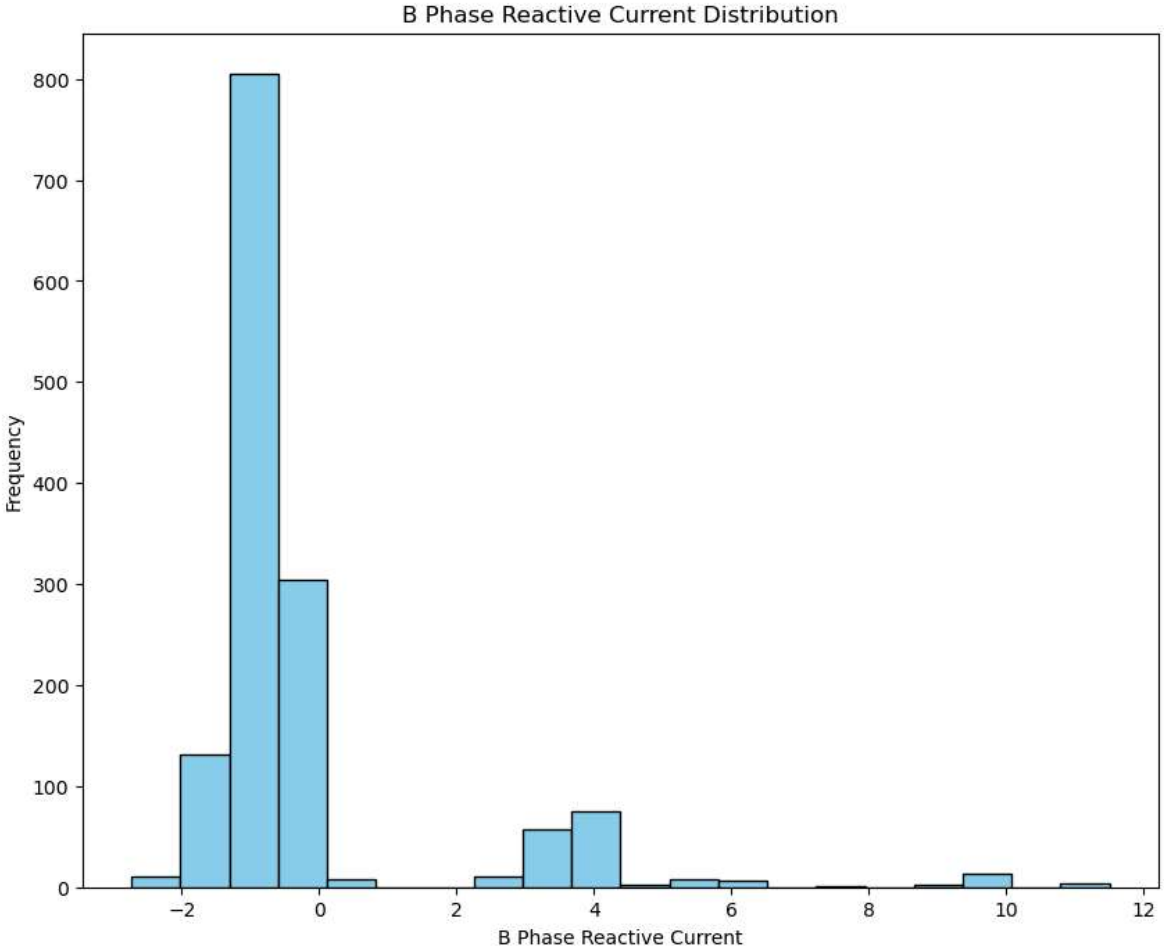

R Phase Active Current Distribution
Frequency
R Phase Active Current
0
50
100
150
200
250
300
20
25
30
35
40

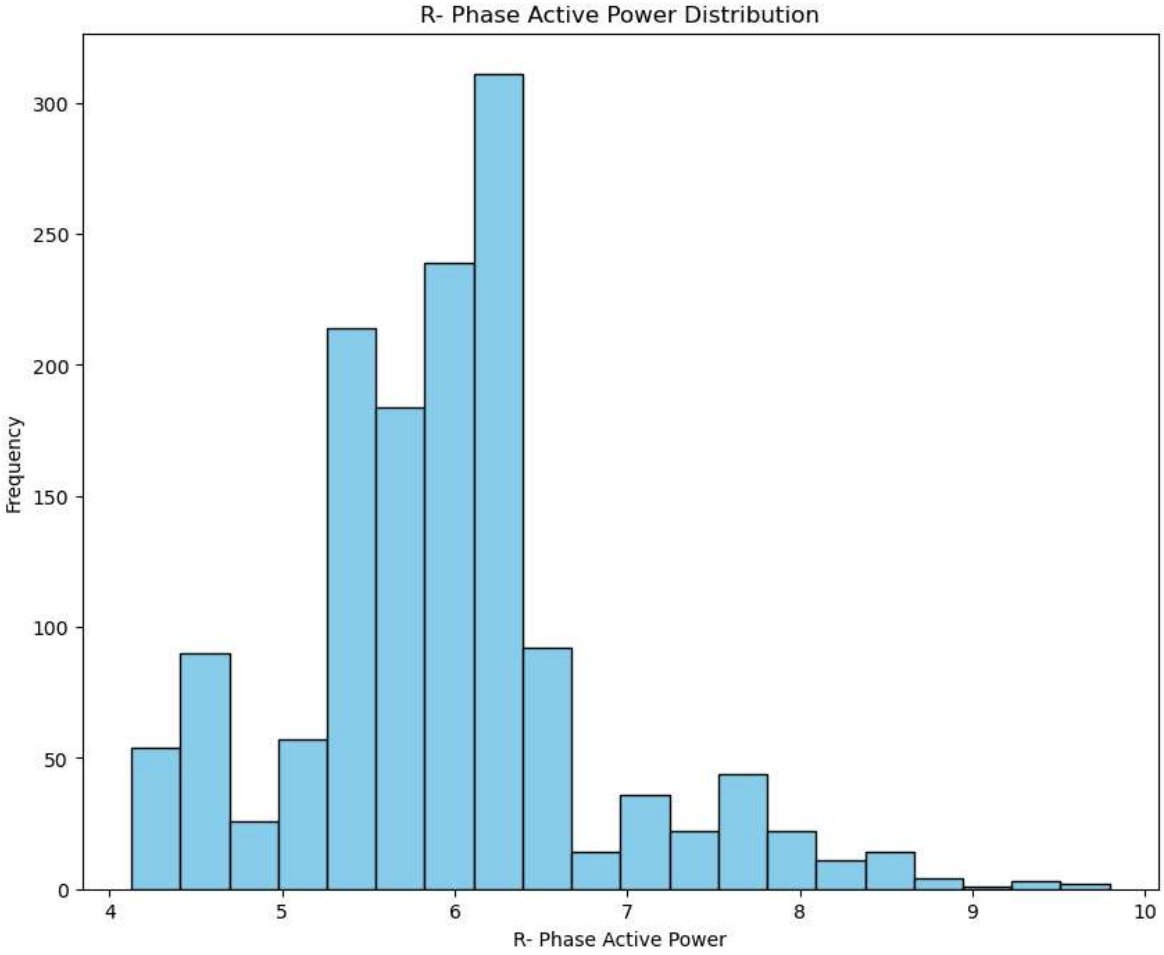

Y Phase Active Current Distribution
Frequency
Y Phase Active Current
0
50
100
150
200
250
300
20
25
30
35
40
45
50
55
60

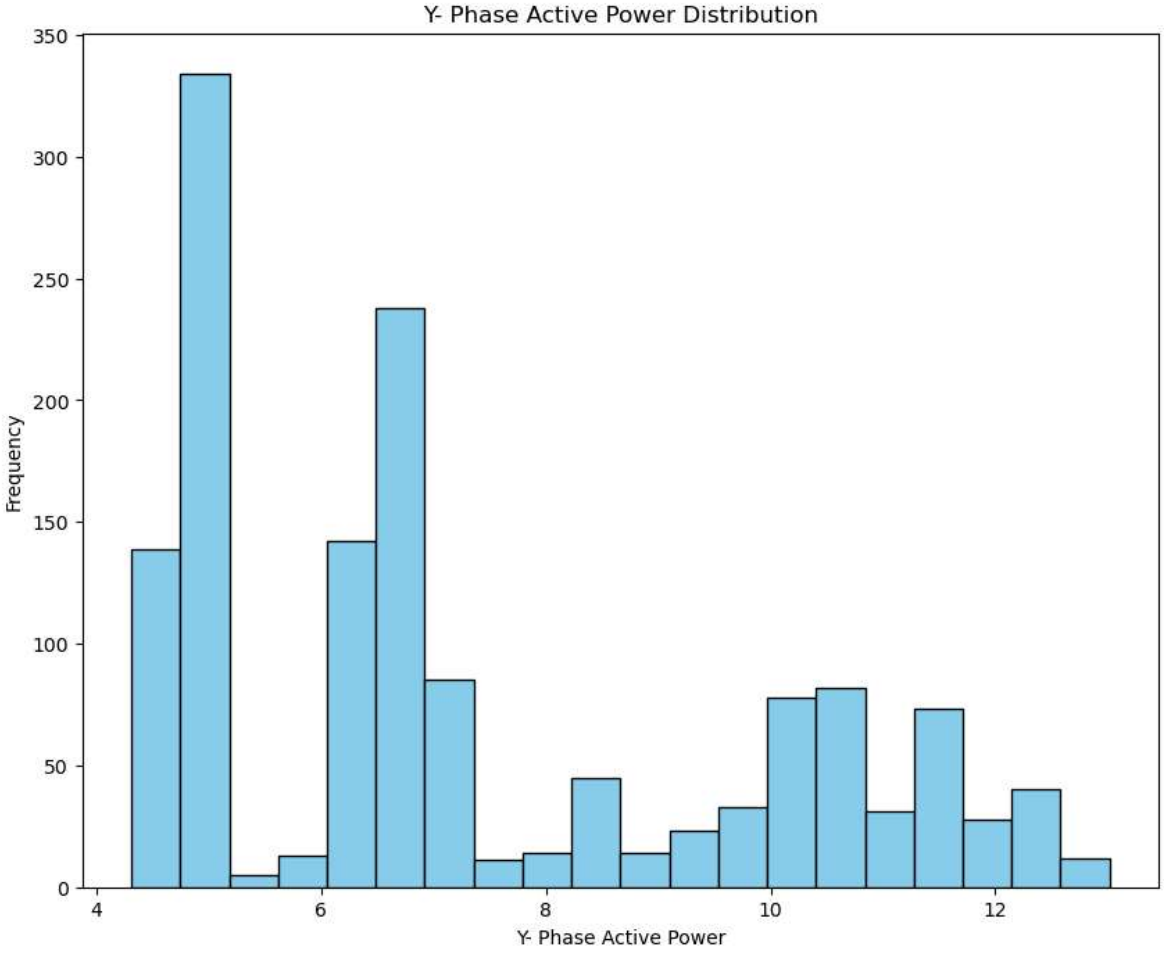

B Phase Active Current Distribution
Frequency
B Phase Active Current
0
50
100
150
200
250
300
350
20
25
30
35
40
45

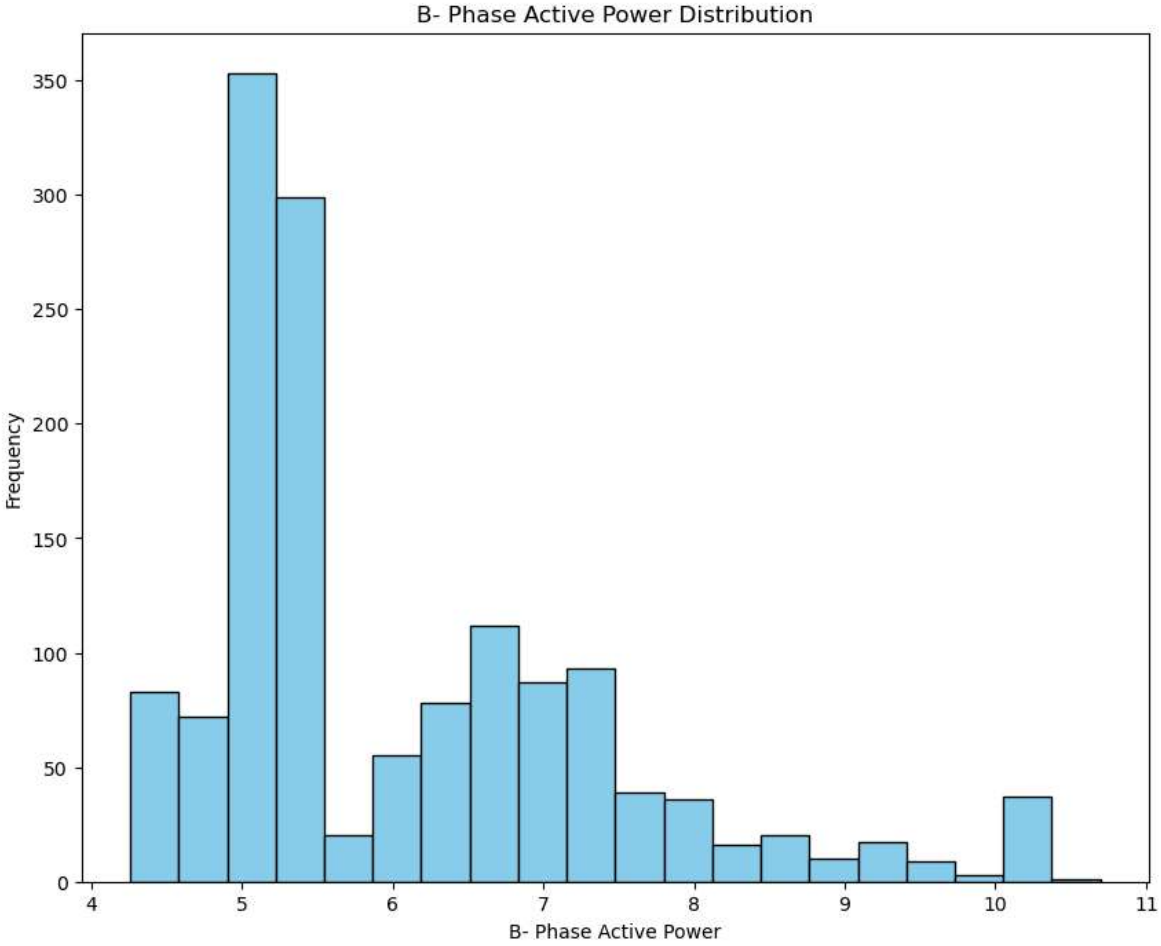

R Phase Reactive Current Distribution
Frequency
R Phase Reactive Current
0
50
100
150
200
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

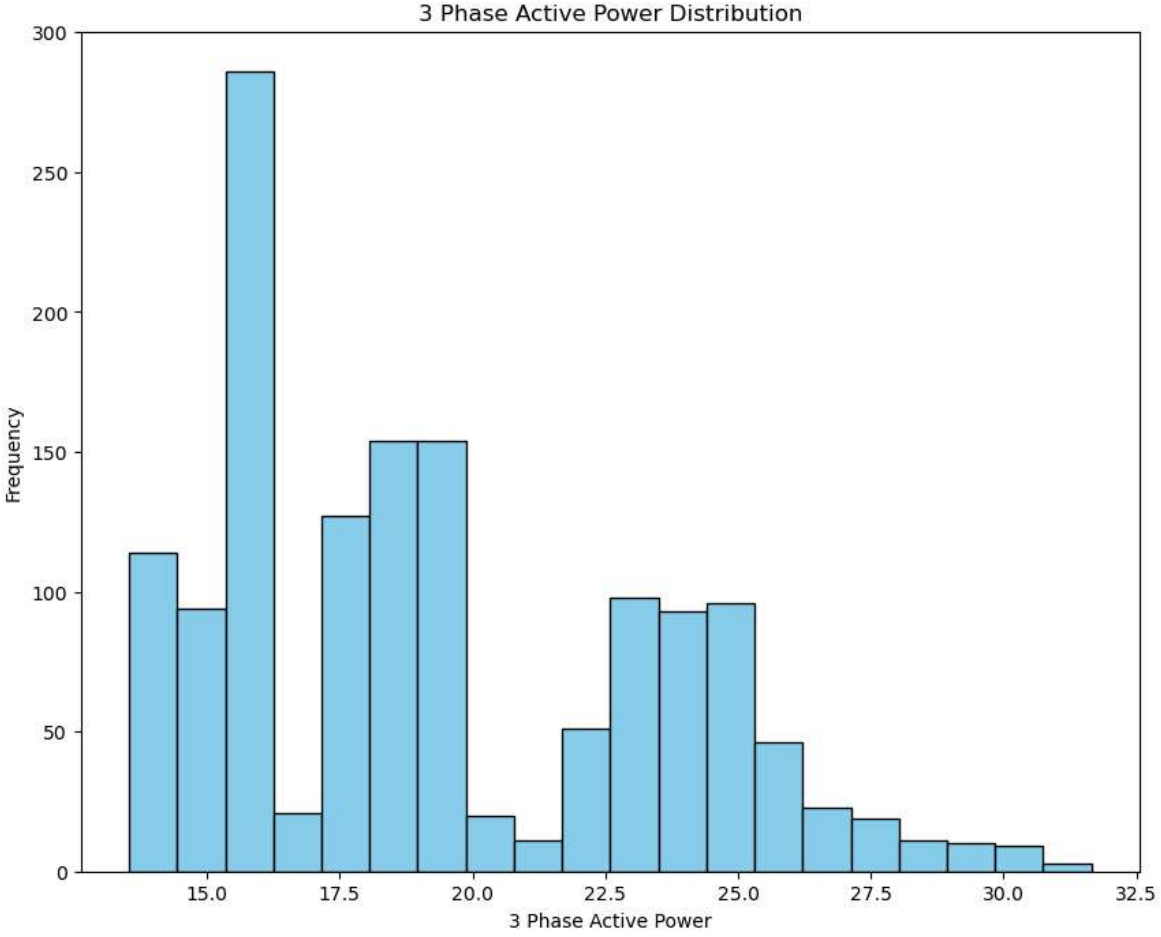

Y Phase Reactive Current Distribution
Frequency
Y Phase Reactive Current
0
100
200
300
400
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

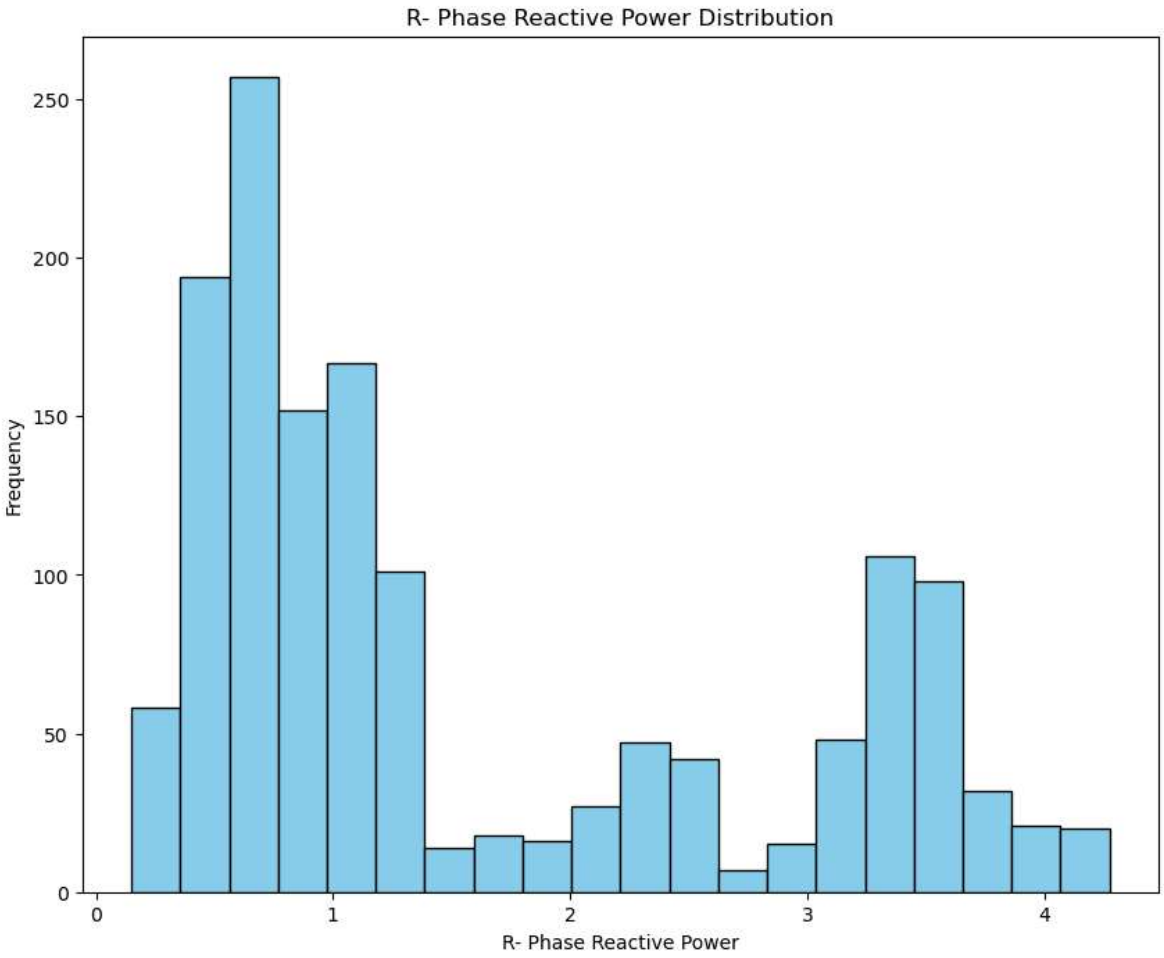

B Phase Reactive Current Distribution
Frequency
B Phase Reactive Current
0
100
200
300
400
500
600
700
800
−2
0
2
4
6
8
10
12

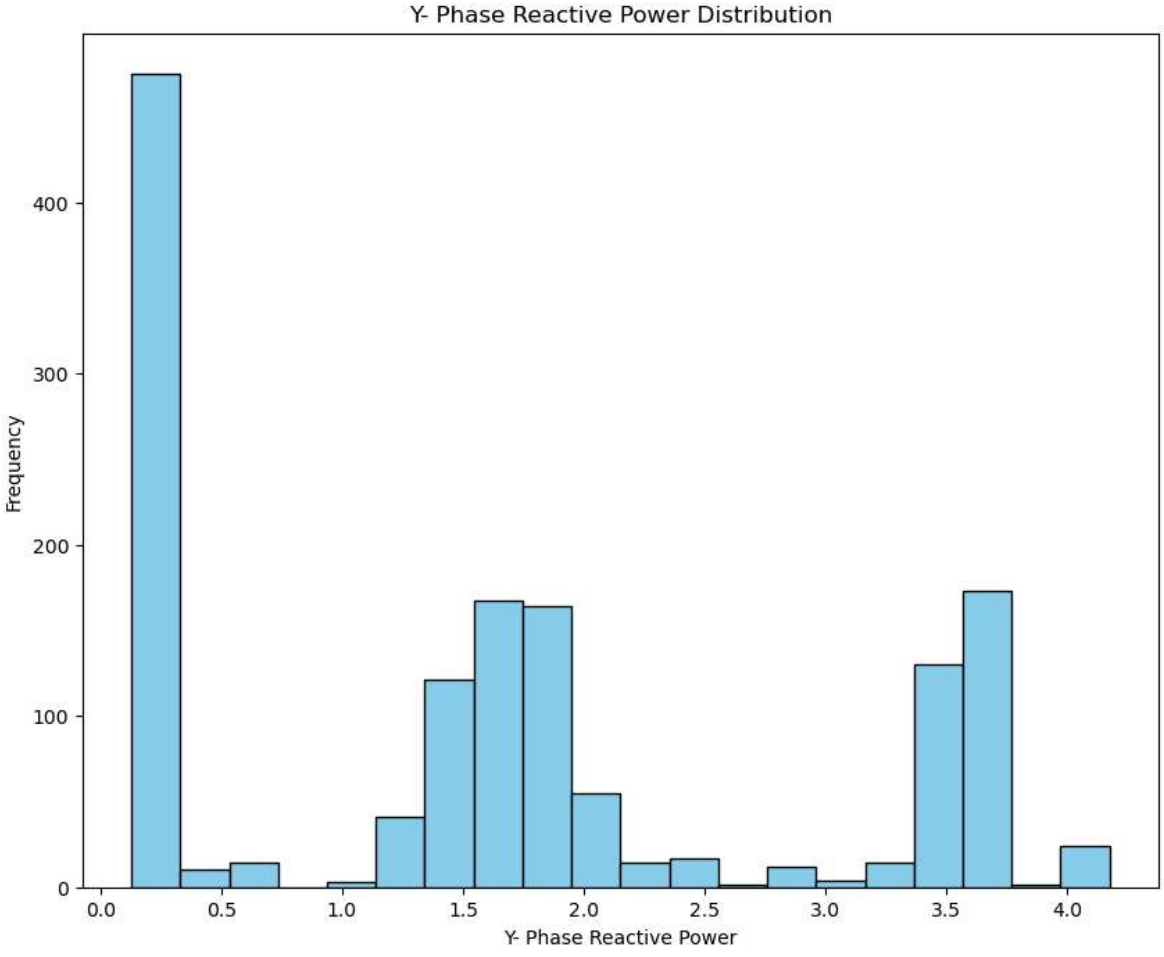

R- Phase Active Power Distribution
Frequency
R- Phase Active Power
0
50
100
150
200
250
300
4
5
6
7
8
9
10

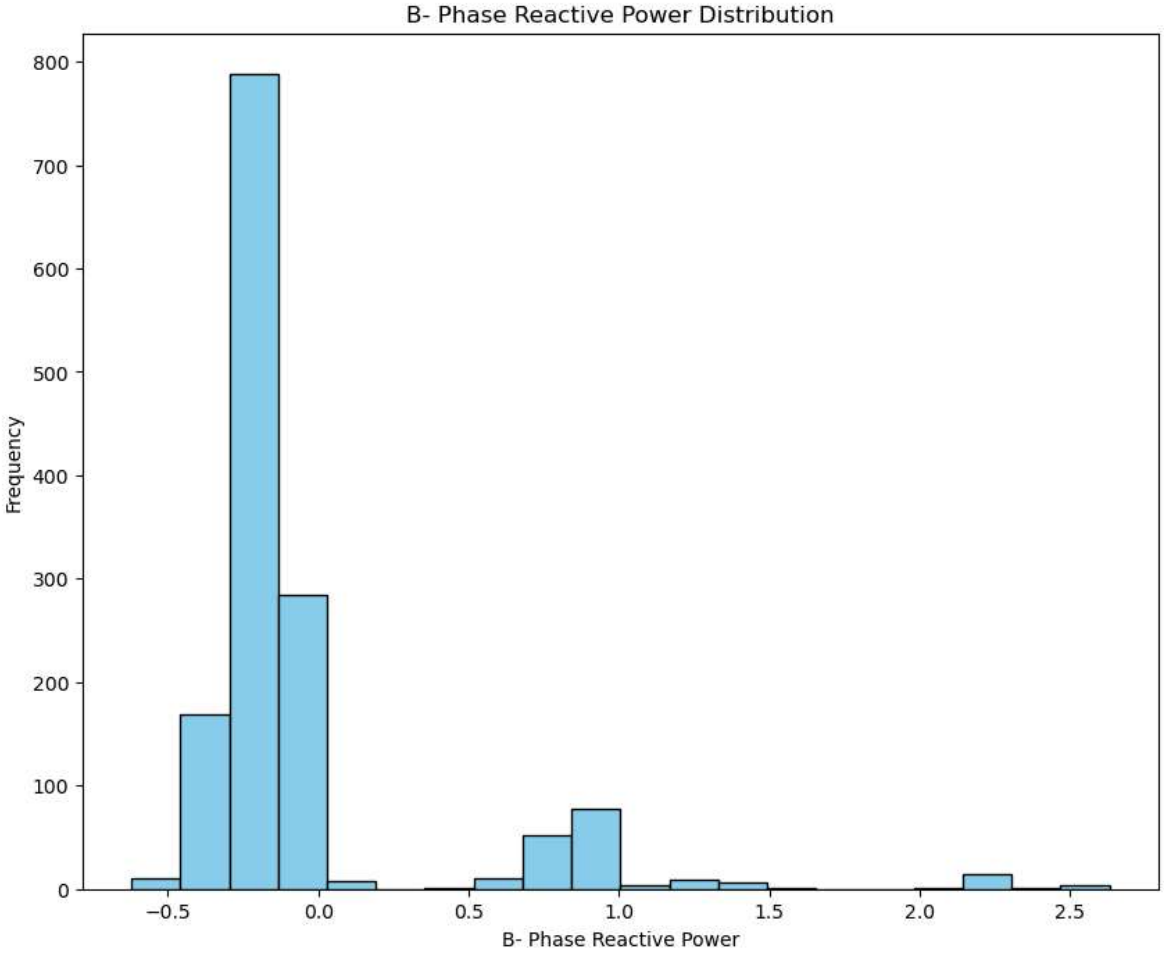

Y- Phase Active Power Distribution
Frequency
Y- Phase Active Power
0
50
100
150
200
250
300
350
4
6
8
10
12

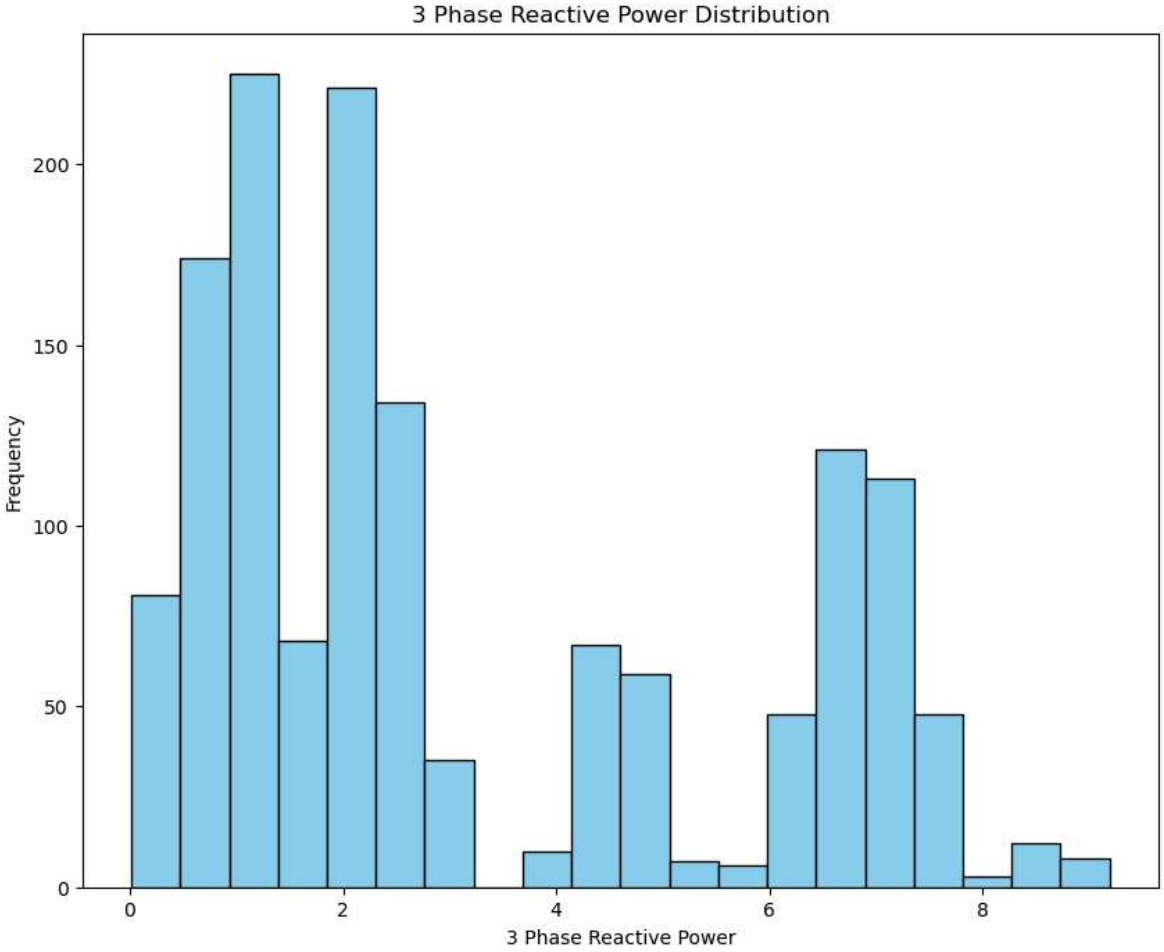

B- Phase Active Power Distribution
Frequency
B- Phase Active Power
0
50
100
150
200
250
300
350
4
5
6
7
8
9
10
11

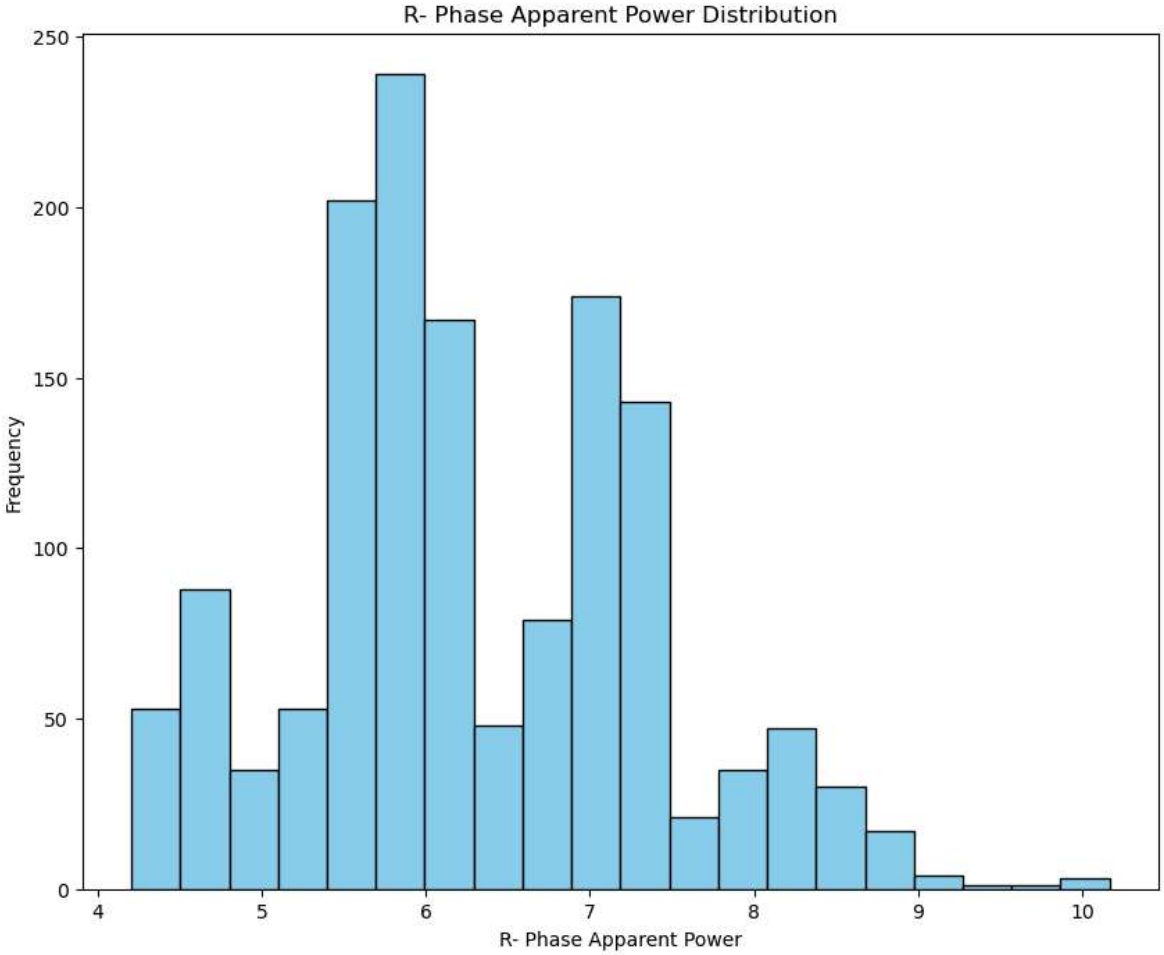

3 Phase Active Power Distribution
Frequency
3 Phase Active Power
0
50
100
150
200
250
300
15.0
17.5
20.0
22.5
25.0
27.5
30.0
32.5

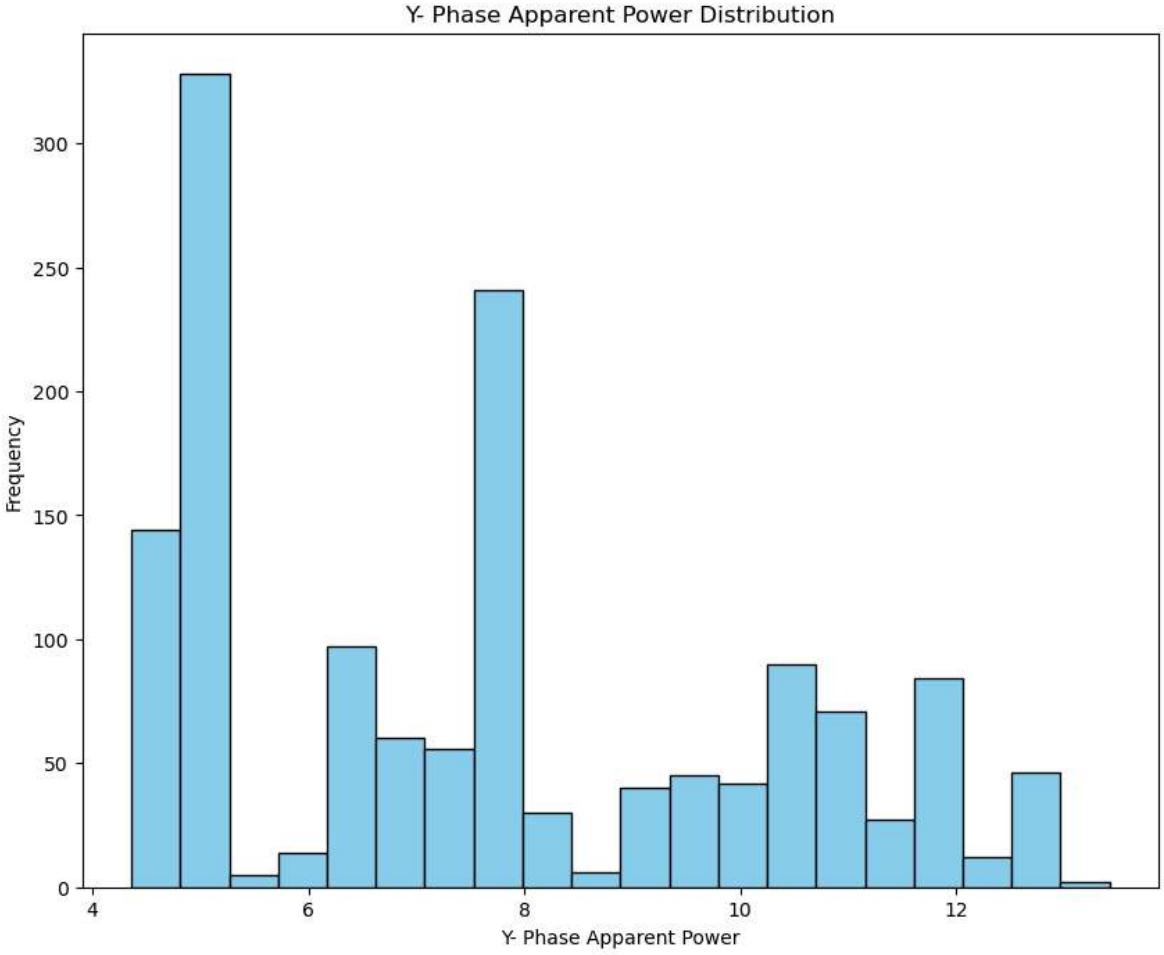

R- Phase Reactive Power Distribution
Frequency
R- Phase Reactive Power
0
50
100
150
200
250
0
1
2
3
4

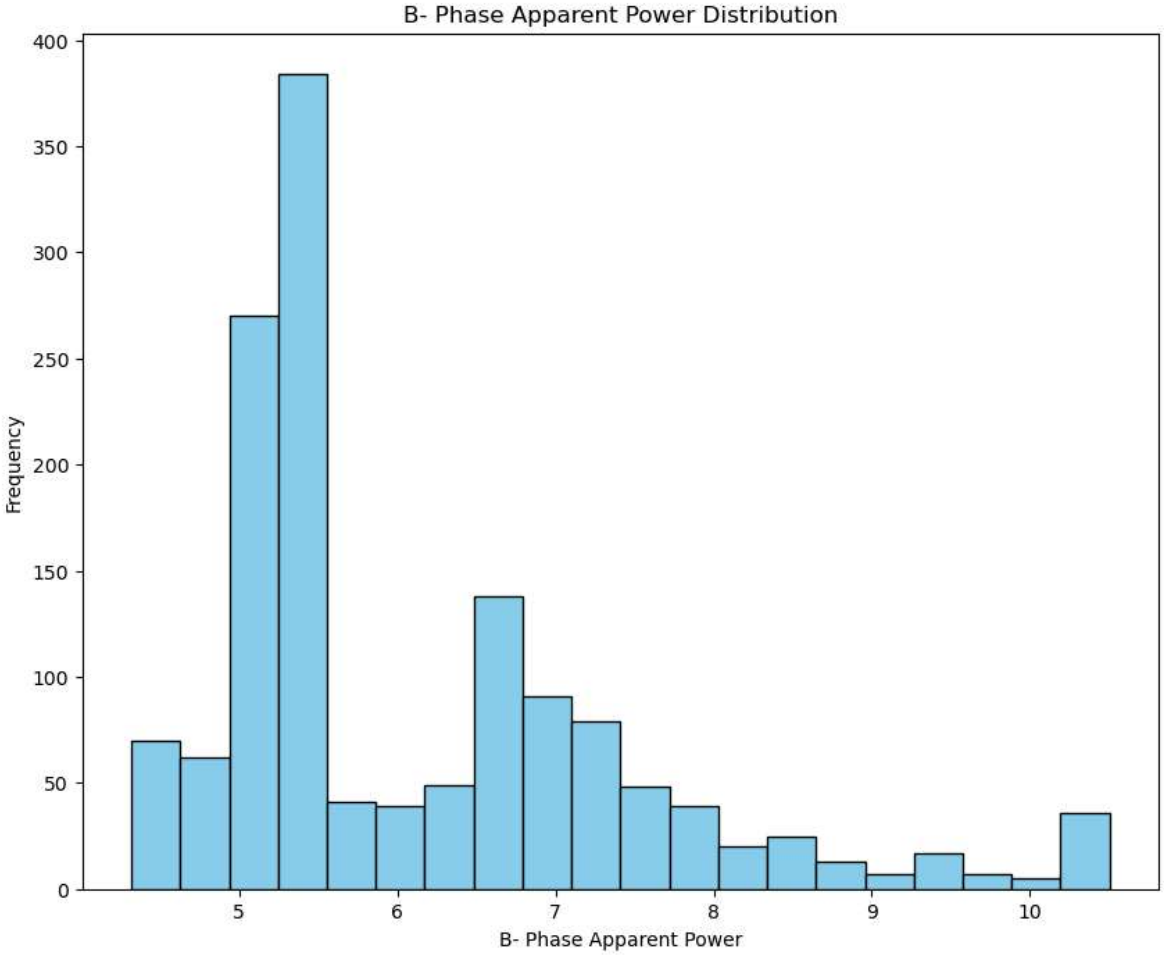

Y- Phase Reactive Power Distribution
Frequency
Y- Phase Reactive Power
0
100
200
300
400
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

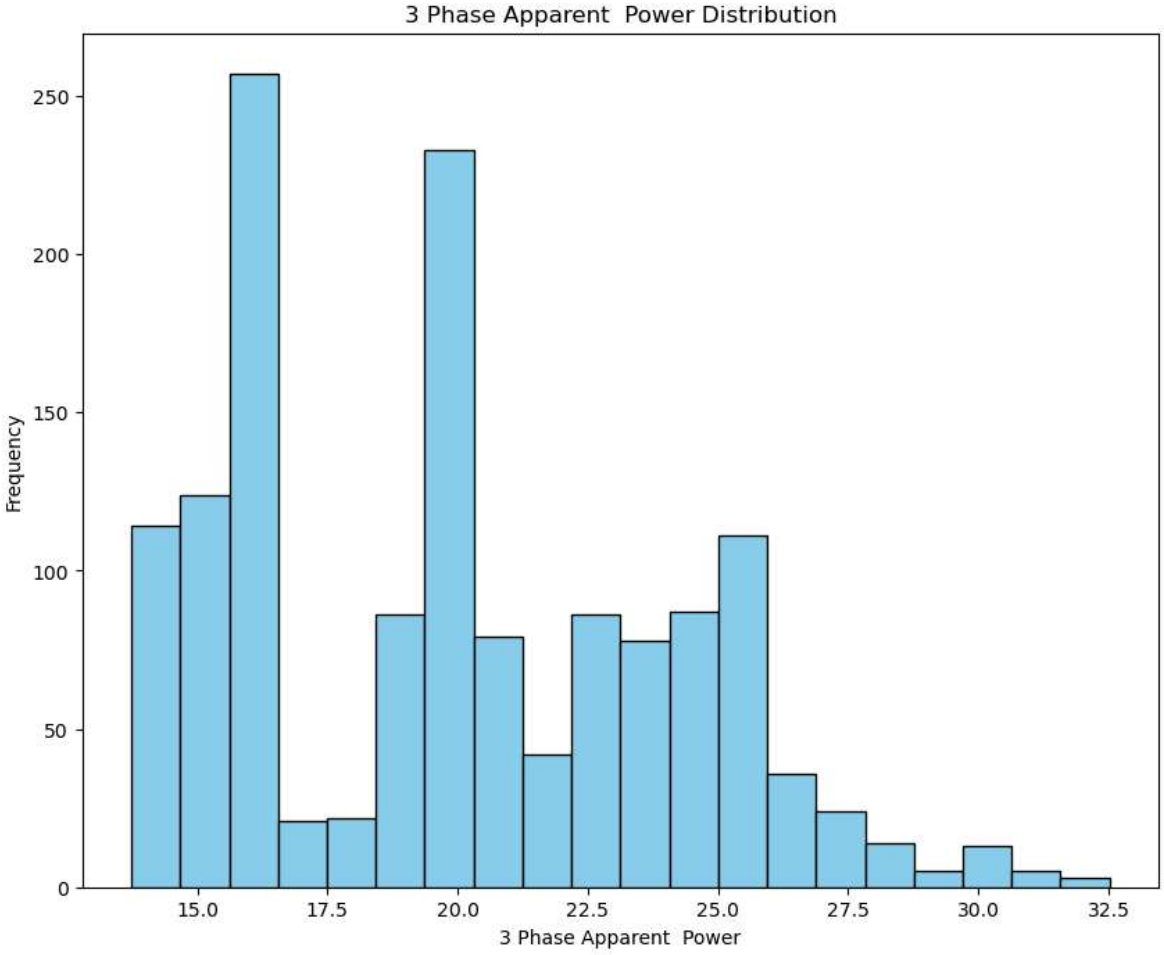

B- Phase Reactive Power Distribution
Frequency
B- Phase Reactive Power
800
700
600
500
400
300
200
100
0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5


3 Phase Reactive Power Distribution
Frequency
3 Phase Reactive Power
200
150
100
50
0
0
2
4
6
8


R- Phase Apparent Power Distribution
Frequency
R- Phase Apparent Power
0
50
100
150
200
250
4
5
6
7
8
9
10


Y- Phase Apparent Power Distribution
Frequency
Y- Phase Apparent Power
0
50
100
150
200
250
300
4
6
8
10
12

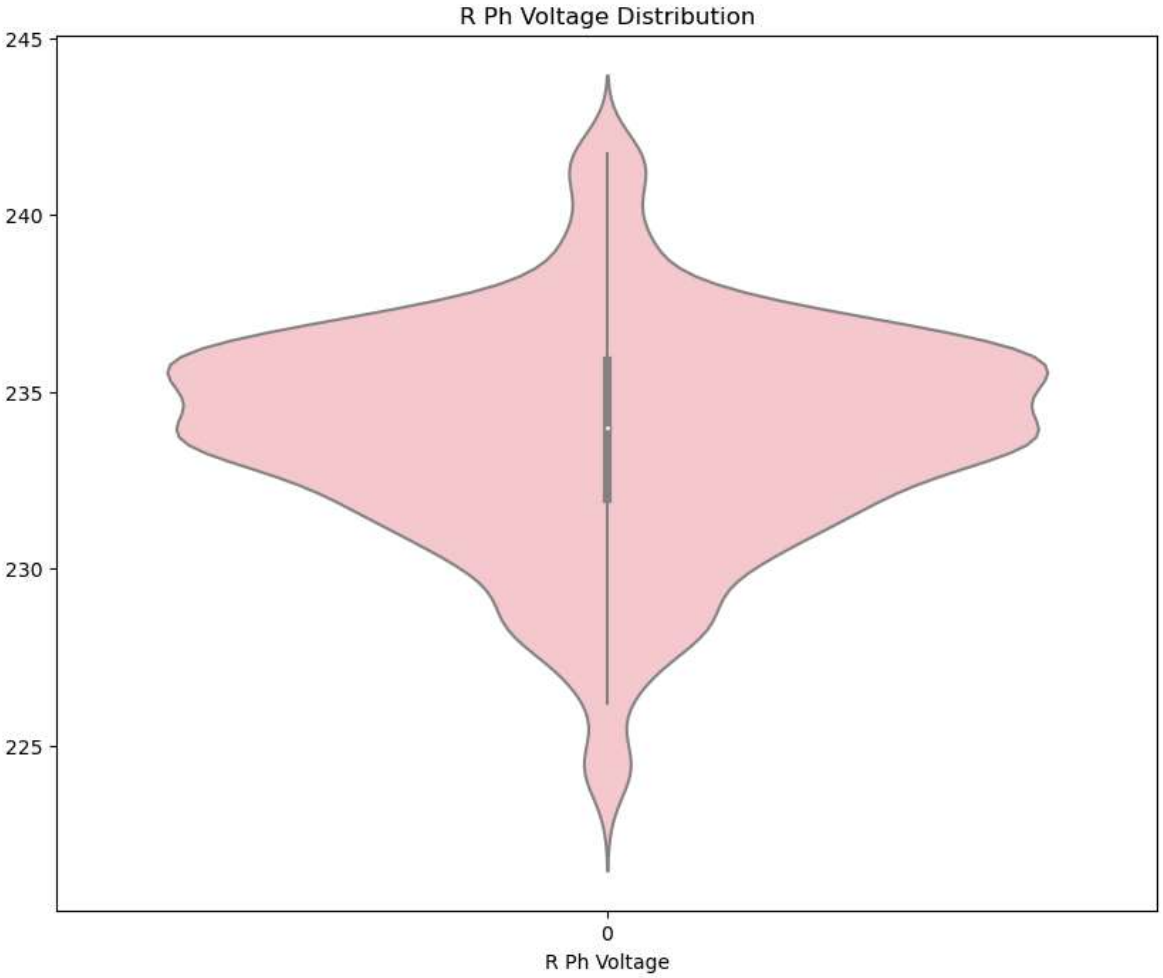




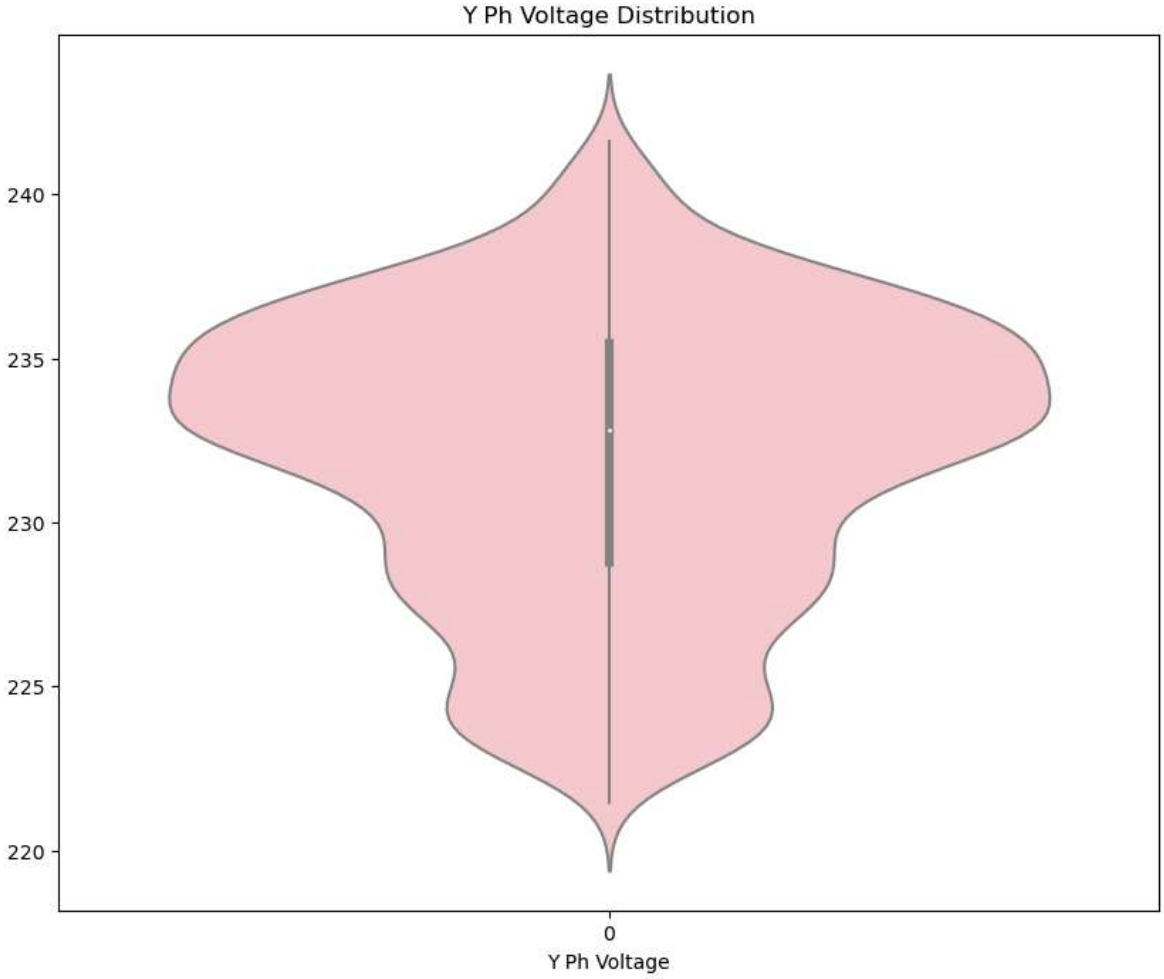
In [28]:
```
for col in data_minute_avg.columns:
    plt.figure(figsize=(10, 8))
```

```
    sns.violinplot(data=data_minute_avg[col], color='pink')
    tit = col + " Distribution"
    plt.title(tit)
    plt.xlabel(col)
    plt.show()
```

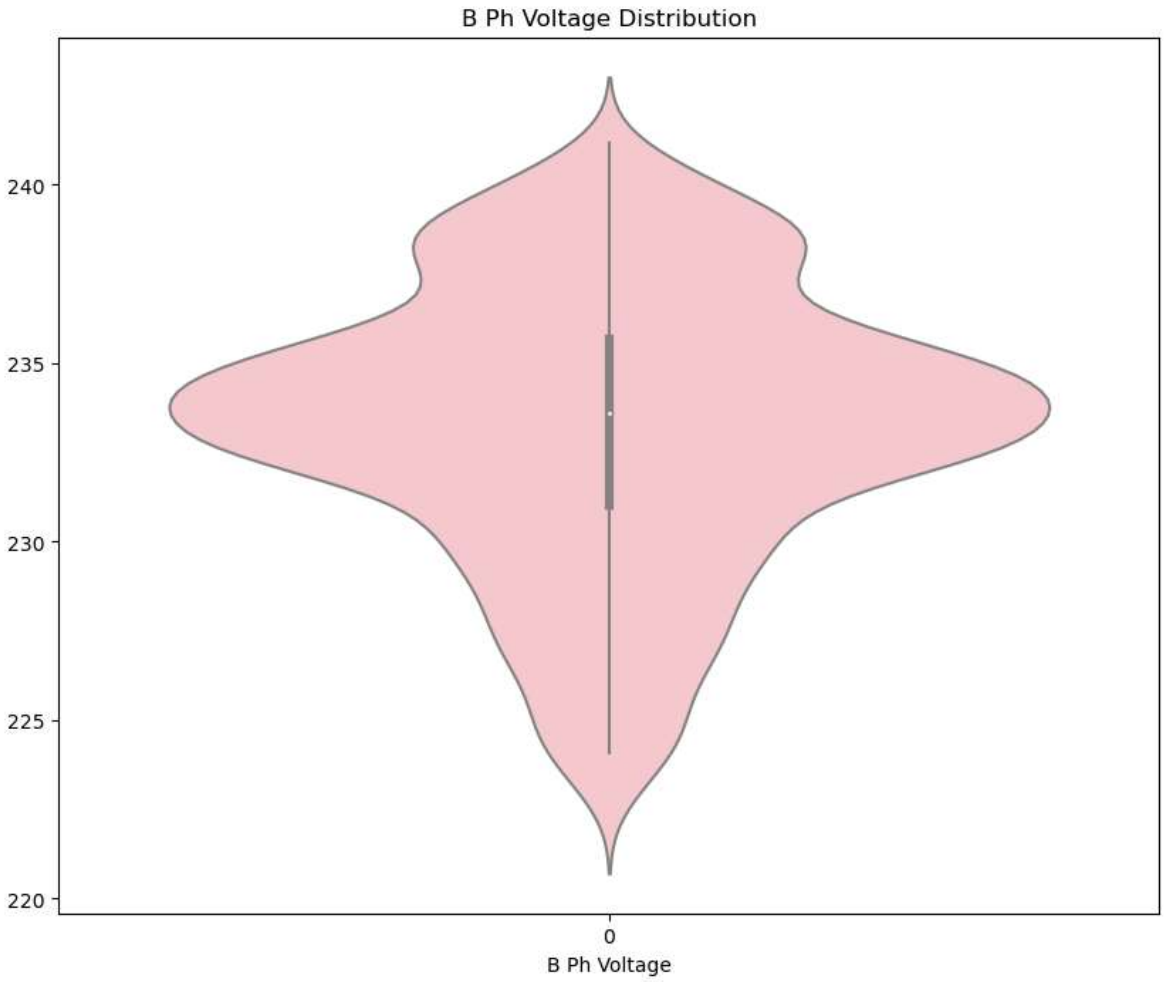
```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



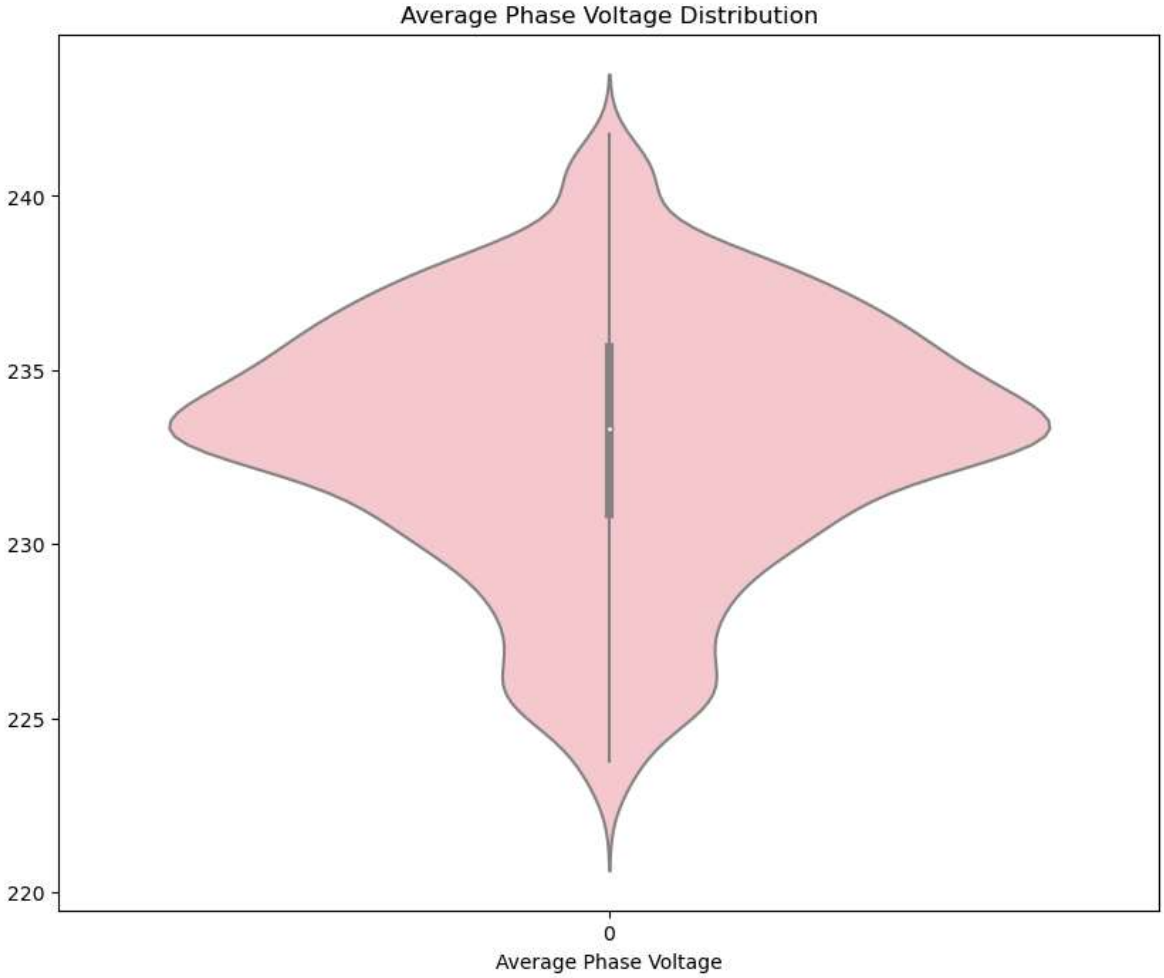

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



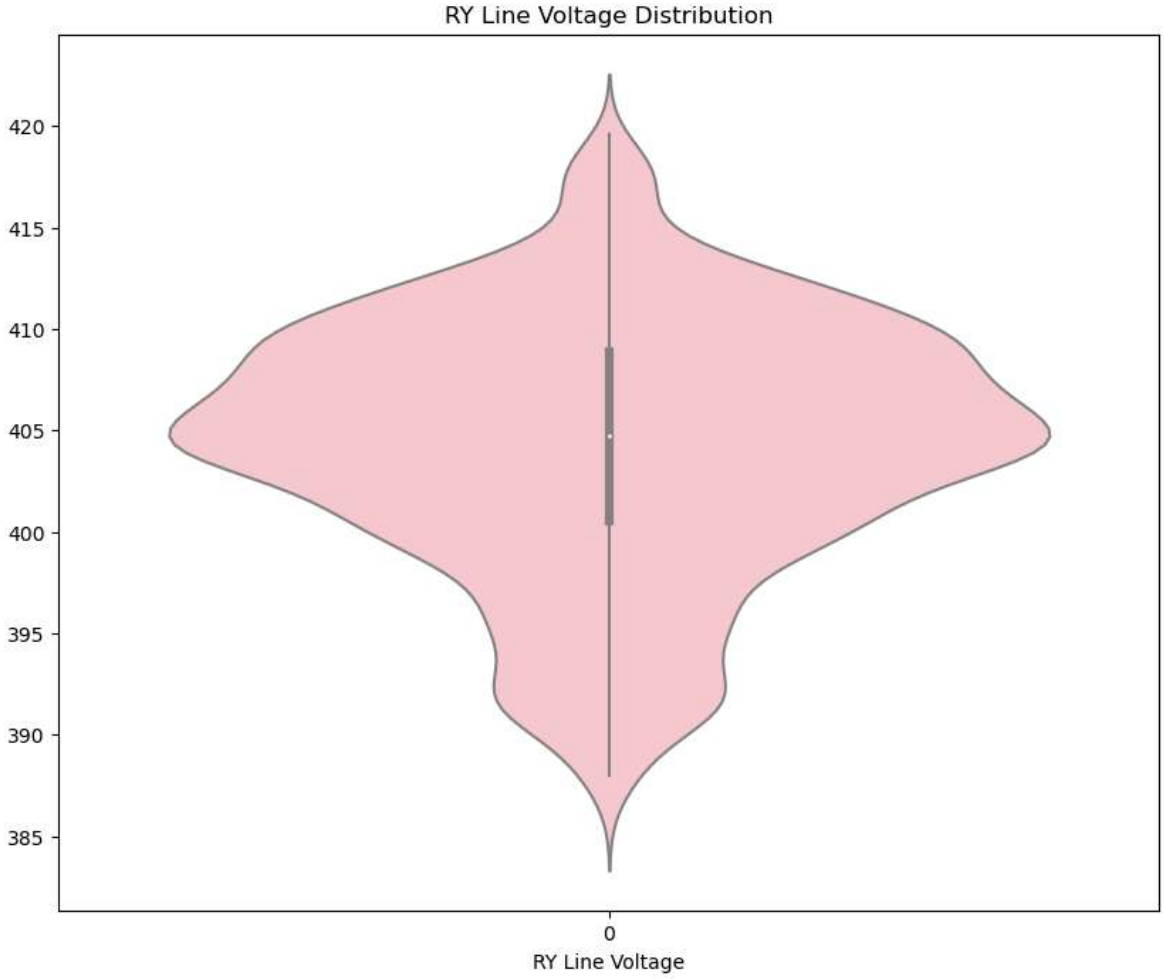

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



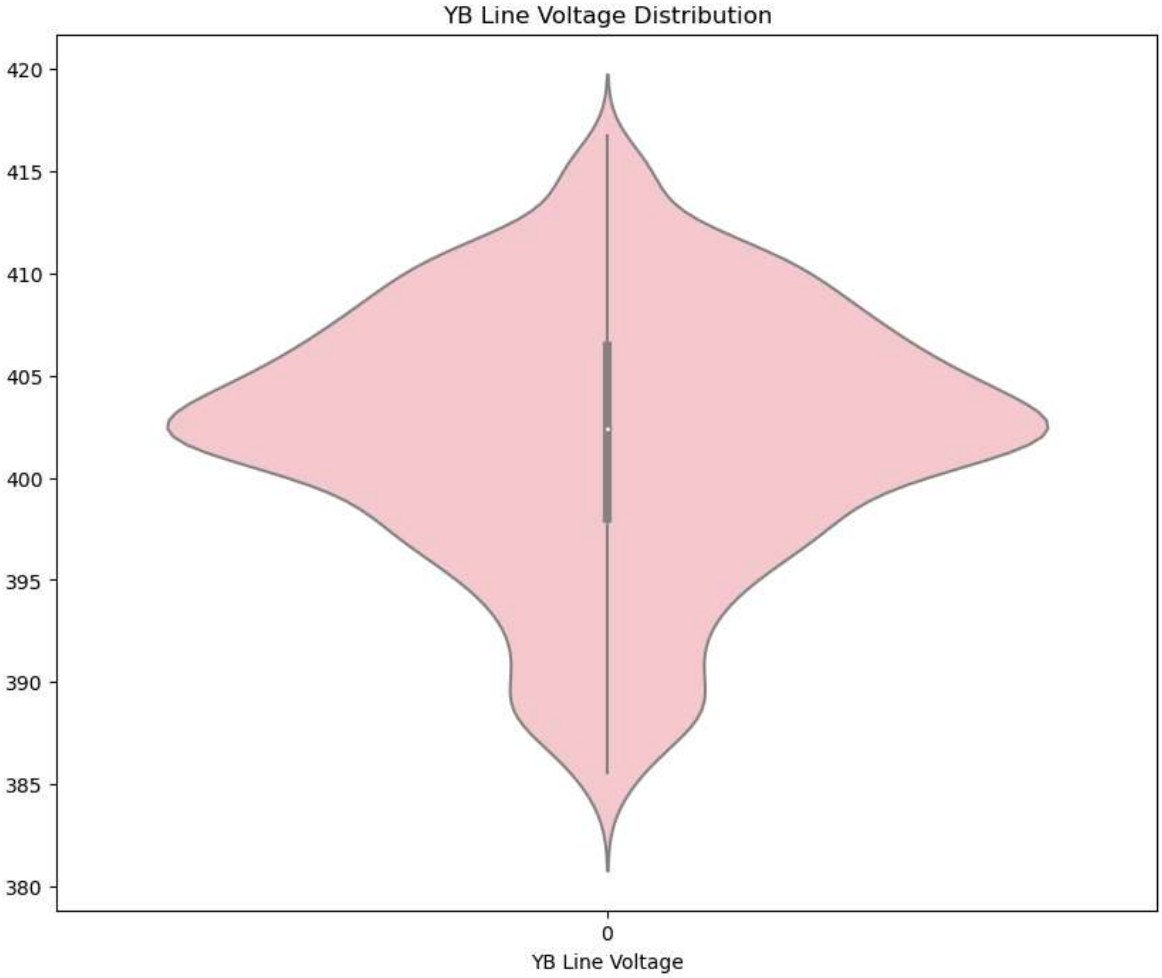

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



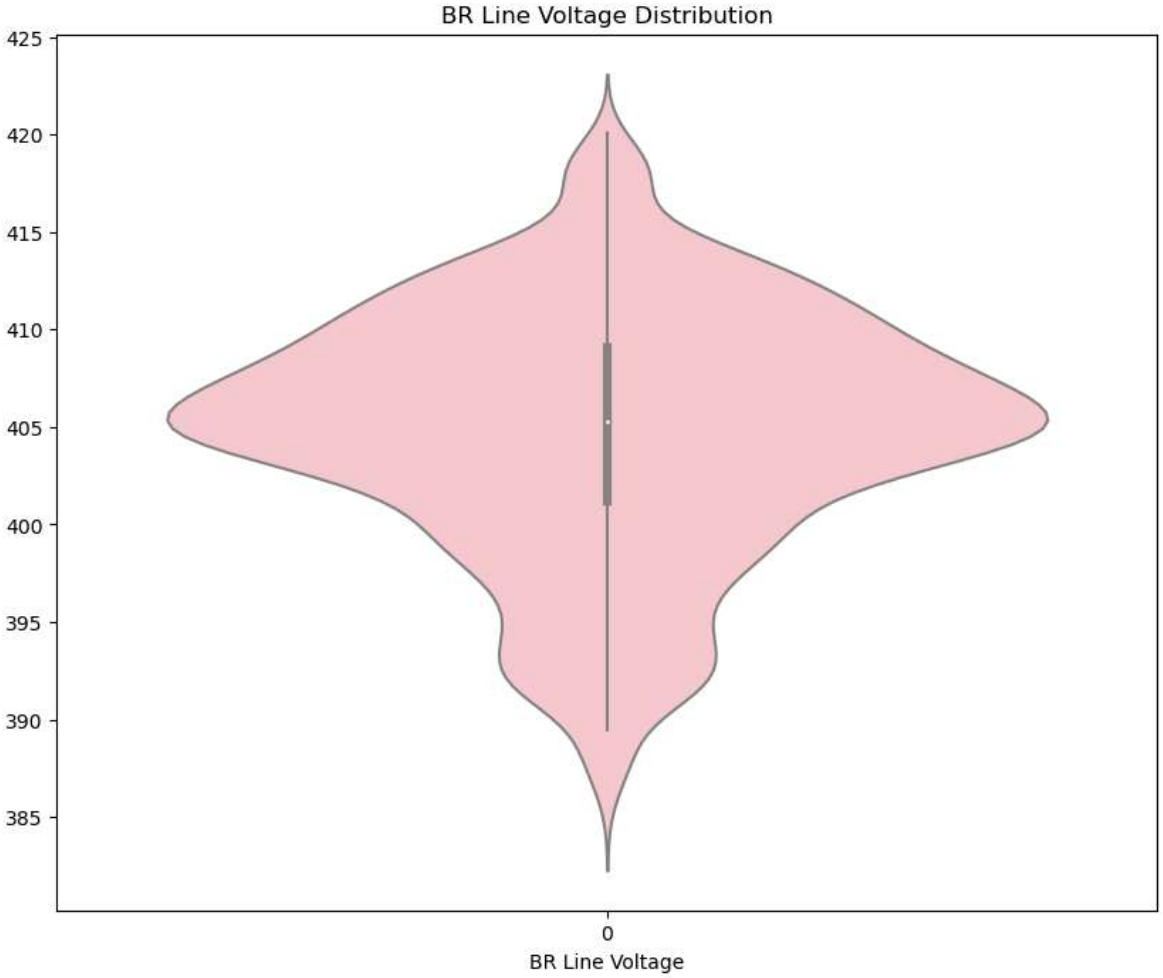

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



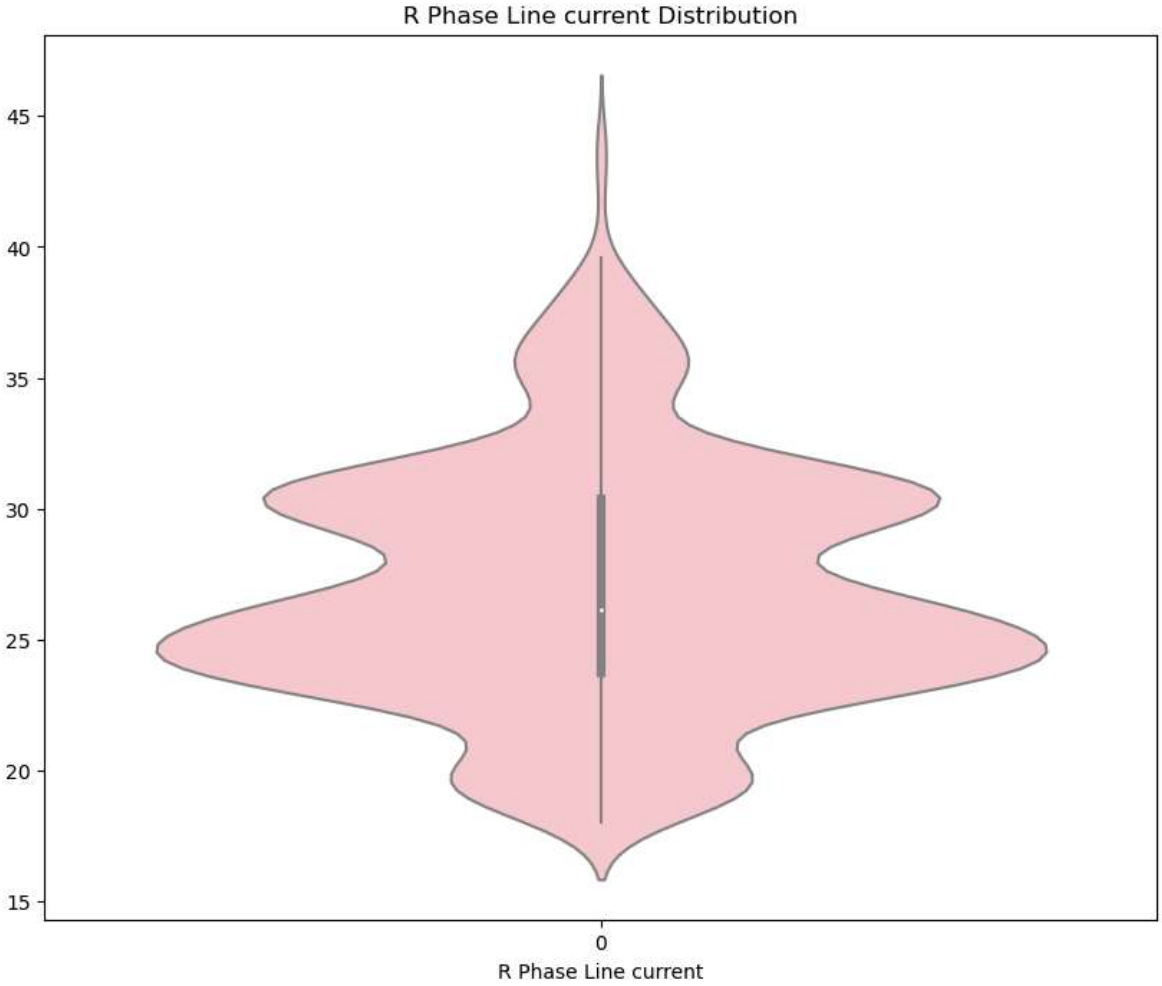

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



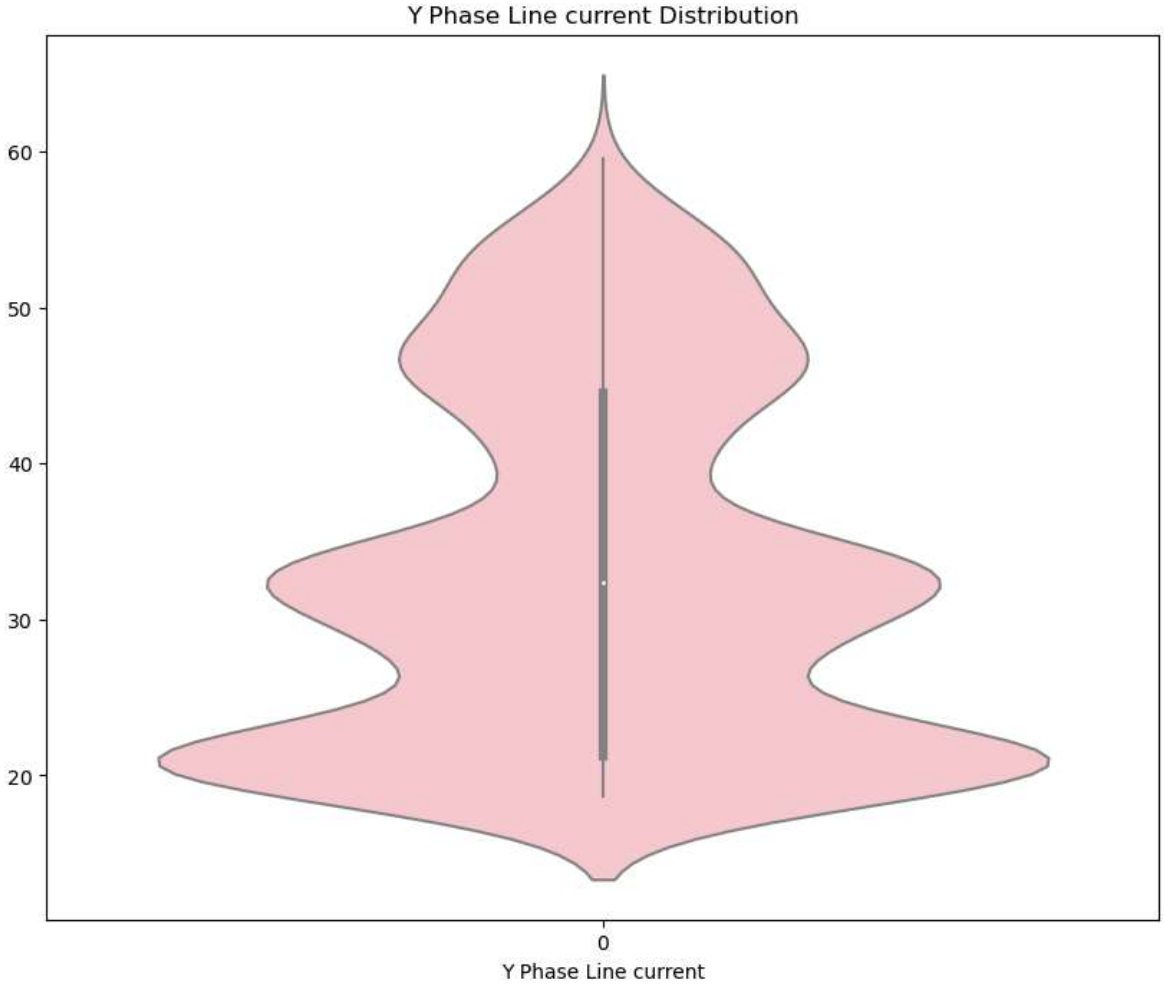

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



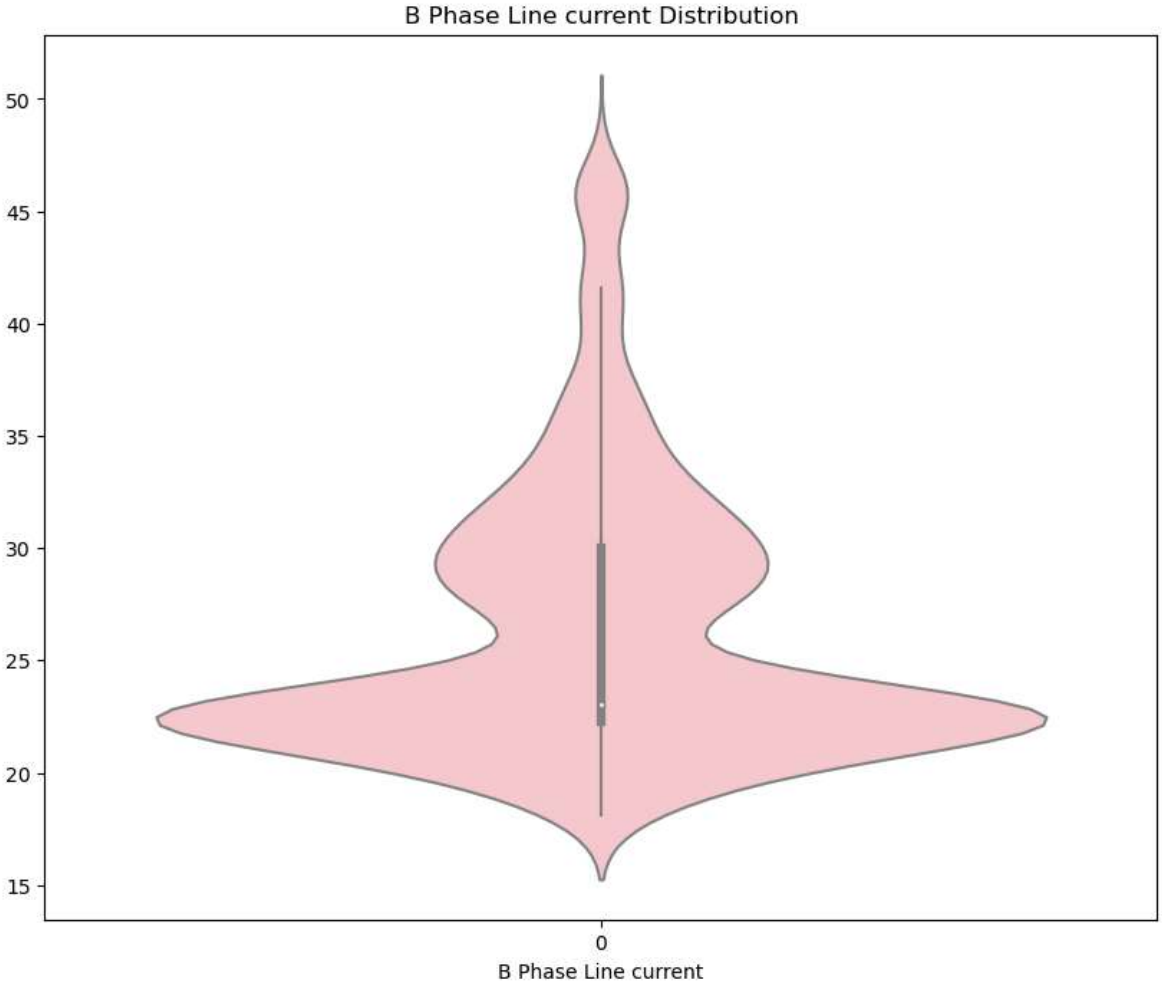

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:486: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



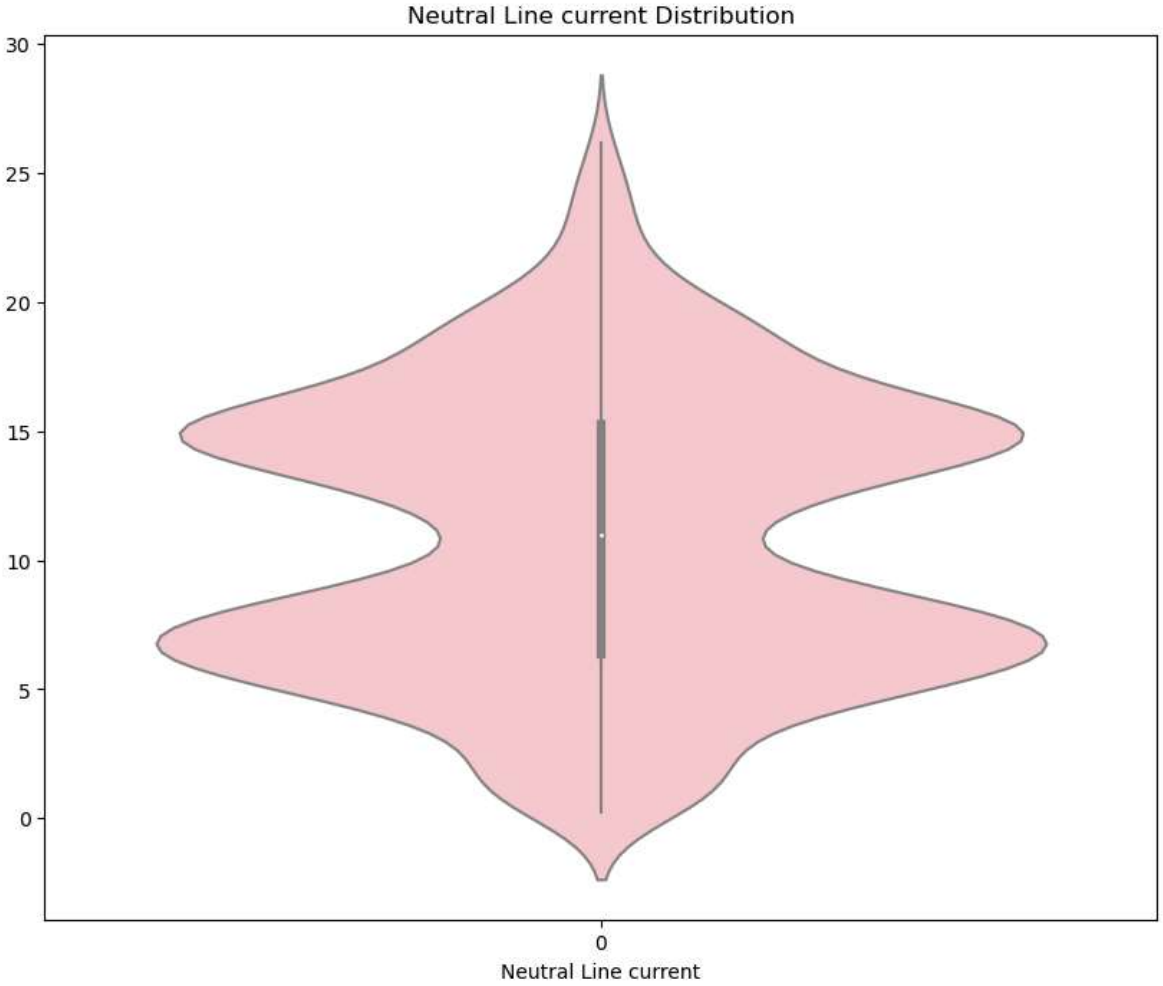

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



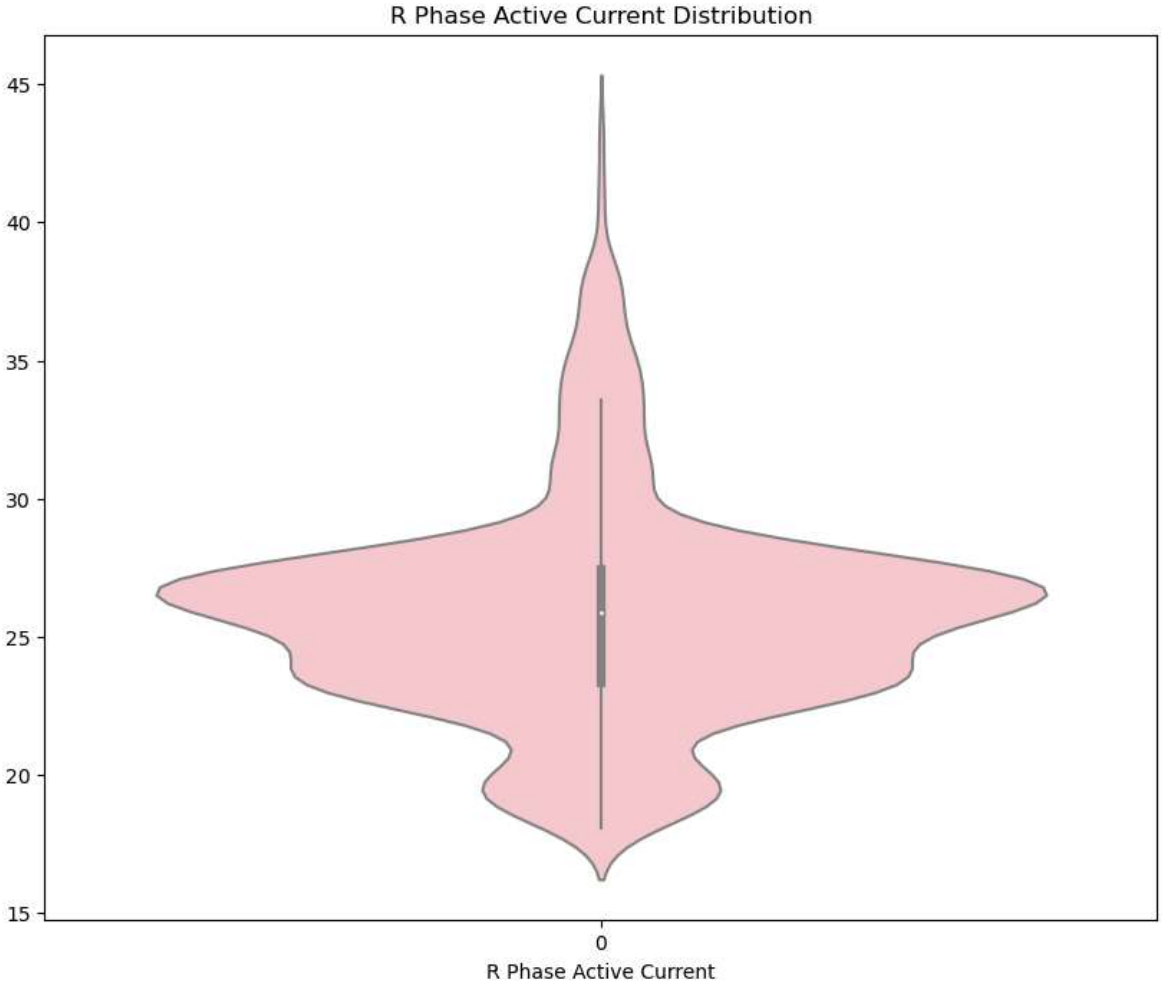

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



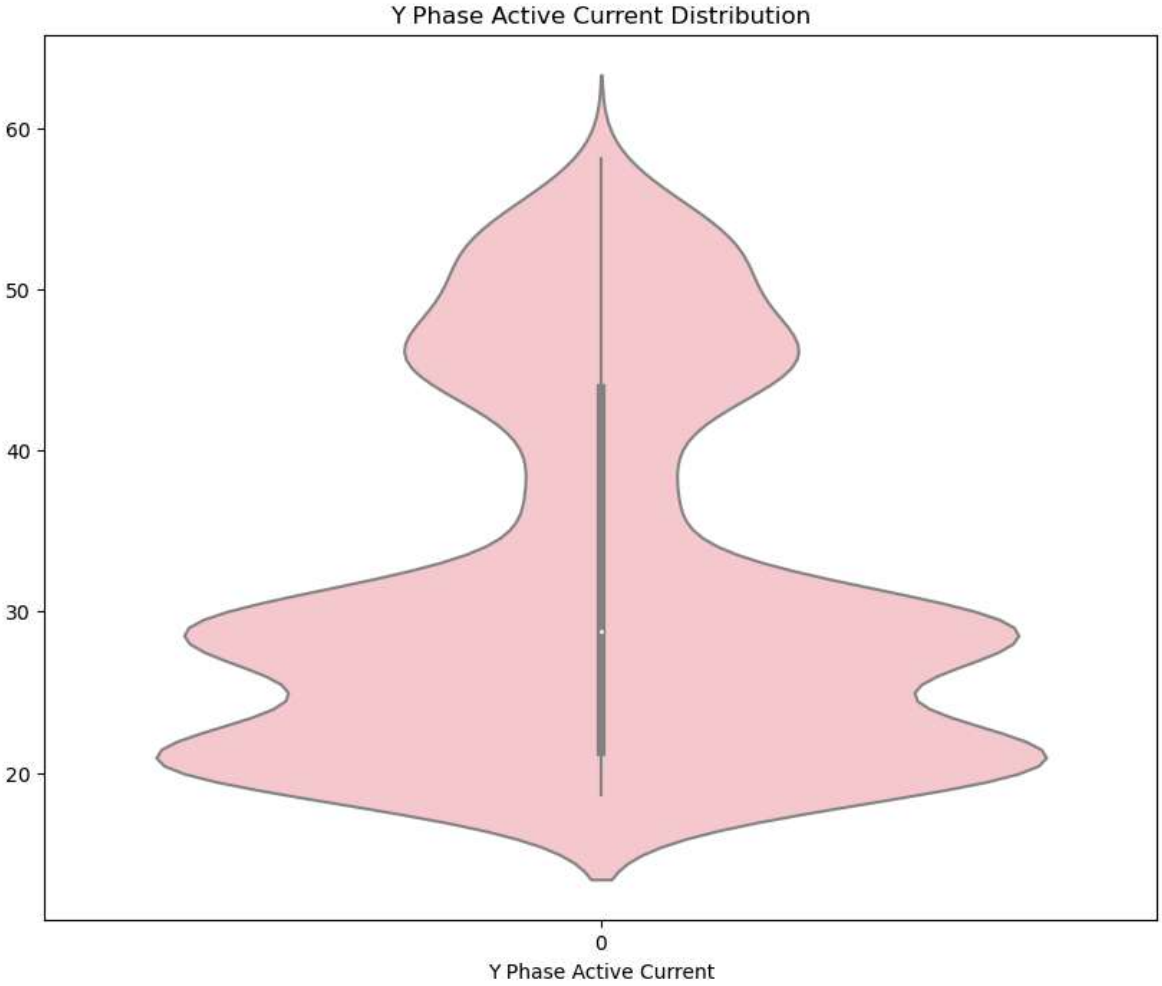

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



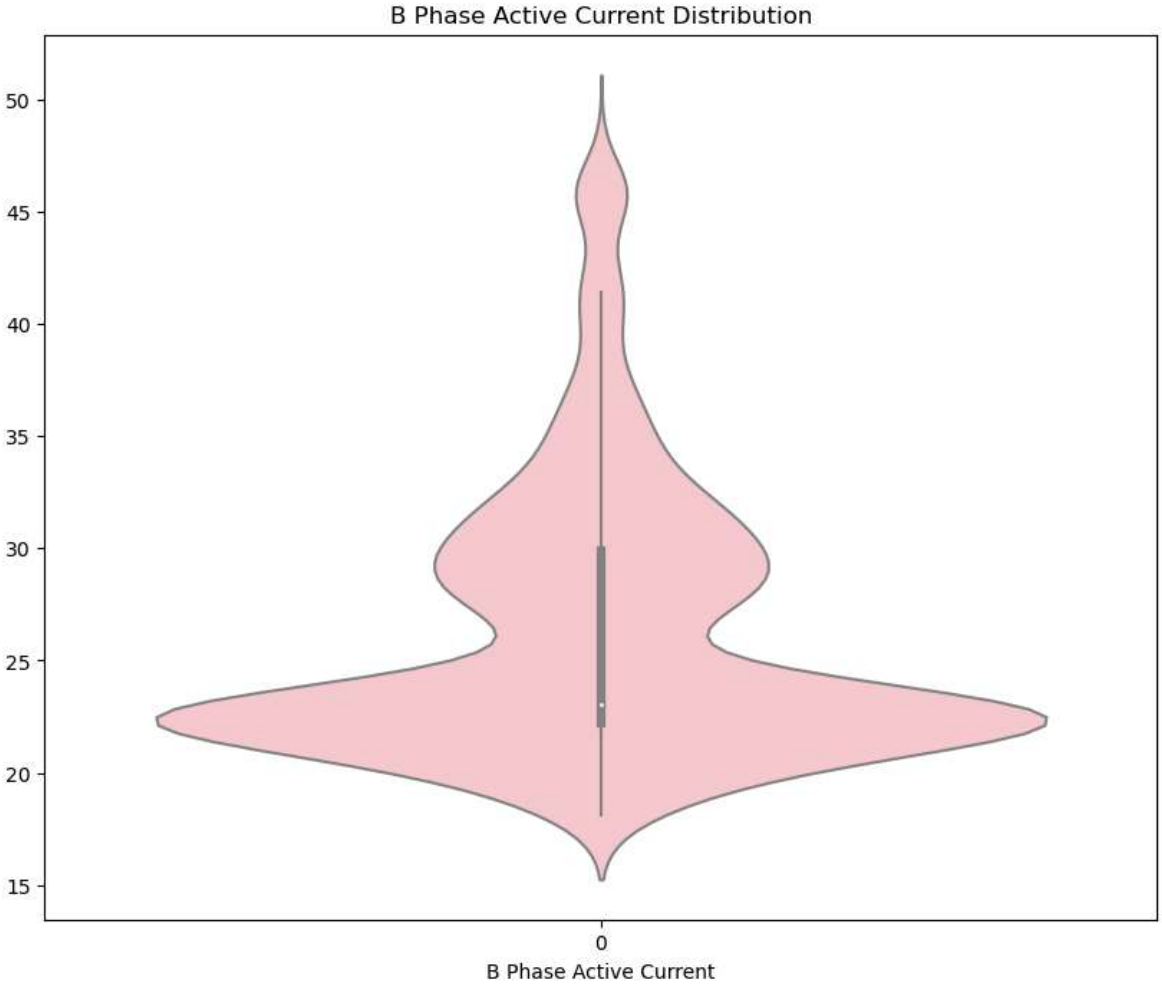

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



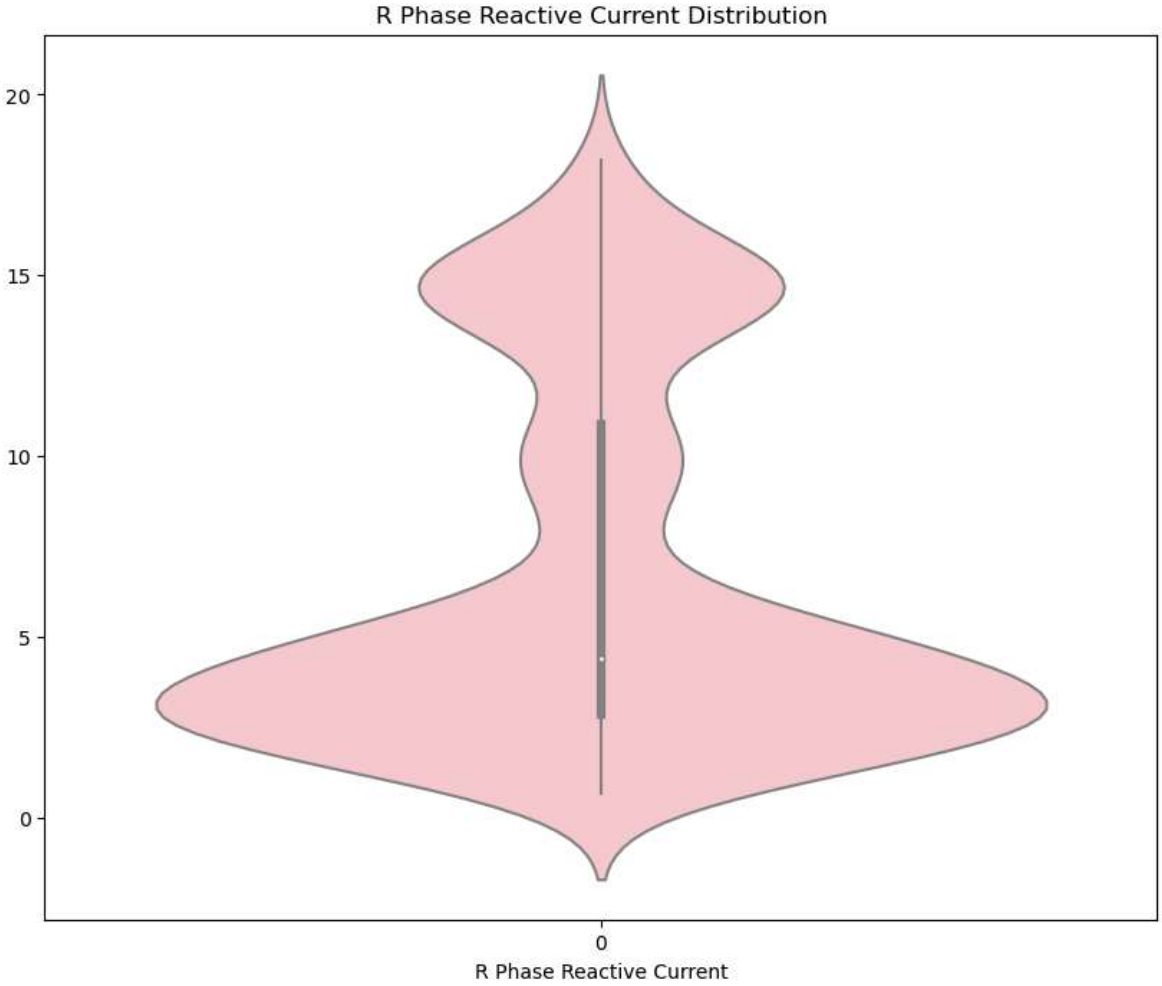

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



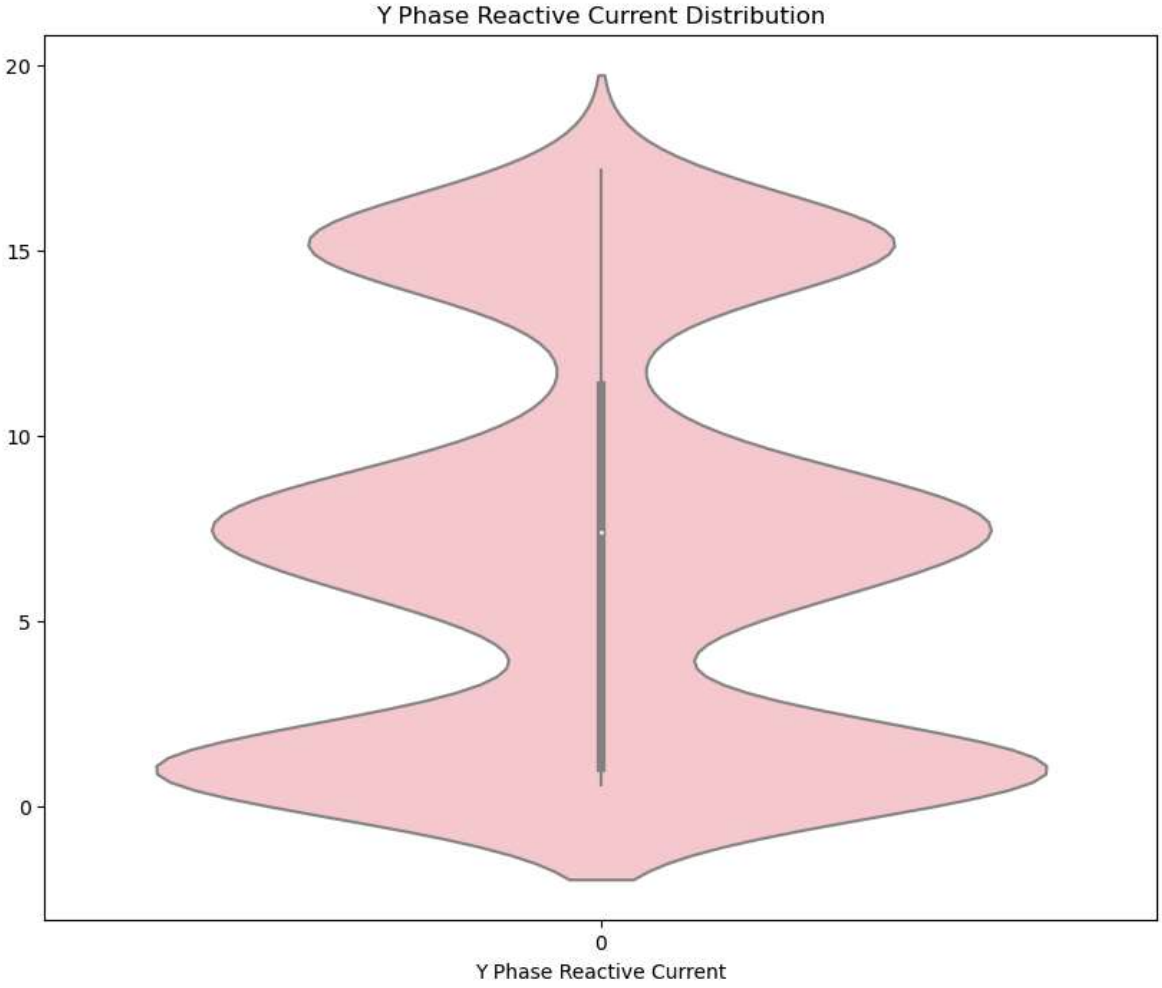

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



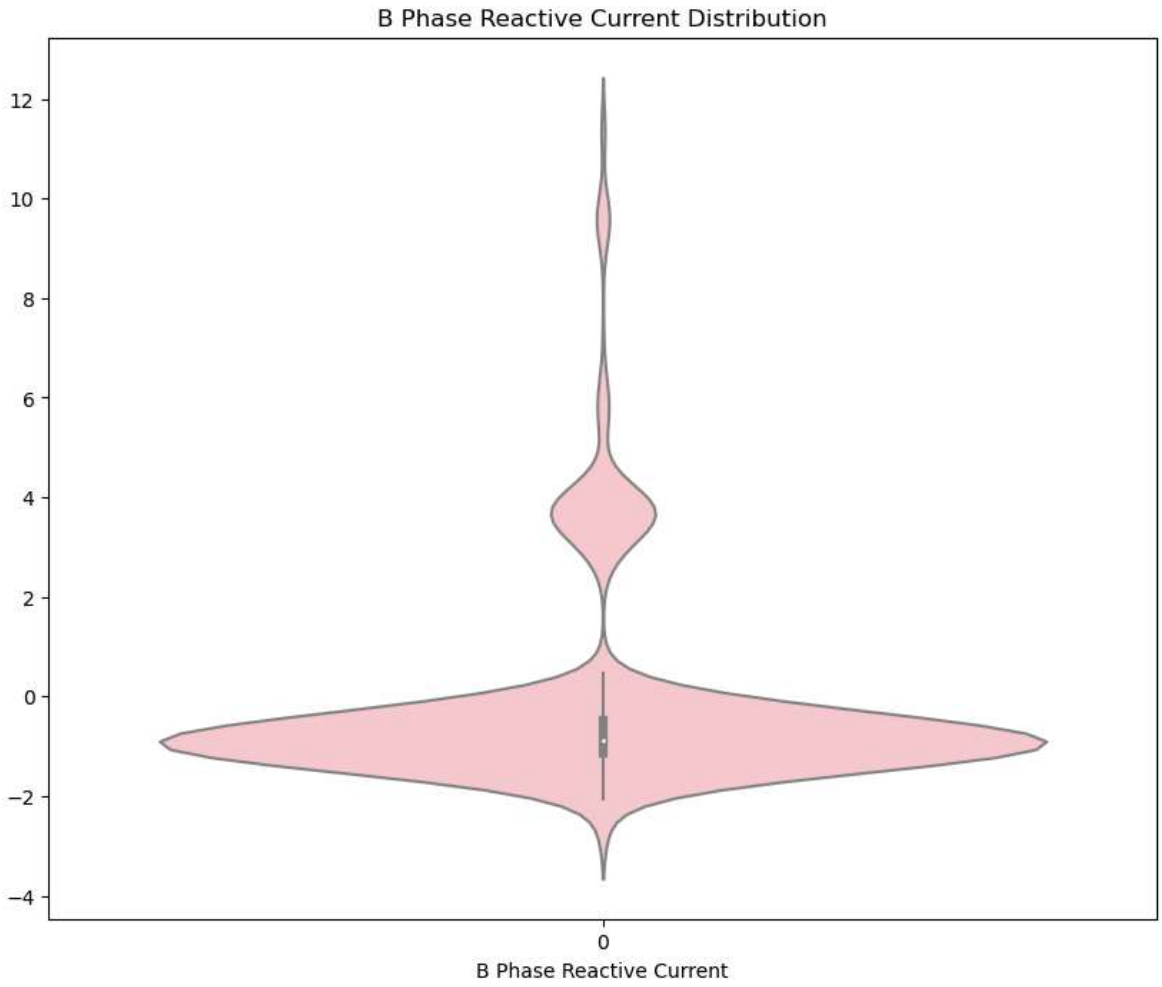

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



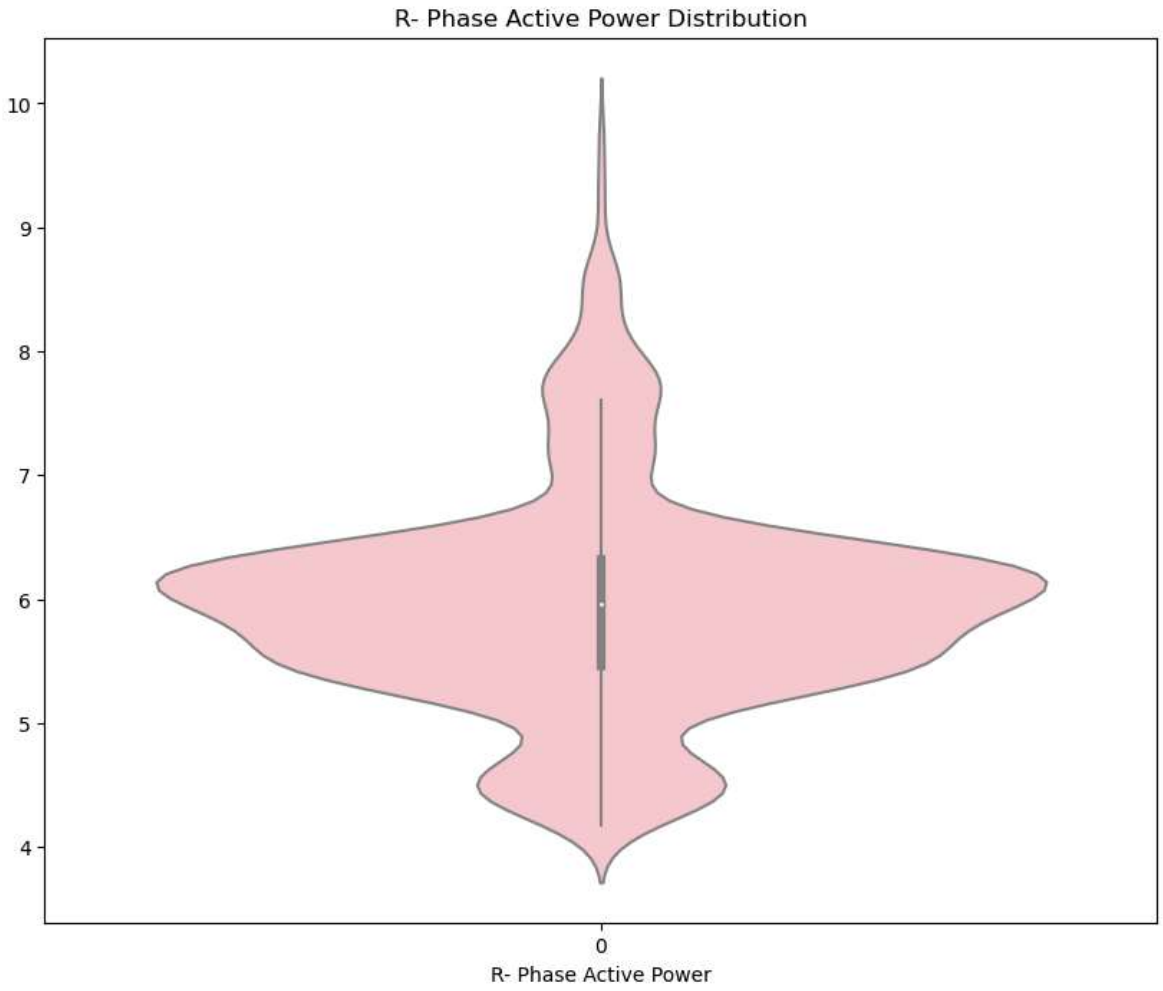

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



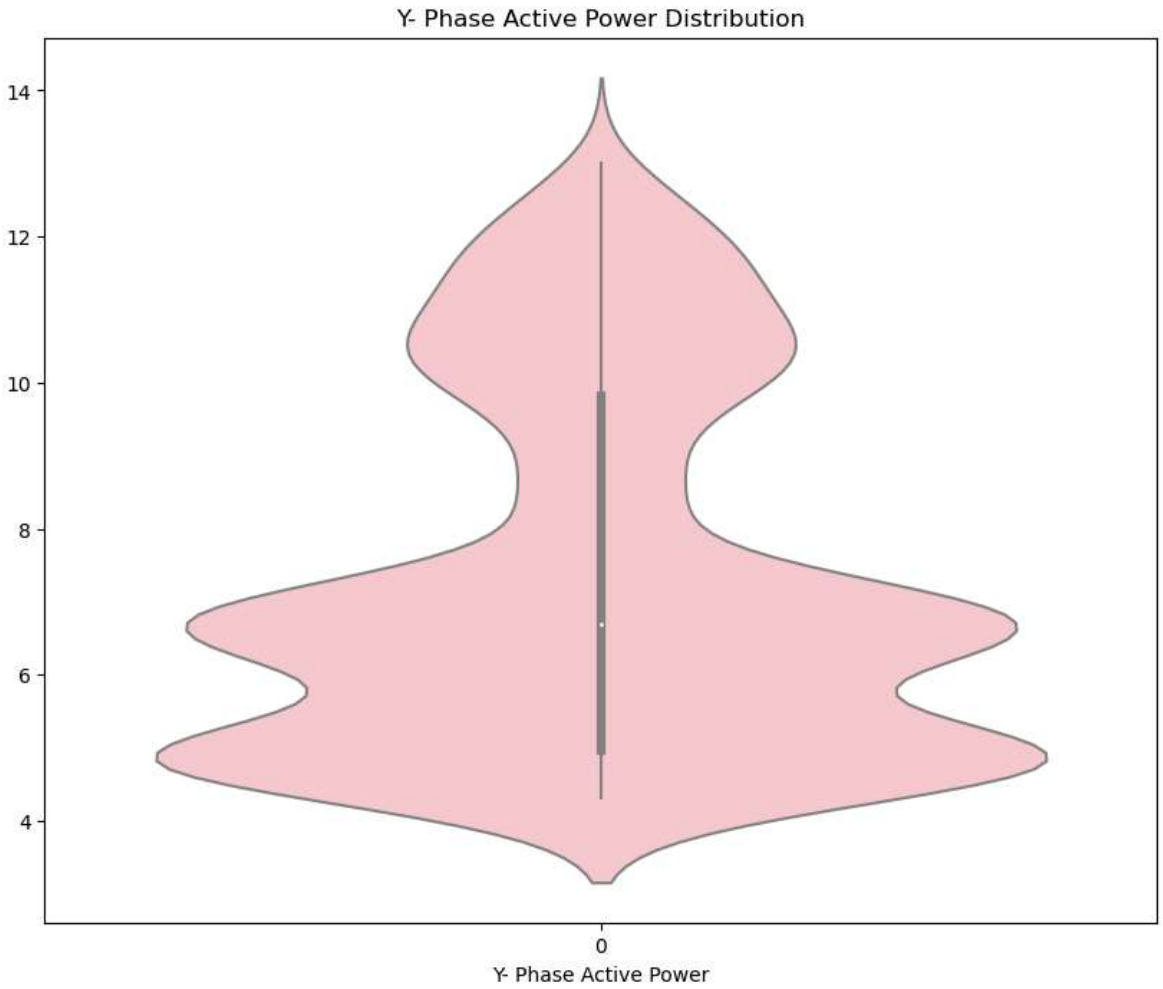

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



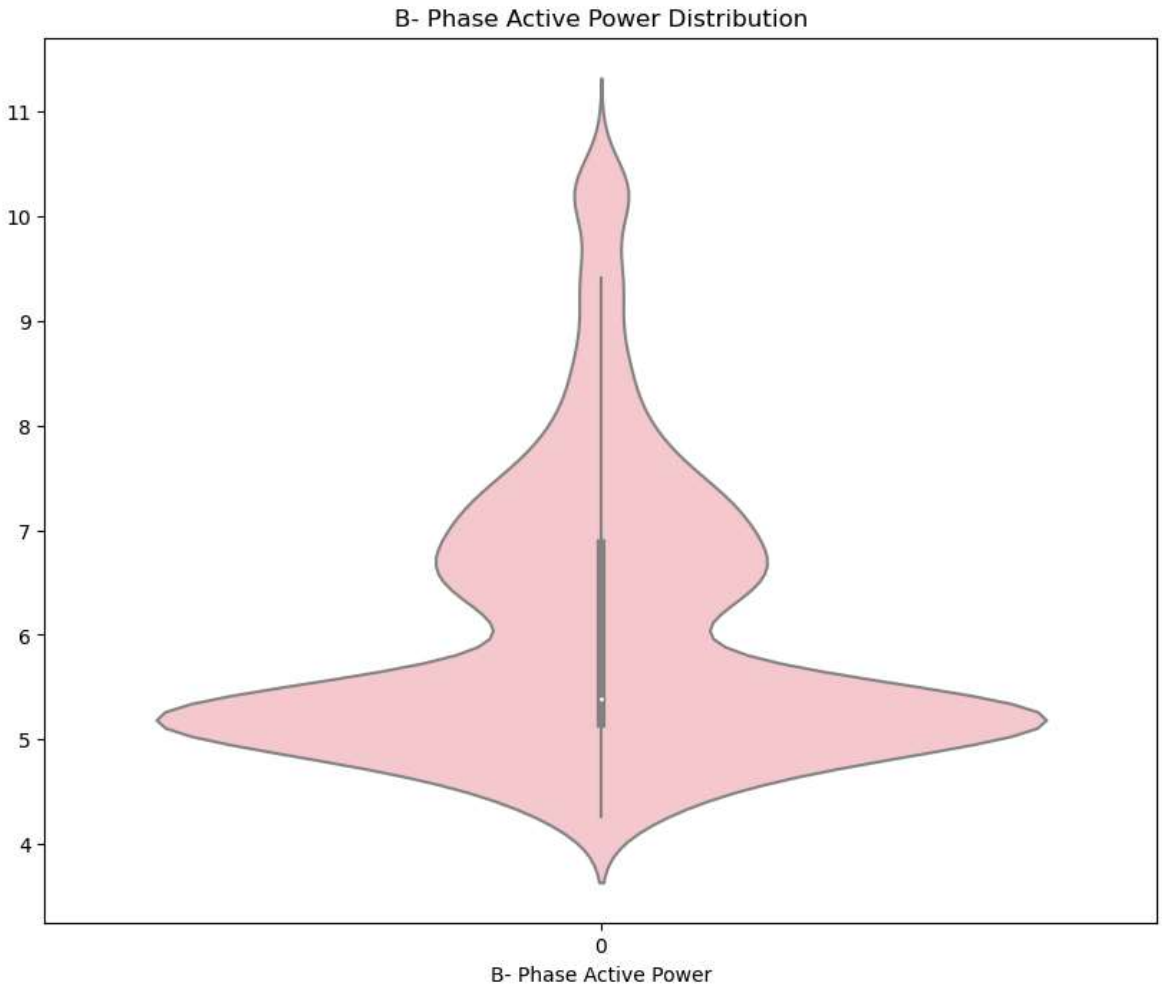

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



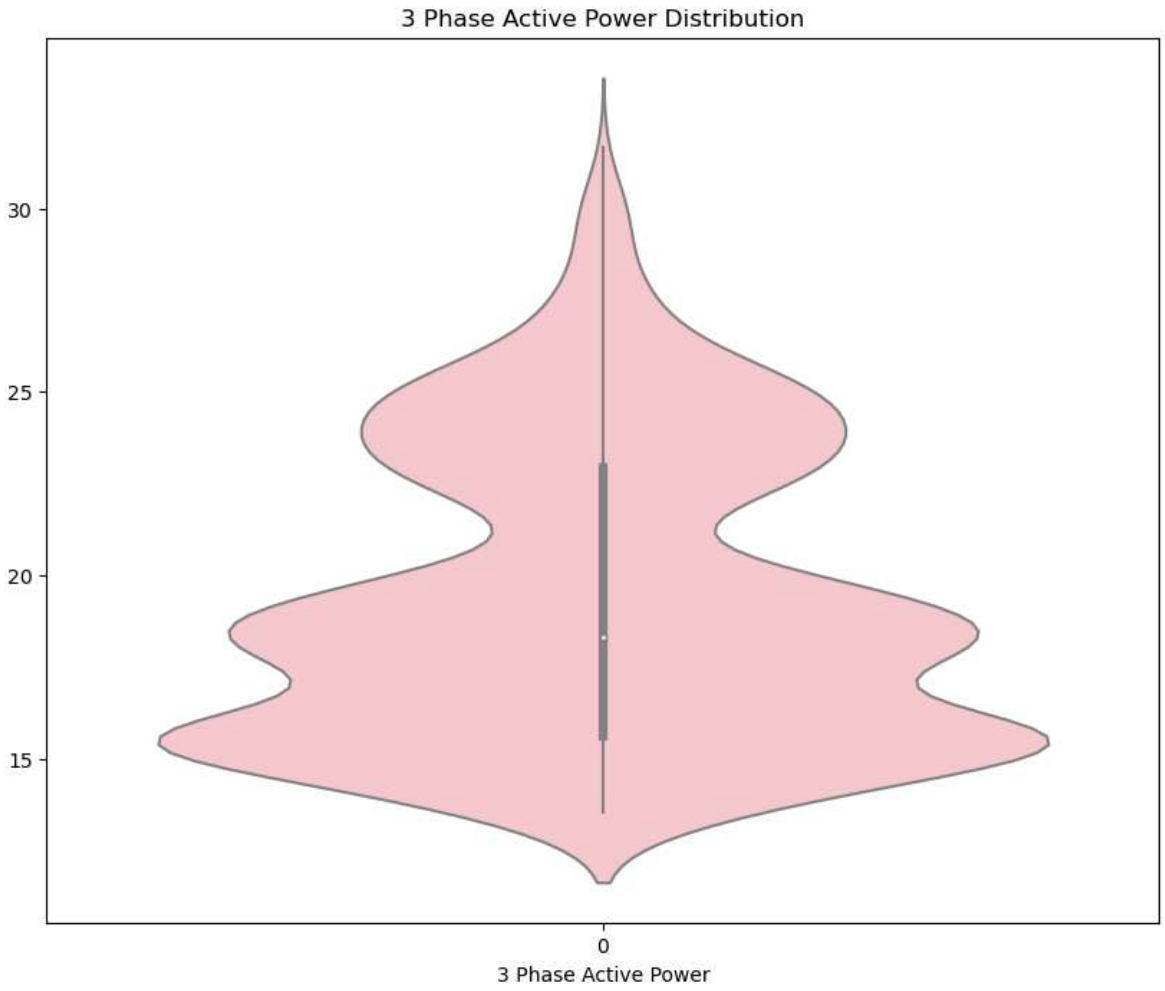

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



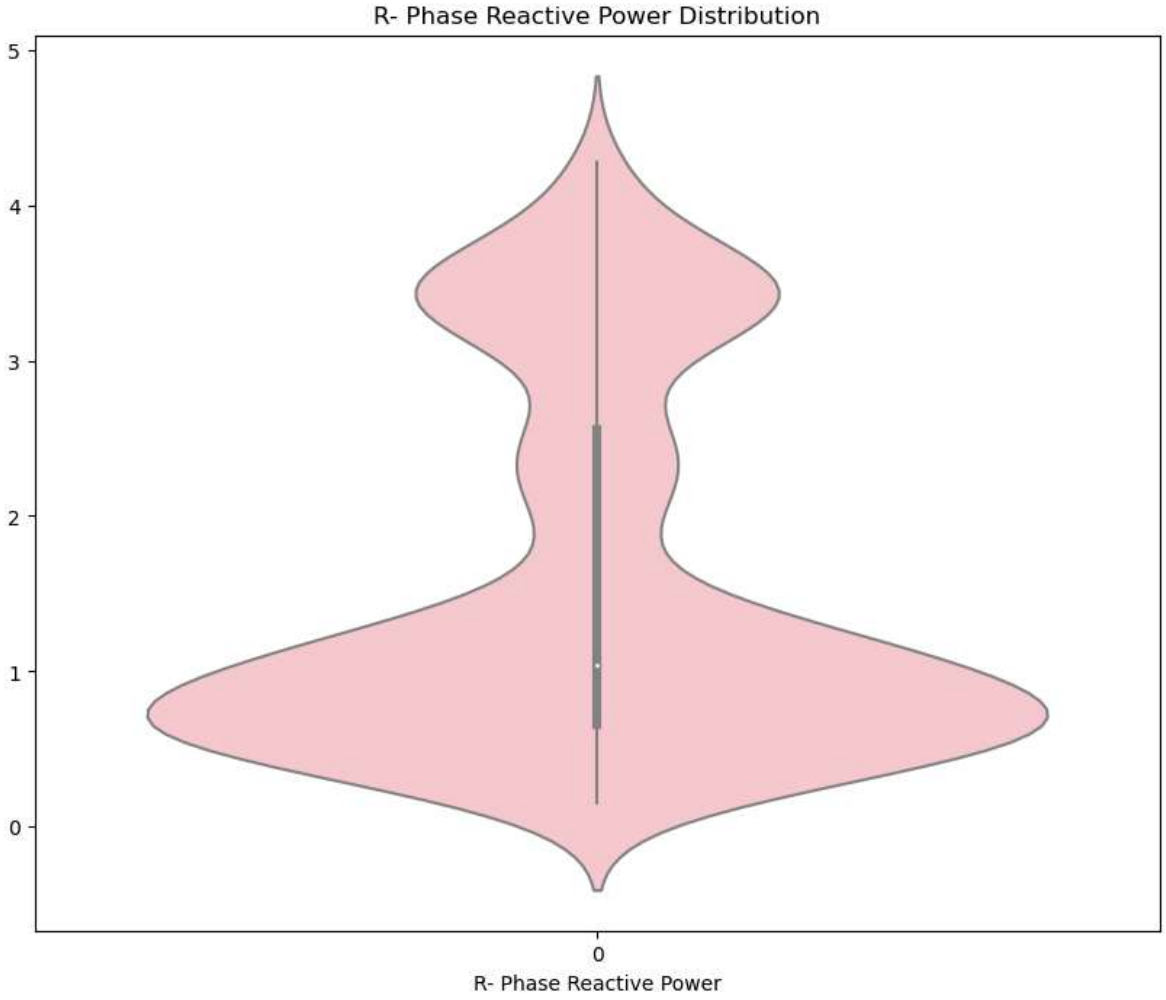

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



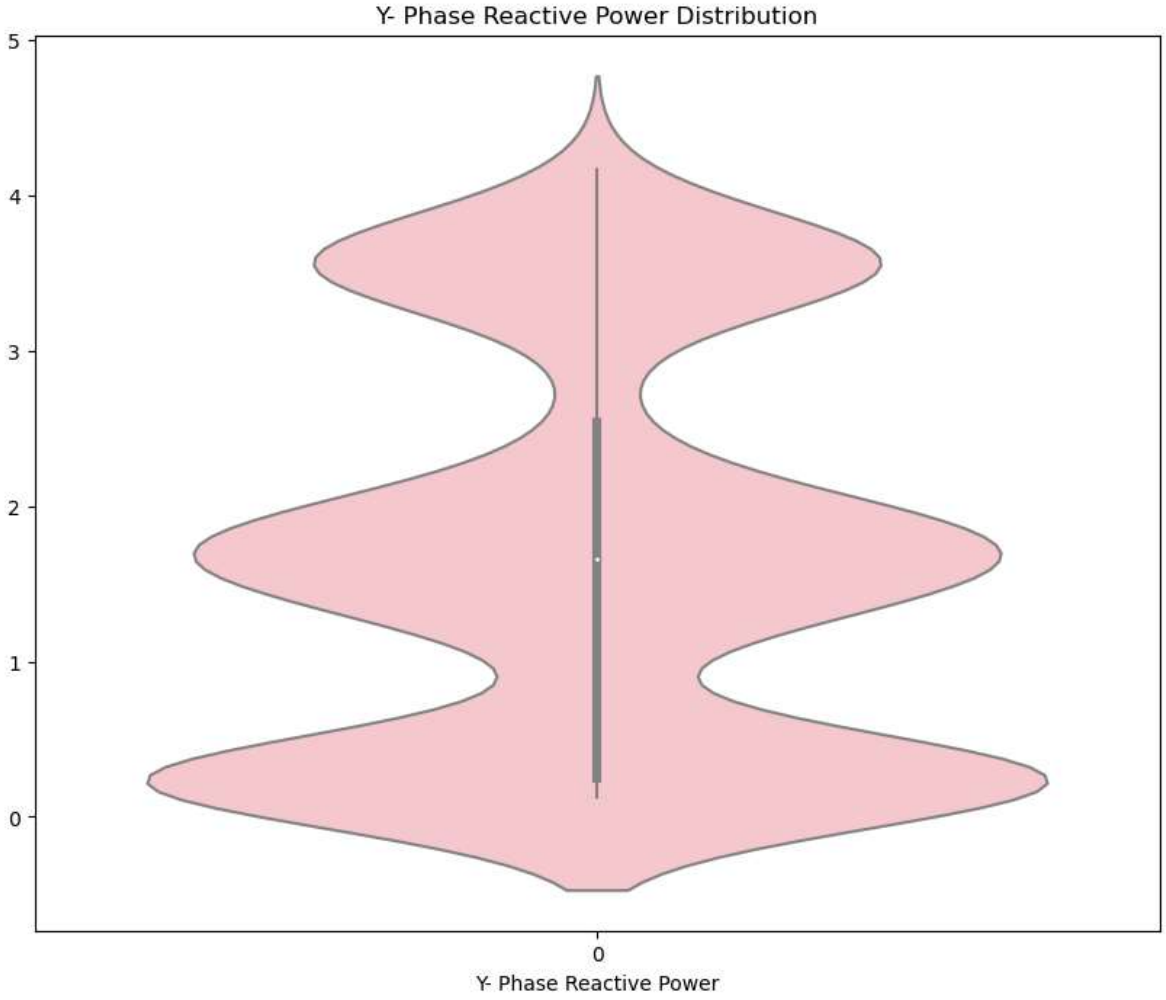

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



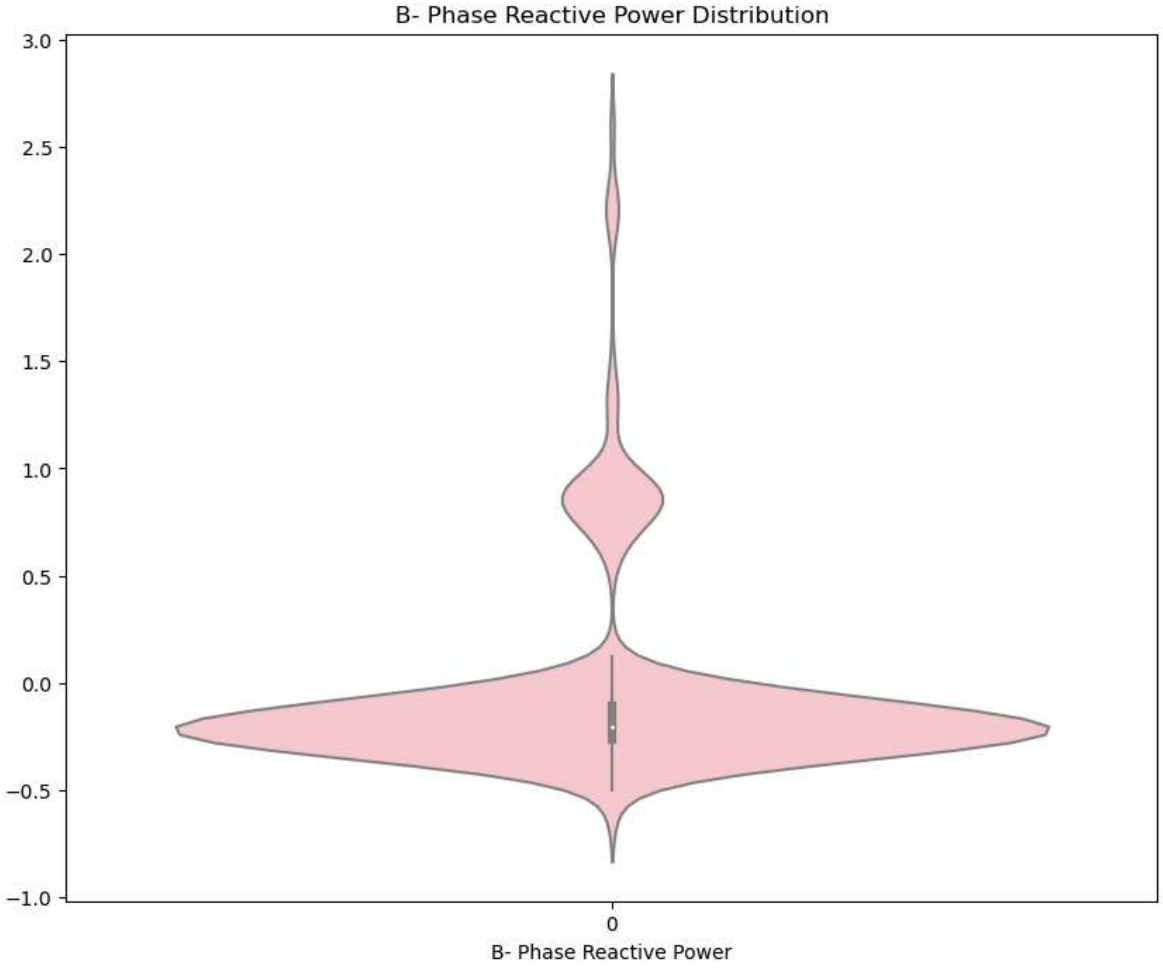

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



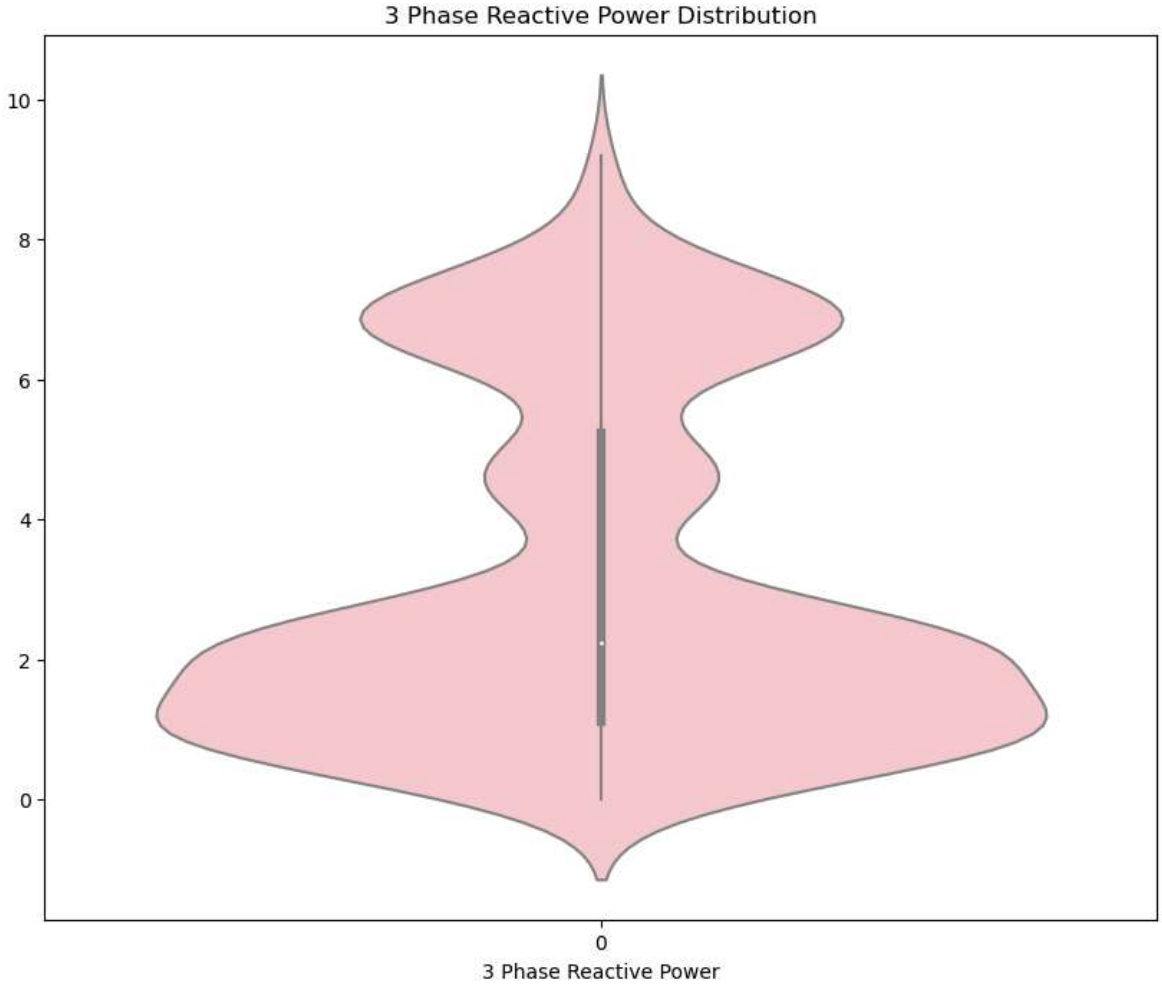

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:486: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



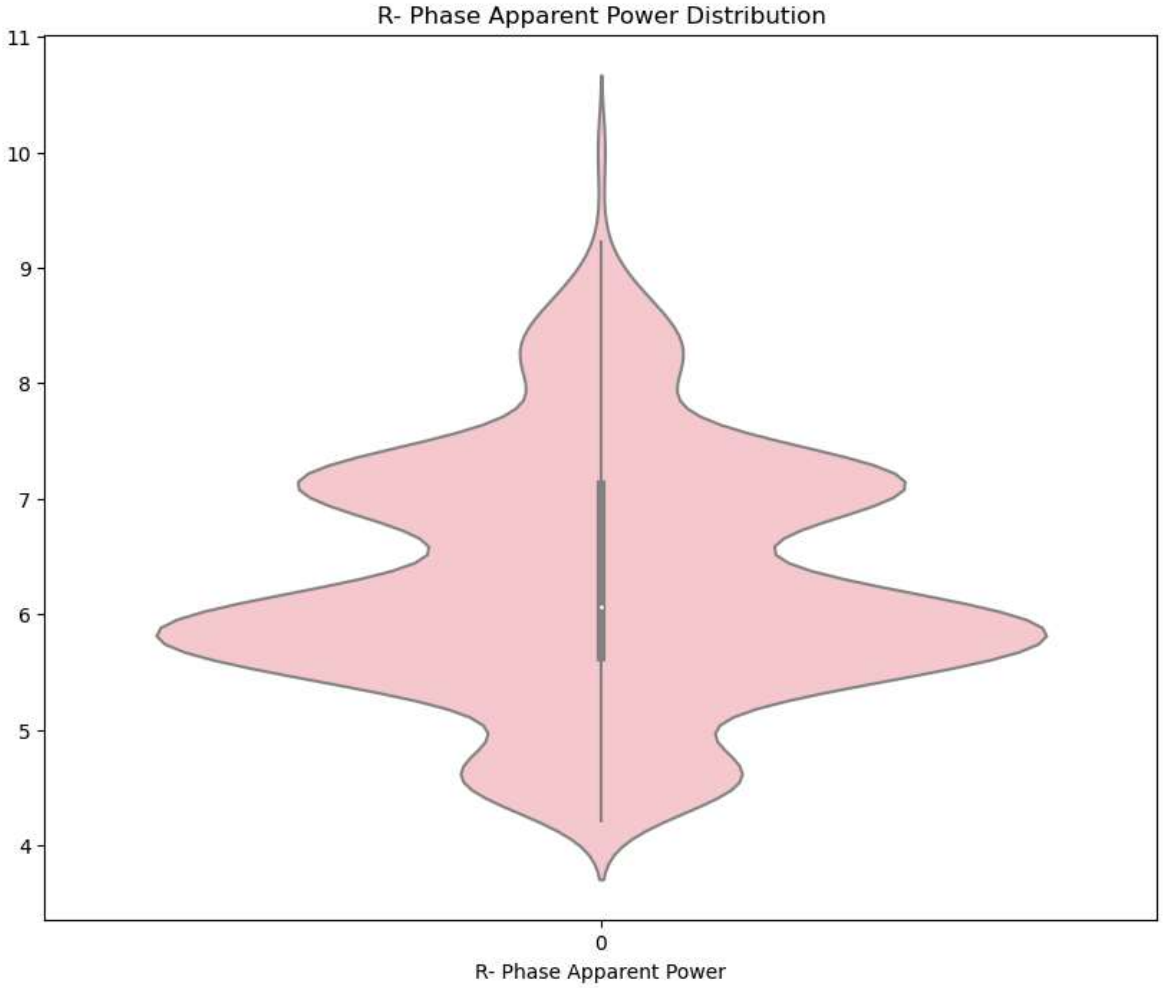

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:486: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



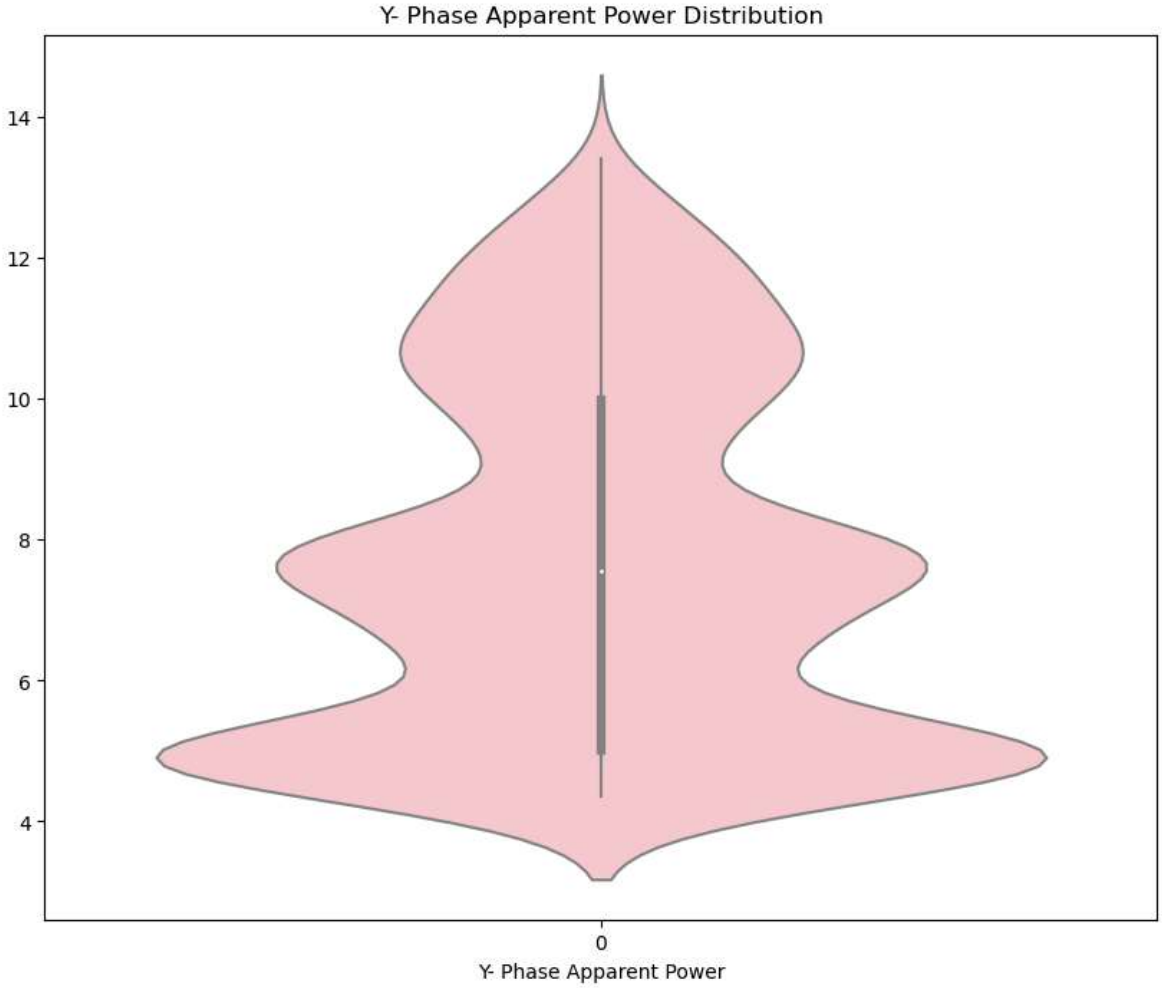

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



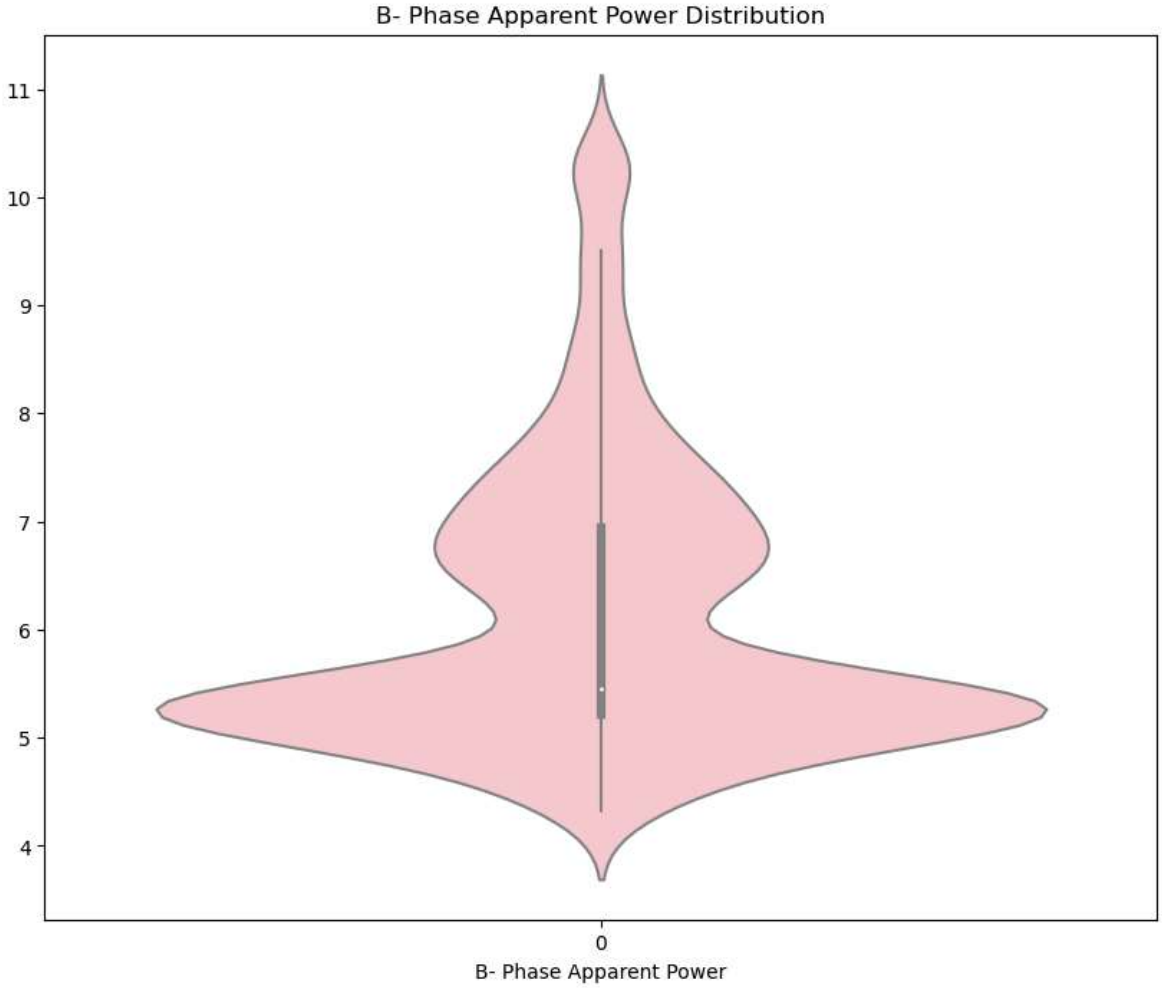

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



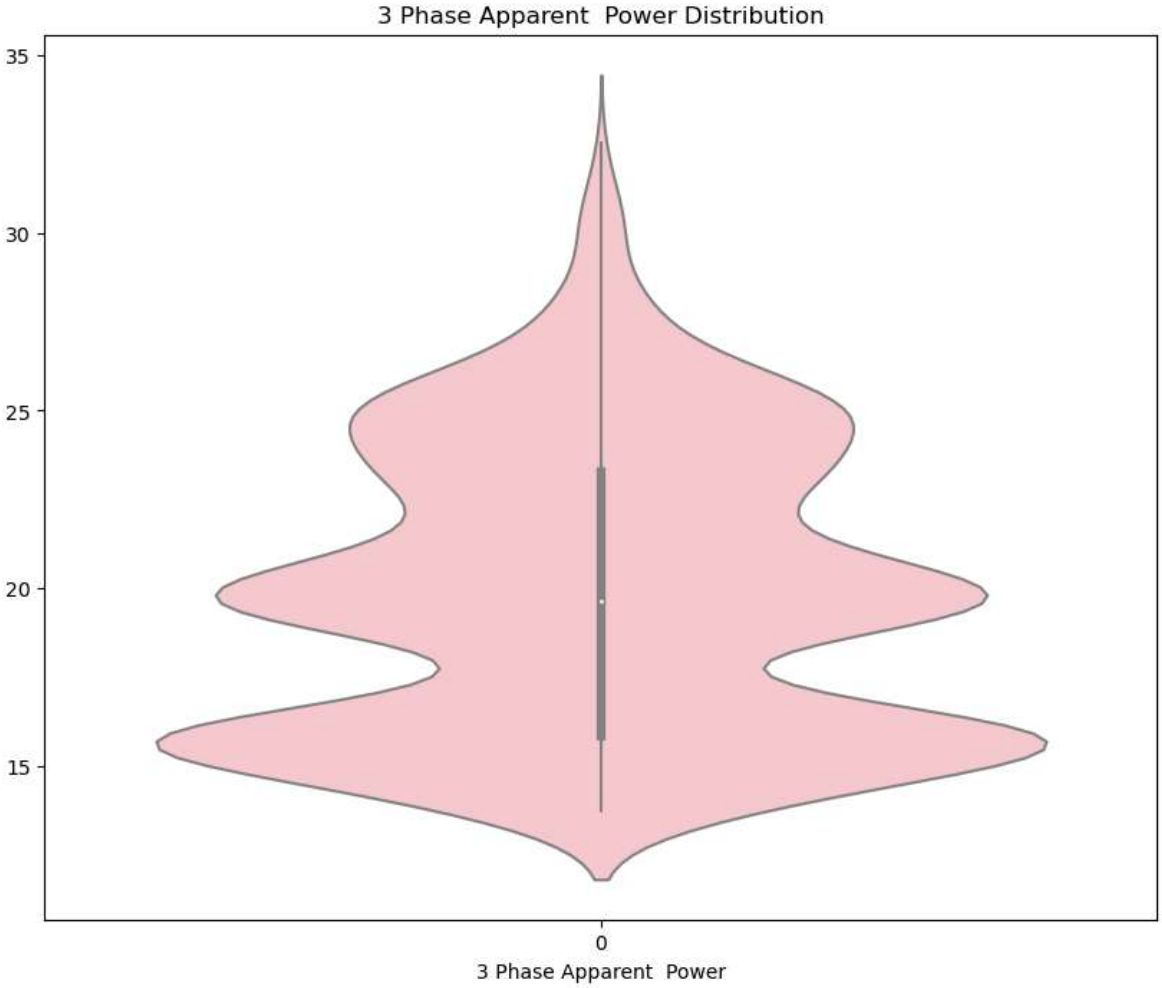

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```




```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```




```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



In [12]:
```
print("\033[1mMean\033[0m")
print()
data_minute_avg.mean()
```

Mean

Out[12]:
```
R Ph Voltage                233.803370
Y Ph Voltage                232.069130
B Ph Voltage                233.211355
Average Phase Voltage       233.028390
RY Line Voltage             404.218610
YB Line Voltage             401.888191
BR Line Voltage             404.724051
R Phase Line current         26.974626
Y Phase Line current         33.126499
B Phase Line current         26.372053
Neutral Line current         10.956307
R Phase Active Current       25.755684
Y Phase Active Current       31.950946
B Phase Active Current       26.302705
R Phase Reactive Current      6.753064
Y Phase Reactive Current      7.156251
B Phase Reactive Current     -0.220495
R- Phase Active Power         5.946020
Y- Phase Active Power         7.332985
B- Phase Active Power         6.060556
3 Phase Active Power         19.338353
R- Phase Reactive Power       1.579657
Y- Phase Reactive Power       1.659164
B- Phase Reactive Power      -0.053623
3 Phase Reactive Power        3.185850
R- Phase Apparent Power       6.302558
Y- Phase Apparent Power       7.653997
B- Phase Apparent Power       6.130370
3 Phase Apparent  Power      19.926172
dtype: float64
```

In [13]:
```
print("\033[1mMedian\033[0m")
print()
data_minute_avg.median()
```

Median

Out[13]:
```
R Ph Voltage                234.002500
Y Ph Voltage                232.836667
B Ph Voltage                233.604167
Average Phase Voltage       233.316667
RY Line Voltage             404.715000
YB Line Voltage             402.435833
BR Line Voltage             405.300000
R Phase Line current         26.135833
Y Phase Line current         32.336667
B Phase Line current         23.060833
Neutral Line current         10.972500
R Phase Active Current       25.870000
Y Phase Active Current       28.778333
B Phase Active Current       23.025833
R Phase Reactive Current      4.414167
Y Phase Reactive Current      7.386667
B Phase Reactive Current     -0.874167
R- Phase Active Power         5.955833
Y- Phase Active Power         6.695000
B- Phase Active Power         5.390000
3 Phase Active Power         18.297500
R- Phase Reactive Power       1.040000
Y- Phase Reactive Power       1.657500
B- Phase Reactive Power      -0.203333
3 Phase Reactive Power        2.239167
R- Phase Apparent Power       6.065000
Y- Phase Apparent Power       7.557500
B- Phase Apparent Power       5.450000
3 Phase Apparent  Power      19.655000
dtype: float64
```

In [14]:
```
print("\033[1mMode\033[0m")
print()
data_minute_avg.mode()
```

Mode

Out[14]:

	R Ph Voltage	Y Ph Voltage	B Ph Voltage	Average Phase Voltage	RY Line Voltage	YB Line Voltage	BR Line Voltage	
0	232.573333	228.783333	232.306667	225.630000	404.096667	401.608333	405.3	2
1	233.233333	230.520000	232.331667	228.840000	410.515000	403.000000	NaN	2
2	233.868333	232.836667	232.335000	233.225000	NaN	NaN	NaN	2
3	235.586667	235.220000	232.626667	233.840000	NaN	NaN	NaN	2
4	236.288333	235.585000	233.541667	236.863333	NaN	NaN	NaN	2
5	236.300000	NaN	234.113333	237.076667	NaN	NaN	NaN	2
6	236.380000	NaN	234.278333	NaN	NaN	NaN	NaN	2
7	236.420000	NaN	234.873333	NaN	NaN	NaN	NaN	2
8	236.770000	NaN	235.306667	NaN	NaN	NaN	NaN	3
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3
11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3

12 rows × 29 columns

In [15]:
```
print("\033[1mStandard Deviation\033[0m")
print()
for col in data_minute_avg.columns:
    print(col, ":", stats.stdev(data_minute_avg[col]))
```

Standard Deviation

```
R Ph Voltage : 3.171990104165955
Y Ph Voltage : 4.4303111461137785
B Ph Voltage : 3.9484335356483107
Average Phase Voltage : 3.7094198194260133
RY Line Voltage : 6.390126820709543
YB Line Voltage : 6.471920320128618
BR Line Voltage : 6.429869152424708
R Phase Line current : 4.723051310280334
Y Phase Line current : 11.42604559426256
B Phase Line current : 6.175437767807627
Neutral Line current : 5.62241095725543
R Phase Active Current : 3.926010492875708
Y Phase Active Current : 11.21629797062077
B Phase Active Current : 6.137722039637347
R Phase Reactive Current : 5.078568639157112
Y Phase Reactive Current : 5.474148615486598
B Phase Reactive Current : 1.9770432121067631
R- Phase Active Power : 0.8781876162327186
Y- Phase Active Power : 2.477352677295844
B- Phase Active Power : 1.3407915350235111
3 Phase Active Power : 4.0749022092344465
R- Phase Reactive Power : 1.189909335784449
Y- Phase Reactive Power : 1.2821559839800512
B- Phase Reactive Power : 0.45585296713415435
3 Phase Reactive Power : 2.4669162846075516
R- Phase Apparent Power : 1.0754180486456097
Y- Phase Apparent Power : 2.530082547940558
B- Phase Apparent Power : 1.3436217221634572
3 Phase Apparent  Power : 4.125061636849044
```

In [16]:
```
print("\033[1mVariance\033[0m")
print()
for col in data_minute_avg.columns:
    print(col, ":", stats.variance(data_minute_avg[col]))
```

Variance

```
R Ph Voltage : 10.061521220926744
Y Ph Voltage : 19.62765685137998
B Ph Voltage : 15.590127385432218
Average Phase Voltage : 13.759795396750517
RY Line Voltage : 40.833720784751456
YB Line Voltage : 41.88575263009371
BR Line Voltage : 41.34321731730283
R Phase Line current : 22.30721367954078
Y Phase Line current : 130.55451792216684
B Phase Line current : 38.13603162406485
Neutral Line current : 31.611504972265923
R Phase Active Current : 15.413558390170161
Y Phase Active Current : 125.80534016575163
B Phase Active Current : 37.67163183585003
R Phase Reactive Current : 25.791859422630118
Y Phase Reactive Current : 29.966303064433838
B Phase Reactive Current : 3.9086998625374276
R- Phase Active Power : 0.7712134893045046
Y- Phase Active Power : 6.137276287704886
B- Phase Active Power : 1.7977219403907034
3 Phase Active Power : 16.60482801482377
R- Phase Reactive Power : 1.4158842273869885
Y- Phase Reactive Power : 1.6439239672558532
B- Phase Reactive Power : 0.2078019276450124
3 Phase Reactive Power : 6.0856759552619275
R- Phase Apparent Power : 1.156523979352731
Y- Phase Apparent Power : 6.4013176993933865
B- Phase Apparent Power : 1.8053193322694945
3 Phase Apparent  Power : 17.01613350780372
```

In [17]:
```
# Calculate the correlation matrix
correlation_matrix = data_minute_avg.corr()
correlation_matrix.head(29)
```

Out[17]:

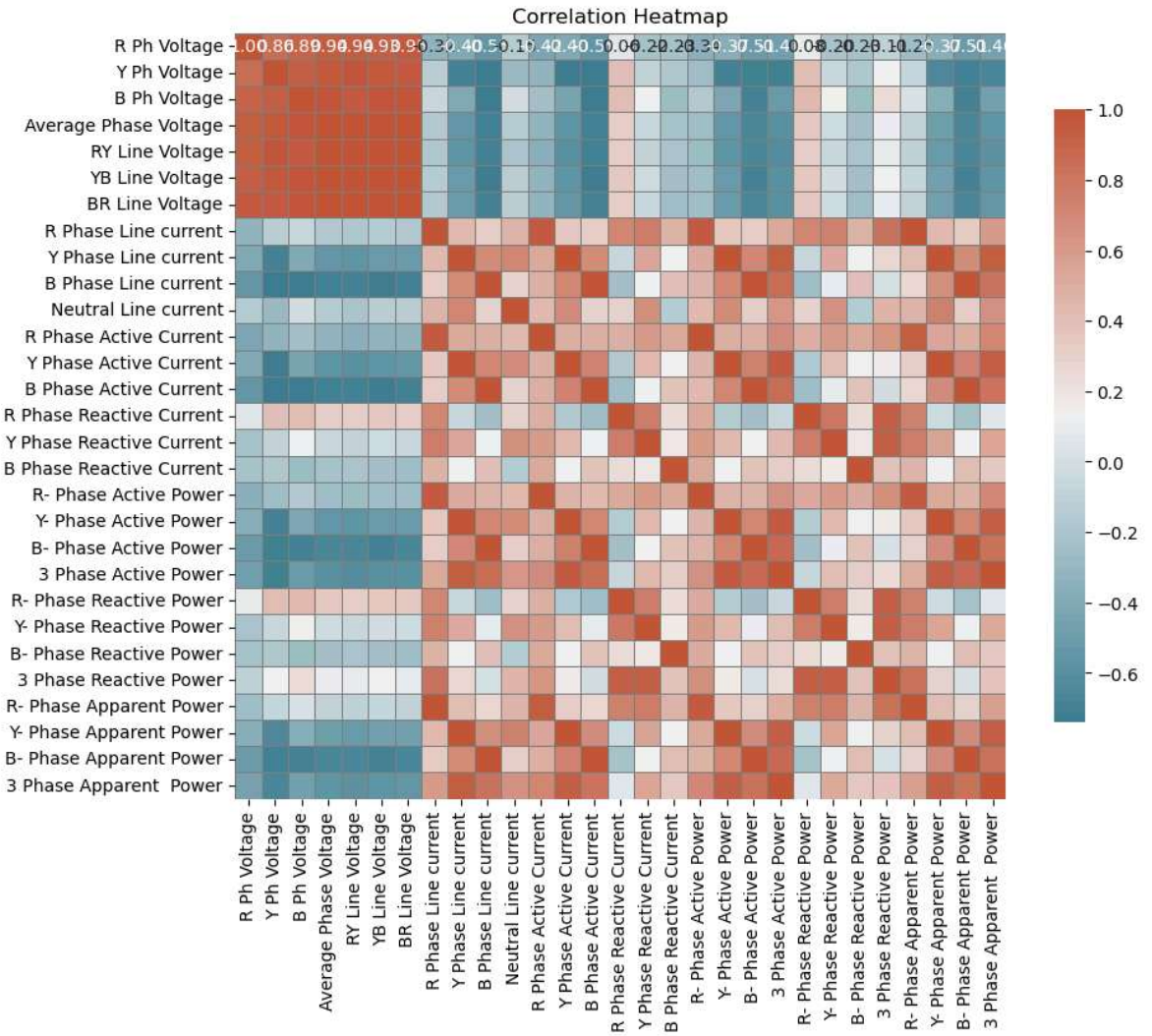
	R Ph Voltage	Y Ph Voltage	B Ph Voltage	Average Phase Voltage	RY Line Voltage	YB Line Voltage	BR Line Voltage	
R Ph Voltage	1.000000	0.858136	0.891371	0.942816	0.943962	0.934201	0.946953	-(
Y Ph Voltage	0.858136	1.000000	0.919343	0.968651	0.974096	0.966600	0.961587	-(
B Ph Voltage	0.891371	0.919343	1.000000	0.974554	0.962987	0.980661	0.975656	-(
Average Phase Voltage	0.942816	0.968651	0.974554	1.000000	0.998420	0.998872	0.998772	-(
RY Line Voltage	0.943962	0.974096	0.962987	0.998420	1.000000	0.996008	0.995802	-(
YB Line Voltage	0.934201	0.966600	0.980661	0.998872	0.996008	1.000000	0.997070	-(
BR Line Voltage	0.946953	0.961587	0.975656	0.998772	0.995802	0.997070	1.000000	-(
R Phase Line current	-0.320222	-0.138245	-0.056489	-0.166530	-0.181127	-0.153022	-0.166466	1
Y Phase Line current	-0.402387	-0.687163	-0.400104	-0.530176	-0.557666	-0.519598	-0.512389	(
B Phase Line current	-0.545634	-0.737715	-0.732542	-0.708683	-0.704483	-0.719711	-0.698850	(
Neutral Line current	-0.145787	-0.275262	-0.002277	-0.152161	-0.189914	-0.137863	-0.129360	(
R Phase Active Current	-0.416939	-0.322069	-0.239846	-0.332181	-0.341901	-0.323548	-0.331032	(
Y Phase Active Current	-0.403498	-0.726187	-0.451364	-0.564147	-0.588409	-0.556703	-0.546068	(
B Phase Active Current	-0.545539	-0.739994	-0.733060	-0.709744	-0.706072	-0.720809	-0.699279	(
R Phase Reactive Current	0.064864	0.401702	0.417914	0.326219	0.309805	0.343933	0.321716	(
Y Phase Reactive Current	-0.218575	-0.091678	0.117656	-0.057452	-0.090502	-0.031537	-0.052336	(

	R Ph Voltage	Y Ph Voltage	B Ph Voltage	Average Phase Voltage	RY Line Voltage	YB Line Voltage	BR Line Voltage	
B Phase Reactive Current	-0.226652	-0.177402	-0.266550	-0.229655	-0.211957	-0.230505	-0.245881	
R- Phase Active Power	-0.339005	-0.254966	-0.165244	-0.256764	-0.266994	-0.248585	-0.254966	
Y- Phase Active Power	-0.372082	-0.698183	-0.416026	-0.531515	-0.556561	-0.523724	-0.513169	
B- Phase Active Power	-0.507007	-0.713740	-0.698537	-0.676030	-0.672949	-0.687314	-0.664852	
3 Phase Active Power	-0.465383	-0.713762	-0.517809	-0.600306	-0.616733	-0.597533	-0.585071	
R- Phase Reactive Power	0.080393	0.413796	0.431228	0.340197	0.323700	0.357765	0.335850	
Y- Phase Reactive Power	-0.196123	-0.064248	0.143029	-0.031133	-0.064150	-0.005185	-0.026210	
B- Phase Reactive Power	-0.227437	-0.178937	-0.268776	-0.231266	-0.213531	-0.232219	-0.247470	
3 Phase Reactive Power	-0.105323	0.133236	0.232671	0.105177	0.083337	0.126960	0.102644	
R- Phase Apparent Power	-0.245333	-0.070397	0.017143	-0.092021	-0.107216	-0.078766	-0.091395	
Y- Phase Apparent Power	-0.368681	-0.653643	-0.359397	-0.492791	-0.521209	-0.481685	-0.474818	
B- Phase Apparent Power	-0.508696	-0.710940	-0.698284	-0.675309	-0.671822	-0.686471	-0.664741	
3 Phase Apparent Power	-0.460145	-0.667879	-0.466605	-0.562414	-0.580930	-0.556945	-0.548265	

29 rows × 29 columns

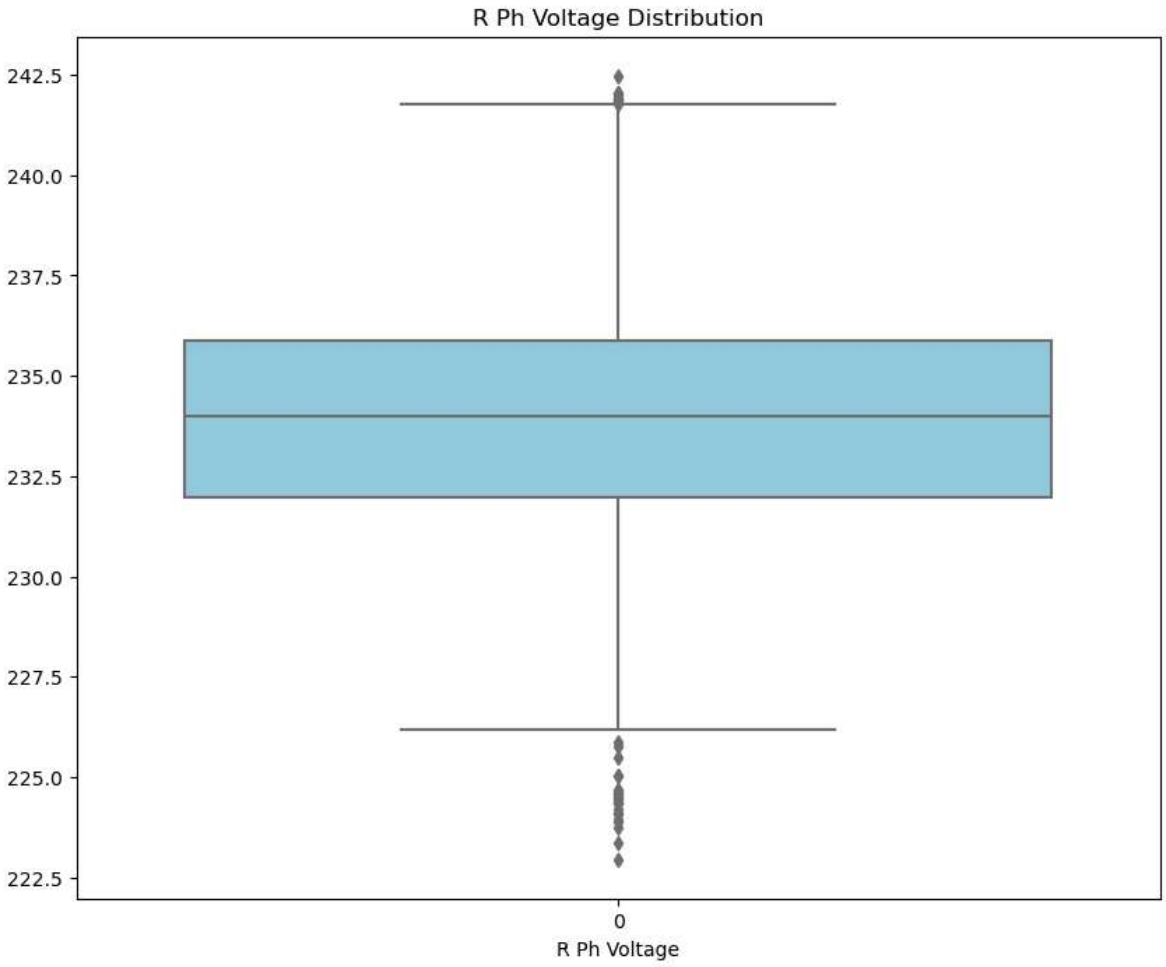
In [22]:
```
plt.figure(figsize = (10, 8))
custom_palette = sns.diverging_palette(220, 20, as_cmap = True)
sns.heatmap(correlation_matrix, annot = True, cmap = custom_palette, fmt = ".2f"
```

```
            linewidths = 0.5, linecolor = 'gray', cbar_kws = {"shrink": 0.8})
plt.xticks(rotation = 90)
plt.yticks(rotation = 0)
plt.title("Correlation Heatmap")
plt.show()
```

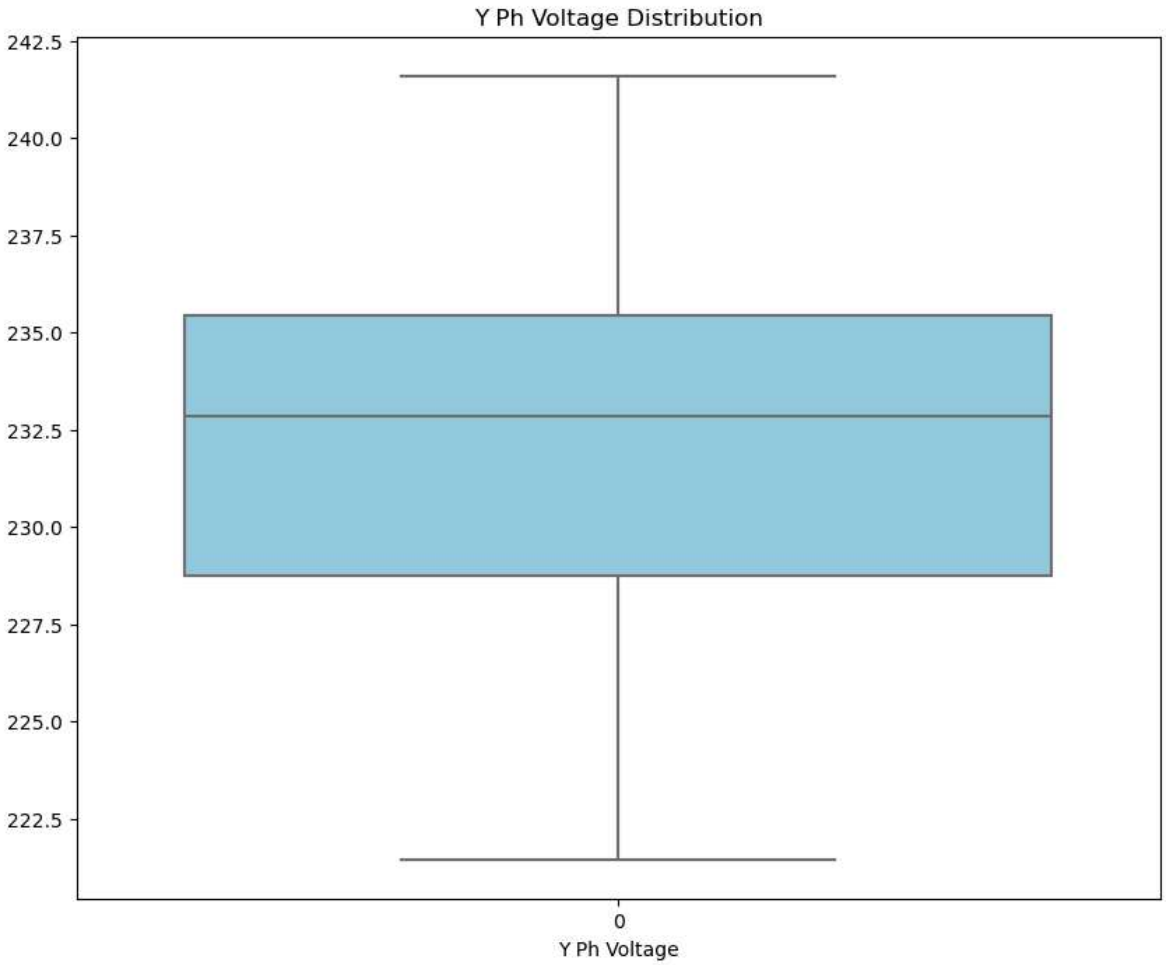


In [19]:
```
for col in data_minute_avg.columns:
    plt.figure(figsize=(10, 8))
    sns.boxplot(data=data_minute_avg[col], color='skyblue')
    tit = col + " Distribution"
    plt.title(tit)
    plt.xlabel(col)
    plt.show()
```

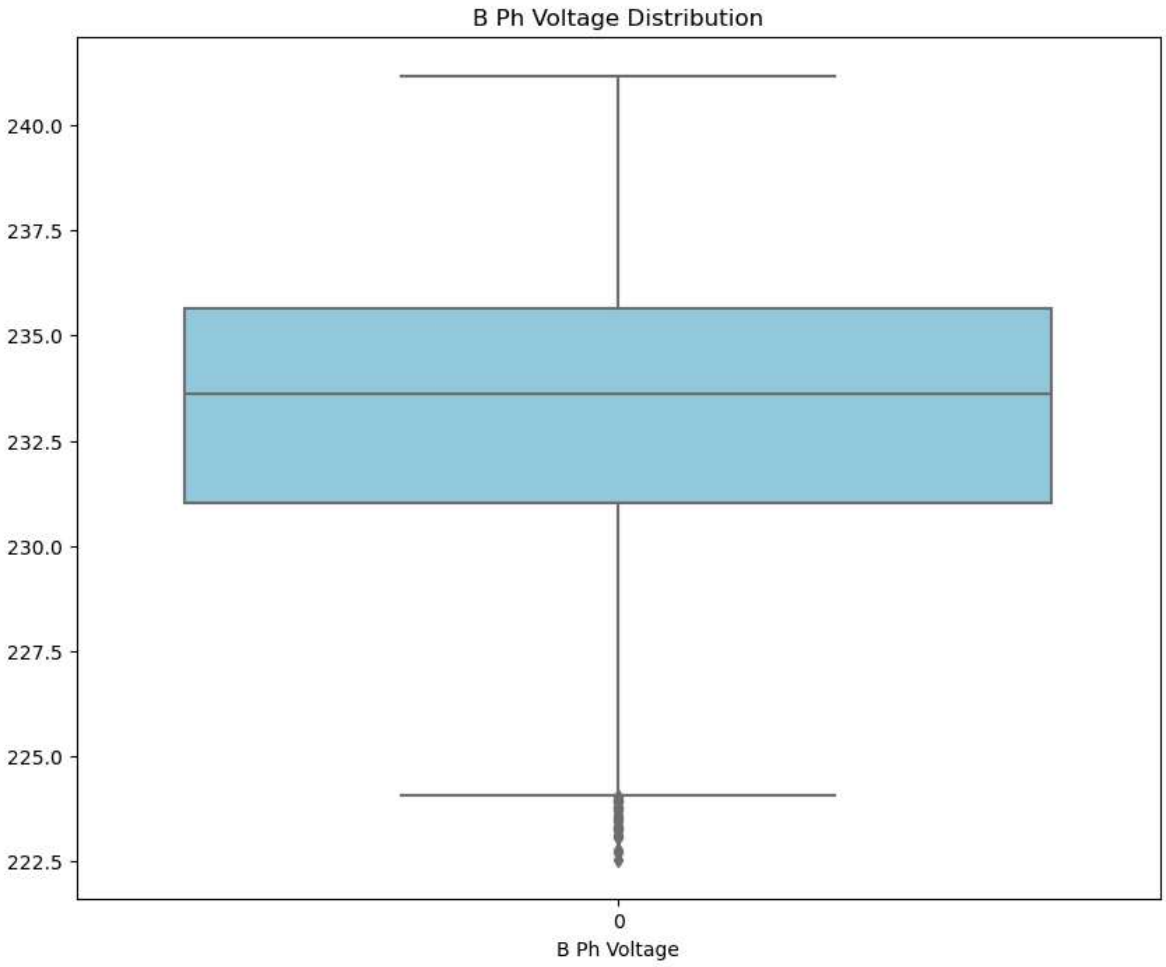
```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:486: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



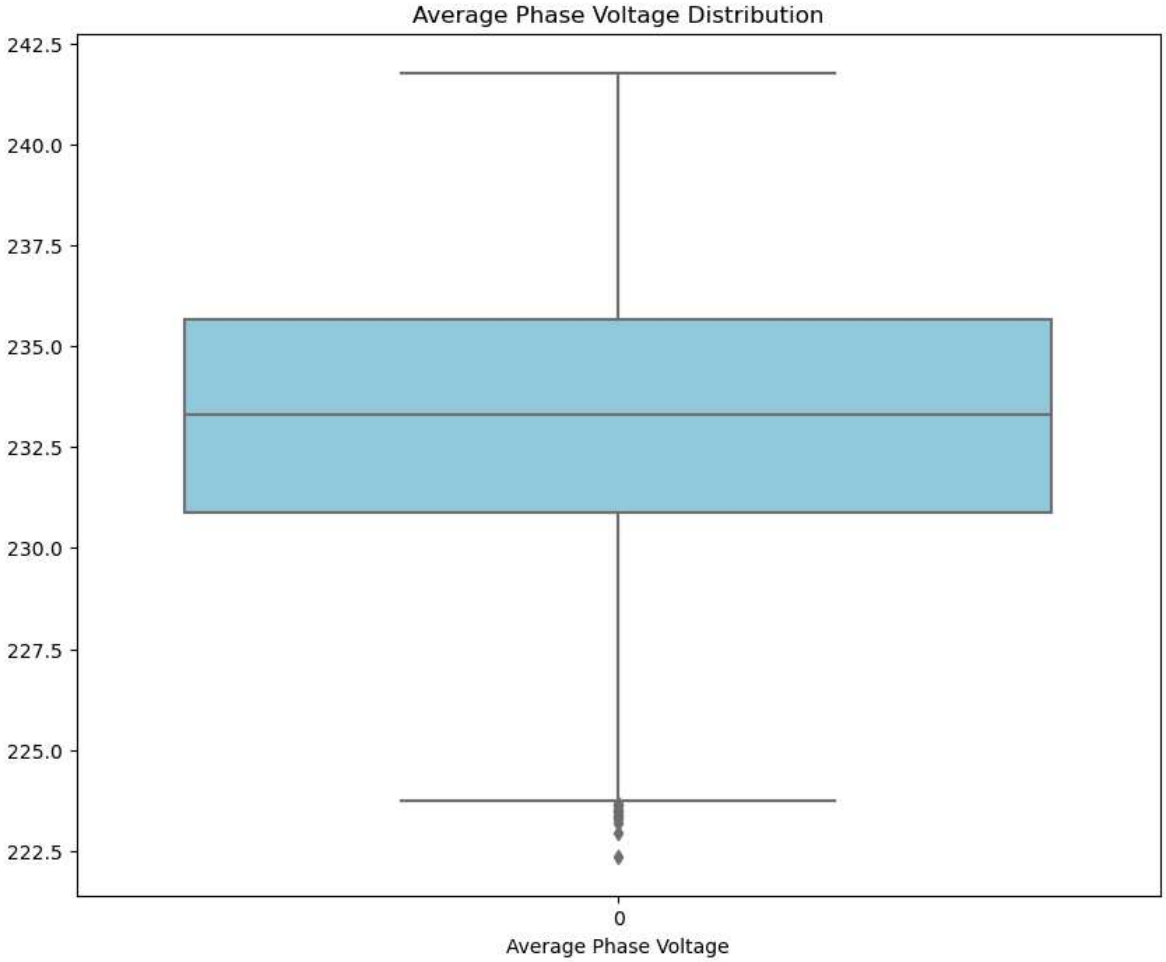

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:486: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



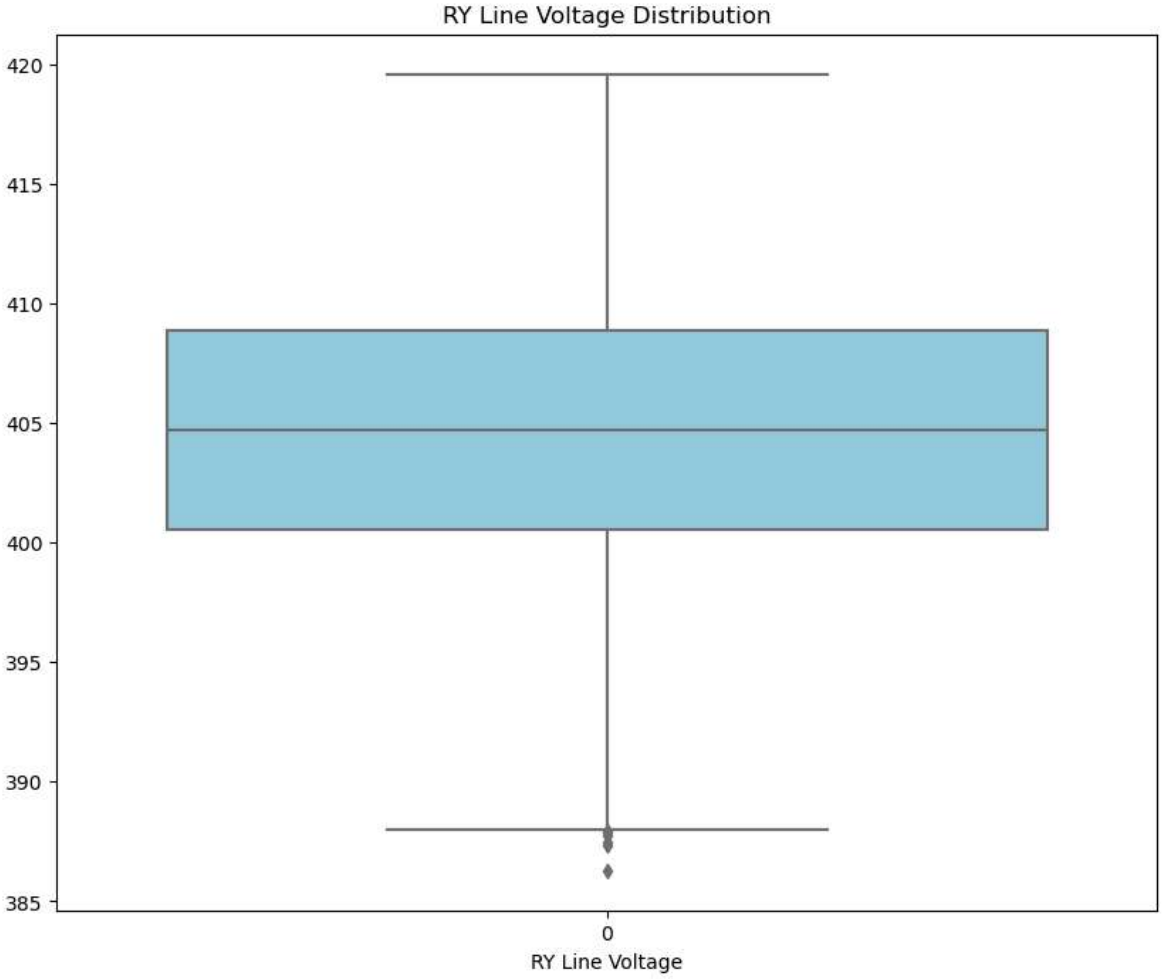

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



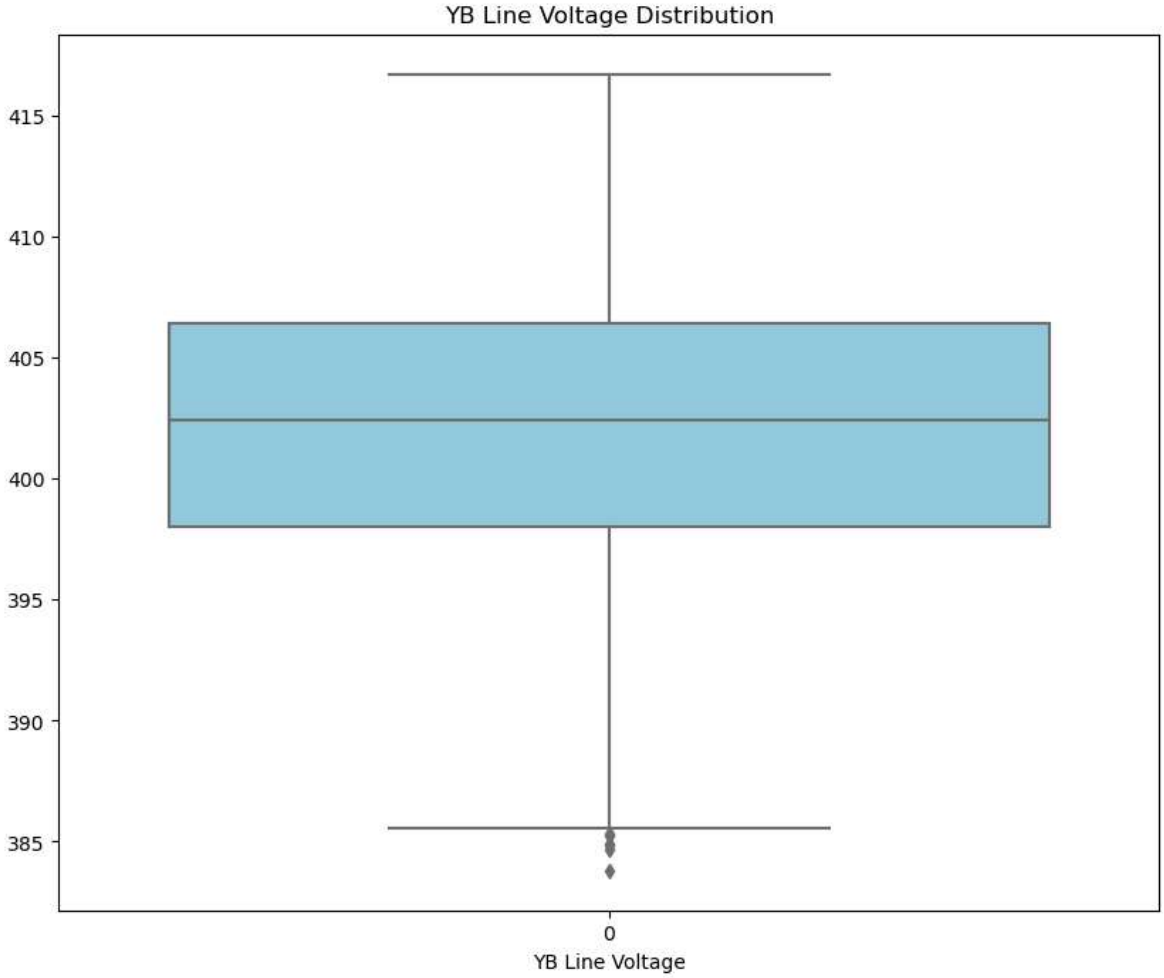

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



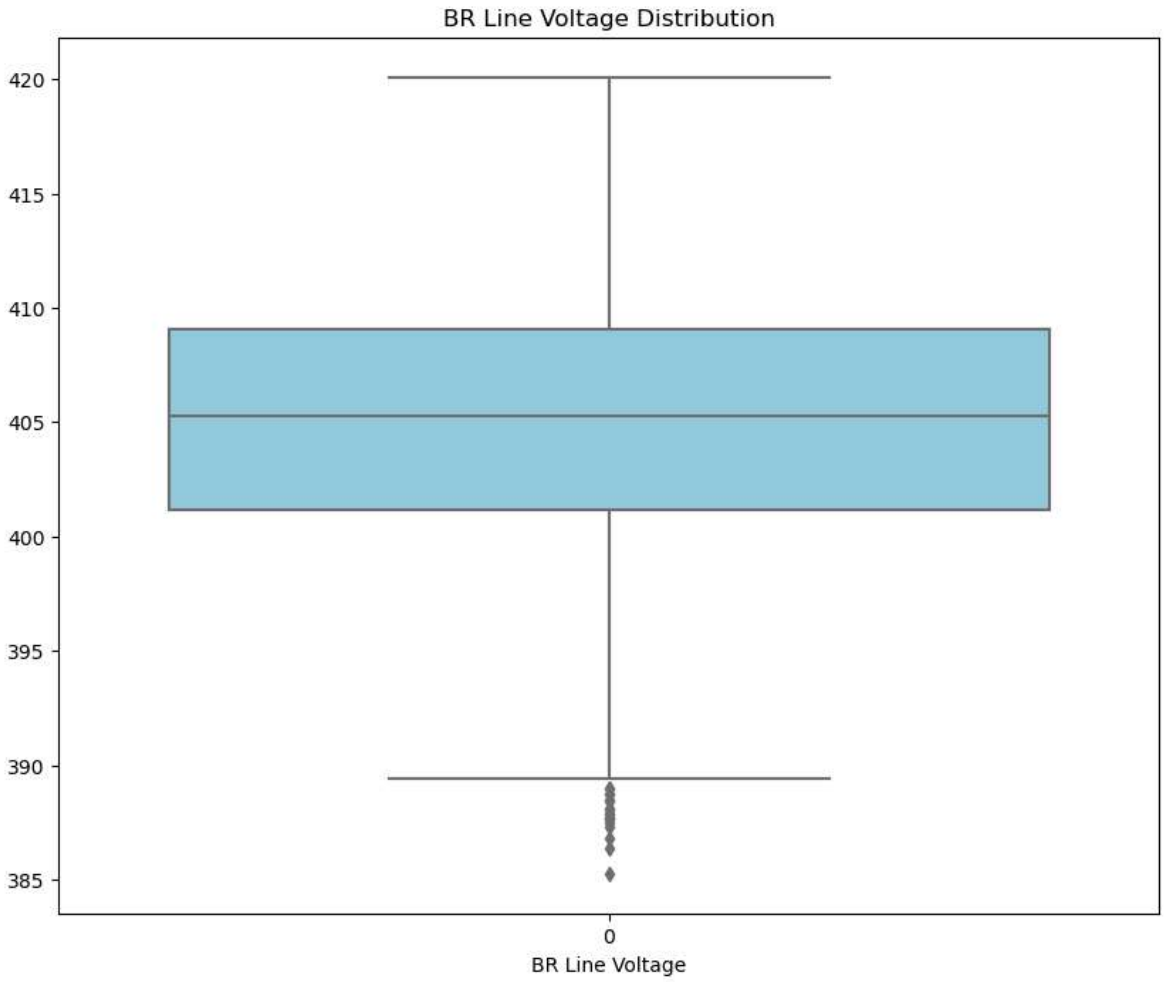

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



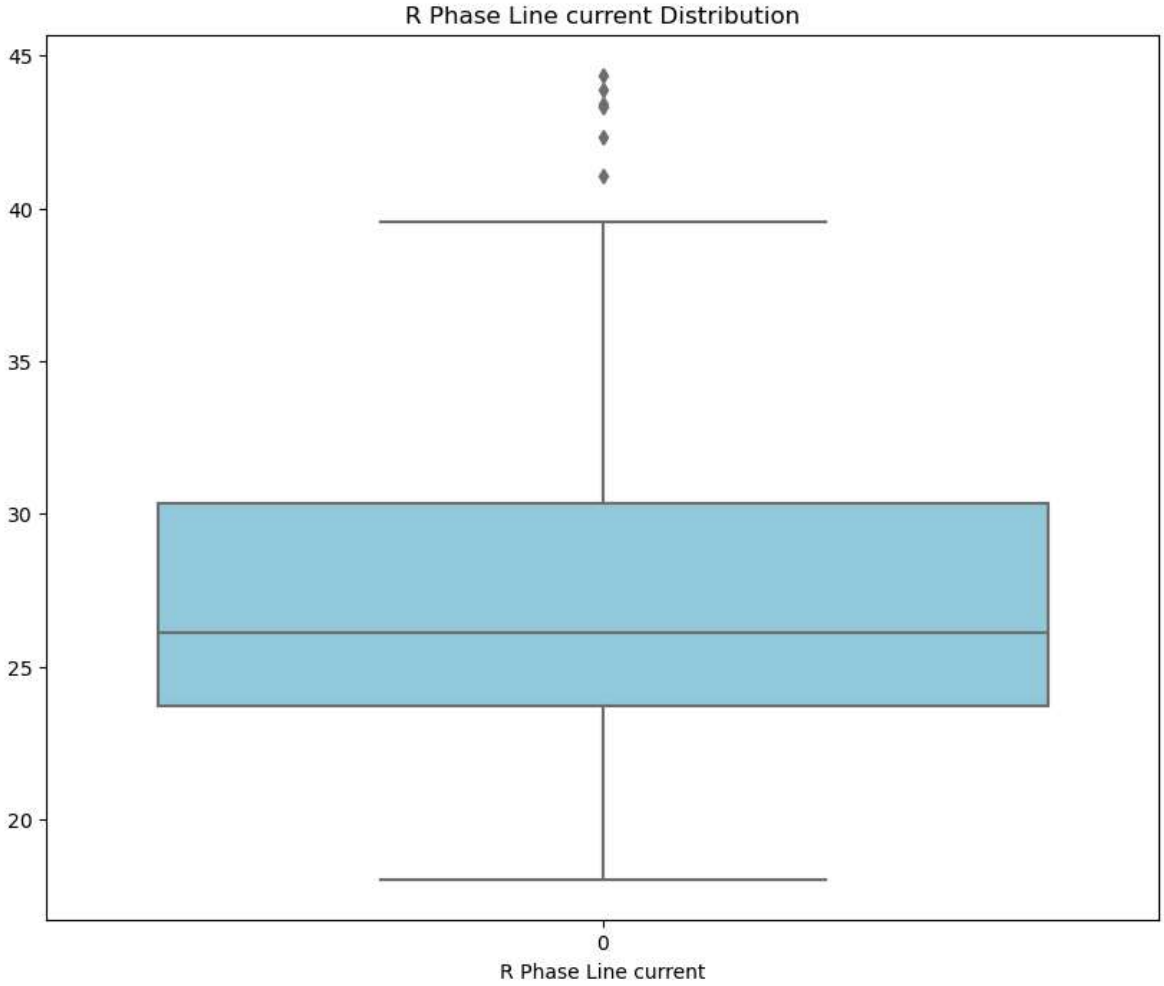

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



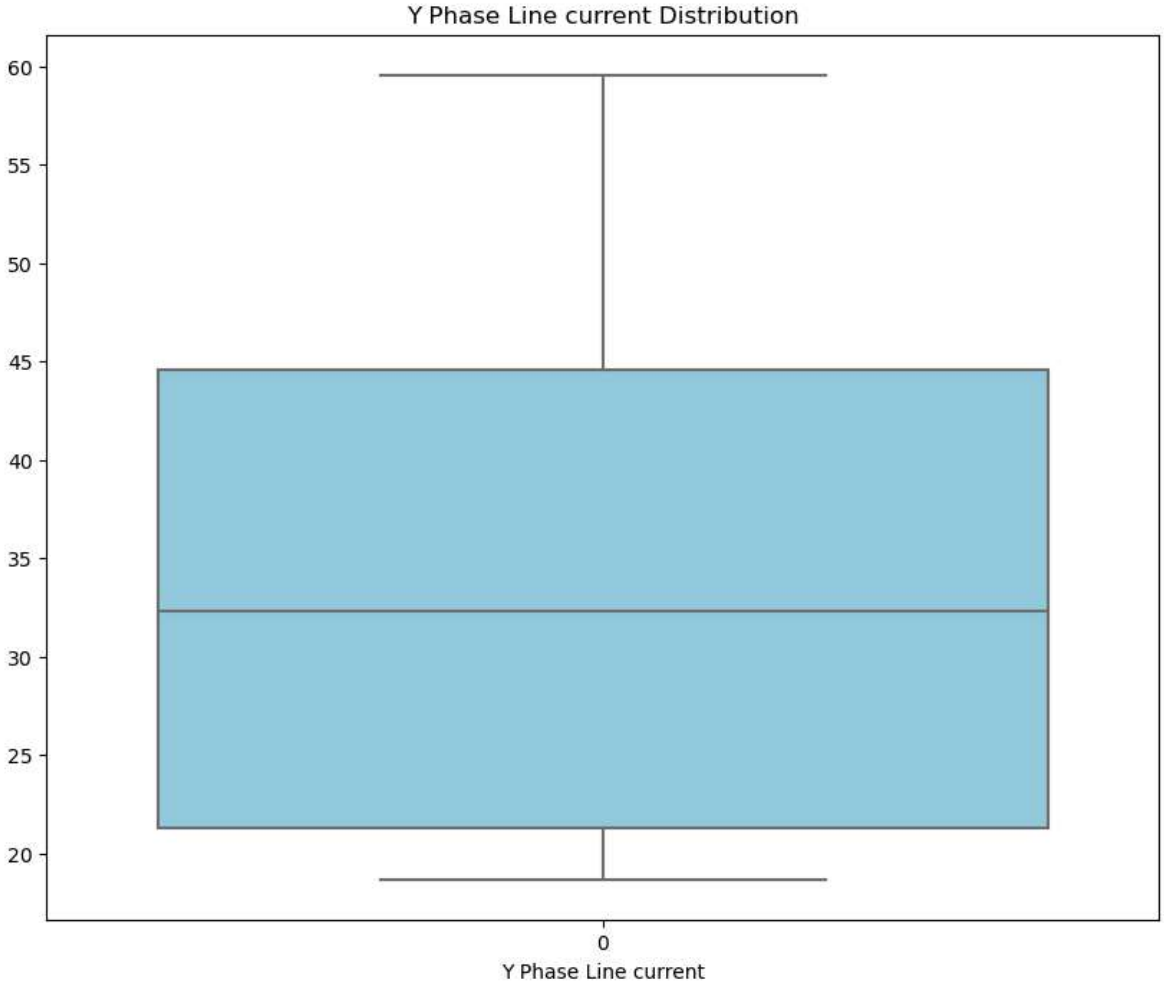

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:486: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



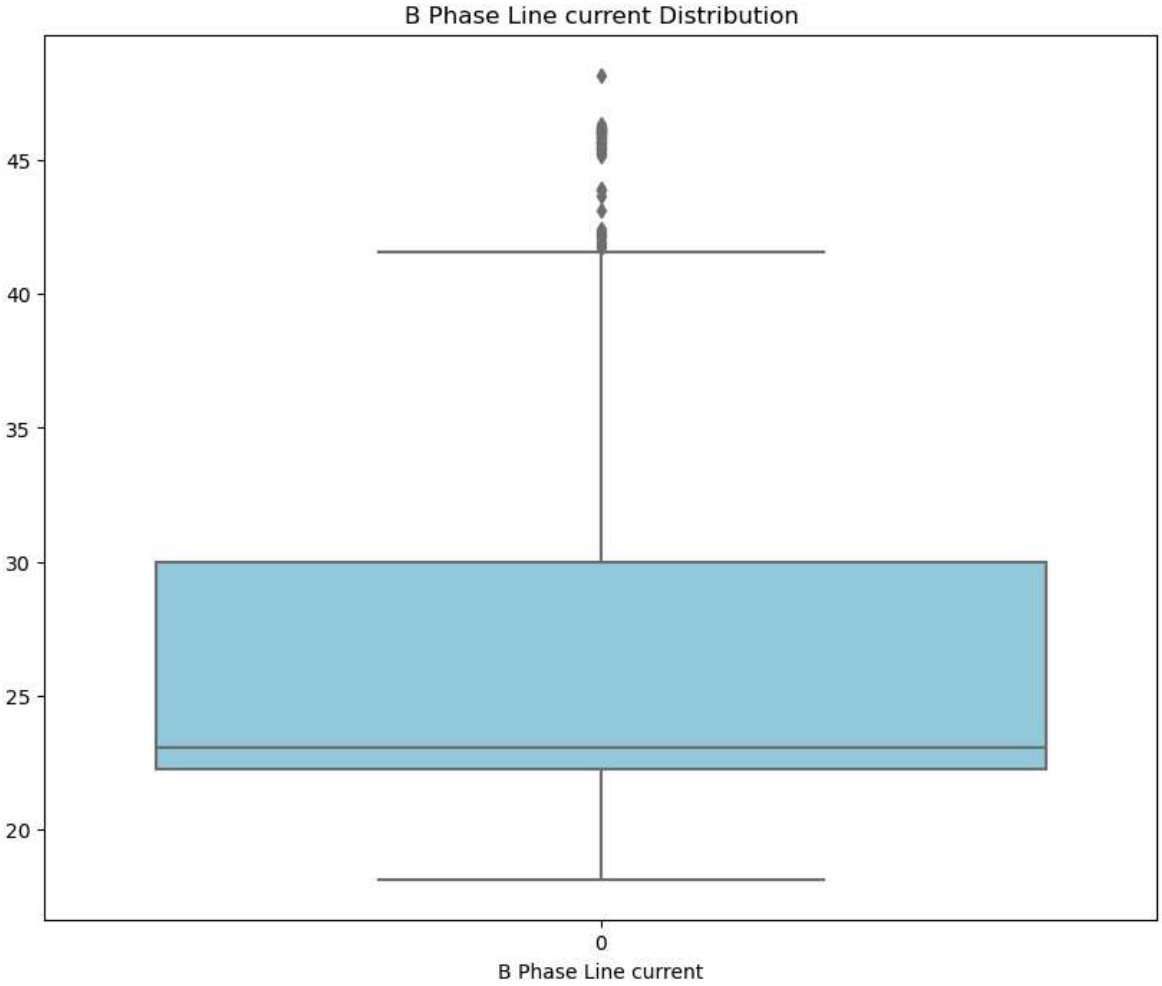

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



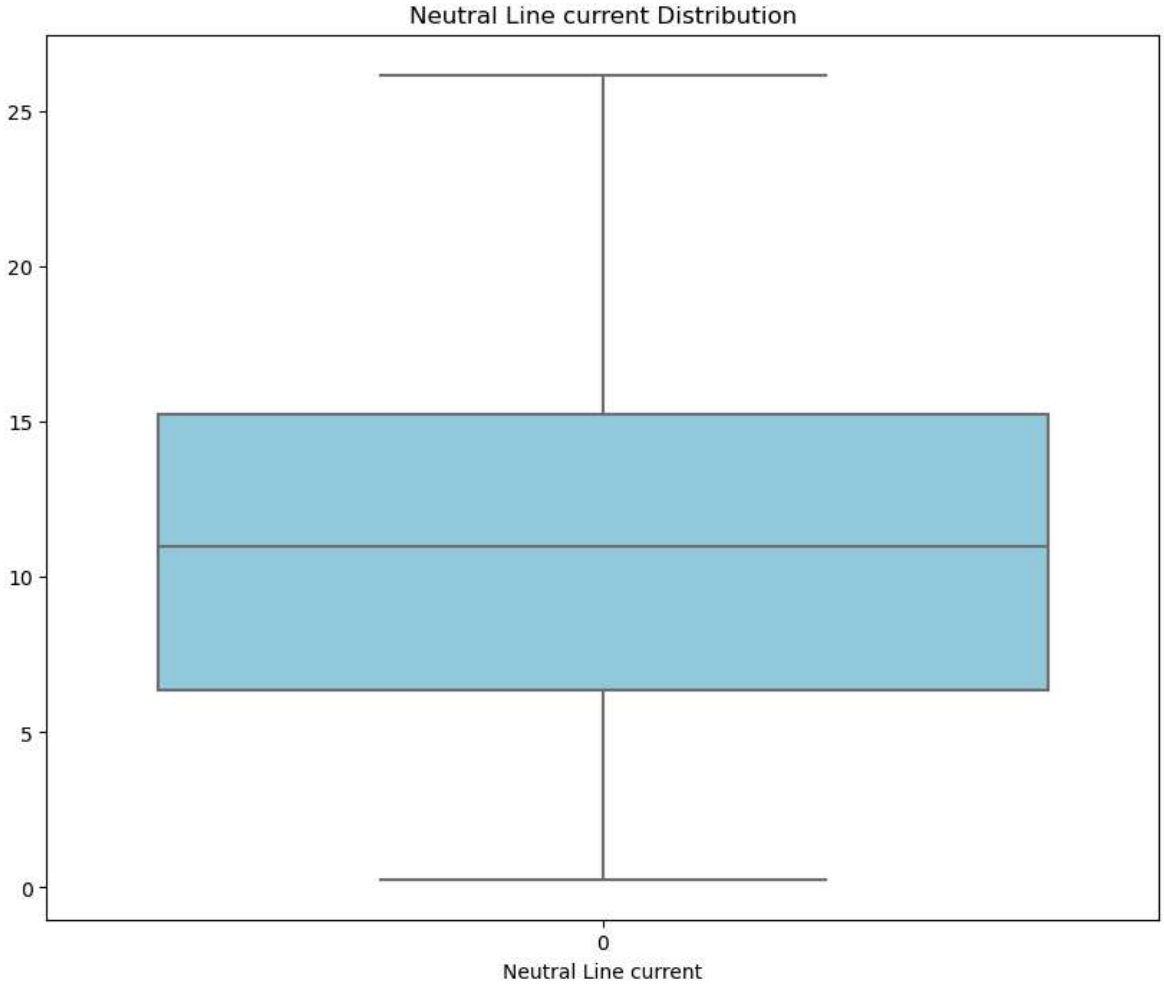

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



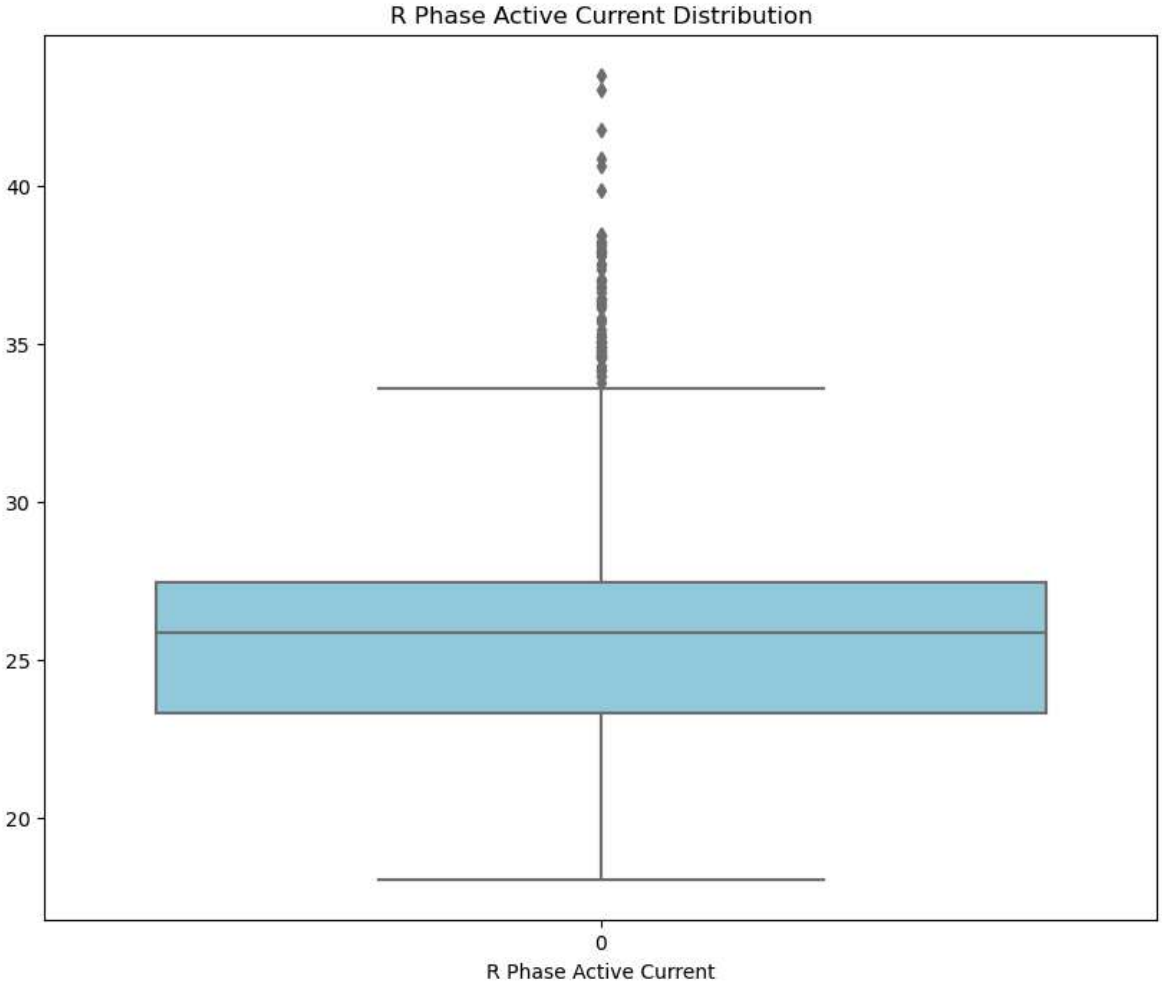

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



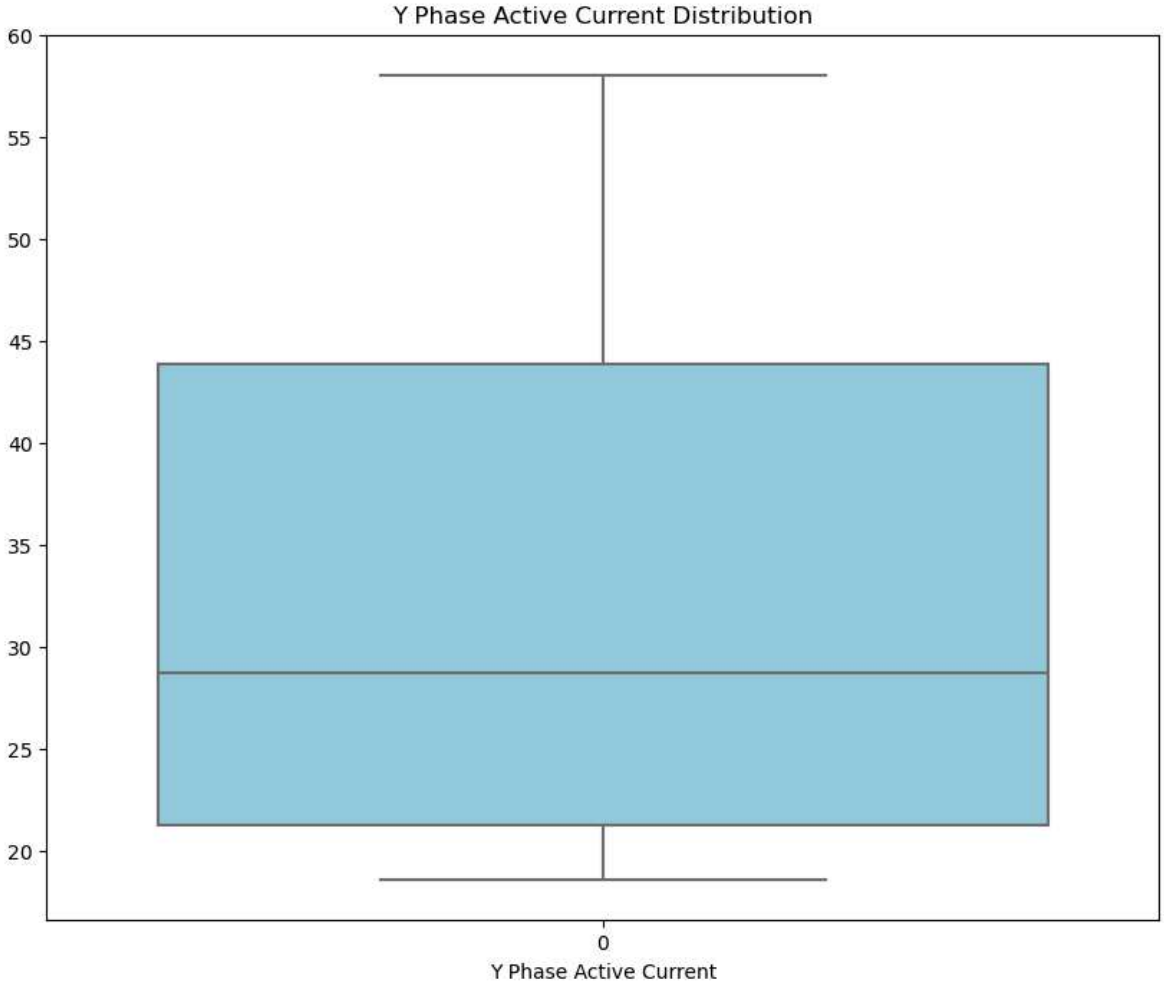

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



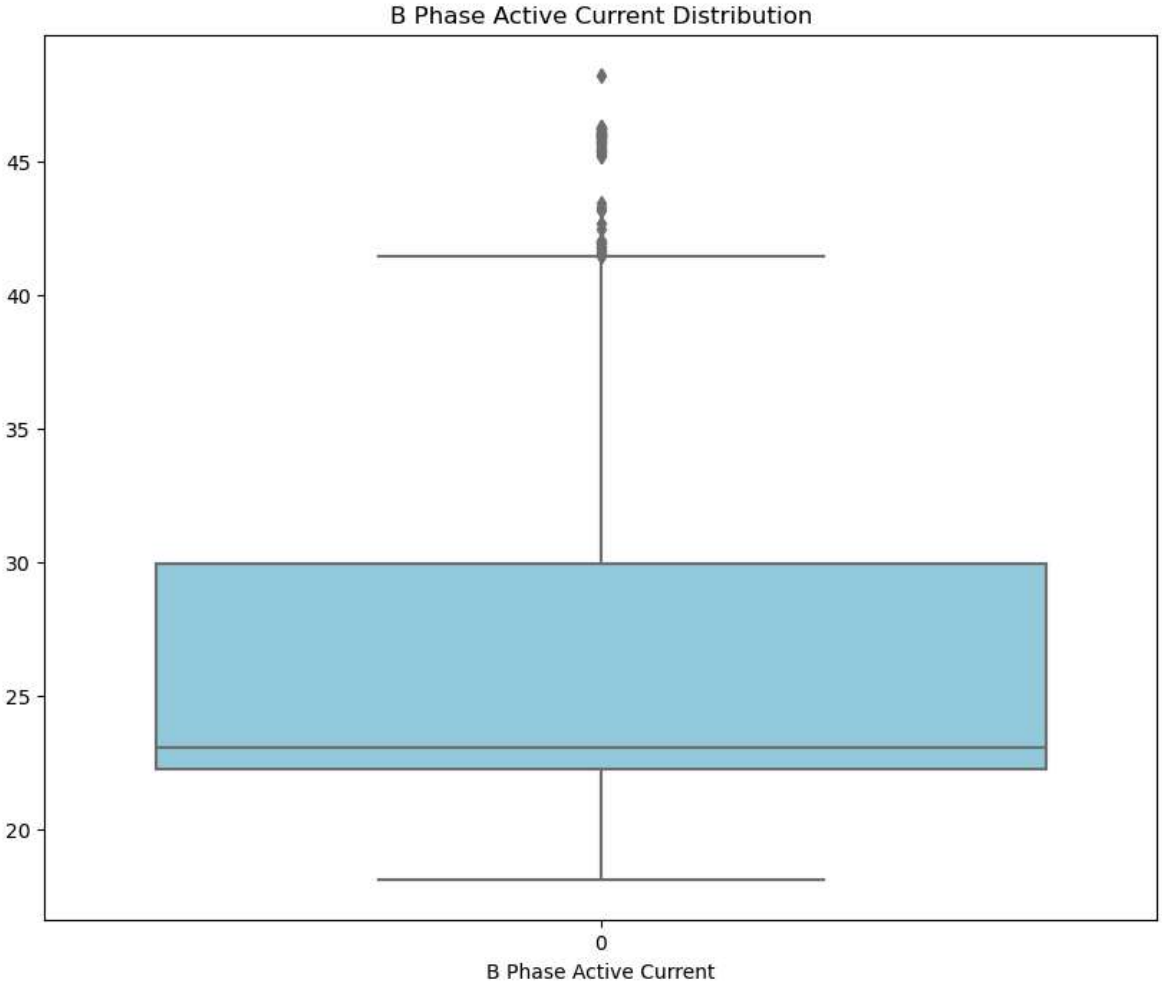

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



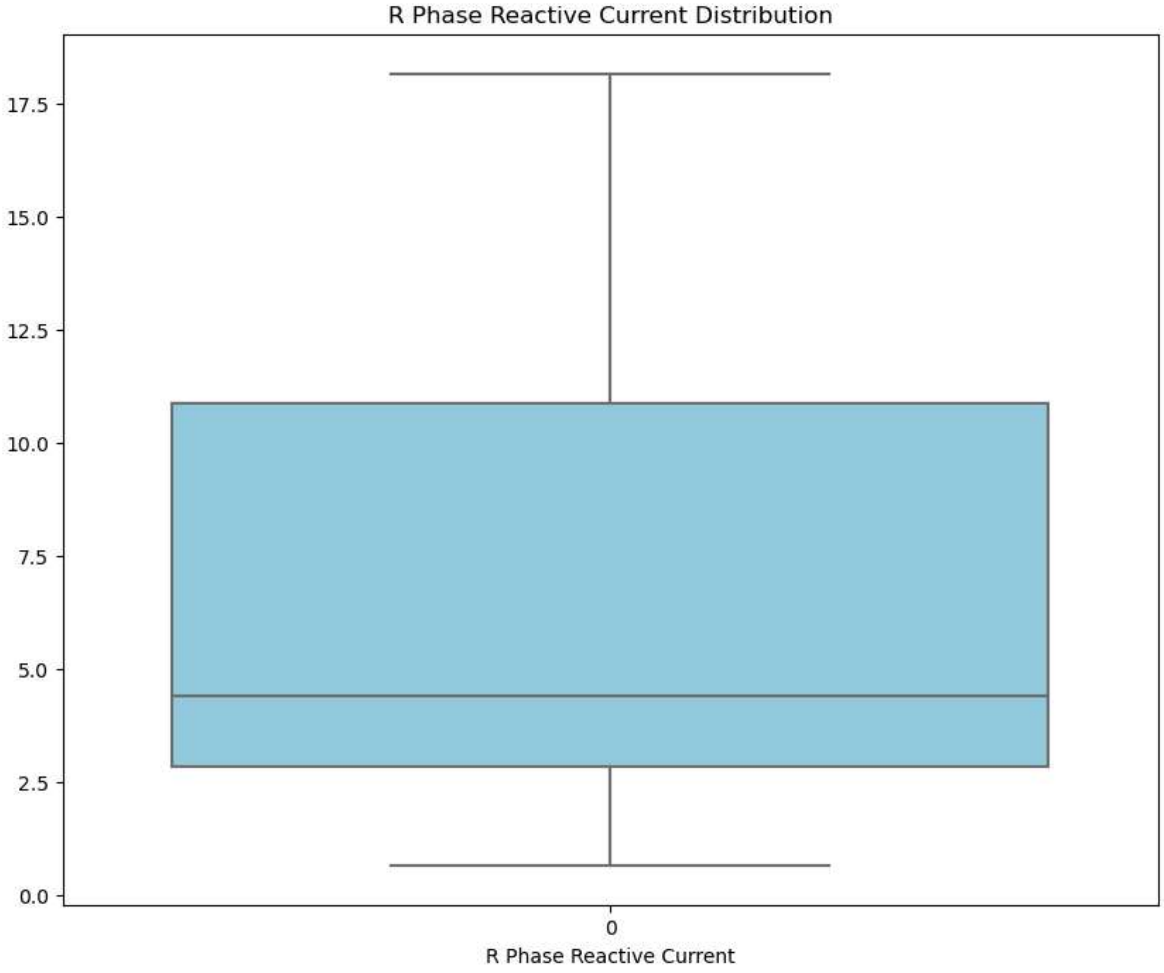

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



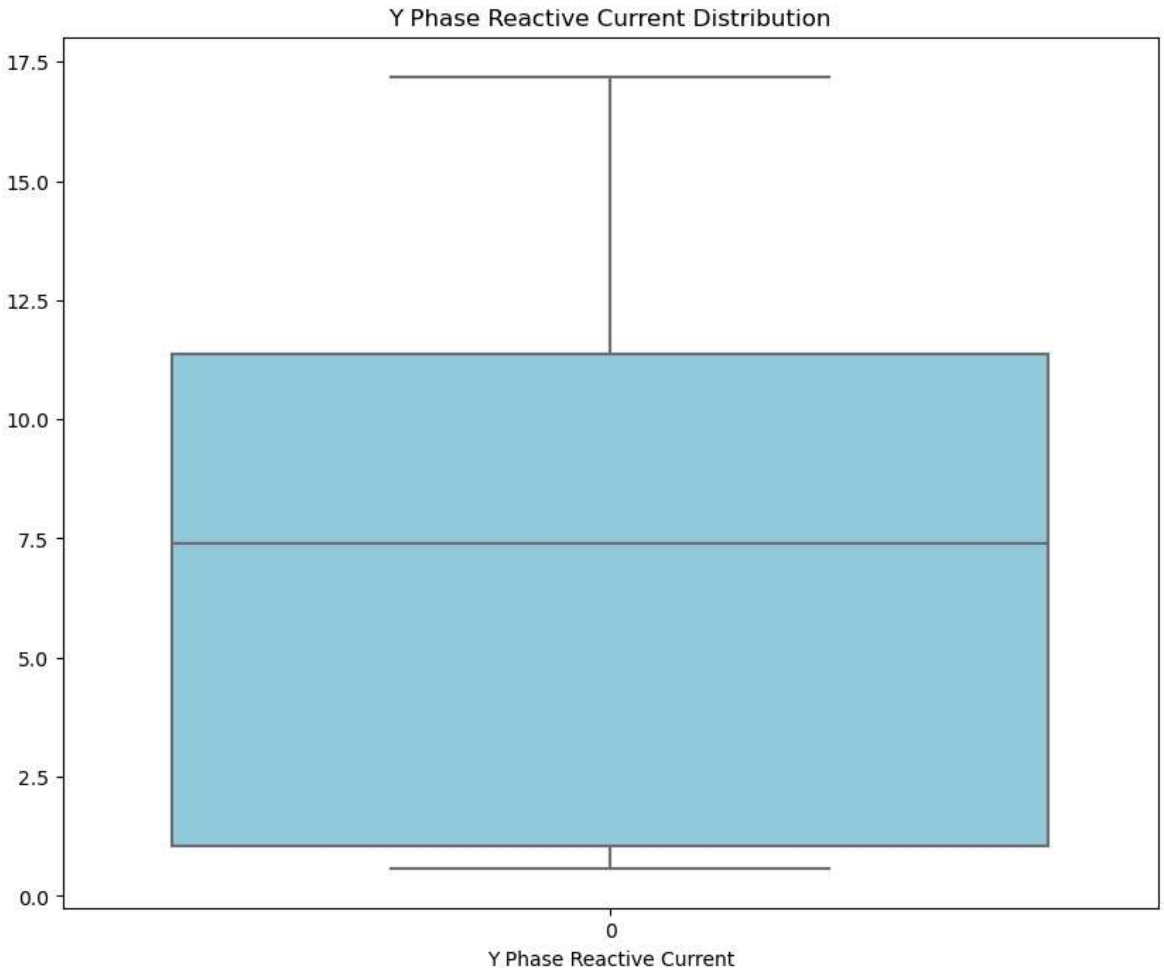

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



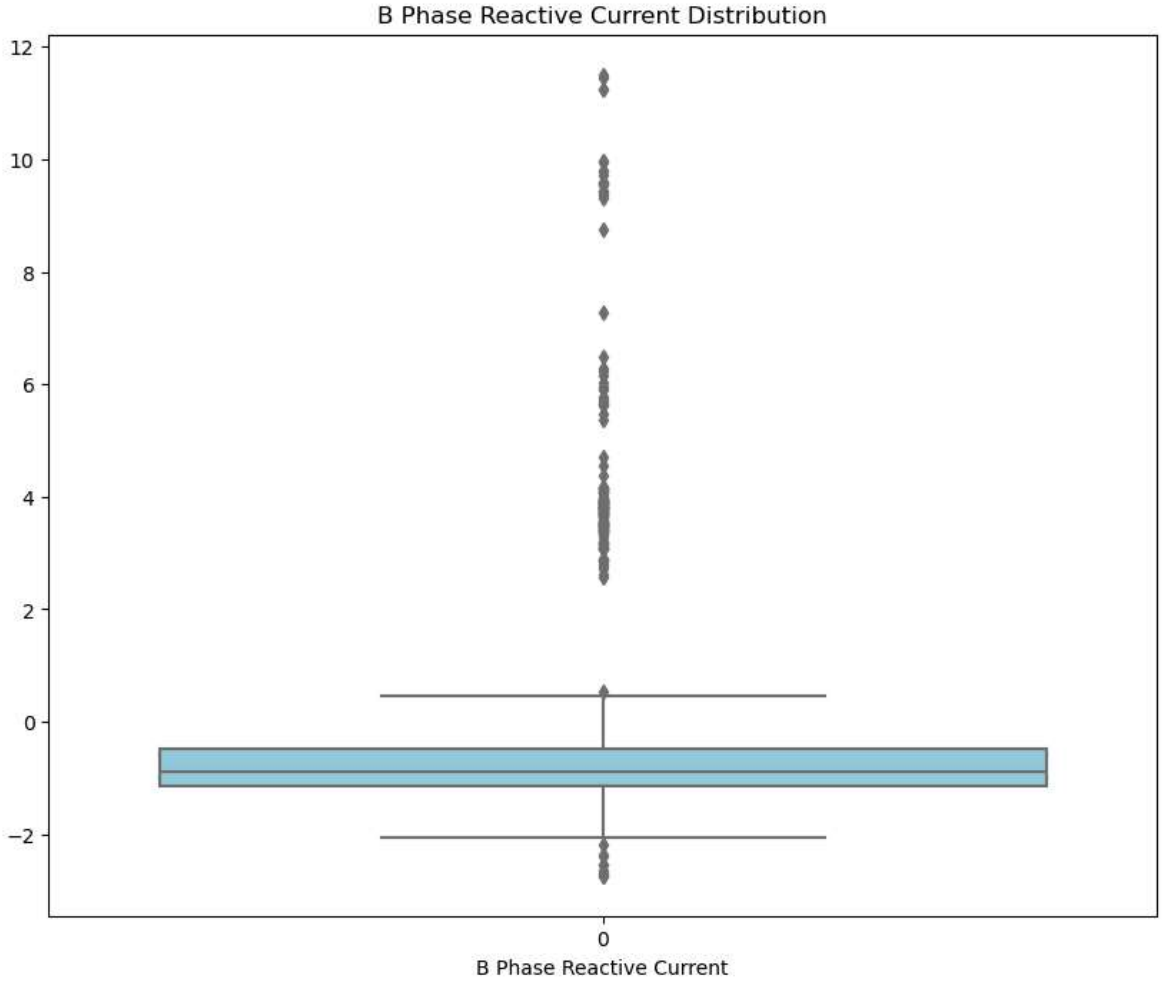

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



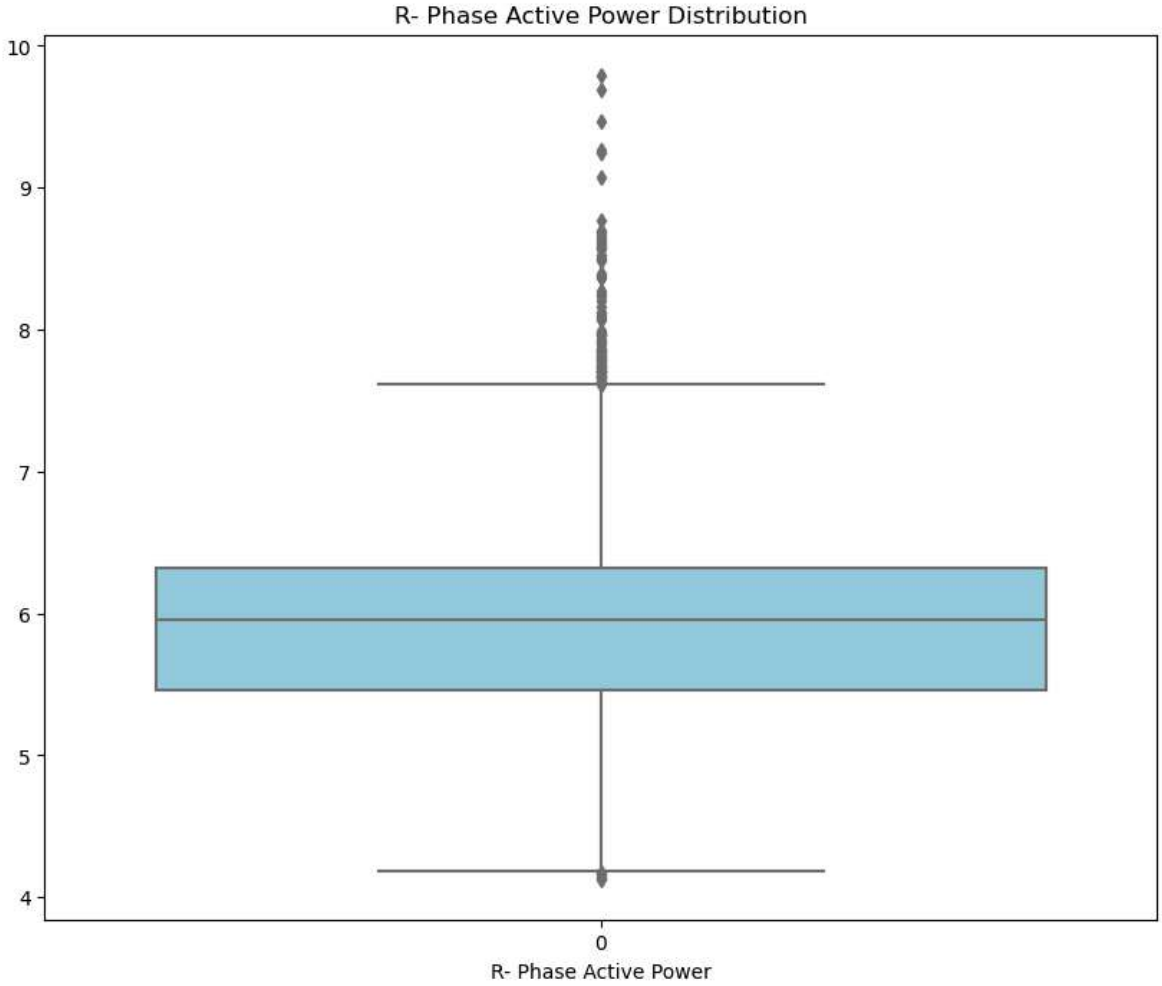

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



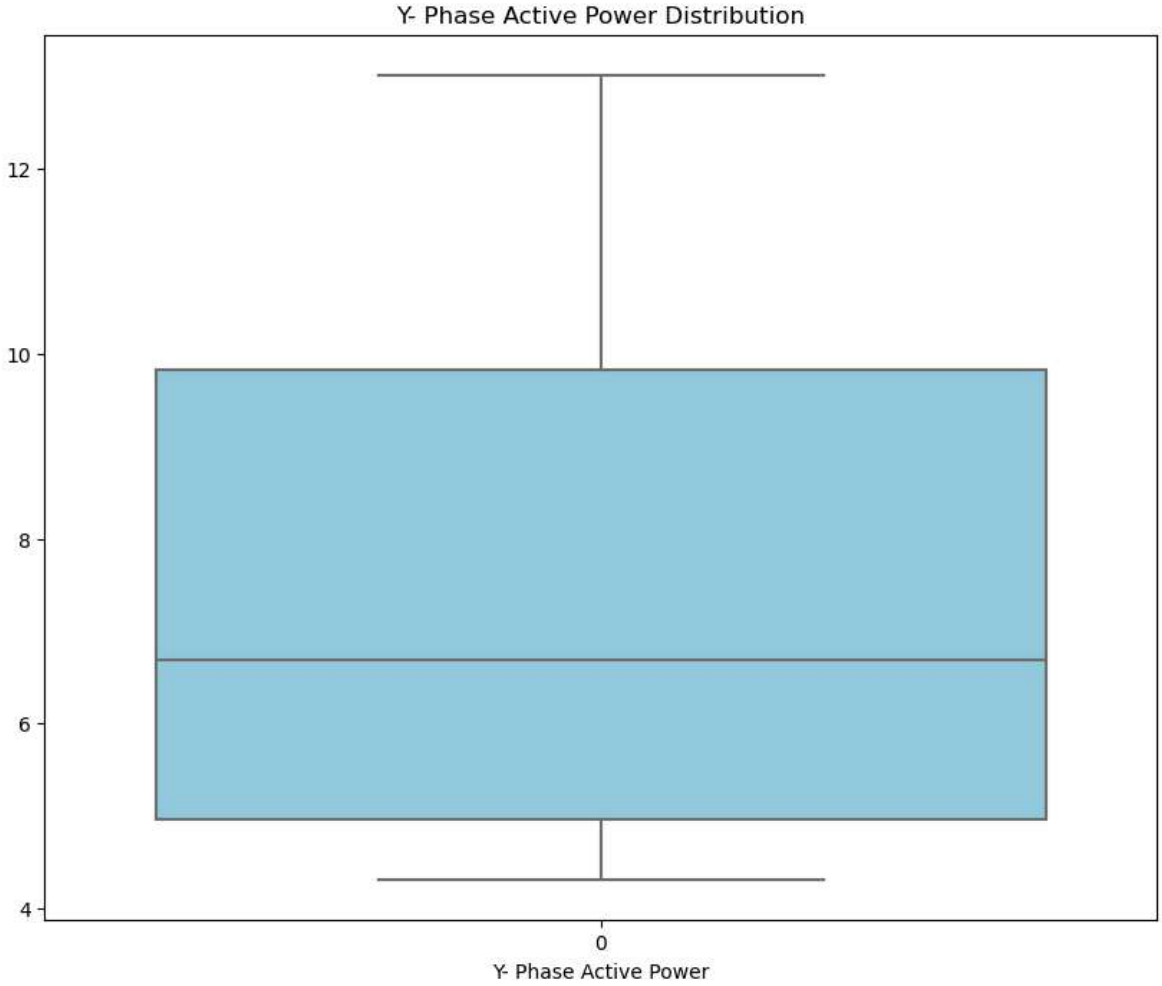

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



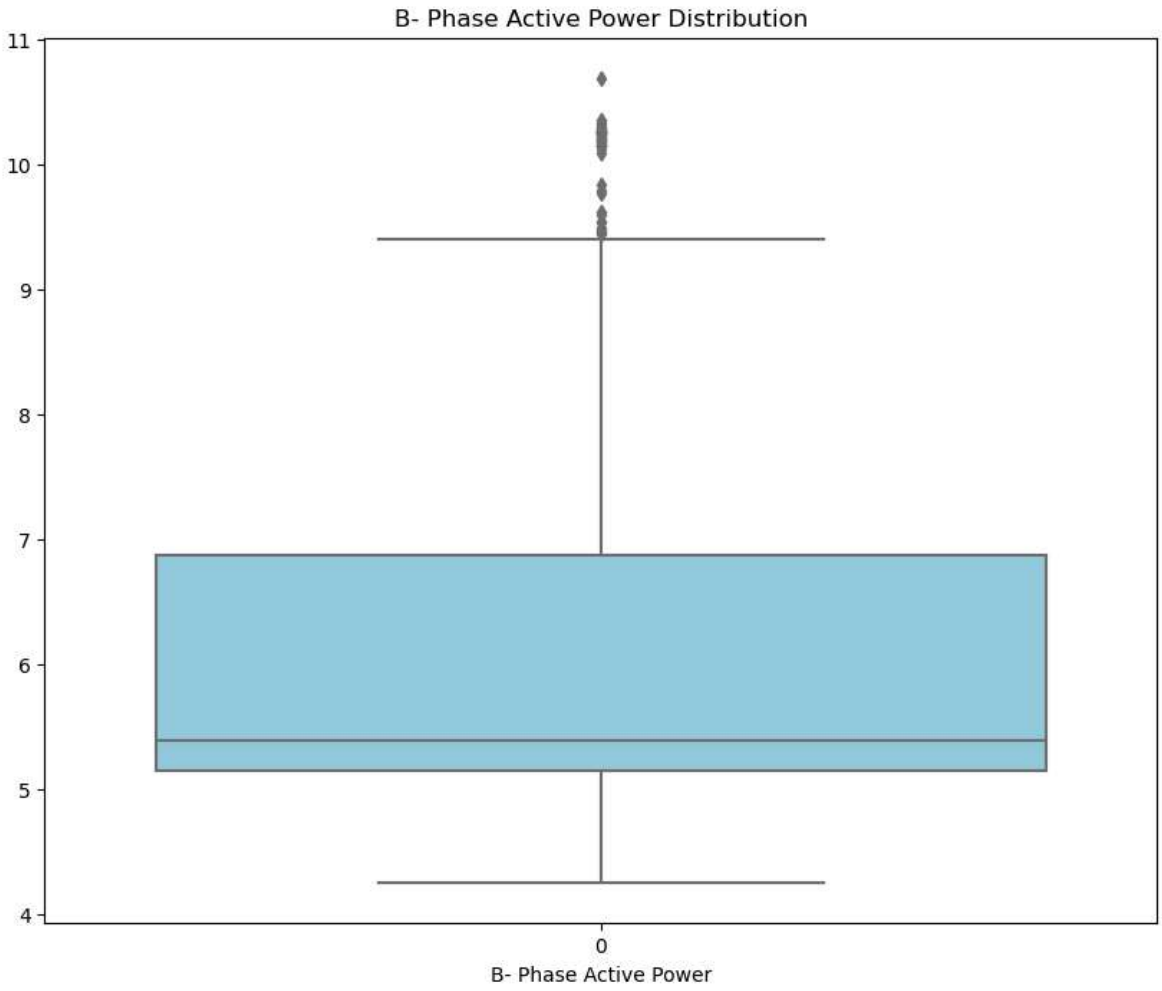

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



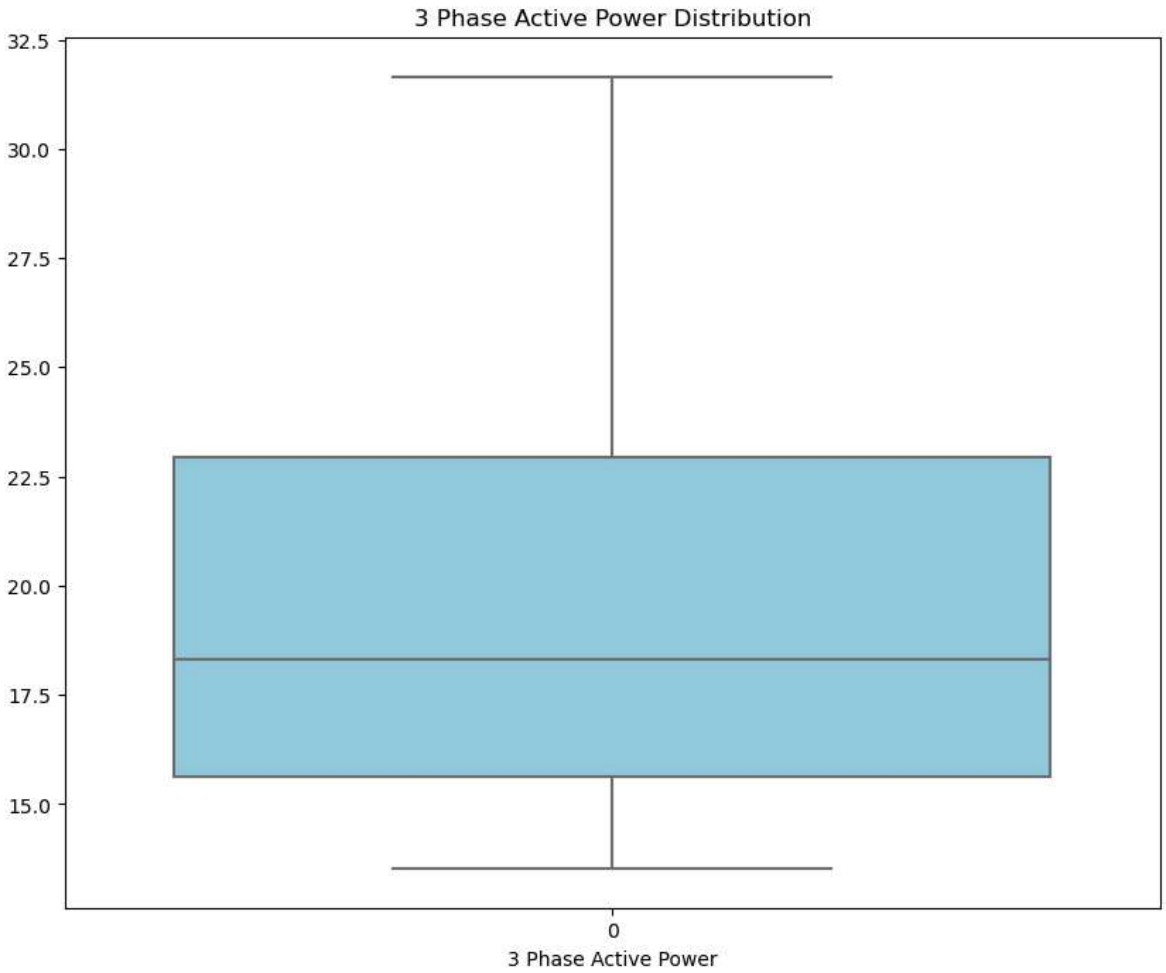

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



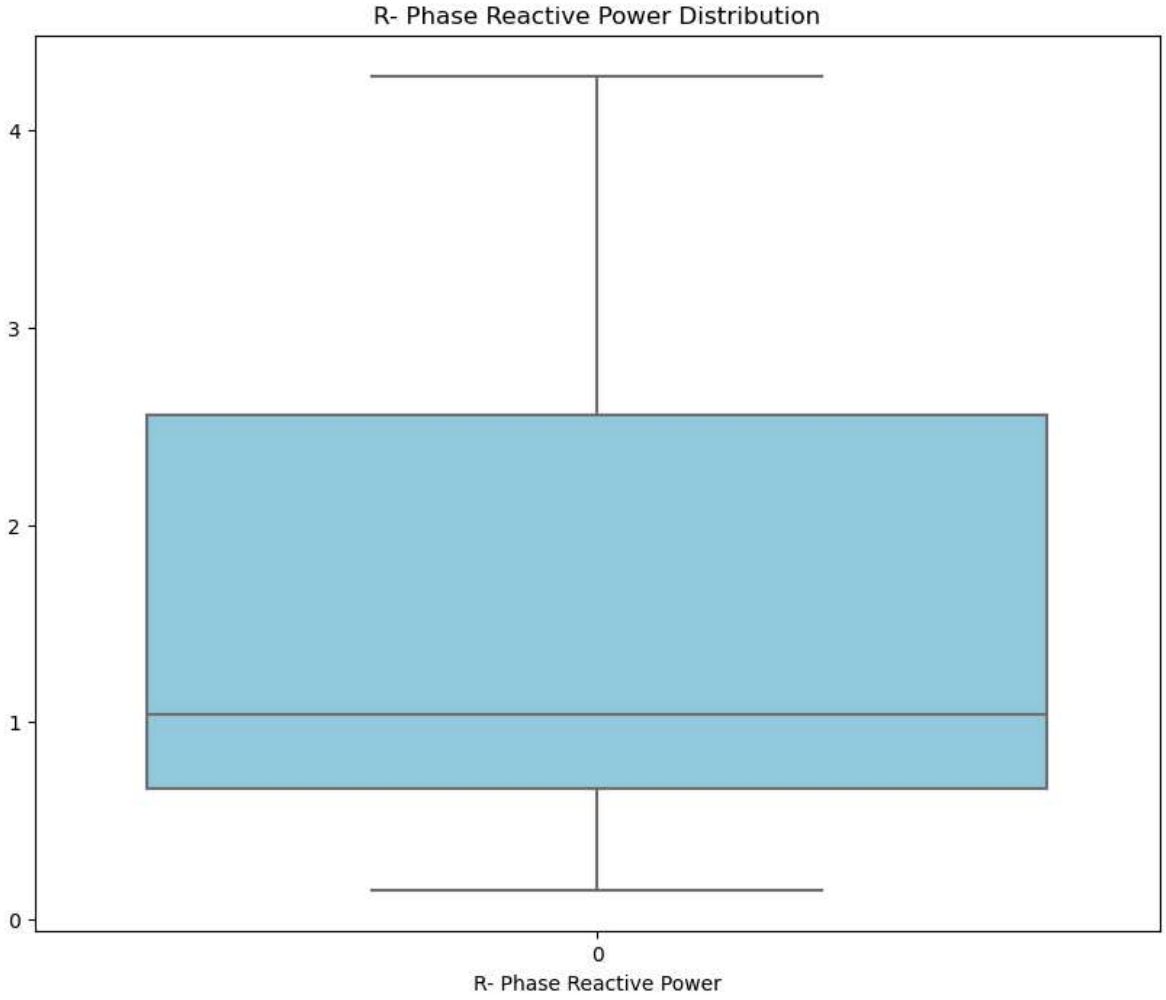

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



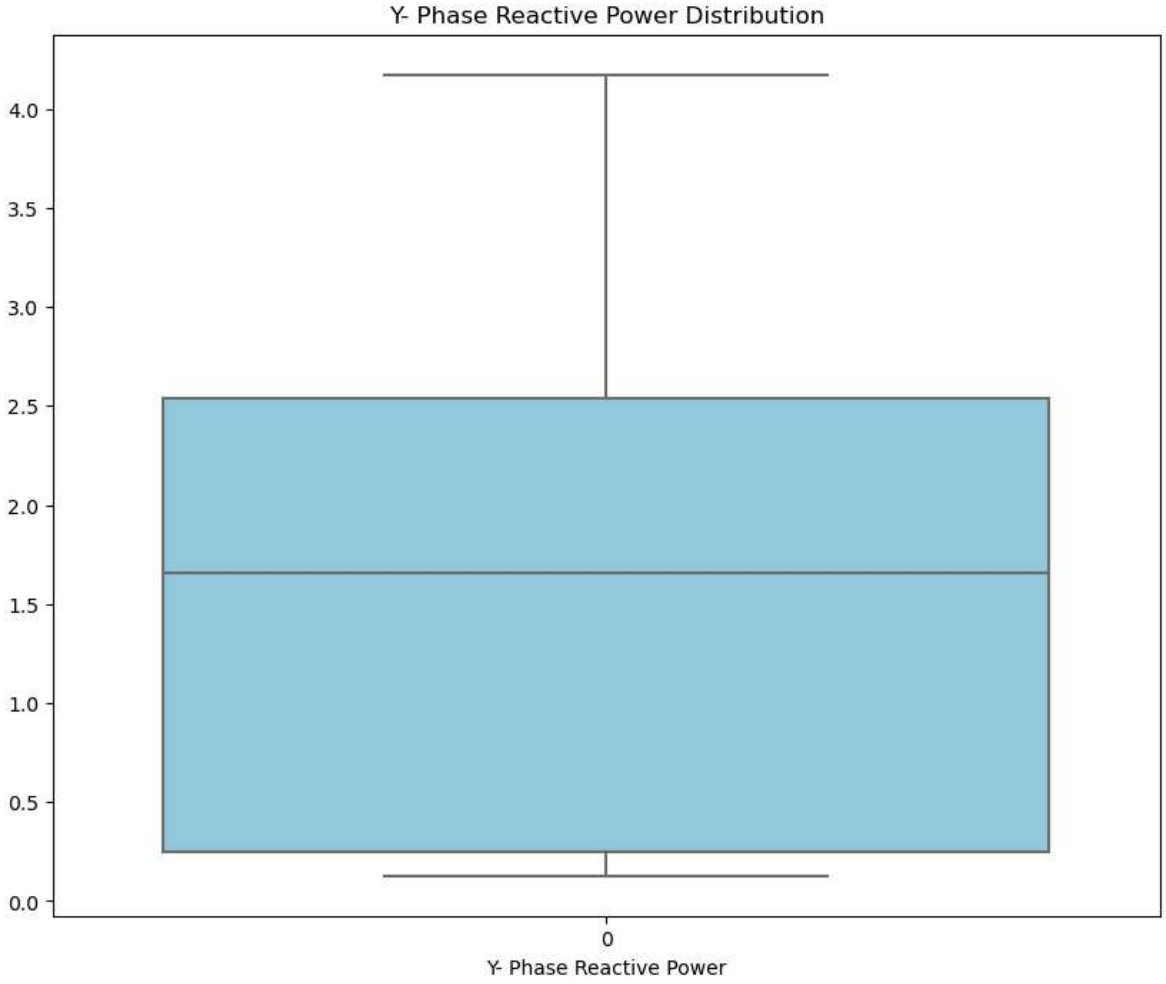

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



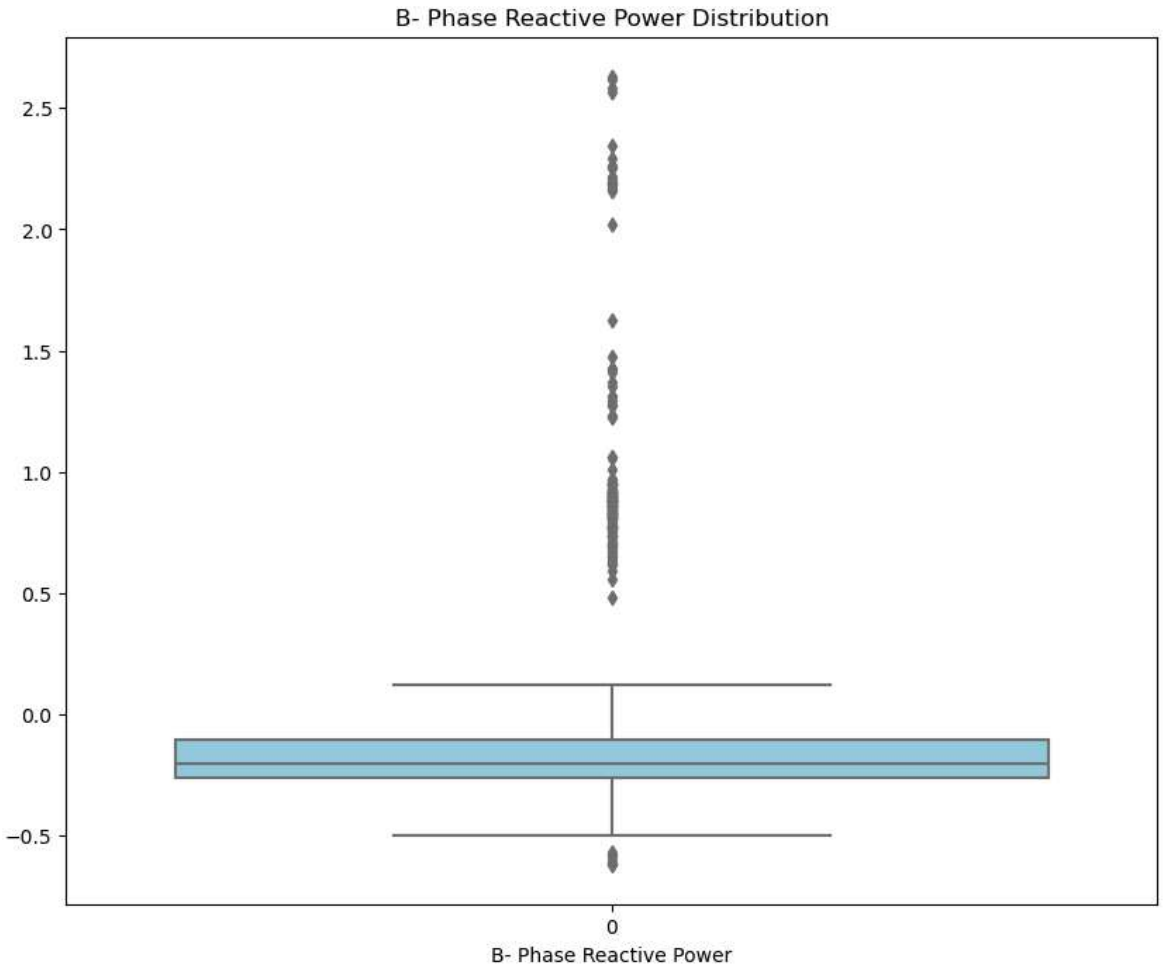

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



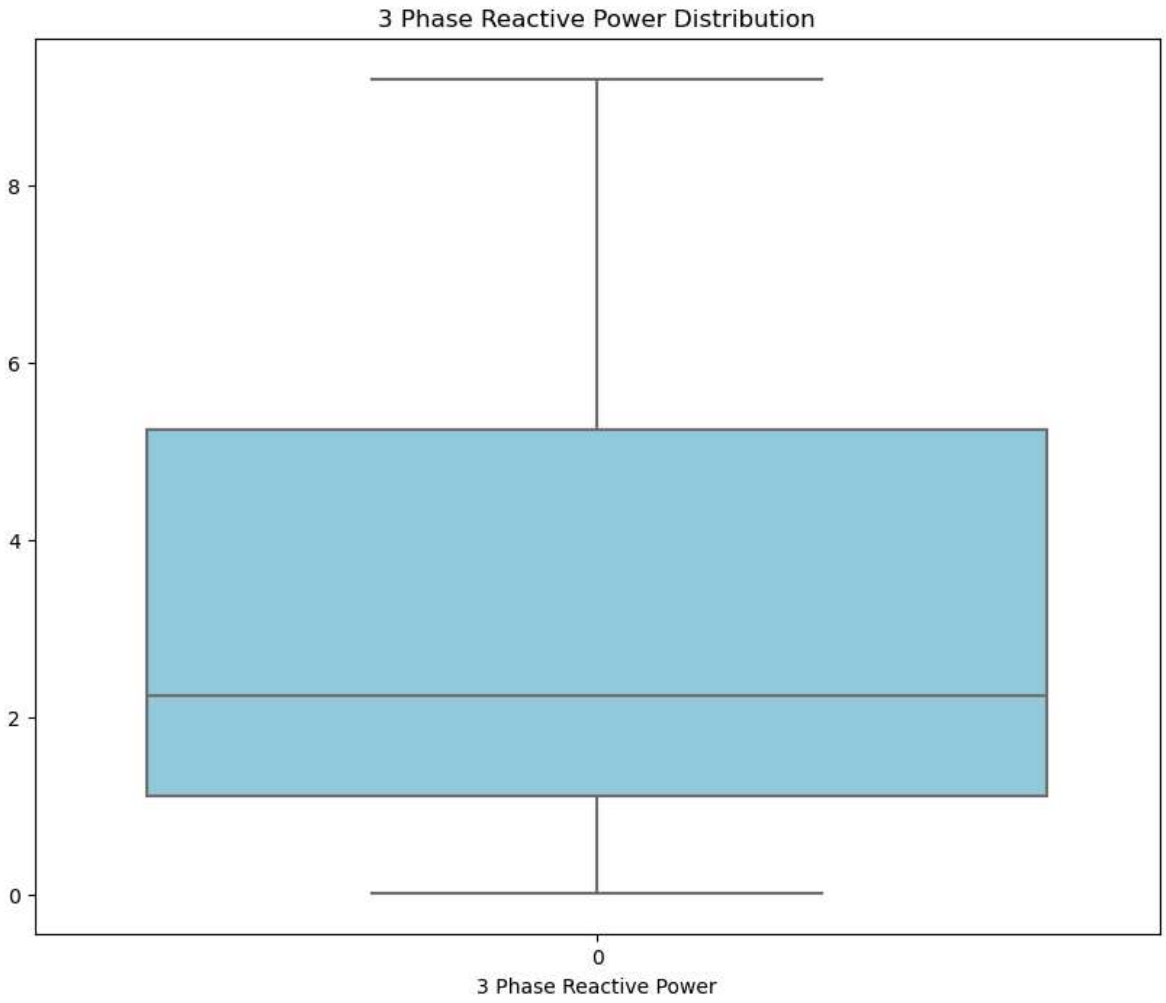

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



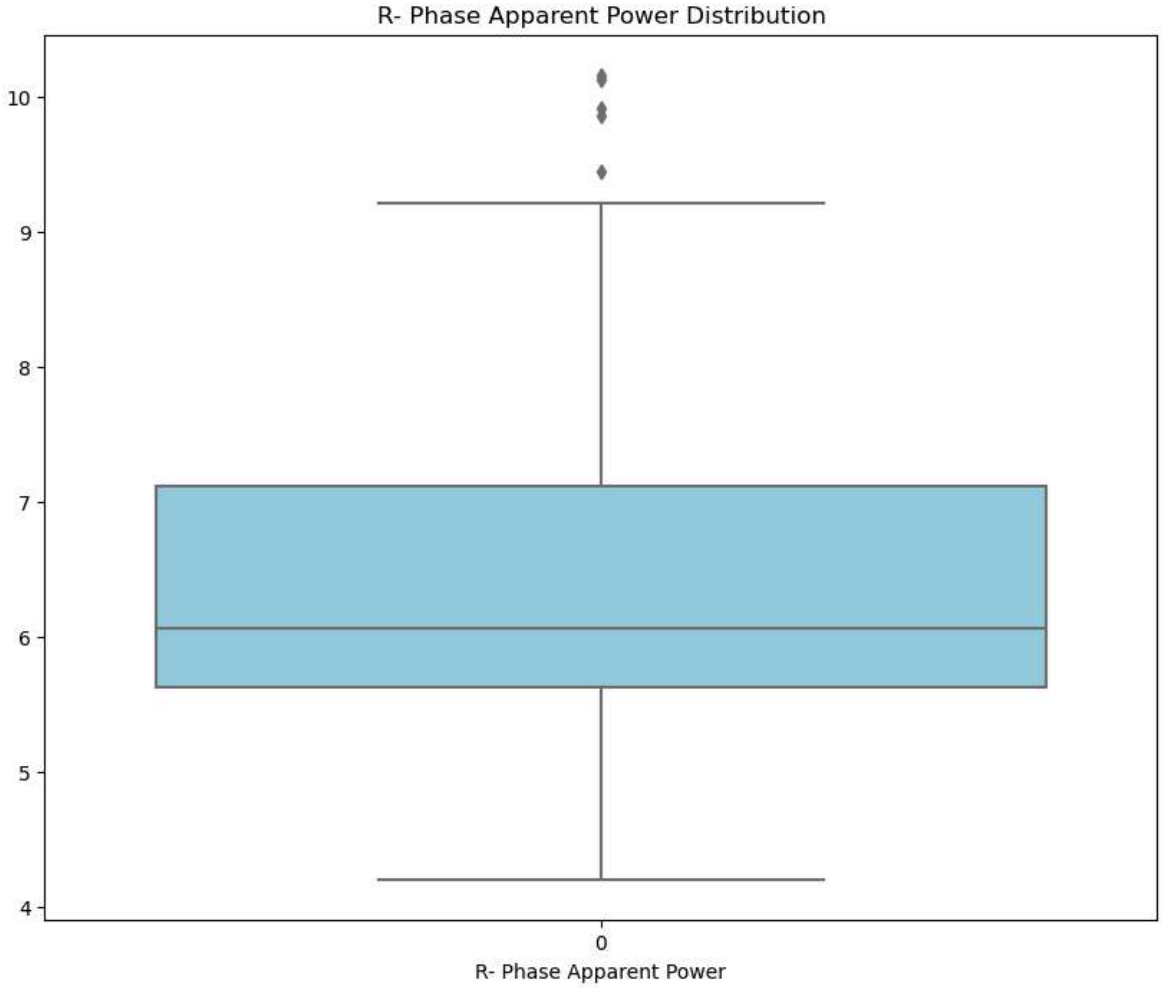

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



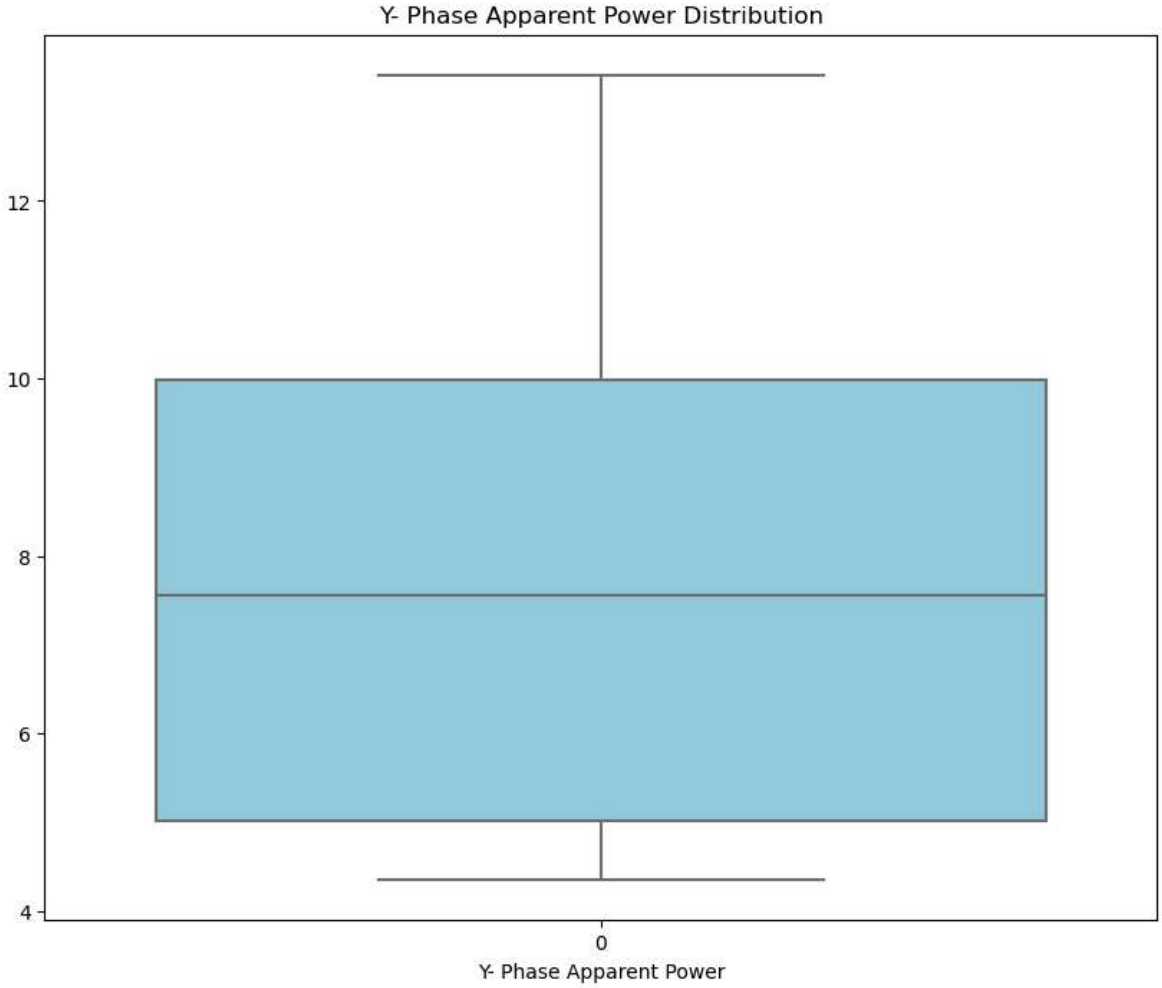

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



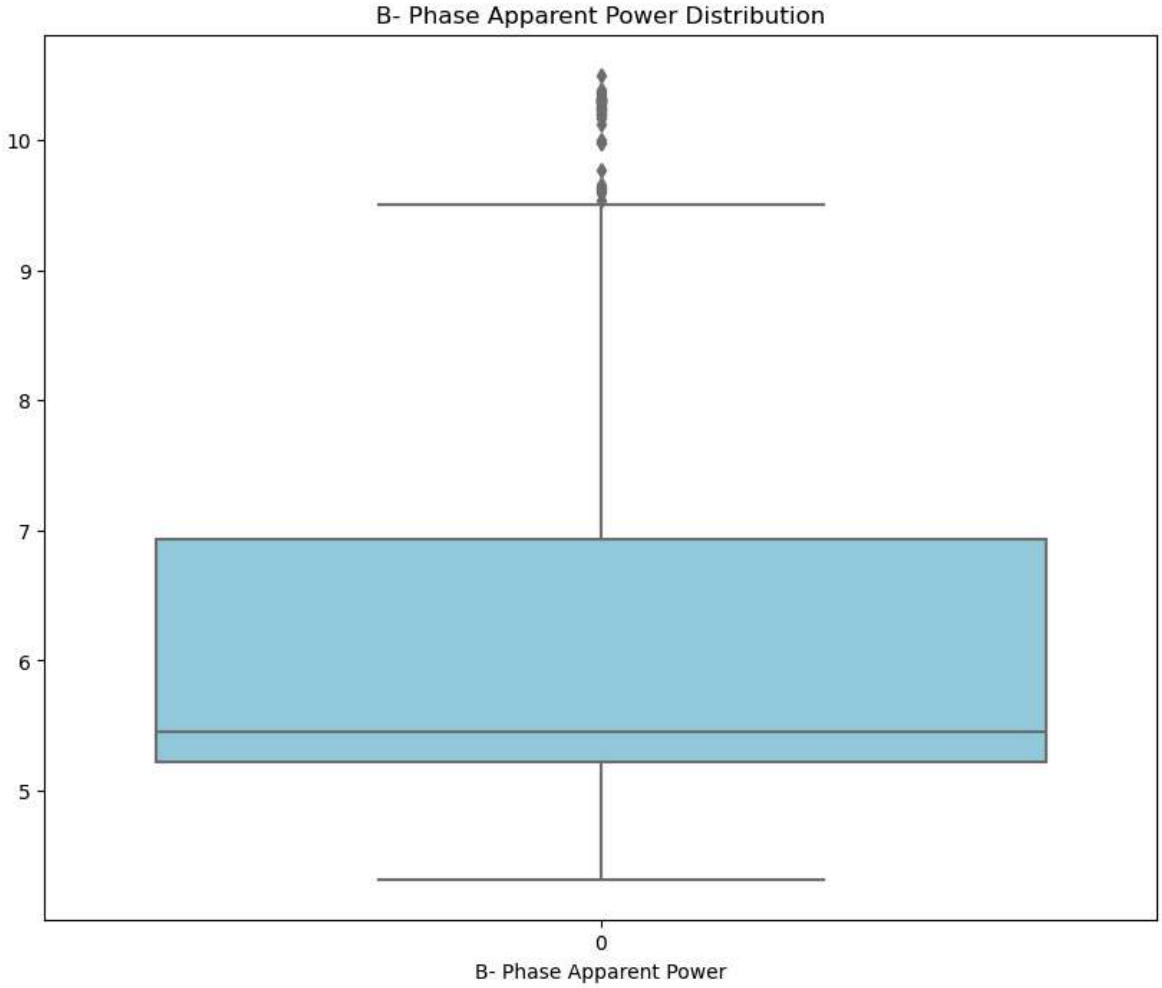

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



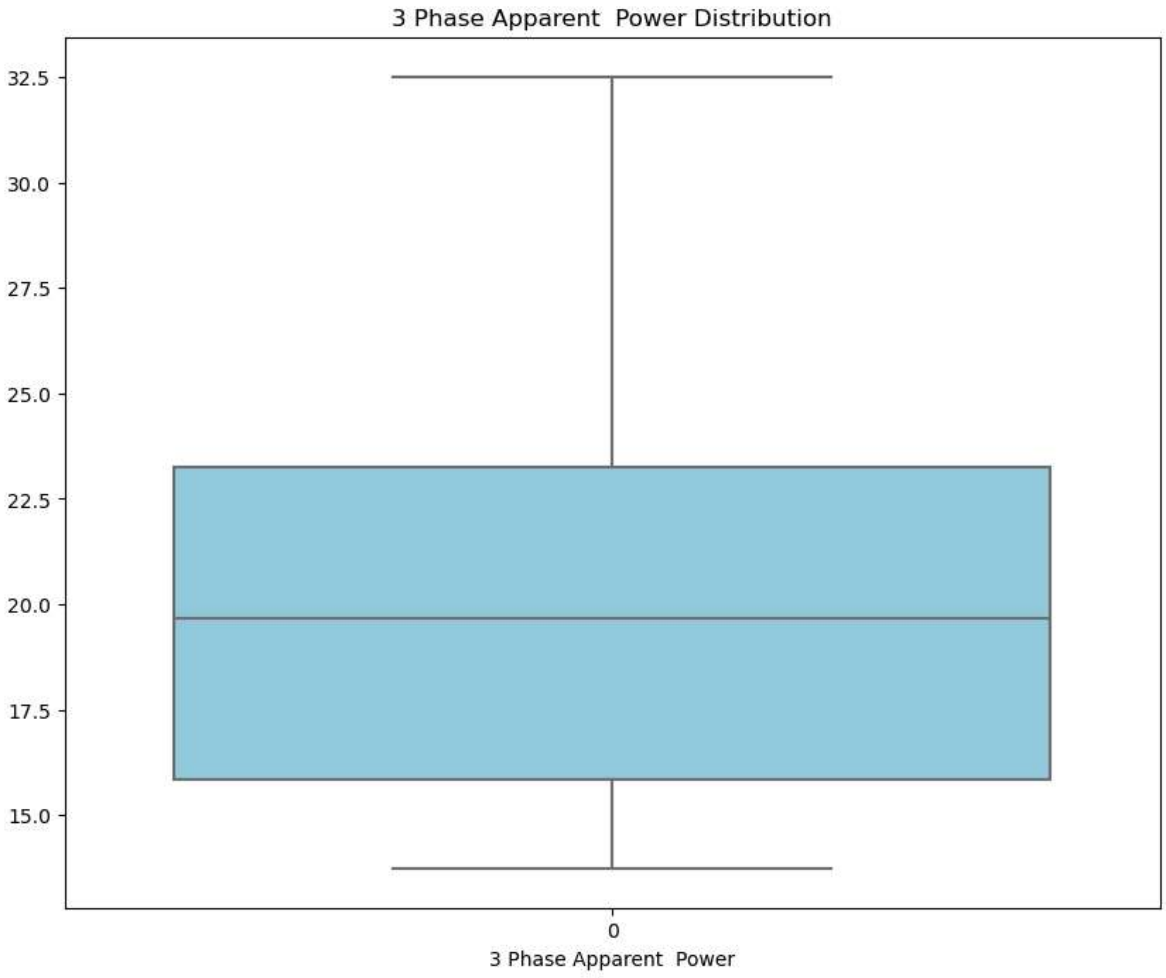

```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```




```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```




```
C:\Users\Tabish Ali Ansari\anaconda3\Lib\site-packages\seaborn\categorical.py:48
6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
a future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  if np.isscalar(data[0]):
```



ANALYSIS TERMINATED