The application will perform the following tasks:

https://github.com/zeto95/Mai_Elzayat_Task/tree/main

1. Load the CSV file using Pandas:

```
df = pd.read_csv(csv_file_path, sep =';')
print(df.head(10))
```

Output:

	timestamp	serial	grid_purchase	grid_feedin	direct_consumption	date
0	2023-05-22T15:40:00.000Z	1711356005	673	0	NaN	2023-05-22
1	2023-05-22T15:54:00.000Z	2105167400	NaN	36	NaN	2023-05-22
2	2023-05-22T16:14:00.000Z	1083091999	7012	0	NaN	2023-05-22
3	2023-05-22T04:57:00.000Z	970568993	0	2045	NaN	2023-05-22
4	2023-05-22T16:30:00.000Z	1559834999	27	0	NaN	2023-05-22
5	2023-05-22T18:21:00.000Z	1028817402	0	0	NaN	2023-05-22
6	2023-05-22T18:27:00.000Z	1559834999	27	0	NaN	2023-05-22
7	2023-05-22T18:23:00.000Z	1083091999	8115	0	NaN	2023-05-22
8	2023-05-22T16:26:00.000Z	1057045008	1378	0	NaN	2023-05-22
9	2023-05-22T18:43:00.000Z	1771861586	0	0	NaN	2023-05-22

2. Convert columns to their appropriate data types.

```
print (df.dtypes)
```

Output:

```
timestamp             object
serial                 int64
grid_purchase         object
grid_feedin           object
direct_consumption    object
date                  object
```

```
df['timestamp'] = pd.to_datetime(df['timestamp'], errors='coerce')
df['grid_purchase'] = pd.to_numeric(df['grid_purchase'], errors='coerce')
df['grid_feedin'] = pd.to_numeric(df['grid_feedin'], errors='coerce')
df['direct_consumption'] = pd.to_numeric(df['direct_consumption'], errors='coerce')
df['date'] = pd.to_datetime(df['date'], errors='coerce')
print (df.dtypes)
```

Output:

```
timestamp             datetime64[ns, UTC]
serial                              int64
grid_purchase                     float64
grid_feedin                       float64
direct_consumption                float64
date                        datetime64[ns]
```

3. Replace missing values with appropriate default values.

Before handling null values, these columns showed null values:

```
print(df.columns[df.isnull().any()])
```

```
Index(['grid_purchase', 'grid_feedin', 'direct_consumption'], dtype='object')
```

Null values could be handled in many ways depending on different aspects:

I have dealt with nulls here in two ways

- Filling the null values in columns `grid_purchase` and `grid_feedin` with the median of the whole column
- Dropping `'direct_consumption'` column as it was all NAN and from my stand it is not used in any other analysis

```
df['grid_purchase'].fillna(df['grid_purchase'].median(), inplace=True)
df['grid_feedin'].fillna(df['grid_purchase'].median(), inplace=True)
df = df.drop('direct_consumption', axis=1)
```

4. Remove duplicates.

There was 8 duplicated rows and now they are zero

```
df.drop_duplicates(inplace=True)
```

5. Calculate the total grid_purchase and grid_feedin over all batteries for each hour of the day.

I have used pandasql package to query the Dataframe

```
df['hour'] = df['timestamp'].dt.hour
query = '''
SELECT hour, SUM(grid_purchase) AS total_grid_purchase, SUM(grid_feedin) AS
total_grid_feedin
FROM df
GROUP BY hour
'''
result = sqldf(query, locals())
print(result)
```

	hour	total_grid_purchase	total_grid_feedin
0	0	2127462.0	50482.0
1	1	1422362.0	71651.0
2	2	1634695.0	118036.0
3	3	772147.0	343602.0
4	4	218597.0	572127.0
5	5	169590.0	127197.0
6	6	263935.0	11423.0
7	7	874886.0	5184.0
8	8	1602817.0	3203.0
9	9	1238718.0	3477.0
10	10	1199536.0	2686.0
11	11	1210960.0	2244.0
12	12	1312284.0	2123.0
13	13	1145995.0	1122.0
14	14	1032393.0	1670.0
15	15	1070127.0	1878.0
16	16	914855.0	1780.0
17	17	878679.0	108.0
18	18	918811.0	73.0
19	19	913793.0	364.0
20	20	1112068.0	62.0
21	21	1376858.0	695.0
22	22	2218811.0	262.0
23	23	3179886.0	20843.0

6. Add a boolean column to indicate the hour with the highest total_grid_feedin.

Adding `is_highest_grid_feed` column to flag the hour with the highest total grid feeding

```
max_hour = result['total_grid_feedin'].idxmax()
df['is_highest_grid_feed'] = df['hour'] == max_hour
```

7. Write the transformed data to an output file in the app directory named output.csv.

```
df.to_csv('app/output.csv', index=False)
```