```
import pandas as pdf
from pulp import LpProblem, LpMinimize, LpVariable, LpContinuous, lpSum, LpStatus,
value, LpBinary, GUROBI_CMD
import numpy as np
import plotly.express as px

df = pd.read_excel("model_1truck24h.xlsx", sheet_name="Single V2G Truck")
T=range(48)

#Create optimization model
model = LpProblem("V2G_Optimization", LpMinimize)

# Facility Fixed Parameters
kWMax = 1250 # Contracted max

# Truck Fixed Parameters
eta_ch = 0.975 # Charging efficiency
eta_dis = 0.975 # Discharging efficiency
P_ch_max = 250 # Maximum charging power in kW
P_dis_max = 250 # Maximum discharging power in kW
SOC_min = 55.2 # Depth of Discharge
SOC_max = 276 # Battery capacity
SOC_start = 276 # SOC at t=0

# Truck Variable Parameters
PRICE = df['PRICE'].fillna(0).to_dict()
R_t = df['R_t'].fillna(0).to_dict()
PV = df['PV'].fillna(0).to_dict()
BASE_LOAD = df['BASE_LOAD'].fillna(0).to_dict()
TRAFO = df['TRAFO'].fillna(0).to_dict()

threshold = np.percentile(list(TRAFO.values()), 80)  # Top 10% = "congested"
CONGESTED = {t: 1 if TRAFO[t] > threshold else 0 for t in T} # Binary flag voor congestie

# Truck Parameters
truck_ids = [1]

trucks = {}
for number in truck_ids:
      trucks[number] = {
         "PARKED": df[f"PARKED_{number}"].fillna(0).to_dict(),
         "DRAIN": df[f"DRAIN_{number}"].fillna(0).to_dict(),
         "V2G": df[f"V2G_{number}"].fillna(0).to_dict()
      }

# Facility decision variables
GRID_IM = {t: LpVariable(f"GRID_IM_{t}", lowBound=0, upBound = kWMax) for t in T}
GRID_EX = {t: LpVariable(f"GRID_EX_{t}", lowBound=0, upBound = kWMax) for t in T}
```

```
CURTAIL = {t: LpVariable(f"CURTAIL_{t}", lowBound=0) for t in T}
PEAK_RED = {t: LpVariable(f"PEAK_RED_{t}", lowBound=0) for t in T}
TRAFO_RELIEF = {t: LpVariable(f"TRAFO_RELIEF_{t}", lowBound=0) for t in T} #
Transformer relief

# Truck Decision Variables
for number in truck_ids:
  trucks[number]["CH_ACTIVE"] = LpVariable.dicts(f"CH_ACTIVE_{number}", T,
cat=LpBinary)
  trucks[number]["CH"] = LpVariable.dicts(f"CH_{number}", T, lowBound=0,
upBound=P_ch_max, cat=LpContinuous)
  trucks[number]["DIS_ACTIVE"] = LpVariable.dicts(f"DIS_ACTIVE_{number}", T,
cat=LpBinary)
  trucks[number]["DIS"] = LpVariable.dicts(f"DIS_{number}", T, lowBound=0,
upBound=P_dis_max, cat=LpContinuous)
  trucks[number]["SOC"] = LpVariable.dicts(f"SOC_{number}", T, lowBound=SOC_min,
upBound=SOC_max)

# Truck Constraints
for number in truck_ids:
  for t in T:
    # Charge/discharge only when parked
    model += trucks[number]["CH"][t] <= P_ch_max * trucks[number]["PARKED"][t]
    model += trucks[number]["DIS"][t] <= P_dis_max * trucks[number]["PARKED"][t]
    # Discharge only when V2G-enabled
    model += trucks[number]["DIS"][t] <= P_dis_max * trucks[number]["V2G"][t]
    # No simultaneous charging and discharging
    model += trucks[number]["CH_ACTIVE"][t] + trucks[number]["DIS_ACTIVE"][t] <= 1
    # CH/DIS only when binary flag is on
    model += trucks[number]["CH"][t] <= P_ch_max * trucks[number]["CH_ACTIVE"][t]
    model += trucks[number]["DIS"][t] <= P_dis_max * trucks[number]["DIS_ACTIVE"][t]
  model += trucks[number]["SOC"][T[-1]] >= SOC_start


# SOC Evolution
for number in truck_ids:
  for t in T:
    if t == 0:
      model += trucks[number]["SOC"][t] == SOC_start
    else:
      model += (
        trucks[number]["SOC"][t] ==
        trucks[number]["SOC"][t-1] +
        eta_ch * trucks[number]["CH"][t] -
        (1 / eta_dis) * trucks[number]["DIS"][t] -
        trucks[number]["DRAIN"][t]
      )
```

```
# Facility Constraints
for t in T:
  EV_DIS = lpSum(trucks[i]["DIS"][t] for i in truck_ids)
  EV_CHARGING = lpSum(trucks[i]["CH"][t] for i in truck_ids)

  model += ((GRID_IM[t] + PV[t] + EV_DIS) == (GRID_EX[t] + BASE_LOAD[t] +
EV_CHARGING + CURTAIL[t])) # Energy balance
  model += GRID_IM[t] <= kWMax  # Physical grid limit (hard stop)
  model += PEAK_RED[t] == kWMax - GRID_IM[t]
  model += PEAK_RED[t] >= 0


  if CONGESTED[t]:    # Reducing grid import compared to baseline during congested hours
     model += TRAFO_RELIEF[t] <= BASE_LOAD[t] - GRID_IM[t]  # load reduction
  else:
     model += TRAFO_RELIEF[t] == 0  # no credit outside congestion

RELIEF_REWARD = {t: 0.2 if CONGESTED[t] else 0 for t in T}  # Spreekt voor zich

# Objective function
#model += lpSum(
#    PRICE[t] * GRID_IM[t] - R_t[t] * PEAK_RED[t]
#    for t in T
#)

model += lpSum(
  PRICE[t] * GRID_IM[t]
  - R_t[t] * PEAK_RED[t]
  - RELIEF_REWARD[t] * TRAFO_RELIEF[t]
  for t in T
)


# Solve the optimization model
model.solve(GUROBI_CMD())

print("Status:", LpStatus[model.status]) #Print solution status
print("Total cost:", value(model.objective)) # Print the total cost


import matplotlib.pyplot as plt
from pulp import value

# Time range
t_range = list(T)  # e.g. range(48)

# Truck to visualize
truck_id = 1
```

```
# Extract values
grid_import = [value(GRID_IM[t]) for t in t_range]
soc_values = [value(trucks[truck_id]["SOC"][t]) for t in t_range]
pv_values = [PV[t] for t in t_range]

# Plot results
plt.figure(figsize=(12, 6))

plt.plot(t_range, grid_import, label="Grid Import [kWh]")
plt.plot(t_range, soc_values, label=f"Truck {truck_id} SOC [kWh]")
plt.plot(t_range, pv_values, label="PV Production [kWh]", linestyle="--")

plt.xlabel("Time step (hour)")
plt.ylabel("Energy [kWh]")
plt.title("Optimized Energy Flow")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```