strategy

September 4, 2022

1 Imports

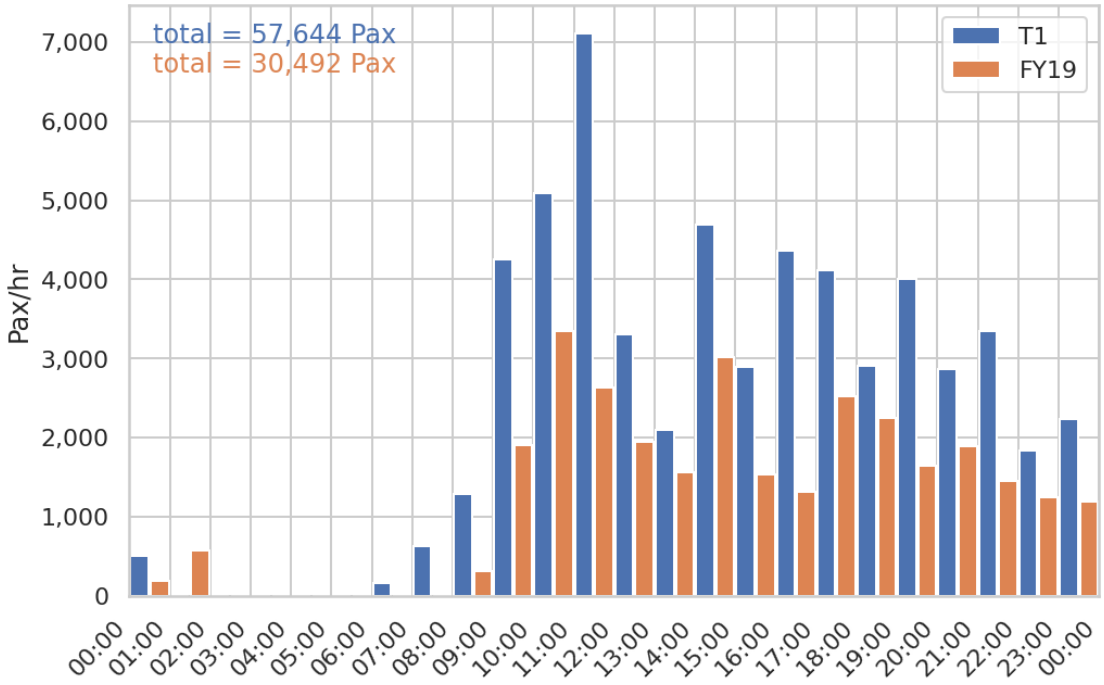
[ ]:
```
%load_ext autoreload
%autoreload 2

from copy import deepcopy

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

from src.simfunc.simulation import Simulation  # class for simulation
from src.utils.optimizer import custmin # custom minimizer
from src.utils.simparam import SimParam  # class for simulation parameters
from src.utils.utils import day_graph, minutes_to_hms

%matplotlib inline
%config InlineBackend.figure_format='retina'
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

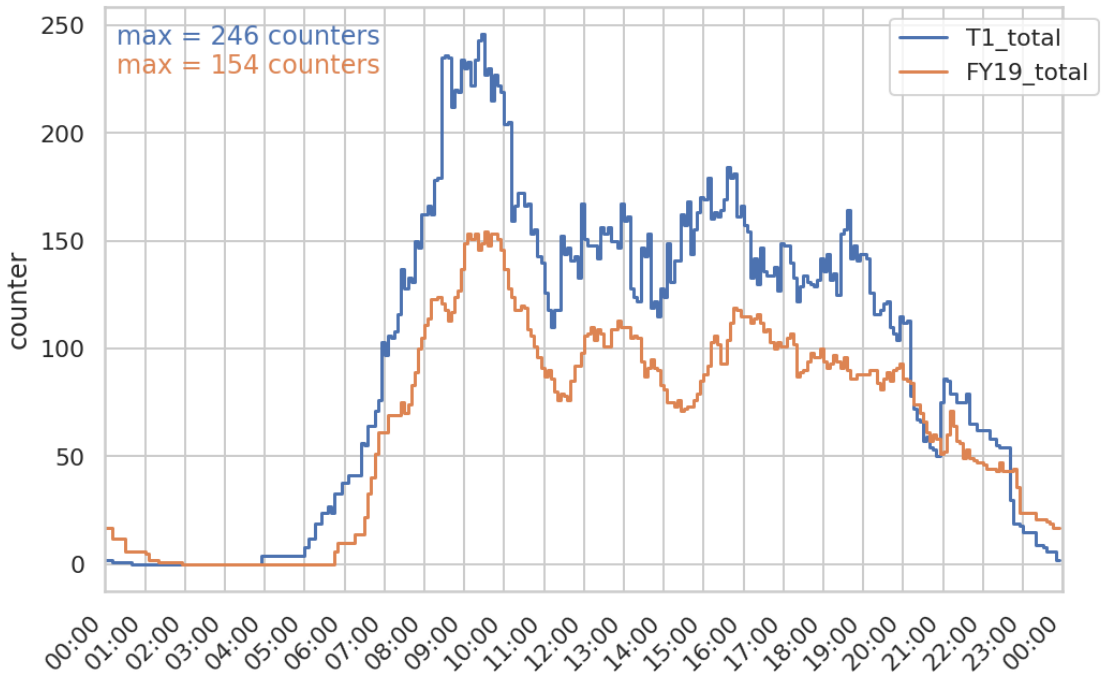
2 STD FY2019 vs T1 renovation design

[ ]:
```
# get FY2019 data
data = pd.read_excel(
    io="/home/antoine/projects/pax-flow-sim/data/raw/FY2019_FY2025 merge.xlsx",
    header=0,
    sheet_name="IntlP_FY19-FY25",
)

# 7kPax/hour std schedule
simparamT1 = SimParam()
path_to_6k_sched = (
    r"/home/antoine/projects/pax-flow-sim/data/processed/Schedule (30th␣
  ↪terminal"
```

```
    r" peak, 6000 pax)_PROCESSED.xlsx"
)
simparamT1.schedule_from_path(path_to_6k_sched)
simparamT1.schedule_cleanup().schedule_filter().assign_check_in()
simparamT1.show_up_from_file().assign_flight_show_up_category_default().
 ↪assign_show_up()

# format the file from aero
mask2019 = (data["FY"] == "FY2019") & (data["Day Of Week"] == "Saturday")
data2019 = data[mask2019].reset_index().copy()
data2019["Sector"] = data2019["Int/Dom"]
data2019["Flight Date"] = pd.to_datetime("2019-10-13")
data2019["Scheduled Time"] = data2019["5min Interval"].apply(
    lambda x: f"{x // 100}:{x % 100}:00",
)
data2019["Category(P/C/O)"] = data2019["Category(P/C/O)"].apply(lambda x: x[0])

# create simparam2019
simparam2019 = SimParam()
simparam2019.schedule_from_df(data2019).schedule_cleanup().schedule_filter(
    date_str="2019-10-13"
).assign_check_in()
simparam2019.show_up_from_file().assign_flight_show_up_category_default().
 ↪assign_show_up()
# define plot name
simparamT1.plot_name = "T1"
simparam2019.plot_name = "FY19"

# plot both for comparison
simparamT1.plot_std(compare_with=simparam2019)
```



[ ]:
```
simparamT1.plot_counters(compare_with=simparam2019)
```



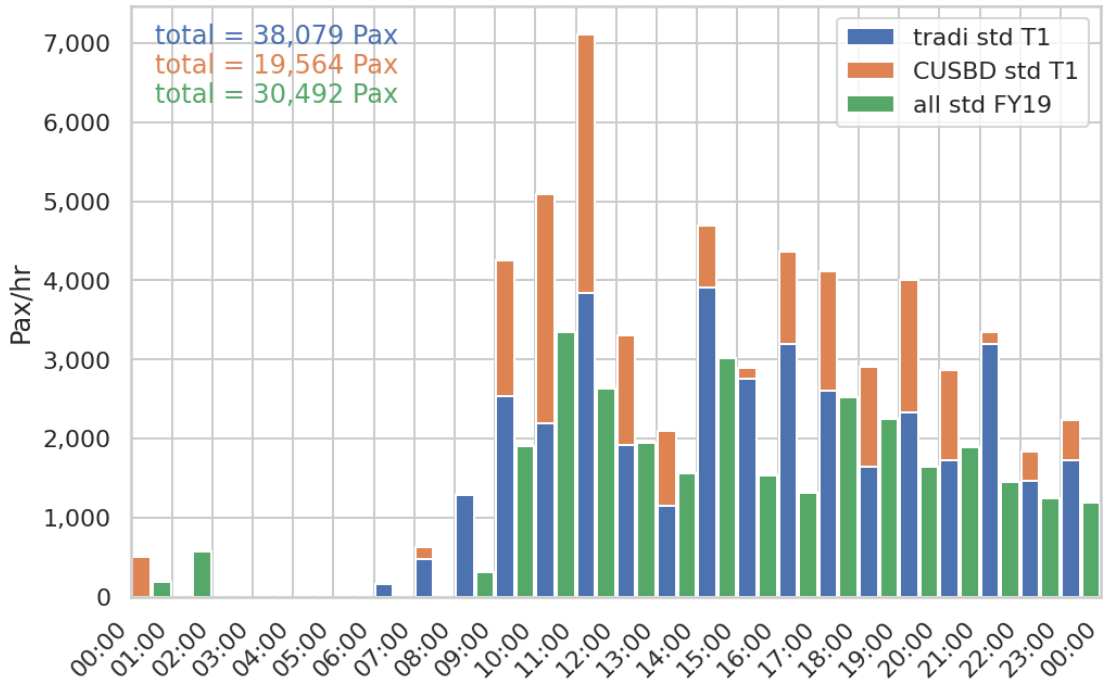
[ ]:
```
# airlines from survey
selected_ALs = [
    [  # Pattern 1: All Green for CUSBD & CUCkIn
        "AAR",
        "ACA",
        "CCA",
        "CPA",
        "CQH",
        "CRK",
        "CSC",
        "CSZ",
        "CXA",
        "EVA",
        "GCR",
        "JJA",
        "JJP",
        "JST",
        "SJX",
        "TAX",
        "XAX",
    ],
    [  # Pattern 2: Not Negative for CUSBD & CUCkIn
        "AAR",
        "ACA",
        "AMU",
        "ANA",
        "ANZ",
        "BAW",
        "CAL",
        "CBJ",
        "CCA",
        "CEB",
        "CES",
        "CPA",
        "CQH",
        "CRK",
        "CSC",
        "CSH",
        "CSN",
        "CSZ",
        "CXA",
        "DKH",
        "DLH",
        "EVA",
        "FIN",
        "GCR",
        "GIA",
```

```
        "HKE",
        "JJA",
        "JJP",
        "JNA",
        "JSA",
        "JST",
        "OKA",
        "SJX",
        "SWR",
        "TAX",
        "THA",
        "TLM",
        "UAE",
        "XAX",
    ],
    [  # Pattern 3: Agree on CUSBD
        "AAR",
        "ABL",
        "ACA",
        "AMU",
        "ANA",
        "ANZ",
        "CAL",
        "CEB",
        "CES",
        "CPA",
        "CQH",
        "CSC",
        "CSH",
        "CXA",
        "DKH",
        "EVA",
        "FIN",
        "GCR",
        "HKE",
        "JAL",
        "JJA",
        "JJP",
        "JNA",
        "JSA",
        "JST",
        "KAL",
        "MAS",
        "OKA",
        "SIA",
        "SJX",
        "TAX",
```

```
        "TWB",
        "UAE",
        "XAX",
    ],
    [  # Pattern 4: Not disagree on CUSBD
        "AAR",
        "ABL",
        "ACA",
        "AFR",
        "AMU",
        "ANA",
        "ANZ",
        "CAL",
        "CCA",
        "CDC",
        "CEB",
        "CES",
        "CPA",
        "CQH",
        "CRK",
        "CSC",
        "CSH",
        "CSZ",
        "CXA",
        "DKH",
        "EVA",
        "FIN",
        "GCR",
        "GIA",
        "HKE",
        "JAL",
        "JJA",
        "JJP",
        "JNA",
        "JSA",
        "JST",
        "KAL",
        "KLM",
        "MAS",
        "OKA",
        "SIA",
        "SJX",
        "TAX",
        "TWB",
        "UAE",
        "XAX",
    ],
```

```
]
```

[ ]:
```
# Most likely case
airlines_CU = selected_ALs[2]
simparamT1.df_Pax["pax_type"] = simparamT1.df_Pax["Airline"].apply(
    lambda x: "CUSBD" if x in airlines_CU else "tradi"
)
simparamT1.schedule["pax_type"] = simparamT1.schedule["Airline Code"].apply(
    lambda x: "CUSBD" if x in airlines_CU else "tradi"
)
simparamT1.plot_std(
    by_col=True, col_name="pax_type", compare_with=simparam2019,␣
 ↪reverse_cat=True
)
```



[ ]:
```
# simulation of waiting times
simparamT1.dct_resource = {
    "kiosk": 92,
    "CUSBD": 32,
}

simparamT1.dct_processes = {
    "kiosk": 70,
    "CUSBD": 60,
```

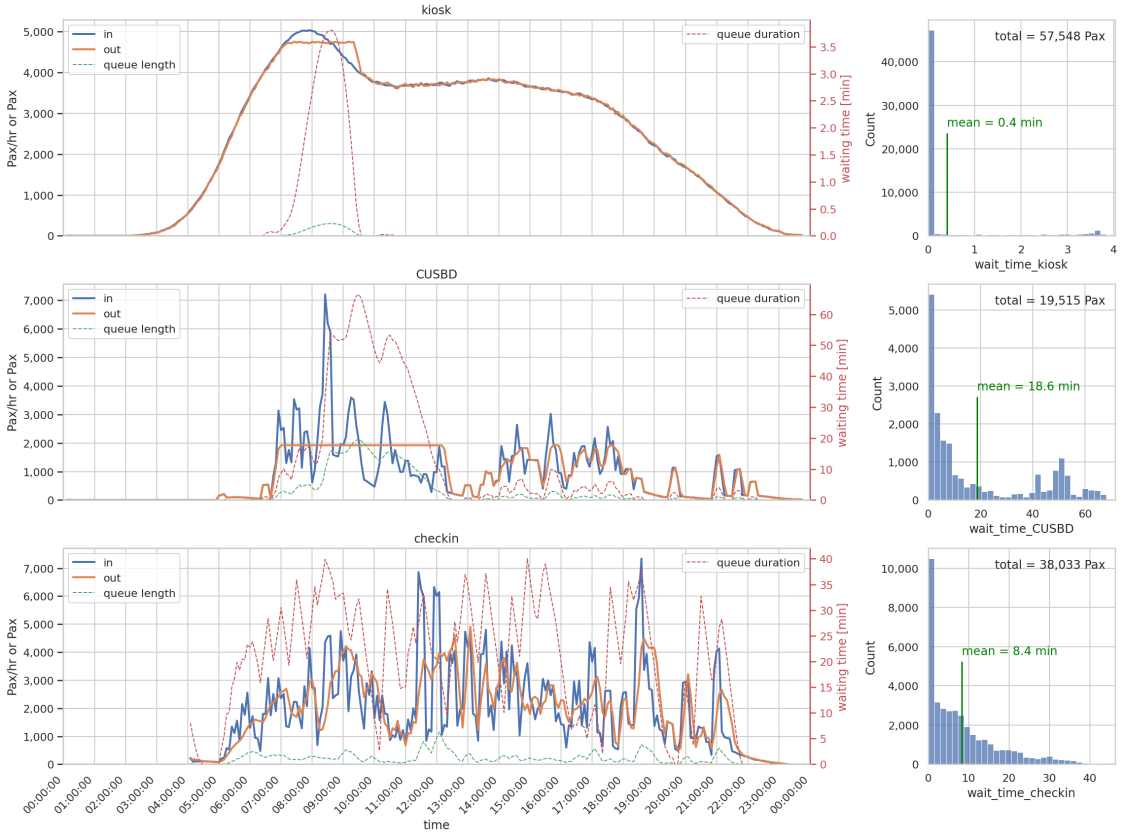
```
    "checkin": 80,
}

simparamT1.dct_process_sequence = {
    "tradi": [
        ["kiosk", "wait_opening", "checkin"],
        {"hour_to_std": 2.5},
    ],
    "CUSBD": [
        ["kiosk", "wait_opening", "CUSBD"],
        {"hour_to_std": 2.5},
    ],
}
```

[ ]:
```
simulationT1_pat3 = Simulation(simparamT1)
simulationT1_pat3.generate_checkin().generate_pax().run()
simulationT1_pat3.format_df_result()
```

```
Simulation running…: 100%|      | 1439/1439 [12:19<00:00,  1.95it/s]
```

[ ]:
```
<src.simfunc.simulation.Simulation at 0x7fb3af4c6ee0>
```

[ ]:
```
simulationT1_pat3.plot_result()
```



[ ]:
```
# simulation of waiting times
simparam2019.dct_resource = {
    "kiosk": 92,
    "CUSBD": 32,
}

simparam2019.dct_processes = {
    "kiosk": 70,
    "checkin": 80,
}

simparam2019.dct_process_sequence = {
    "tradi": [
        ["kiosk", "wait_opening", "checkin"],
        {"hour_to_std": 2.5},
    ],
}
```

[ ]:
```
simparam2019.df_Pax["pax_type"] = "tradi"
simulation2019 = Simulation(simparam2019)
```

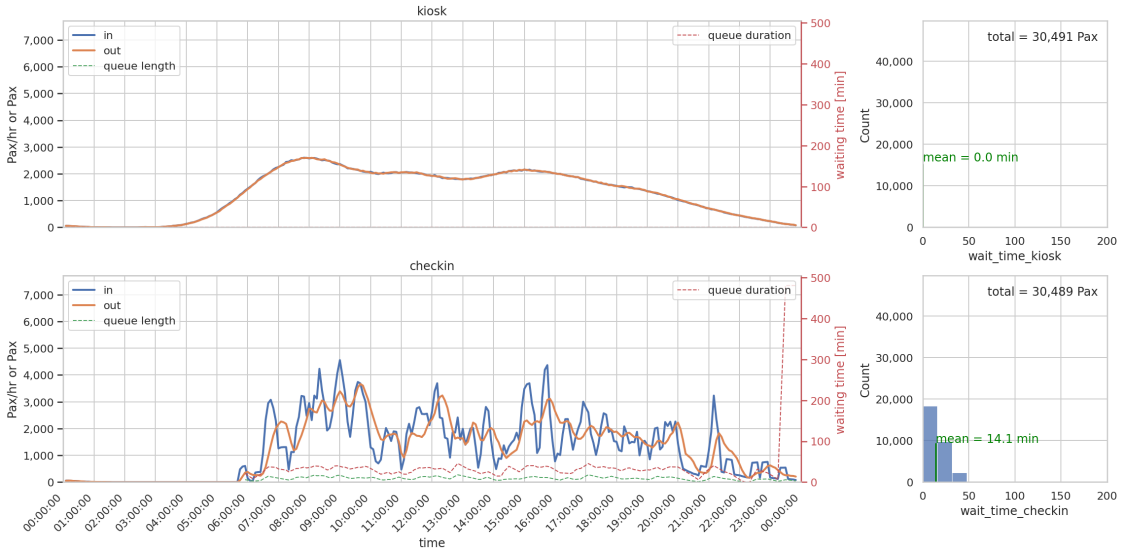
[ ]:
```
simulation2019.generate_checkin().generate_pax().run()
```

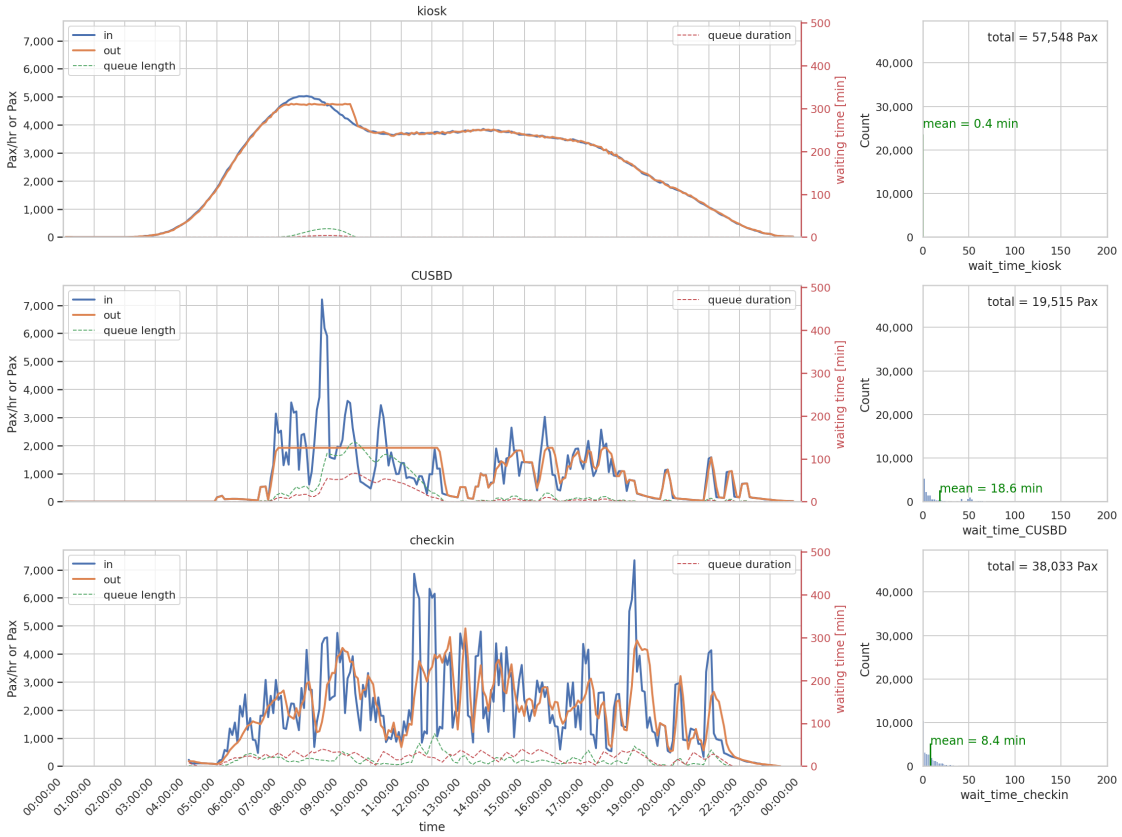
```
Simulation running…: 100%|      | 1439/1439 [03:58<00:00,  6.04it/s]
```

[ ]:
```
simulation2019.format_df_result()
```

[ ]:
```
<src.simfunc.simulation.Simulation at 0x7fb386fa2f70>
```

[ ]:
```
simulation2019.plot_result(same_scale_as=simulationT1_pat3)
simulationT1_pat3.plot_result(same_scale_as=simulation2019)
```


kiosk
in
out
queue length
queue duration
Pax/hr or Pax
waiting time [min]
total = 30,491 Pax
mean = 0.0 min
Count
wait_time_kiosk
checkin
total = 30,489 Pax
mean = 14.1 min
wait_time_checkin
time


kiosk
in
out
queue length
queue duration
Pax/hr or Pax
waiting time [min]
total = 57,548 Pax
mean = 0.4 min
Count
wait_time_kiosk
CUSBD
total = 19,515 Pax
mean = 18.6 min
wait_time_CUSBD
checkin
total = 38,033 Pax
mean = 8.4 min
wait_time_checkin
time

3 cost function test for N

[ ]:
```
# in this part, we will try to find the optimal number of CUSBD (N)
# to have the target waiting time (top90%=10 minutes)
# this is an exercize of minimizing the result of a cost function
# We create the cost function that takes N as argument and will output
# the "cost". the "cost" will be big when waiting time is different from target
# and low when waiting time is close to target (see below)
def cost_func(
    simparam: SimParam,
    target: float,  # target value for waiting time
    resource: str,  # name of the process
    N,  # N that will be tested
):
    """
    cost function for target waiting time depenging on variable
    variable can be any of simparam attributes
    cost is only based on waiting time but can be improved in the future
    """

    # set number of resource to N
    simparam.dct_resource[resource] = N

    # test to keep the latest simulation run in memory
    global simulation_last

    # run the simulation and get the results
    simulation_last = Simulation(simparam)
    simulation_last.generate_checkin().generate_pax().run()
    simulation_last.format_df_result()

    # evaluate the cost for given criteria: we use simple square distance
    cost = (simulation_last.dct_hist_wait_time[resource].quantile(q=0.9) -␣
 ↪target) ** 2

    # it is important to have a convex(?) function
    # so we need to make sure the extremities have a slope
    # if nobody waits, penalize high N
    if simulation_last.dct_hist_wait_time[resource].quantile(q=0.9) == 0:
        cost += N / 1000

    # if the top90% Pax waits 8hrs or more, penalize low N
    if simulation_last.dct_hist_wait_time[resource].quantile(q=0.9) >= 7.9 * 60:
        cost -= N / 1000

    return cost
```

[ ]:
```
# optimize
f = lambda N: cost_func(
    simparam=simparamT1,
    target=10,
    resource="CUSBD",
    N=N,
)
# "custmin" stands for custom minimizer
# it is a small function writtent to find minimum
# of a convex function
# in a very simple way with steps
res = custmin(
    f,
    guess=40,
    steps=[1],
)
print(res)

# the cost function keeps the global simulation_last in memory to be able
# to plot the graph of the last simulation run by custmin
simulation_last.plot_result()
```

```
Simulation running…: 100%|      | 1439/1439 [05:54<00:00,  4.06it/s]
x=40   error=272.25  function evaluated 0 times step taken: 1
```


```
Simulation running…: 100%|      | 1439/1439 [05:51<00:00,  4.10it/s]
x=41   error=222.50694444444449  function evaluated 2 times step taken: 1
```

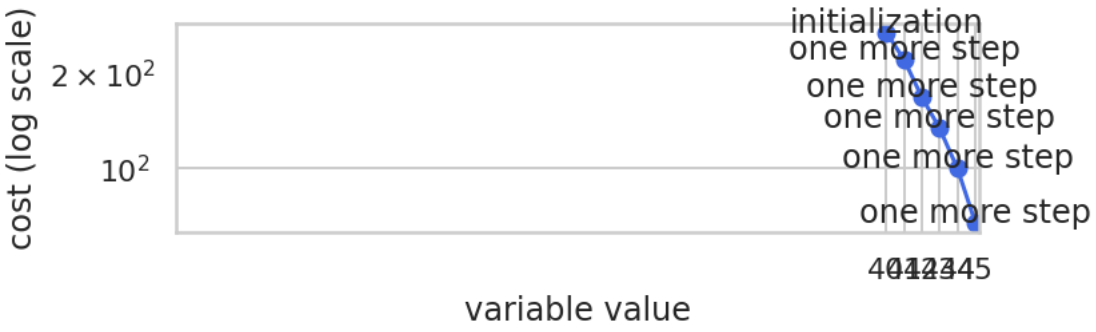

```
Simulation running…: 100%|      | 1439/1439 [07:33<00:00,  3.18it/s]
x=42   error=169.0  function evaluated 3 times step taken: 1
```


```
Simulation running…: 100%|      | 1439/1439 [05:47<00:00,  4.14it/s]
x=43   error=134.1736111111111  function evaluated 4 times step taken: 1
```

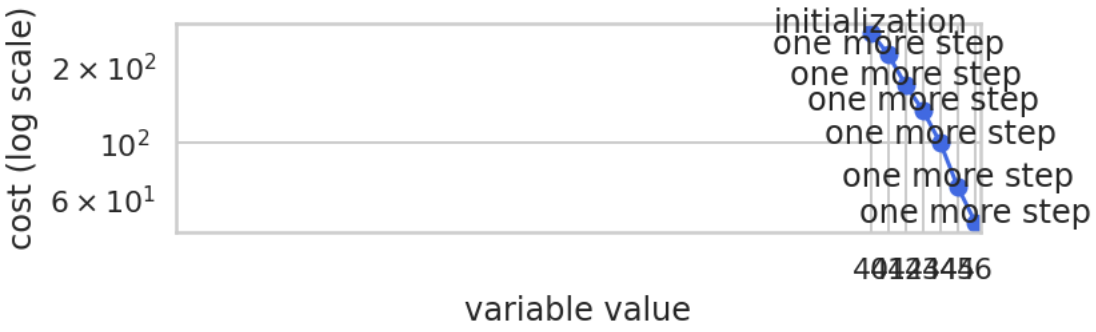

```
Simulation running…: 100%|      | 1439/1439 [05:30<00:00,  4.35it/s]
```

```
x=44   error=100.0  function evaluated 5 times step taken: 1
```


```
Simulation running…: 100%|      | 1439/1439 [05:25<00:00,  4.43it/s]
x=45   error=67.24  function evaluated 6 times step taken: 1
```



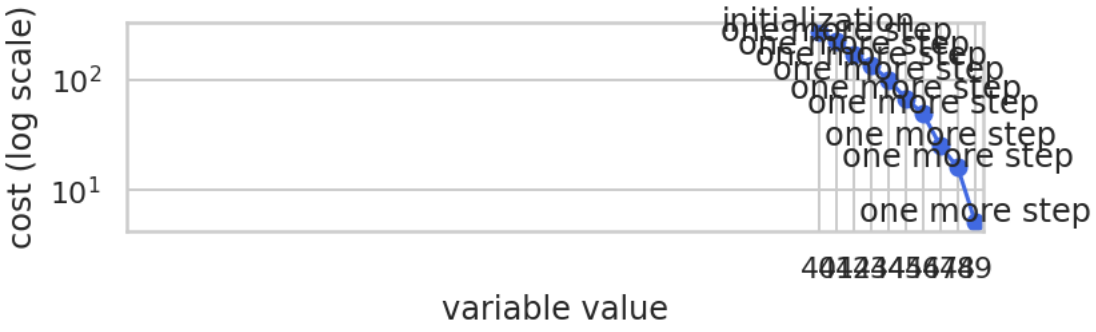

```
Simulation running…: 100%|      | 1439/1439 [05:25<00:00,  4.42it/s]
x=46   error=49.0  function evaluated 7 times step taken: 1
```



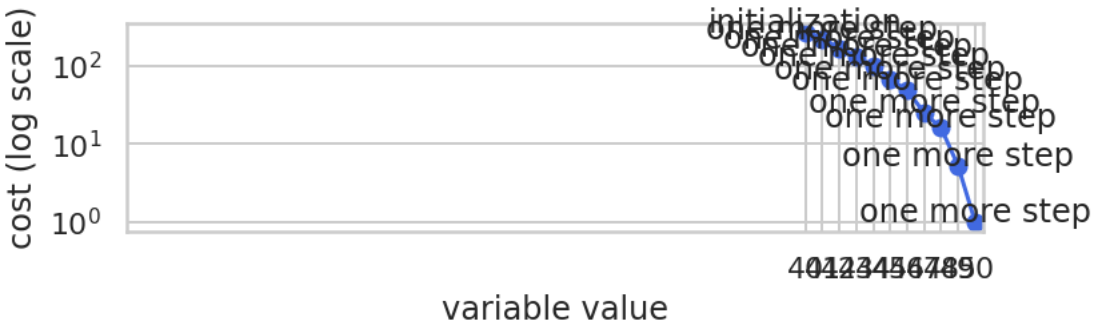
```
Simulation running…: 100%|      | 1439/1439 [05:25<00:00,  4.42it/s]
x=47   error=25.0  function evaluated 8 times step taken: 1
```


```
Simulation running…: 100%|      | 1439/1439 [06:25<00:00,  3.73it/s]
x=48   error=16.0  function evaluated 9 times step taken: 1
```

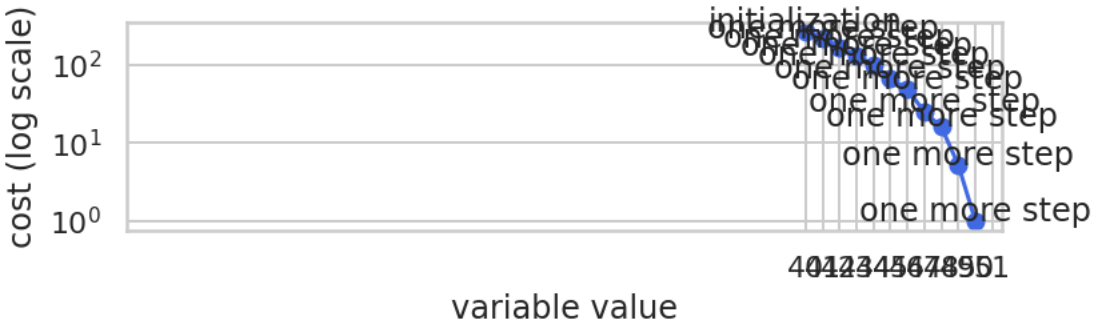

```
Simulation running…: 100%|      | 1439/1439 [06:10<00:00,  3.88it/s]
x=49   error=5.213611111111111  function evaluated 10 times step taken: 1
```



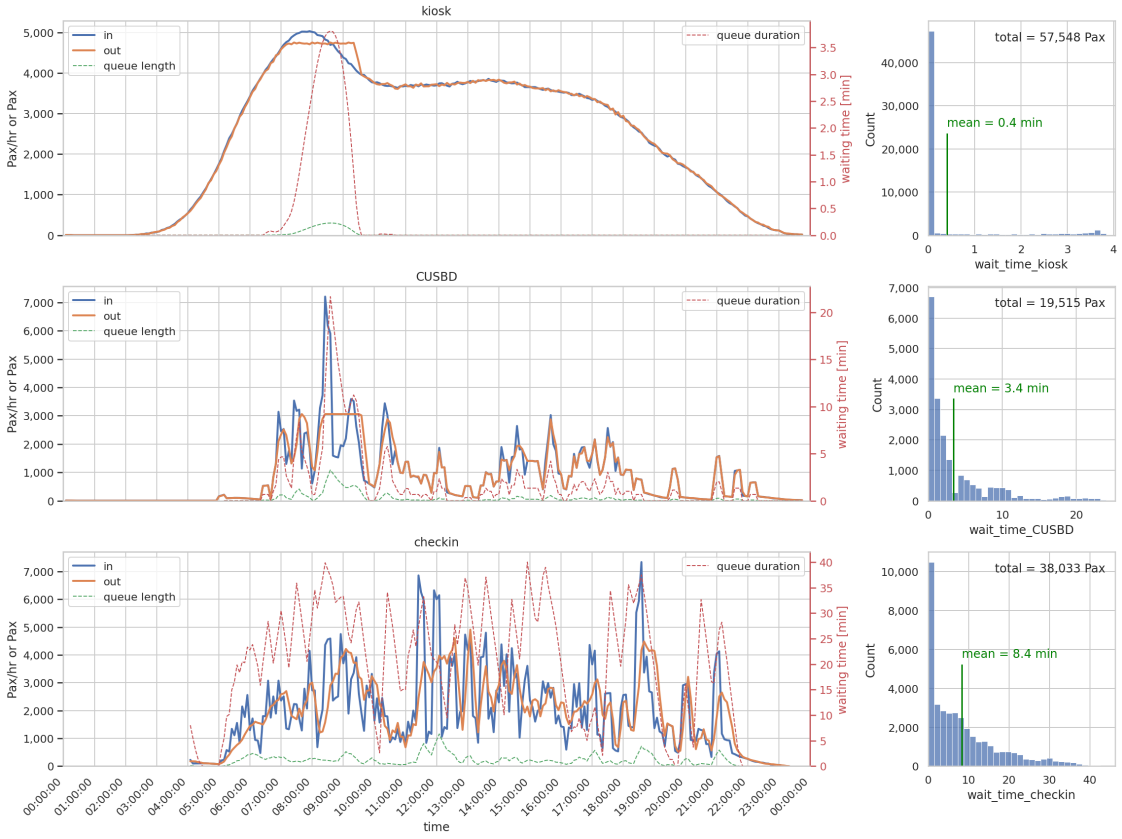
```
Simulation running…: 100%|      | 1439/1439 [06:10<00:00,  3.89it/s]
x=50   error=1.0  function evaluated 11 times step taken: 1
```




```
Simulation running…: 100%|      | 1439/1439 [06:33<00:00,  3.65it/s]
x=51   error=0.0  function evaluated 12 times step taken: 1
```




```
(51, 0.0, 'tolerance reached')
```



4 if early check-in for CUSBD

[ ]:
```
simparamT1_earlyCUSBD = deepcopy(simparamT1)

simparamT1_earlyCUSBD.dct_process_sequence = {
    "tradi": [
        ["kiosk", "wait_opening", "checkin"],
        {"hour_to_std": 2.5},
    ],
    "CUSBD": [
        ["kiosk", "wait_opening", "CUSBD"],
        {"hour_to_std": 3.5},
    ],
}
```

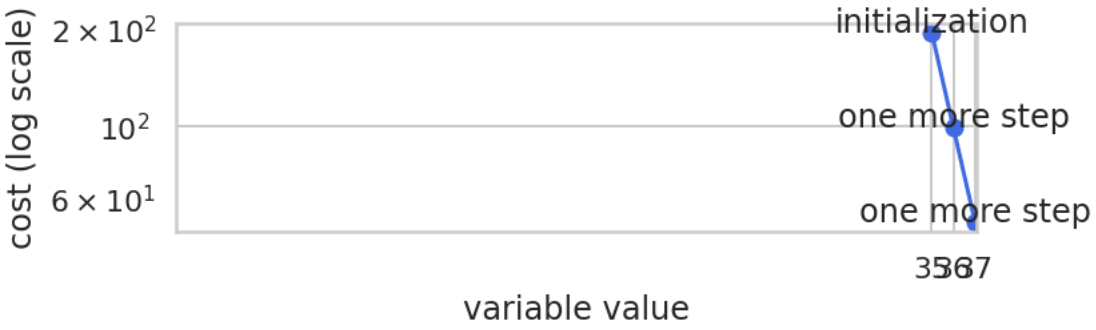
[ ]:
```
# optimize
f = lambda N: cost_func(
    simparam=simparamT1_earlyCUSBD,
    target=10,
    resource="CUSBD",
```

```
    N=N,
)
res = custmin(
    f,
    guess=35,
    steps=[1],
)
print(res)

simulation_last.plot_result()
```

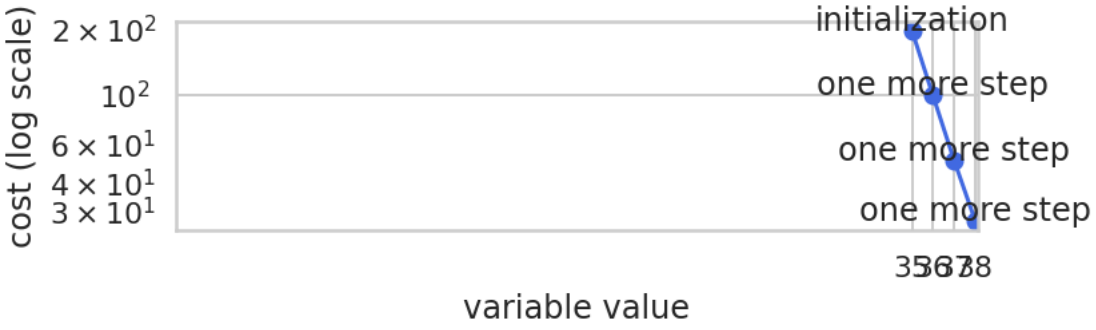
```
Simulation running…: 100%|      | 1439/1439 [06:36<00:00,  3.63it/s]
x=35   error=196.0  function evaluated 0 times step taken: 1
```


```
Simulation running…: 100%|      | 1439/1439 [05:53<00:00,  4.07it/s]
x=36   error=99.53387777777849  function evaluated 2 times step taken: 1
```


```
Simulation running…: 100%|      | 1439/1439 [05:50<00:00,  4.11it/s]
x=37   error=50.88444444444444  function evaluated 3 times step taken: 1
```



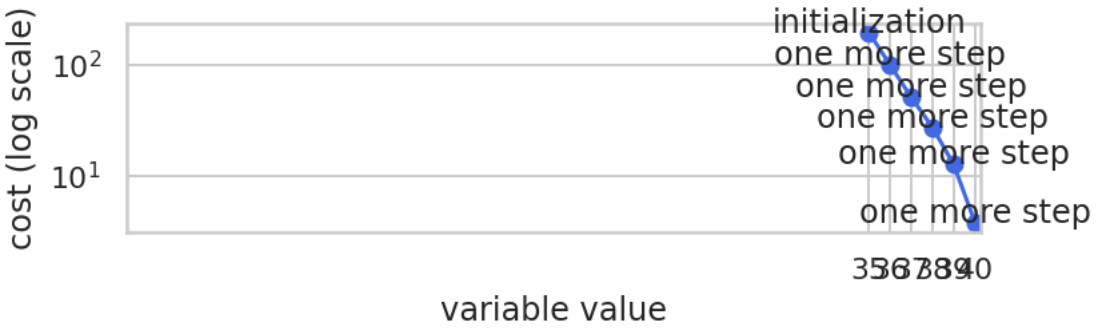

```
Simulation running…: 100%|      | 1439/1439 [05:44<00:00,  4.17it/s]
x=38   error=27.213611111111113  function evaluated 4 times step taken: 1
```



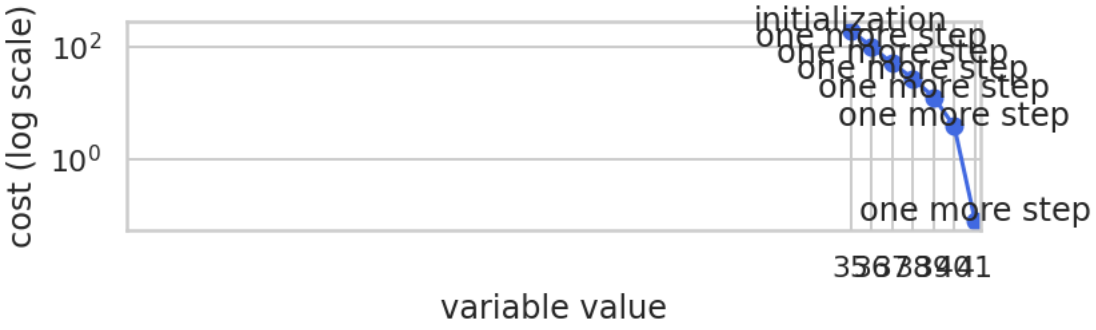

```
Simulation running…: 100%|      | 1439/1439 [05:41<00:00,  4.22it/s]
x=39   error=12.602500000000004  function evaluated 5 times step taken: 1
```


```
Simulation running…: 100%|      | 1439/1439 [05:38<00:00,  4.25it/s]
```

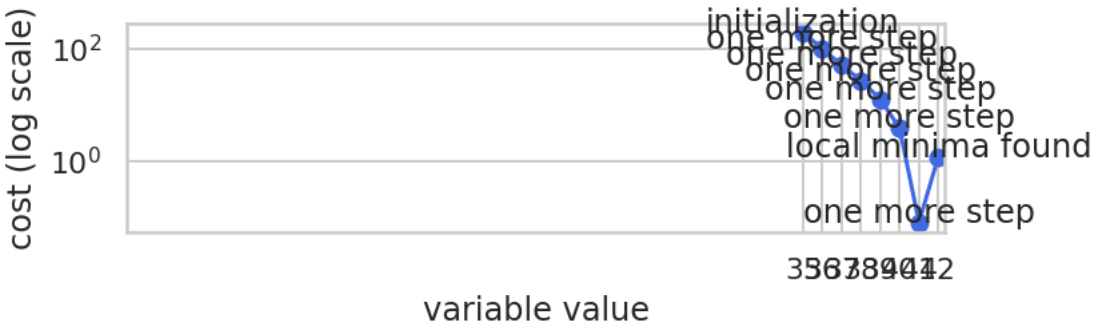
```
x=40   error=3.802499999999997  function evaluated 6 times step taken: 1
```



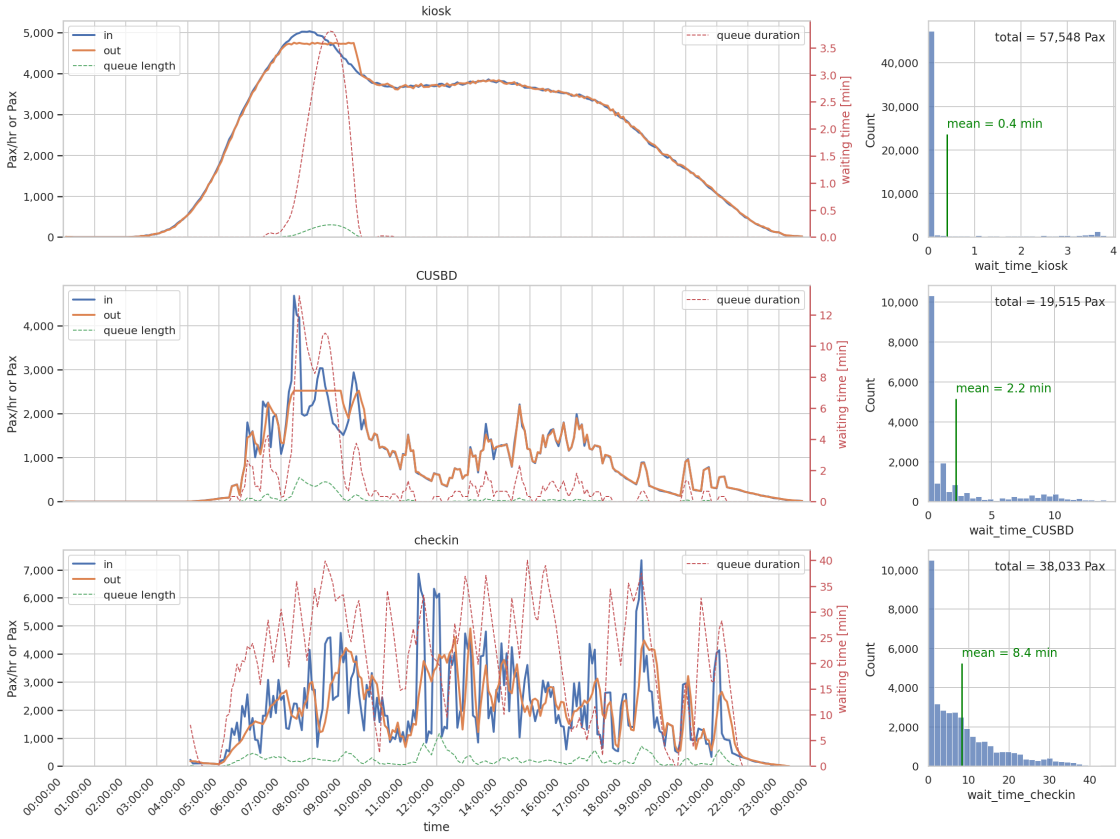

```
Simulation running…: 100%|      | 1439/1439 [05:44<00:00,  4.18it/s]
x=41   error=0.08027777777777771  function evaluated 7 times step taken: 1
```




```
Simulation running…: 100%|      | 1439/1439 [05:43<00:00,  4.19it/s]
x=42   error=1.1736111111111125  function evaluated 8 times step taken: 1
```



```
(41, 0.08027777777777771, 'local minima found')
```



5 if early checkin tradi + cusbd

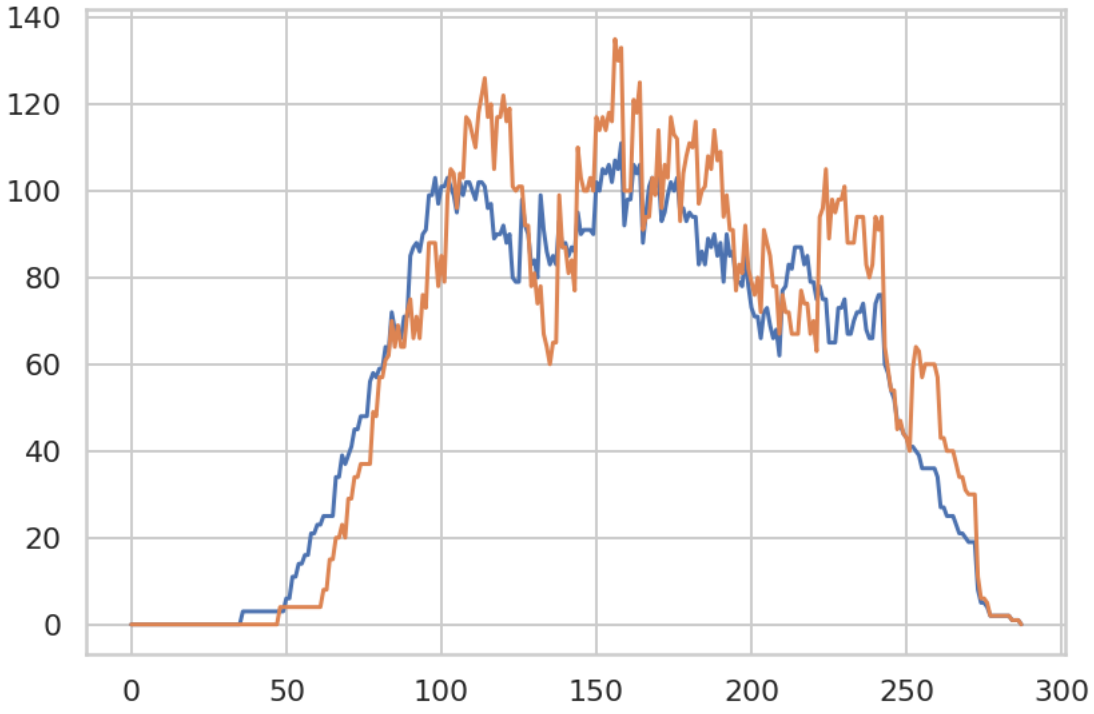
[ ]:
```
simparamT1_earlyall = deepcopy(simparamT1)

# let's set the optimum number of CUSBD
simparamT1_earlyall.dct_resource = {
    "kiosk": 92,
    "CUSBD": 41,
}

# set opening time to 3.5 hours before std
simparamT1_earlyall.dct_process_sequence = {
    "tradi": [
        ["kiosk", "wait_opening", "checkin"],
        {"hour_to_std": 3.5},
    ],
    "CUSBD": [
        ["kiosk", "wait_opening", "CUSBD"],
```

```
        {"hour_to_std": 3.5},
    ],
}

# change allocation rule for tradi counters
simparamT1_earlyall.assign_check_in(
    base_n_counter=3,
    onecounter_time=0.75,
    seats_per_add_counter=110,
    start_time=3.5,
)
```

[ ]: <src.utils.simparam.SimParam at 0x7fb3956b1df0>

```
# plot the number of counter saved
airlines_tradi = [al for al in simparamT1_earlyall.df_Counters.columns if al␣
 ↪not in airlines_CU and al != "total"]
simparamT1_earlyall.df_Counters[airlines_tradi].sum(axis=1).plot()
simparamT1_earlyCUSBD.df_Counters[airlines_tradi].sum(axis=1).plot()
```

[ ]: <AxesSubplot:>

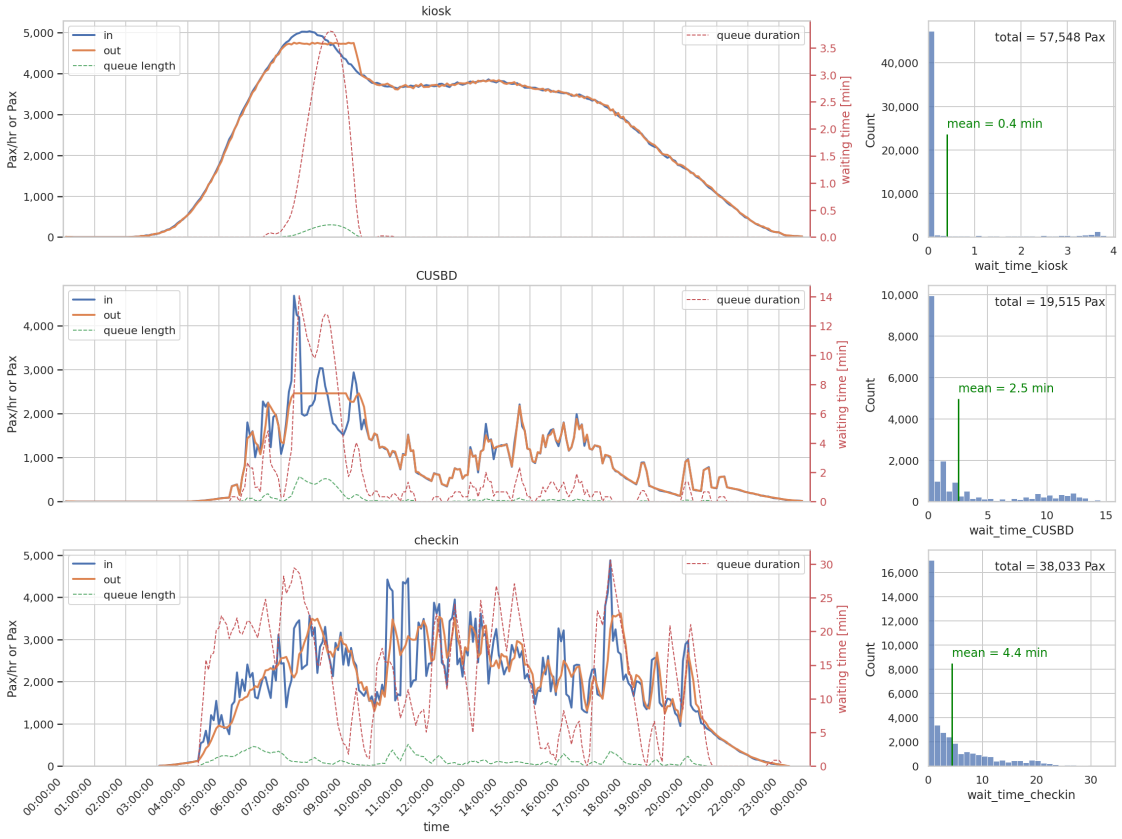


[ ]:
```
# let's check the waiting time to confirm they are still OK
simulationT1_earlyall = Simulation(simparamT1_earlyall)
simulationT1_earlyall.generate_checkin().generate_pax().run()
simulationT1_earlyall.format_df_result()
```

```
Simulation running…: 100%|      | 1439/1439 [11:48<00:00,  2.03it/s]
```

[ ]:
```
<src.simfunc.simulation.Simulation at 0x7fb36fec5cd0>
```

[ ]:
```
simulationT1_earlyall.plot_result()
```



[ ]:
```

```