Introducing `StaffPlanalytics` for Optimizing Technician Staffing Levels

Overview

`StaffPlanalytics` is a "Python package" designed to assist staff planning of service centers. The package simulates the operations of service centers based on given number of technicians, detailed service duration data and service demands arrival patterns. This package leverages discrete event simulation and agenet-based modeling techniques to provide actionable insights for service center management. It mimics the real-world service center operations and enables visualizing the impact of different staffing levels on service efficiency and customer wait times.

The developed DES package provides the following advantages:

1. **Realistic Operational Modeling**: modeling detailed operational process of service centers, including technician schedules, service duration variability, customer arrival patterns, and queue management rules.

2. **Queue Management Analysis**: Simulating service requests arrivals, service durations, service completions to track the queue lengths and wait times under different staffing levels as well as service duration and volume scenarios for each month. This helps identify whether the current number of technicians can manage peak demand periods without causing excessive delays. The implemented queue management logic is **first come, first serve**.

3. **Resource Utilization**: Tracking the utilization rates of technicians in terms of number of services finished and the duration of services. This allows us to identify periods of underutilization (suggesting overstaffing) or constant high utilization (indicating potential understaffing or the need for process improvements).

4. **Scenario Testing**: Allowing for "what-if" analyses under various conditions, such as increase in service demand volumes for particular service types, changes in service duration distributions, or variations in technician efficiency (introducing technicians fatigue or skill level factors). This helps in determining the robustness of current staffing levels against possible future changes.

5. **Bottleneck Identification**: The developed DES allows us to analyze the flow of services through service centers. It allows us to identify bottlenecks where services might be delayed, indicating areas where additional technicians or process improvements could enhance overall efficiency.

6. **Customer Satisfaction Metrics**: Beyond just measuring service times and utilization, the package allows us to simulate the impact of staffing levels on customer satisfaction metrics, such as the percentage of services completed within a target timeframe. We specifically track the number of fulfilled services each month by service type and in total.

Getting Started

The `StaffPlanalytics` consists of three modules: `declare`, `utils`, and `main`. The declare modules defines all input parameters and required data for the simulation. The `utils` module contains all functions required to run the simulation. The `main` module runs the simulation process.

Below is the `main` module which runs the simulation process.

```
# Import packages
from utils import *
import declare as dcl


# first step:
#calculate the monthly vehicle volume of each service center
_,service_center_monthly_volumes_dict=calculate_service_centers_monthly_delivered_v


# second step:
#create service centers objects with no technicians
service_centers=create_service_centers(number_of_service_centers=dcl.number_of_serv


# Initialize a list to store performance metrics from each iteration
all_performance_metrics = []

# Initialize a list to store DataFrames from each iteration
all_monthly_performance_metrics = []

#simulate each service centers performance based on the given input parameters
and store the results of each month and scenarios in the monthly performanc
report
for service_center in service_centers:
    for number_of_technicians in range(dcl.minimum_number_of_technicians,
dcl.maximum_number_of_technicians+1):
        monthly_performance_report, _ =
simulate_service_centers_performance(service_center=service_center,
year=dcl.year, number_of_months=dcl.number_of_months,
new_vehicles_monthly_volumes=service_center_monthly_volumes_dict[f'Service
Center {service_center.id}'],
number_of_technicians_to_hire=number_of_technicians,
number_of_scenarios=dcl.number_of_simulation_scenarios)

        # Initialize a dictionary to store the average monthly performance
metrics
        average_monthly_performance_metrics = {}

        for key in dcl.reportings_monthly_performance_metrics_keys:
            # Compute the mean across scenarios (axis=0) for each month
            average_monthly_performance_metrics[key] =
np.mean(monthly_performance_report[key], axis=0)

        # Create a DataFrame for the current iteration
```

```
            df_current_monthly = pd.DataFrame(average_monthly_performance_metrics)
            df_current_monthly['Month'] = np.arange(1, dcl.number_of_months + 1)

            # Add identifiers for Service Center ID and Number of Technicians
            df_current_monthly['Service Center ID'] = service_center.id
            df_current_monthly['Number of Technicians'] = number_of_technicians

            # Append the current DataFrame to the list
            all_monthly_performance_metrics.append(df_current_monthly)
            #
plot_summary_results(service_center=service_center,monthly_performance_report=montl
# Concatenate all DataFrames in the list into a final DataFrame
df_final_monthly_performance_metrics =
pd.concat(all_monthly_performance_metrics)

# Reset the index if you want the final DataFrame to have a continuous index
df_final_monthly_performance_metrics.reset_index(drop=True, inplace=True)

# Save the final DataFrame to a CSV file
df_final_monthly_performance_metrics.to_csv('all_service_centers_average_monthly_pe
```

Note that the main module uses the declare and utils module to run the process.

Introducing main functions of the package

To simulate the operations of service centers, we use object-oriented programming technique. We define three classes for three main objects, Vehicle, Technician, and Service centers to showcase how they interact with each other. Each object has specific set of attributes:

Vehicle Class

```
class Vehicle:
    def __init__(self, arrival_month,service_type,service_center_id):
        self.arrival_month=arrival_month #Arrival month
        self.service_type = service_type  # "pre-delivery" ,  "post-delivery" ,
"None"
        self.under_service=False
        self.start_service_time=None
        self.remaining_service_time=None
        self.service_duration = None
        self.assigned_technician=None
        self.service_center_id=service_center_id
        self.waiting_time=0


    #reseting the information of vehicle after being serviced
    def reset_after_serviced(self):
        self.arrival_month = None
        self.service_type = None  # "pre-delivery" ,  "post-delivery" , "None"
        self.under_service = False
        self.start_service_time = None
        self.remaining_service_time = None
```

```
        self.service_duration = None
        self.assigned_technician = None

    def start_getting_service(self,time,technician):
        self.assigned_technician=technician
        self.start_service_time=time
        self.under_service=True
        self.waiting_time+=time
        self.remaining_service_time=self.service_duration


    def request_post_delivery_service(self,request_month,service_duration):
        self.arrival_month=request_month
        self.service_type=dcl.post_delivery_service_type_str
        self.set_service_duration(service_duration)

    def set_service_duration(self, duration):
        self.service_duration = duration

    def get_assigned_technician_info(self):
        if self.assigned_technician!=None:
            return self.assigned_technician.id
        else: 'NA'

    def all_attributes(self):
        attributes = {"vehicle info: "
        "arrival month": self.arrival_month,  # Arrival month
        "service_type":self.service_type , # "pre-delivery" or "post-delivery"
        "start_service_time":self.start_service_time ,
        "remaining_service_time":self.remaining_service_time ,
        "service_duration":self.service_duration ,
        "assigned technician id":self.get_assigned_technician_info(),
        " assigned service center id":self.service_center_id
        }
        return attributes
```

Technician Class

```
class Technician:
    def __init__(self, id):
        self.id = id
        self.type='Full time'
        self.is_available=True
        self.worked_hours=0
        self.idle_time=np.ones(168)
        self.available_hours = 168  # Monthly available hours
        self.number_of_services_finished=0
        self.time_to_finish_service=None

    def reset_new_month(self):
        if self.is_available or self.time_to_finish_service==0:
            self.is_available = True
            self.worked_hours = 0
```

```
            self.available_hours =
dcl.max_working_hours_of_technicians_in_month  # Monthly available hours
            self.number_of_services_finished = 0
            self.time_to_finish_service = None
        else:
            self.is_available = False
            self.worked_hours = 0
            self.available_hours =
dcl.max_working_hours_of_technicians_in_month-self.time_to_finish_service  #
Monthly available hours
            self.number_of_services_finished = 0

    def start_service_task(self,task_duration):
        self.is_available=False
        self.time_to_finish_service=task_duration
        self.available_hours-=task_duration

    def finished_service_task(self,task_duration):
        self.is_available=True
        self.number_of_services_finished+=1
        self.time_to_finish_service=0

    def still_working_on_vehicle(self):
        self.time_to_finish_service-=1
        self.worked_hours+=1

    def all_attributes(self):
        attributes = {
        "technician id":self.id,
        "is_available":self.is_available,
        "time to finish service":self.time_to_finish_service,
        "worked_hours": self.worked_hours,
        "available_hours":self.available_hours,  # Monthly available hours
        "number_of_services_finished": self.number_of_services_finished
        }
        return attributes
```

Service Center Class

```
class ServiceCenter:
    _Time_tracker=0
    def
__init__(self,id:int,max_working_hours:int,pre_delivery_service_duration_mean,pre_d
        self.id=id

self.pre_delivery_service_duration_mean=pre_delivery_service_duration_mean

self.pre_delivery_service_duration_variance=pre_delivery_service_duration_variance
        self.post_delivery_service_arrival_mean=post_delivery_arrival_mean

self.post_delivery_service_arrival_variance=post_delivery_arrival_variance

self.post_delivery_service_durations_data=post_delivery_service_durations_events_da
```

```
        self.technicians = []
        self.vehicles_under_service=[]
        self.queue = []
        self.max_working_hours=max_working_hours

        self.vehicles_serviced={(month+1,service_type):[]for month in
range(dcl.number_of_months) for service_type in dcl.list_of_service_types}
        self.queue_waiting_time={month+1:[]for month in
range(dcl.number_of_months)}
        self.service_demands_durations_this_month=
{dcl.pre_delivery_service_type_str:[],dcl.post_delivery_service_type_str:[]}
#dict to save new services durations of the month
        self.service_demands_durations_fulfilled_this_month=
{(month+1,service_type):[]for month in range(dcl.number_of_months) for
service_type in dcl.list_of_service_types} #to save the duration of services
that are not fulfilled this month
        self.service_demands_durations_backlog={(month+1,service_type):[]for
month in range(dcl.number_of_months) for service_type in
dcl.list_of_service_types} #to save the duration of services that are not
fulfilled this month
        # and are shifted to the next month

    def reset_monthly_service_demand_durations_list(self):
        self.service_demands_durations_this_month=
{dcl.pre_delivery_service_type_str:[],dcl.post_delivery_service_type_str:[]}
        self.service_demands_durations_fulfilled_this_month=
{(month+1,service_type):[]for month in range(dcl.number_of_months) for
service_type in dcl.list_of_service_types} #to save the duration of services
that are not fulfilled this month

    def reset_for_new_scenarios(self):
        self.queue=[]
        self.vehicles_under_service=[]
        self.vehicles_serviced={(month+1,service_type):[]for month in
range(dcl.number_of_months) for service_type in dcl.list_of_service_types}
        self.queue_waiting_time={month+1:[]for month in
range(dcl.number_of_months)}
        self.service_demands_durations_this_month=
{dcl.pre_delivery_service_type_str:[],dcl.post_delivery_service_type_str:[]}
#dict to save new services durations of the month
        self.service_demands_durations_fulfilled_this_month=
{(month+1,service_type):[]for month in range(dcl.number_of_months) for
service_type in dcl.list_of_service_types} #to save the duration of services
that are not fulfilled this month
        self.service_demands_durations_backlog={(month+1,service_type):[]for
month in range(dcl.number_of_months) for service_type in
dcl.list_of_service_types} #to save the duration of services that are not
fulfilled this month


    def add_technician(self, technician):
        self.technicians.append(technician)
```

```
    def add_vehicle_to_queue(self, vehicle):
        self.queue.append(vehicle)

self.service_demands_durations_this_month[vehicle.service_type].append(vehicle.serv


    def set_pre_delivery_service_duration(self):
        # ------------ generate service duration -------------
        std_dev = np.sqrt(self.pre_delivery_service_duration_variance)
        # Draw a random number from a normal distribution
        lower_bound = 0
        upper_bound = np.inf  # Using np.inf as there's no upper limit
        # Convert bounds to z-scores
        a = (lower_bound - self.pre_delivery_service_duration_mean) / std_dev
        b = (upper_bound - self.pre_delivery_service_duration_mean) / std_dev
        # Generate one random number
        service_duration = np.ceil(truncnorm.rvs(a, b,
loc=self.pre_delivery_service_duration_mean, scale=std_dev))
        return service_duration

    def set_post_delivery_service_duration(self):
        return np.random.choice(self.post_delivery_service_durations_data)

    # New vehicles require pre_delivery services
    def generate_new_vehicles(self,month, pre_delivery_volume):
        vehicles_requested_for_pre_delivery_service=[]

        for _ in range(int(pre_delivery_volume)): # create vehicle
            vehicle =
Vehicle(arrival_month=month,service_type=dcl.pre_delivery_service_type_str,service_

            #generate service duration
            service_duration=self.set_pre_delivery_service_duration()
            #-----------------------------------------------------
            vehicle.set_service_duration(service_duration)
            self.add_vehicle_to_queue(vehicle)
            vehicles_requested_for_pre_delivery_service.append(vehicle)
        log_info(f'< {len(vehicles_requested_for_pre_delivery_service)} > new
vehicles with pre-delivery service requests joined the queue')
        return vehicles_requested_for_pre_delivery_service

    def generate_post_delivery_requests(self,month,vehicles_on_the_road):
        vehicles_requested_for_post_delivery_service=[]
        for vehicle in vehicles_on_the_road:
            if random.random()< self.post_delivery_service_arrival_mean:
                service_duration=self.set_post_delivery_service_duration()

vehicle.request_post_delivery_service(request_month=month,service_duration=service_
                vehicles_requested_for_post_delivery_service.append(vehicle)
                self.add_vehicle_to_queue(vehicle)
        log_info(f'(0-{month}-2023)|
{len(vehicles_requested_for_post_delivery_service)} post delivery requests
```

```
created')

        return vehicles_requested_for_post_delivery_service

    def simulate_month(self, month):
        number_of_serviced_vehicles_this_month = 0
        # beginning of the month
        time = 0
        while time < self.max_working_hours:

            temp_vehicles_serviced = []
            # first check if there is any vehicles under maintneance from
previous time periods
            for vehicle in self.vehicles_under_service:
                # if a vehicle has been under service from previous months
                if vehicle.remaining_service_time>0:
                    vehicle.assigned_technician.still_working_on_vehicle()
#time_to_finish_service-=1 & worked_hours+=1
                    vehicle.remaining_service_time -= 1

                if vehicle.remaining_service_time == 0:
                    log_info(f'technician {vehicle.assigned_technician.id}
finished its {vehicle.service_type} service at t={time}')
                    number_of_serviced_vehicles_this_month += 1

self.vehicles_serviced[(month,vehicle.service_type)].append(vehicle)

self.service_demands_durations_fulfilled_this_month[(month,vehicle.service_type)].a

vehicle.assigned_technician.finished_service_task(task_duration=vehicle.service_dur
                    # log_objects_status('updated technician info',
vehicle.assigned_technician)
                    # print(vehicle.assigned_technician.all_attributes())
                    vehicle.reset_after_serviced()
                    temp_vehicles_serviced.append(vehicle)

            # updating the under service and serviced vehicles lists
            for vehicle in temp_vehicles_serviced:
                self.vehicles_under_service.remove(vehicle)

            temp_vehicles_under_service = []
            if any(technician.is_available for technician in self.technicians):
                for vehicle in self.queue:
                    for technician in self.technicians:
                        if technician.available_hours >=
vehicle.service_duration and technician.is_available and not
vehicle.under_service:
                            log_info(f'technician {technician.id} starts a
{vehicle.service_type} service at hour={time} for duration of
{vehicle.service_duration} for the following vehicle:')
                            vehicle.start_getting_service(time=time,
technician=technician)
```

```
                self.queue_waiting_time[month].append(vehicle.waiting_time)

                technician.start_service_task(task_duration=vehicle.service_duration)
                                        temp_vehicles_under_service.append(vehicle)
                                        log_info(f'vehicle service_type
{vehicle.service_type} | remaining_service_time
{vehicle.remaining_service_time} | waiting_time={vehicle.waiting_time} |
service duration {vehicle.service_duration} | technician
{vehicle.assigned_technician.id}')
                                        break  # move to the next vehicle
                    else:
                        log_info(f"All technicians are busy @ hour = {time} and can't
service the queue")

                    for vehicle in temp_vehicles_under_service:
                        self.queue.remove(vehicle)
                        self.vehicles_under_service.append(vehicle)
                    log_info(f'hour {time}-({month}-{dcl.year}) | queue size :
{len(self.queue)}')

                    # check to see if the queue is empty and all services are finished
by technicians
                    if len(self.queue) == 0 and len(self.vehicles_under_service) == 0:
                        log_info(f'The queue is empty & All service requests are
fullfilled @ hour = {time}')

                        # set the time to the end of month
                        time = dcl.max_working_hours_service_center_in_month - 1
                    time += 1
                    if time!=0:
                        print('-' * 120)
                        print(f"{'':<50}status info: service center {'':<50}")
                        print(self.all_attributes(month))
                        log_objects_status('technicians status', self.technicians)
                    log_info('rolling forward to simulate the next hour')
                    print('-'*120)
                    log_info(f' Date:{month}-{dcl.year}| Hour {time}')
                    print('-' * 120)


                #=============================================================================
                for vehicle in self.vehicles_under_service:
                    fulfilled_service=vehicle.service_duration-
vehicle.remaining_service_time
                    self.service_demands_durations_fulfilled_this_month[(month,
vehicle.service_type)].append(fulfilled_service)

                #=============================================================================
                self.current_time = time
                log_info(info_str=f'Date: {month}-{dcl.year} | Hour {time} | end of
simulation of this month')

                #=============================================================================
```

```
    def
monthly_summary_metrics(self,month,scenario,new_generated_vehicles_this_month,vehic
        #----------------------------------New Incoming service demand info
of the month-----------------------------------------------------
        #Number of service requests made this month
        dcl.reporting_monthly_performance_metrics[dcl.number_of_services_str]
[scenario-1][month-1] = len(new_generated_vehicles_this_month) +
len(vehicles_post_delivery_service_this_month)

dcl.reporting_monthly_performance_metrics[dcl.number_of_pre_delivery_services_str]
[scenario-1][month-1] = len(new_generated_vehicles_this_month)

dcl.reporting_monthly_performance_metrics[dcl.number_of_post_delivery_services_str]
[scenario-1][month-1] = len(vehicles_post_delivery_service_this_month)

        # requested service demand durations of in the month

dcl.reporting_monthly_performance_metrics[dcl.pre_delivery_service_demands_duratior
[scenario-1][month-1] =
np.sum(self.service_demands_durations_this_month[dcl.pre_delivery_service_type_str]

dcl.reporting_monthly_performance_metrics[dcl.post_delivery_service_demands_duratic
[scenario-1][month-1]=
np.sum(self.service_demands_durations_this_month[dcl.post_delivery_service_type_str

dcl.reporting_monthly_performance_metrics[dcl.total_service_demands_duration_str]
[scenario-1][month-1] =
np.sum(self.service_demands_durations_this_month[dcl.pre_delivery_service_type_str]
+
np.sum(self.service_demands_durations_this_month[dcl.post_delivery_service_type_str

        #-------------------------------------------------------------------
-------------------------------------------------------------------
        #number of vehicles start getting service this month , including under
service vehicles

dcl.reporting_monthly_performance_metrics[dcl.number_of_fulfilled_services_str]
[scenario-1][month-1] =
len(self.vehicles_serviced[(month,dcl.pre_delivery_service_type_str)]) +
len(self.vehicles_serviced[(month,dcl.post_delivery_service_type_str)]) +
len([1 for _ in self.vehicles_under_service])

dcl.reporting_monthly_performance_metrics[dcl.number_of_fulfilled_pre_delivery_serv
[scenario-1][month-1]=
len(self.vehicles_serviced[(month,dcl.pre_delivery_service_type_str)]) + len([1
for vehicle in self.vehicles_under_service if
vehicle.service_type==dcl.pre_delivery_service_type_str])

dcl.reporting_monthly_performance_metrics[dcl.number_of_fulfilled_post_delivery_ser
[scenario-1][month-1] =
len(self.vehicles_serviced[(month,dcl.post_delivery_service_type_str)]) +
len([1 for vehicle in self.vehicles_under_service if
vehicle.service_type==dcl.post_delivery_service_type_str])
```

```
        #service durations fulfilled , including under service vehicles

dcl.reporting_monthly_performance_metrics[dcl.duration_of_fulfilled_services_str]
[scenario-1][month-1]=
np.sum(self.service_demands_durations_fulfilled_this_month[(month,dcl.pre_delivery_
+
np.sum(self.service_demands_durations_fulfilled_this_month[(month,dcl.post_delivery

dcl.reporting_monthly_performance_metrics[dcl.duration_of_fulfilled_pre_delivery_se
[scenario-1][month-1] =
np.sum(self.service_demands_durations_fulfilled_this_month[(month,dcl.pre_delivery_

dcl.reporting_monthly_performance_metrics[dcl.duration_of_fulfilled_post_delivery_s
[scenario-1][month-1]=
np.sum(self.service_demands_durations_fulfilled_this_month[(month,dcl.post_delivery



dcl.reporting_monthly_performance_metrics[dcl.technicians_worked_hours_str]
[scenario-1][month-1]=np.sum([technician.worked_hours for technician in
self.technicians])
        #waiting time of the vehicles serviced
        dcl.reporting_monthly_performance_metrics[dcl.queue_waiting_time_str]
[scenario-1][month-1]=np.mean(self.queue_waiting_time[month])


        #requested service demand durations of in the month

dcl.reporting_monthly_performance_metrics[dcl.pre_delivery_service_demands_duratior
[scenario-1][month-
1]=np.sum(self.service_demands_durations_this_month[dcl.pre_delivery_service_type_s

dcl.reporting_monthly_performance_metrics[dcl.post_delivery_service_demands_duratic
[scenario-1][month-
1]=np.sum(self.service_demands_durations_this_month[dcl.post_delivery_service_type_

dcl.reporting_monthly_performance_metrics[dcl.total_service_demands_duration_str]
[scenario-1][month-
1]=np.sum(self.service_demands_durations_this_month[dcl.pre_delivery_service_type_s
        if month!=1:

dcl.reporting_monthly_performance_metrics[dcl.backlog_service_demand_durations_str
[scenario-1][month-1]=np.sum(self.service_demands_durations_backlog[(month-
1,dcl.pre_delivery_service_type_str)]+self.service_demands_durations_backlog[(month
1,dcl.post_delivery_service_type_str)]) #backlog service of month 1 would be
counted in month 2 #todo check


        return dcl.reporting_monthly_performance_metrics
    def comulative_summary_metrics(self,month,scenario):

dcl.reporting_comulative_performance_metrics[dcl.number_of_services_str]
```

```
[scenario-1][month-1] =
np.sum(dcl.reporting_monthly_performance_metrics[dcl.number_of_services_str]
[scenario-1][:month])

dcl.reporting_comulative_performance_metrics[dcl.number_of_pre_delivery_services_st
[scenario-1][month-1] =
np.sum(dcl.reporting_monthly_performance_metrics[dcl.number_of_pre_delivery_service
[scenario-1][:month])

dcl.reporting_comulative_performance_metrics[dcl.number_of_post_delivery_services_s
[scenario-1][month-1] =
np.sum(dcl.reporting_monthly_performance_metrics[dcl.number_of_post_delivery_servic
[scenario-1][:month])



dcl.reporting_comulative_performance_metrics[dcl.number_of_fulfilled_services_str]
[scenario-1][month-1] =
np.sum(dcl.reporting_monthly_performance_metrics[dcl.number_of_fulfilled_services_s
[scenario-1][:month])

dcl.reporting_comulative_performance_metrics[dcl.number_of_fulfilled_pre_delivery_s
[scenario-1][month-1] =
np.sum(dcl.reporting_monthly_performance_metrics[dcl.number_of_fulfilled_pre_delive
[scenario-1][:month])

dcl.reporting_comulative_performance_metrics[dcl.number_of_fulfilled_post_delivery_
[scenario-1][month-1]=
np.sum(dcl.reporting_monthly_performance_metrics[dcl.number_of_fulfilled_post_deliv
[scenario-1][:month])


dcl.reporting_comulative_performance_metrics[dcl.technicians_worked_hours_str]
[scenario-1][month-
1]=np.sum(dcl.reporting_monthly_performance_metrics[dcl.technicians_worked_hours_st
[scenario-1][:month])
        return dcl.reporting_comulative_performance_metrics

    def all_attributes(self,month):
        attributes = {
        "service center id": self.id,
        "technicians":{'free':len([1 for technician in self.technicians if
technician.is_available]), 'busy':len([1 for technician in self.technicians if
not technician.is_available]) , 'total':len(self.technicians)},
        "vehicles in the queue for":{dcl.pre_delivery_service_type_str: len([1
for vehicle in self.queue if
vehicle.service_type==dcl.pre_delivery_service_type_str]),dcl.post_delivery_service
len([1 for vehicle in self.queue if vehicle.service_type ==
dcl.post_delivery_service_type_str]),"total": len(self.queue) },
        "serviced vehicles":{dcl.pre_delivery_service_type_str:len(
self.vehicles_serviced[(month,dcl.pre_delivery_service_type_str)]),dcl.post_deliver
        "under service vehicles":{dcl.pre_delivery_service_type_str:len([1 for
vehicle in self.vehicles_under_service if
```

```
vehicle.service_type==dcl.pre_delivery_service_type_str]),dcl.post_delivery_service
for vehicle in self.vehicles_under_service if
vehicle.service_type==dcl.post_delivery_service_type_str]),'total':len(self.vehicle

        }
        return attributes
```

Main Simulation function

the following function simulated the performance of the service centers.

```
def
simulate_service_centers_performance(service_center,year,number_of_months,new_vehi


#=============================================================================
    print(dcl.section)
    log_info(f'simulating the operations of service center {service_center.id}
in {year}')
    print(dcl.section)
    #=========================================================================

#^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    #assumption:
    # 1. no vehicles are serviced by this service center before
    # 2. if a vehicle is serviced by this service center, it would always be
serviced by the same service center
    #=========================================================================
    #add technicians to the service center
    for i in range(number_of_technicians_to_hire):
        service_center.add_technician(Technician(i+1))

#^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    # assumption:
    # number of hired technicians remain the same over the year
    # can drop this assumption by hiring full-time and part-time technicians
    # number of full time technicians can be defined based on min of 95%
confidence interval of the average service time
    # number of part-time technicians can be determined based on other service
duration distribution statistics such as Q3 to manage the peak loads

#^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    #=========================================================================
    for scenario in range(1,number_of_scenarios+1):
        service_center_vehicles_volume = []  # list of vehicle volumes
        service_center.reset_for_new_scenarios()
        for month in range(1,number_of_months+1):

            vehicles_requested_for_post_delivery_service=[] #list of vehicles
request for post-delivery services
            service_center.reset_monthly_service_demand_durations_list() #reset
a list that records each month's service demand durations (backlog of previous
```

```
months, unfulfilled services from previous months, are not included in this
list)


            log_info(f"Date({month}-{year}) | upcoming pre-delivery service
requests | {new_vehicles_monthly_volumes[month-1]}")
            log_info('generate new vehicles based on the vehicle demand
volume') # Volume is assumed to be known and deterministic

            # new vehicles arrive at the service center to get pre-delivery
services
            #these vehicles are added to the service center's queue
            #queueing system : first-come first-serve
            #
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
            #assumptions:
            # All service demand request are revealed at the beginning of each
month ; they arrive at the hour=0 of each month
            # In the first month, only pre-delivery services are requested;
i.e. no vehicles are on the road to request for post delivery services
            #
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

vehicles_requested_for_pre_delivery_service=service_center.generate_new_vehicles(mo
1])

            #adding the new produced vehicles to the list of

service_center_vehicles_volume.extend(vehicles_requested_for_pre_delivery_service)
            vehicles_on_the_road_not_requested_service = [vehicle for vehicle
in service_center_vehicles_volume if vehicle.service_type not in ['pre-
delivery', 'post-delivery']]
            if len(vehicles_on_the_road_not_requested_service)!=0: #if some
vehicles are in the service center territory
                #count how many of them 1. finished it's predelivery service
and 2. have not yet requested for any post-delievry services services. these
vehicles are potential candidats for post-delivery service this month
                log_info(f'{len(vehicles_on_the_road_not_requested_service)}
vehicles are on the road and may request for post-delivery service')
                log_info('generate post delivery requests')

                #generate post delivery service requests using 1. number of
vehicles on the road, 2. post-delivery arrival mean, 3.post-delivery arrival
variance
                #these vehicles are added to the service center's queue

vehicles_requested_for_post_delivery_service=service_center.generate_post_delivery_
                log_info(f"Date({month}-{year}) | upcoming post-delivery
service requests | {len(vehicles_requested_for_post_delivery_service)}")

#================================================================================
            # service centers status at the beginning of the month
            #
```

```
            print('-'*120)
            print(f"{'':<50}status info: service center {'':<50}")
            print(service_center.all_attributes(month))
            log_objects_status(f"Technicians status
",service_center.technicians)
            log_objects_status(f"Queue status ",service_center.queue)


        #=============================================================================
            # Simulate the month
            log_info(f" simulation started")
            service_center.simulate_month(month)
            log_info(info_str=f"simulation finished")
            log_objects_status(info_str=f"
({dcl.max_working_hours_service_center_in_month}-{month}-
{dcl.year})|",val=service_center)
           #-----------------------------------------------
            # updating the statuses for the next round of simulation

            for vehicle in service_center.queue:
                vehicle.waiting_time += service_center.max_working_hours

        service_center.queue_waiting_time[month].append(vehicle.waiting_time)
                service_center.service_demands_durations_backlog[(month,
        vehicle.service_type)].append(vehicle.service_duration)
            for vehicle in service_center.vehicles_under_service:
                service_center.service_demands_durations_backlog[(month,
        vehicle.service_type)].append(vehicle.remaining_service_time)


        monthly_performance_report=service_center.monthly_summary_metrics(month=month,scena

        cumulative_performance_report=service_center.comulative_summary_metrics(month=montl


            for technician in service_center.technicians:
                technician.reset_new_month()
            if not dcl.disable_print_summary_metrics:

        log_summary_metrics(month=month,monthly_summary_metrics=monthly_performance_report,
            log_info(info_str='rolling forward to simulate the next month')
            print(section)
    return monthly_performance_report,cumulative_performance_report
```

In [ ]:

In [ ]: