Libraries

```
#Importing all the libraries required for the project.
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import scipy.stats as stats
import math
import numpy as np
import random

!pip install simpy
import simpy
```

SHOW HIDDEN OUTPUT

```
ε = 0.00001

def isZero(x):
    return abs(x)<ε
```

Entities

```
# Time tolerance: when at current speed difference a crash might occur within that number of
CRITICAL_TIME_TOLERANCE = 4  # [s]
LANE_CHANGE_TIME = 2 # [s]
MIN_TIME_DIFF = 0.5

MIN_SPEED_DIFF = 3 # [m/s] min speed diff to trigger overtaking

CAR_LENGTH = 4 # [m]

FAR_AWAY_IN_FRONT = 200 # [m]  distance at which a car in front can be ignored
FAR_AWAY_IN_BACK = 80   # [m]  distance at which a car behind can be ignored
```

Lanes

```
def normaliseDirection(d):
    d = d.lower()
    if d=='r' or d=='fast':
        return 'fast'
    elif d=='l' or d=='slow':
        return 'slow'
    else:
        return None


LANE_ID = 0

class Lane:

    ## some additional code
    def __init__(self, length, speedLimit):

        global LANE_ID
        self.id = LANE_ID
        LANE_ID += 1

        self.length = length
        self.speedLimit = speedLimit
        self.vehicles = []

        self.next = None
        self.prev = None

        # lane attached to the left/right
        self.left = None
        self.right = None

    # defines generic str() method for Lanes
    # extends the method with list of vehicles on the lane
    def __str__(self):
        l = "" if self.left is None else f" L:{self.left.id:d}"
        r = "" if self.right is None else f" R:{self.right.id:d}"
        vs = "" if len(self.vehicles)==0 else " "
        for v in self.vehicles:
            vs += str(v)
        return f"[{self.id:d} {int(self.length):d}m"+l+r+vs+"]" + \
                    ("-" + str(self.next) if self.next is not None else "")

    def getLane(self, direction):
        if direction=='slow':
            return self.left
        elif direction=='fast':
            return self.right
        else:
            return None
    # adding parallel lane on right side
    def attachRight(self, lane):
        self.right = lane
        lane.left = self

    # adding parallel lane on right side
    def attachLeft(self, lane):
        self.left = lane
        lane.right = self

    # constructs a number of lane segments of the same length
    # and attaches them to the right
    def widenRight(self):
        lane = self
        newLane = Lane(lane.length, lane.speedLimit)
        lane.attachRight(newLane)
        while lane.next is not None:
            lane = lane.next
            newLane = Lane(lane.length, lane.speedLimit)
            lane.attachRight(newLane)
            newLane.prev = lane.prev.right
            newLane.prev.next = newLane
        return self.right

    # constructs a number of lane segments of the same length
    # and attaches them to the right
    def widenLeft(self):
        lane = self
        newLane = Lane(lane.length, lane.speedLimit)
        lane.attachLeft(newLane)
        while lane.next is not None:
            lane = lane.next
            newLane = Lane(lane.length, lane.speedLimit)
            lane.attachLeft(newLane)
            newLane.prev = lane.prev.left
            newLane.prev.next = newLane
        return self.left

    # defines concatenation of lanes
    def extend(self, lane):
        l = self
        while l.next is not None:
            l = l.next
        l.next = lane
        lane.prev = l
        return self

    def totalLength(self):
        total = self.length
        l = self
        while l.next is not None:
            l = l.next
            total += l.length
        return total

    ## additional code
    ## new generalised access method needed to calculate sideway view
    ## returns all vehicles between pos+distFrom and pos+distTo
    def at(self, pos, distFrom=-CAR_LENGTH/2, distTo=CAR_LENGTH/2):
        # make sure that the position of all cars is accurate
        # at this point in time
        for v in self.vehicles:
            v.updateOnly()
                # normally the list should be sorted, but just in case
        self.vehicles.sort(key=lambda v: v.pos)
        res = []
        for v in self.vehicles:
            if pos+distFrom < v.pos and v.pos < pos+distTo:
                res.append(v)
        # if the required distance reaches over the end of the lane segment
        if pos+distTo > self.length and self.next is not None:
            res = res + self.next.at(0, distFrom=0, distTo=distTo-(self.length-pos))
        if pos+distFrom < 0 and self.prev is not None:
            res = self.prev.at(self.prev.length, distFrom=pos+distFrom, distTo=0) + res
        return res

    def inFront(self, pos, far=FAR_AWAY_IN_FRONT):
        # make sure that the position of all cars is accurate
        # at this point in time
        for v in self.vehicles:
            v.updateOnly()
        # normally the list should be sorted, but just in case
        self.vehicles.sort(key=lambda v: v.pos)
        for v in self.vehicles:
            if v.pos > pos:
                return v if v.pos-pos<far else None
        # there is none in front in this lance
        # if the free lane in front is long enough or there is no next lane
        if self.length-pos>far or self.next is None:
            return None
        else:
            return self.next.inFront(0, far=far-(self.length-pos))

    def behind(self, pos, far=FAR_AWAY_IN_BACK):
        # make sure that the position of all cars is accurate
        # at this point in time
        for v in self.vehicles:
            v.updateOnly()
        # This time we sort in reverse order
        self.vehicles.sort(key=lambda v: v.pos, reverse=True)
        for v in self.vehicles:
            if v.pos < pos:
                return v if pos-v.pos<far else None
        # there is none behind in this lance
        # if the free lane in behind is long enough or there is no previous lane
        if pos>far or self.prev is None:
            return None
        else:
            return self.prev.behind(self.prev.length, far=far-pos)

    def enter(self, vehicle, pos=0):
        self.vehicles.insert(0, vehicle)
        vehicle.pos = pos
        vehicle.lane = self
        vehicle.rec.record(vehicle, event="enter lane")

    def leave(self, vehicle):
        vehicle.rec.record(vehicle, event="leave lane")
        vehicle.lane = None
        # in the meantime the vehicle may have have moved
        # to one of the next lane segments...
        lane = self
        while lane is not None:
            if vehicle in lane.vehicles:
                lane.vehicles.remove(vehicle)
                break
            else:
                lane = lane.next
```

Vehicles

[ ] ↳ *4 cells hidden*

Recorder

[ ] ↳ *2 cells hidden*

Multiple Vehicles with Fixed Speed

```
VMAX=120/3.6
#Creating Lanes
l = Lane(3000, VMAX)
r = l.widenRight()
print("Left Lane: ", l)
print("Right Lane:", r)
```

```
Left Lane:  [2 3000m R:3]
```

```
     Right Lane: [3 3000m L:2]
```

```
a = [50, 80 , 100, 120]     #Creating different maximum speed profiles depending on the motor
for x in range(len(a)):
  a[x] = a[x]/3.6

avg_time=[]
through_put=[]


VMAX = a[1]
N = 300
DT = 4 # time difference between start
env = simpy.Environment()
rec = SimpleRecorder(env, 0, 1500, 1)
l = Lane(3000, VMAX)
r = l.widenRight()
for i in range(N):
  v = Vehicle(env, rec, startingLane=l, t0=i*DT,dx0=VMAX)
  #v.traceOvertake = True
rec.run()
```

```
df = rec.getData()
start, end = {},{}
time_taken = []
for i in range(len(df)):
  if df["id"][i] not in start:
    start[df["id"][i]] = df["t"][i]
    end[df["id"][i]] = df["t"][i]
  end[df["id"][i]] = df["t"][i]

for i in start:
  time_taken.append(end[i]-start[i])
  #print(start)
  #print(end)
average = sum(time_taken) / len(time_taken)
print(f"The Average time for speed limit {VMAX} kmph is {average} seconds")
```

```
     The Average time for speed limit 13.88888888888889 kmph is 216.0 seconds
```

```
#Throughputs calculation
event = "end"
for index, row in df.iloc[::-1].iterrows():
  if df["event"][index]=="end":
    number = df["id"][index]
    time = df["t"][index]
    break
```

```
#print(number)
#print(time)
factor = 3600/time
throughput = number*factor
print(f"and the Throughput is {throughput} cars per hour")
```

```
and the Throughput is 762.3229461756374 cars per hour
```

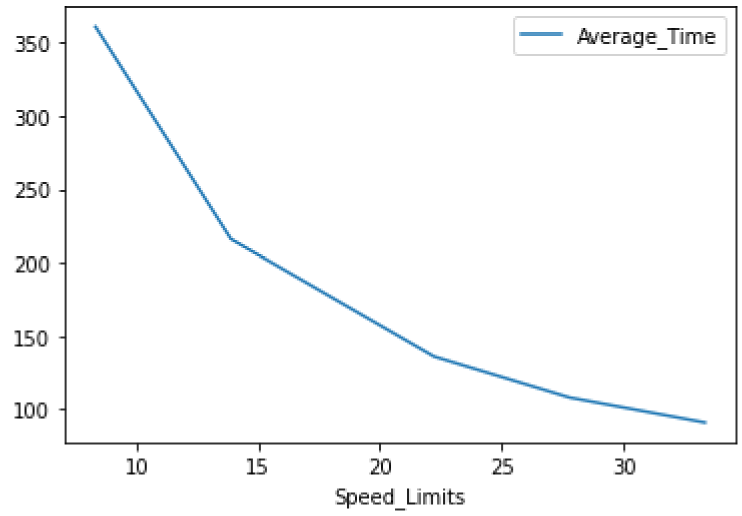
```
avg_time=[360.92, 216.00, 136.00, 108.00, 90.92]
through_put=[156.52, 234.25, 322.64, 371.74, 409.58]
```

```
Result1 = pd.DataFrame(list(zip(a, avg_time,through_put)),columns =['Speed_Limits', 'Average_
Result1
```

	Speed_Limits	Average_Time	Throughput
0	8.333333	360.92	156.52
1	13.888889	216.00	234.25
2	22.222222	136.00	322.64
3	27.777778	108.00	371.74
4	33.333333	90.92	409.58

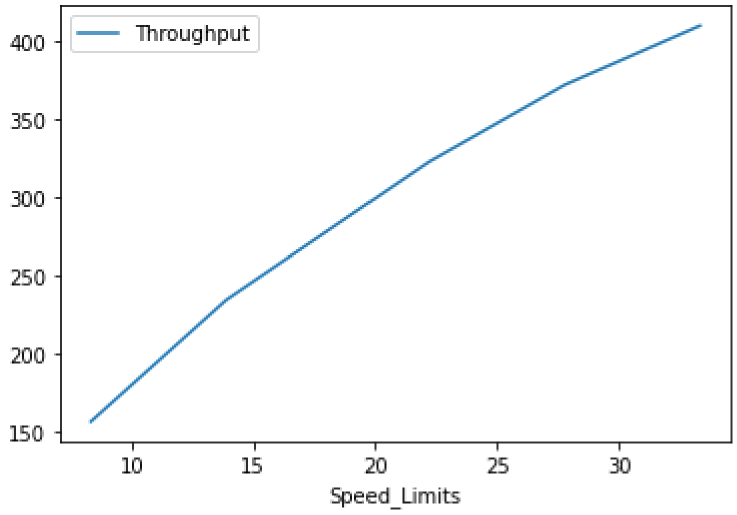
```
Result1.plot(x ='Speed_Limits', y='Average_Time', kind = 'line')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58c525a8d0>
```



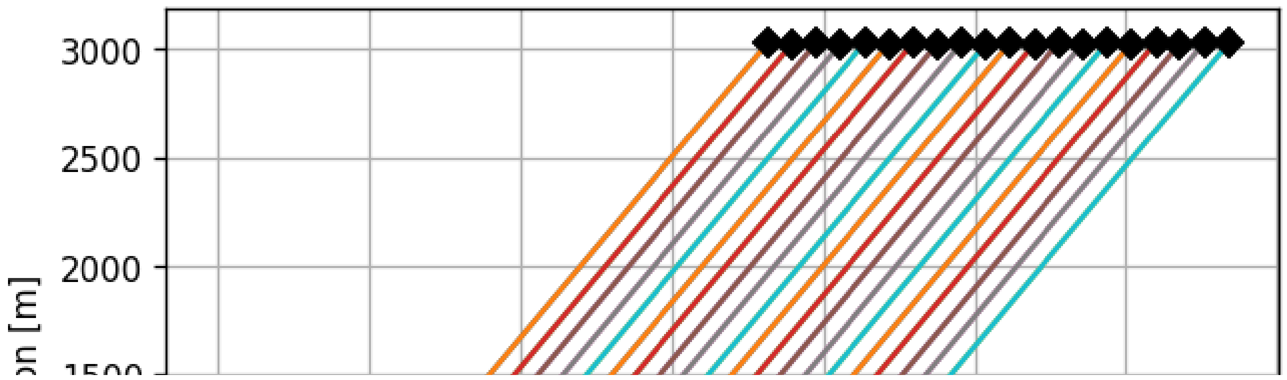
```
Result1.plot(x ='Speed_Limits', y='Throughput', kind = 'line')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58c63e03d0>
```

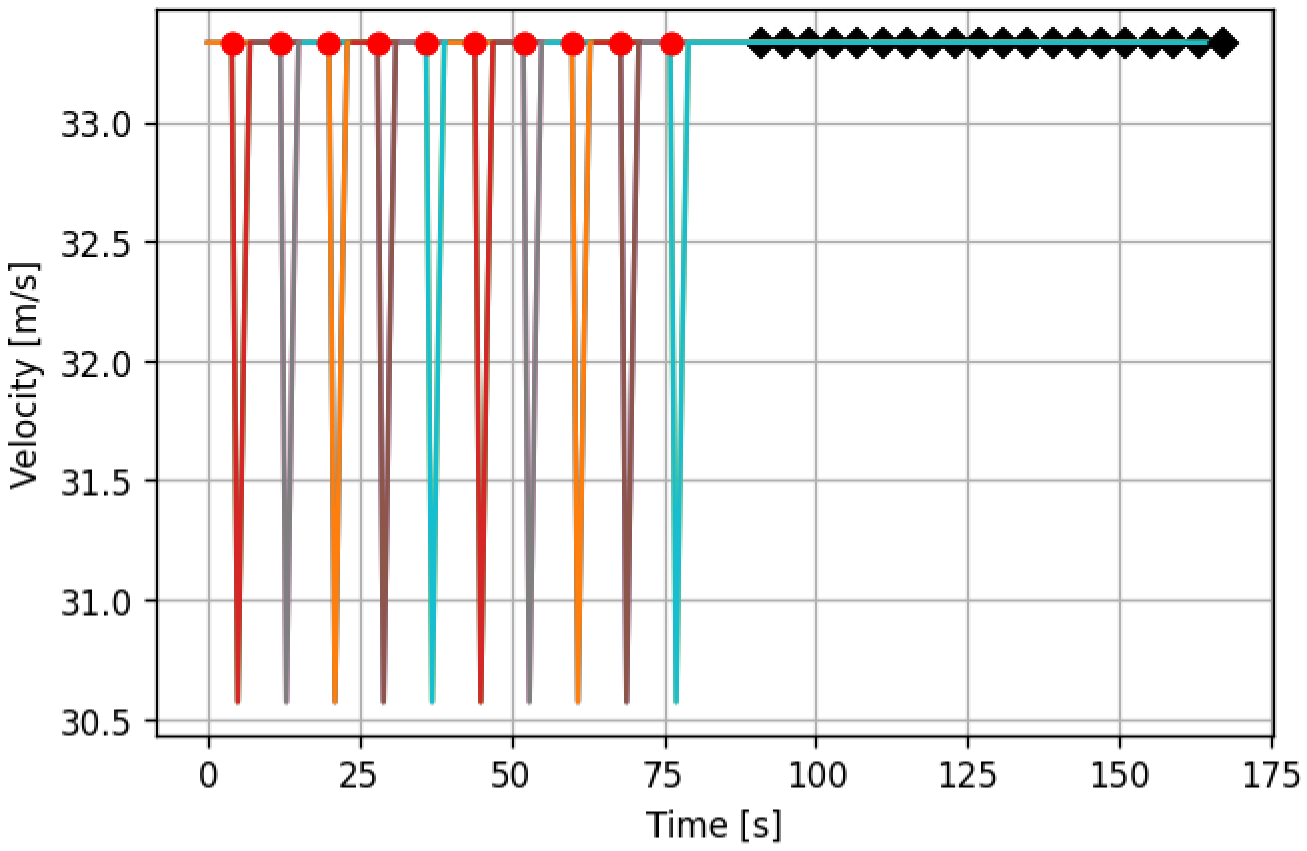


```
VMAX = 120/3.6
N = 20
DT = 4 # time difference between start
env = simpy.Environment()
rec = SimpleRecorder(env, 0, 500, 1)

l = Lane(3000, VMAX)
r = l.widenRight()
for i in range(N):
  v = Vehicle(env, rec, startingLane=l, t0=i*DT,dx0=VMAX)
  v = Vehicle(env, rec, startingLane=r, t0=i*DT,dx0=VMAX)
  #v.traceOvertake = True
rec.run()
  #Average time calculation
df1 = rec.getData()
```

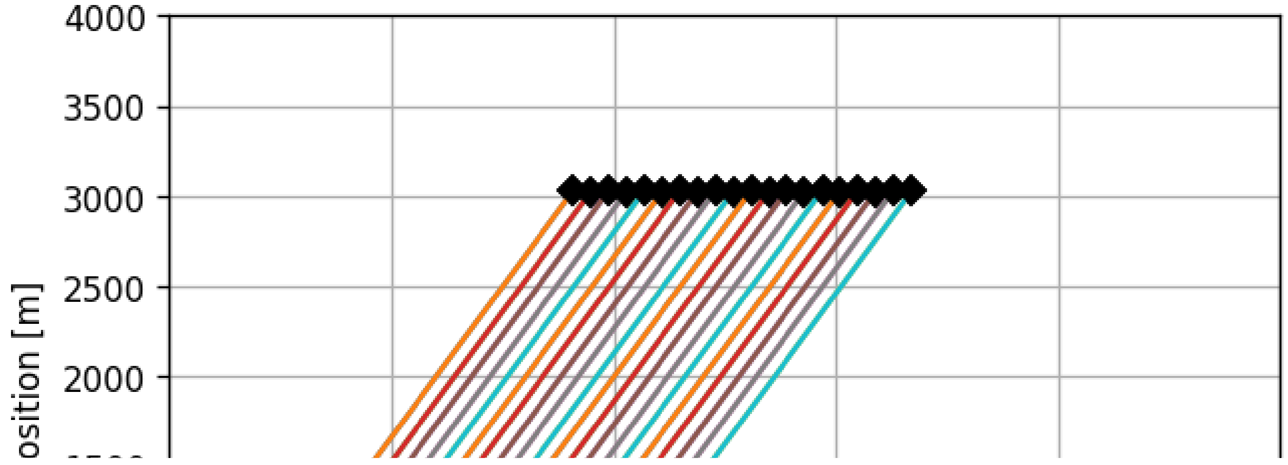
```
rec.plot('t', 'x')
```



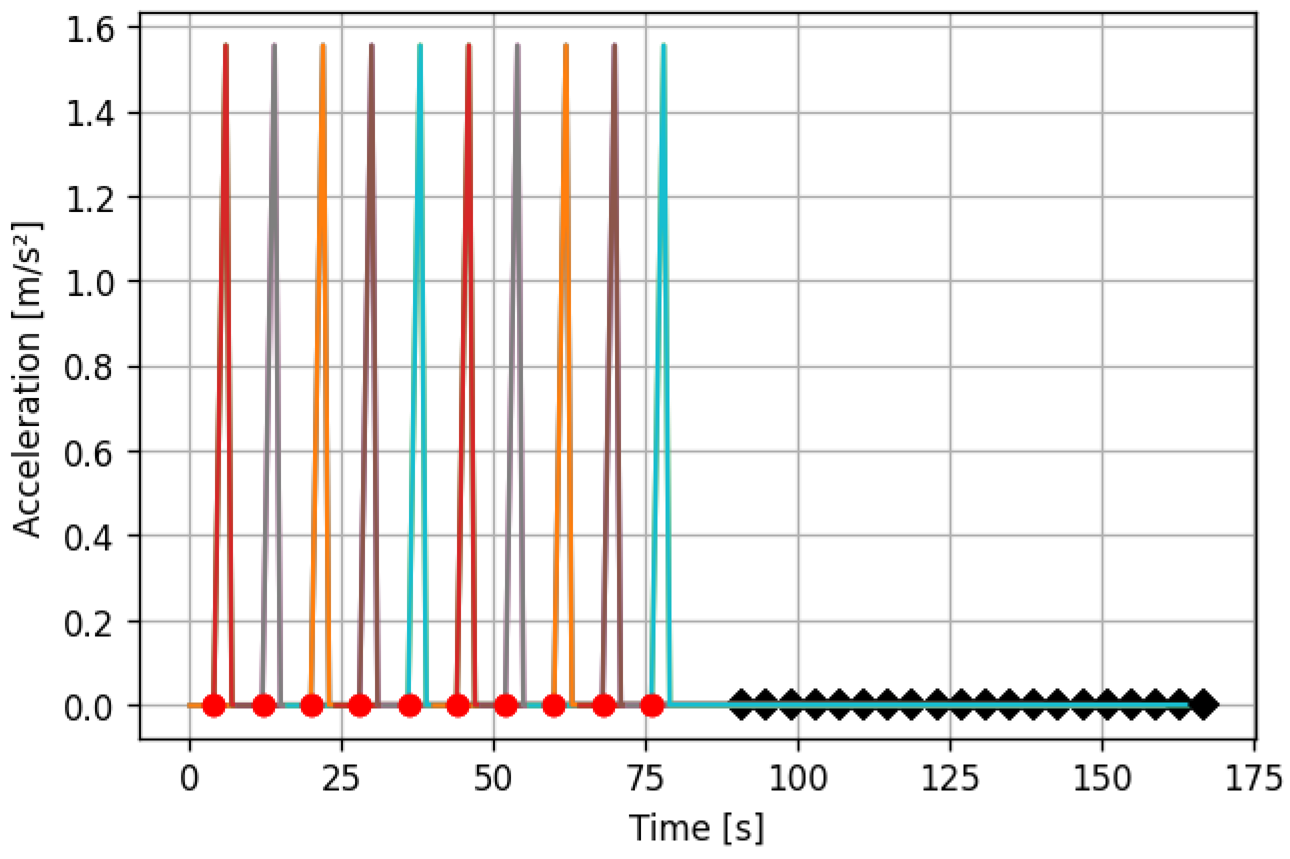
```
rec.plot('t', 'v')
```



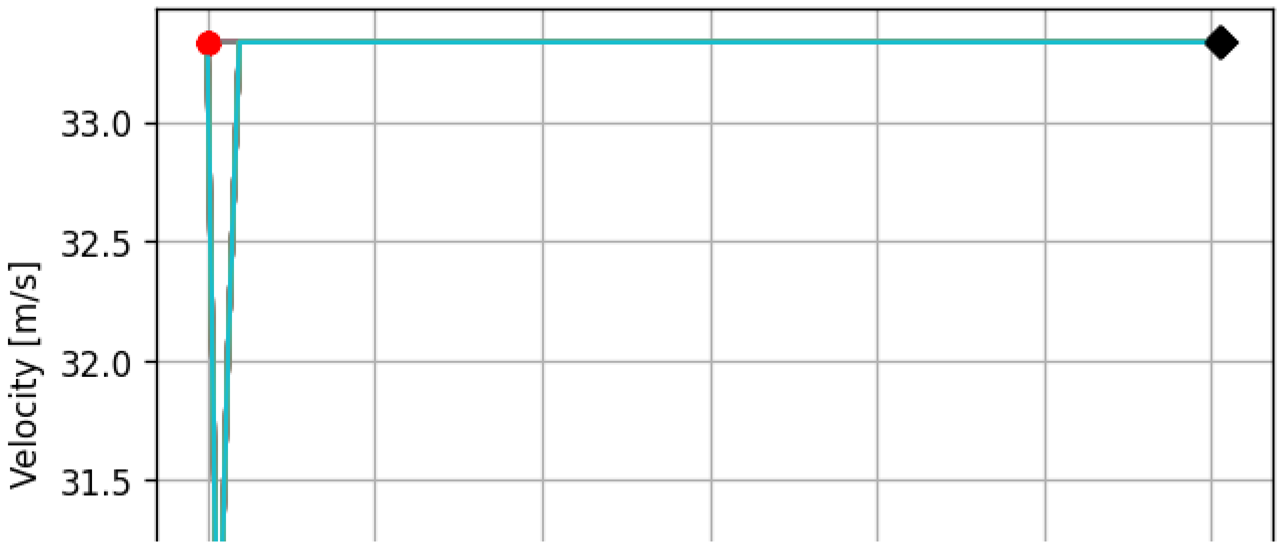
```
rec.plot('t', 'x', xmin=0, xmax=250, ymin=0, ymax=4000)
```



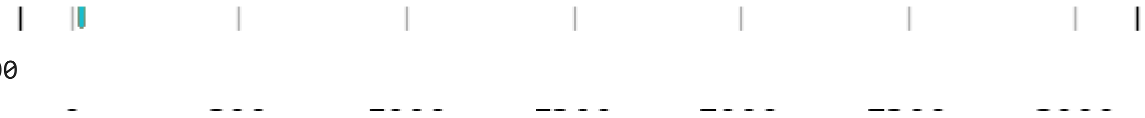
```
rec.plot('t', 'a')
```



```
rec.plot('x', 'v')
```



Generating Random Speeds



```
SLOW_CYCLE = 100

def randomIntervals(cycles):
    # return [ random.expovariate(1.0/SLOW_CYCLE)+10 for i in range(cycles) ]
    return [ max(0, random.normalvariate(SLOW_CYCLE, SLOW_CYCLE/3)) for i in range(cycles) ]


SPEED_VARIATION = 0.05
def randomSpeedVariation(vmax, cycles, cv=SPEED_VARIATION):
    return [ vmax + (-1)**i*abs(random.normalvariate(0, vmax*cv)) for i in range(cycles) ]
```

Simulating Cars on Lanes

```
#Creating Lanes
VMAX= 120/3.6
l = Lane(3000, VMAX)
r = l.widenRight()
print("Left Lane: ", l)
print("Right Lane:", r)
```

```
     Left Lane:  [0 3000m R:1]
     Right Lane: [1 3000m L:0]
```

```
VMAX = 120/3.6
N = 80 # number of points
DT =  4# time difference between start
random.seed(13)
env = simpy.Environment()
```

```
rec = SimpleRecorder(env, 0, 1000, 1)

for i in range(N):
    CYCLES = random.randint(1, 3)
    times = randomIntervals(CYCLES)
    #speed = randomSpeedVariation(VMAX+i, CYCLES)
    speed = randomSpeedVariation(VMAX, CYCLES)
    v = Vehicle(env, rec, startingLane=l, t0=i*DT, dx0=speed[-1], t=times, v=speed)
    #v = Vehicle(env, rec, startingLane=r, t0=i*DT, dx0=speed[-1], t=times, v=speed)
    v.traceOvertake = True
    #v.traceSurround = True
rec.run()
```

```
t=  115.0s Overtaking v21 overtakes v20 at x=  805.4m
t=  160.0s Overtaking v24 overtakes v23 at x=1,747.7m
t=  160.0s Overtaking v29 overtakes v28 at x=1,080.9m
t=  181.0s Overtaking v25 overtakes v23 at x=2,263.4m
t=  201.0s Overtaking v25 returns to slow lane at x=2,930.7m
t=  221.0s Overtaking v31 overtakes v30 at x=2,542.2m
t=  226.0s Overtaking v52 overtakes v51 at x=  293.0m
t=  232.0s Overtaking v31 returns to slow lane at x=2,905.7m
t=  241.0s Overtaking v34 overtakes v33 at x=2,550.9m
t=  247.0s Overtaking v45 overtakes v44 at x=1,251.9m
t=  253.0s Overtaking v34 returns to slow lane at x=2,943.8m
t=  290.0s Overtaking v58 overtakes v57 at x=  854.5m
t=  307.0s Overtaking v74 overtakes v73 at x=  152.7m
Crash p76 into p75 at t=311.000 x=  168.0
t=  313.0s Overtaking v47 overtakes v46 at x=2,682.2m
Crash p77 into p76 at t=316.000 x=  164.5
t=  319.0s Overtaking v47 returns to slow lane at x=2,881.5m
Crash p78 into p77 at t=320.000 x=  162.3
t=  331.0s Overtaking v58 returns to slow lane at x=1,797.9m
t=  332.0s Overtaking v52 returns to slow lane at x=2,977.5m
Crash p79 into p78 at t=338.000 x=  158.7
t=  339.0s Overtaking v60 overtakes v59 at x=1,752.4m
t=  351.0s Overtaking v61 overtakes v59 at x=2,001.2m
t=  363.0s Overtaking v60 returns to slow lane at x=2,482.2m
t=  376.0s Overtaking v72 overtakes v71 at x=1,057.3m
t=  411.0s Overtaking v66 overtakes v65 at x=2,766.1m
t=  419.0s Overtaking v69 overtakes v68 at x=2,503.5m
t=  419.0s Overtaking v74 returns to slow lane at x=2,494.3m
t=  422.0s Overtaking v74 overtakes v68 at x=2,592.4m
```

```
df1=rec.getData()
```

```
df1.tail(30)
```

```
#df1[df1["id"]==94].head(40)
```

Average Time

```
start, end = {},{}
time_taken = []
for i in range(len(df1)):
    if df1["id"][i] not in start:
        start[df1["id"][i]] = df1["t"][i]
        end[df1["id"][i]] = df1["t"][i]
    end[df1["id"][i]] = df1["t"][i]

for i in start:
    time_taken.append(end[i]-start[i])
print(start)
print(end)
average = sum(time_taken) / len(time_taken)
print(average)
```

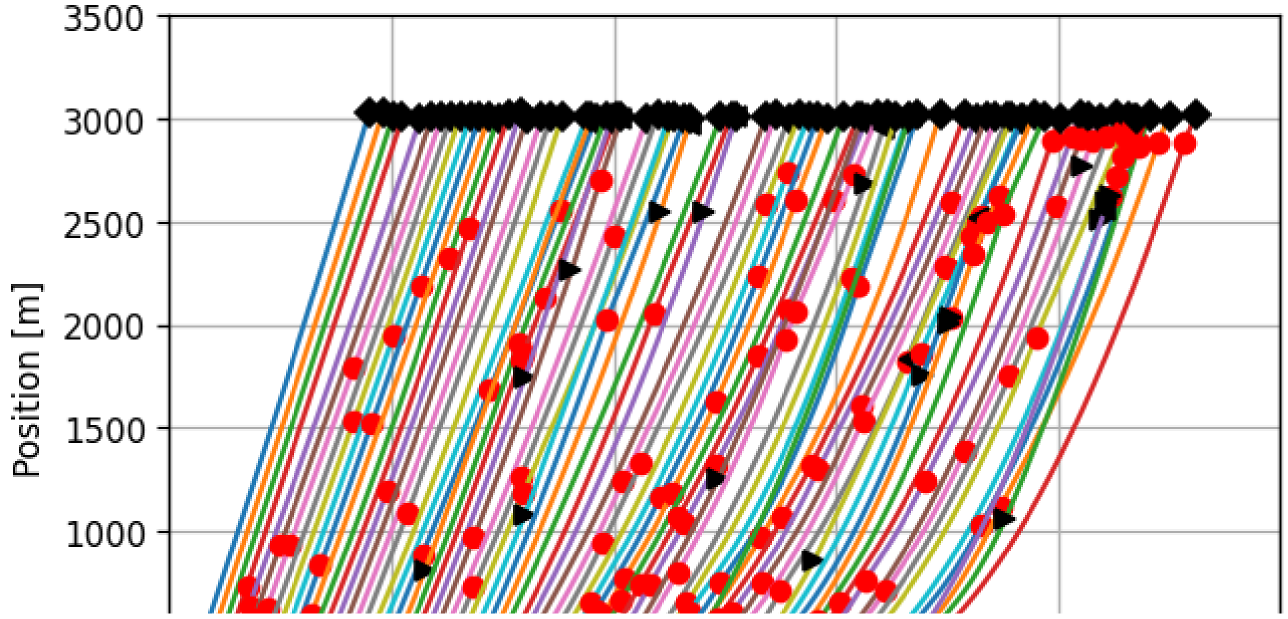
```
{0: 0, 1: 4, 2: 8, 3: 12, 4: 16, 5: 20, 6: 24, 7: 28, 8: 32, 9: 36, 10: 40, 11: 44, 12:
{0: 89.85736281659443, 1: 96, 2: 100.44715000000001, 3: 104.8472, 4: 112.9, 5: 118.1745
118.25170282198505
```

Throughput

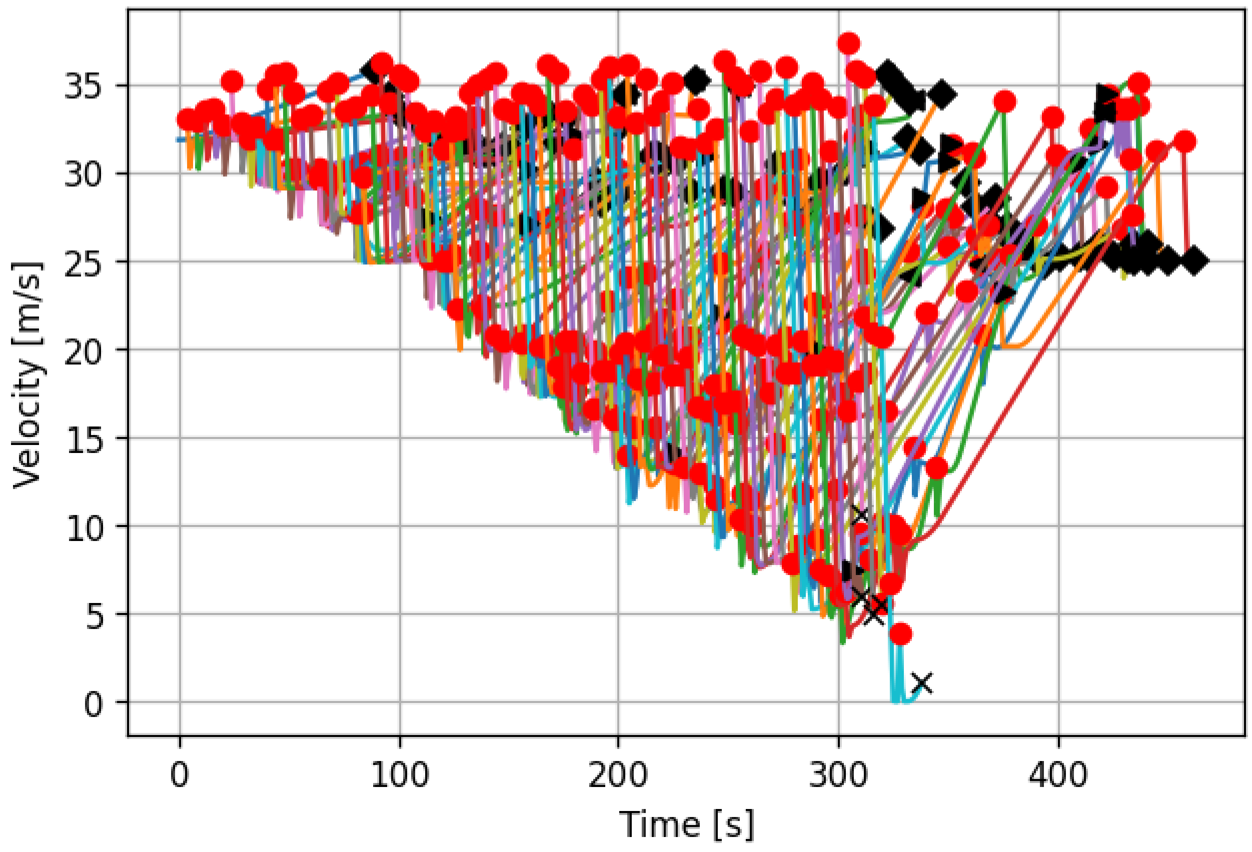
```
event = "end"
for index, row in df1.iloc[::-1].iterrows():
    if df1["event"][index]=="end":
        number = df1["id"][index]
        time = df1["t"][index]
        break

#print(number)
#print(time)
factor = 3600/time
#print(factor)
throughput = number*factor
print(f"The Throughput is {throughput} cars per hour")
```

```
The Throughput is 568.8311688311688 cars per hour
```

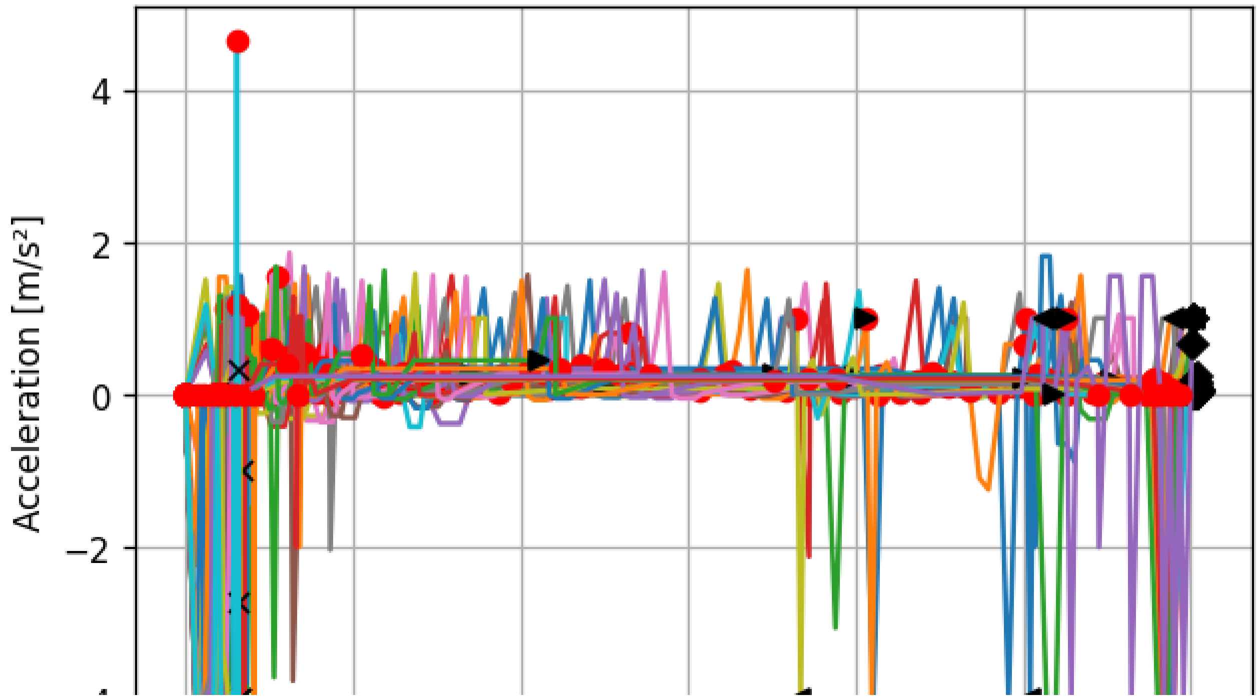
```
rec.plot('t', 'x', xmin=0, xmax=500, ymin=0, ymax=3500)
```



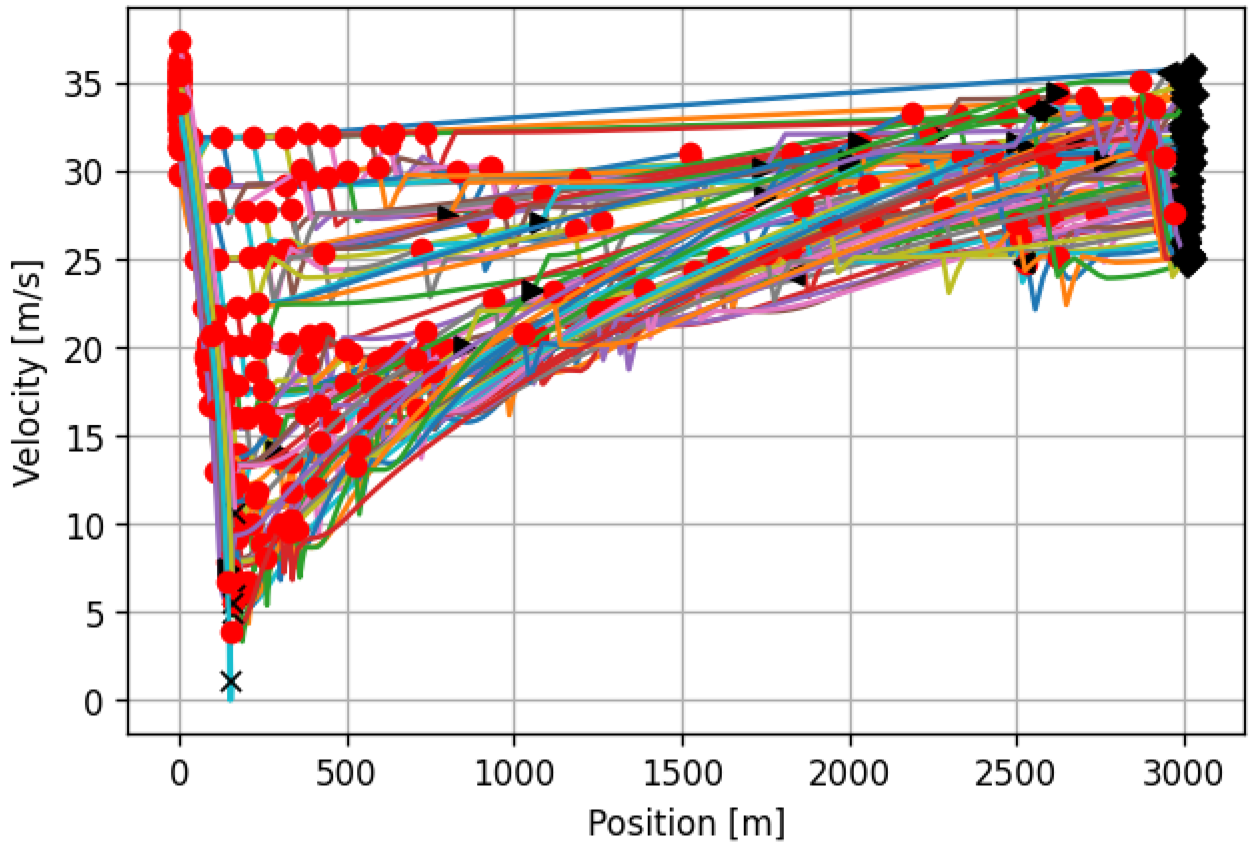
```
rec.plot('t', 'v')
```



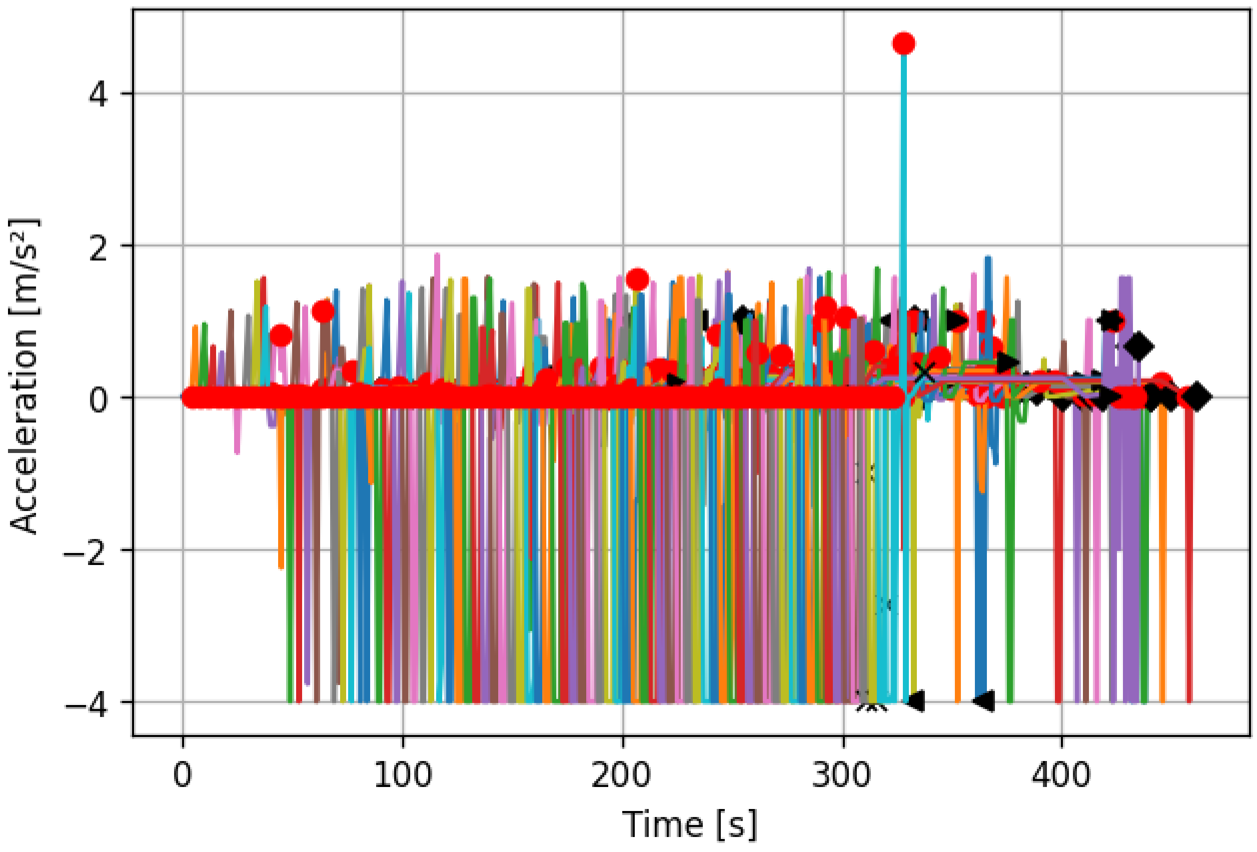
```
rec.plot('x', 'a')
```



```
rec.plot('x', 'v')
```



```
rec.plot('t', 'a')
```



Executing (5m 5s) > > > ! > ... > ... > (> r(> _... > _seti... > _setite... > a| > c > ... > concat... > _con... > con...