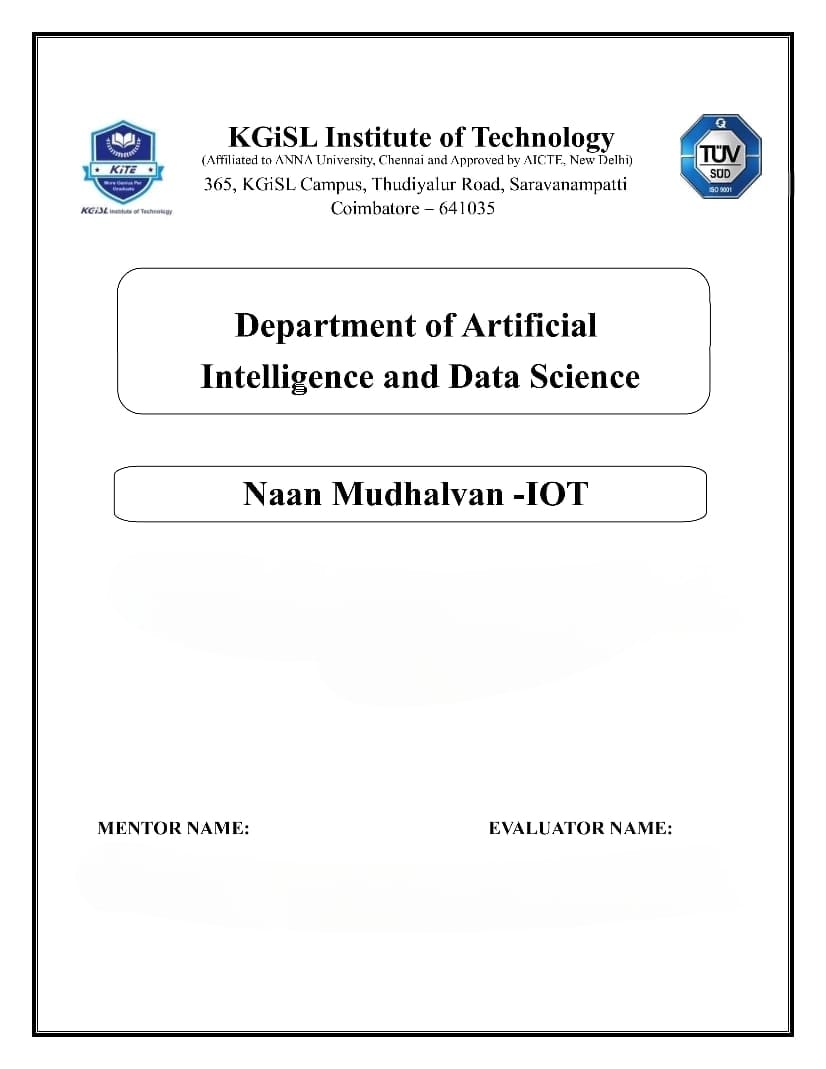
****

**PROBLEM STATEMENT : PUBLIC TRANSPORT OPTIMIZATION**

Mr.MOHANKUMAR.M

Ms.AKILANDESWARI.M

**REQUIRED COMPONENTS :**

**1. GPS Sensors:**



**2. Traffic Flow Sensors:**



**3. Ticket Validation Sensors:**



**REQUIREMENT:**

confluent-kafka[avro]==1.1.0

faust==1.7.4

tornado==6.0.3

**SOURCE CODE :**

import logging

import confluent\_kafka

from confluent\_kafka import Consumer

from confluent\_kafka.avro import AvroConsumer

from confluent\_kafka.avro.serializer import SerializerError

from tornado import gen

logger = logging.getLogger(\_\_name\_\_)

class KafkaConsumer:

"""Defines the base kafka consumer class"""

def \_\_init\_\_(

self,

topic\_name\_pattern,

message\_handler,

is\_avro=True,

offset\_earliest=False,

sleep\_secs=1.0,

consume\_timeout=0.1,

):

"""Creates a consumer object for asynchronous use"""

self.topic\_name\_pattern = topic\_name\_pattern

self.message\_handler = message\_handler

self.sleep\_secs = sleep\_secs

self.consume\_timeout = consume\_timeout

self.offset\_earliest = offset\_earliest

self.broker\_properties = {

'bootstrap.servers': 'PLAINTEXT://localhost:9094',

'default.topic.config': {'auto.offset.reset': 'earliest'},

'group.id': topic\_name\_pattern

}

if is\_avro is True:

self.broker\_properties["schema.registry.url"] = "http://localhost:8081"

self.consumer = AvroConsumer(self.broker\_properties)

else:

self.consumer = Consumer(self.broker\_properties)

pass

self.consumer.subscribe([topic\_name\_pattern],on\_assign=self.on\_assign)

def on\_assign(self, consumer, partitions):

"""Callback for when topic assignment takes place"""

for p in partitions:

consumer.seek(p)

logger.info(f"partitions assigned for {self.topic\_name\_pattern}")

consumer.assign(partitions)

async def consume(self):

"""Asynchronously consumes data from kafka topic"""

while True:

num\_results = 1

while num\_results > 0:

num\_results = self.\_consume()

await gen.sleep(self.sleep\_secs)

def \_consume(self):

"""Polls for a message. Returns 1 if a message was received, 0 otherwise"""

try:

msg = self.consumer.poll(timeout=1.0)

if msg is not None:

if msg.error() is not None:

self.message\_handler(msg)

return 1

else:

logger.error(msg.error())

return 0

else:

logger.debug("no message")

return 0

except SerializerError as error:

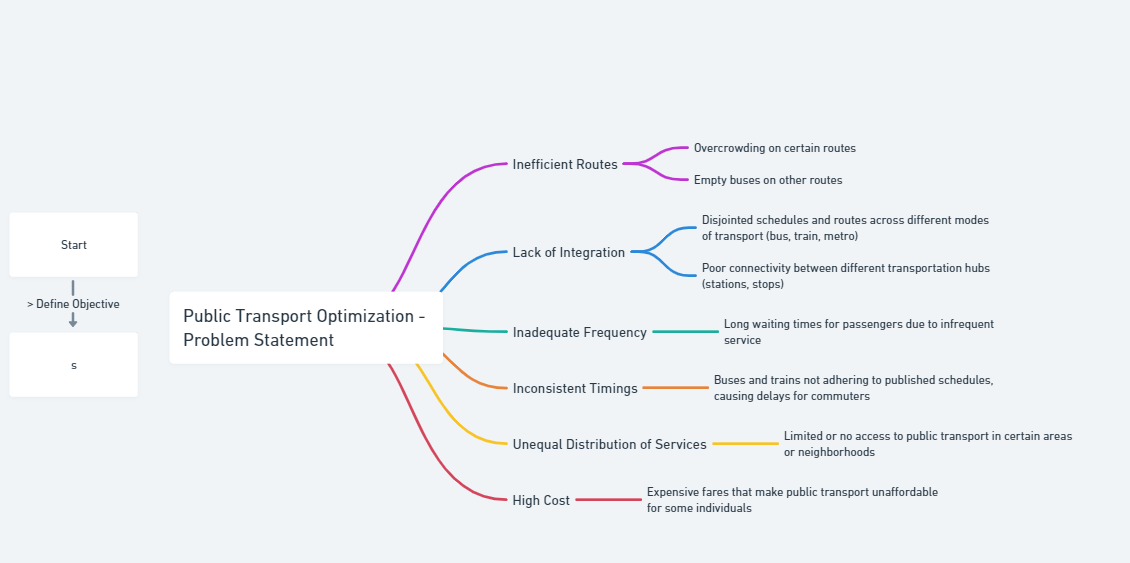
logger.error(f"Error consuming data: {error.message}")

return 0

def close(self):

self.consumer.close()

**This flowchart provides an overview of the entire system, from data collection to user interaction, to address the lack of accessible real-time air quality data in your region.**

****