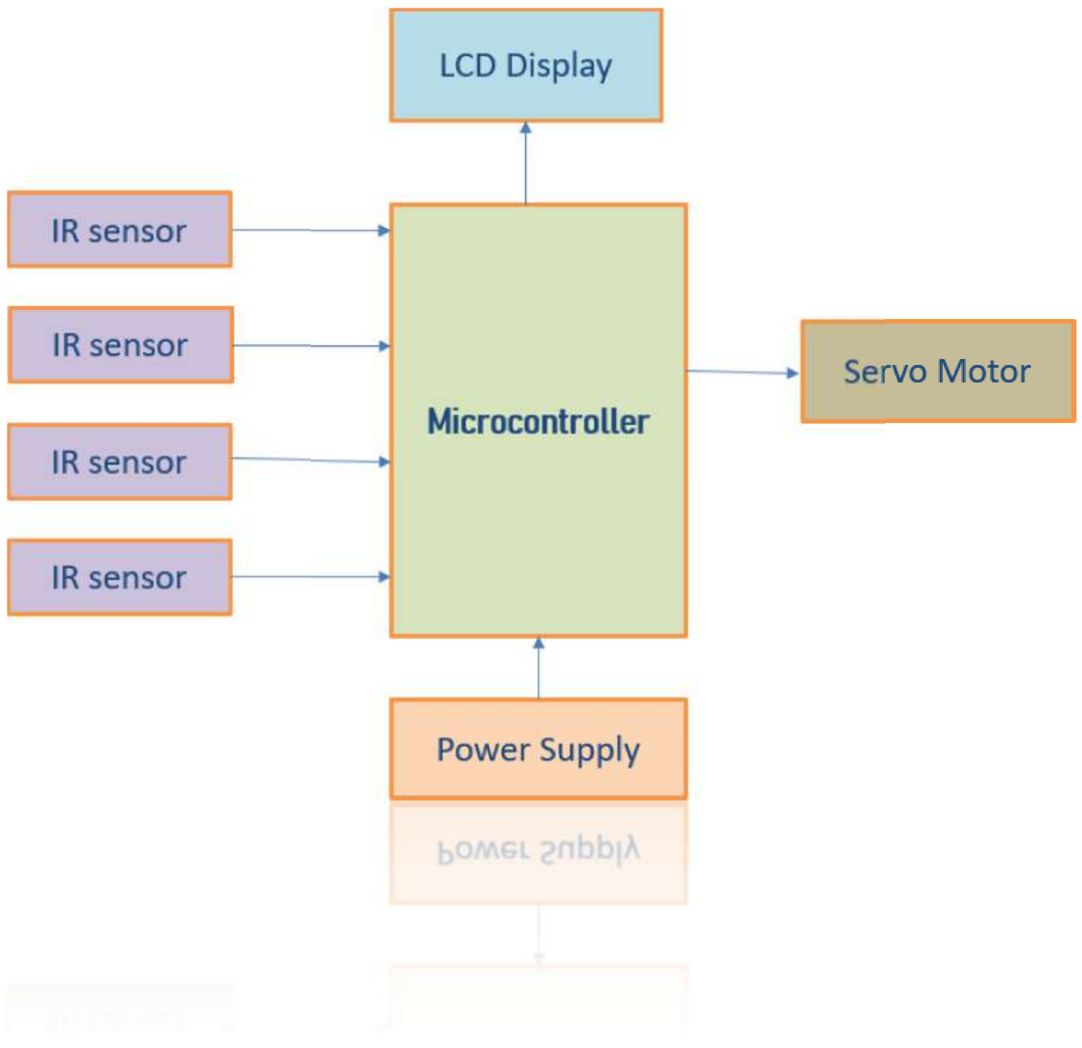
SMART PARKING-IOT

Phase 3: Development Part 1

Introduction:

Smart parking systems use IoT devices and sensors to collect real-time data on parking lot occupancy and transmit this information to the cloud or local network. The data is then periodically sent to websites and mobile applications in real time. The traditional parking system is poorly managed which has a substantial negative impact on cities. poorly managed parking increases traffic congestion, pollution ,and frustrates drivers .however iot offers a uniqus method of smart parking that can be incorporated into any parking management system.



Introduction to IoT Smart Parking with Python:

An IoT(Internet of Things) smart parking system is designed to make parking management more efficient by leavraging sensors and technology to monitor and manage parking spaces

Development steps:

To connect and install sensors in each parking space. These sensors will detect the presence of a vehicle and send data to the microcontroller.

To write python code on the microcontroller to read sensor data and send it to the cloud platform. You'll need libraries for data communication.

Set up a cloud server or use a platform like AWS IoT or Azure IoT hub to receive and process data from the microcontroller. You'll need to create a database to store parking space status.

Store information about each parking space's occupancy status and location in a database.You can use a database service like amazon dynamoDB or google cloud firestore.

Develop a web or mobile app using python frameworks like flash or django to display parking space availability in real time.

Design the system to scale easily as you add more parking spaces and sensors.

Test the system thoroughly, and then deploy it in your parking facility.

Loading and preprocessing datasets in smart parking system:

To load and preprocess datasets in a smart parking system using IoT with a Python script, you can follow these steps:

1. Data Collection with IoT Devices:

- Deploy IoT devices (e.g., sensors, cameras) to collect parking data.

- Ensure these devices send data to a central location. Use appropriate communication protocols such as MQTT or HTTP.

2. Data Storage:

- Choose a database to store the collected data. For this example, we'll use SQLite, a lightweight database.

```
python
import sqlite3
# Create a SQLite database or connect to an existing one
conn = sqlite3.connect("parking_data.db")
cursor = conn.cursor()
# Create a table to store parking data
cursor.execute('''CREATE TABLE IF NOT EXISTS parking_data (
  timestamp TIMESTAMP,
  location TEXT,
  occupancy INT )''')
```

```
# Save the changes and close the connection
conn.commit()
conn.close()
```

3. Data Preprocessing:

- Write Python functions to preprocess the incoming data. This may involve cleaning, normalizing, and converting data into a suitable format. For example, parsing data from IoT devices and inserting it into the database:

```
python
import datetime
def preprocess_and_store_data(location, occupancy):
    conn = sqlite3.connect("parking_data.db")
    cursor = conn.cursor()
 # Get the current timestamp
 timestamp = datetime.datetime.now()
 # Insert data into the database
cursor.execute("INSERT INTO parking_data (timestamp, location, occupancy) VALUES (?, ?, ?)", (timestamp, location, occupancy))
 # Save the changes and close the connection
conn.commit()
conn.close()
```

4. Real-time Processing:

- Depending on your requirements, you may need real-time processing to monitor parking availability and update user interfaces. This might involve implementing a WebSocket or REST API to serve real-time data.

5.Data Retrieval:

- You can retrieve and analyze the data from the SQLite database when needed. For example, to get the latest occupancy data:

```
python
def get_latest_occupancy(location):
conn = sqlite3.connect("parking_data.db")
cursor = conn.cursor()
cursor.execute("SELECT occupancy FROM parking_data WHERE location = ? ORDER BY timestamp DESC LIMIT 1", (location,))
result = cursor.fetchone()
 conn.close()
 return result[0] if result else None
```

6. Visualization and Analytics:

- Use Python libraries such as Matplotlib, Plotly, or pandas for data visualization and analytics.

7.Alerts and Notifications:

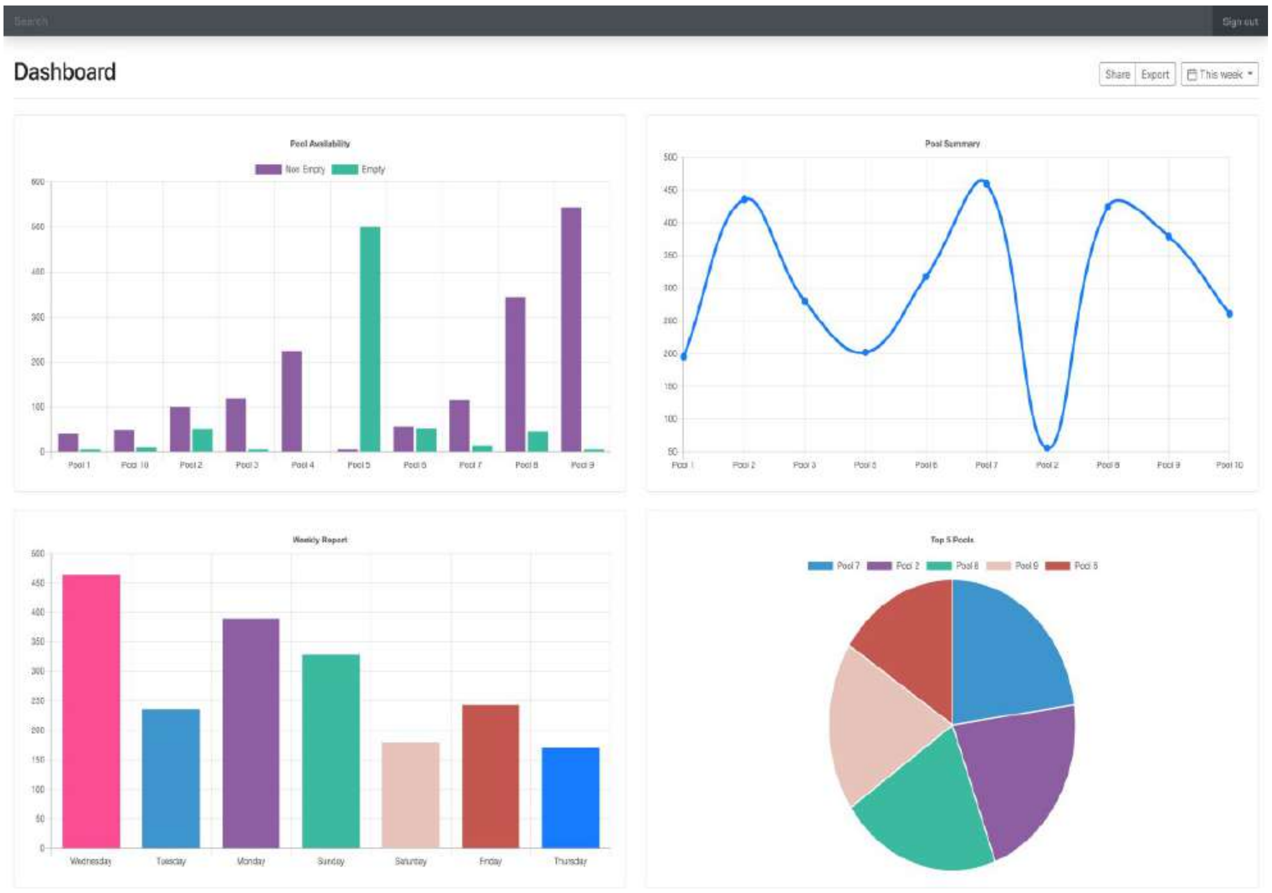
- Implement alerting systems using email, SMS, or other notification mechanisms based on your analysis.

8.Security:

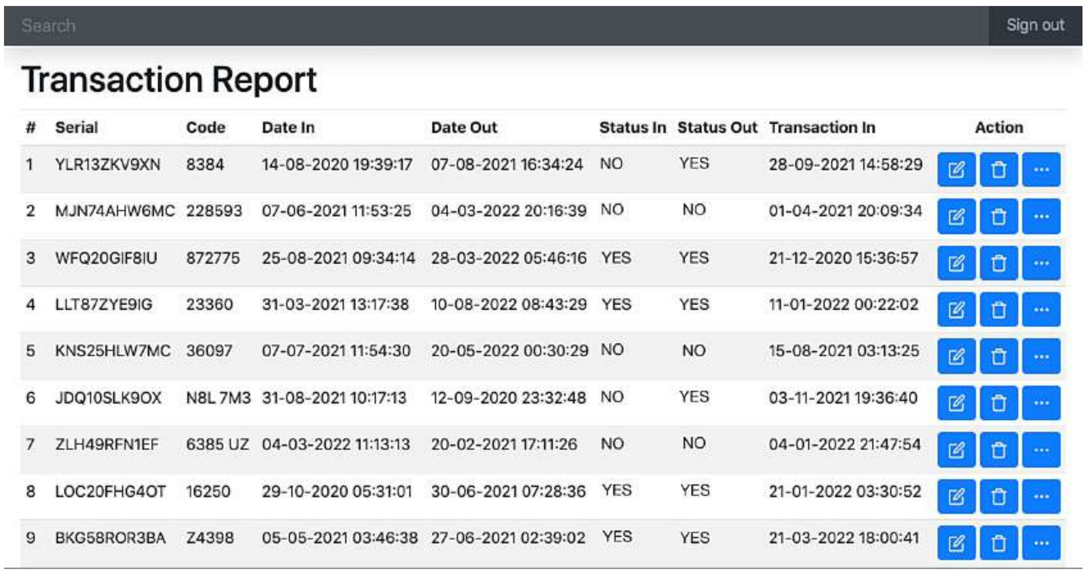
- Ensure data security and user authentication, especially if you're exposing the data to external users.

These are the foundational steps to get started with loading and preprocessing datasets for a smart parking system using Python. You can expand upon this foundation to build more advanced features and integrations as needed for your specific project.

Dashboard Interface:



Transaction Interface:



Transaction Report

#	Serial	Code	Date In	Date Out	Status In	Status Out	Transaction In	Action
1	YLR13ZKV9XN	8384	14-08-2020 19:39:17	07-08-2021 16:34:24	NO	YES	28-09-2021 14:58:29	✎ 🗑 ⋯
2	MJN74AHW6MC	228593	07-06-2021 11:53:25	04-03-2022 20:16:39	NO	NO	01-04-2021 20:09:34	✎ 🗑 ⋯
3	WFQ20GIF8IU	872775	25-08-2021 09:34:14	28-03-2022 05:46:16	YES	YES	21-12-2020 15:36:57	✎ 🗑 ⋯
4	LLT87ZYE9IG	23360	31-03-2021 13:17:38	10-08-2022 08:43:29	YES	YES	11-01-2022 00:22:02	✎ 🗑 ⋯
5	KNS25HLW7MC	36097	07-07-2021 11:54:30	20-05-2022 00:30:29	NO	NO	15-08-2021 03:13:25	✎ 🗑 ⋯
6	JDQ10SLK9OX	N8L 7M3	31-08-2021 10:17:13	12-09-2020 23:32:48	NO	YES	03-11-2021 19:36:40	✎ 🗑 ⋯
7	ZLH49RFN1EF	6385 UZ	04-03-2022 11:13:13	20-02-2021 17:11:26	NO	NO	04-01-2022 21:47:54	✎ 🗑 ⋯
8	LOC20FHG4OT	16250	29-10-2020 05:31:01	30-06-2021 07:28:36	YES	YES	21-01-2022 03:30:52	✎ 🗑 ⋯
9	BKG58ROR3BA	Z4398	05-05-2021 03:46:38	27-06-2021 02:39:02	YES	YES	21-03-2022 18:00:41	✎ 🗑 ⋯

Program:

```
import RPi.GPIO as GPIO

import time

# Set up GPIO pins for sensors

TRIG = 23

ECHO = 24

GPIO.setmode(GPIO.BCM)

GPIO.setup(TRIG, GPIO.OUT)

GPIO.setup(ECHO, GPIO.IN)

# Function to measure distance

def measure_distance():

GPIO.output(TRIG, True)

time.sleep(0.00001)

GPIO.output(TRIG, False)

while GPIO.input(ECHO) == 0:

pulse_start = time.time()

while GPIO.input(ECHO) == 1:

pulse_end = time.time()

pulse_duration = pulse_end - pulse_start

distance = pulse_duration * 17150

return distance

try:

while True:

distance = measure_distance()
```

```
if distance < 10:  # Adjust this threshold as needed
# Vehicle detected in the parking space
# Send data to IoT platform
pass
time.sleep(1)
except KeyboardInterrupt:
GPIO.cleanup()
```

Output:

The program will output a message if a vehicle is detected in the parking space.

```
if distance < 10:  # Adjust this threshold as needed
   # Vehicle detected in the parking space
   # You can add an output message here
   Pass
```

Conclusion:

The implementation of a Smart parking system using IoT technology offers numerous advantages, such as efficient space utilization, reduced traffic congestion, and enhanced user convenience. By integrating sensors, connectivity and data analytics, this system provides real time parking information to both drivers and operators, leading to improved urban mobility and environmental sustainability.