**1. Importing Necessary Libraries**

|  |
| --- |
| import cv2  import imutils  import pytesseract  import winsound  import pandas as pd  from datetime import datetime, timedelta  import os  from openpyxl import load\_workbook  import matplotlib.pyplot as plt |

* cv2: OpenCV library for image processing.
* imutils: Helper functions for OpenCV, like resizing images.
* pytesseract: OCR (Optical Character Recognition) tool to extract text from images.
* winsound: To play sounds on Windows (e.g., a beep sound when a number plate is detected).
* pandas: For data manipulation and analysis, especially for handling Excel files.
* datetime: For date and time manipulation.
* os: For interacting with the operating system, such as file paths.
* openpyxl: For reading/writing Excel files.
* matplotlib.pyplot: For plotting graphs.

**2. Tesseract OCR Path Configuration**

|  |
| --- |
| # Set the Tesseract OCR executable path  pytesseract.pytesseract.tesseract\_cmd = r"C:\Program Files\Tesseract-OCR\tesseract.exe" |

Sets the path to the Tesseract executable, which is necessary for OCR functionality.

### 3. Helper Functions

#### Check if a String Exists in a File

|  |
| --- |
| # Function to check if a string exists in a file  def check\_if\_string\_in\_file(file\_name, string\_to\_search):  try:  with open(file\_name, 'r') as read\_obj:  for line in read\_obj:  if string\_to\_search in line:  return True  except FileNotFoundError:  return False  return False |

Checks if a given string (number plate) exists in a specified file.

#### Process an Image and Detect the Number Plate

|  |
| --- |
| # Function to process an image and detect the number plate  def process\_image(image\_path):  image = cv2.imread(image\_path)  if image is None:  print(f"Error: Could not open or read the image file at {image\_path}")  return None, None, None  # Resize and standardize the image  image = imutils.resize(image, width=500)  # Convert the image to grayscale  gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  # Reduce noise and smooth the image  gray = cv2.bilateralFilter(gray, 11, 17, 17)  # Find edges in the image  edged = cv2.Canny(gray, 170, 200)  # Find contours based on the edges  cnts, \_ = cv2.findContours(edged.copy(), cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_SIMPLE)  cnts = sorted(cnts, key=cv2.contourArea, reverse=True)[:30]  NumberPlateCount = None  # Loop over contours to find the best possible contour of the number plate  for i in cnts:  perimeter = cv2.arcLength(i, True)  approx = cv2.approxPolyDP(i, 0.02 \* perimeter, True)  if len(approx) == 4:  NumberPlateCount = approx  x, y, w, h = cv2.boundingRect(i)  crp\_img = image[y:y + h, x:x + w]  return crp\_img, image, gray  return None, image, gray |

Processes an image to detect the number plate. It reads the image, converts it to grayscale, applies a bilateral filter to smooth it, detects edges, and finds contours. It then checks for a contour with four sides, which is likely to be a number plate, and crops that part of the image.

### 4. Directory and File Paths

|  |
| --- |
| # Directory containing images of vehicles  dataset\_dir = 'Dataset'  image\_files = [f for f in os.listdir(dataset\_dir) if os.path.isfile(os.path.join(dataset\_dir, f))]  # Path to the database file  database\_file = './Database/Database.txt'  # Path to the Excel file for saving detection logs  excel\_path = 'detection\_log.xlsx' |

* dataset\_dir: Directory containing images of vehicles.
* image\_files: List of image files in the dataset directory.
* database\_file: Path to the file containing registered number plates.
* excel\_path: Path to the Excel file for saving detection logs.

### 5. Utility Functions

#### Format Duration

|  |
| --- |
| # Function to format duration in "days, hours, minutes"  def format\_duration(duration):  days = duration.days  hours, remainder = divmod(duration.seconds, 3600)  minutes, \_ = divmod(remainder, 60)  return f"{days} days, {hours} hours, {minutes} minutes" |

Formats a timedelta object into a string showing days, hours, and minutes.

#### Get the Next Available Slot

|  |
| --- |
| # Function to get the next available slot  def get\_next\_available\_slot(df):  used\_slots = set(df['Slot'].dropna())  for slot in range(1, 11): # Limited to 10 slots  if slot not in used\_slots:  return slot  return None |

Finds the next available parking slot from 1 to 10.

#### Clear the Existing Data in the Excel File

|  |
| --- |
| # Function to clear the existing data in the Excel file  def clear\_excel\_file():  df = pd.DataFrame(columns=['Number Plate', 'Entry Date', 'Entry Time', 'Exit Date', 'Exit Time', 'Duration', 'Status', 'Slot'])  df.to\_excel(excel\_path, index=False) |

Clears the existing data in the Excel file by creating a new empty DataFrame with the specified columns.

#### Update or Add Data in the Excel File

|  |
| --- |
| # Function to update or add data in the Excel file  def update\_excel(entry\_data, exit\_data=None):  if os.path.exists(excel\_path):  wb = load\_workbook(excel\_path)  ws = wb.active  max\_row = ws.max\_row  # Read the existing DataFrame  df = pd.read\_excel(excel\_path)  plate\_exists = False  for row in range(2, max\_row + 1): # Assuming the first row is the header  if ws.cell(row=row, column=1).value == entry\_data['Number Plate']:  if ws.cell(row=row, column=4).value and ws.cell(row=row, column=5).value:  # Calculate the duration between entry and exit times  entry\_datetime\_str = f"{ws.cell(row=row, column=2).value} {ws.cell(row=row, column=3).value}"  exit\_datetime\_str = f"{ws.cell(row=row, column=4).value} {ws.cell(row=row, column=5).value}"  entry\_datetime = datetime.strptime(entry\_datetime\_str, "%Y-%m-%d %H:%M:%S")  exit\_datetime = datetime.strptime(exit\_datetime\_str, "%Y-%m-%d %H:%M:%S")  duration = exit\_datetime - entry\_datetime  duration\_str = format\_duration(duration)  ws.cell(row=row, column=6, value=duration\_str)  ws.cell(row=row, column=7, value="Outside")  ws.cell(row=row, column=8, value='') # Clear the slot number  continue # Skip if exit date and time are filled  if exit\_data:  ws.cell(row=row, column=4, value=exit\_data['Exit Date'])  ws.cell(row=row, column=5, value=exit\_data['Exit Time'])  # Calculate the duration between entry and exit times  entry\_datetime\_str = f"{ws.cell(row=row, column=2).value} {ws.cell(row=row, column=3).value}"  exit\_datetime\_str = f"{exit\_data['Exit Date']} {exit\_data['Exit Time']}"  entry\_datetime = datetime.strptime(entry\_datetime\_str, "%Y-%m-%d %H:%M:%S")  exit\_datetime = datetime.strptime(exit\_datetime\_str, "%Y-%m-%d %H:%M:%S")  duration = exit\_datetime - entry\_datetime  duration\_str = format\_duration(duration)  ws.cell(row=row, column=6, value=duration\_str)  ws.cell(row=row, column=7, value="Outside")  ws.cell(row=row, column=8, value='') # Clear the slot number  plate\_exists = True  break  if not plate\_exists or exit\_data is None:  # Append new row for entry data  slot\_number = get\_next\_available\_slot(df)  if slot\_number is not None:  ws.append([entry\_data['Number Plate'], entry\_data['Entry Date'], entry\_data['Entry Time'], '', '', '', 'Inside', slot\_number])  else:  print("All parking slots are occupied. Cannot allow entry.")  return False # Indicate that entry is not allowed  wb.save(excel\_path)  else:  # Create a new Excel file with headers  df = pd.DataFrame(columns=['Number Plate', 'Entry Date', 'Entry Time', 'Exit Date', 'Exit Time', 'Duration', 'Status', 'Slot'])  slot\_number = get\_next\_available\_slot(df)  if slot\_number is not None:  entry\_data['Slot'] = slot\_number  new\_row = pd.DataFrame([entry\_data])  df = pd.concat([df, new\_row], ignore\_index=True)  df.to\_excel(excel\_path, index=False)  else:  print("All parking slots are occupied. Cannot allow entry.")  return False # Indicate that entry is not allowed  return True # Indicate that entry is allowed |

This function updates the Excel file with new entry or exit data. It handles both adding new entries and updating existing ones based on the number plate. If no slots are available, it indicates that entry is not allowed.

### 6. Initialize and Process Images

|  |
| --- |
| # Clear the existing data in the Excel file before running the script  clear\_excel\_file()  # Dictionary to store the latest status of each number plate  latest\_status = {}  # Process each image in the dataset  for image\_file in image\_files:  image\_path = os.path.join(dataset\_dir, image\_file)  cropped\_image, original\_image, gray\_image = process\_image(image\_path)  if cropped\_image is not None:  text = pytesseract.image\_to\_string(cropped\_image, lang='eng').strip()  print(f"Detected Text from {image\_file}: {text}")  text = ''.join(e for e in text if e.isalnum())  if text:  is\_registered = check\_if\_string\_in\_file(database\_file, text)  mod\_time = os.path.getmtime(image\_path)  mod\_datetime = datetime.fromtimestamp(mod\_time)  current\_date = mod\_datetime.strftime("%Y-%m-%d")  current\_time = mod\_datetime.strftime("%H:%M:%S")  print(f"Date and Time for {image\_file}: {current\_date} {current\_time}")  if is\_registered:  print(f'Number Plate {text} is Registered')  winsound.Beep(2500, 1200)  entry\_data = {  'Number Plate': text,  'Entry Date': current\_date,  'Entry Time': current\_time  }  exit\_data = {  'Number Plate': text,  'Exit Date': current\_date,  'Exit Time': current\_time  }  car\_entering = True # Assume the car is entering by default  if os.path.exists(excel\_path):  existing\_df = pd.read\_excel(excel\_path)  if text in existing\_df['Number Plate'].values:  # Check if the car is exiting  row = existing\_df.loc[existing\_df['Number Plate'] == text].iloc[-1]  if pd.isna(row['Exit Date']) and pd.isna(row['Exit Time']):  car\_entering = False # The car is exiting  if not update\_excel(entry\_data, exit\_data):  cv2.putText(original\_image, "No space inside to park", (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 0, 255), 2)  cv2.imshow("Car Status", original\_image)  cv2.imshow("Grayscale Image", gray\_image)  cv2.waitKey(0)  cv2.destroyAllWindows()  continue  car\_entering = False  else:  if not update\_excel(entry\_data):  cv2.putText(original\_image, "No space inside to park", (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 0, 255), 2)  cv2.imshow("Car Status", original\_image)  cv2.imshow("Grayscale Image", gray\_image)  cv2.waitKey(0)  cv2.destroyAllWindows()  continue  else:  if not update\_excel(entry\_data):  cv2.putText(original\_image, "No space inside to park", (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 0, 255), 2)  cv2.imshow("Car Status", original\_image)  cv2.imshow("Grayscale Image", gray\_image)  cv2.waitKey(0)  cv2.destroyAllWindows()  continue  else:  if not update\_excel(entry\_data):  cv2.putText(original\_image, "No space inside to park", (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 0, 255), 2)  cv2.imshow("Car Status", original\_image)  cv2.imshow("Grayscale Image", gray\_image)  cv2.waitKey(0)  cv2.destroyAllWindows()  continue  status = "Outside" if exit\_data and not car\_entering else "Inside"  latest\_status[text] = (image\_file, status)  # Display the car image with "Allowed" text only if the car is entering  if car\_entering:  cv2.putText(original\_image, "Allowed", (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 2)  cv2.imshow("Car Status", original\_image)  cv2.imshow("Grayscale Image", gray\_image)  cv2.waitKey(0)  cv2.destroyAllWindows()  else:  print(f'Number Plate {text} is Not Registered')  # Display the car image with "Not Allowed" text  cv2.putText(original\_image, "Not Allowed", (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 0, 255), 2)  cv2.imshow("Car Status", original\_image)  cv2.imshow("Grayscale Image", gray\_image)  cv2.waitKey(0)  cv2.destroyAllWindows()  else:  print(f"No valid text found in image {image\_file}")  else:  print(f"Error: No valid number plate contour found for image {image\_file}") |

This part of the code initializes the system by clearing the existing Excel file, then processes each image in the dataset. It detects the number plate, checks if it is registered, and updates the Excel file accordingly. If a registered vehicle is exiting, it updates the exit time and marks the status as 'Outside'. If a registered vehicle is entering and a slot is available, it updates the entry time and marks the status as 'Inside'.

### 7. Visualization

|  |
| --- |
| # Read the latest status from the Excel file  if os.path.exists(excel\_path):  df = pd.read\_excel(excel\_path)  for index, row in df.iterrows():  number\_plate = row['Number Plate']  status = row['Status']  slot = row['Slot']  image\_file = latest\_status[number\_plate][0] if number\_plate in latest\_status else None  if image\_file:  latest\_status[number\_plate] = (image\_file, status, slot)  # Plot the parking slot status  def plot\_parking\_slots(latest\_status):  total\_slots = 10 # Limited to 10 slots  slot\_status = ["Available"] \* total\_slots  for number\_plate, (\_, status, slot) in latest\_status.items():  if status == "Inside" and slot is not None:  slot = int(slot) # Ensure slot is an integer  slot\_status[slot - 1] = f"Occupied by {number\_plate}"  fig, ax = plt.subplots(figsize=(12, 8))  ax.barh(range(1, total\_slots + 1), [1] \* total\_slots, color=['green' if status == "Available" else 'red' for status in slot\_status])  ax.set\_yticks(range(1, total\_slots + 1))  ax.set\_yticklabels([f"Slot {i}" for i in range(1, total\_slots + 1)])  ax.set\_xlabel("Status")  ax.set\_title("Parking Slot Status")  for i, status in enumerate(slot\_status):  ax.text(0.5, i + 1, status, va='center', ha='center', color='white')  plt.show()  plot\_parking\_slots(latest\_status) |

This part of the code reads the Excel file to determine the status of each parking slot, then generates a bar chart to visualize the occupancy status of the slots. Red bars indicate occupied slots, and green bars indicate available slots. This script implements a comprehensive vehicle monitoring and parking management system, handling image processing, OCR-based number plate detection, database interactions, Excel logging, notification, and parking slot visualization.

### HTML Generation

After processing all the images and updating the status in latest\_status, the script generates an HTML report to display the vehicle status.

|  |
| --- |
| html\_content = """  <!DOCTYPE html>  <html>  <head>  <title>Vehicle Status</title>  <style>  body {  font-family: Arial, sans-serif;  }  .vehicle {  border: 1px solid #ccc;  margin-bottom: 20px;  padding: 10px;  }  .vehicle img {  max-width: 500px;  height: auto;  }  .vehicle-info {  margin-top: 10px;  }  </style>  </head>  <body>  <h1>Vehicle Status</h1>  """  for number\_plate, (image\_file, status, slot) in latest\_status.items():  html\_content += f"""  <div class="vehicle">  <img src="{os.path.join(dataset\_dir, image\_file)}" alt="Vehicle Image">  <div class="vehicle-info">  <p>Number Plate: {number\_plate}</p>  <p>Status: {status}</p>  <p>Slot Number: {slot}</p>  </div>  </div>  """  html\_content += """  </body>  </html>  """  # Save the HTML content to a file  html\_file\_path = "vehicle\_status.html"  with open(html\_file\_path, "w") as html\_file:  html\_file.write(html\_content)  print(f"HTML file generated: {html\_file\_path}") |

1. **Setting Up the Page Design**:
   * The code starts by creating a basic structure for an HTML page. It includes a title ("Vehicle Status") and some style instructions to make the page look neat and organized. For example, it sets a specific font for the text and adds a border around each vehicle's information.
2. **Adding Vehicle Information**:
   * The program then goes through each vehicle's data stored in latest\_status. For each vehicle, it creates a section in the HTML called <div class="vehicle">. Inside this section, it puts:
     + The vehicle's image, which it finds in the Dataset folder.
     + Details like the vehicle's number plate (license plate), its current status (whether it's inside a parking slot or outside), and if it's inside, it also mentions the slot number.
3. **Closing the Page**:
   * After adding all the vehicle information, the program finishes building the HTML page by closing all the open tags (</body> and </html>). This step is important to make sure the HTML page is properly formed and can be displayed correctly by web browsers.
4. **Saving to a File**:
   * Finally, the completed HTML page is saved to a file named vehicle\_status.html. This file will contain all the vehicle information formatted in HTML, ready to be opened in a web browser.
5. **Confirmation**:
   * The program prints a message confirming that the HTML file has been successfully generated and tells you where you can find it.

**Purpose:**

The purpose of this HTML generation part is to create a clear and organized report of all the vehicles processed by the program. Each vehicle's status and information are displayed neatly with its image, making it easy to see which vehicles are inside parking slots and which ones are outside. This HTML report serves as a visual summary of the program's activities, showing the current state of each vehicle in the parking area.

**Pattern**

|  |
| --- |
| import pandas as pd  import matplotlib.pyplot as plt  # Function to plot peak times  def plot\_peak\_times(df, column, title):  # Convert the time column to datetime  df[column] = pd.to\_datetime(df[column], format='%H:%M:%S').dt.time  # Extract the hour from the time column  df['Hour'] = pd.to\_datetime(df[column], format='%H:%M:%S').dt.hour  peak\_times = df['Hour'].value\_counts().sort\_index()  # Plotting  plt.figure(figsize=(10, 6))  plt.plot(peak\_times.index, peak\_times.values, marker='o')  plt.xlabel('Hour of the Day')  plt.ylabel('Number of Vehicles')  plt.title(title)  plt.grid(True)  plt.xticks(range(24))  plt.show()  # Read the Excel file  excel\_path = 'detection\_log.xlsx'  df = pd.read\_excel(excel\_path)  # Plot peak times for entry and exit  plot\_peak\_times(df, 'Entry Time', 'Peak Entry Times')  plot\_peak\_times(df, 'Exit Time', 'Peak Exit Times') |

The code reads an Excel file containing vehicle entry and exit times, then plots the number of vehicles per hour of the day for both entry and exit times using the plot\_peak\_times function.

**Steps**

1. **Import Libraries**: Import pandas for data manipulation and matplotlib.pyplot for plotting.
2. **Define plot\_peak\_times Function**:
   * Convert the time column to datetime.
   * Extract the hour from the time column.
   * Count occurrences of each hour.
   * Plot the hourly counts.
3. **Read Excel File**: Load the Excel file containing detection logs into a DataFrame.
4. **Plot Peak Times**:
   * Call plot\_peak\_times for 'Entry Time' to plot peak entry times.
   * Call plot\_peak\_times for 'Exit Time' to plot peak exit times.

This code will generate two plots showing the number of vehicles entering and exiting per hour of the day based on the data in the provided Excel file.