Problem Statement

Title: "Vehicle Movement Analysis and Insight Generation in a College Campususing Edge AI"

Objective:

The main goal of this project is to create a smart system using Edge AI. This system will analyze how vehicles move into and out of a college campus by using images from cameras that capture vehicle photos and license plates. The system aims to give us valuable information about:

- **Vehicle Movement Patterns:** Understanding when and how often vehicles enter and exit the campus
- **Parking Occupancy:** Monitoring which parking lots are used most frequently and at what times.
- **Vehicle Matching:** Identifying vehicles by matching their license plates to a database of approved vehicles.

By doing this, we can improve campus traffic management and security while also ensuring that parking resources are used efficiently.

Problem Description:

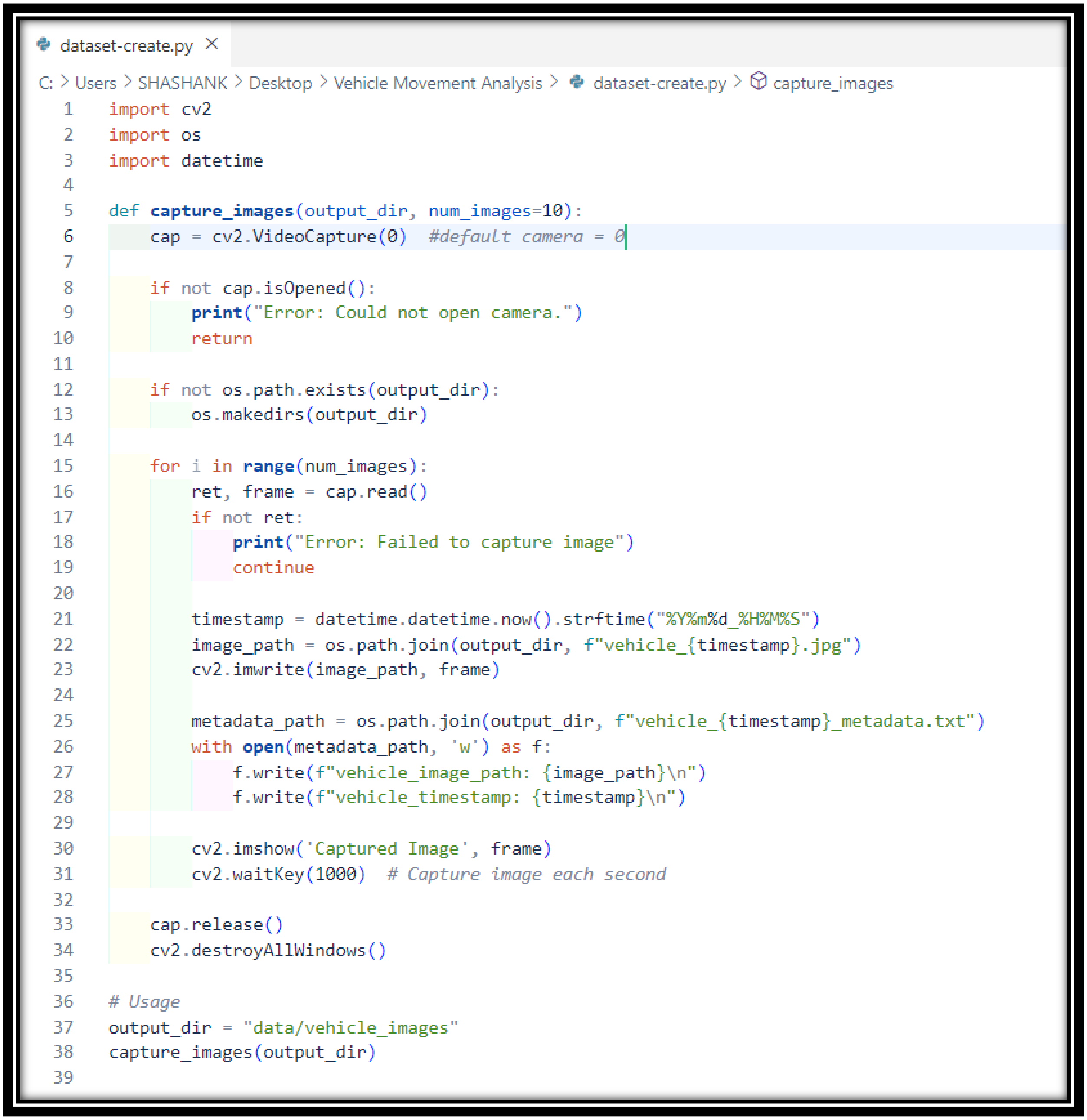
Managing vehicle movement and parking on a college campus can be quite challenging. To address this, we aim to develop a smart system using Edge AI technology. This system will analyze how vehicles move in and out of the campus by processing images from cameras that capture vehicle photos and license plates. The goal is to provide insights in real-time on three key aspects:

- **Vehicle Movement Patterns:** This involves studying how often vehicles enterand exit the campus, and identifying peak times and recurring patterns of movement.
- **Parking Occupancy:** The system will monitor the real-time occupancy of parking lots across the campus. It will highlight which lots are frequently occupied and when they are most used.
- **Vehicle Matching:** By comparing captured vehicle images and license plates with an approved database, the system can quickly identify unauthorized vehicles on campus.

Step-by-Step Solution

Step 1: Creation of Real-time Dataset

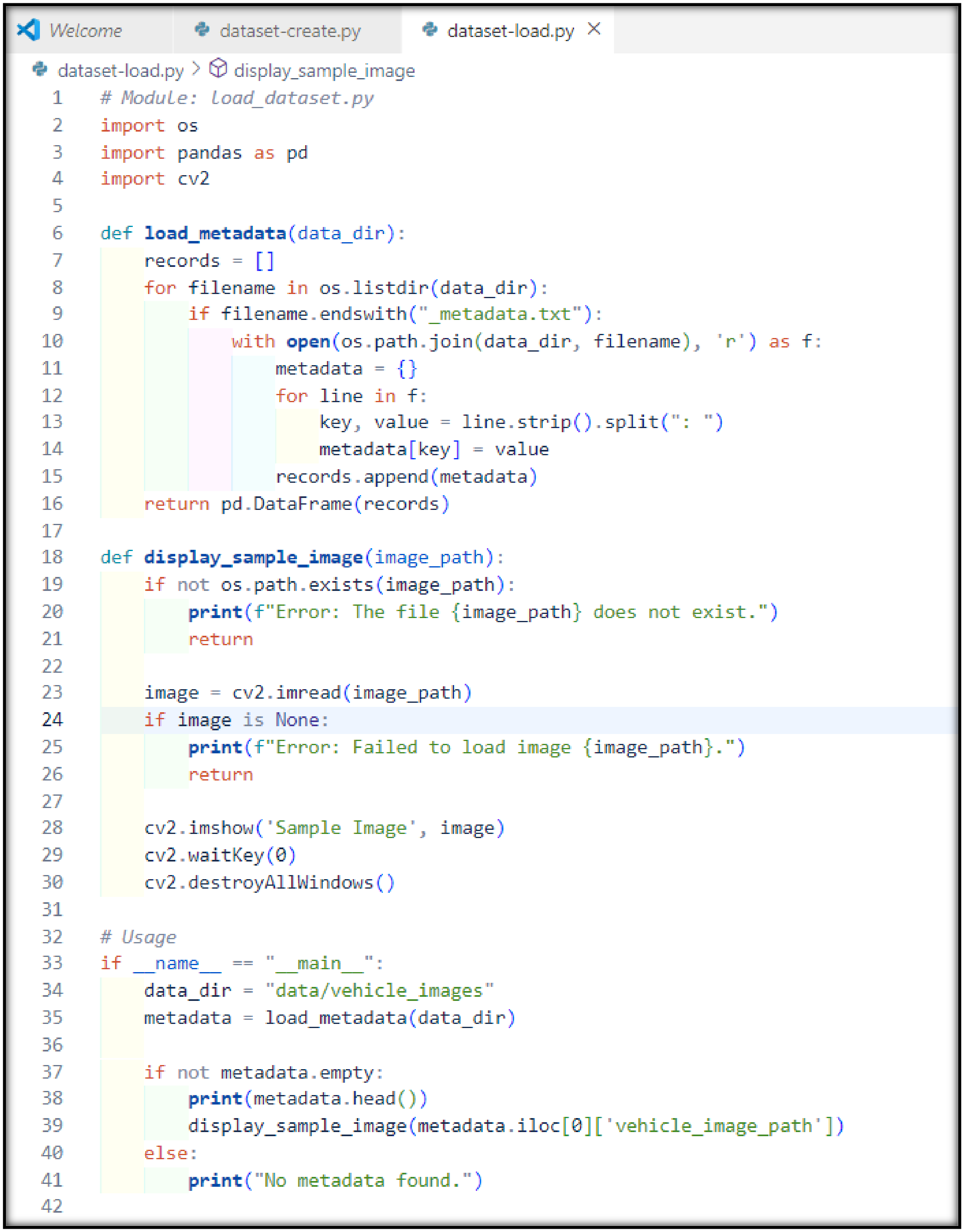
- **Tools Used**: Python, OpenCV
- **Techniques**: Capture images and timestamps
- **Code**:


dataset-create.py
C: > Users > SHASHANK > Desktop > Vehicle Movement Analysis > dataset-create.py > capture_images

```python
import cv2
import os
import datetime

def capture_images(output_dir, num_images=10):
    cap = cv2.VideoCapture(0)  #default camera = 0

    if not cap.isOpened():
        print("Error: Could not open camera.")
        return

    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    for i in range(num_images):
        ret, frame = cap.read()
        if not ret:
            print("Error: Failed to capture image")
            continue

        timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
        image_path = os.path.join(output_dir, f"vehicle_{timestamp}.jpg")
        cv2.imwrite(image_path, frame)

        metadata_path = os.path.join(output_dir, f"vehicle_{timestamp}_metadata.txt")
        with open(metadata_path, 'w') as f:
            f.write(f"vehicle_image_path: {image_path}\n")
            f.write(f"vehicle_timestamp: {timestamp}\n")

        cv2.imshow('Captured Image', frame)
        cv2.waitKey(1000)  # Capture image each second

    cap.release()
    cv2.destroyAllWindows()

# Usage
output_dir = "data/vehicle_images"
capture_images(output_dir)
```

Step 2: Load Real-time Dataset

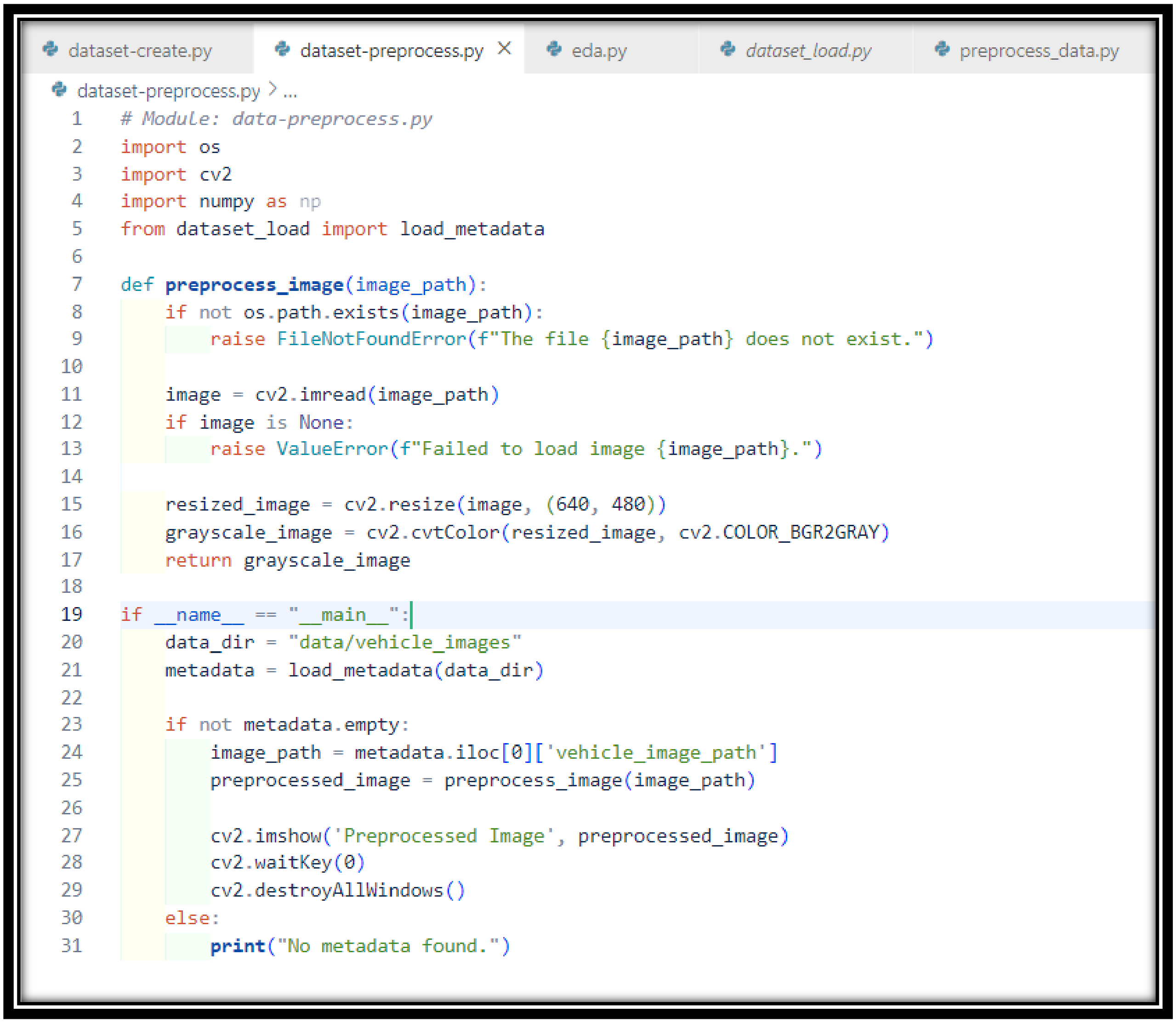
- **Tools Used**: Python, OpenCV
- **Techniques**: Load images and timestamps, display sample images
- **Code**:



```python
# Module: load_dataset.py
import os
import pandas as pd
import cv2

def load_metadata(data_dir):
    records = []
    for filename in os.listdir(data_dir):
        if filename.endswith("_metadata.txt"):
            with open(os.path.join(data_dir, filename), 'r') as f:
                metadata = {}
                for line in f:
                    key, value = line.strip().split(": ")
                    metadata[key] = value
                records.append(metadata)
    return pd.DataFrame(records)

def display_sample_image(image_path):
    if not os.path.exists(image_path):
        print(f"Error: The file {image_path} does not exist.")
        return

    image = cv2.imread(image_path)
    if image is None:
        print(f"Error: Failed to load image {image_path}.")
        return

    cv2.imshow('Sample Image', image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Usage
if __name__ == "__main__":
    data_dir = "data/vehicle_images"
    metadata = load_metadata(data_dir)

    if not metadata.empty:
        print(metadata.head())
        display_sample_image(metadata.iloc[0]['vehicle_image_path'])
    else:
        print("No metadata found.")
```

Step 3: Data Preprocessing

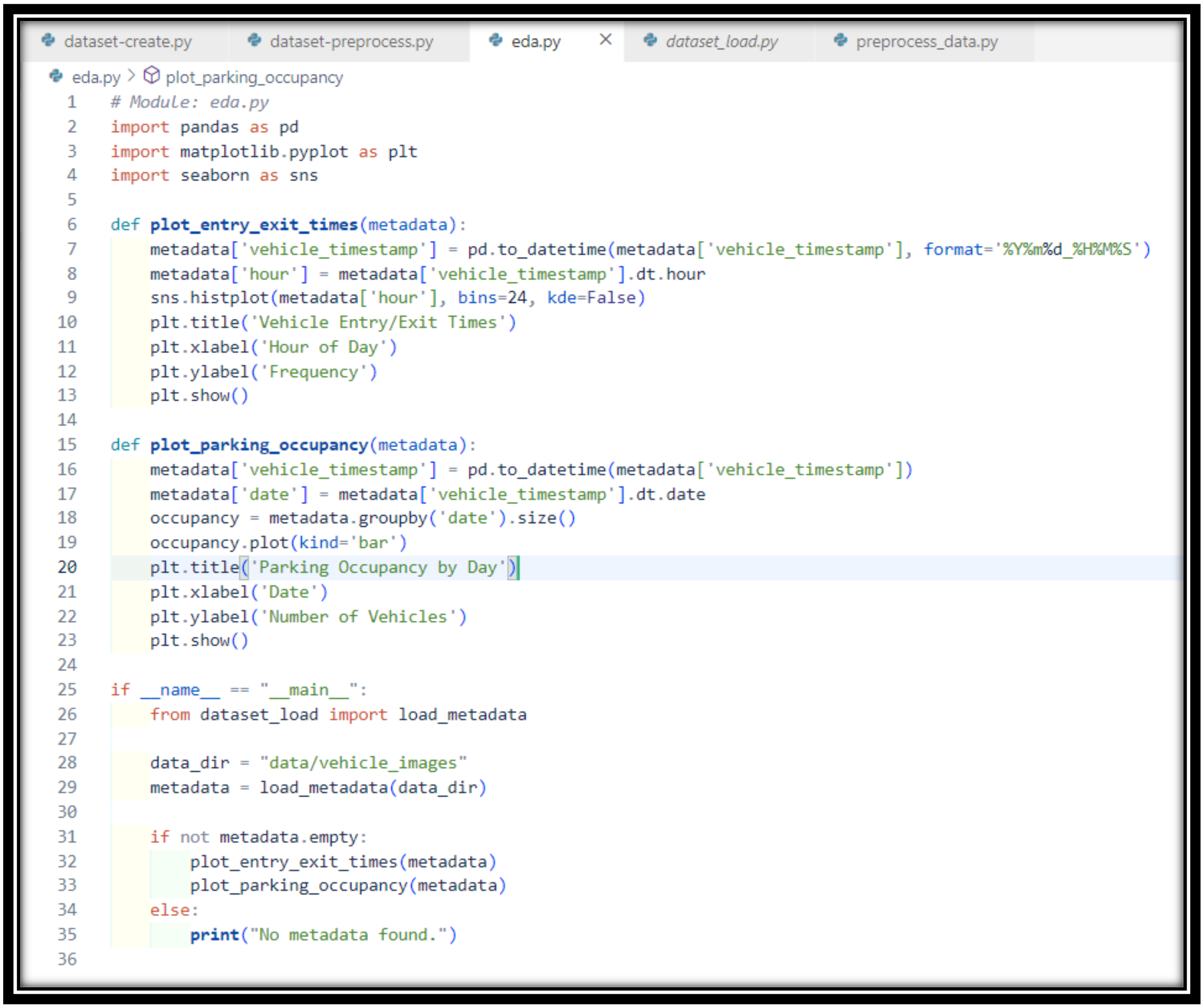
- **Tools Used**: OpenCV, Pandas, NumPy
- **Techniques**: Image resizing, grayscale conversion, handling missing values
- **Code**:



```python
# Module: data-preprocess.py
import os
import cv2
import numpy as np
from dataset_load import load_metadata

def preprocess_image(image_path):
    if not os.path.exists(image_path):
        raise FileNotFoundError(f"The file {image_path} does not exist.")

    image = cv2.imread(image_path)
    if image is None:
        raise ValueError(f"Failed to load image {image_path}.")

    resized_image = cv2.resize(image, (640, 480))
    grayscale_image = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)
    return grayscale_image

if __name__ == "__main__":
    data_dir = "data/vehicle_images"
    metadata = load_metadata(data_dir)

    if not metadata.empty:
        image_path = metadata.iloc[0]['vehicle_image_path']
        preprocessed_image = preprocess_image(image_path)

        cv2.imshow('Preprocessed Image', preprocessed_image)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
    else:
        print("No metadata found.")
```

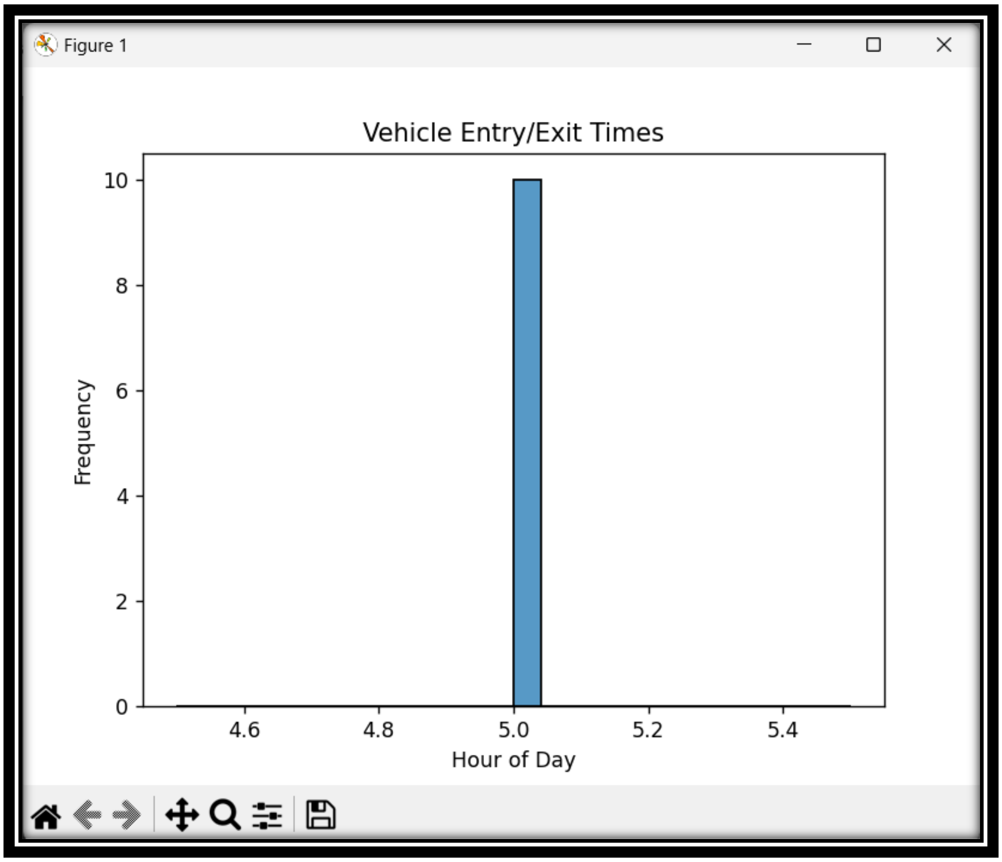
Step 4: Exploratory Data Analysis (EDA)

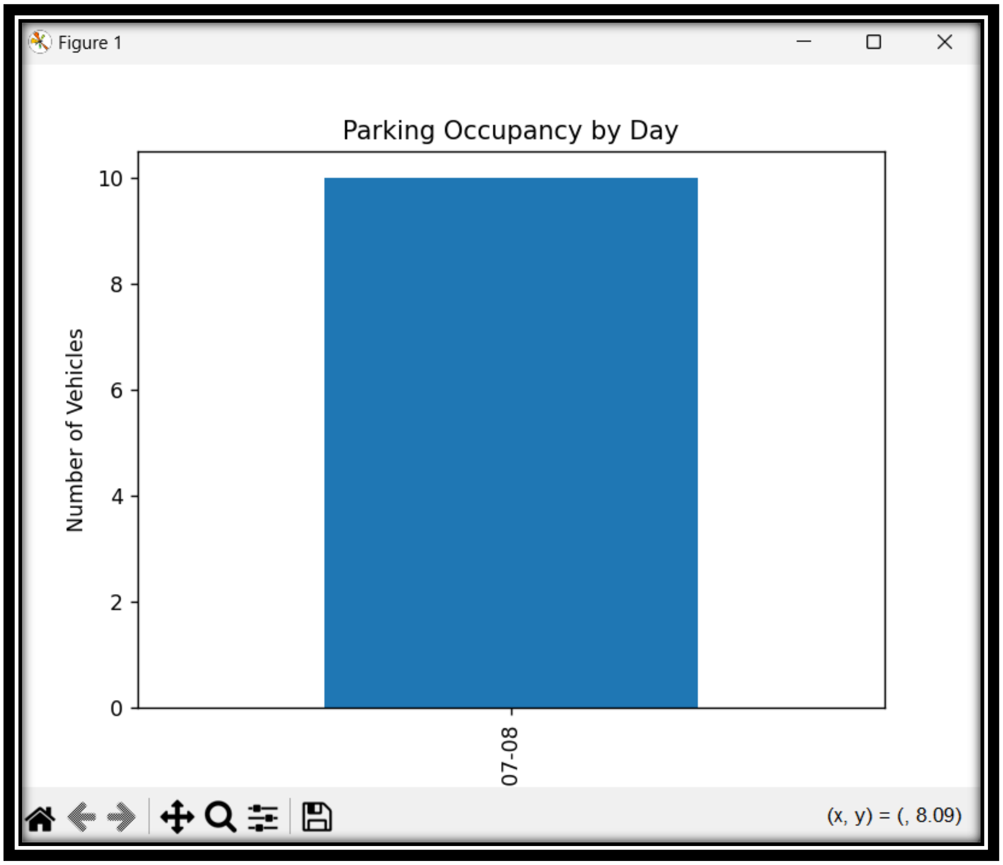
- **Tools Used:** Matplotlib, Seaborn
- **Techniques:** Plotting vehicle entry/exit times, occupancy trends
- **Code**:



```python
# Module: eda.py
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

def plot_entry_exit_times(metadata):
    metadata['vehicle_timestamp'] = pd.to_datetime(metadata['vehicle_timestamp'], format='%Y%m%d_%H%M%S')
    metadata['hour'] = metadata['vehicle_timestamp'].dt.hour
    sns.histplot(metadata['hour'], bins=24, kde=False)
    plt.title('Vehicle Entry/Exit Times')
    plt.xlabel('Hour of Day')
    plt.ylabel('Frequency')
    plt.show()

def plot_parking_occupancy(metadata):
    metadata['vehicle_timestamp'] = pd.to_datetime(metadata['vehicle_timestamp'])
    metadata['date'] = metadata['vehicle_timestamp'].dt.date
    occupancy = metadata.groupby('date').size()
    occupancy.plot(kind='bar')
    plt.title('Parking Occupancy by Day')
    plt.xlabel('Date')
    plt.ylabel('Number of Vehicles')
    plt.show()

if __name__ == "__main__":
    from dataset_load import load_metadata

    data_dir = "data/vehicle_images"
    metadata = load_metadata(data_dir)

    if not metadata.empty:
        plot_entry_exit_times(metadata)
        plot_parking_occupancy(metadata)
    else:
        print("No metadata found.")

```

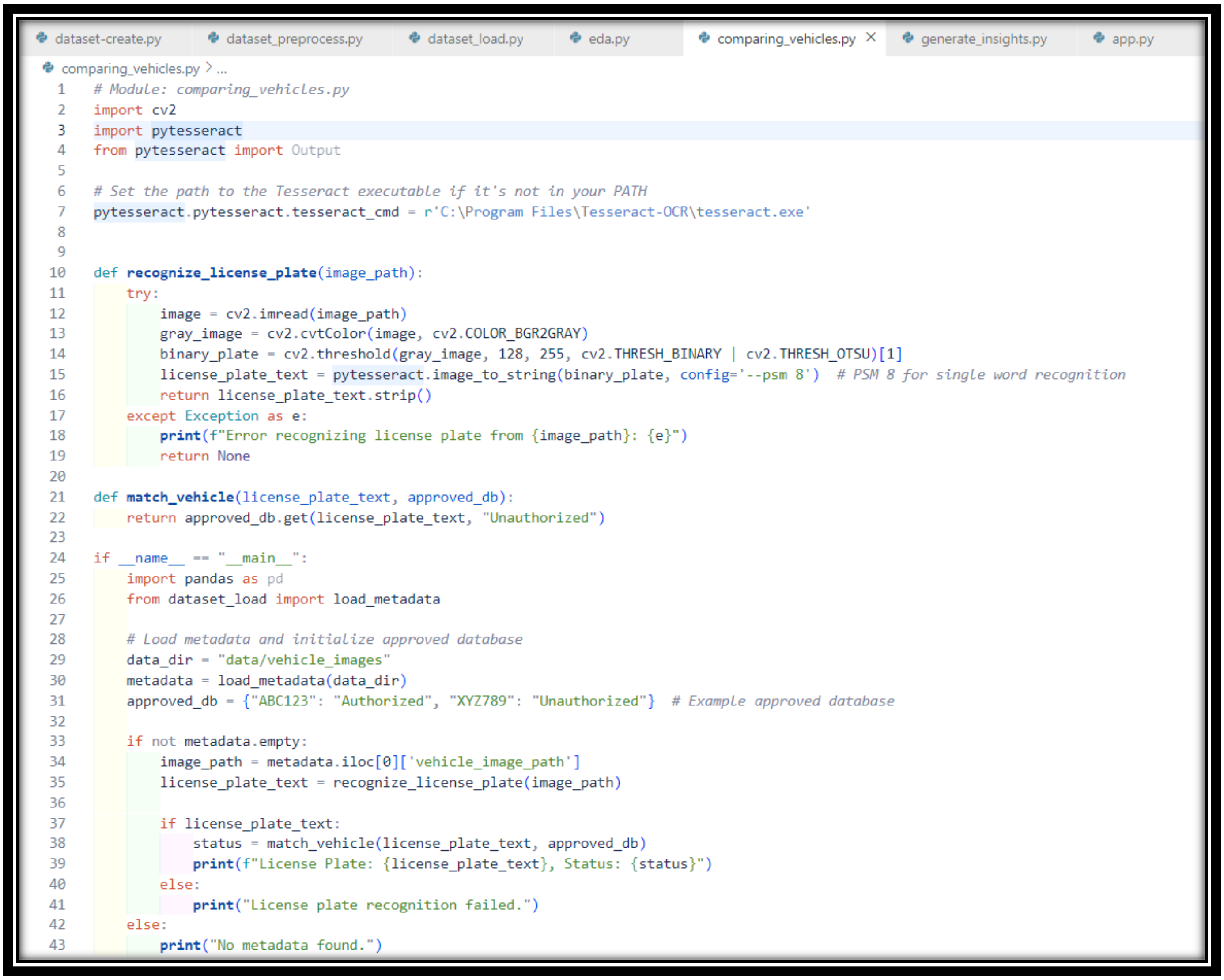
Sample Graph Visualizations:





Step 5: Vehicle Matching

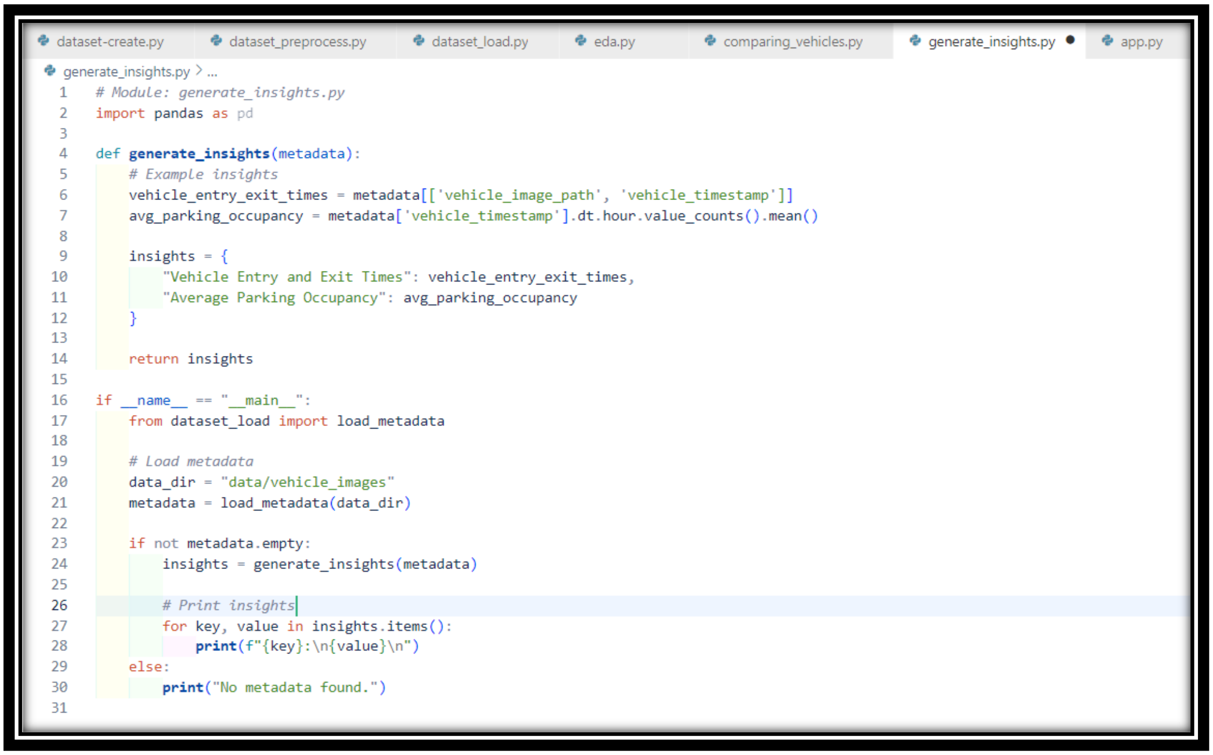
- **Tools Used:** OpenCV, Tesseract OCR
- **Techniques:** License plate recognition, database matching
- **Code**:



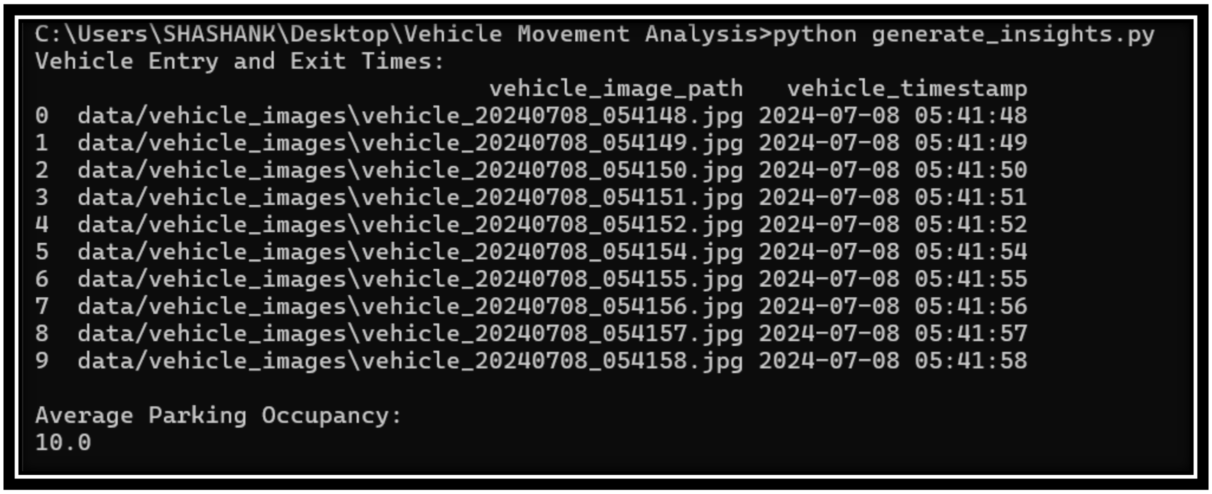
```python
# Module: comparing_vehicles.py
import cv2
import pytesseract
from pytesseract import Output

# Set the path to the Tesseract executable if it's not in your PATH
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'


def recognize_license_plate(image_path):
    try:
        image = cv2.imread(image_path)
        gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        binary_plate = cv2.threshold(gray_image, 128, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
        license_plate_text = pytesseract.image_to_string(binary_plate, config='--psm 8')  # PSM 8 for single word recognition
        return license_plate_text.strip()
    except Exception as e:
        print(f"Error recognizing license plate from {image_path}: {e}")
        return None

def match_vehicle(license_plate_text, approved_db):
    return approved_db.get(license_plate_text, "Unauthorized")

if __name__ == "__main__":
    import pandas as pd
    from dataset_load import load_metadata

    # Load metadata and initialize approved database
    data_dir = "data/vehicle_images"
    metadata = load_metadata(data_dir)
    approved_db = {"ABC123": "Authorized", "XYZ789": "Unauthorized"}  # Example approved database

    if not metadata.empty:
        image_path = metadata.iloc[0]['vehicle_image_path']
        license_plate_text = recognize_license_plate(image_path)

        if license_plate_text:
            status = match_vehicle(license_plate_text, approved_db)
            print(f"License Plate: {license_plate_text}, Status: {status}")
        else:
            print("License plate recognition failed.")
    else:
        print("No metadata found.")
```

Step 6: Insight Generation

- **Tools Used:** Pandas, Matplotlib
- **Techniques:** Generating insights from movement patterns, parking data
- **Code**:



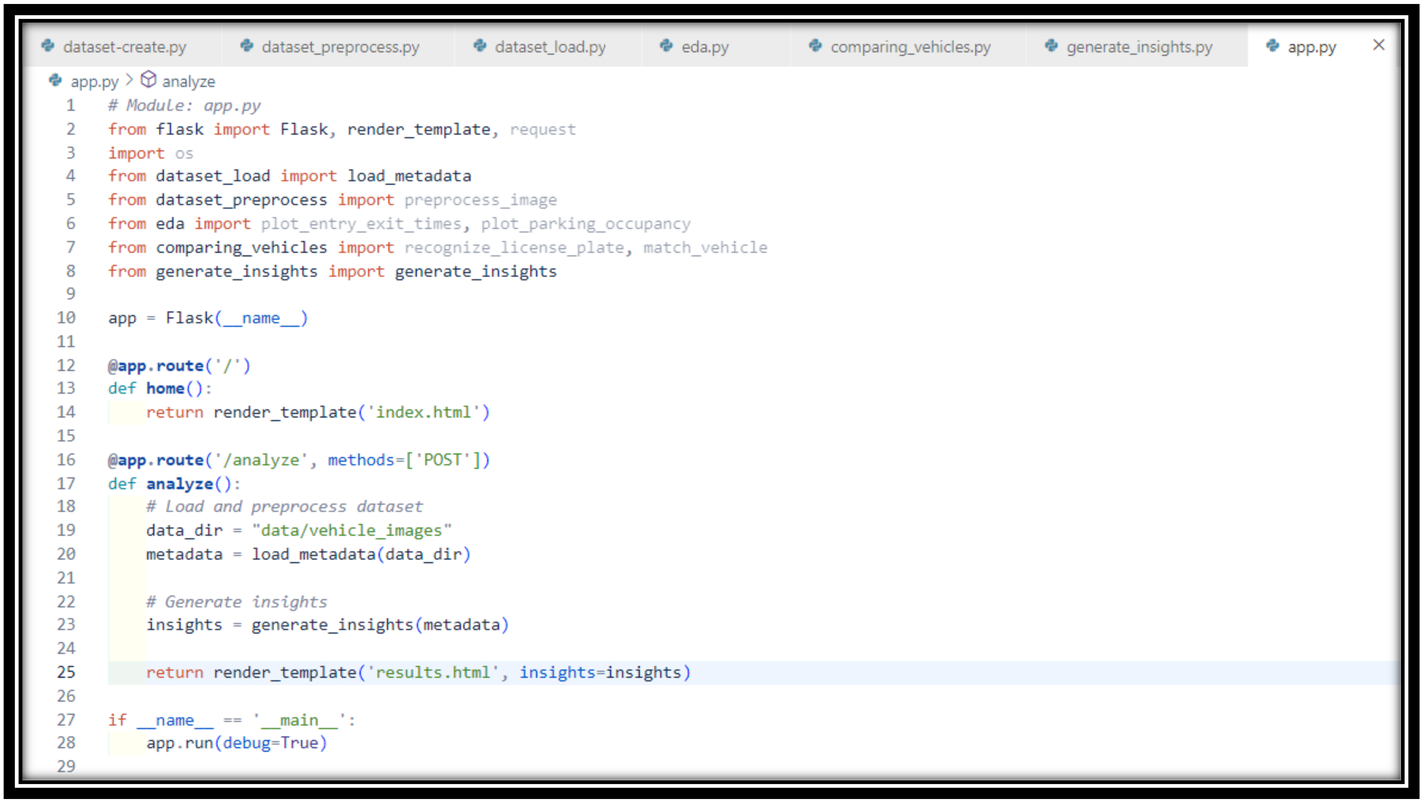
```python
# Module: generate_insights.py
import pandas as pd

def generate_insights(metadata):
    # Example insights
    vehicle_entry_exit_times = metadata[['vehicle_image_path', 'vehicle_timestamp']]
    avg_parking_occupancy = metadata['vehicle_timestamp'].dt.hour.value_counts().mean()

    insights = {
        "Vehicle Entry and Exit Times": vehicle_entry_exit_times,
        "Average Parking Occupancy": avg_parking_occupancy
    }

    return insights

if __name__ == "__main__":
    from dataset_load import load_metadata

    # Load metadata
    data_dir = "data/vehicle_images"
    metadata = load_metadata(data_dir)

    if not metadata.empty:
        insights = generate_insights(metadata)

        # Print insights
        for key, value in insights.items():
            print(f"{key}:\n{value}\n")
    else:
        print("No metadata found.")

```



```
C:\Users\SHASHANK\Desktop\Vehicle Movement Analysis>python generate_insights.py
Vehicle Entry and Exit Times:
                                vehicle_image_path   vehicle_timestamp
0  data/vehicle_images\vehicle_20240708_054148.jpg 2024-07-08 05:41:48
1  data/vehicle_images\vehicle_20240708_054149.jpg 2024-07-08 05:41:49
2  data/vehicle_images\vehicle_20240708_054150.jpg 2024-07-08 05:41:50
3  data/vehicle_images\vehicle_20240708_054151.jpg 2024-07-08 05:41:51
4  data/vehicle_images\vehicle_20240708_054152.jpg 2024-07-08 05:41:52
5  data/vehicle_images\vehicle_20240708_054154.jpg 2024-07-08 05:41:54
6  data/vehicle_images\vehicle_20240708_054155.jpg 2024-07-08 05:41:55
7  data/vehicle_images\vehicle_20240708_054156.jpg 2024-07-08 05:41:56
8  data/vehicle_images\vehicle_20240708_054157.jpg 2024-07-08 05:41:57
9  data/vehicle_images\vehicle_20240708_054158.jpg 2024-07-08 05:41:58

Average Parking Occupancy:
10.0
```

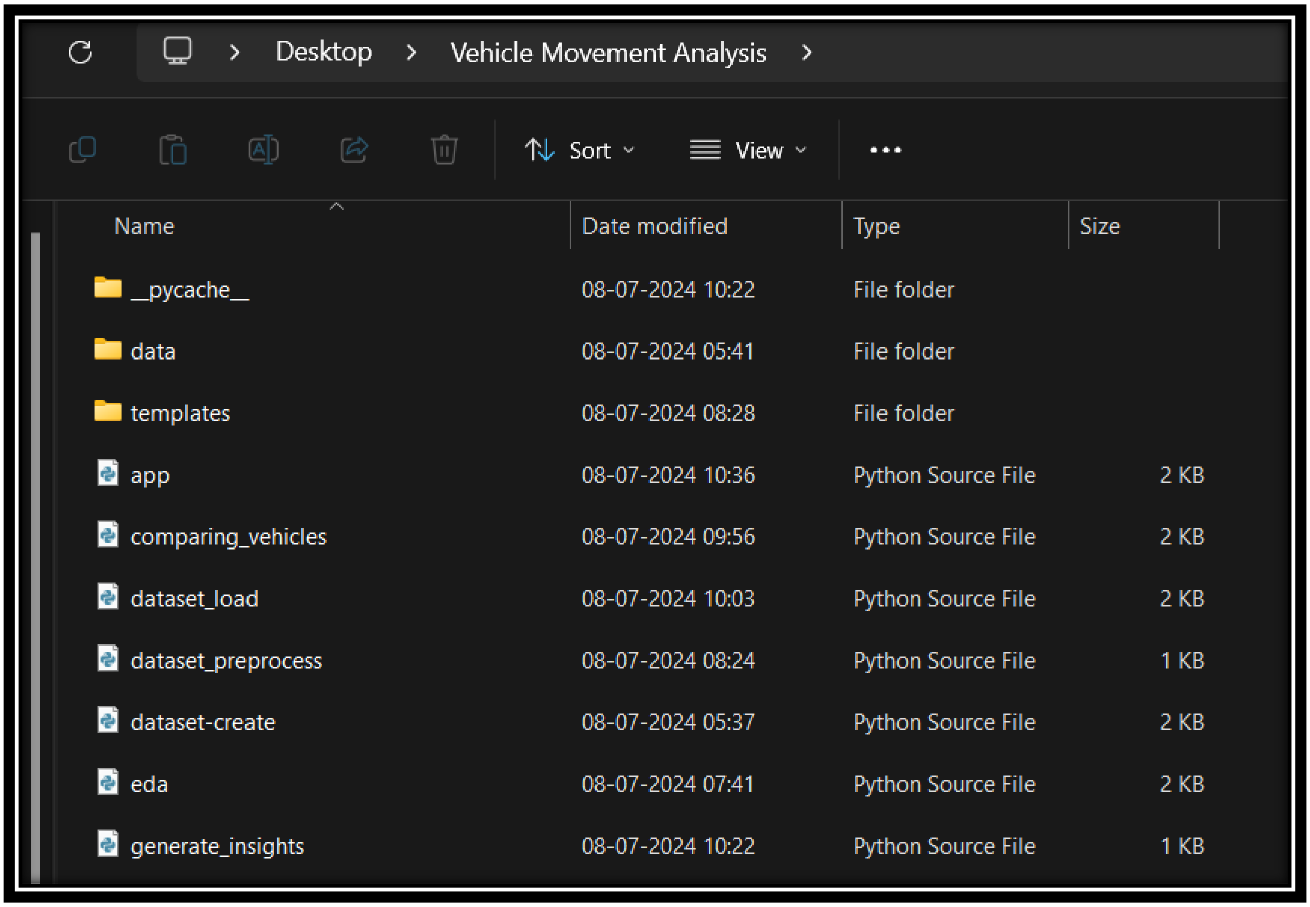
Step 7: Implementing the Solution in a Scalable Manner

- **Tools Used**: TensorFlow Lite, OpenVINO
- **Techniques:** Deploying AI models on Edge devices
- **Description**: The user-friendly interface module uses Flask to develop a web application that displays the generated insights in an accessible and interactive manner. This module providesa web interface where users can view visualizations and reports on vehicle movement patterns, parking occupancy, and vehicle matching status. The interface is designed to be intuitive and easy to navigate, allowing users to access and interpret the data effortlessly.
- **Code**:



```python
# Module: app.py
from flask import Flask, render_template, request
import os
from dataset_load import load_metadata
from dataset_preprocess import preprocess_image
from eda import plot_entry_exit_times, plot_parking_occupancy
from comparing_vehicles import recognize_license_plate, match_vehicle
from generate_insights import generate_insights

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/analyze', methods=['POST'])
def analyze():
    # Load and preprocess dataset
    data_dir = "data/vehicle_images"
    metadata = load_metadata(data_dir)

    # Generate insights
    insights = generate_insights(metadata)

    return render_template('results.html', insights=insights)

if __name__ == '__main__':
    app.run(debug=True)

```

Folder Struct



Name	Date modified	Type	Size
__pycache__	08-07-2024 10:22	File folder	
data	08-07-2024 05:41	File folder	
templates	08-07-2024 08:28	File folder	
app	08-07-2024 10:36	Python Source File	2 KB
comparing_vehicles	08-07-2024 09:56	Python Source File	2 KB
dataset_load	08-07-2024 10:03	Python Source File	2 KB
dataset_preprocess	08-07-2024 08:24	Python Source File	1 KB
dataset-create	08-07-2024 05:37	Python Source File	2 KB
eda	08-07-2024 07:41	Python Source File	2 KB
generate_insights	08-07-2024 10:22	Python Source File	1 KB