

Automated Parking System using ML

TEAM – OXFORD

RAGHAV AGARWAL (10X20EC046)

S NARASIMHA (10X20EC051)

To start a virtual environment for your machine learning project on Windows, you can follow these steps using Python's `venv` module:

1. **Open Command Prompt:** Press the Windows key, type "cmd," and press Enter to open the Command Prompt.
2. **Navigate to Your Project Directory:** Use the `cd` command to change the directory to your project folder. For example:

```
bash
cd path\to\your\project\folder
```


3. **Create a Virtual Environment:** Run the following command to create a virtual environment. Replace `<env_name>` with your desired environment name (e.g., "myenv").
- 4.

Replace `<env_name>` with your desired environment name (e.g., "myenv"):

```
php
python -m venv <env_name>
```


For example:

```
python -m venv myenv
```

 Copy code


Activate the Virtual Environment: To activate the virtual environment, use the following command:

```
php  
  
<env_name>\Scripts\activate
```

 Copy code

Using our previous example:

```
myenv\Scripts\activate
```

 Copy code

Once activated, your Command Prompt will show the environment name in parentheses, indicating that you are now working within the virtual environment.

Install Required Libraries: You can now use `pip` to install the required libraries and dependencies for your machine learning project. For example:

Creating a requirement text file (usually named `requirements.txt`) is a good practice in Python development to document the dependencies of your project. You can use this file to easily install all the required packages using `pip`. Here's how you can create one for your project:

1. Open a text editor (e.g., Notepad) or use a code editor like Visual Studio Code.
2. Create a new text file and add the following lines to list your project's dependencies. It appears that you have several duplicate imports in your code. Make sure to include each library only once:

mysql-connector-python

PyQt5

numpy

opencv-python

imutils

pytesseract

pandas

requests

3. Save the file with the name `requirements.txt` in the root directory of your project.

Now, anyone who wants to set up your project can use the `requirements.txt` file to install all the required libraries using `pip`. They can open a command prompt or terminal, navigate to the project's directory, and run the following command:

```
bash
pip install -r requirements.txt
```

This command will automatically install all the libraries listed in the `requirements.txt` file, ensuring that your project's dependencies are consistent across different environments.

Additionally, please note that you have multiple imports of the same libraries in your code, which is unnecessary. It's a good practice to organize your imports and remove duplicates to make your code cleaner and more maintainable.

```
mainCar.py > ...
1  import mysql.connector
2  import datetime
3  import sys
4  import re
5  import time
6
7  from PyQt5 import QtCore, QtWidgets, uic
8
9  mydb = mysql.connector.connect(host = "localhost", user = "smoke",
10                               passwd = "hellomoto", database = "car", autocommit=True)
11  mycursor = mydb.cursor()
12
13  mycursor.execute("DROP TABLE slot")
14  mycursor.execute("DROP TABLE duration")
15  mycursor.execute("DROP TABLE entry")
16  mycursor.execute("DROP TABLE exits")
17  mycursor.execute("DROP TABLE cost")
18
19  mycursor.execute("CREATE TABLE slot(carNumber VARCHAR(15), slot int)")
20  mycursor.execute("CREATE TABLE entry(carNumber VARCHAR(15), entry
```

This Python code appears to be a part of a parking management system application. It utilizes the PyQt5 library for creating a graphical user interface (GUI) and interacts with a MySQL database to manage parking slots,

entry, exit, and calculate costs for vehicles. Here's a breakdown of what the code does:

1. Imports: The code starts by importing various Python libraries, including ``mysql.connector``, ``datetime``, ``sys``, ``re``, and ``time``, for database operations, date and time handling, regular expressions, and other functionalities. Additionally, it imports PyQt5 for building the GUI.

2. Database Connection: It establishes a connection to a MySQL database using the ``mysql.connector`` library. The connection details like host, user, password, and database name are provided. Auto-commit mode is enabled to automatically commit database changes.

3. Database Setup: Several database tables are created using SQL ``CREATE TABLE`` statements. These tables include ``slot``, ``entry``, ``exits``, ``duration``, and ``cost``, which likely store information about parking slots, entry times, exit times, duration, and associated costs.

4. Slots Lis: The ``slots`` list is initialized with 16 elements, representing parking slots. Each slot is initially set to ``False``, indicating it is empty.

5. GUI Initialization: The code sets up the graphical user interface (GUI) using PyQt5. It loads the UI layout from the "front.ui" file and connects UI elements like buttons and text input fields to specific functions.

6. Functions: Inside the GUI class, several functions are defined:

- ``xd()``: This function is called when the "ENTRYBUTTON" is pressed. It checks if a given car number already exists in the database's ``slot`` table to prevent duplicate entries.

- ``bla()``: This function is called when a valid car number is provided. It clears the text input field and proceeds to the ``entry()`` function.

- ``entry()``: When a valid car number is provided and not already in use, this function assigns the car to the next available parking slot, records the entry time, and updates the database accordingly. It also updates the GUI to reflect the slot number.

- ``blank()``: This function is intended to handle cases where the car number is missing. However, the implementation is currently commented out.

- ``exit()``: This function handles vehicle exits. It records the exit time, calculates the parking duration and cost, updates the database with these details, and updates the GUI accordingly. It also marks the parking slot as available.

7. Main Function: The ``main()`` function initializes the PyQt5 application, creates an instance of the GUI class, and displays the GUI window.

8. Execution: The script is set to execute the ``main()`` function if it is the main module, meaning if the script is run directly.

In summary, this code combines a PyQt5-based GUI with MySQL database operations to manage parking slots, track vehicle entries and exits, and calculate parking costs. However, there are a few improvements that can be made, such as better error handling and refactoring the code for improved readability and maintainability.

To see the MySQL database, you can use a database management tool like MySQL Workbench. If you don't have it installed, you can follow these steps to install

MySQL Server and MySQL Workbench on your Windows system:

1. Download MySQL Installer:

- Visit the official MySQL website at <https://dev.mysql.com/downloads/installer/>.
- Scroll down to find the "MySQL Installer for Windows" section.
- Download the "MySQL Installer for Windows" (usually the web community version, which is smaller in size).

2. Run the Installer:

- Locate the downloaded installer file (e.g., `mysql-installer-web-community.exe`) and double-click it to run it.

3. Choosing Setup Type:

- In the installer, you will be presented with different setup types. Choose "Developer Default" for a typical development setup that includes MySQL Server and MySQL Workbench. You can also customize your

installation by selecting "Custom" and choosing specific components.

4. MySQL Server Configuration:

- During the installation process, you will be asked to configure MySQL Server. You can choose the default configuration settings or customize them based on your needs. Remember the MySQL root password you set during this step; you will need it to access the database.

5. MySQL Workbench Installation:

- The installer will also offer to install MySQL Workbench as part of the setup. Make sure to select it, as this is the graphical tool you will use to interact with the database.

6. Completing the Installation:

- Follow the remaining installation steps and complete the installation process.

7. Starting MySQL Workbench:

- After the installation is complete, you can find MySQL Workbench in your Windows Start menu. Launch it.

8. Connect to Your MySQL Server:

- In MySQL Workbench, click on the "Local Instance MySQL" connection to connect to your MySQL Server using the root username and password you specified during installation.

9. Exploring the Database:

- Once connected, you can explore and manage your MySQL databases using the MySQL Workbench interface. You can view tables, execute SQL queries, and perform various database-related tasks.

That's it! You should now have MySQL Server and MySQL Workbench installed on your Windows machine, and you can use MySQL Workbench to access and manage your database created in your Python script.

```
mes.py > ...
1  import re
2  import requests
3  import mysql.connector
4  import numpy as np
5  import cv2
6  import imutils
7  import pytesseract
8  import pandas as pd
9  import time
10 import mysql.connector
11 import time
12 import requests
13
14
15 def entryMessage():
16     url = "https://www.fast2sms.com/dev/bulkV2"
17     # mycursor.execute("select name from userInfo where vnumber =
18     %s", (n,))
19     # name = str(re.sub("[^() ]" , "" , str(mycursor.fetchone())))
```

The provided Python code defines two functions, `entryMessage()` and `exitMessage()`, which are responsible for sending SMS messages using the Fast2SMS API. Here's what each function does:

1. `entryMessage()`:

- This function is used to send an entry message to a user who has parked their vehicle in a parking slot.
- It constructs a message that includes the user's name, allocated parking slot number, and a link to view a map of the parking area.
- The message is then sent to the user's phone number using the Fast2SMS API.
- The message content is personalized with the user's name and slot number.

2. `exitMessage()`:

- This function is used to send an exit message to a user who is leaving the parking area.
- It includes information about the parking fee (amount) and a link for the user to make a payment.
- The message is then sent to the user's phone number using the Fast2SMS API.
- The message content includes the parking fee and a payment link.

Both functions use the Fast2SMS API to send SMS messages. They construct the message content with relevant information and make a GET request to the Fast2SMS API endpoint to send the SMS.

Please note that the actual SMS messages will be sent to the phone number specified in the `number` variable, and the message content will be based on the variables and text provided in the functions. Additionally, the code assumes that the Fast2SMS API credentials (`authorization`) are correctly configured and that the Fast2SMS service is accessible.

```

detect.py > ...
1  import numpy as np
2  import cv2
3  import imutils
4  import pytesseract
5  import pandas as pd
6  import time
7  import mysql.connector
8  import datetime
9  import sys
10 import re
11 import time
12 import requests
13 from PyQt5 import QtCore, QtWidgets, uic
14 image_path = 'images/10(1).jpg'
15
16 img = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
17 img = imutils.resize(img, width=500)
18 cv2.imshow(image_path, img)
19
20 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

```

The provided Python code performs Optical Character Recognition (OCR) on an image of a vehicle license plate using the Tesseract OCR engine. It then stores the recognized text (vehicle number) in a CSV file. Here's an explanation of each part of the code:

1. Import Libraries:

- The code starts by importing the necessary libraries, including `numpy`, `cv2` (OpenCV), `imutils`, `pytesseract`, `pandas`, `time`, `mysql.connector`, and other standard libraries.

2. Image Loading and Preprocessing:

- It loads an image from the specified path (``image_path``) using OpenCV (``cv2``).
- Resizes the image to a width of 500 pixels using ``imutils``.
- Converts the image to grayscale.
- Applies bilateral filtering and Canny edge detection to preprocess the image.

3. Contour Detection:

- It detects contours in the processed image using ``cv2.findContours``.
- Sorts the contours by area in descending order and keeps the top 30 largest contours.
- Searches for a contour with four vertices (assuming it's the license plate contour) by checking for approximations with four sides.

4. Number Plate Masking:

- Creates a mask to isolate the region of the image containing the license plate.
- Applies the mask to the original image to obtain a new image containing only the license plate.

5. Tesseract OCR:

- Configures Tesseract OCR with the English language and specific page segmentation mode (``config``).
- Runs Tesseract OCR on the isolated license plate image (``new_image``) to extract text (the vehicle number).
- Stores the recognized text in the ``text`` variable.

6. Data Storage:

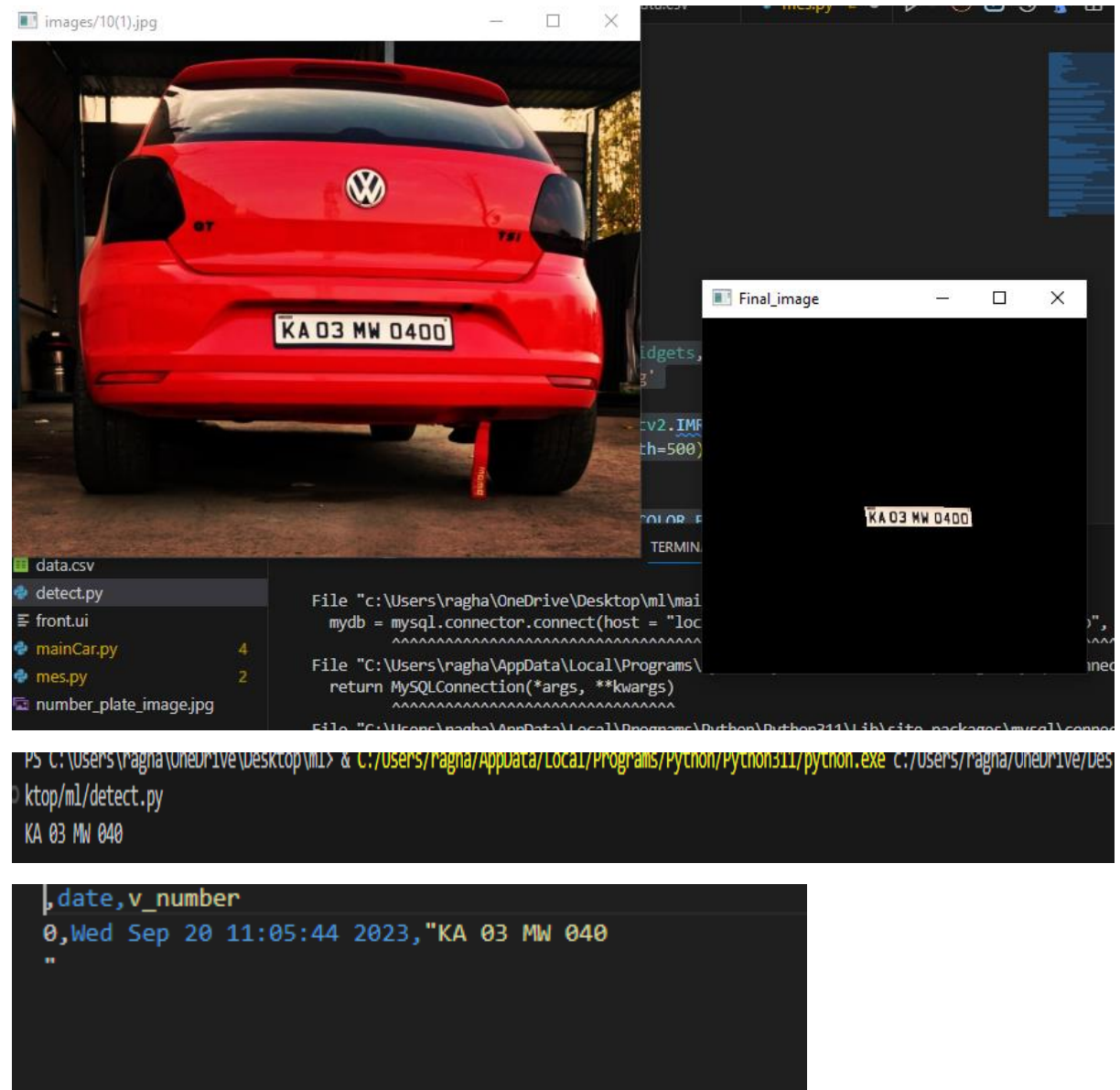
- Captures the current date and time using ``time.asctime()`` and stores it in the 'date' column of a DataFrame.
- Stores the recognized vehicle number (text) in the 'v_number' column of the same DataFrame.
- Saves this data to a CSV file named 'data.csv' using ``pandas``.

7. Display and Output:

- Displays the recognized text (vehicle number) on the console.
- Displays the final isolated license plate image using OpenCV's ``cv2.imshow``.
- Waits for a key press to exit the OpenCV window.

This code is designed to process an image of a vehicle license plate, extract the license plate number using OCR,

and save the recognized number along with the timestamp in a CSV file. It is a common use case for automated license plate recognition (ALPR) systems.



```

front.ui
577
578
579     </string>
580     </property>
581     <property name="text">
582         <string>16</string>
583     </property>
584     <property name="checkable">
585         <bool>false</bool>
586     </property>
587 </widget>
588 <widget class="QPushButton" name="s14">
589     <property name="geometry">
590         <rect>
591             <x>640</x>
592             <y>550</y>
593             <width>111</width>
594             <height>51</height>
595         </rect>
596     </property>

```

The provided code appears to be an XML file that defines the user interface (UI) layout for a PyQt5 application. This XML file is typically created using Qt Designer, which is a visual tool for designing Qt-based user interfaces. Let's break down what this XML code represents:

1. XML Declaration:

- The first line `<?xml version="1.0" encoding="UTF-8"?>` is the XML declaration, specifying the version and encoding of the XML file.

2. UI Version:

- `<ui version="4.0">` indicates the version of the Qt Designer UI file format being used (version 4.0 in this case).

3. Main Window Definition:

- `<widget class="QMainWindow" name="CarParkingManager">` defines a QMainWindow named "CarParkingManager" as the main window for the application. This main window will contain various widgets.

4. Window Properties:

- Several properties of the main window are defined, such as its size, minimum and maximum size, and window title.

5. Central Widget:

- `<widget class="QWidget" name="centralwidget">` defines a QWidget named "centralwidget" that serves as the central container for other widgets in the main window.

6. Frame:

- `<widget class="QFrame" name="frame">` defines a QFrame named "frame" within the central widget. This frame likely serves as a container for grouping and organizing other widgets.

7. Buttons:

- Several QPushButton widgets (s1, s2, s3, etc.) are defined with properties like position, size, text, style, and colors. These buttons are placed within the frame and may represent various functionalities or options in the application.

8. Labels and Line Edit:

- QLabel and QLineEdit widgets are defined to display text and receive user input, respectively.

9. Push Buttons for Entry and Exit:

- Two QPushButton widgets (ENTRYBUTTON and EXITBUTTON) are defined for entry and exit actions in the application. They have specific styles for background color and text color.

10. Stylesheets:

- The code includes stylesheets for various widgets, specifying their appearance and behavior, including colors, fonts, and border-radius.

11. Resource and Connection Sections:

- The `<resources/>` and `<connections/>` sections are currently empty. In Qt Designer, resources can be added to manage external files like images, and connections can

define how signals from widgets are connected to slots (functions) in the application.

This XML code is essentially a visual representation of the user interface layout for a car parking management application. When loaded into a PyQt5 application, this XML file will define the structure and appearance of the application's main window and its various widgets, buttons, labels, and input fields. Developers can use this XML file as a template to create the corresponding Python code for the application logic and event handling.