**Public Transport Optimization**

**Phase 4 : Development Part 2**

**Title :** Continue buliding the project by performing different activities like feature engineering, model training, evalution etc...

**Introduction:**

It is defined as the data on the number of public transport users,collected with IoT sensors,allows routes to be optimized or new ones to be designed.These sensors can be in the vehicle or also in the security cameras.

By defining these project objectives and detailing the IoT sensor design, real-time transit information platform, and integration approach, We will have a comprehensive roadmap for improving public transportation services with real-time data and information for passengers.

Certainly, I can provide a general outline of the typical activities involved in a data science or machine learning project, including feature engineering, model training, and evaluation. Keep in mind that the specific steps and techniques can vary based on your project's goals and the nature of your data. Here's a high-level overview.

***Technologies used:***

**HTML:**

IoT devices often have web-based user interfaces that allow users to interact with and control them remotely through a web browser. HTML is used to create the web pages and user interfaces that users access to monitor and control their IoT devices.

HTML can be a useful tool for creating user interfaces and visualizing data in IoT applications, the core functionality and communication between IoT devices often rely on other technologies and protocols, such as MQTT, CoAP, HTTP, and various networking protocols. HTML is just one component of the overall IoT ecosystem, and its use is typically limited to the presentation layer of IoT application.

***C++:***

Designwise, C++ lends itself to embedded development because the language lies in between higher-level software and hardware, allowing you to access and control hardware directly without cutting the advantages of a high-level language. It’s particularly useful for hardware that will need to be around for a while, as programs written in C++ can operate for a long time due to the language’s high stability.

Here are some more reasons for choosing C++ to build smart IoT applications.

Low-level manipulation: Since C++ is closely related to C, which is a procedural language closely linked to the machine language, C++ provides low-level manipulation of data at a specific level. That is the reason why Embedded systems and compilers are developed with the help of C++.

Multi-paradigm: C++ is a multi-paradigm programming language. The phrase “Paradigm” refers to programming style. It involves the logic, structure, and procedure of the program. The three paradigms of C++ are generic, imperative, and object-oriented. Generic programming means using a single approach to serve several objectives. Imperative programming, on the other hand, means to use the statements that change a program’s status.

Memory management: C++ gives the developer the capability of total control over memory management. It can be an asset and a liability as it increases the engagement of the user to manage memory rather than being managed itself by the Garbage collector. This method is implemented using DMA (Dynamic memory allocation) using pointers.

Object-oriented: The greatest advantage of C++ is the feature of object-oriented programming that involves concepts like classes, inheritance, polymorphism, data abstraction, and encapsulation that enable code reusability and makes a program even more secure. It also helps us deal with real-world problems by interpreting data as an object. C was previously lacking this feature and so it was created, proving to be highly significant. This feature gave birth to numerous job prospects and the future of c++ programming and development.

Portability: C++ offers the feature of portability also known as platform independence which lets the user run the same program on different operating systems or interfaces easily. For example, you write a program in LINUX OS, and for some reason, you switch to Windows, in that case, you would be able to run the same program in windows as well without any restraint.

***JAVASCRIPT:***

It is the most popular lightweight, interpreted compiled programming language. It can be used for both Client-side as well as Server-side developments. JavaScript is also known as a scripting language for web pages.

However, it has become increasingly popular for building Internet of Things (IoT) applications. IoT refers to the connection of everyday devices to the internet, allowing them to communicate with each other and with us. In this blog, we will explore how to use JavaScript to build IoT applications, including controlling hardware devices such as sensors and actuators, and how to use popular platforms such as Raspberry Pi and Arduino.

To build an IoT application with JavaScript, we need to use a hardware platform that supports it. Two popular hardware platforms that support JavaScript are Raspberry Pi and Arduino.

* **Problem Definition and Data Collection:**
* Clearly define the problem you want to solve.
* Gather relevant data for your project. Ensure data quality and integrity.
* **Exploratory Data Analysis (EDA):**
* Understand the structure of your data.
* Visualize and analyze data to gain insights.
* Handle missing data and outliers as needed.
* **Feature Engineering:**
* Identify and create relevant features from the data.
* Transform, scale, or encode features as required.
* Consider domain-specific knowledge for feature selection.
* **Data Preprocessing:**
* Split the data into training, validation, and test sets.
* Normalize or standardize features if needed.
* Encode categorical variables.
* **Model Selection:**
* Choose an appropriate machine learning algorithm for your problem.
* Consider different models and techniques (e.g., regression, decision trees, neural networks) based on your data.
* **Model Training:**
* Train your chosen model using the training data.
* Tune hyperparameters to optimize model performance.
* **Model Evaluation:**
* Evaluate the model using the validation dataset.
* Choose evaluation metrics relevant to your problem (e.g., accuracy, precision, recall, F1-score, RMSE).
* Consider cross-validation to assess model generalization.
* **Model Fine-Tuning:**
* Adjust the model or hyperparameters based on evaluation results.
* Guard against overfitting or underfitting.
* **Final Model Selection:**
* Choose the best-performing model based on validation results.
* **Model Testing:**
* Assess the selected model's performance on the test dataset.
* This helps ensure your model's generalization to unseen data.
* **Deployment (if applicable):**
* If your project involves deploying a model, integrate it into the production environment.
* **Documentation:**
* Document your process, including data sources, preprocessing steps, and model details.
* **Reporting and Visualization:**
* Communicate your findings and results through reports or presentations.
* **Project Submission:**
* If this is part of an assignment or competition, follow the specific submission instructions, which may involve uploading your code and documentation to a platform like GitHub.

Please provide more specific details about your project, and I can offer more targeted guidance on any of these steps.



#define BLYNK\_TEMPLATE\_ID "TMPL26V4fGv5q"

#define BLYNK\_TEMPLATE\_NAME "Test"

#define BLYNK\_AUTH\_TOKEN "XEHxNF\_Ur1Nt2p7wB5B20dNI1ZUwj34P"

#include <WiFi.h>

#include <WiFiClient.h>

#include <BlynkSimpleEsp32.h>

int duration1 = 0;

int distance1 = 0;

int duration2 = 0;

int distance2 = 0;

int dis1 = 0;

int dis2 = 0;

int dis\_new1 = 0;

int dis\_new2 = 0;

int entered = 0;

int left = 0;

int inside = 0;

#define LED 2

#define PIN\_TRIG1 15

#define PIN\_ECHO1 14

#define PIN\_TRIG2 13

#define PIN\_ECHO2 12

BlynkTimer timer;

char auth[] = BLYNK\_AUTH\_TOKEN;

char ssid[] = "Wokwi-GUEST";   // your network SSID (name)

char pass[] = "";

#define BLYNK\_PRINT **Serial**

long get\_distance1() {

  // Start a new measurement:

  digitalWrite(PIN\_TRIG1, HIGH);

  delayMicroseconds(10);

  digitalWrite(PIN\_TRIG1, LOW);

  // Read the result:

  duration1 = pulseIn(PIN\_ECHO1, HIGH);

  distance1 = duration1 / 58;

  return distance1;

}

long get\_distance2() {

  // Start a new measurement:

  digitalWrite(PIN\_TRIG2, HIGH);

  delayMicroseconds(10);

  digitalWrite(PIN\_TRIG2, LOW);

  // Read the result:

  duration2 = pulseIn(PIN\_ECHO2, HIGH);

  distance2 = duration2 / 58;

  return distance2;

}

void myTimer() {

**Serial**.println("100");

  dis\_new1 = get\_distance1();

  dis\_new2 = get\_distance2();

  if (dis1 != dis\_new1 || dis2 != dis\_new2){

**Serial**.println("200");

    if (dis1 < dis2){

**Serial**.println("Enter loop");

      entered = entered + 1;

      inside = inside + 1;

      digitalWrite(LED, HIGH);

      Blynk.virtualWrite(V0, entered);

      Blynk.virtualWrite(V2, inside);

      dis1 = dis\_new1;

      delay(1000);

      digitalWrite(LED, LOW);

    }

    if (dis1 > dis2){

**Serial**.println("Leave loop");

      left = left + 1;

      inside = inside - 1;

      Blynk.virtualWrite(V1, left);

      Blynk.virtualWrite(V2, inside);

      dis2 = dis\_new2;

      delay(1000);

    }

  }

}

 void setup() {

**Serial**.begin(115200);

  pinMode(LED, OUTPUT);

  pinMode(PIN\_TRIG1, OUTPUT);

  pinMode(PIN\_ECHO1, INPUT);

  pinMode(PIN\_TRIG2, OUTPUT);

  pinMode(PIN\_ECHO2, INPUT);

  Blynk.begin(auth, ssid, pass, "blynk.cloud", 8080);

  timer.setInterval(1000L, myTimer);

}

void loop() {

  Blynk.run();

  timer.run();

}