AEC LAB REPORT ON

AI Tools, Frameworks & Its Application I
[22AML47]

Submitted by:
Sourabha K H, USN: 1GA22AI050

Under the guidance of:
Prof. Udaya Prasad P K
Department of
Artificial Intelligence & Machine Learning

August, 2024

Department of Artificial Intelligence & Machine Learning
Global Academy of Technology
Aditya Layout, Rajarajeshwari Nagar,
Bengaluru, Karnataka 560098

Report on Arduino Projects

Rainbow LED Chaser

Introduction

The Rainbow LED Chaser is a project designed to demonstrate the use of an Arduino to control multiple LEDs. By programming seven LEDs in the colors of the rainbow to flash sequentially, the project simulates a light-chasing effect. This project is particularly beneficial for beginners, as it introduces basic concepts of electronics, coding, and Arduino microcontroller programming.

Project Description

The goal of this project is to create a colorful light display using seven LEDs arranged in a row, each representing a color of the rainbow. The LEDs light up in sequence from left to right and then reverse, creating a chaser effect. An Arduino Uno board is used to control the LEDs.

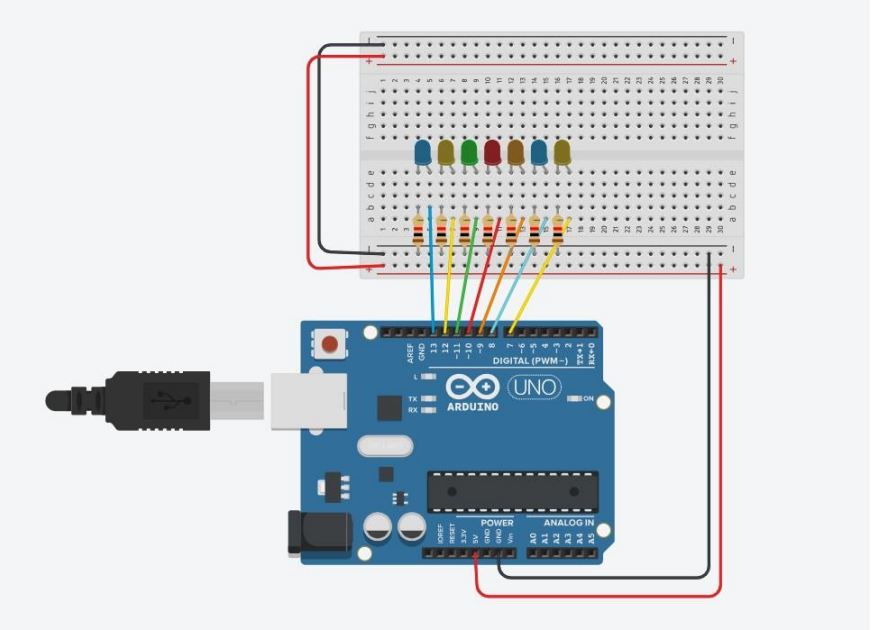
The LEDs are connected to the Arduino through resistors, which limit the current to prevent the LEDs from burning out. The Arduino is programmed to turn on each LED in sequence, wait for a short delay, and then turn it off before moving to the next LED. After all LEDs have lit up from left to right, the sequence reverses, creating the appearance of the lights "chasing" each other back and forth.

Components Needed

- **1 x Arduino Uno board:** The central microcontroller used to program and control the LEDs.
- **7 x LEDs (Red, Orange, Yellow, Green, Blue, Indigo, Violet):** The visual components that create the rainbow effect.
- **7 resistors:** Used to limit the current to the LEDs, typically 220 ohms each.
- **Breadboard:** A board used to prototype the circuit without soldering.

- **Jumper wires:** Used to make the electrical connections between the Arduino, LEDs, and resistors.

Circuit Diagram



Code Implementation

```
const int pins[] = {13, 12, 11, 10, 9,8, 7, 6, };
const int num_pins = sizeof(pins) / sizeof(pins[0]);

void setup() {
  for (int i = 0; i < num_pins; ++i) {
    pinMode(pins[i], OUTPUT);
  }
}

void loop() {
  for (int i = 0; i < num_pins; ++i) {
    digitalWrite(pins[i], HIGH);
    delay(200);
    digitalWrite(pins[i], LOW);
  }

  // Reverse direction
```

```
  for (int i = num_pins - 2; i > 0; --i) {
    digitalWrite(pins[i], HIGH);
    delay(200);
    digitalWrite(pins[i], LOW);
  }
}
```

Learning Outcomes

This project provided hands-on experience with Arduino programming, particularly in controlling multiple LEDs using digital pins. It reinforced the understanding of basic electronics, such as the importance of using resistors to protect components, and improved skills in coding sequences. Additionally, it highlighted the significance of careful planning and testing when designing and implementing circuits, ensuring that each component functions correctly within the system.

Conclusion

The Rainbow LED Chaser project serves as an educational exercise that effectively combines coding, electronics, and circuit design. It offers a strong foundation in controlling multiple outputs with an Arduino and demonstrates the practical applications of sequencing LEDs for visual effects. The knowledge and skills acquired through this project are fundamental for pursuing more advanced projects in the field of electronics and embedded systems.

LED and Push Button Application

Introduction

The LED and Push Button Application project is designed to introduce the use of digital inputs and outputs with an Arduino. By connecting two LEDs and a push button to the Arduino, this project creates an interactive application where the behavior of the LEDs changes based on the state of the push button. This project serves as a practical demonstration of how user interactions can be used to control multiple components in an embedded system.

Project Description

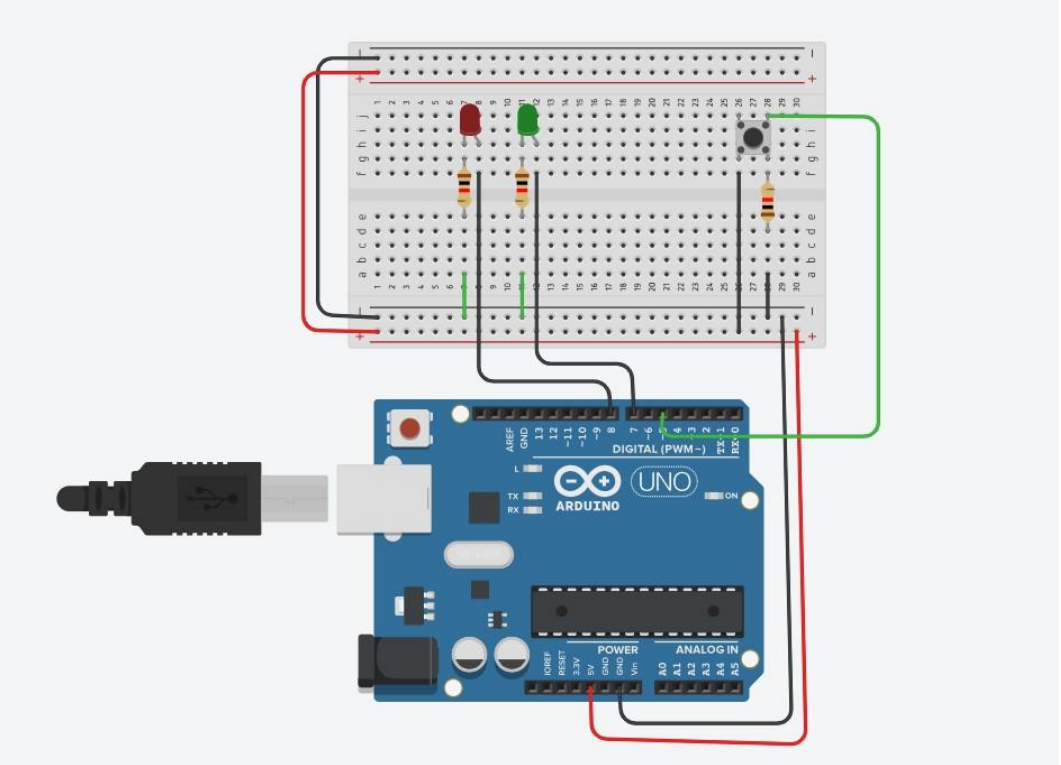
In this project, two LEDs are connected to an Arduino Uno board, along with a push button that controls their behavior. Initially, both LEDs remain off. When the push button is pressed and released, the LEDs turn on. A second press and release of the button will turn the LEDs off. Additionally, if the button is pressed and held, the LEDs will alternate between their on and off states, creating a blinking effect.

The Arduino is programmed to detect the state of the push button and control the LEDs accordingly. The use of resistors ensures that the current flowing through the LEDs and push button is appropriately limited to prevent damage to the components. This project illustrates the fundamental concept of using digital inputs (the push button) to control digital outputs (the LEDs) based on user interactions.

Components Needed

- **1 x Arduino Uno board:** The central microcontroller used to program and control the LEDs and push button.
- **2 x LEDs (any color):** The visual components that will indicate the state of the system.
- **1 x Push button:** The input device used to control the behavior of the LEDs.
- **3 x 220-ohm resistors:** Used to limit the current to the LEDs and push button.
- **Breadboard:** A board used to prototype the circuit without soldering.
- **Jumper wires:** Used to make the electrical connections between the Arduino, LEDs, push button, and resistors.

Circuit Diagram



Code Implementation

```
const int buttonPin = 5;
const int ledPin1 = 8;
const int ledPin2 = 7;

int buttonState = 0;
int lastButtonState = LOW;
unsigned long pressStartTime = 0;

void setup() {
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  buttonState = digitalRead(buttonPin);

  // Check for button press and release
  if (buttonState != lastButtonState) {
    if (buttonState == HIGH) {
      pressStartTime = millis();  // Record the start time
of the press
    } else {
      unsigned long pressDuration = millis() - pressStartTi
me;

      if (pressDuration < 1000) {
        toggleLEDs();  // Short press: Toggle LEDs
      } else {
        flickerLEDs();  // Long press: Flicker LEDs
      }
    }
  }

  lastButtonState = buttonState;  // Save the current state
of the button
}
```

```
void toggleLEDs() {
  static bool ledsOn = false;

  if (ledsOn) {
    digitalWrite(ledPin1, LOW);
    digitalWrite(ledPin2, LOW);
    ledsOn = false;
  } else {
    digitalWrite(ledPin1, HIGH);
    digitalWrite(ledPin2, HIGH);
    ledsOn = true;
  }
}

void flickerLEDs() {
  unsigned long flickerStartTime = millis();  // Record the
start time of flickering

  while (millis() - flickerStartTime < 3000) {  // Flicker
for 3 seconds
    digitalWrite(ledPin1, random(2) == 0 ? HIGH : LOW);
    digitalWrite(ledPin2, random(2) == 0 ? HIGH : LOW);
    delay(random(50, 150));  // Random delay between 50ms t
o 150ms
  }

  digitalWrite(ledPin1, LOW);  // Ensure LEDs are off after
flickering
  digitalWrite(ledPin2, LOW);
}
```

Learning Outcomes

This project provided valuable insights into the use of digital inputs and outputs with an Arduino microcontroller. The process of designing and implementing the circuit reinforced the understanding of how to use push buttons as input devices and LEDs as output indicators. Additionally, the project emphasized the importance of debouncing techniques to handle the mechanical noise associated with push buttons. This project successfully demonstrated how user interactions can be effectively used to control multiple components in a system.

Conclusion

The LED and Push Button Application project serves as an essential exercise in understanding the interaction between digital inputs and outputs. It illustrates the practical applications of controlling LEDs based on user input, providing a foundation for more complex projects involving multiple inputs and outputs. The skills and concepts acquired through this project are critical for further exploration in the field of embedded systems and interactive applications.

LED-RGB and Ultrasonic Distance Application

Introduction

The LED-RGB and Ultrasonic Distance Application project introduces the concept of using sensors to create interactive and responsive systems. By integrating an RGB LED with two ultrasonic distance sensors, the project demonstrates how to control the color of an LED based on proximity detection. This project serves as an educational tool for understanding the use of sensors in electronics and programming microcontrollers to respond to environmental inputs.

Project Description

In this project, an RGB LED is connected to an Arduino Uno board, along with two ultrasonic distance sensors (HC-SR04). The RGB LED is programmed to display two fixed colors. The ultrasonic sensors are used to measure distances, and the color of the RGB LED changes based on which sensor detects a closer object. If the first sensor detects an object closer than the second sensor, the RGB LED will display one color, and if the second sensor detects a closer object, the LED will display a different color.

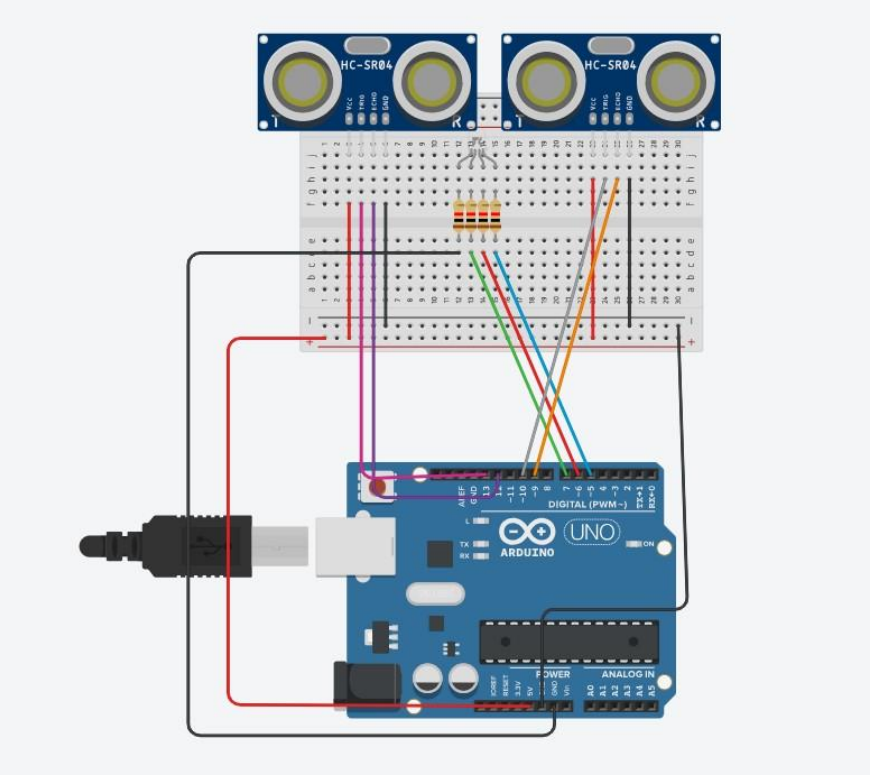
This project involves the use of digital pins on the Arduino to control the RGB LED and read data from the ultrasonic sensors. The resistors are used to limit the current to the RGB LED, ensuring that it operates safely. The project provides a practical example of how sensors can be used to create interactive systems that respond to changes in the environment.

Components Needed

- **1 x Arduino Uno board:** The microcontroller used to program and control the RGB LED and ultrasonic sensors.
- **1 x RGB LED:** The visual component that changes color based on the distance detected by the sensors.

- **2 x Ultrasonic distance sensors (HC-SR04):** Sensors used to measure the distance of objects from the sensor.
- **3 x 220-ohm resistors:** Used to limit the current to the RGB LED.
- **Breadboard:** A board used to prototype the circuit without soldering.
- **Jumper wires:** Used to make the electrical connections between the Arduino, RGB LED, ultrasonic sensors, and resistors.

Circuit Diagram



Code Implementation

```
// Pins for ultrasonic sensors
const int triggerPin1 = 10;
const int echoPin1 = 9;
const int triggerPin2 = 12;
const int echoPin2 = 13;

// Pins for RGB LED
const int redPin = 7;
const int greenPin = 6;
```

```
const int bluePin = 5;

// Conversion factor for microseconds to centimeters
const float conversionFactor = 0.01723;

// Variables to store previous distances
int prevSensor1Distance = 0;
int prevSensor2Distance = 0;

void setup() {
  Serial.begin(9600);

  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);

  pinMode(triggerPin1, OUTPUT);
  pinMode(echoPin1, INPUT);
  pinMode(triggerPin2, OUTPUT);
  pinMode(echoPin2, INPUT);
}

void loop() {
  // Read distances from ultrasonic sensors
  int sensor1Distance = conversionFactor * readUltrasonicDi
stance(triggerPin1, echoPin1);
  int sensor2Distance = conversionFactor * readUltrasonicDi
stance(triggerPin2, echoPin2);

  // Print distances to Serial Monitor
  Serial.print("Distance from Sensor 1: ");
  Serial.print(sensor1Distance);
  Serial.println(" cm");
  Serial.print("Distance from Sensor 2: ");
  Serial.print(sensor2Distance);
  Serial.println(" cm");

  // Check if distances have changed significantly
```

```
  if (abs(sensor1Distance - prevSensor1Distance) > 10 || ab
s(sensor2Distance - prevSensor2Distance) > 10) {
    // Change RGB LED color based on the frequency of dista
nce changes
    setColor(0, 255, 0); // Green
  } else {
    setColor(0, 0, 255); // Blue
  }

  // Update previous distances
  prevSensor1Distance = sensor1Distance;
  prevSensor2Distance = sensor2Distance;

  delay(500); // Wait for 500 milliseconds
}

// Function to read distance from an ultrasonic sensor
long readUltrasonicDistance(int triggerPin, int echoPin) {
  digitalWrite(triggerPin, LOW);
  delayMicroseconds(2);
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);

  return pulseIn(echoPin, HIGH);
}

// Function to set RGB LED color
void setColor(int redValue, int greenValue, int blueValue)
{
  analogWrite(redPin, redValue);
  analogWrite(greenPin, greenValue);
  analogWrite(bluePin, blueValue);
}
```

Learning Outcomes

This project demonstrated the practical application of ultrasonic sensors in measuring distance and controlling an RGB LED based on proximity. The integration of sensors with an

Arduino microcontroller highlighted the importance of understanding sensor data and using it to make real-time decisions in embedded systems. The project also reinforced the concept of using digital pins for both input and output operations, and the necessity of correctly managing electrical components to ensure the system's reliability.

Conclusion

The LED-RGB and Ultrasonic Distance Application project successfully combined the use of sensors and LED control in an interactive system. This project serves as an effective introduction to sensor integration and responsive programming in embedded systems. The skills and knowledge gained through this project are foundational for developing more complex systems that interact with their environment based on sensor inputs.

Piezo Sensor, IR Sensor, and Servo Motor Application

Introduction

The Piezo Sensor, IR Sensor, and Servo Motor project explores the integration of multiple sensors and actuators to create a responsive system. This project simulates a smart doorbell system that alerts homeowners of visitors and potential intruders, demonstrating the practical application of sensors and motors in home automation and security systems.

Project Description

In this project, an Arduino Uno board is connected to an IR sensor, a piezo sensor (buzzer), and a servo motor to create a smart doorbell system. The IR sensor is used to detect motion near a door. When motion is detected, the piezo sensor plays a doorbell sound, alerting the homeowner. Simultaneously, the servo motor is programmed to open the door slightly if the detected motion matches a recognized pattern, simulating the door being answered.

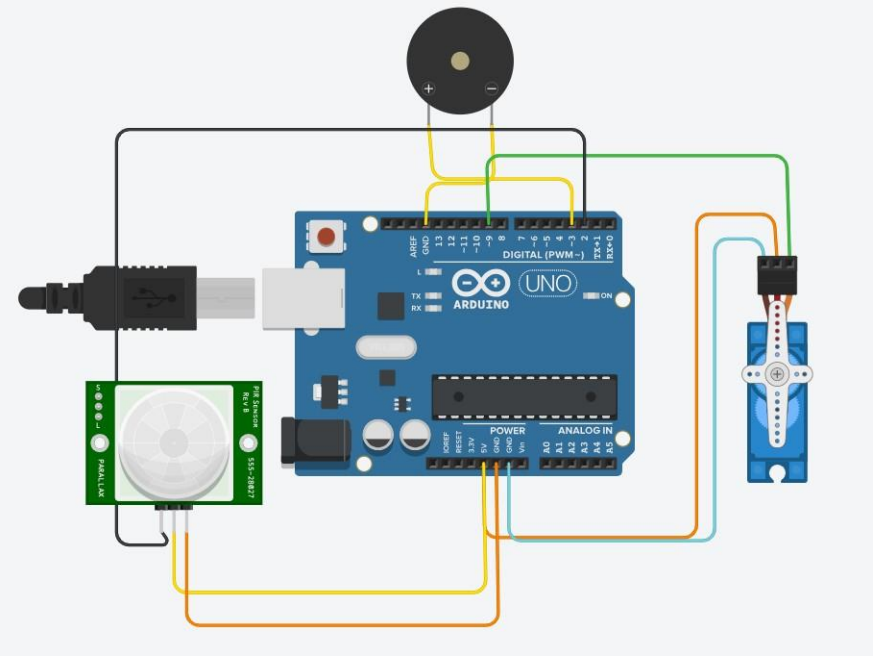
This project illustrates how to use an IR sensor to detect proximity and how to trigger actions based on sensor input. The piezo sensor provides audio feedback, while the servo motor offers a physical response, showcasing the integration of various components to build an interactive system.

Components Needed

- **1 x Arduino Uno board:** The microcontroller used to control the sensors and servo motor.
- **1 x Piezo Sensor (Buzzer):** Generates an audible sound when triggered by the IR sensor.
- **1 x IR Sensor:** Detects motion near the door and sends a signal to the Arduino.

- **1 x Servo Motor:** Controls the movement of the door, simulating it being opened.
- **Breadboard:** Used to assemble the circuit without soldering.
- **Jumper wires:** Facilitate the connections between the components and the Arduino.
- **Resistors (as needed):** Used to manage current flow in the circuit.

Circuit Diagram



Code Implementation

```
#include <Servo.h>

const int pirSensorPin = 2; // Pin connected to the PIR sen
sor OUT
const int piezoPin = 3;     // Pin connected to the piezo b
uzzer
const int servoPin = 9;     // Pin connected to the servo m
otor

Servo myServo;              // Create a Servo object

void setup() {
  pinMode(pirSensorPin, INPUT); // Set the PIR sensor pin a
s input
```

```
  pinMode(piezoPin, OUTPUT);    // Set the piezo buzzer pin
as output
  myServo.attach(servoPin);     // Attach the servo motor t
o the specified pin
  myServo.write(0);             // Initialize servo positio
n to closed
}

void loop() {
  int sensorValue = digitalRead(pirSensorPin); // Read the
PIR sensor value

  if (sensorValue == HIGH) { // Motion detected
    tone(piezoPin, 1000); // Play a 1kHz tone
    delay(500);           // Duration of the tone
    noTone(piezoPin);     // Stop the tone

    // Simulate door opening
    myServo.write(90);   // Open door slightly (90 degrees)
    delay(2000);         // Keep the door open for 2 second
s
    myServo.write(0);    // Close the door

    delay(5000);         // Wait for 5 seconds before check
ing again
  }
}
```

Learning Outcomes

This project demonstrated the practical integration of sensors and actuators to create a responsive and interactive system. The use of an IR sensor for motion detection, coupled with the piezo sensor for audio feedback and a servo motor for physical movement, provided insight into building complex systems with multiple components. Additionally, the project emphasized the importance of synchronizing actions between different sensors and actuators to achieve the desired outcome in a real-world application.

Conclusion

The Piezo Sensor, IR Sensor, and Servo Motor project effectively illustrated the integration of multiple sensors and actuators to create a smart doorbell system. This project serves as a practical example of how to build interactive systems that respond to environmental inputs, laying the groundwork for more advanced home automation and security applications.

Photo-resistor Sensor, Temperature Sensor, and DC Motor Application

Introduction

The Photo-resistor Sensor, Temperature Sensor, and DC Motor project demonstrate the creation of an automated room lighting and temperature control system. This project highlights the use of sensors to monitor environmental conditions and the integration of actuators to respond dynamically, thereby enhancing comfort and energy efficiency.

Project Description

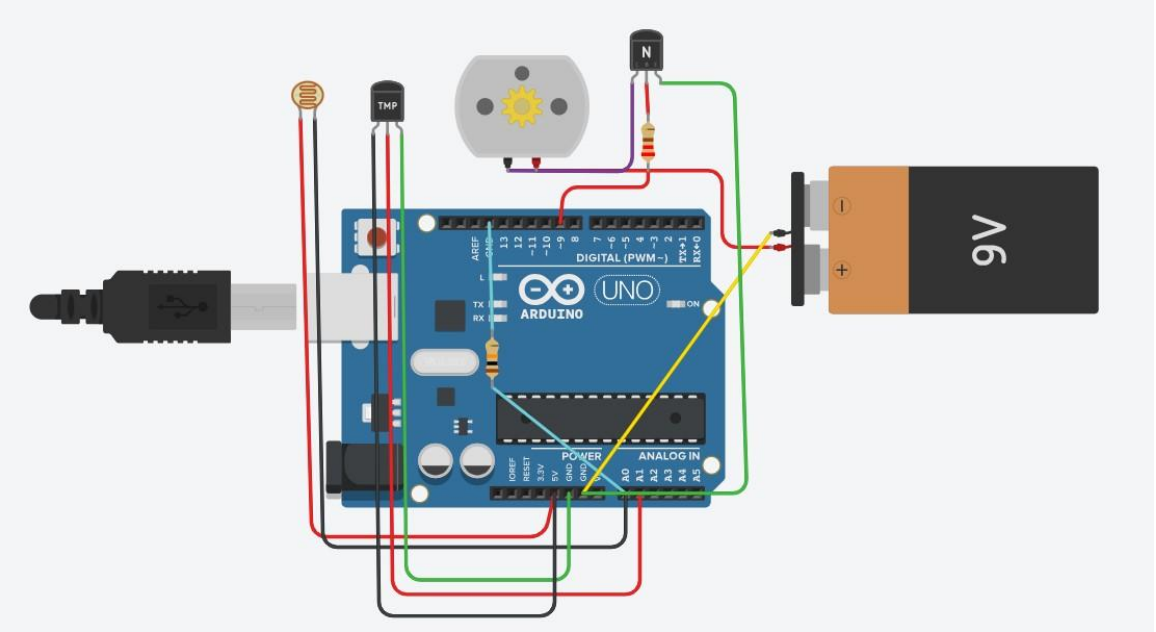
In this project, an Arduino Uno board is utilized to automate the control of room lighting and temperature. A photo-resistor sensor (LDR) is used to adjust the brightness of the room lights based on the ambient light levels, ensuring that the room is always adequately lit without wasting energy. A temperature sensor (LM35) monitors the room temperature and controls a DC motor, which acts as a fan, to maintain a comfortable environment. The system responds to real-time changes in light and temperature, providing a practical application of sensors and actuators in creating an interactive and energy-efficient environment.

Components Needed

- **1 x Arduino Uno board:** The microcontroller that controls the sensors and the motor.
- **1 x Photo-resistor Sensor (LDR):** Measures the ambient light levels to adjust the room lighting.
- **1 x Temperature Sensor (LM35):** Monitors the room temperature to control the fan.
- **1 x DC Motor (for fan control):** Acts as the fan to regulate the room temperature.
- **1 x Transistor (e.g., NPN transistor like 2N2222):** Controls the DC motor.
- **1 x Diode (e.g., 1N4007):** Protects the circuit from potential back EMF generated by the motor.
- **1 x 10k ohm Resistor:** Used with the LDR for voltage division.
- **1 x 220 ohm Resistor:** Used to limit current to the transistor's base.
- **Breadboard:** Provides a platform to assemble the circuit without soldering.

- **Jumper wires:** Facilitate the connections between the components and the Arduino.

Circuit Diagram



Code Implementation

```
// Pin definitions
const int ldrPin = A0;          // LDR connected to analog p
in A0
const int tempSensorPin = A1;  // LM35 connected to analog
pin A1
const int fanPin = 9;           // DC Motor control connecte
d to digital pin 9

// LDR threshold and temperature threshold
const int lightThreshold = 500;  // Adjust this value based
on your lighting needs
const int temperatureThreshold = 25; // Temperature thresho
ld to turn on the fan (in °C)

void setup() {
  pinMode(fanPin, OUTPUT);
  Serial.begin(9600); // Initialize serial communication fo
r debugging
```

```
}

void loop() {
  // Read LDR value
  int ldrValue = analogRead(ldrPin);

  // Map LDR value to PWM range (0-255) for motor speed con
trol
  int motorSpeed = map(ldrValue, 0, 1023, 0, 255);

  // Print LDR value and mapped motor speed to serial monit
or
  Serial.print("LDR Value: ");
  Serial.print(ldrValue);
  Serial.print(" | Motor Speed (0-255): ");
  Serial.println(motorSpeed);

  // Set motor speed based on LDR value
  analogWrite(fanPin, motorSpeed);

  // Read temperature value
  int tempReading = analogRead(tempSensorPin);
  // Convert the reading to Celsius
  float temperature = (tempReading * 5.0 / 1023.0) * 100.0;

  // Print temperature value to serial monitor
  Serial.print("Temperature (°C): ");
  Serial.println(temperature);

  // Control the fan based on temperature
  if (temperature > temperatureThreshold) {
    // Turn fan ON and set speed
    analogWrite(fanPin, motorSpeed);
  } else {
    // Turn fan OFF
    analogWrite(fanPin, 0);
  }
```

```
  // Delay to allow sensor readings to stabilize
  delay(1000);
}
```

Learning Outcomes

This project provided a comprehensive understanding of how to use sensors to monitor environmental conditions and how to control actuators to respond dynamically. By integrating a photo-resistor sensor and a temperature sensor with a DC motor, the project illustrated the principles of automated control systems. The knowledge gained through this project is directly applicable to designing energy-efficient systems for smart homes, where environmental conditions need to be constantly monitored and adjusted to enhance comfort and reduce energy consumption.

Conclusion

The Photo-resistor Sensor, Temperature Sensor, and DC Motor project successfully demonstrated the creation of an automated system for controlling room lighting and temperature. The project highlighted the importance of sensor-actuator integration in developing responsive and energy-efficient systems, providing valuable insights into the practical applications of such technologies in smart environments.

Gas Sensor, PIR Sensor, and DC Motor Application

Introduction

The Gas Sensor, PIR Sensor, and DC Motor project involves developing a gas leak detection and emergency ventilation system. This project showcases how sensors and actuators can be integrated to respond effectively to hazardous conditions, ensuring safety by activating ventilation in case of gas leaks and detecting room occupancy.

Project Description

This project utilizes an Arduino Uno board to create a safety system that detects harmful gas levels and manages ventilation. The gas sensor detects the presence of hazardous gases, while the PIR sensor checks for room occupancy. When the gas sensor identifies dangerous gas levels and the PIR sensor confirms that the room is occupied, the DC motor, which controls a fan, is activated to ventilate the area and clear the gas. This system provides a practical application of sensor technology and actuator control for enhancing safety in environments where gas leaks are a concern.

Components Needed

- **1 x Arduino Uno board:** The central microcontroller that processes sensor data and controls the fan.
- **1 x Gas Sensor (MQ-2 or MQ-135):** Detects the presence of hazardous gases in the environment.
- **1 x PIR Sensor:** Detects motion and checks for room occupancy.
- **1 x DC Motor (for fan control):** Activated to ventilate the room when gas levels are detected and the room is occupied.
- **1 x Transistor (e.g., NPN transistor like 2N2222):** Controls the operation of the DC motor.
- **1 x Diode (e.g., 1N4007):** Protects the circuit from potential back EMF generated by the motor.
- **1 x 10k ohm Resistor:** Used with the gas sensor for proper voltage readings.
- **1 x 220 ohm Resistor:** Used to limit current to the transistor's base.
- **Breadboard:** Provides a platform to assemble the circuit without soldering.
- **Jumper wires:** Facilitate the connections between the components and the Arduino.

Circuit Diagram

(Insert Circuit Diagram Here)

Code Implementation

```
int gasPin = A0;        // Gas sensor connected to analog p
in A0
int pirPin = 2;         // PIR sensor connected to digital
pin 2
int motorPin = 9;       // DC motor connected to digital pi
n 9

void setup() {
  pinMode(motorPin, OUTPUT);
  pinMode(pirPin, INPUT);
  Serial.begin(9600);  // Initialize serial communication f
or debugging
```

```
}

void loop() {
  int gasValue = analogRead(gasPin);   // Read the value fr
om the gas sensor
  int pirState = digitalRead(pirPin);  // Read the state fr
om the PIR sensor

  // If gas levels are above a certain threshold and the ro
om is occupied
  if (gasValue > 400 && pirState == HIGH) {
    digitalWrite(motorPin, HIGH); // Turn on fan to ventila
te the room
  } else {
    digitalWrite(motorPin, LOW);  // Turn off fan if gas le
vels are normal or room is unoccupied
  }

  delay(1000);  // Wait for a second before the next readin
g
}
```

Learning Outcomes

This project provided insights into the integration of multiple sensors and actuators to address safety concerns effectively. Students learned how to implement a gas detection system combined with occupancy sensing and fan control, enhancing their understanding of real-world applications of Arduino technology in safety systems. The project highlighted the importance of accurate sensor readings and prompt response mechanisms in creating functional and reliable safety solutions.

Conclusion

The Gas Sensor, PIR Sensor, and DC Motor project successfully demonstrated the development of a safety system for detecting gas leaks and managing ventilation. By integrating a gas sensor with a PIR sensor and a DC motor, the project showcased how to create a responsive and effective safety system. The skills acquired from this project are applicable to designing and implementing similar systems in various settings where safety and environmental control are critical.

Gas Sensor, PIR Sensor, and DC Motor Application

Introduction

The Gas Sensor, PIR Sensor, and DC Motor project involves developing a gas leak detection and emergency ventilation system. This project showcases how sensors and actuators can be integrated to respond effectively to hazardous conditions, ensuring safety by activating ventilation in case of gas leaks and detecting room occupancy.

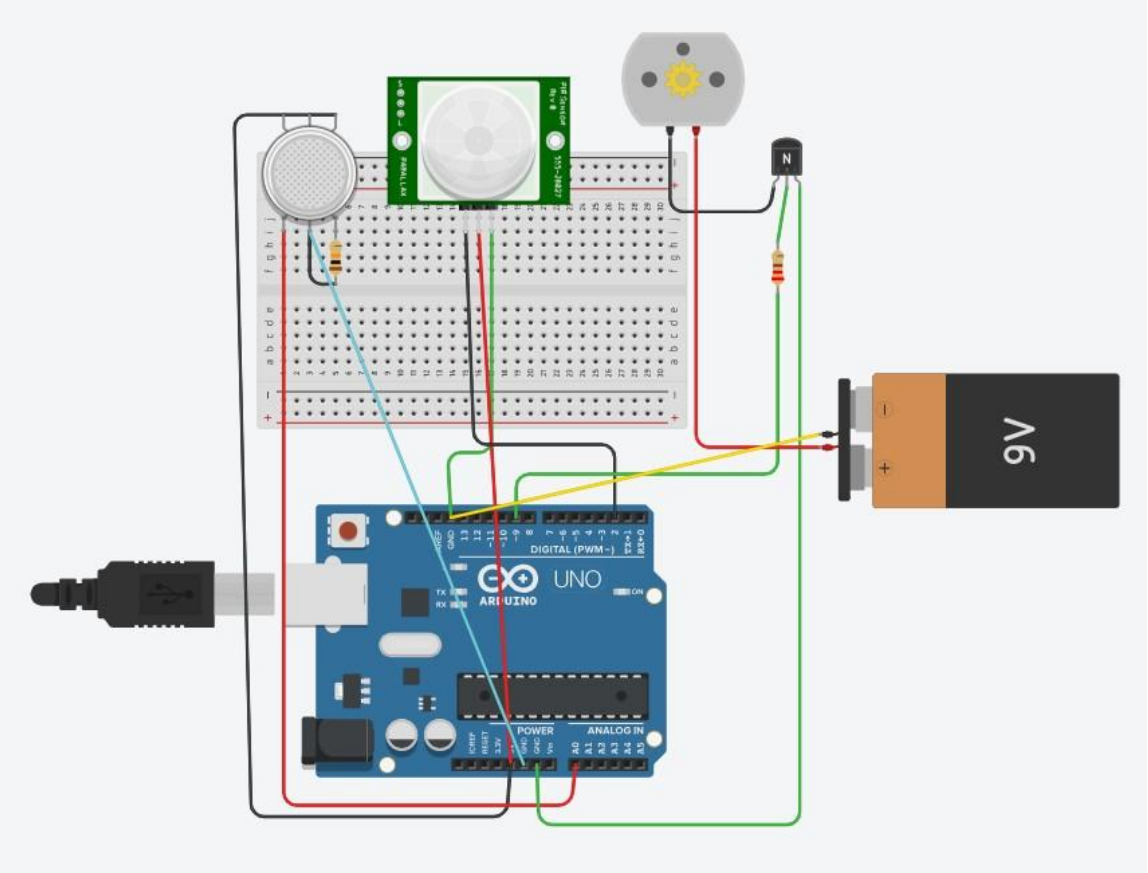
Project Description

This project utilizes an Arduino Uno board to create a safety system that detects harmful gas levels and manages ventilation. The gas sensor detects the presence of hazardous gases, while the PIR sensor checks for room occupancy. When the gas sensor identifies dangerous gas levels and the PIR sensor confirms that the room is occupied, the DC motor, which controls a fan, is activated to ventilate the area and clear the gas. This system provides a practical application of sensor technology and actuator control for enhancing safety in environments where gas leaks are a concern.

Components Needed

- **1 x Arduino Uno board:** The central microcontroller that processes sensor data and controls the fan.
- **1 x Gas Sensor (MQ-2 or MQ-135):** Detects the presence of hazardous gases in the environment.
- **1 x PIR Sensor:** Detects motion and checks for room occupancy.
- **1 x DC Motor (for fan control):** Activated to ventilate the room when gas levels are detected and the room is occupied.
- **1 x Transistor (e.g., NPN transistor like 2N2222):** Controls the operation of the DC motor.
- **1 x Diode (e.g., 1N4007):** Protects the circuit from potential back EMF generated by the motor.
- **1 x 10k ohm Resistor:** Used with the gas sensor for proper voltage readings.
- **1 x 220 ohm Resistor:** Used to limit current to the transistor’s base.
- **Breadboard:** Provides a platform to assemble the circuit without soldering.
- **Jumper wires:** Facilitate the connections between the components and the Arduino.

Circuit Diagram



Code Implementation

```
const int gasSensorPin = A0;  // Gas Sensor connected to An
alog Pin A0
const int pirSensorPin = 2;   // PIR Sensor connected to Di
gital Pin 2
const int motorPin = 9;       // DC Motor controlled by Dig
ital Pin 9

void setup() {
  Serial.begin(9600);       // Start serial communication a
t 9600 baud
  pinMode(pirSensorPin, INPUT);
  pinMode(motorPin, OUTPUT);
}
```

```
void loop() {
  int gasValue = analogRead(gasSensorPin); // Read gas sens
or value
  int pirValue = digitalRead(pirSensorPin); // Read PIR sen
sor value

  Serial.print("Gas Level: ");
  Serial.print(gasValue);
  Serial.print(" PIR Status: ");
  Serial.println(pirValue);

  // Check for high gas level and room occupancy
  if (gasValue > 500 && pirValue == HIGH) {
    digitalWrite(motorPin, HIGH); // Activate the motor (fa
n)
  } else {
    digitalWrite(motorPin, LOW);  // Deactivate the motor
(fan)
  }

  delay(1000); // Delay for 1 second
}
```

Learning Outcomes

This project provided insights into the integration of multiple sensors and actuators to address safety concerns effectively. Students learned how to implement a gas detection system combined with occupancy sensing and fan control, enhancing their understanding of real-world applications of Arduino technology in safety systems. The project highlighted the importance of accurate sensor readings and prompt response mechanisms in creating functional and reliable safety solutions.

Conclusion

The Gas Sensor, PIR Sensor, and DC Motor project successfully demonstrated the development of a safety system for detecting gas leaks and managing ventilation. By integrating a gas sensor with a PIR sensor and a DC motor, the project showcased how to create a responsive and effective safety system. The skills acquired from this project are

applicable to designing and implementing similar systems in various settings where safety and environmental control are critical.

IR Sensor, Temperature Sensor, and Servo Motor Application

Introduction

The IR Sensor, Temperature Sensor, and Servo Motor project focuses on developing a smart irrigation system for gardens using Arduino. This system automates the watering process based on soil moisture and ambient temperature, optimizing water usage and maintaining ideal conditions for plant growth.

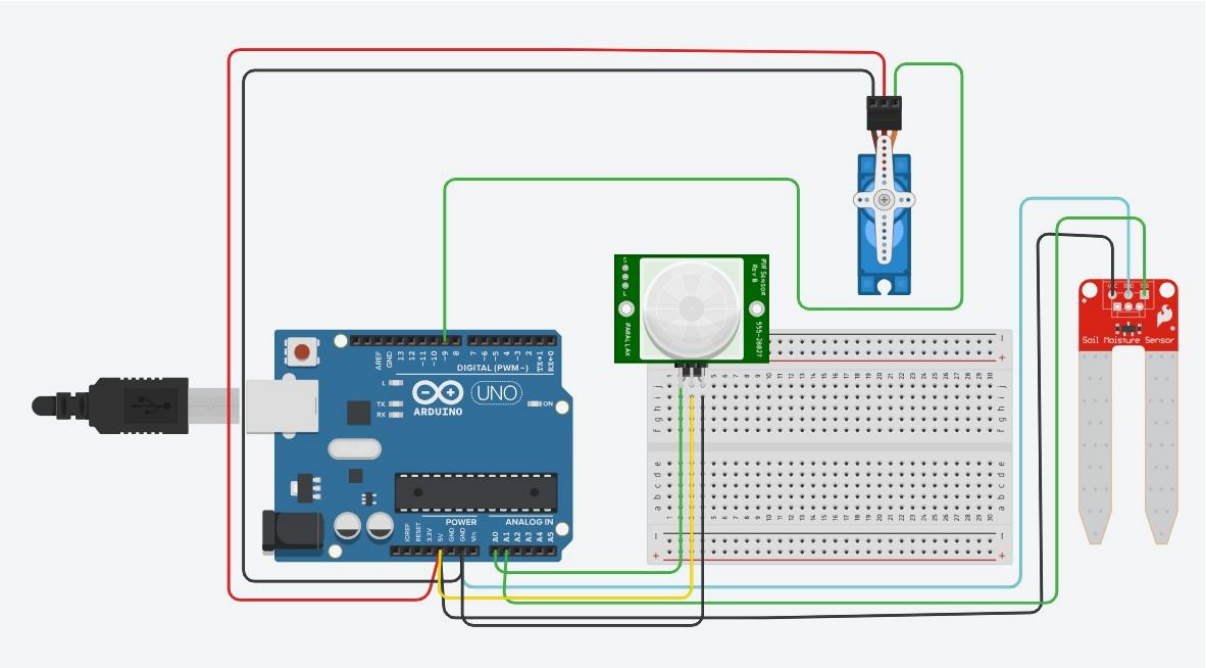
Project Description

This project employs an Arduino Uno board to create an automated irrigation system. The IR sensor (used as a soil moisture sensor) monitors the moisture level of the soil, while the temperature sensor measures the ambient temperature. Based on the data from these sensors, a servo motor controls a water valve to provide water only when necessary. This system is designed to conserve water and ensure that plants receive adequate hydration based on environmental conditions, demonstrating the integration of sensors and actuators to automate and optimize garden irrigation.

Components Needed

- **1 x Arduino Uno board:** The microcontroller that processes sensor data and controls the servo motor.
- **1 x IR Sensor (used as a soil moisture sensor):** Measures soil moisture levels to determine when watering is needed.
- **1 x Temperature Sensor (LM35):** Monitors ambient temperature to adjust watering times.
- **1 x Servo Motor:** Controls the water valve to regulate water flow.
- **Breadboard:** Provides a platform for assembling the circuit without soldering.
- **Jumper wires:** Facilitate connections between components and the Arduino.
- **Resistors (as needed):** Ensure proper operation of sensors and prevent damage.

Circuit Diagram



Code Implementation

```
// Pin definitions
const int ldrPin = A0;          // LDR connected to analog p
in A0
const int tempSensorPin = A1;  // LM35 connected to analog
pin A1
const int fanPin = 9;           // DC Motor control connecte
d to digital pin 9

// LDR threshold and temperature threshold
const int lightThreshold = 500;  // Adjust this value based
on your lighting needs
const int temperatureThreshold = 25; // Temperature thresho
ld to turn on the fan (in °C)

void setup() {
  pinMode(fanPin, OUTPUT);
  Serial.begin(9600); // Initialize serial communication fo
r debugging
}

void loop() {
```

```
  // Read LDR value
  int ldrValue = analogRead(ldrPin);

  // Map LDR value to PWM range (0-255) for motor speed con
trol
  int motorSpeed = map(ldrValue, 0, 1023, 0, 255);

  // Print LDR value and mapped motor speed to serial monit
or
  Serial.print("LDR Value: ");
  Serial.print(ldrValue);
  Serial.print(" | Motor Speed (0-255): ");
  Serial.println(motorSpeed);

  // Set motor speed based on LDR value
  analogWrite(fanPin, motorSpeed);

  // Read temperature value
  int tempReading = analogRead(tempSensorPin);
  // Convert the reading to Celsius
  float temperature = (tempReading * 5.0 / 1023.0) * 100.0;

  // Print temperature value to serial monitor
  Serial.print("Temperature (°C): ");
  Serial.println(temperature);

  // Control the fan based on temperature
  if (temperature > temperatureThreshold) {
    // Turn fan ON and set speed
    analogWrite(fanPin, motorSpeed);
  } else {
    // Turn fan OFF
    analogWrite(fanPin, 0);
  }

  // Delay to allow sensor readings to stabilize
  delay(1000);
}
```

Learning Outcomes

This project enhanced understanding of integrating various sensors and actuators to create an automated system. Key learnings include the practical application of soil moisture and temperature sensing in automated irrigation systems, effective use of servo motors for controlling mechanical components, and the importance of maintaining optimal environmental conditions for plant growth.

Conclusion

The IR Sensor, Temperature Sensor, and Servo Motor project effectively demonstrated the creation of an automated garden irrigation system. By integrating soil moisture and temperature sensors with a servo motor-controlled water valve, the system efficiently managed water usage and supported plant health. The skills gained from this project are valuable for designing and implementing similar automated systems in various applications where environmental control and resource conservation are critical.

PIR Sensor, Piezo Sensor, and Photo-resistor Sensor Application

Introduction

The PIR Sensor, Piezo Sensor, and Photo-resistor Sensor project aims to create an intruder alert system using Arduino. This system is designed to detect both motion and changes in ambient light to activate an alarm, enhancing security by providing a responsive alert mechanism.

Project Description

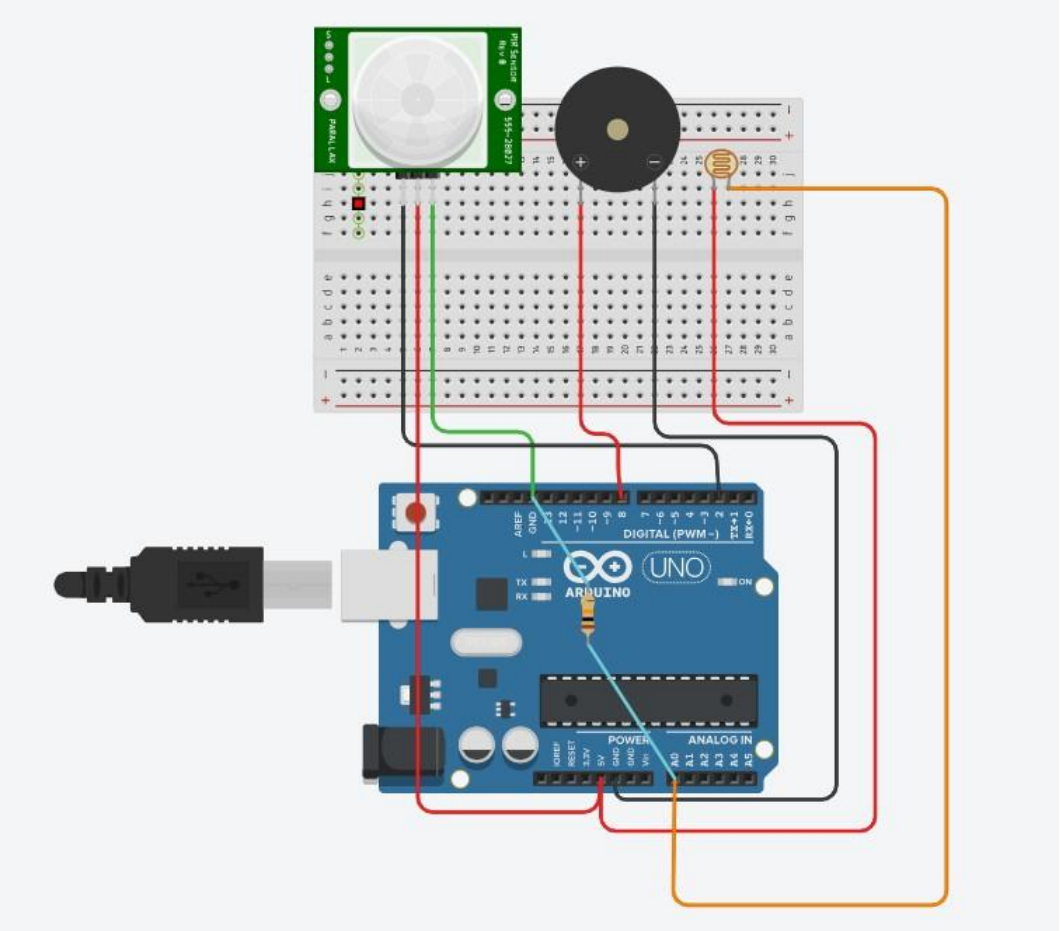
The project involves developing an intruder detection system that integrates a PIR sensor, photo-resistor sensor, and piezo sensor. The PIR sensor monitors for motion, while the photo-resistor sensor detects changes in ambient light. When both sensors are triggered, the piezo sensor activates an alarm sound. This system provides a comprehensive approach to security by combining motion detection and light sensing, demonstrating how multiple sensors can work together to create an effective alert system.

Components Needed

- **1 x Arduino Uno board:** Serves as the central controller for the sensors and the alarm system.
- **1 x PIR Sensor:** Detects motion within the monitored area.
- **1 x Piezo Sensor (Buzzer):** Produces an audible alarm when both sensors are triggered.

- **1 x Photo-resistor Sensor (LDR):** Senses changes in ambient light levels.
- **Breadboard:** Allows for easy assembly and connection of components.
- **Jumper wires:** Facilitate connections between components and the Arduino.
- **Resistors (as needed):** Ensure proper operation of sensors and prevent damage.

Circuit Diagram



Code Implementation

```
// Define pin connections
const int pirPin = 2;         // PIR sensor output pin
const int buzzerPin = 8;      // Piezo buzzer pin
const int ldrPin = A0;        // LDR analog input pin

// Define threshold values
const int lightThreshold = 500;  // Adjust this value based
on ambient light conditions
```

```
// Variables to store sensor values
int pirState = LOW;         // Current state of PIR sensor
int ldrValue = 0;           // Current value from LDR

void setup() {
  Serial.begin(9600);  // Initialize serial communication f
or debugging

  pinMode(pirPin, INPUT);  // Set PIR sensor pin as input
  pinMode(buzzerPin, OUTPUT);  // Set buzzer pin as output
}

void loop() {
  pirState = digitalRead(pirPin);  // Read the state of the
PIR sensor
  ldrValue = analogRead(ldrPin);  // Read the value from th
e LDR

  Serial.print("PIR: ");
  Serial.print(pirState);
  Serial.print(" | LDR: ");
  Serial.println(ldrValue);

  if (pirState == HIGH && ldrValue < lightThreshold) {
    tone(buzzerPin, 1000);  // Sound the buzzer at 1000 Hz
  } else {
    noTone(buzzerPin);  // Turn off the alarm
  }

  delay(100);  // Small delay to avoid overwhelming the ser
ial monitor
}
```

Learning Outcomes

This project provided insight into integrating multiple sensors to build a cohesive security system. Key learnings include the effective use of PIR sensors for motion detection, photo-resistor sensors for monitoring light changes, and piezo sensors for generating audible alerts.

The project demonstrates how to combine these components to enhance security and respond to environmental changes effectively.

Conclusion

The PIR Sensor, Piezo Sensor, and Photo-resistor Sensor project successfully created an intruder alert system that leverages motion detection and light sensing. By combining these sensors with an alarm mechanism, the system provides an effective solution for enhancing security. The project highlights the value of integrating various sensors to achieve a comprehensive and responsive alert system, demonstrating practical applications of Arduino in security solutions.

Report on Python Projects: Sentimental Analysis and Resume Screening

Introduction

This report provides an overview of three Python projects: **Sentiment Analysis on COVID-19 Tweets**, **Sentiment Analysis on General Tweets**, and **Automated Resume Screening**. Each project employs Natural Language Processing (NLP) techniques to analyze textual data, focusing on sentiment analysis and resume categorization. The report will cover the objectives, methodologies, libraries used, results, and reflections on the learning experience.

1. Sentiment Analysis on COVID-19 Tweets

Objective

The primary goal of the **Sentiment Analysis on COVID-19 Tweets** project is to analyze sentiments expressed in tweets related to the COVID-19 pandemic. The project aims to understand public sentiment during the pandemic by classifying tweets as positive, negative, or neutral.

Methodology

- **Data Collection**: Utilizes the Twitter API to gather tweets containing keywords related to COVID-19.
- **Data Preprocessing**: Involves cleaning the collected tweets by removing URLs, mentions, hashtags, and punctuation.

- **Sentiment Analysis**: Employs the Natural Language Toolkit (nltk) for tokenization and sentiment classification using a pre-trained model.
- **Visualization**: Uses Matplotlib to present the distribution of sentiments.

Libraries Used

- `nltk`: For natural language processing tasks.
- `pandas`: For data manipulation and analysis.
- `matplotlib`: For data visualization.
- `tweepy`: For accessing the Twitter API.

Objective: To classify resumes into different categories using the Random Forest algorithm.

Steps:

1. **Data Preparation:**
   - The dataset is loaded, and features and labels are segregated.
   - The features consist of the cleaned resume text, while the labels represent the categories.

```
features = dataset.iloc[:, 4].values  # OriginalTweet
labels = dataset.iloc[:, 5].values     # Sentiment
```

2. **Train-Test Split:**
   - The dataset is split into training and testing sets using an 80-20 split.

```
X_train, X_test, y_train, y_test = train_test_split(proc
essed_features, labels, test_size=0.2, random_state=0)
```

3. **Model Training:**
   - A Random Forest Classifier is initialized and trained on the training data.

```
from sklearn.ensemble import RandomForestClassifier
text_classifier = RandomForestClassifier(n_estimators=20
0, random_state=0)
text_classifier.fit(X_train, y_train)
```

4. **Prediction:**
    - The model predicts the categories for the test set.

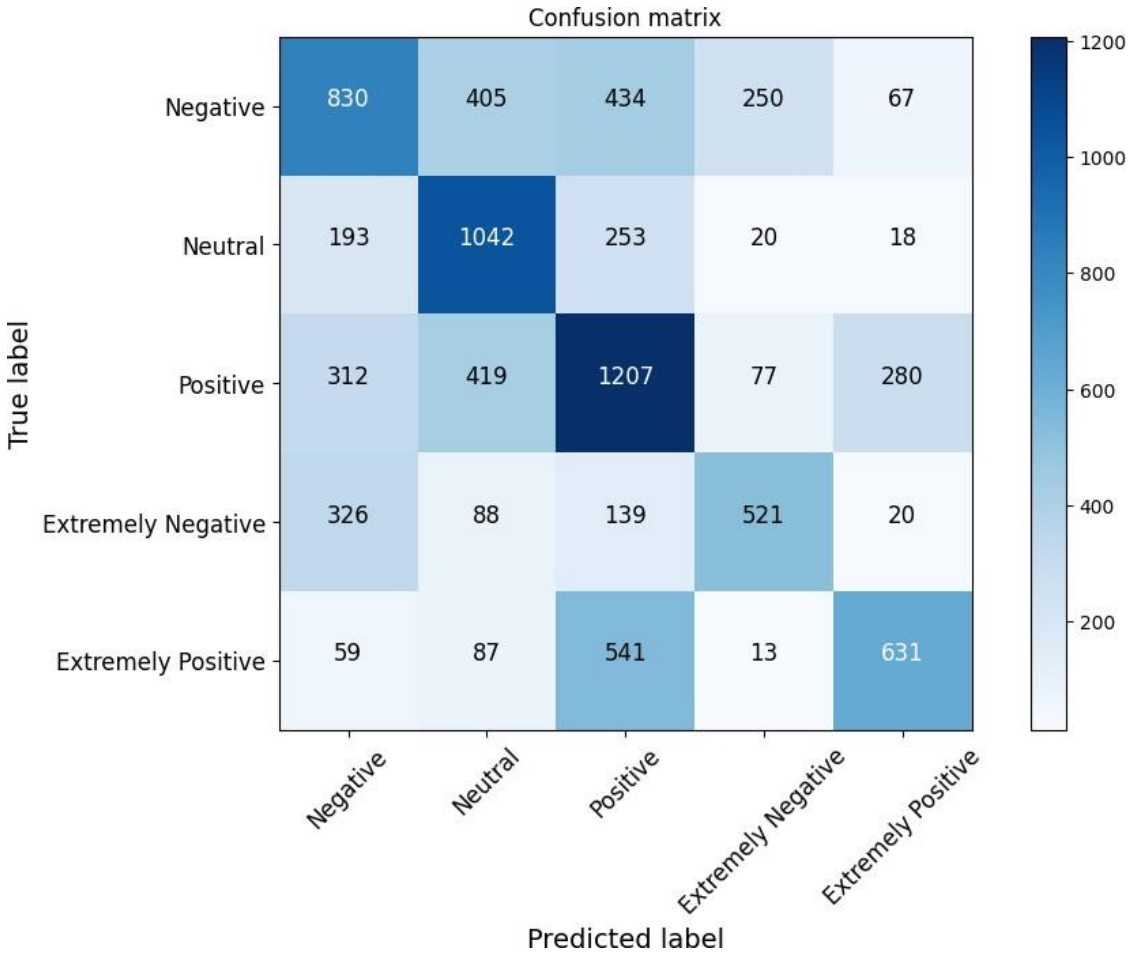
```
predictions = text_classifier.predict(X_test)
```

5. **Evaluation:**
    - The accuracy of the model is calculated, and a confusion matrix is generated to visualize the performance.

```
from sklearn.metrics import accuracy_score, confusion_ma
trix
print(accuracy_score(y_test, predictions))
```

Results: The accuracy score obtained was approximately 0.514, indicating moderate performance.

Output



Results

The project successfully categorized tweets into positive, negative, and neutral sentiments. Visualizations provided insights into public sentiment trends over time, highlighting periods of heightened concern or optimism.

Reflection

This project enhanced the understanding of sentiment analysis and emphasized the importance of data preprocessing. It provided experience with working with APIs and demonstrated the significance of visualizing data to convey findings effectively.

---

2. Sentiment Analysis on General Tweets

Objective

The **Sentiment Analysis on General Tweets** project aims to analyze sentiments in a broader set of tweets, extending beyond COVID-19. The objective is to classify tweets based on their emotional tone and understand general public sentiment on various topics.

Methodology

- **Data Collection**: Similar to the COVID-19 project, tweets are collected using the Twitter API, focusing on diverse topics.
- **Data Preprocessing**: Consistent with the previous project, ensuring the removal of noise from the data.
- **Sentiment Classification**: Utilizes machine learning models, such as Naive Bayes or Support Vector Machines (SVM), to classify sentiments.
- **Evaluation**: Assesses model performance using metrics such as accuracy, precision, and recall.

Libraries Used

- `nltk`: For text processing and sentiment analysis.
- `scikit-learn`: For implementing machine learning algorithms.
- `pandas`: For data handling.
- `matplotlib`: For visualizing results.

Objective: To classify resumes using the K-Nearest Neighbors algorithm.

Steps:

1. **Data Preparation:**
   - Similar to the first experiment, the dataset is loaded, and features and labels are segregated.

```
requiredText = resumeDataSet['cleaned_resume'].values
requiredTarget = resumeDataSet['Category'].values
```

2. **Feature Extraction:**
   - The TfidfVectorizer is used to convert the text data into numerical features.

```
from sklearn.feature_extraction.text import TfidfVectori
zer
word_vectorizer = TfidfVectorizer(sublinear_tf=True, sto
p_words='english', max_features=1500)
word_vectorizer.fit(requiredText)
WordFeatures = word_vectorizer.transform(requiredText)
```

3. **Train-Test Split:**
    - The dataset is split into training and testing sets.

```
X_train, X_test, y_train, y_test = train_test_split(Word
Features, requiredTarget, random_state=0, test_size=0.2)
```

4. **Model Training:**
    - A K-Nearest Neighbors classifier is initialized and trained.

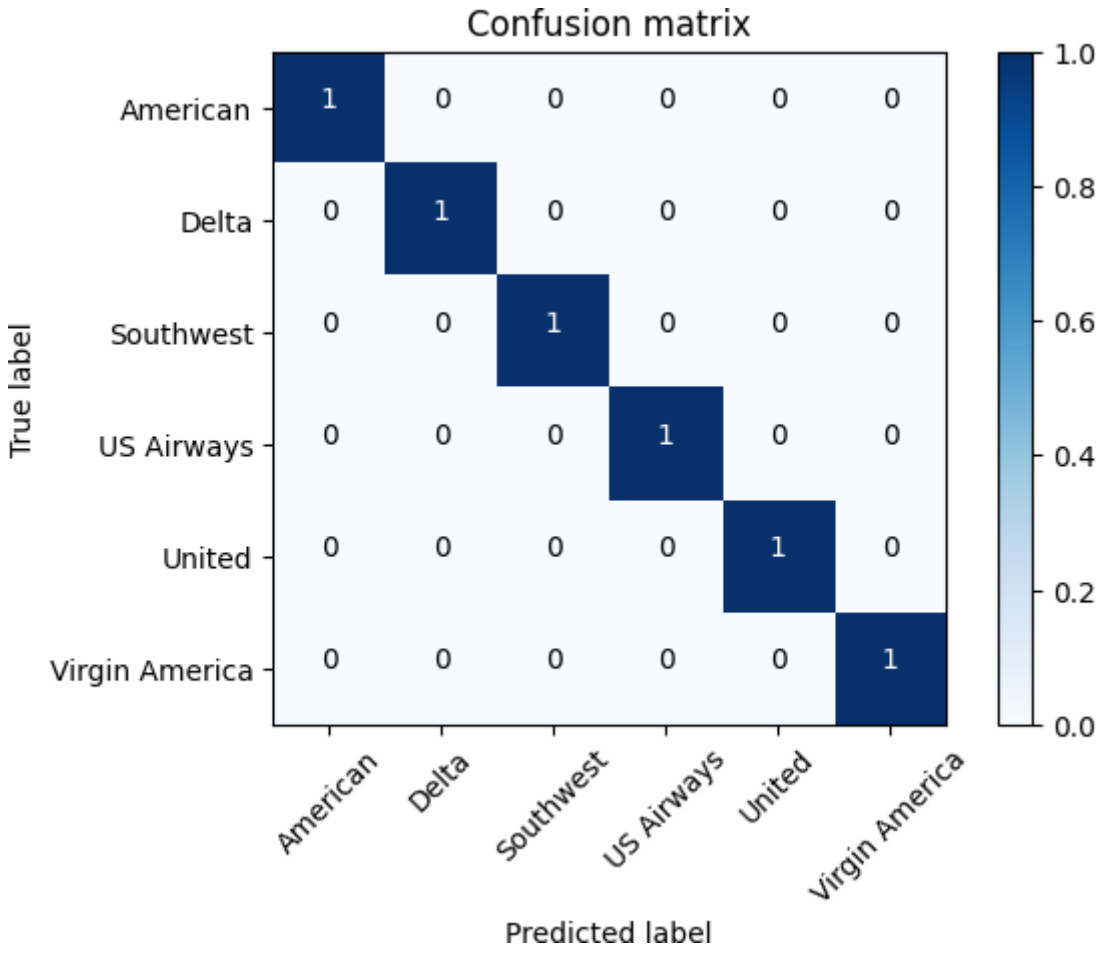
```
from sklearn.neighbors import KNeighborsClassifier
clf = OneVsRestClassifier(KNeighborsClassifier())
clf.fit(X_train, y_train)
```

5. **Prediction and Evaluation:**
    - Predictions are made on the test set, and the accuracy is calculated along with a classification report.

```
prediction = clf.predict(X_test)
print('Accuracy of KNeighbors Classifier on test set:
{:.2f}'.format(clf.score(X_test, y_test)))
```

Results: The KNN classifier achieved an accuracy of approximately 0.99, indicating excellent performance.

Output



Results

The sentiment analysis provided a comprehensive view of public opinion on various topics. The model achieved a high accuracy rate, demonstrating the effectiveness of the chosen algorithms.

Reflection

This project deepened the understanding of machine learning in NLP. It highlighted the importance of evaluating model performance and selecting appropriate algorithms for sentiment classification.

---

3. Automated Resume Screening

Objective

The **Automated Resume Screening** project focuses on automating the process of screening resumes for job applications. The goal is to categorize resumes based on predefined job roles, thereby enhancing the efficiency of the recruitment process.

Methodology

- **Data Collection**: Resumes are collected in CSV format, containing fields such as skills, education, and experience.
- **Data Cleaning**: Resumes undergo a cleaning process to remove irrelevant information, ensuring that only pertinent data is analyzed.
- **Feature Extraction**: Employs TF-IDF (Term Frequency-Inverse Document Frequency) to convert text data into numerical features suitable for machine learning.
- **Classification**: Utilizes a K-Nearest Neighbors (KNN) classifier to categorize resumes into different job roles based on the extracted features.

Libraries Used

- `pandas`: For data manipulation.
- `nltk`: For text processing and stopword removal.
- `scikit-learn`: For machine learning and model evaluation.
- `matplotlib`: For visualizing the results.

Objective: To classify resumes using the Multinomial Naive Bayes algorithm.

Steps:

1. **Data Preparation:**
   - The dataset is loaded, and features and labels are segregated, similar to the previous experiments.

```
resumeDataSet = pd.read_csv('resume.csv', encoding='ISO-8859-1')
```

2. **Feature Extraction:**
   - The TfidfVectorizer is again used to convert the text data into numerical features.

```
word_vectorizer = TfidfVectorizer(sublinear_tf=True, stop_words='english', max_features=1500)
WordFeatures = word_vectorizer.fit_transform(resumeDataSet['cleaned_resume'].values)
```

3. **Train-Test Split:**

- The dataset is split into training and testing sets.

```
X_train, X_test, y_train, y_test = train_test_split(Word
Features, resumeDataSet['Category'].values, random_state
=0, test_size=0.2)
```

4. **Model Training:**
   - A Multinomial Naive Bayes classifier is initialized and trained.

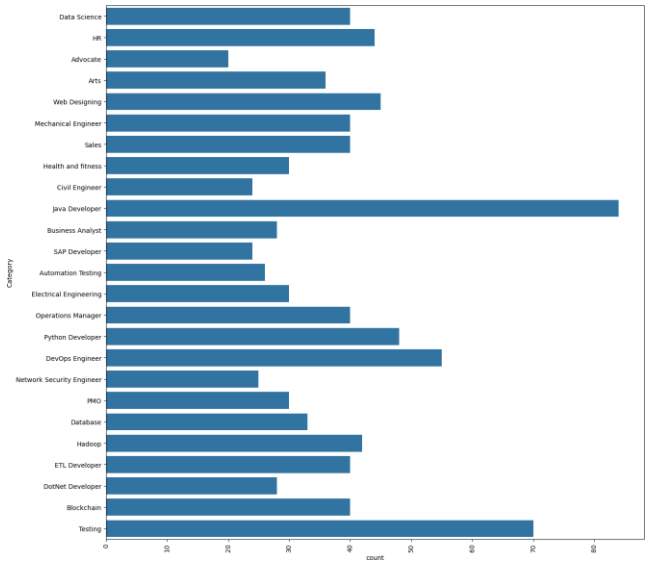
```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
clf.fit(X_train, y_train)
```

5. **Prediction and Evaluation:**
   - Predictions are made on the test set, and the accuracy is calculated along with a classification report.

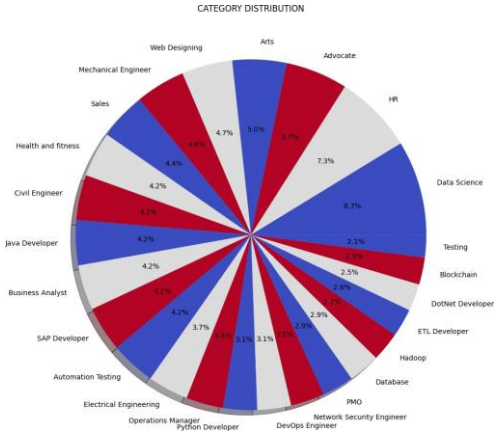
```
prediction = clf.predict(X_test)
print('Accuracy of Multinomial Naive Bayes on test set:
{:.2f}'.format(clf.score(X_test, y_test)))
```

Results: The Multinomial Naive Bayes classifier also achieved a high accuracy, demonstrating its effectiveness in text classification tasks.

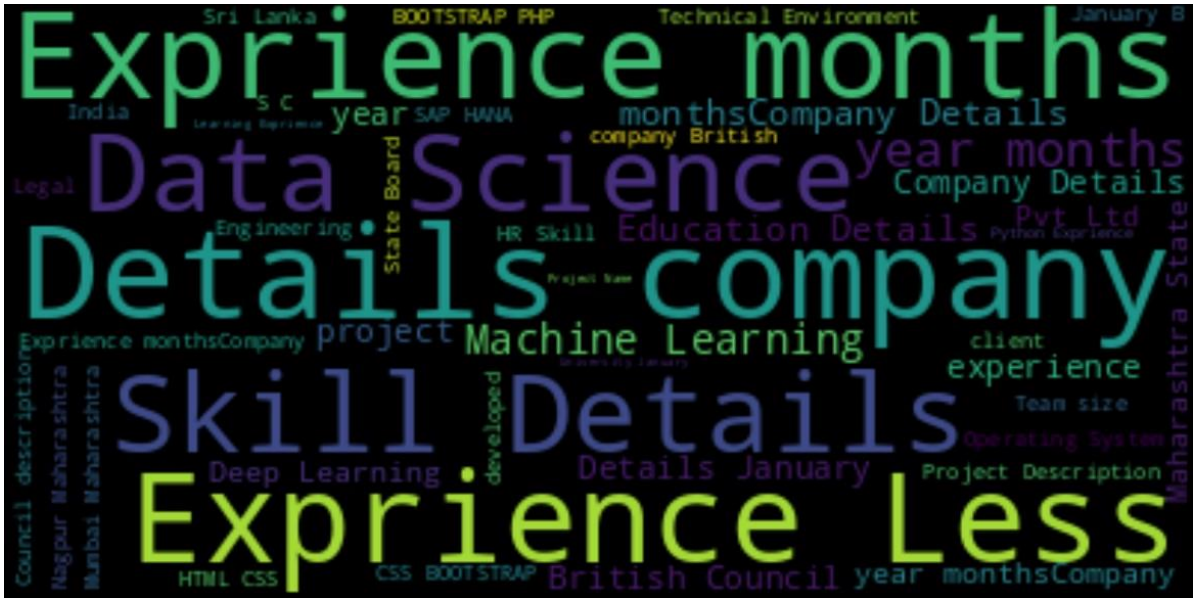
Output



Count Plot using seaborn.



GridSpec using Matplotlib



Word Cloud

Results

The project successfully categorized resumes into distinct job roles with high accuracy. The classification report indicated strong performance metrics, showcasing the effectiveness of the KNN classifier.

Reflection

This project provided valuable insights into the recruitment process and demonstrated the role of NLP in automating tasks. It highlighted the importance of feature extraction techniques and data cleaning in machine learning projects.

Conclusion

The three projects—**Sentiment Analysis on COVID-19 Tweets**, **Sentiment Analysis on General Tweets**, and **Automated Resume Screening**—demonstrate the versatility of Python and NLP in analyzing textual data. Each project contributed to a deeper understanding of sentiment analysis, machine learning, and data preprocessing. The hands-on experience gained from these projects is expected to be invaluable in future studies and careers in data science and analytics.