Download temperature data from below link.
https://www.kaggle.com/venky73/temperaturesofindia?
select=temperatures.csv This data consists of temperatures of I
averaging the temperatures of all places month wise. Temperat
values are
recorded in CELSIUS
a. Apply Linear Regression using suitable library function and p
the Monthwisetemperature.
b. Assessthe performance of regression models using MSE, MA
R-Square
metrics
c. Visualize simple regression model

To accomplish the tasks you've described, let's break them down into steps:

1. **Download and Load the Data**: You'll first need to download the dataset from the prov Kaggle link. Once you have the dataset, we can load it into Python using `pandas`.
2. **Apply Linear Regression**: We'll use `LinearRegression` from `sklearn` to train a model predicts the average temperature for a given month, using month numbers as the feat
3. **Assess Model Performance**: We'll evaluate the model using metrics like Mean Squarec (MSE), Mean Absolute Error (MAE), and R-squared.
4. **Visualize the Model**: We'll use `matplotlib` or `seaborn` to visualize the regression line fit of the model.

Step-by-Step Guide:

1. Download and Load the Data

First, download the dataset from Kaggle. After downloading, you can load the dataset usin The file you need to read is `temperatures.csv`.

python

```
import pandas as pd # Load the dataset df = pd.read_csv('path_to_temperatures.csv
Preview the data df.head()
```

2. Apply Linear Regression

We'll use the `LinearRegression` model from `sklearn`. Before applying the model, we nee process the data. Since you're interested in predicting temperatures based on months, we'l the `Month` and `Temperature` columns.

python

```
from sklearn.model_selection import train_test_split from sklearn.linear_model im
LinearRegression from sklearn.metrics import mean_squared_error, mean_absolute_er
r2_score import matplotlib.pyplot as plt # Prepare the data X = df[['Month']] # F
Month y = df['Temperature'] # Target: Temperature # Split the data into training
testing sets X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=
random_state=42) # Initialize the Linear Regression model model = LinearRegressio
Train the model model.fit(X_train, y_train) # Make predictions on the test set y_
model.predict(X_test)
```

3. Assess Model Performance

Now that we have predictions, we can assess the performance of the model using MSE, MA squared.

python

```
# Calculate MSE, MAE, and R-squared mse = mean_squared_error(y_test, y_pred) mae
mean_absolute_error(y_test, y_pred) r2 = r2_score(y_test, y_pred) # Display the
```

```
performance metrics print(f'Mean Squared Error (MSE): {mse}') print(f'Mean Absolu
Error (MAE): {mae}') print(f'R-squared: {r2}')
```

4. Visualize the Simple Regression Model

To visualize the regression model, we can plot the data points and the regression line.

python

```
# Plot the data points plt.scatter(X, y, color='blue', label='Actual Data') # Plo
regression line plt.plot(X, model.predict(X), color='red', label='Regression Line
Add labels and title plt.xlabel('Month') plt.ylabel('Temperature (°C)')
plt.title('Temperature vs Month (Linear Regression)') plt.legend() # Show the plo
plt.show()
```

Full Code Example

python

```
import pandas as pd from sklearn.model_selection import train_test_split from
sklearn.linear_model import LinearRegression from sklearn.metrics import
mean_squared_error, mean_absolute_error, r2_score import matplotlib.pyplot as plt
the dataset df = pd.read_csv('path_to_temperatures.csv') # Prepare the data X =
df[['Month']] # Feature: Month y = df['Temperature'] # Target: Temperature # Spli
data into training and testing sets X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2, random_state=42) # Initialize the Linear
Regression model model = LinearRegression() # Train the model model.fit(X_train,
y_train) # Make predictions on the test set y_pred = model.predict(X_test) # Calc
MSE, MAE, and R-squared mse = mean_squared_error(y_test, y_pred) mae =
mean_absolute_error(y_test, y_pred) r2 = r2_score(y_test, y_pred) # Display the
performance metrics print(f'Mean Squared Error (MSE): {mse}') print(f'Mean Absolu
Error (MAE): {mae}') print(f'R-squared: {r2}') # Plot the data points plt.scatter
color='blue', label='Actual Data') # Plot the regression line plt.plot(X,
model.predict(X), color
```

more infomation about above sta

Certainly! Here's a more detailed explanation of each step and its purpose in the process o linear regression to predict the average monthly temperatures in India:

Step-by-Step Breakdown of the Approach

1. Download and Load the Data

You first need to **download the dataset** from Kaggle and load it into your Python environm dataset contains temperature data for various locations in India, averaged by month. It is cr understand the structure of this data before proceeding with the analysis.

Dataset Exploration:

- The dataset likely contains columns such as:
  - `Month` : The month of the year (1 for January, 2 for February, etc.)
  - `Temperature` : The average temperature in Celsius for that month.

We can load the data using the `pandas` library, which provides powerful tools for reading a processing CSV files.

python

```
import pandas as pd # Load the dataset (make sure to replace 'path_to_temperature
with the actual file path) df = pd.read_csv('path_to_temperatures.csv') # Preview
first few rows of the dataset to understand its structure df.head()
```

The `head()` function helps us view the first five rows of the dataset, ensuring that we have loaded the data and understand its structure.

2. Apply Linear Regression

Once you have the dataset loaded, you can use **linear regression** to model the relationship the month (as the feature or independent variable) and the temperature (as the target or d variable).

Linear regression assumes a **linear relationship** between the input variable (month) and the variable (temperature), which means it attempts to fit a straight line to the data.

Steps:

- **Feature Selection**: In this case, we are only using `Month` as the feature (independent v and `Temperature` as the target (dependent variable).
- **Model Training**: We'll split the dataset into **training** and **testing** sets. The training set i train the model, while the test set will be used to evaluate its performance.

python

```
from sklearn.model_selection import train_test_split from sklearn.linear_model im
LinearRegression # Prepare the data (X is the independent variable, y is the targ
df[['Month']] # Feature: Month (we only have one feature here) y = df['Temperatur
Target: Temperature # Split the data into training and testing sets (80% training
testing) X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) # Initialize the Linear Regression model model = LinearRegressio
Train the model on the training data model.fit(X_train, y_train) # Make predictio
the test data y_pred = model.predict(X_test)
```

3. Assess Model Performance

To assess how well the model has performed, we use three **evaluation metrics**:

1. **Mean Squared Error (MSE)**: This metric gives you the average squared difference betw actual and predicted values. A lower value indicates a better fit.
   - Formula:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

   Where:

   - y_i is the actual value.
   - $\hat{y}_i$ is the predicted value.
2. **Mean Absolute Error (MAE)**: This metric gives you the average of the absolute differer between the actual and predicted values. It is less sensitive to outliers than MSE.
   - Formula:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

3. **R-squared**: This is a measure of how well the regression line explains the variance in th An R-squared value of 1 means the model perfectly predicts the temperature, while a v means the model does not explain the variance at all.

python

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score #
Calculate MSE, MAE, and R-squared mse = mean_squared_error(y_test, y_pred) mae =
mean_absolute_error(y_test, y_pred) r2 = r2_score(y_test, y_pred) # Print the eva
metrics print(f'Mean Squared Error (MSE): {mse}') print(f'Mean Absolute Error (MA
{mae}') print(f'R-squared: {r2}')
```

- **MSE**: A lower MSE indicates the model is more accurate in predicting the temperature.
- **MAE**: A lower MAE means the model is more consistent in its predictions.
- **R-squared**: A higher R-squared value indicates the model does a good job of fitting th

4. Visualize the Simple Regression Model

Once you've trained the model and made predictions, it's useful to visualize how well the re line fits the data. We can plot the **actual data points** (month vs. temperature) and overlay the **regression line** (predicted temperatures).

Steps:

- Plot the original data points using a scatter plot.
- Plot the regression line to see how well it fits the data.

python

```
import matplotlib.pyplot as plt # Plot the actual data points (Month vs Temperatu
plt.scatter(X, y, color='blue', label='Actual Data') # Plot the regression line (
prediction) plt.plot(X, model.predict(X), color='red', label='Regression Line') #
labels and a title to the plot plt.xlabel('Month') plt.ylabel('Temperature (°C)')
plt.title('Temperature vs Month (Linear Regression)') # Show a legend plt.legend(
Display the plot plt.show()
```

Key Takeaways from Visualization:

- **Scatter plot**: Shows the actual data points — each point represents a month and its corresponding average temperature.
- **Regression line**: This line represents the model's prediction for each month based on t training data. Ideally, this line should closely follow the general trend of the actual data

Interpreting Results

- **R-squared value**: If the R-squared value is close to 1, it means the linear regression mo the data well. If it's much lower, it indicates that a linear model may not be the best ch you may need to consider more complex models (e.g., polynomial regression).
- **MSE and MAE**: Both of these metrics should ideally be low, as lower values indicate be predictions. However, for datasets with extreme outliers, these metrics can be sensitive additional tuning or more sophisticated methods may be required.

Additional Considerations

- **Seasonality**: Temperature might have seasonal patterns that aren't purely linear. For ex months like June or December may exhibit temperature trends that are not perfectly represented by a simple linear model. In such cases, polynomial regression or time ser forecasting models might be more appropriate.
- **Data Preprocessing**: Depending on the dataset, you may need to clean or preprocess (e.g., handle missing values, remove outliers, or normalize values).

Summary of the Process:

1. **Data Loading**: Load the dataset using `pandas`.
2. **Modeling**: Use **Linear Regression** to predict temperature based on month number.
3. **Evaluation**: Assess the model's performance using **MSE**, **MAE**, and **R-squared**.
4. **Visualization**: Plot the actual data and the regression line to visually inspect the model

By following these steps, you'll be able to apply a linear regression model to predict month temperatures in India and evaluate its performance. Let me know if you'd like more detail c

part of the process!

Every year many students give the GRE exam to get admission foreign
Universities. The data set contains GRE Scores (out of 340), TOE Scores (out of
120), University Rating (out of 5), Statement of Purpose strengt of 5), Letter
of Recommendation strength (out of 5), Undergraduate GPA (o 10), Research
Experience (0=no, 1=yes), Admitted (0=no, 1=yes). Admitted is target variable.
Data Set Available on kaggle (The last column of the dataset ne be changed
to 0 or 1)Data Set :
https://www.kaggle.com/mohansacharya/graduate-admissions
The counselor of the firm is supposed check whether the stude get an
admission or not based on his/her GRE score and Academic Sc to help the
counselor to take appropriate decisions build a machine learnir model classifier
using Decision tree to predict whether a student will get admis not. Apply
Data pre-processing (Label Encoding, Data Transformation….) techniques if
necessary. Perform data-preparation (Train-Test Split)
C. Apply Machine Learning Algorithm
D. Evaluate Model.

To solve the problem of predicting whether a student will get admission to a university bas their GRE score, academic score, and other factors, we can follow a systematic approach th data preprocessing, model training, evaluation, and prediction. Below are the detailed step implement this machine learning pipeline using a **Decision Tree Classifier**:

Steps to Follow:

1. **Data Preprocessing**:
   - Load and clean the dataset.
   - Handle missing values if any.
   - Encode categorical variables using label encoding or one-hot encoding.
   - Normalize or scale the features if necessary.
2. **Data Preparation**:
   - Split the dataset into training and test sets.
3. **Apply Machine Learning Algorithm**:
   - Train a **Decision Tree Classifier** model on the training set.
4. **Evaluate the Model**:
   - Assess model performance using accuracy, confusion matrix, precision, recall, and

Step-by-Step Implementation

1. **Loading and Preprocessing the Data**

First, we need to load the dataset from Kaggle and perform necessary preprocessing steps.

python

```
import pandas as pd import numpy as np from sklearn.model_selection import
train_test_split from sklearn.tree import DecisionTreeClassifier from sklearn.met
import accuracy_score, confusion_matrix, classification_report from
sklearn.preprocessing import LabelEncoder import matplotlib.pyplot as plt import
as sns # Load the dataset (replace 'path_to_file.csv' with the actual file path)
"https://raw.githubusercontent.com/datasets/graduate-
admissions/master/admissions_data.csv" df = pd.read_csv(url) # Display the first
rows of the dataset df.head()
```

2. **Data Preprocessing**

- **Label Encoding**: Since some of the columns like `Research` `Experience` are categorical we need to make sure these columns are encoded properly.
- **Handling Missing Values**: If any columns contain missing values, we need to handle th
- **Feature Engineering**: We will select the relevant features for training (e.g., GRE Score, T Score, University Rating, GPA, Research Experience) and ensure that the target variable (`Admitted`) is correctly formatted.

python

```
# Check for missing values print(df.isnull().sum()) # If any missing values are
present, we can fill them with the mean or mode (depending on the column type)
df.fillna(df.mean(), inplace=True) # Label encode the target variable 'Admitted'
should already be 0 or 1 but confirm it) df['Admitted'] = df['Admitted'].apply(la
1 if x == 'Yes' else 0) # For categorical columns, like 'Research', if necessary
it seems binary) # df['Research'] = df['Research'].map({'yes': 1, 'no': 0}) # Fea
selection: Define X (features) and y (target) X = df[['GRE Score', 'TOEFL Score',
'University Rating', 'SOP', 'LOR', 'CGPA', 'Research']] y = df['Admitted'] # Spli
dataset into training and testing sets (80% training, 20% testing) X_train, X_tes
y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

3. **Training the Decision Tree Classifier**

We now train the **Decision Tree Classifier** using the training data. The decision tree model automatically handle both numerical and categorical features and works well for this type problem.

python

```
# Initialize the Decision Tree Classifier model =
DecisionTreeClassifier(random_state=42) # Train the model model.fit(X_train, y_tr
Make predictions on the test data y_pred = model.predict(X_test)
```

4. **Model Evaluation**

After training the model, we need to evaluate its performance using various metrics like acc confusion matrix, precision, recall, and F1-score.

python

```
# Evaluate the model using accuracy accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}') # Confusion Matrix conf_matrix =
confusion_matrix(y_test, y_pred) print(f'Confusion Matrix:\n{conf_matrix}') #
Classification Report (Precision, Recall, F1-score) class_report =
classification_report(y_test, y_pred) print(f'Classification Report:\n{class_repo
```

5. **Visualizing the Decision Tree**

It's helpful to visualize the trained decision tree to understand how it is making decisions. V plot the decision tree using `plot_tree` from `sklearn` .

python

```
from sklearn.tree import plot_tree # Plot the decision tree plt.figure(figsize=(1
10)) plot_tree(model, filled=True, feature_names=X.columns, class_names=['Not Adm
```

```
'Admitted'], rounded=True, fontsize=12) plt.show()
```

This will display a visual representation of the decision tree, showing how the model splits t based on different features.

Complete Code Example:

python

```
import pandas as pd import numpy as np from sklearn.model_selection import
train_test_split from sklearn.tree import DecisionTreeClassifier, plot_tree from
sklearn.metrics import accuracy_score, confusion_matrix, classification_report im
matplotlib.pyplot as plt import seaborn as sns # Load the dataset url =
"https://raw.githubusercontent.com/datasets/graduate-
admissions/master/admissions_data.csv" df = pd.read_csv(url) # Check for missing
print(df.isnull().sum()) # Fill missing values with the mean df.fillna(df.mean(),
inplace=True) # Label encode the target variable 'Admitted' (0 or 1) df['Admitted
df['Admitted'].apply(lambda x: 1 if x == 'Yes' else 0) # Select features and targ
variable X = df[['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR', '
'Research']] y = df['Admitted'] # Split the dataset into training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) # Initialize and train the Decision Tree model model =
DecisionTreeClassifier(random_state=42) model.fit(X_train, y_train) # Make predic
on the test data y_pred = model.predict(X_test) # Evaluate the model accuracy =
accuracy_score(y_test, y_pred) print(f'Accuracy: {accuracy:.4f}') # Confusion Mat
conf_matrix = confusion_matrix(y_test, y_pred) print(f'Confusion
Matrix:\n{conf_matrix}') # Classification Report (Precision, Recall, F1-score)
class_report = classification_report(y_test, y_pred) print(f'Classification
Report:\n{class_report}') # Visualize the Decision Tree plt.figure(figsize=(15, 1
plot_tree(model, filled=True, feature_names=X.columns, class_names=['Not Admitted
'Admitted'], rounded=True, fontsize=12) plt.show()
```

Explanation of Key Parts:

1. **Data Preprocessing**:
   - We handle missing values by filling them with the mean (`df.fillna(df.mean())`), this can be adjusted based on the column type.
   - We ensure that the `Admitted` column is in a binary format (0 or 1) for the classifie understand.
2. **Model Training**:
   - We split the data into training and test sets, training the **Decision Tree Classifier** c training set and making predictions on the test set.
3. **Model Evaluation**:
   - We evaluate the model using **accuracy**, **confusion matrix**, and a **classification rep** includes precision, recall, and F1-score, which are helpful metrics for imbalanced c
4. **Visualization**:
   - We plot the trained decision tree using `plot_tree` to visualize how decisions are based on the features.

Model Performance Interpretation:

- **Accuracy**: It gives you the proportion of correct predictions. However, for imbalanced (if the number of admitted vs. not admitted students is unequal), accuracy might not b

- **Confusion Matrix**: Helps you see how many true positives (admitted correctly), true ne (not admitted correctly), false positives (admitted incorrectly), and false negatives (not incorrectly) are present.
- **Precision, Recall, F1-score**: Precision tells you the proportion of true positives among a predictions. Recall tells you how many actual positives the model correctly identified. F combines both precision and recall into one metric.

By following these steps, you should be able to successfully train and evaluate a **Decision T Classifier** to predict whether a student will get admission to a university based on their feat

Design and implement IOT system using Arduino –uno using ultrasonic sensor
and servo motor. more information about it

Design and Implement an IoT System Using Arduino UNO, Ultrasoni Sensor, and Servo Motor

To design and implement an IoT system with Arduino UNO, an ultrasonic sensor, and a serv we will go through the following steps:

- **System Overview**:
  - **Ultrasonic Sensor**: Measures distance by sending out sound waves and measuring it takes for the sound to reflect back. This will be used to detect how far an object the sensor.
  - **Servo Motor**: A small motor that can rotate to a specific angle based on the input We will use the servo motor to rotate based on the distance detected by the ultra sensor.
  - **Arduino UNO**: The central controller that will read the data from the ultrasonic se control the servo motor.
  - **IoT Integration**: We'll connect the Arduino to the internet to send sensor data rem (using an IoT platform or a web server).

Components Required:

1. **Arduino UNO** - The microcontroller that will control the system.
2. **Ultrasonic Sensor (HC-SR04)** - Measures distance.
3. **Servo Motor** (e.g., SG90) - Rotates based on distance data.
4. **Jumper Wires** - For connecting components.
5. **Breadboard** - For connecting components.
6. **Wi-Fi Module (e.g., ESP8266 or ESP32)** - To enable the IoT functionality, allowing the to send data to the cloud.
7. **Power Supply** - A 5V USB or battery to power the Arduino.
8. **IoT Platform** (optional but recommended) - For real-time data monitoring (e.g., Thing Blynk, or Firebase).

Wiring the Components:

1. **Ultrasonic Sensor (HC-SR04)**:

- **VCC** pin → 5V on Arduino
- **GND** pin → GND on Arduino
- **Trig** pin → Pin 9 on Arduino
- **Echo** pin → Pin 10 on Arduino

2. **Servo Motor (SG90)**:
   - **VCC** pin → 5V on Arduino
   - **GND** pin → GND on Arduino
   - **Control** pin → Pin 11 on Arduino
3. **Wi-Fi Module (e.g., ESP8266)** (Optional for IoT integration):
   - **VCC** pin → 3.3V on Arduino
   - **GND** pin → GND on Arduino
   - **TX** pin → RX on Arduino
   - **RX** pin → TX on Arduino

Code Implementation:

1. Basic Arduino Code for Ultrasonic Sensor and Servo Motor Control

Here, the ultrasonic sensor will detect the distance of an object and control the servo moto accordingly.

cpp

```
#include <Servo.h> #define TRIG_PIN 9 // Trigger pin of the Ultrasonic sensor #de
ECHO_PIN 10 // Echo pin of the Ultrasonic sensor #define SERVO_PIN 11 // Pin for
motor Servo myServo; // Create Servo object void setup() { Serial.begin(9600); //
the serial communication pinMode(TRIG_PIN, OUTPUT); // Set the trigger pin as out
pinMode(ECHO_PIN, INPUT); // Set the echo pin as input myServo.attach(SERVO_PIN);
Attach the servo motor to pin 11 } void loop() { long duration, distance; // Clea
Trigger pin digitalWrite(TRIG_PIN, LOW); delayMicroseconds(2); // Set the Trigger
HIGH to send out a pulse digitalWrite(TRIG_PIN, HIGH); delayMicroseconds(10);
digitalWrite(TRIG_PIN, LOW); // Measure the time it takes for the echo to return
duration = pulseIn(ECHO_PIN, HIGH); // Calculate the distance in centimeters dist
(duration / 2) * 0.0344; // Print distance to Serial Monitor for debugging
Serial.print("Distance: "); Serial.print(distance); Serial.println(" cm"); // Map
distance to a servo motor angle (0 to 180 degrees) int angle = map(distance, 0, 2
180); // Map distance to angle // Control the servo motor position based on dista
myServo.write(angle); delay(500); // Delay to prevent rapid servo movements }
```

Explanation of the Code:

1. **Libraries**:
   - We include the `Servo.h` library to control the servo motor easily.
2. **Pin Setup**:
   - The `TRIG_PIN` and `ECHO_PIN` are assigned to pins 9 and 10, respectively, to interf the ultrasonic sensor.
   - `SERVO_PIN` is assigned to pin 11 for controlling the servo motor.
3. **Distance Calculation**:
   - We send a 10-microsecond pulse on the `TRIG_PIN` to trigger the ultrasonic senso
   - The sensor then sends back an echo, and we calculate the time it takes for the ech return using the `pulseIn()` function.
   - The distance is calculated using the speed of sound formula:

$$\text{Distance} = (\frac{\text{Duration}}{2}) \times 0.0344\,(\text{cm})$$

- The distance is then mapped to an angle between 0 and 180 degrees to control th motor.

4. **Servo Control**:
   - The `myServo.write(angle)` function moves the servo to the calculated angle.
5. **Serial Output**:
   - The distance is printed to the Serial Monitor for debugging purposes.

2. IoT Integration (Using ESP8266 with ThingSpeak)

To add IoT capabilities to this system, we can use an **ESP8266 Wi-Fi module** to send distan the cloud, for example, to **ThingSpeak**.

- **ThingSpeak** is an open-source IoT platform that allows you to store and visualize data devices in real time.

1. **ThingSpeak Account Setup**:
   - Create a ThingSpeak account (https://thingspeak.com/).
   - Create a new **channel** and get the **Write API Key** (you will use this key to send dat
2. **Arduino Code for IoT Integration** (using ESP8266):

Here is an updated version of the code that sends the distance data to ThingSpeak.

cpp

```
#include <Servo.h> #include <ESP8266WiFi.h> #include <ThingSpeak.h> #define TRIG_
#define ECHO_PIN 10 #define SERVO_PIN 11 // Wi-Fi credentials const char *ssid =
"your_SSID"; const char *password = "your_PASSWORD"; // ThingSpeak settings unsig
long channelID = your_channel_ID; const char *writeAPIKey = "your_Write_API_Key";
WiFiClient client; Servo myServo; void setup() { Serial.begin(115200); // Connect
Fi WiFi.begin(ssid, password); while (WiFi.status() != WL_CONNECTED) { delay(1000
Serial.println("Connecting to WiFi..."); } Serial.println("Connected to WiFi");
ThingSpeak.begin(client); pinMode(TRIG_PIN, OUTPUT); pinMode(ECHO_PIN, INPUT);
myServo.attach(SERVO_PIN); } void loop() { long duration, distance;
digitalWrite(TRIG_PIN, LOW); delayMicroseconds(2); digitalWrite(TRIG_PIN, HIGH);
delayMicroseconds(10); digitalWrite(TRIG_PIN, LOW); duration = pulseIn(ECHO_PIN,
distance = (duration / 2) * 0.0344; // Print to Serial Monitor Serial.print("Dist
"); Serial.print(distance); Serial.println(" cm"); // Send data to ThingSpeak
ThingSpeak.setField(1, distance); int responseCode = ThingSpeak.writeFields(chanr
writeAPIKey); if (responseCode == 200) { Serial.println("Data sent to ThingSpeak
successfully."); } else { Serial.print("Error sending data. HTTP error code: ");
Serial.println(responseCode); } // Control the servo based on distance int angle
map(distance, 0, 200, 0, 180); myServo.write(angle); delay(10000); // Send data e
seconds }
```

Explanation of IoT Code:

1. **Wi-Fi and ThingSpeak Setup**:
   - We connect the ESP8266 to the Wi-Fi network using `WiFi.begin(ssid, password)`
   - We use the **ThingSpeak library** to send data to the cloud. We also need to specify **API Key** to push data to ThingSpeak.
2. **Sending Data to ThingSpeak**:
   - Every time the distance is calculated, we send the data to ThingSpeak using `ThingSpeak.setField(1, distance)`.
   - The `ThingSpeak.writeFields()` function sends the data to the ThingSpeak chann
3. **Servo Motor Control**:

- As in the previous version, the servo motor is controlled based on the distance da

4. **Sending Data Periodically**:
    - We delay for 10 seconds (`delay(10000)`) between