**1. Backend Development (Python)**

**1.1 Data Downloading & Preprocessing:**

**1.2 Marginal Emission Calculations**

**1.3 APIs for Backend:**

* Create RESTful APIs using **Flask** or **FastAPI** that will expose:
  + **/data**: Endpoint to retrieve/download data
  + **/analyze**: Perform marginal emissions analysis based on parameters (location, energy type, etc.)
  + **/visualize**: Return graphical representation of data (emissions trend, marginal cost, etc.)

**2. Frontend Development**

**2.1 User Interface (GUI):**

* **Frontend Framework**: Use **React.js** to create a responsive web-based GUI.
* **UI Features**:
  + **File Upload** (if users want to upload specific datasets)
  + **Location Input** (for regional-based analysis)
  + **Energy Source Selection** (e.g., coal, natural gas, renewable energy)
  + **Interactive Data Visualizations** (via integration with plotly.js or chart.js)
  + **Run Analysis Button**: Trigger the backend API to run marginal emissions analysis.

**2.2 API Integration:**

* The frontend should communicate with the backend Flask/FastAPI using Axios or Fetch API.
* **Endpoints**:
  + **GET /data**: To fetch preloaded datasets or download new ones.
  + **POST /analyze**: To request emissions analysis based on user-selected parameters.
  + **GET /visualize**: To retrieve data visualizations from the backend.

**2.3 Hosting:**

* **Frontend Hosting**: Host the static files (HTML, JS, CSS) on **AWS S3** and use **AWS CloudFront** for content delivery.

**3. Cloud Infrastructure (AWS)**

**3.1 Storage (S3)**

* **AWS S3**:
  + Store the data files (raw, processed, results) in separate S3 buckets.
  + Use S3 as storage for static files (HTML, CSS, JS) for the frontend.

**3.2 Compute Resources (Lambda or EC2)**

**Option 1: AWS Lambda + API Gateway (Serverless Approach)**

* **Lambda**: Use AWS Lambda to run the Python code for data downloading and emissions analysis.
  + This is suitable if the workloads are lightweight and run occasionally.
  + **API Gateway**: Create an API Gateway that triggers Lambda functions. Lambda will interact with S3 to fetch/store data and perform calculations.

**Option 2: EC2 Instances (For heavier workloads)**

* **EC2**: Deploy the Python backend on an EC2 instance for more intensive computation.
  + EC2 provides more control for continuous processes and complex tasks.
  + **Auto-scaling**: Based on the number of requests, configure EC2 auto-scaling for high-traffic scenarios.

**Which to choose**:

* **Lambda**: Good for infrequent usage or small datasets.
* **EC2**: Better for more robust applications or if high-performance is needed (e.g., long-running tasks).

**3.3 Serverless Execution Example (with Lambda)**

* **Execution Process**:
  1. User inputs parameters via the GUI.
  2. API call triggers Lambda function.
  3. Lambda fetches the data from S3, runs the analysis, and stores the results back in S3.
  4. API call fetches the results and renders them on the frontend.

**4. Security & Performance**

**4.1 User Authentication:**

* Implement **Cognito** for user authentication if user-specific data access is required.
* Ensure secure API endpoints using HTTPS (handled via API Gateway or CloudFront).

**4.2 Performance Optimizations:**

* **Caching**: Implement caching at the API Gateway level for repeated data fetch requests.
* **Auto-scaling**: Set up auto-scaling for Lambda (concurrent executions) or EC2 (number of instances).

**4.3 Monitoring & Logging:**

* **CloudWatch**: Use AWS CloudWatch for monitoring Lambda functions or EC2 instances.
* **S3 Logging**: Track access logs for files in S3 buckets.

**5. Deployment**

**5.1 CI/CD Pipeline:**

* Use **GitHub Actions** or **AWS CodePipeline** for CI/CD.
  + Automatically deploy the frontend on S3 and update Lambda functions or EC2 instances when new code is pushed to the repository.

**5.2 API & Domain Setup:**

* Set up a custom domain using **Route 53** and associate it with the API Gateway or EC2 instance.

**7. Timeline**

| **Task** | **Duration** |
| --- | --- |
| Backend API (data handling, analysis) | 2 weeks |
| Frontend (UI/UX development) | 2 weeks |
| AWS Infrastructure Setup (Lambda/S3) | 1 week |
| Integration (Frontend & Backend APIs) | 1 week |
| Testing & Debugging | 1 week |
| Deployment | 3 days |

Total time: **6-7 weeks**