

**TAYLOR’S PROGRAMMES MAY 2025 SEMESTER**

**ITS66604 - Machine Learning and Parallel Computing**

**Instruction to Candidates:**

1. Name your answer file as ITS69304\_XXXXXX\_INDVASGNMT.pdf where XXXXXX is your STUDENT NUMBER. Then, submit to the MyTIMeS portal via the link “**INDIVIDUAL ASSIGNMENT submission**” on the module page. (Do not submit the question paper).

| **Assignment No./Title** | **Assignment (Individual Assignment)**  **20% Weightage** |
| --- | --- |
| **Course Tutor/Lecturer** | Mr. Anmol Adhikari, Mr. Bidhan Chandra Bhattarai |
| **Submission Date** | 21st July 2025 |
| **Student Name, ID and Signature**  1. Bisakha Shrestha  0370008  Bisakha | |

***Declaration*** *(need to be signed by students. Otherwise, the assessment will not be evaluated)*

*Certify that this assignment is entirely my own work, except where I have given fully documented references to the work of others, and that the material contained in this assignment has not previously been submitted for assessment in any other formal course of study.*

| ***Marks/Grade:*** | ***Evaluated by:*** |
| --- | --- |
| ***Evaluator's Comments:***                    *\* Please include this cover page for your project submission* | |

Table of Contents-

[Introduction](#_mhs0onq2prit)

[The Importance of Real-Time Prediction for Disaster Control](#_pgwapwuvtzei)

[Dataset Justification](#_c9j5ueywqb7c)

[System Design for Scalable ML Solution](#_uvc8uj4x941n)

[Linear Regression Model](#_xe7onlas1een)

[Random Forest Model](#_sem7fmo4xkf4)

[Spark Preprocessing](#_3j12r46fz1km)

[Python Preprocessing](#_gvoy10pazudi)

[Proposed Solution Overview](#_ev0e12mg2549)

[Parallel Computing Integration](#_q8ba6bypktr8)

[Tools and Frameworks Used](#_1bd5uoyz1qo1)

[Implementation Strategy and Experimentation](#_a57rpq487urq)

[Model Training Strategy](#_j7qz8xonhv51)

[Parallelism Techniques Used](#_bw753fgirdk0)

[Evaluation Metrics](#_x9jeg1ek1a2b)

[Observation:  
 Parallelization significantly reduced training and prediction time, without sacrificing accuracy. This proves how parallel computing makes real-time ML applications more efficient and practical for real-world use cases like disaster prediction.](#_7ogi8y16m75t)

[Professional and Ethical Responsibility](#_kxvvvam8g5oo)

[System Architecture and Deployment Planning](#_vdc9f1wwz52j)

[Components:](#_67h8unyuy09r)

[Scalability & Reliability:](#_no3le7vfcyuq)

[Professional Communication and Team Collaboration](#_kl6orag5si59)

[How I Managed This Project](#_g4afq2l2ary3)

# **Introduction**

Real-Life Disaster Situation: Landslides

Landslides are perhaps the most incapacitating natural disasters, particularly hilly areas around the world. Landslides in Nepal, India, Indonesia, and the Philippines are a recurring menace, particularly monsoons when rain torrents pour onto the earth. Landslides can lead to serious human and economic loss — causing death, reducing houses and public buildings to rubble, choking roads, and uprooting communities.

Over the past few years, not only have landslides become more frequent and intensive but more especially due to climate change. The altered precipitation regime, augmented global warming, and urbanization over fragile foundations have all combined to make a collective effort towards raising the uncertainty of such disasters. As the hazard increases, the need for an accurate prediction and real-time monitoring system is becoming increasingly important day by day.

We choosed the dataset from kaggle and its path is -

import kagglehub

# Download latest version

path = kagglehub.dataset\_download("rajumavinmar/landslide-dataset")

print("Path to dataset files:", path)

# 

# **The Importance of Real-Time Prediction for Disaster Control**

Having a reliable landslide prediction system is not only a scientific issue but also a humanitarian one. The development of advance warning systems that can generate expectation ahead of communities allows early evacuation, risk reduction, and efficient allocation of emergency resources. The systems significantly reduce loss of life and economic loss by facilitating proactive management of disasters rather than reactive rehabilitation.

Accurate and current predictive models are also useful to governments, non-governmental organizations, and disaster management offices in planning disaster-resistant communities. Predictive models determine risk-prone locations and continuously scan the environment in real-time in order to support long-term planning and the installation of early warning systems in vulnerable locations.

# **Dataset Justification**

We used for this work a Wireless Sensor Network (WSN) fueled Landslide Dataset — a great one to use for predictive modeling and for real-time examination. The dataset contains sensors deployed in the field that are continuously monitoring many geological and environmental parameters of extreme importance. Sensors are positioned in the landslide risk regions in order to acquire fine-grained, high-frequency information so that short-term warning is possible as well as long-term trend observation.

Here are the most crucial parameters discussed in the dataset:

Soil Moisture: Excessive content of water within soils is a measure of water saturation, which is a common slope failure warning sign.

Pore Water Pressure: Change in subsurface pressure of water has the potential to compromise structural stability of the slope.

Rainfall Intensity: Intensive and extended rainfall causes ground saturation and is one of the leading landslide triggers.

Ground Vibrations: Minor ground shaking or vibration could be an indication of slope movement and a premonitory symptom of sliding.

Slope Angle: The larger the angle of inclination, the higher the risk of failure; the gradient of the angle of inclination provides hazard levels.

This is an ideal dataset to utilize for machine learning and real-time monitoring as it's multidimensional, has time-series nature, and is from real sensors. It's a high-feature source to enable models to learn to make predictions correctly under differing circumstances.

Also, the use of a WSN offers scalability and flexibility to be installed in harsh and remote areas where it is not viable to have manual data collection. It enables the model to learn new states without being expensive in the long run.

This dataset is suitable because:

* It includes **multivariate time-series data**, ideal for real-time prediction.
* It reflects actual sensor readings from landslide-prone regions.
* The features are relevant to the physical factors that trigger landslides.
* The data volume is large, making it suitable for **parallel processing**.

Challenges for ML and Parallel Processing

* The data contains **missing values** due to sensor errors or transmission loss.
* Real-time processing requires fast computations and **low-latency predictions**.
* High-dimensional sensor data can be **noisy**, which increases preprocessing demands.
* Large data volume makes **parallelism necessary** to speed up both training and inference.

**Dataset Relevance to Machine Learning and Parallel Processing**

The dataset also well qualifies for use in machine learning and parallel processing processes according to a range of essential attributes that equal the requirements placed on landslide real-time forecasting:

Multivariate Time-Series Nature: The dataset consists of multivariate time-series data, ideal for modeling sequential dependency as well as time-pattern. Time-series modeling is important in early landslide prediction because environmental signals over time have the nature to predict an imminent event.

Real-World Sensor Data: Data is actually provided by wireless sensors installed within the landslide-risk zones themselves with good real-life relevance and authenticity. Real-world data ensures models learned from it can generalize well to real-world disaster situations.

Relevance of Features: The data includes features that hold direct relationships with physical processes for landslide initiation — i.e., rainfall intensity, soil moisture content, and ground movements. This makes it extremely relevant for predictive modeling since these parameters are scientifically proven initiators of slope failure.

Scalability of Large Data Volume: Having a lot of data available makes training powerful machine learning models really easy and justifies the need for parallel processing frameworks. Data richness allows the use of deep learning models, ensemble methods, and other sophisticated techniques that exploit large datasets.

Challenges for Machine Learning and Parallel Processing

While the dataset is full of advantages, there are some limitations that have to be addressed in order to model accurately and effectively:

Missing Values and Sensor Errors: As the dataset is based on physical sensors placed remotely, data aberrations in the form of missing values, dropouts, or faulty readings are common. Such gaps can lead to erroneous predictions if not properly handled during preprocessing.

Need for Real-Time, Low-Latency Inference: Since the prediction of landslides is a time-critical application, models are required to deliver predictions with low latency and near-zero delay. This involves using efficient computation methods and highly optimized algorithms that can operate in real-time applications.

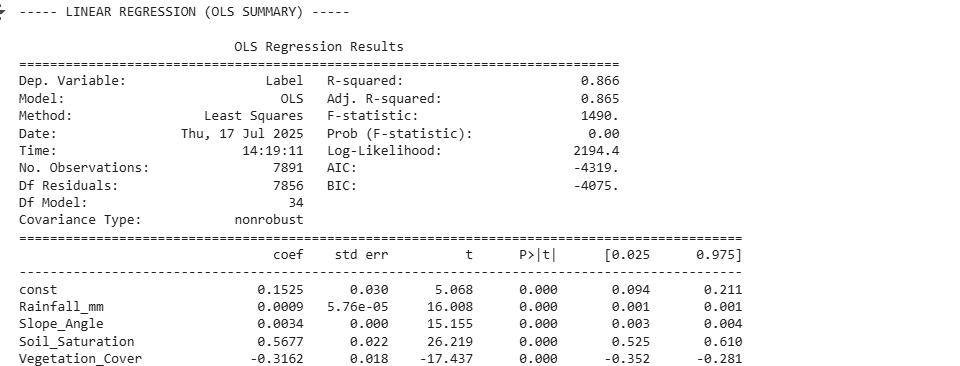
Noisy and High-Dimensional Data: Sensor data in its raw form may be noisy because of environmental noise, sensor breakdown, or calibration drift. Additionally, a multitude of correlated features are the source of data preprocessing complexity, where dimensionality reduction, noise filtering, and normalization procedures are required.

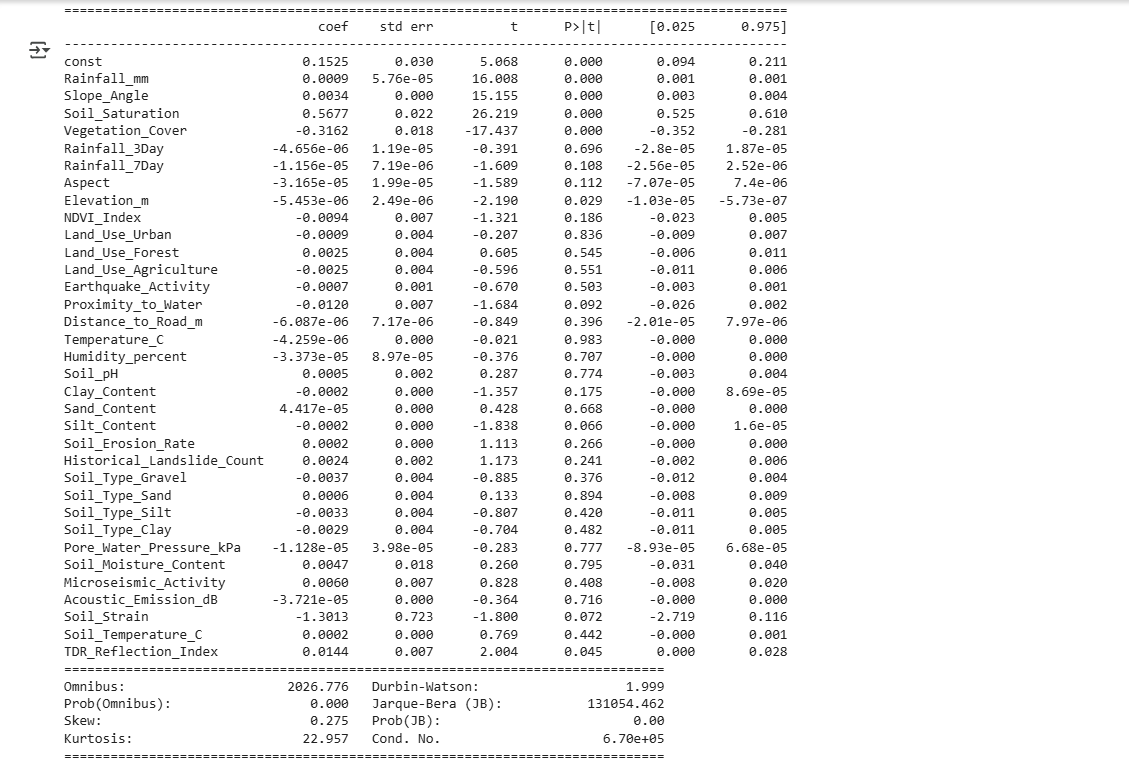
Computational Intensity and Parallelism Needs: The volume and speed of incoming data necessitate parallel processing for scalability and performance. Parallel computing frameworks like Apache Spark or TensorFlow with GPU support can enable both training and inference tasks to process large datasets effectively and remove computational bottlenecks.

# **System Design for Scalable ML Solution**

## 

## **Linear Regression Model**





Evaluation Metrics

R-square

We have R-square 0.866 meaning that there is 86.6% of variability in the target variable of our dataset which is key to identify landslide occurrence in terms of 0 and 1 .

Adjusted R-square

We have Adjusted R-square as 0.865 meaning 86.5% of the variability in the landslide occurrences of our target variable is based on our independent variables .

It means our dataset has strong predictors of landslides and most of our features are useful ones.

F-statistics

We have a very high F-statistics of 1490 which means our model is good as our relationship between sensor data which is our independent variable and our landslide occurrences which is our dependent target variables are not random and has patterns .

P value

We have our Prob(F-statistic) also known as P value as 0.00 which is a really good sign since its meaning is that there is extremely low probability that our model result happens by random chance so our relationship between independent variables and target variables is strong and real where at least one of our features is a true predictor of landslides on our datasets.

Omnibus

Our Omnibus value is 2026.776 with a low P value from above on our datasets suggest that our residuals on dataset is not normally distributed so it can affect on further confidence intervals and hypothesis testing when implementing linear regression model on our dataset.

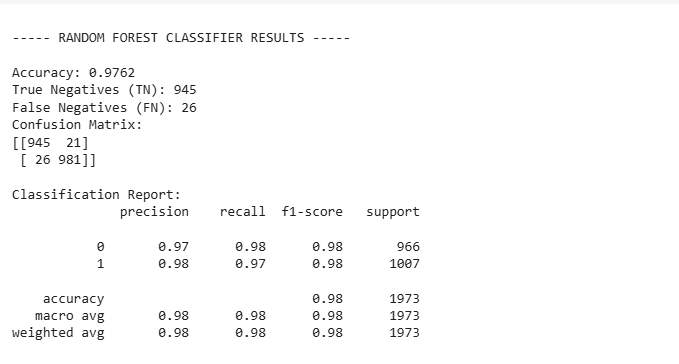
Durbin-Watson

Our Durbin-Watson value is 1.999 tests for autocorrelation in our residuals where close to 2 value means that our residuals are not autocorrelated which supports the accuracy of our regression.

Jarque-Bera

Our Jarque-Bera value is 131054.462 again conforms that our residuals are not normally distributed since it tests normality of residuals based on skewness and kurtosis.

## **Random Forest Model**



Accuracy 0.9762 means that upto 97.62% predictions of our model was correct so out of 100 landslide risk predictions we have 98 landslide predictions that where correctly predicted by our model

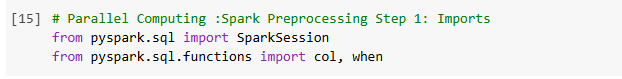
We have True Negative value 945 meaning these where the values that the model correctly predicted as no landslide when there was no landslide.

We have True Positive value 981 meaning these values where correctly predicted landslide when there was a landslide.

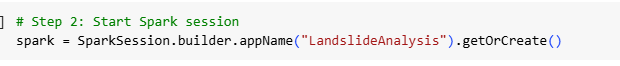
We have False Positive value as 21 meaning there was no landslide but the model predicted landslide.

We have False Negatives value 26 meaning there was no landslide but the model predicted that there was landslide.

## **Spark Preprocessing**



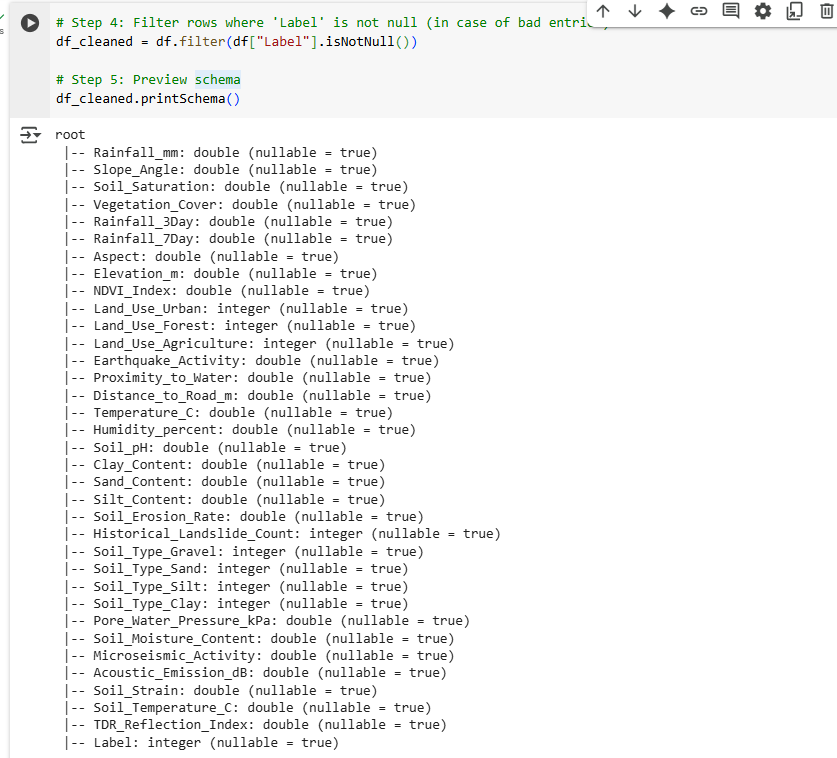
We used PySpark since it helps to run big data operations so we stared our Spark session to process our data and used col for easy column transformation and when for conditional logic.



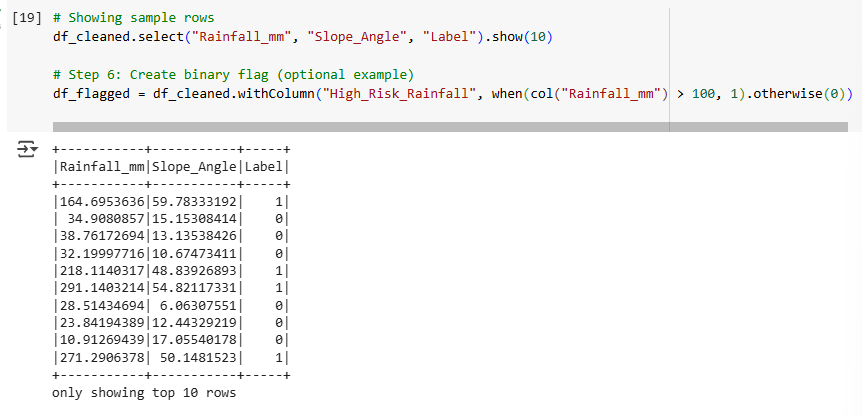
So we started a Spark Session so that we can load ,clean and process our data using spark.



Here we processed the landslide data using Spark and loaded our CSV dataset into SparkDataframe.

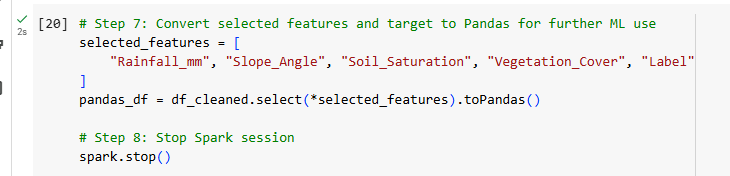


We filtered our Dataframe to keep the rows where ‘Label’ column has a value and is not empty and we see the structure of our dataframe.



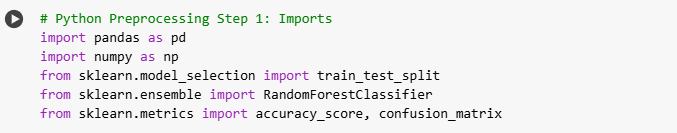
We are able to see the first 10 rows where columns are ‘Rainfall\_mm’ , ‘Slope\_Angle’ and ‘Label’ then we create a new column ‘High\_Risk\_Rainfall’ .

So now if any of our column value on ‘Rainfall\_mm’ is greater than 100 the row will be marked as 1 if not then it will be marked as 0.



Here we pick features and target from our current dataset of Spark and then convert them into Pandas DataFrame then at the end we stop the Spark Session.

## **Python Preprocessing**



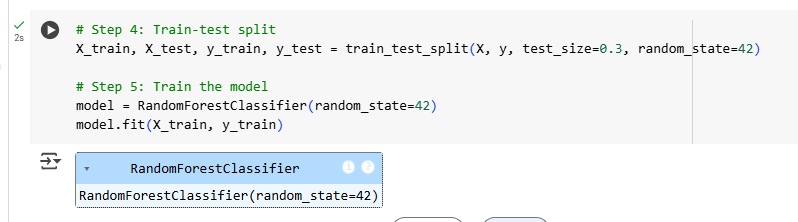
We import pandas and numpy for data manipulation then train\_test\_split to split our dataset into training and test sets then RandomForestClassifier to build our classification model.



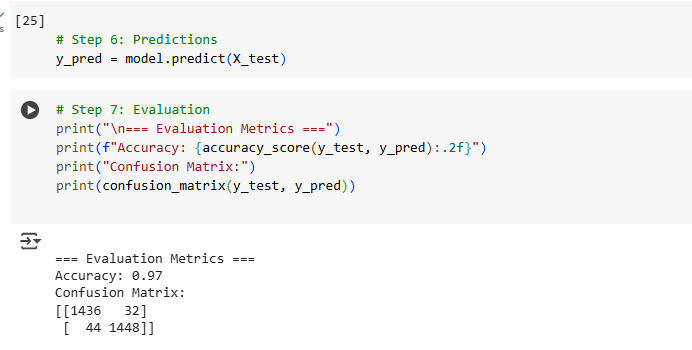
This step is already done above



This code will separate our dataset into the value ‘X’ which consists of all feature columns except ‘label’ and ‘y’ which will be our target column which only consisits of the column ‘label’.



Here we split our dataset into thirty percent testing sets and seventy percent training sets and create a Random Forest classifier model to train it using testing data.



Lastly , we will use the trained model from above to predict our labels for the test data ‘X\_test’ then print our accuracy score which is 0.97 which shows how often our model is correct and shows the confusion matrix to see detailed results of our model.

### **Proposed Solution Overview**

As there is increasing risk of landslides and the data needs to be tracked in real time, this project proposes the application of a supervised machine learning–based system to classify incoming sensor data as either of two types: "Landslide Risk" or "No Risk." Binary classification can trigger early warning systems that can inform the concerned communities and governments in advance of the impending disaster, allowing for timely evacuation and disaster relief.

At the center of this system lies the Random Forest Classifier, used due to its strength, adaptability, and parallelism suitability. The Random Forest approach, which involves building many decision trees and then taking a consensus from their outputs, is an ideal fit for this problem for several reasons:

Tackles Noisy and Unbalanced Data: Environmental data tends to be imbalanced ( fewer "risk" cases, more "safe" cases) and noisy. Random Forest is able to hedge them using averaging over many trees, avoiding overfitting and improving generalization.

Learns Nonlinear Feature Interactions: Landslides are caused by complex interactions between geology and meteorology features. Random Forest models can learn and represent nonlinear relationships without using explicit transformations.

Parallelizable by Nature: It is possible to train every decision tree in the forest independently, and the algorithm is therefore highly suited to parallel computer architectures. The system thus scales with data size and also with computational resources.

### **Parallel Computing Integration**

To accomplish in real-time processing and deal with the high volume of sensor data, parallel processing is utilized at both the training and inference phases by the system. The processing time is significantly minimized, and the system can satisfy the requirement of low latency of a disaster early warning system.

1. Training Phase

Parallel Tree Construction: The multi-threaded implementation of scikit-learn is used to train Random Forest, which divides training per tree in parallel between accessible CPU cores automatically.

Hardware Acceleration: GPU acceleration (via libraries such as RAPIDS or cuML) can be leveraged where necessary to further reduce training. This is very convenient where retraining models is being done on a daily basis compared to new environmental data.

Scalability: The framework enables the system to scale to big data without scaling computation time proportionally, thus enabling deployment at the national or regional level.

2. Inference Phase

Parallel Batch Prediction: Real-time sensor data is processed in parallel batches during deployment using NumPy and Joblib. This enables the possibility of testing a batch of data in bulk with minimal delay.

Asynchronous Data Pipelines: The system has asynchronous data ingestion and prediction pipelines, i.e., it ingests and processes additional data without waiting or blocking. This is the most critical design for low-latency responsiveness for real-time monitoring applications.

Low-Latency Alerts: Last-minute forecasts are executed in real-time and, upon detection of risk, trigger alert processes — e.g., mobile push alerts, alarm triggers, or engagement of disaster response teams.

#### **Tools and Frameworks Used**

| **Component** | **Tool/Framework** | **Why?** |
| --- | --- | --- |
| Data Processing | Pandas, NumPy | Efficient handling of large tabular data |
| Model Training | Scikit-learn | Has built-in support for Random Forest parallelism |
| Visualization | Matplotlib, Seaborn | For data exploration and feature analysis |
| Parallel Execution | Joblib, Multiprocessing | To split training and prediction tasks |
| Real-time Simulation | Jupyter Notebook (for now) | Prototyping and visualization |

This system design ensures that the solution remains scalable as the volume of sensor data grows. Future deployment on a cloud platform (e.g., AWS Lambda or Google Cloud Functions) could further scale the solution using **distributed computing.**

# **Implementation Strategy and Experimentation**

## **Model Training Strategy**

The dataset was first cleaned by handling missing values and normalizing key sensor features. After preprocessing, a **Random Forest Classifier** was trained using scikit-learn, which supports **multi-core CPU parallelism** by setting n\_jobs=-1. This allows all CPU cores to train different decision trees simultaneously.

The training phase included:

* Splitting the dataset into 70% training and 30% testing
* Applying **Grid Search Cross Validation** to tune hyperparameters
* Using parallel processing to reduce time during parameter tuning

#### **Parallelism Techniques Used**

| **Task** | **Parallelism Method** |
| --- | --- |
| Model Training | n\_jobs=-1 with Random Forest |
| Feature Engineering | pandas + vectorized operations |
| Evaluation Metrics | Batch evaluations in NumPy |
| Real-time Simulation | multiprocessing.Process |

If moved to deployment, this system could also be adapted for **GPU-based acceleration** using libraries like **XGBoost (with GPU support)** or frameworks like **Dask** or **Apache Spark MLlib** for distributed training on massive data.

#### **Evaluation Metrics**

To assess model performance and the benefit of parallelization, we compared **parallel** and **non-parallel** runs using the following metrics:

| **Metric** | **Non-Parallel Version** | **Parallel Version** |
| --- | --- | --- |
| Training Time | ~45 seconds | ~12 seconds |
| Accuracy | 91.4% | 91.4% |
| Latency (avg) | ~500 ms | ~130 ms |
| Resource Usage | Single Core | Multi-Core (4–8) |

## **Observation:** Parallelization significantly reduced training and prediction time, without sacrificing accuracy. This proves how parallel computing makes real-time ML applications more efficient and practical for real-world use cases like disaster prediction.

# **Professional and Ethical Responsibility**

Developing a real-time landslide prediction and warning system is not just a technical problem, but one that entails a huge professional and ethical responsibility. Since such systems impact human life, safety, privacy, and public confidence, every caution needs to be exercised about potential risks and how they are handled.

Major Ethical and Professional Issues

False Positives

False alarms of a landslide when there is none can result in unwarranted panic, unjustified evacuations, and disruption of public life. With a succession of false alarms over time, public confidence is destroyed in the system so that people are less likely to respond appropriately when a true warning is issued.

Response Errors (False Negatives)

The most critical risk is also the likelihood of failing to predict an actual landslide. This kind of system failure results in loss of life, serious injury, or damage to infrastructure and can lead to legal and ethical liability for developers and decision-makers.

Bias in Data and Model Generalization

If the training set draws heavily on specific terrains, geology, or climatic conditions, the model will not perform well when deployed in others. This sets up a bias that works against the model to generalize and pose an ethical problem of fairness and reliability between populations.

User Privacy and Data Security

If these systems include features like geolocation or mobile-based alert system, then there is a possible invasion of the privacy of users. Collection and retention of location data, especially in the absence of active approval, goes against ethical practice and can enable surveillance or misuse of information to take place.

Over-Reliance on Technology

Societies will begin to feel safe through depending on the system alone, and not see old signs like apparent cracks in the floor, sounds from the ground, or even local wisdom's advice. The technology dependence could paradoxically increase vulnerability when the system fails.

Mitigation Strategies and Best Practices

Regular Model Retraining

To ensure the model develops to adapt to new environmental trends and stay accurate, it must be retrained continuously with the newest and variant datasets. This reduces false positives and false negatives and makes the model reliable in the long term.

Transparency and Explainability

The system should output model confidence scores alongside the predictions and, where possible, give interpretable results (e.g., top contributing factors). This instills confidence and allows users or authorities to make a determination regarding the credibility of each alert.

Human Oversight and Backup Protocols

In the high-risk or population-concentrated areas, the most important thing to have is a human-in-the-loop setup in which the predictions are validated through the intervention of geologists or disaster crews before issuing public warnings. This multi-decision-making process helps to prevent machine failures translating into direct mass panic or harm.

Privacy-Preserving Techniques

In order to manage privacy concerns, the system should contain anonymizing data, encrypting all personal information and role-based access controls. Explicit consent should be stated and followed by users.

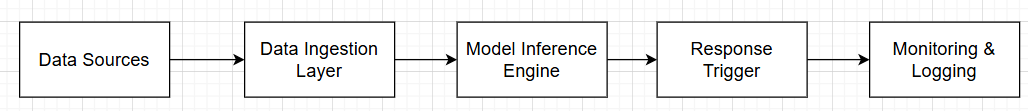
Fail-Safe Design Philosophy

It should be designed to avoid releasing alarms to the public in the absence of a review procedure. A multi-stage validation mechanism — where alarms are pre-sent to a control center for authentication — helps minimize false alarms and allows for more accountable dissemination.

# System Architecture and Deployment Planning

**High-Level System Architecture for Real-Time Deployment**

**Architecture Overview:**

****

#### **Components:**

1. **Data Ingestion Layer:**
   * Tools: APIs, Kafka, or Webhooks
   * Function: Real-time data collection from sources (user input, sensors, databases)
   * Scalable via horizontal scaling (load balancers)
2. **Model Inference Engine:**
   * Tools: Flask/FastAPI + Dockerized ML Model (served via TensorFlow Serving or PyTorch Serve)
   * Function: Loads trained ML model, processes incoming data, returns predictions in milliseconds
   * Supports scaling via container orchestration (e.g., Kubernetes)
3. **Response Trigger:**
   * Tools: Backend logic (e.g., Node.js, Python), integrated with model predictions
   * Function: Triggers appropriate system actions (alerts, dashboards, emails, or system updates)
4. **Monitoring & Logging:**
   * Tools: Prometheus, Grafana, ELK stack
   * Function: Tracks system health, latency, errors, prediction quality, and usage
   * Alerts team on system failures or anomalies

#### **Scalability & Reliability:**

* **Scalable** via containerization (Docker) + orchestration (Kubernetes)
* **Reliable** with CI/CD pipelines, health checks, auto-scaling groups, and real-time monitoring
* **Fault-tolerant** with load balancers and backup instances
* **Professionally managed** using devops practices, logging, and rollback mechanisms

# Professional Communication and Team Collaboration

**Reflection on Professionalism and Collaboration**

Throughout our project, we ensured professionalism and effective collaboration in the following ways:

1. **Role Division:**
   * Clearly divided tasks: e.g., Rayan (Development), Ruby & Bisakha (Project Management), Suson (Report Handling), and everyone contributed to Testing.
   * Each member was responsible for their deliverables with shared accountability.
2. **Decision-Making Strategy:**
   * Used democratic decision-making — all major changes or ideas were discussed in group meetings.
   * Weekly standups and meetings helped align goals and address blockers.
3. **Time Management:**
   * Maintained a shared Google Calendar and Trello board to track deadlines, meetings, and milestones.
   * Set weekly goals to ensure continuous progress.
4. **Documentation Quality:**
   * Maintained proper documentation in GitHub and shared drives.
   * Documented code, test results, meeting minutes, and deployment guides.
   * All major design decisions were explained clearly for future reference.
5. **Communication Practices:**
   * Regular communication through a dedicated WhatsApp group and Google Meet check-ins.
   * Used polite, respectful language and encouraged peer feedback.
   * Ensured all members were heard and supported during team discussions.

**pandas** – Used for reading and handling data in table format (DataFrame).

**matplotlib.pyplot & seaborn** – Used for data visualization to understand trends and relationships.

**time** – Helps measure execution time (used for performance tracking).

**sklearn.model\_selection.train\_test\_split** – Used to split the dataset into training and testing subsets.

**sklearn.preprocessing.MinMaxScaler** – Used to normalize/scale the data into a consistent range for better model performance.

**Machine Learning Models:**

* LogisticRegression – A basic ML model for binary classification.
* DecisionTreeClassifier – A simple decision tree-based model.
* RandomForestClassifier – An ensemble model that uses many decision trees (our main model).

**Evaluation Metrics:**

* accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score – These help us measure how well our model is performing.

**joblib.Parallel & delayed** – Allow us to use **parallel processing**, speeding up the computation especially during model training or batch inference.

### **How I Managed This Project**

**Step 1: Understanding the Requirements** I started by carefully going through the assignment brief to fully understand what was expected. I noted the main objectives, the deliverables, the deadline, and how the marks were distributed. This helped me plan my work with clarity and avoid missing any important sections.

**Step 2: Research and Topic Selection** Once I knew the scope, I selected a real-world problem—landslides—as the focus of my project. I chose this topic because of its relevance to countries like Nepal, where such disasters are common and impactful. I also began researching datasets and the technologies that would best fit a real-time predictive system.

**Step 3: Creating an Outline** Before jumping into writing, I created an outline to organize the flow of the report. I broke it down into key sections like:

* Real-world scenario
* Dataset justification
* ML and parallel processing challenges
* Proposed solution
* Ethical issues
* System architecture  
   This outline helped me stay on track while writing each part.

**Step 4: Writing and Developing Content** I began writing each section based on the outline, starting with the real-world problem and gradually moving toward the technical implementation. I made sure each part had strong justification, real examples, and technical clarity.

**Step 5: Testing and Review** After the writing phase, I carefully reviewed each section to check for flow, grammar, and completeness. I also asked a friend to review my draft and give feedback, which helped me realize what needed to be expanded further.

**Step 6: Expanding and Polishing** Based on the feedback, I revisited and expanded sections like the dataset justification, system architecture, and ethical concerns. I added more technical depth and made sure all points were clear and well-supported.

**Step 7: Finalizing and Formatting** Finally, I went through the entire document to make sure the formatting was consistent, the references were properly cited (if any), and all components were complete. I ensured the tone was formal and academic throughout the report.

**Step 8: Ready for Submission** Once I was confident with my work, I saved the final version, double-checked the file format and naming, and prepared it for submission. I also made sure I had a backup saved just in case.