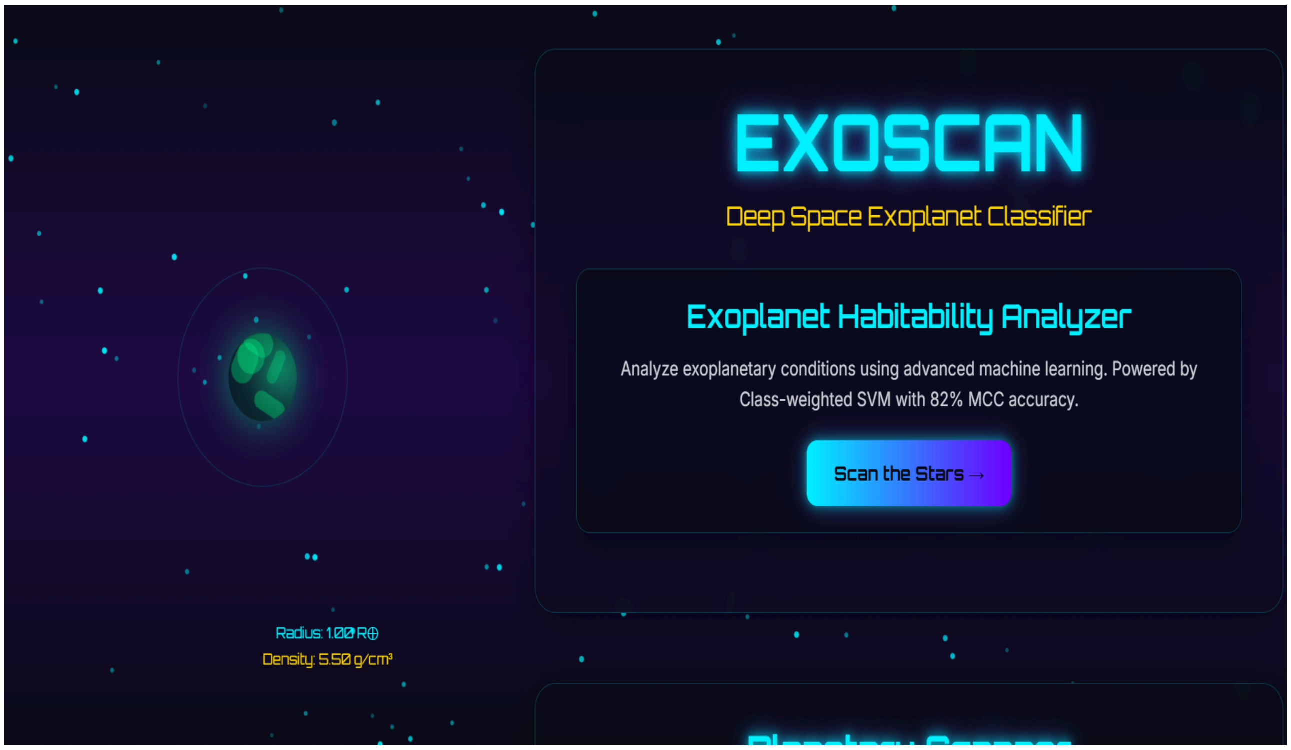
NAME : M.Abirami

DEGREE: B.Tech(Artificial intelligence & Data Science)

COLLEGE NAME: S.A. Engineering College, Tamilnadu

FRONTEND IMAGE:

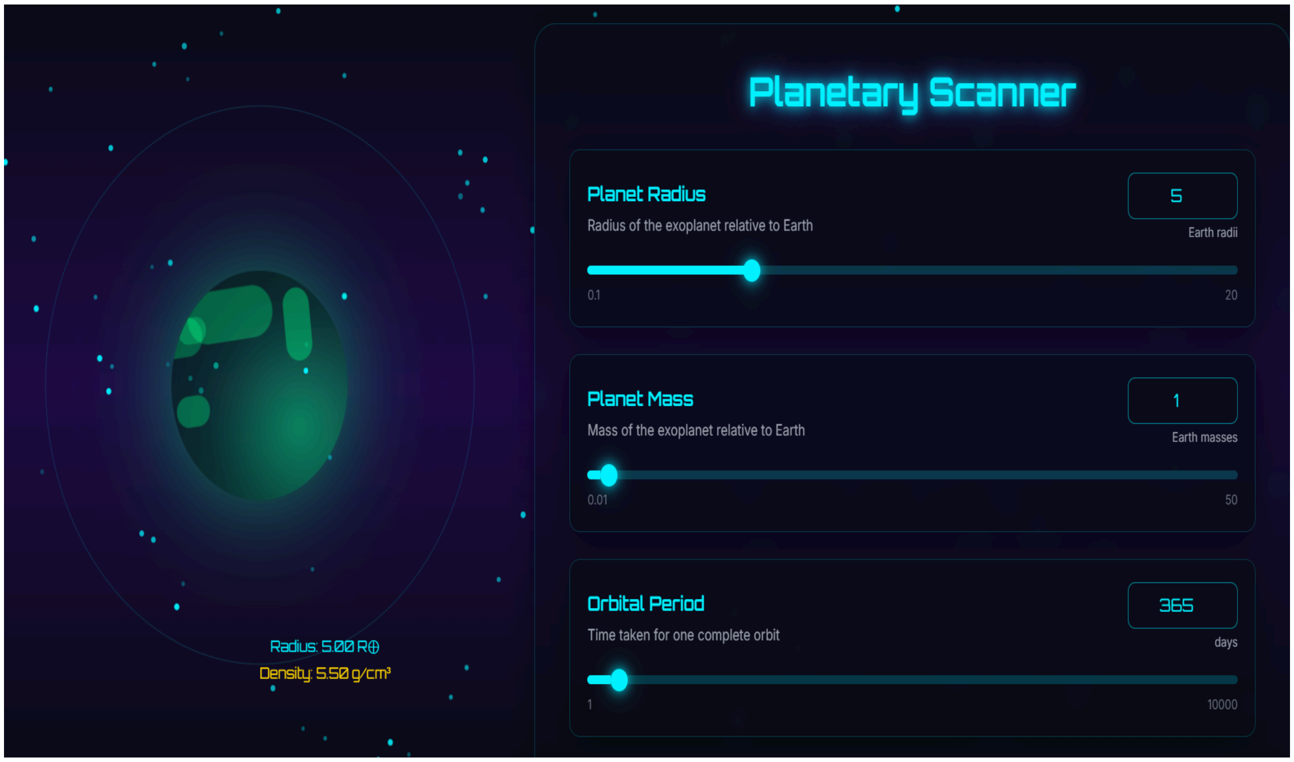


EXOSCAN

Deep Space Exoplanet Classifier

Exoplanet Habitability Analyzer

Analyze exoplanetary conditions using advanced machine learning. Powered by Class-weighted SVM with 82% MCC accuracy.



Planetary Scanner

Planet Radius

Radius of the exoplanet relative to Earth

5 Earth radii

0.1 20

Planet Mass

Mass of the exoplanet relative to Earth

1 Earth masses

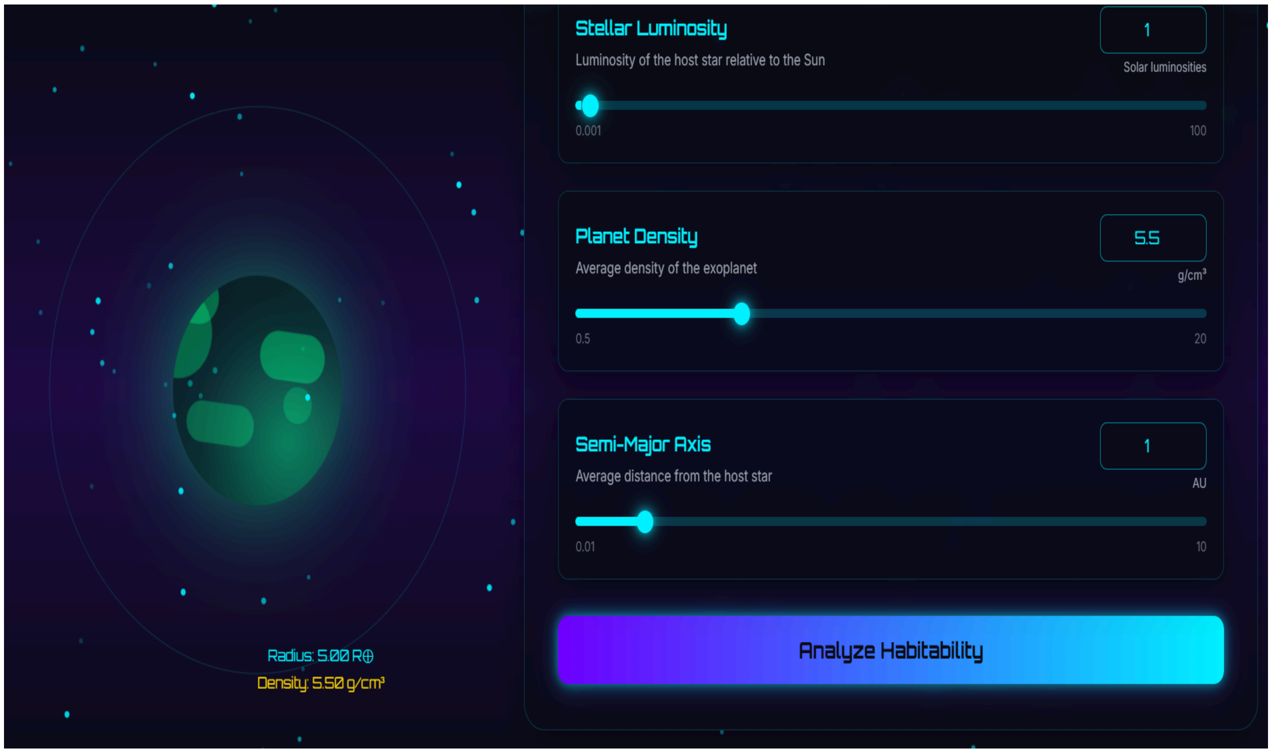
0.01 50

Orbital Period

Time taken for one complete orbit

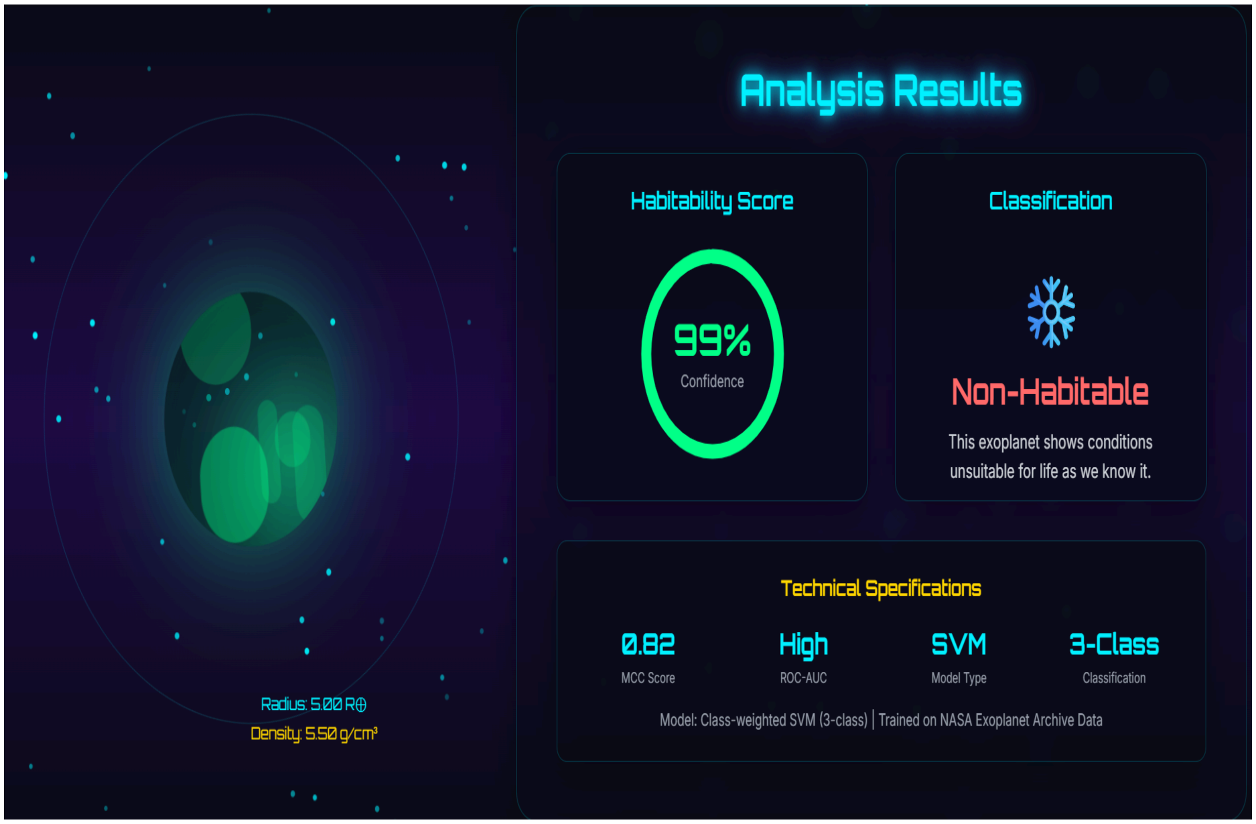
365 days

1 10000


Stellar Luminosity
Luminosity of the host star relative to the Sun
1
Solar luminosities
0.001 100

Planet Density
Average density of the exoplanet
5.5
g/cm³
0.5 20

Semi-Major Axis
Average distance from the host star
1
AU
0.01 10


Analysis Results

Habitability Score
99%
Confidence

Classification
Non-Habitable
This exoplanet shows conditions unsuitable for life as we know it.

Technical Specifications

0.82	High	SVM	3-Class
MCC Score	ROC-AUC	Model Type	Classification

Model: Class-weighted SVM (3-class) | Trained on NASA Exoplanet Archive Data

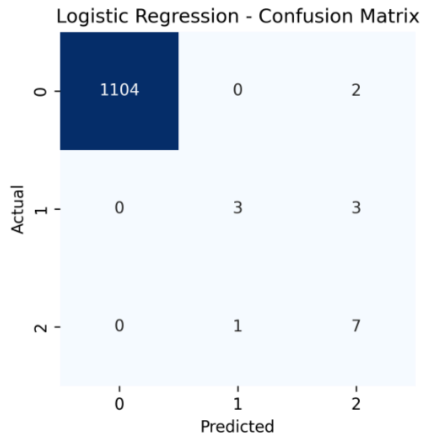
Mile Stone 1 - Completed

- Handling missing values - done
- Scaling /. Preprocessing - done
- Splitting of dataset - done
- Feature Engineering - done

Mile Stone 2 - Completed

- Feature engineering - done
  - Baseline models - Logistic Regression, K-Nearest Neighbors (KNN), and Naive Bayes.
    - From the 3 models Logistic reg works well
    - After logistic reg, naive bayes works well which means it works better than KNN.

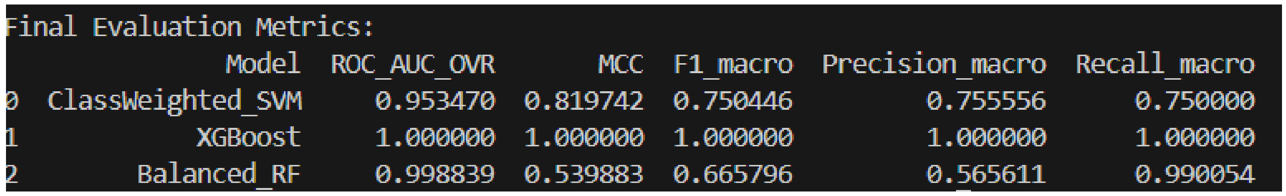
  - Smote is applied after baseline model building to evaluate the performance of the model in cleaned data.
  - Again the baseline model is trained on the smote dataset.
  - Pca and t-sne is performed after smote have been applied and visualized to understand the data dimensionality.
  - Best model : (from baseline)
    - logistics reg : recall : 1:00
      Precision :1 .00
      Macro f1 = 77%
      Accuracy : 99%



- Now I am in the ML pipeline building phase.
  - Models used :
    - classweighted SVM

- Xgboost
- Balanced random forest

Among all other models the Classweighted SVM is the best one.



```
Final Evaluation Metrics:
              Model  ROC_AUC_OVR       MCC  F1_macro  Precision_macro  Recall_macro
0  ClassWeighted_SVM     0.953470  0.819742  0.750446         0.755556      0.750000
1            XGBoost     1.000000  1.000000  1.000000         1.000000      1.000000
2        Balanced_RF     0.998839  0.539883  0.665796         0.565611      0.990054
```

ClassWeighted SVM:

- High MCC (0.82)
- Balanced precision & recall
- Strong ROC -AUC

- Among the evaluated models, the class-weighted SVM achieved the best balance between precision and recall, reflected by a high MCC score of 0.82.
- XGBoost achieved perfect classification scores, indicating strong separability but also potential overfitting risks.
- Balanced Random Forest prioritized recall for minority classes at the cost of precision, making it suitable for recall-critical scenarios.

Mile Stone 3 - Completed

BACKEND :

- Backend - For Backend I Built a Flask web server to get the real time data.
- Created API endpoints that take in planetary data (like mass and radius) and send back a habitability prediction.

FRONTEND UI DEVELOPMENT:

- Designed a clean and easy-to-use website using HTML, CSS, and Bootstrap that works well on both computers and phones.
- Created a user-friendly form where anyone can type in space data to get a prediction.

GIT UPDATED TILL MILESTONE 3

WEEK 1: Module 1: Data Collection and Management

WEEK 1: Day 1 (1.12.2025)

- Source link 1(From Kaggle) : https://www.kaggle.com/datasets/shivamb/all-exoplanets-dataset
- Observation :
    1. Contain 4500+ rows
    2. Contain lots of missing values
    3. Contain all the features mentioned in the Problem statement document.
- Useful if its cleaned
- Question: Do I need all the 4000+ data for analysis or having 2000 data enough ?
- Commends: Optional dataset as it contains lots of missing data.

- Source link 2(From NASA Exoplanet Archive): https://exoplanetarchive.ipac.caltech.edu/cgi-bin/TblView/nph-tblView?app=ExoTbls&config=PS
- Observation:
    1. Contain nearly 40K rows, 90+ columns.
    2. This dataset too contains all the required features for analysis.
    3. This dataset contains briefs of each column in the beginning for user understanding, As these features were shortened in actual column name which is not understandable.
    4. Contain a few missing data.
- Useful : yes (I planned to work with this dataset)
- Question: Do we need categorical data for our analysis?
  Still this dataset contains few columns of categorical data removing those will affect ?

WEEK 1: Day 2 (2.12.2025) - no class

WEEK 1: Day 3 (3.12.2025)

REQUIRED FEATURES : Planet radius, mass, density, surface temperature, orbital period, distance from star, Host star type, luminosity, temperature, metallicity.

- **Planet Radius** : It is used to predict the Size and type of planet, if a planet is rocky then it is fit for habitability.
- **Planet mass:** Mass tells how much material the planet has.It influences **gravity**, atmosphere, and temperature. It should have a balanced mass to support habitability.
- **Density:** Density helps identify the type of planet
    - High density → rocky (good for life)
    - Low density → gas planet (not habitable)
- **Temperature:** It tells about the water state, in which a temp range should be in a certain range to support habitability, if temp is too high → water cannot exist or if it is too low then water will freeze.

- ➔ Even though both the dataset contain the required features i'm gonna choose source link 2(Nasa exoplanet dataset) as it contains few missing values compared to the kaggle dataset.
- ➔ Question : Do I need all the rows of data for analysis ?
- ➔ Question : Is this we gonna apply preprocessing in the program itself ?
- ➔ Dataset having habitability/ Non Habitable column: NO(Still searching)

WEEK 1: Day 4 (4.12.2025)

Today's task is all about finding techniques to make our dataset balanced, as it is imbalanced with more non-habitable values and very few numbers of habitable values.

- Techniques i found useable is :

    1.SMOTE :

    - ➔ SMOTE creates new synthetic habitable planets based on existing ones.
    - ➔ But it is not creating duplicates.
    - ➔ Random oversampling duplicates the same minority samples which causes biased outcomes.
    - ➔ SMOTE fixes this by:
    - ➔ Creating new, slightly different samples
      Based on the existing minority (habitable) samples
      So the model learns better and generalizes better

2. Combination of Undersampling + SMOTE

- ➔ First, use SMOTE → add synthetic habitable planets
- ➔ Then, undersample majority lightly → remove only a few 0s
- ➔ Technique name: SMOTEENN or SMOTETomek

OUTCOME : I Will do research about working of both the above techniques.

WEEK 1: Day 5 (5.12.2025)

- Today had a discussion about Descriptive statistic
- And how distribution, percentile and percentage differ
- Next I have to do practical visu of univariate and bivariate analysis using exoplanet dataset.

WEEK 2: Module 2: Data Cleaning and Feature Engineering

WEEK 2: Day 1 (8.12.2025)

Week 2 : Goal

• Handle missing values, outliers, and inconsistent entries.
• Encode categorical features (e.g., star type) using one-hot encoding.
• Feature engineering examples:
• Habitability Score Index based on key planetary parameters.
• Stellar Compatibility Index to measure host star influence.
• Normalize numerical features.
• Validate data quality using descriptive statistics and visualization.

I need to analyse the research paper and get to know about the algorithms and concepts they have used in those research paper and understand how they have approached the project.

- lhtm
- /uploads/V3ISSUE2/ijrpr2746-detection-of-exoplanets-using-machine-learning.pdf
- https://arxiv.org/pdf/2011.14135
- https://www.kaggle.com/code/nickoreese/exoplanet-habitability-prediction

Component	Malik et al. (arXiv)	IJRPR Paper	Kaggle Notebook
Data Type	Light curves	Light curves / Kepler	Exoplanet physical catalog
Feature Extraction	TSFresh (789+ features), FFT components, folded LC features	Light curve preprocessing, sometimes CNN encodings	PCA, t-SNE, derived physical features
Models Used	LightGBM (main), Logistic Regression, Random Forest	SVM, Random Forest, CNN, Autoencoder	Logistic Regression, Random Forest, XGBoost, SVM, KNN
Imbalance Handling	Threshold tuning, recall focus	Not strong, usually normal	SMOTE, class weights, balanced splits
Evaluation Metrics	ROC-AUC, PR-AUC, Recall	Accuracy, Precision/Recall	ROC-AUC, F1, PR-AUC
Goal	Detect transit signals	Compare ML models for detection	Predict habitability class

WEEK 2: Day 2 (9.12.2025)

Reading task

WEEK 2: Day 3 (10.12.2025)

Github branch creation - Joined the project GitHub repository and created a new branch to manage my works and updates. Successfully pushed the initial project files and scripts to the branch to ensure all work is version-controlled and backed up.

WEEK 2: Day 4 (11.12.2025)

GIT :

- The project repository was set up on GitHub, and relevant source code and dataset files were pushed to the assigned branch.
- Pushed some files, dataset and coding files in my github branch.

WEEK 2: Day 5(12.12.2025)

Initially I Performed data processing using the Python programming language within the Visual Studio Code.Performed initial data cleaning and preprocessing on the exoplanet dataset which contain more missing values, used Pandas functions to handle the data for missing values and duplicates.I Used the mean and median functions for handlng the numerical values and mode for categorical values in my dataset.

WEEK 3: Module 3: Machine Learning Dataset Preparation

• Split data into training and testing sets (e.g., 80:20).
• Select features for model input based on correlation with habitability.
• Define target variable: Habitability class (Habitable/Non-Habitable) or habitability score.
• Create data pipelines with scaling, encoding, and feature selection.

WEEK 3: Day 1(15.12.2025):

Again checked the consistency of the dataset I have.I ran the preprocessing code one more time to double-check the results. I wanted to make sure the data stayed consistent and that the cleaning process worked exactly the same way again without any errors. I've spent the few days cleaning the data I have and making sure the data is actually consistent for training.

WEEK 3: Day 2(16.12.2025):

Partitioned the dataset into an 80:20 ratio for the train-test data set. To address the imbalance in the classes of the dataset, I applied the SMOTE technique after performing splitting of data. This smote technique will create a synthetic minority class. Here, it creates synthetic Habitable data without duplicating it. It helps to keep my dataset balanced.

WEEK 3: Day 3(17.12.2025)

Task Overview

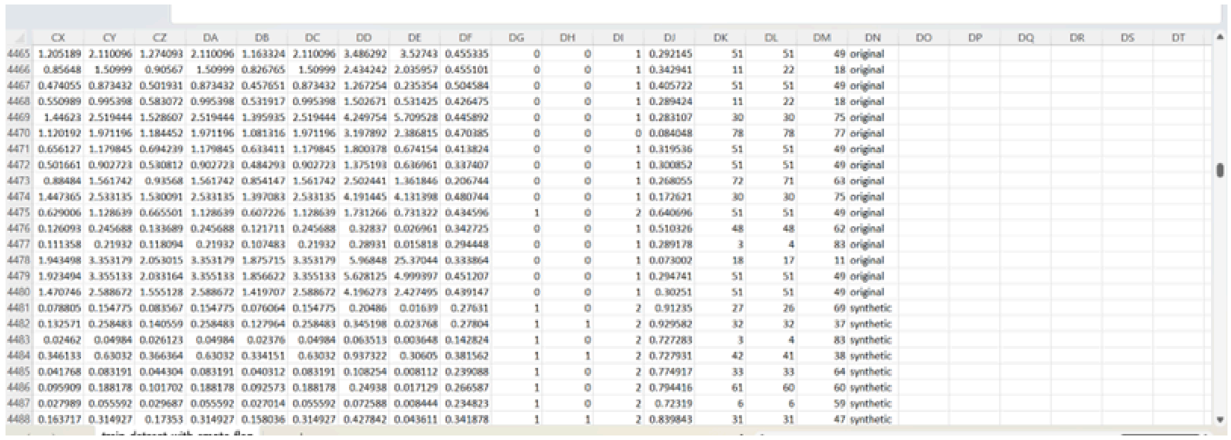
On this day, the focus was on improving the transparency and interpretability of the training dataset after applying data balancing techniques.

SMOTE Application and Data Segregation

- Applied SMOTE to balance the training dataset.
- Added a `smote_flag` column to distinguish original samples from synthetic samples.
- Maintained separation between analytical metadata and model input features.
- Improved interpretability and auditability of the training dataset.

Outcome

- Achieved improved dataset transparency by maintaining clear traceability between real and synthetic data points.
- Established a clean and reproducible workflow for handling imbalanced data while preserving the integrity of the machine learning pipeline.



	CX	CY	CZ	DA	DB	DC	DD	DE	DF	DG	DH	DI	DJ	DK	DL	DM	DN	DO	DP	DQ	DR	DS	DT
4465	1.205189	2.110096	1.274093	2.110096	1.163324	2.110096	3.486292	3.52743	0.455335	0	0	1	0.292145	51	51	49	original						
4466	0.85648	1.50999	0.90567	1.50999	0.826765	1.50999	2.434242	2.035957	0.455101	0	0	1	0.342941	11	22	18	original						
4467	0.474055	0.873432	0.501931	0.873432	0.457651	0.873432	1.267254	0.235354	0.504584	0	0	1	0.405722	51	51	49	original						
4468	0.550989	0.995398	0.583072	0.995398	0.531917	0.995398	1.502671	0.531425	0.426475	0	0	1	0.289424	11	22	18	original						
4469	1.44623	2.519444	1.528607	2.519444	1.395935	2.519444	4.249754	5.709528	0.445892	0	0	1	0.283107	30	30	75	original						
4470	1.120192	1.971196	1.184452	1.971196	1.081316	1.971196	3.197892	2.386815	0.470385	0	0	0	0.084048	78	78	77	original						
4471	0.656127	1.179845	0.694239	1.179845	0.633411	1.179845	1.800378	0.674154	0.413824	0	0	1	0.319536	51	51	49	original						
4472	0.501661	0.902723	0.530812	0.902723	0.484293	0.902723	1.375193	0.636961	0.337407	0	0	1	0.300852	51	51	49	original						
4473	0.88484	1.561742	0.93568	1.561742	0.854147	1.561742	2.502441	1.361846	0.206744	0	0	1	0.268055	72	71	63	original						
4474	1.447365	2.533135	1.530091	2.533135	1.397083	2.533135	4.191445	4.131398	0.480744	0	0	1	0.172621	30	30	75	original						
4475	0.629006	1.128639	0.665501	1.128639	0.607226	1.128639	1.731266	0.731322	0.434596	1	0	2	0.640696	51	51	49	original						
4476	0.126093	0.245688	0.133689	0.245688	0.121711	0.245688	0.32837	0.026961	0.342725	0	0	1	0.510326	48	48	62	original						
4477	0.111358	0.21932	0.118094	0.21932	0.107483	0.21932	0.28931	0.015818	0.294448	0	0	1	0.289178	3	4	83	original						
4478	1.943498	3.353179	2.053015	3.353179	1.875715	3.353179	5.96848	25.37044	0.333864	0	0	1	0.073002	18	17	11	original						
4479	1.923494	3.355133	2.033164	3.355133	1.856622	3.355133	5.628125	4.999397	0.451207	0	0	1	0.294741	51	51	49	original						
4480	1.470746	2.588672	1.555128	2.588672	1.419707	2.588672	4.196273	2.427495	0.439147	0	0	1	0.30251	51	51	49	original						
4481	0.078805	0.154775	0.083567	0.154775	0.076064	0.154775	0.20486	0.01639	0.27631	1	0	2	0.91235	27	26	69	synthetic						
4482	0.132571	0.258483	0.140559	0.258483	0.127964	0.258483	0.345198	0.023768	0.27804	1	1	2	0.929582	32	32	37	synthetic						
4483	0.02462	0.04984	0.026123	0.04984	0.02376	0.04984	0.063513	0.003648	0.142824	1	0	2	0.727283	3	4	83	synthetic						
4484	0.346133	0.63032	0.366364	0.63032	0.334151	0.63032	0.937322	0.30605	0.381562	1	1	2	0.727931	42	41	38	synthetic						
4485	0.041768	0.083191	0.044304	0.083191	0.040312	0.083191	0.108254	0.008112	0.239088	1	0	2	0.774917	33	33	64	synthetic						
4486	0.095909	0.188178	0.101702	0.188178	0.092573	0.188178	0.24938	0.017129	0.266587	1	0	2	0.794416	61	60	60	synthetic						
4487	0.027989	0.055592	0.029687	0.055592	0.027014	0.055592	0.072588	0.008444	0.234823	1	0	2	0.72319	6	6	59	synthetic						
4488	0.163717	0.314927	0.17353	0.314927	0.158036	0.314927	0.427842	0.043611	0.341878	1	1	2	0.839843	31	31	47	synthetic						

WEEK 3: Day 4(18.12.2025)

Feature Selection, Target Variable Definition, and Machine Learning Pipeline Development

Task Overview

The primary objective of this task was to define the target variable, identify the most relevant features influencing planetary habitability, and construct a robust machine learning pipeline to ensure consistent preprocessing and model training.

Feature Selection and ML Pipeline Implementation

- Defined planetary habitability as the target classification variable.
- Performed supervised feature selection to identify key habitability-related features.

- Built an end-to-end machine learning pipeline integrating scaling, feature selection, and SVM classification.
- Evaluated model performance using standard classification metrics.

Outcome

- Successfully established a clean and actionable ML pipeline.
- Improved model interpretability by identifying key features correlated with planetary habitability.
- Enhanced prediction robustness through standardized preprocessing and pipeline-based modeling.

WEEK 3: Day 5(19.12.2025):

Review and Analysis of Model Outcomes

- Reviewed all previously implemented stages of the project, including data preprocessing, SMOTE-based class imbalance handling, feature selection, and machine learning pipeline development.
- Analyzed model outputs such as class distributions, confusion matrix, classification metrics, and selected features to understand their implications on planetary habitability prediction.
- Assessed overall project progress and model working.
- Identified and researched the meaning of complex technical terms encountered during analysis and evaluation.

WEEK 3 Check Point :

1. I performed Data validation in which the percentage of missing values in each column is visualized .
2. I did Data preprocessing in which the numerical and categorical data is handled with statistical concepts such as mean, median and mode.Defined a threshold of 40 to remove a col containing 40 or more than 40% of missing values.
3. Splitting of dataset into 80: 20 for training and testing before applying SMOTE
4. Feature selection is performed with target column
5. ML pipeline building

From module 2 and 3 - 70% of task is done.

WEEK 4: Module 4: AI Model for Habitability Prediction

WEEK 4: Day 1(22.12.2025):

- Trained baseline classification models including Logistic Regression, K-Nearest Neighbors (KNN), and Naive Baye**s** on the cleaned but imbalanced dataset.
- SMOTE was then applied exclusively to the training data to address class imbalance.
- The same classifiers were retrained using the SMOTE-balanced dataset to ensure fair and consistent comparison.

- Evaluated model performance using precision, recall, F1-score, and confusion matrices.
- Observed high accuracy across models but low recall for minority classes, confirming the misleading nature of accuracy in imbalanced datasets.
- Identified Logistic Regression as the strongest baseline model based on macro-averaged F1-score and minority class recall.

WEEK 4: Day 2(23.12.2025):

- After that pca and t-SNE were performed :.
- Applied t-SNE visualization to project high-dimensional feature space into two dimensions.
- Analyzed minority class separability and observed significant overlap between habitable and non-habitable classes.
- Used visualization results to explain the difficulty of minority class classification and limited model performance.
- Pca shows that habitable planets are rare and very similar to one another, while non-habitable planets are common and vary wildly..

WEEK 4: Day 3(24.12.2025):

I spent today building out the full ML pipeline, which includes scaling, sampling, and the final model fitting. By wrapping everything into one pipeline with Stratified K-Fold, I have made sure the validation data stays completely unseen during the sampling process.

WEEK 4: Day 4(25.12.2025):

Christmas Holiday

WEEK 4: Day 5(26.12.2025):

Coding work

WEEK 4 Checkpoint :

Mile Stone 1 - Completed

- Handling missing values - done
- Scaling /. Preprocessing - done
- Splitting of dataset - done
- Feature Engineering - done

Mile Stone 2 :

- Feature engineering - done
    - Baseline models - Logistic Regression, K-Nearest Neighbors (KNN), and Naive Bayes.
        - From the 3 models Logistic reg works well
        - After logistic reg, naive bayes works well which means it works better than KNN.

    - Smote is applied after baseline model building to evaluate the performance of the model in cleaned data.
    - Pca and t-sne is performed after smote have been applied.
    - Best model :
    - logistics reg : recall : 1:00
      Precision :1 .00
      Macro f1 = 77%
      Accuracy : 99%
- Now I am in the ML pipeline building phase. Getting some error in this phase will sort out this issue and provide the evaluation metrics.
    - Models used :

- classweighted SVM
- Xgboost
- Balanced random forest

WEEK 5: Module 5: Flask Backend API

WEEK 5: Day 1(29.12.2025):

- Started the backend by knowing some basic and api endpoints and tools for backend creation. The backend of the project was developed using FastAPI.
  s
- A machine learning model is loaded in the backend to make predictions.

- API endpoints were created to support communication between frontend and backend.

WEEK 5: Day 2(30.12.2025):

Coding work

WEEK 5: Day 3(31.12.2025):

- Finished the Backend Design and implemented a RESTful API using FastAPI to handle prediction requests.

- Created structured endpoints to accept input features from the frontend in a clean and validated format.

- Integrated a machine learning model for habitability prediction using pre-trained artifacts loaded with `joblib.`

WEEK 5: Day 4(01.01.2026):

New year Holiday

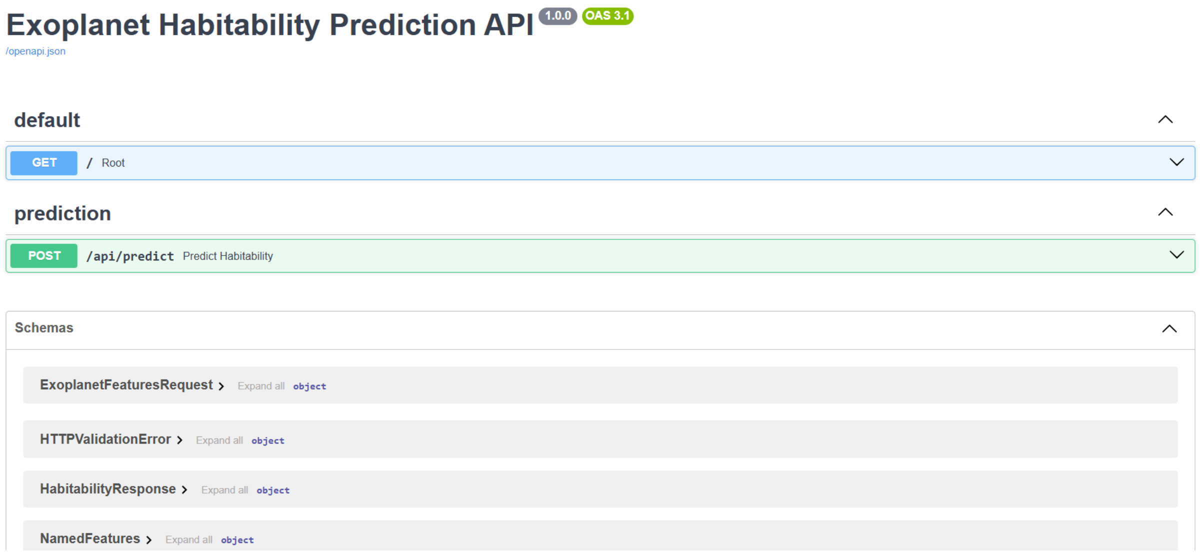
WEEK 5: Day 5(2.01.2026):

- Wrote Python code for API routes, request schemas, and response handling.

- Implemented model loading and inference logic.
- Handled backend error messages and edge cases to ensure stability.

WEEK 3 Checkpoint :

Outcome of this phase:



Exploplanet Habitability Prediction API 1.0.0 OAS 3.1

/openapi.json

default

GET / Root

prediction

POST /api/predict Predict Habitability

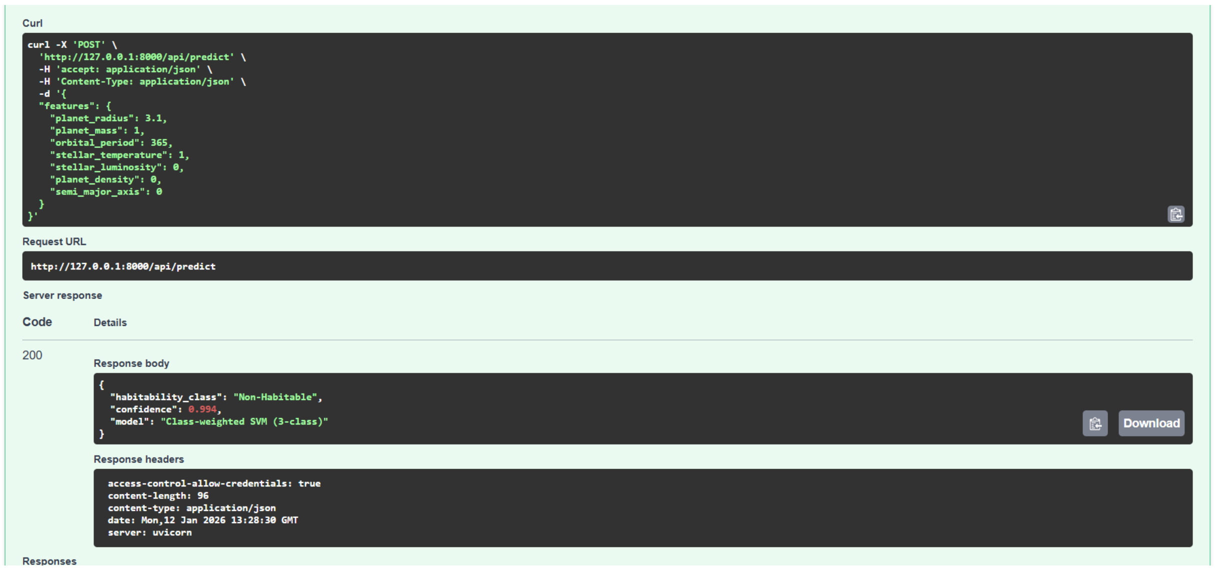
Schemas

ExoplanetFeaturesRequest Expand all object

HTTPValidationError Expand all object

HabitabilityResponse Expand all object

NamedFeatures Expand all object



Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/api/predict' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "features": {
    "planet_radius": 3.1,
    "planet_mass": 1,
    "orbital_period": 365,
    "stellar_temperature": 1,
    "stellar_luminosity": 0,
    "planet_density": 0,
    "semi_major_axis": 0
  }
}'
```

Request URL

```
http://127.0.0.1:8000/api/predict
```

Server response

Code Details

200

Response body

```
{
  "habitability_class": "Non-Habitable",
  "confidence": 0.994,
  "model": "Class-weighted SVM (3-class)"
}
```

Download

Response headers

```
 access-control-allow-credentials: true
 content-length: 96
 content-type: application/json
 date: Mon,12 Jan 2026 13:28:30 GMT
 server: uvicorn
```

Responses

To learn :

1.What is the Rest API ?

A REST API works like a messenger between two applications. When a user performs an action (such as opening a webpage or clicking a button), the client sends a request to the server through a REST API. The server processes this request, accesses the required data, and sends back a response, usually in JSON format.

For example, when a mobile app requests student details, it sends a GET request to the REST API. The server fetches the student data from the database and returns it to the app. REST APIs are stateless, meaning each request contains all the information needed, making them fast, scalable, and easy to use in web, mobile, and AI-based applications.

2.What is Synchronous Request and Asynchronous Request?

Synchronous Request :

A synchronous request is a type of request in which the client waits until the server completes the request and sends a response before moving to the next task.In a synchronous request, the client sends a request and then pauses all other operations until the response is received. This means the program execution is blocked during this time.

Asynchronous Request :

An asynchronous request is a type of request in which the client does not wait for the server’s response and continues executing other tasks while the request is being processed.

In an asynchronous request, the client sends a request and continues working. Once the server finishes processing, it sends the response back, and the client handles it using callbacks, promises, or async/await mechanisms.

3. Difference between Multithreading and Concurrency.

Multithreading:

Multithreading is the ability of a program to run multiple threads at the same time within a single process, where each thread executes a part of the program independently.

A thread is the smallest unit of execution inside a process. In multithreading, a program is divided into multiple threads so that tasks can be executed in parallel.

Concurrency:

Concurrency is the ability of a system to manage multiple tasks at the same time, by switching between them, even if they are not executing simultaneously.
Concurrency is about structure and design, not actual simultaneous execution. The CPU quickly switches between tasks, giving the illusion that many tasks are running at once.

4.How Many Threads Can Run in Parallel?

The number of threads that can run in parallel is equal to the number of CPU cores (or logical cores with hyperthreading).

- If a processor has 16 cores,
  Maximum 16 threads can run truly in parallel

- Extra threads will wait or be time-sliced.
- Only 16 threads can run in parallel on a 16-core processor.

5.How Many Concurrent Tasks Can Run in Parallel?

Concurrency is about **handling many tasks**, not executing them at the same instant.

A system can handle **a very large (theoretically unlimited) number of concurrent tasks**, but only a limited number can execute in parallel based on CPU cores.

The OS scheduler switches between tasks

Tasks may be waiting for I/O, network, timers

This allows thousands or millions of concurrent tasks.

6.What is the difference between flask and fast api?. If we have flask , why do we need fastapi , any advantage in terms of asynchronous?

Flask

- Flask is a lightweight, synchronous Python web framework used to build web applications and REST APIs easily with minimal setup.
- Flask was designed to be simple and flexible. It follows a “do-it-yourself” approach where developers choose their own tools for validation, authentication, and documentation.
- By default, Flask handles requests synchronously, meaning one request is processed per worker at a time.

FastAPI

- **FastAPI** is a modern, high-performance Python web framework used to build REST APIs, designed for **asynchronous programming** and **automatic API documentation**.
- FastAPI is built on **ASGI** and uses Python’s `async` and `await`. This allows it to handle many requests efficiently, especially when tasks involve waiting (database calls, network calls, ML inference).
- Flask is simple but synchronous by default, while FastAPI is designed for **high-performance asynchronous APIs**, making it better suited for modern, scalable, and data-intensive applications.

Feature	**Flask**	**FastAPI**
Type	Micro framework	Modern async framework
Default execution	Synchronous	Asynchronous
Performance	Moderate	Very high

Async support	Limited	Native
Data validation	Manual	Automatic
API docs	Manual	Auto (Swagger)
Best for	Small apps	High-performance APIs

7.What is Throughput?

- Throughput is the amount of work or number of requests that a system can process successfully in a given period of time.
- Throughput tells us how much output a system can produce over time. In computer systems, it usually means how many tasks, requests, or transactions are completed per second.
- A system with **high throughput** can handle more users or requests efficiently, while a system with **low throughput** becomes slow under heavy load.
- A REST API that processes **1000 requests per second** has a throughput of **1000 RPS**

- A database that completes **500 transactions per minute** has a throughput of **500 TPM**

Throughput in Web Applications & APIs:

- Number of HTTP requests handled per second

- Number of users served per minute

- Number of API calls completed successfully

8.What is Inference Speed?

- **Inference speed** is the time taken by a trained machine learning or deep learning model to **generate a prediction (output) from a given input**.
- After a model is trained, it is used to make predictions on new data—this process is called **inference**.
  Inference speed measures **how fast** the model can produce this prediction.
- It is a critical factor in **real-time applications** where quick responses are required.
- Inference speed refers to the time required by a trained machine learning model to produce an output for a given input.

WEEK 6: Module 6: Frontend UI Development

WEEK 6: Day 1 (5.01.2026):

- Built an interactive user interface using React components using the cursor tool.
- Designed input forms to collect planetary feature values from users.
- Integrated the frontend with the backend API using Axios for HTTP requests.
- Implemented proper API request formatting to match backend expectations.

WEEK 6: Day 2 (6.01.2026):

Coding work

WEEK 6: Day 3 (7.01.2026):

- Wrote JavaScript and JSX code for UI components and state management.
- Implemented API service functions for backend communication.
- Managed asynchronous API calls and response handling.
- Structured the frontend codebase for maintainability and clarity.

WEEK 6: Day 4 (8.01.2026):

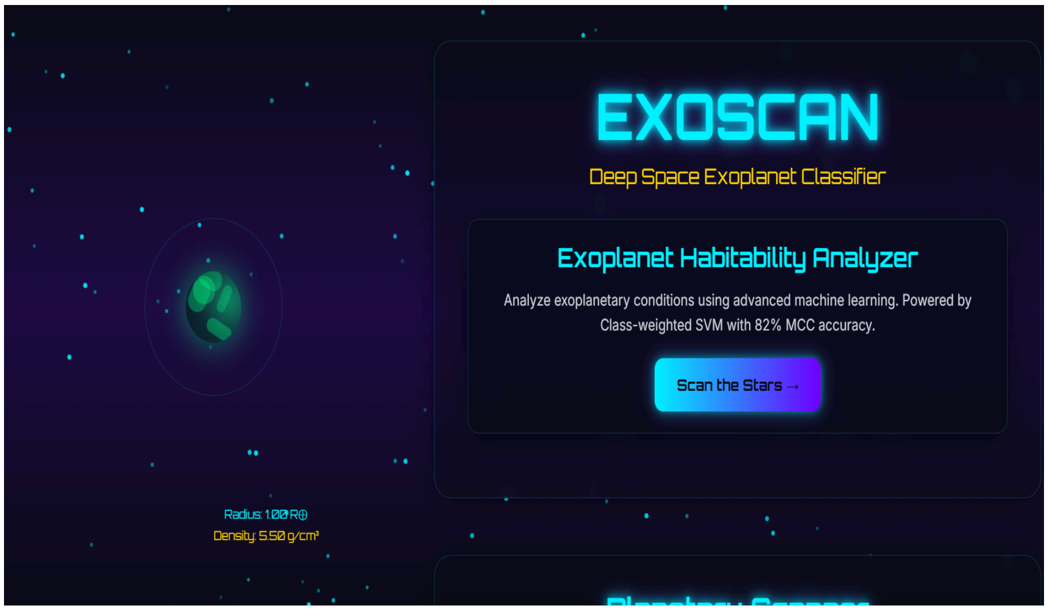
Frontend and backend connection testing through api endpoint.

- Successfully connected the React frontend with the FastAPI backend.
- Verified end-to-end data flow from user input to model prediction output.
- Tested API endpoints independently and through the frontend interface.

WEEK 6: Day 5 (9.01.2026):

Successfully connected the React frontend with the FastAPI backend.

Outcome of this phase:



EXOSCAN

Deep Space Exoplanet Classifier

Exoplanet Habitability Analyzer

Analyze exoplanetary conditions using advanced machine learning. Powered by Class-weighted SVM with 82% MCC accuracy.

WEEK 7: Module 7: Visualization & Dashboard