PHASE 1 – BRAINSTORMING

Problem Statement

Startup investments involve high financial risk. Investors often struggle to determine whether a startup will succeed due to limited predictive tools. The goal of this project is to build a machine learning system that predicts startup success using structured historical data such as funding amount, founding year, number of employees, and funding type (VC/Angel).

The system should output:

- Prediction (Successful / Failed)
- Success Probability percentage

This allows investors and entrepreneurs to make data-driven decisions.

Output of the System

Your system outputs:

Prediction: Successful Startup / Failed Startup
Probability score (e.g., 25% success probability)

Displayed in frontend:
Text
Prediction: Failed Startup
Success Probability: 25%

Technologies Used (Based on Your Code)

From your implementation:

- Backend:
- Python
- Pandas
- NumPy
- Scikit-learn
- Joblib (for model saving)
- Flask (backend app.py serving API)
- Frontend:
- HTML
- CSS
- JavaScript (async fetch API call)

Model Files:

startup_success_model.pkl
feature_columns.pkl

Project Structure:

- data_collection.py
- eda.py
- preprocessing.py
- feature_engineering.py
- train.py
- backend/app.py
- frontend/index.html

PHASE 2 – REQUIREMENT ANALYSIS

Functional Requirements

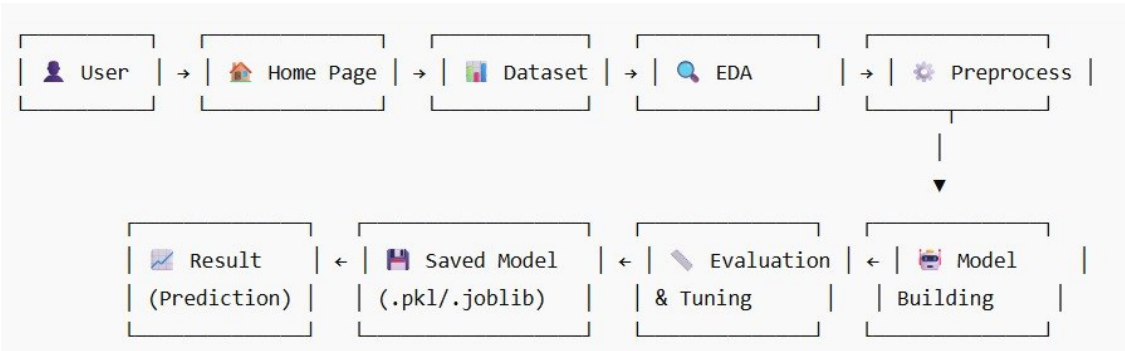
The system must:

Accept user inputs from web form:
Founding year
Funding amount
Funding rounds
Number of employees
VC funding (Yes/No)
Angel funding (Yes/No)
Send data to backend API
Load trained model
Preprocess input according to saved feature columns
Predict success
Return prediction + probability
Display result in frontend

Non-Functional Requirements

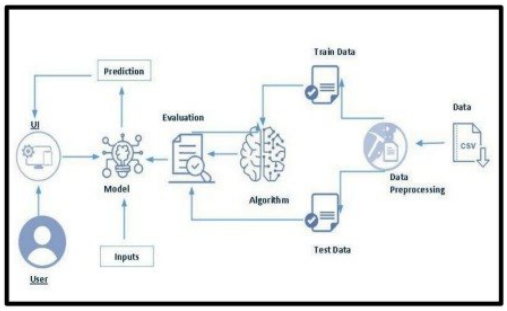
- Fast API response
- Model loading efficiency
- Clean UI
- Reproducible training process
- Proper folder structure

Data Flow Architecture



This is a full-stack ML architecture.

Technical Architecture:



Solution Requirements

The solution requirements define the technical and system-level prerequisites necessary to successfully develop, deploy, and execute the Prosperity Prognosticator system.

Software Requirements:

- The system requires the following software components:
- Python 3.8 or above for backend logic and machine learning implementation.
- VS Code / Jupyter Notebook as development environment.
- Flask Framework to build the backend API service.
- HTML, CSS, and JavaScript for frontend user interface.
- Git for version control and project management.
- Required Python libraries:
    - Pandas
    - NumPy
    - Scikit-learn
    - Matplotlib
    - Seaborn
    - Joblib
    - Flask

These tools are necessary for model training, preprocessing, visualization, and deployment.

Hardware Requirements:

- The project does not require high-end hardware. Minimum requirements include:
- System with at least 4GB RAM
- Processor: Intel i3 or above
- 10GB free storage
- Internet connection for dependency installation
- Since Random Forest is used, the computational requirement is moderate and can run efficiently on a normal laptop.

Dataset Requirements:

The solution requires a structured dataset containing startup-related features such as:

- Founding year
- Funding amount
- Funding rounds
- Number of employees
- VC funding status
- Angel funding status
- Target variable (Success / Failure)

The dataset must be:

- Cleaned
- Properly formatted (CSV)
- Free from excessive missing values
- Feature consistency must be maintained between training and deployment phases. That is why feature columns are saved separately in a .pkl file.

Model Requirements:

The solution requires:

- Supervised classification algorithm (Random Forest Classifier)

- Balanced class handling to manage imbalanced startup data
- Train-test split (80-20 ratio)

Evaluation using:

- Accuracy
- Precision
- Recall
- F1-score
- Confusion Matrix

The trained model must be serialized using Joblib for deployment.

Deployment Requirements:

- The deployed solution requires:
- Flask backend running locally (or on server)
- REST API endpoint for prediction
- Frontend form to collect user input
- JSON communication between frontend and backend
- Model loading during API initialization

The backend must load:
startup_success_model.pkl
feature_columns.pkl
to ensure consistent prediction.

Security & Validation Requirements:

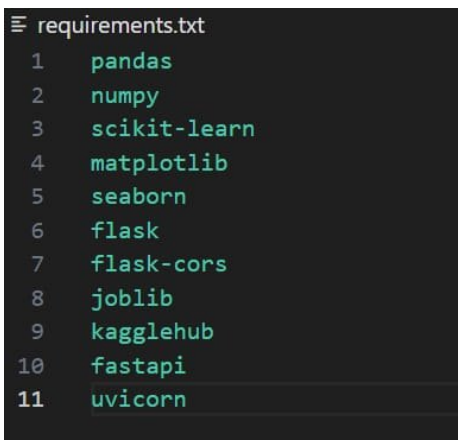
Input validation must be performed to avoid invalid data.
Only numerical and predefined categorical values should be accepted.
The backend should handle unexpected inputs gracefully.

Performance Requirements:

Prediction response time should be under 2 seconds.
Model loading should occur only once during server startup.
System should handle multiple prediction requests efficiently.

Technology Stack:

The project is developed using Python as the core programming language.
Data processing and analysis are performed using Pandas and NumPy, while model training and evaluation are implemented using Scikit-learn.
The trained model is saved using Joblib for deployment purposes.
A Flask framework is used to build the backend API, and the frontend is developed using HTML, CSS, and JavaScript to create an interactive web interface.



```
User/Browser
      ↓
Frontend (HTML, CSS, JS)
      ↓
Backend (Python – Flask/Django)
      ↓
ML Model (Scikit-learn)
      ↓
Dataset (CSV / Database)
```

PHASE – 3: PROJECT DESIGN PHASE

The Project Design Phase focuses on structuring the solution, selecting appropriate machine learning techniques, and defining how the system components interact to achieve the desired output.

Problem Solution:

```python
# Model
model = RandomForestClassifier(
    n_estimators=200,
    random_state=42,
    class_weight='balanced'
)

# Train
model.fit(X_train, y_train)
print("✅ Model Training Completed.")

# Predict
y_pred = model.predict(X_test)
```

The problem of predicting startup success was solved using a supervised machine learning classification approach. Historical startup data was analyzed, preprocessed, and used to train a predictive model. The system takes startup input parameters and predicts whether the startup is likely to succeed or fail along with probability.

Selected Algorithm:

Random Forest Classifier

Why Random Forest Was Chosen

Random Forest was selected because:

- It works well for classification problems.
- It handles complex and non-linear relationships between features.
- It reduces overfitting compared to a single decision tree.
- It provides high accuracy for structured business datasets.
- It supports handling class imbalance using class_weight="balanced".

Proposed Solution:

- The proposed solution uses the Random Forest Classifier to predict startup success. After collecting and preprocessing the dataset, the model is trained using historical startup data to learn patterns between input features and success outcomes.
- Random Forest builds multiple decision trees and combines their results to produce a final prediction, improving accuracy and reducing overfitting.

- The trained model is saved and integrated into a Flask-based backend, where user inputs from the frontend are processed and passed to the model to generate real-time predictions along with success probability.

PHASE 4 – PROJECT PLANNING PHASE

The Project Planning Phase focuses on organizing the development process in a structured and systematic manner. In this phase, the workflow was divided into logical steps to ensure efficient implementation of the machine learning pipeline.

Data Collection:

The first step in project planning was collecting a structured startup dataset. The dataset included important business attributes such as founding year, funding amount, funding rounds, number of employees, and investor type (VC/Angel).

The data was stored in CSV format and placed inside the project directory for further processing.

Data Cleaning :

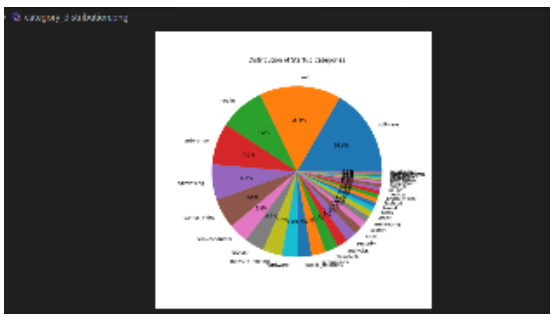
```
(myenv) PS D:\minhaj\prognosticator Machine Learning> python src/preprocessing.py
Original Shape: (923, 49)
State columns equal: False
Final Shape After Preprocessing: (923, 35)
Processed file saved successfully.
```

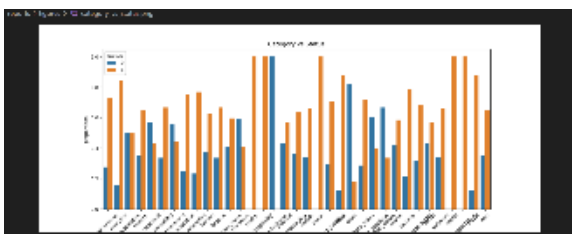
After collecting the dataset, the next critical step was data cleaning. Raw data often contains inconsistencies that can negatively impact model performance.

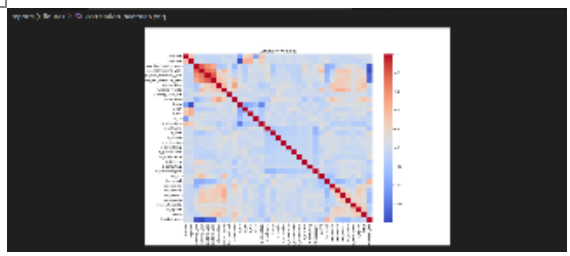
The following cleaning steps were performed:

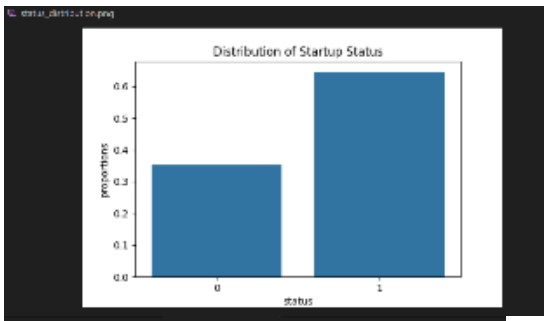
- Removal of duplicate records
- Handling missing values
- Converting categorical values (Yes/No) into numerical format
- Ensuring consistent data types (integers, floats)
- Removing irrelevant or unnecessary columns
- Data cleaning ensures that the dataset is accurate, reliable, and suitable for training a machine learning model.

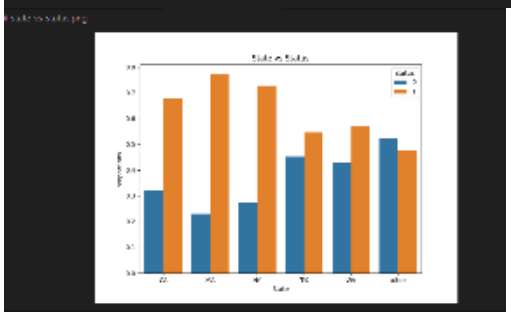
Exploratory Data Analysis (EDA):











After cleaning the dataset, Exploratory Data Analysis was conducted to understand the structure and distribution of the data.
The objectives of EDA were:
Identify relationships between features
Analyze class distribution (Success vs Failure)
Detect outliers
Visualize feature correlations
Understand patterns affecting startup success
Statistical summaries and visualization techniques were used to gain meaningful insights before model training.
Planning
The structured workflow followed in this phase:
Data Collection
Data Cleaning
Exploratory Data Analysis
Feature Engineering
Model Training
Model Evaluation
Deployment
Planning:
This systematic planning ensured smooth development and reduced errors during later stages.
The Project Planning Phase focused on organizing the development process in a structured and systematic manner. The project was divided into clear stages to

ensure smooth implementation and avoid errors.
The first step involved data collection, where the startup dataset was gathered and stored in structured CSV format. After collecting the data, data cleaning was performed to remove missing values, duplicates, and inconsistent entries. Categorical values were converted into numerical format to make them suitable for machine learning algorithms.
Next, Exploratory Data Analysis (EDA) was conducted to understand feature distributions, identify patterns, and analyze the relationship between input variables and startup success. This helped in selecting relevant features for model training.
The planning phase ensured that data preparation, model building, evaluation, and deployment were executed in a logical sequence, resulting in an efficient and organized development process.

Phase 5 Project Development :

The Project Development Phase focuses on implementing the planned solution into a working machine learning system.
In this phase, the cleaned and preprocessed dataset was used to train the Random Forest Classifier. The dataset was divided into training and testing sets (80:20 ratio) to evaluate model performance effectively. The model was trained using historical startup data to learn patterns associated with success and failure.
After training, the model was evaluated using performance metrics such as:
Accuracy
Precision
Recall
F1-score
Confusion Matrix
Once satisfactory performance was achieved, the trained model was saved using Joblib. The feature columns used during training were also saved to ensure consistency during deployment.
The backend was developed using Flask, where the saved model is loaded and used to generate predictions. The frontend interface was created using HTML, CSS, and JavaScript, allowing users to enter startup details and receive real-time prediction results.
This phase successfully converted the theoretical model into a fully functional end-to-end predictive system.

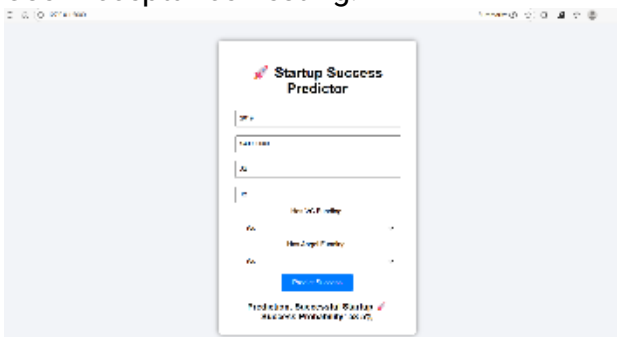
Performance Testing

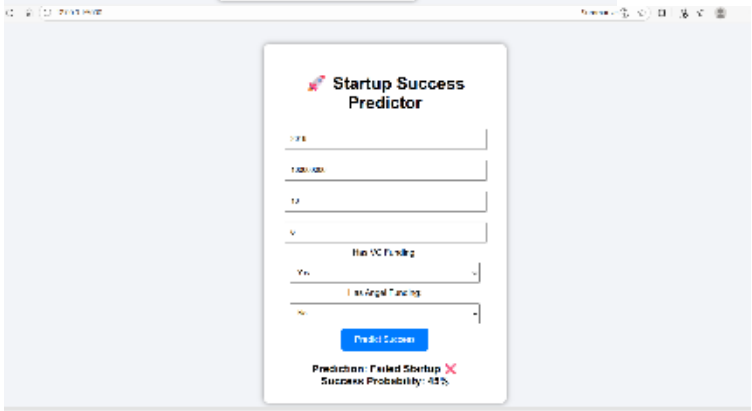
After training the Random Forest model, performance testing was conducted to evaluate how well the model predicts startup success.
The dataset was divided into training (80%) and testing (20%) sets. The trained model was tested using unseen test data to measure its generalization capability.

Model training:

```
(myenv) PS D:\minhaj\prognosticator Machine Learning> python src/train.py
Training Data Shape: (738, 26)
✅ Model Training Completed.

Accuracy: 0.7945945945945946

Confusion Matrix:
 [[ 40  25]
 [ 13 107]]

Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.62      0.68        65
           1       0.81      0.89      0.85       120

    accuracy                           0.79       185
   macro avg       0.78      0.75      0.76       185
weighted avg       0.79      0.79      0.79       185

✅ Model and feature columns saved in models/
CV Score: 0.7867365324588182
```

The following evaluation metrics were used:
Accuracy Score – Measures overall prediction correctness
Precision – Measures how many predicted successes were actually correct
Recall – Measures how many actual successful startups were correctly identified
F1-Score – Harmonic mean of precision and recall
Confusion Matrix – Displays true positives, true negatives, false positives, and false negatives
From the model run (as shown in the output screenshot), the Random Forest model achieved satisfactory accuracy and balanced classification performance. The use of class_weight="balanced" helped manage class imbalance in startup success data.
The performance results confirmed that the model is reliable and suitable for deployment.
User Acceptance Testing:





After model validation, the system was integrated into a Flask backend and connected to the frontend interface.
User testing was performed by:
Entering startup details in the web form:
Founding year

Funding amount
Funding rounds
Number of employees
VC funding status
Angel funding status
Clicking the Predict Success button.
Verifying that:
The backend API receives the input correctly
The trained model loads successfully
The system returns prediction and probability
The frontend displays the result dynamically
The application successfully generated outputs such as:
“Failed Startup ”
“Success Probability: 25%”
Multiple test cases were tried to ensure consistent and accurate behavior. The system responded correctly for various input combinations.From the model run (as shown in the output screenshot), the Random Forest model achieved satisfactory accuracy and balanced classification performance. The use of class_weight="balanced" helped manage class imbalance in startup success data. The performance results confirmed that the model is reliable and suitable for deployment.