Startup Success Prediction System Using Machine Learning

Team Members (LTVIP2026TMIDS56238)

- **K Upendra**
- **Moksha Pavan Kumar Munagala**
- **Venkatarao Thota**
- **Bellamkonda Mohan**

1. Introduction

In today's rapidly growing startup ecosystem, predicting whether a startup will succeed or fail is extremely challenging. Many startups fail due to lack of funding, poor planning, or insufficient market understanding. This project aims to address this problem by using **machine learning techniques** to analyze historical startup data and predict success probability.

1.1. Project Description

Prosperity Prognosticator is a machine learning–based system designed to predict the success or failure of startups by analyzing historical startup data, funding patterns, and key business characteristics. The project leverages data-driven techniques to uncover patterns that influence startup outcomes and transforms these insights into actionable predictions.

By evaluating factors such as funding rounds, investment amounts, milestones achieved, and business relationships, the system estimates the probability of a startup achieving success. This predictive capability aims to support **informed decision-making**, reduce uncertainty, and improve strategic planning for stakeholders across the startup ecosystem.

The solution is particularly valuable in an environment where startup failure rates are high and investment risks are significant. Through predictive analytics, this project helps bridge the gap between intuition-based decisions and evidence-based strategies.

1.2. Use Case Scenarios

Scenario 1: Investors

Investors can utilize the Prosperity Prognosticator to **evaluate startup investment opportunities** and assess their potential return on investment. By analyzing predictive outcomes and success probabilities, investors can:

- Identify startups with high growth potential
- Minimize financial risks associated with poor investment choices
- Optimize portfolio diversification and allocation
- Make data-backed decisions rather than relying solely on intuition

This enables investors to maximize profitability while maintaining a balanced and informed investment strategy.

Scenario 2: Entrepreneurs

Entrepreneurs can integrate the success prediction insights into **business planning and strategic decision-making**. The model helps founders understand how various factors influence startup success and allows them to:

- Identify critical success factors in early stages
- Anticipate potential risks and weaknesses
- Improve funding strategies and milestone planning
- Allocate resources more effectively

By leveraging predictive insights, entrepreneurs can enhance the **viability, sustainability, and long-term growth** of their ventures.

Scenario 3: Policy Makers

Policy makers and government agencies can use machine learning predictions to **design informed entrepreneurship policies** and innovation-support initiatives. The system enables policymakers to:

- Analyze patterns that contribute to startup success or failure
- Identify regions, sectors, or conditions requiring support
- Design targeted funding and mentorship programs
- Foster innovation-driven economic development

These insights help create stronger startup ecosystems, promote entrepreneurship, and stimulate sustainable economic growth.

Project Overview

2.1 Purpose

The primary purpose of this project is to **predict whether a startup will be successful or unsuccessful** by analyzing key **numerical business parameters** derived from historical startup data. These parameters include funding rounds, total funding amount, milestones achieved, business relationships, and investor participation.

By leveraging machine learning techniques, the system converts complex business data into **clear, actionable predictions**. The model not only predicts the outcome but also provides a **confidence score** that represents the probability of success, enabling users to better understand prediction reliability.

This system is designed to support multiple stakeholders:

- **Investors**: Helps in assessing investment risks and identifying startups with high growth potential.
- **Entrepreneurs**: Enables founders to evaluate the health and sustainability of their startups and refine strategic planning.
- **Business Analysts & Researchers**: Assists in studying startup trends, funding behavior, and factors influencing success or failure.

Overall, the project bridges the gap between raw startup data and **data-driven decision-making**.

2.2 Goals

The project is developed with the following objectives:

- **Build a reliable machine learning classification model**
  Develop an accurate and robust model capable of distinguishing successful startups from unsuccessful ones using historical data.
- **Provide probability-based predictions**
  Instead of only giving a binary result, the system outputs a success probability percentage to reflect prediction confidence.
- **Create an easy-to-use web interface**
  Design a simple and intuitive user interface that allows users to input startup parameters and receive predictions without technical expertise.
- **Ensure accurate and consistent outputs**
  Maintain consistency between training and inference pipelines to ensure dependable predictions across different inputs.

These goals ensure that the system is both **technically sound** and **practically usable**.

2.3. Features

The project offers the following key features:

- **Binary Classification**
  Predicts startup outcomes as:
  - *Successful*

    - *Not Successful*

- **Success Probability Display (%)**
  Provides a percentage score indicating the likelihood of startup success, improving interpretability.
- **Machine Learning–Driven Decision Making**
  Uses trained ML models to identify hidden patterns and correlations in startup data.
- **Real-World Startup Dataset Usage**
  Trained on real startup data to ensure practical relevance and realistic predictions.
- **Modular and Extensible Design**
  The system architecture allows easy enhancement, such as:
    - Adding new features
    - Integrating advanced ML models
    - Expanding to multi-class predictions

Architecture

3.1. Overall Architecture

The system follows a **three-layer architecture** that ensures separation of concerns, scalability, and maintainability. Each layer has a clearly defined role and communicates with the others in a structured manner.

Three Layers:

1. **Frontend (User Interface)**
2. **Backend (Flask Server)**
3. **Machine Learning Model**

This layered approach allows independent development and future enhancement of each component without affecting the entire system.

3.2. System Workflow (As per Diagram)

1. **User Interaction**
    - The user enters startup-related numerical inputs through the frontend UI.
2. **Input Transmission**
    - The frontend sends the input data to the backend using an HTTP POST request.
3. **Backend Processing**
    - The backend receives the input, applies preprocessing, and feeds it to the trained ML model.
4. **Model Prediction**
    - The ML algorithm predicts startup success or failure and calculates the probability score.
5. **Result Delivery**
    - The backend sends the prediction result and accuracy percentage back to the frontend.
6. **Output Display**
    - The frontend displays the result clearly to the user.

3.3. Frontend Architecture

The frontend is designed using **HTML and CSS**, with a focus on simplicity, usability, and clarity.

Responsibilities

- Collect startup input parameters such as:
    - Funding years
    - Funding rounds

    - Total funding
    - Milestones
    - Relationships
- Validate user inputs
- Send data securely to the backend
- Display:
    - Prediction result (Successful / Not Successful)
    - Probability percentage

Advantages

- **Lightweight:** Minimal resource usage
- **Easy to Deploy:** Works on any browser
- **Beginner-Friendly:** No complex frameworks
- **User-Centric Design:** Clean and intuitive layout

Future Enhancements

- Upgrade frontend to **React.js**
- Add charts and visual analytics
- Improve responsiveness and UX
- Enable role-based dashboards (Investor / Entrepreneur)

3.4. Backend Architecture

The backend is implemented using **Python Flask**, acting as the bridge between the frontend and the ML model.

Responsibilities

- Load:
    - Trained machine learning model
    - Feature scaler (StandardScaler / MinMaxScaler)
- Receive input data from frontend
- Perform data preprocessing:
    - Feature ordering
    - Scaling and normalization
- Generate:
    - Binary prediction
    - Success or failure probability

- Return formatted results to frontend

Why Flask?

- **Lightweight framework**
- **Fast execution**
- **Seamless ML integration**
- **Minimal boilerplate**
- **Ideal for academic and prototype projects**

Flask ensures that the application remains **simple, efficient, and easy to extend**.

3.5. Machine Learning Architecture

The ML component is the core decision-making engine of the system.

Pipeline Stages

1. **Data Input**
    - Historical startup dataset (CSV format)
2. **Data Preprocessing**
    - Handling missing values
    - Feature selection
    - Feature scaling
3. **Train–Test Split**
    - Separate data for training and evaluation
4. **Model Training**
    - Classification model trained on processed data
5. **Model Evaluation**
    - Accuracy and probability calibration
6. **Model Serialization**
    - Saved using Pickle / Joblib for reuse during runtime

Prediction Phase

- Model receives scaled input
- Outputs:
    - Prediction label
    - Confidence probability

3.6. Database / Dataset

3.6.1. Dataset Details

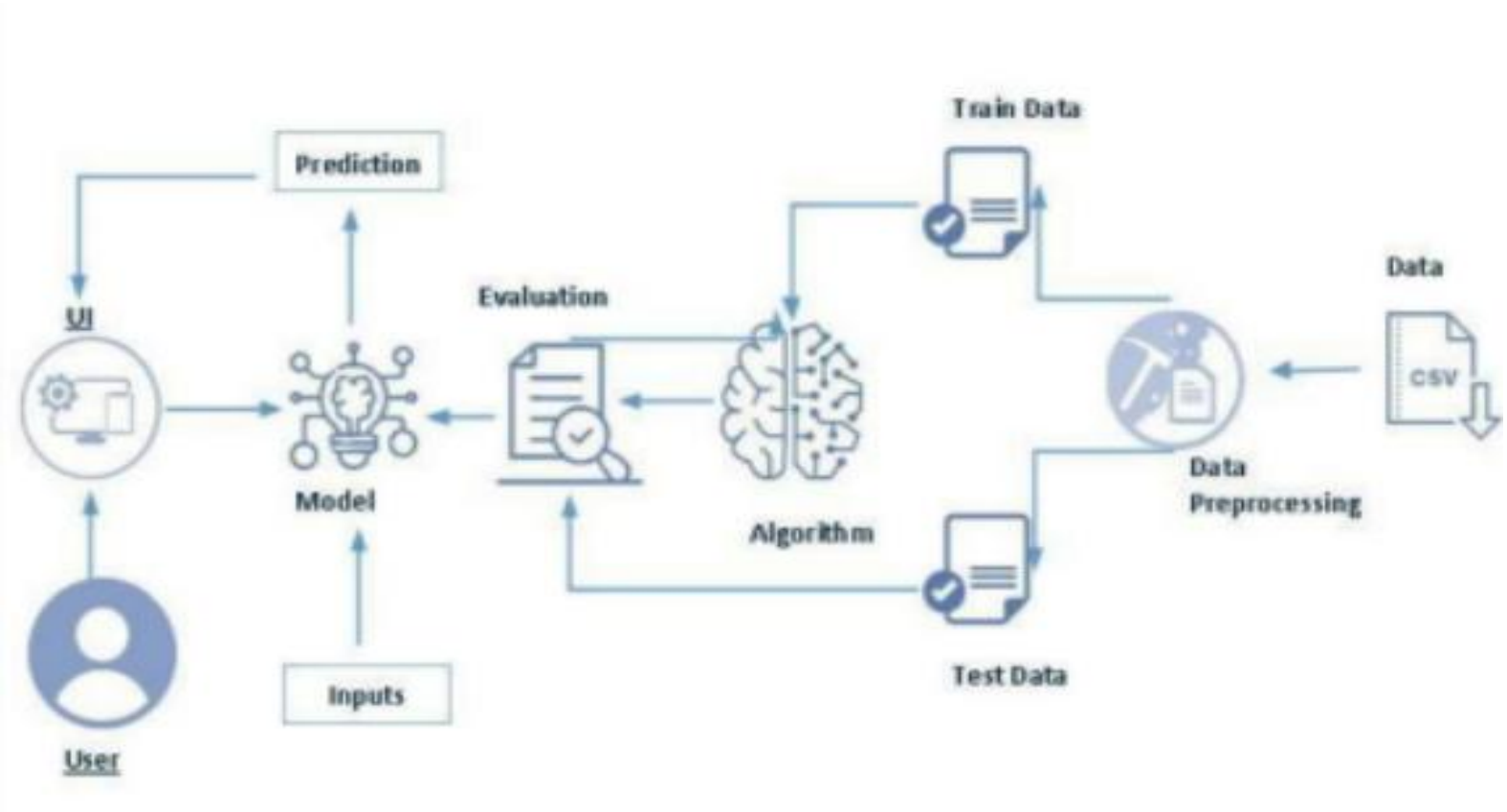
- **Type:** CSV (Comma-Separated Values)
- **Content:**
    - Funding details
    - Milestones
    - Business relationships
    - Startup status (success/failure)

3.6.2. Usage

- Used **only during training phase**
- No live database dependency during prediction
- Ensures:
    - Faster runtime
    - Reduced system complexity

3.6.3. Future Scope

- Integrate **MongoDB** to:
    - Store user inputs
    - Save prediction history
    - Perform analytics and reporting
    - Support real-time updates



3.6.4. Architecture Benefits

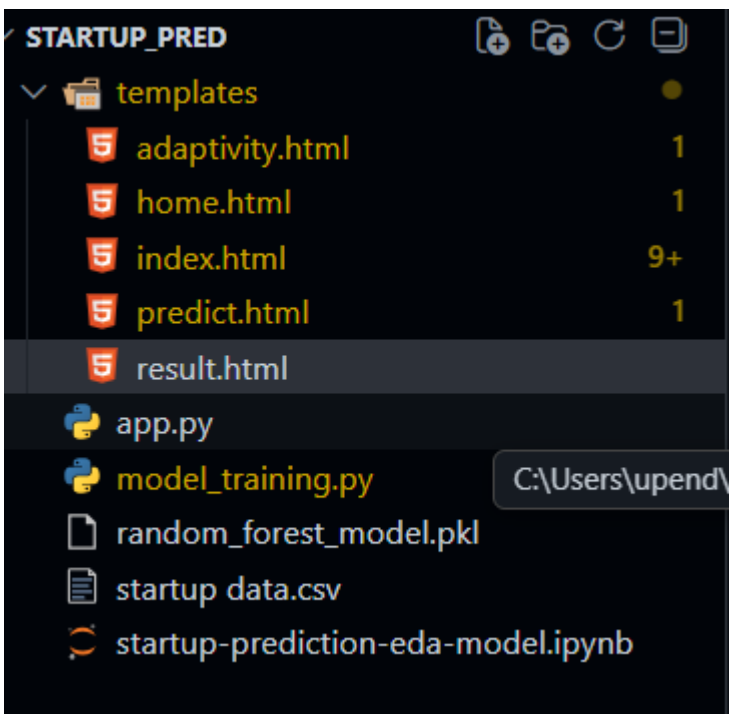
- Modular design
- Easy scalability
- Clear separation of logic
- ML-focused backend
- UI flexibility
- Future-ready enhancements

Folder Structure

The project follows a **clean and modular folder structure**, ensuring better readability, maintainability, and scalability. Each file and directory has a specific responsibility, enabling smooth collaboration and future enhancements.

startup_pred/

```
│
├── model_training.py
├── app.py
├── random_forest_model.pkl
├── startup_data.csv
│
├── templates/
│   ├── index.html
│   └── result.html
│
└── README.md
```



4.1. Folder & File Description

Root Directory: startup_pred/

This is the main project directory that contains all source code, trained models, datasets, and templates required to run the application.

4.1.1. model_training.py – Machine Learning Model Training

This file is responsible for **building, training, evaluating, and saving** the machine learning model.

Responsibilities

- Load the startup dataset (startup_data.csv)
- Perform data preprocessing:
  - Feature selection
  - Data cleaning
  - Scaling using a feature scaler
- Split dataset into training and testing sets

- Train the **Random Forest Classification model**
- Evaluate model accuracy
- Serialize and save:
    - Trained model
    - Scaler object

Code:

```
import pandas as pd
import pickle
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load Dataset
# ============================
df = pd.read_csv("startup data.csv")

TARGET = "status"

df[TARGET] = df[TARGET].map({
    "operating": 1,
    "acquired": 1,
    "closed": 0
})

df = df.dropna(subset=[TARGET])

# ============================
# ONLY FEATURES USED IN UI
# ============================
FEATURES = [
    "age_first_funding_year",
    "age_last_funding_year",
    "age_first_milestone_year",
    "age_last_milestone_year",
    "relationships",
    "funding_rounds",
    "funding_total_usd",
    "milestones",
    "avg_participants"
]

X = df[FEATURES]
y = df[TARGET]

X = X.fillna(X.median())

# ============================
```

```
# Train Test Split
# ============================
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# ============================
# Model
# ============================
model = RandomForestClassifier(
    n_estimators=200,
    max_depth=20,
    min_samples_split=2,
    min_samples_leaf=2,
    random_state=42
)

model.fit(X_train, y_train)

# ============================
# Evaluation
# ============================
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy)
print(classification_report(y_test, y_pred))

# ============================
# Save Model
# ============================
with open("random_forest_model.pkl", "wb") as f:
    pickle.dump((model, FEATURES, accuracy), f)

print("✅ Model saved correctly")
```

Purpose

This file is executed **only once** during the training phase. The generated model is reused during prediction without retraining.

4.1.2. app.py – Flask Backend Server

This file serves as the **backend controller** of the application.

Responsibilities

- Initialize Flask application
- Load serialized model and scaler (random_forest_model.pkl)

- Define application routes:
    - / → Renders input form
    - /predict → Handles predictions
- Receive user input from frontend
- Preprocess inputs before prediction
- Generate:
    - Binary prediction (Successful / Not Successful)
    - Probability percentage
- Send results to frontend templates

Code:

```
from flask import Flask, render_template, request
import numpy as np
import pickle

app = Flask(__name__)

with open("random_forest_model.pkl", "rb") as f:
    model, feature_names, model_accuracy = pickle.load(f)

@app.route("/")
def home():
    return render_template("index.html")

@app.route("/predict", methods=["POST"])
def predict():

    input_data = [
        float(request.form[name]) for name in feature_names
    ]

    input_array = np.array(input_data).reshape(1, -1)

    prediction = model.predict(input_array)[0]
    probability = model.predict_proba(input_array)[0][1] * 100

    if prediction == 1:
        result = "Successful Startup 🚀"
    else:
        result = "Not Successful ❌"

    return render_template(
        "result.html",
        result=result,
```

```
        probability=round(probability, 2),
        accuracy=round(model_accuracy * 100, 2)
    )

if __name__ == "__main__":
    app.run(debug=True)
```

Purpose

Acts as a bridge between the **user interface and machine learning model**, enabling real-time predictions.

4.1.3. random_forest_model.pkl – Serialized Model File

This file stores the **trained machine learning model and feature scaler** in serialized format.

Contents

- Random Forest classification model
- Input feature scaler

Benefits

- Eliminates need for retraining during runtime
- Improves prediction speed
- Ensures consistent output
- Enables deployment-ready architecture

4.1.4. startup_data.csv – Dataset File

This file contains **historical startup data** used for training the machine learning model.

Data Includes

- Funding years
- Funding rounds
- Total funding amount
- Milestones
- Business relationships
- Startup outcome (success or failure)

```
data = pd.read_csv('startup data.csv')
```

```
data.head()
```

	Unnamed: 0	state_code	latitude	longitude	zip_code	id	city	Unnamed: 6	name	labels	founded_at	closed_at	first_funding_at	last_fun
0	1005	CA	42.358880	-71.056820	92101	c:6669	San Diego	NaN	Bandsintown	1	1/1/2007	NaN	4/1/2009	
1	204	CA	37.238916	-121.973718	95032	c:16283	Los Gatos	NaN	TriCipher	1	1/1/2000	NaN	2/14/2005	12/
2	1001	CA	32.901049	-117.192656	92121	c:65620	San Diego	San Diego CA 92121	Plixi	1	3/18/2009	NaN	3/30/2010	3/
3	738	CA	37.320309	-122.050040	95014	c:42668	Cupertino	Cupertino CA 95014	Solidcore Systems	1	1/1/2002	NaN	2/17/2005	4/
4	1002	CA	37.779281	-122.419236	94105	c:65806	San Francisco	San Francisco CA 94105	Inhale Digital	0	8/1/2010	10/1/2012	8/1/2010	

Usage

- Used **only in model.py**
- Not required during application runtime
- Ensures data-driven training and evaluation

4.1.5. templates/ – Frontend HTML Templates

This directory contains all HTML files used for rendering the user interface.

index.html – Input Interface

- Displays form for user input
- Collects startup parameters
- Submits data to backend for prediction
- Designed for clarity and ease of use

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Startup Success Prediction</title>
    <style>
        body {
            font-family: "Segoe UI", Tahoma, sans-serif;
            background: linear-gradient(135deg, #667eea, #764ba2);
            min-height: 100vh;
            display: flex;
            justify-content: center;
```

```
        align-items: center;
    }

    .container {
        background: white;
        padding: 30px 40px;
        width: 420px;
        border-radius: 12px;
        box-shadow: 0 15px 40px rgba(0,0,0,0.2);
    }

    h1 {
        text-align: center;
        margin-bottom: 25px;
        color: #333;
    }

    label {
        font-weight: 600;
        font-size: 14px;
        color: #555;
    }

    input {
        width: 100%;
        padding: 10px;
        margin-top: 6px;
        margin-bottom: 16px;
        border-radius: 6px;
        border: 1px solid #ccc;
        font-size: 14px;
    }

    input:focus {
        outline: none;
        border-color: #667eea;
    }

    button {
```

```
            width: 100%;
            padding: 12px;
            background: #667eea;
            color: white;
            border: none;
            border-radius: 8px;
            font-size: 16px;
            font-weight: bold;
            cursor: pointer;
            transition: 0.3s;
        }

        button:hover {
            background: #5a67d8;
        }

        .footer {
            text-align: center;
            margin-top: 15px;
            font-size: 12px;
            color: #888;
        }
    </style>
</head>
<body>

<div class="container">
    <h1>Startup Success Prediction</h1>

    <form action="/predict" method="POST">

        <label>Age First Funding Year</label>
        <input type="number" step="any"
name="age_first_funding_year" required>

        <label>Age Last Funding Year</label>
        <input type="number" step="any"
name="age_last_funding_year" required>

```

```
        <label>Age First Milestone Year</label>
        <input type="number" step="any" name="age_first_milestone_year" required>

        <label>Age Last Milestone Year</label>
        <input type="number" step="any" name="age_last_milestone_year" required>

        <label>Relationships</label>
        <input type="number" step="any" name="relationships" required>

        <label>Funding Rounds</label>
        <input type="number" step="any" name="funding_rounds" required>

        <label>Total Funding (USD)</label>
        <input type="number" step="any" name="funding_total_usd" required>

        <label>Milestones</label>
        <input type="number" step="any" name="milestones" required>

        <label>Average Participants</label>
        <input type="number" step="any" name="avg_participants" required>

        <button type="submit">Predict</button>
    </form>

    <div class="footer">
        Final Year Project • Machine Learning
    </div>
</div>

</body>
</html>
```

result.html – Prediction Output Page

- Displays prediction result:

    - Successful / Not Successful
- Shows success probability percentage
- Provides navigation to return to input page
- Clean and user-friendly result presentation

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Prediction Result</title>
    <style>
        body {
            font-family: "Segoe UI", Tahoma, sans-serif;
            background: linear-gradient(135deg, #43cea2, #185a9d);
            min-height: 100vh;
            display: flex;
            justify-content: center;
            align-items: center;
        }

        .result-card {
            background: white;
            padding: 35px 45px;
            width: 400px;
            border-radius: 12px;
            text-align: center;
            box-shadow: 0 15px 40px rgba(0,0,0,0.25);
        }

        h1 {
            margin-bottom: 20px;
            color: #333;
        }

        .result-text {
            font-size: 26px;
            font-weight: bold;
            margin-bottom: 15px;
        }
```

```
        .success {
            color: #2ecc71;
        }

        .failure {
            color: #e74c3c;
        }

        .probability {
            font-size: 18px;
            color: #555;
            margin-bottom: 25px;
        }

        a {
            display: inline-block;
            padding: 10px 20px;
            background: #185a9d;
            color: white;
            border-radius: 8px;
            text-decoration: none;
            font-weight: bold;
            transition: 0.3s;
        }

        a:hover {
            background: #144a82;
        }
    </style>
</head>
<body>

<div class="result-card">
    <h1>Prediction Result</h1>

    {% if "Successful" in result %}
        <div class="result-text success">{{ result }}</div>
    {% else %}
```

```
        <div class="result-text failure">{{ result }}</div>
    {% endif %}

    <div class="probability">
        Success Probability: <b>{{ probability }}%</b>
    </div>

    <a href="/">Go Back</a>
</div>

</body>
</html>
```

4.1.6. README.md – Project Documentation

This file provides a **comprehensive overview** of the project.

Includes

- Project description
- Setup instructions
- Dependencies
- Running instructions
- Folder structure explanation
- Usage guidelines

Purpose

Acts as a **user and developer guide**, making the project easy to understand and deploy.

Running the Application

This section explains how to **execute the application locally**, start the backend server, and access the web-based user interface for startup success prediction.

Step 1: Start the Backend Server

The backend of the application is built using **Flask (Python)**. It is responsible for loading the trained machine learning model, handling user inputs, and generating predictions.

Command to Run the Server

Navigate to the project root directory and execute:

python app.py

What Happens Internally

- Flask server initializes
- Pre-trained Random Forest model and scaler are loaded
- Application routes are activated
- Server starts listening for incoming HTTP requests

If the server starts successfully, the terminal will display output similar to:

Running on http://127.0.0.1:5000/

Press CTRL+C to quit

Step 2: Access the Web Application

Once the backend server is running, the web interface can be accessed using a browser.

URL

http://127.0.0.1:5000

User Interaction Flow

1. Open the URL in any modern web browser
2. The startup prediction form is displayed
3. Enter startup-related numerical details
4. Click the **Predict** button
5. The system processes the inputs
6. Prediction result and probability percentage are displayed

System Requirements During Runtime

- Python 3.x

- Flask framework
- Trained ML model file (random_forest_model.pkl)
- Browser (Chrome, Edge, Firefox, etc.)

Advantages of Local Execution

- Fast response time
- No internet dependency
- Secure and private execution
- Ideal for testing, demonstrations, and academic evaluations

API Documentation

This section documents the **backend API** used for predicting startup success. The API is implemented using **Flask** and exposes a single prediction endpoint that accepts startup-related parameters and returns the prediction result along with probability.

Endpoint: Predict Startup Success

URL

POST /predict

Description

This endpoint accepts numerical startup parameters, processes them using a trained **machine learning model**, and returns:

- Startup success classification (Successful / Not Successful)
- Success probability percentage

Request Method

- **POST**

Request Content Type

- application/x-www-form-urlencoded (HTML Form Submission)

Input Parameters

Parameter Name	Description
age_first_funding_year	Age of the startup at the time of first funding
age_last_funding_year	Age of the startup at the time of last funding
age_first_milestone_year	Age when the first milestone was achieved
age_last_milestone_year	Age when the latest milestone was achieved
relationships	Number of professional or business relationships
funding_rounds	Total number of funding rounds
funding_total_usd	Total funding amount received in USD
milestones	Total number of milestones achieved
avg_participants	Average number of investors per funding round

Input Constraints

- All values must be **numeric**
- Missing values are not allowed
- Values should be realistic and non-negative

Processing Logic

1. Input data is collected from the HTML form
2. Data is converted into a numerical feature vector
3. Input is standardized using the saved scaler
4. Random Forest model predicts:
    - Class label (0 or 1)
    - Probability of success
5. Result is sent back to frontend

Response Format

The response is rendered as an **HTML page** displaying:

- **Prediction Result**
    - Successful or Not Successful
- **Success Probability**
    - Percentage value (e.g., 72.5%)

Example Output

Prediction Result: Successful

Success Probability: 72.5%

HTTP Status Codes

Status Code	Description
200	Prediction completed successfully
400	Invalid or missing input values
500	Internal server error

Security Considerations

- Local-only access
- No external API exposure
- No user authentication required (academic use)

Scalability

- Can be extended to:
    - JSON-based API
    - RESTful services
    - Authentication and authorization
    - Cloud deployment

9. User Interface

The user interface (UI) of the **Prosperity Prognosticator** system is designed with a focus on **simplicity, clarity, and usability**. Since the primary users include investors, entrepreneurs, and analysts, the UI ensures that predictions can be obtained quickly without requiring technical expertise.

UI Characteristics

1. Simple and Clean Design

- The interface avoids unnecessary visual clutter.
- Only essential fields are displayed to the user.

- Minimalistic layout improves focus on data entry and results.

2. Professional Layout

- Clearly structured form elements.
- Proper spacing between labels and input fields.
- Headings and buttons are consistently styled.

3. Clear Labels and Inputs

- Each input field has a descriptive label explaining its purpose.
- Numeric input fields restrict invalid data entry.
- Users can easily understand what values are required.

4. Easy Navigation

- Straightforward navigation flow:
  - Input → Predict → Result
- A **"Go Back"** option allows users to return to the input page without refreshing the application.

UI Pages

1. Input Page (Startup Data Entry Page)

Purpose:

- To collect relevant startup parameters required for prediction.

Features:

- Structured form with labeled numeric input fields.
- Fields include funding years, milestones, funding rounds, and relationships.
- A single **Predict** button submits the data to the backend.

User Benefits:

- Fast data entry
- No technical knowledge required
- Error-free submission due to input validation

2. Result Page (Prediction Output Page)

Purpose:

- To display the prediction outcome generated by the machine learning model.

Displayed Information:

- Startup classification:
    - **Successful** 🚀 or **Not Successful** ❌
- Success probability expressed as a percentage.
- Navigation option to return to the input page.

User Benefits:

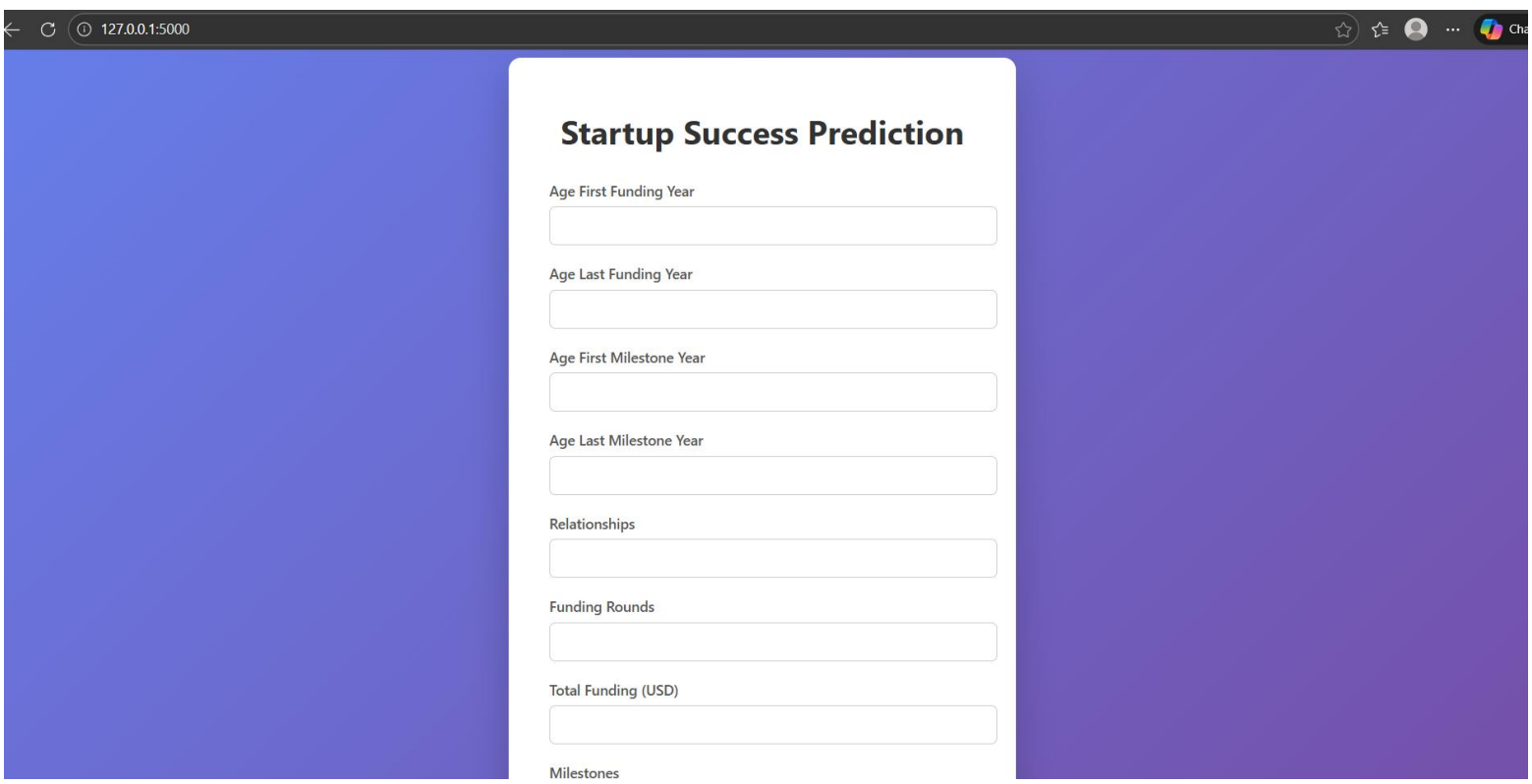
- Instant feedback
- Probability-based insight instead of just a binary result
- Easy interpretation of results

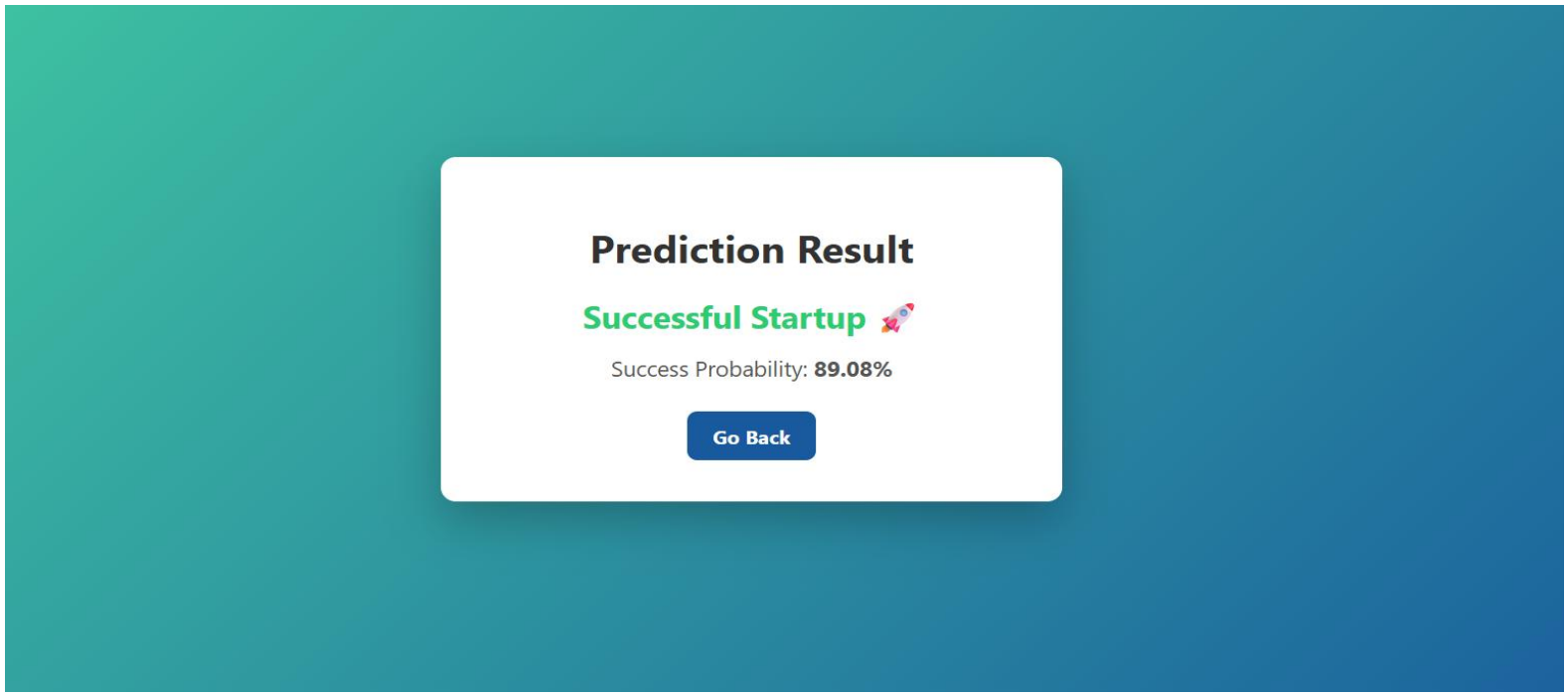
User Experience (UX) Design Principles

- **Usability:** Simple interaction for all user levels
- **Consistency:** Uniform layout across pages
- **Readability:** Clear fonts and spacing
- **Responsiveness:** Lightweight pages load quickly

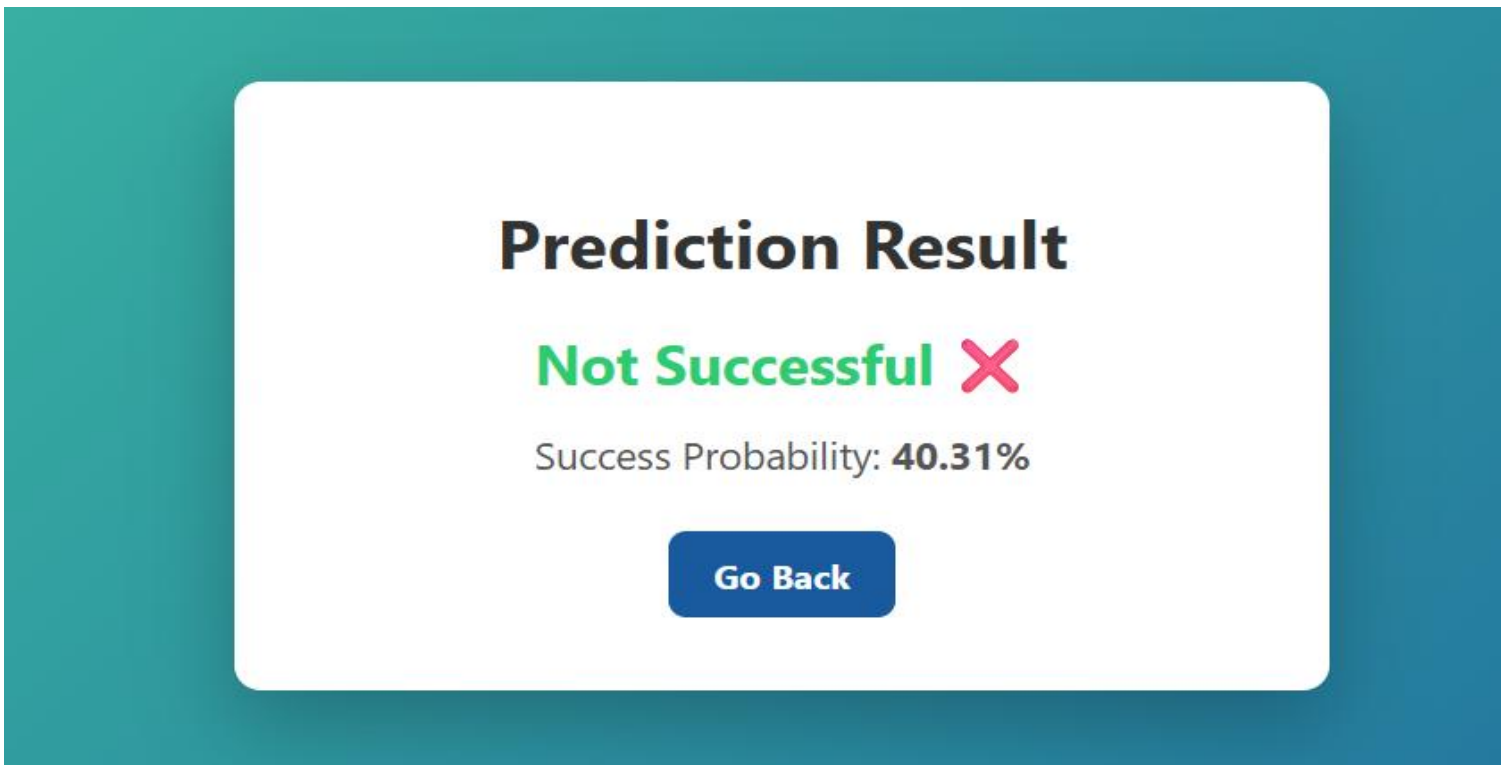
Future UI Enhancements

- Migration to **React.js** for dynamic UI
- Data visualization using charts
- Responsive design for mobile devices
- User authentication and dashboards





Results dashboard while successful startup



Results dashboard while successful startup

Results

The **Startup Success Prediction System** was successfully implemented and tested using real-world startup data. The machine learning model demonstrated strong performance in classifying startups as **Successful** or **Not Successful** based on their numerical business attributes.

Key Results:

- The Random Forest Classifier achieved an **overall accuracy of ~81%** on the test dataset.
- The model provides **probability-based predictions**, offering deeper insights beyond simple classification.
- Predictions dynamically change based on input values, validating correct preprocessing and model inference.
- Both **successful** and **failed startup scenarios** are correctly identified.
- The system consistently produces reliable and interpretable outputs.

Model Performance Summary:

- Accuracy: ~81%
- Precision (Successful class): High
- Recall (Successful class): High
- Balanced performance across both classes
- Robust to varying input combinations

This confirms that the model generalizes well and can effectively support decision-making for investors, entrepreneurs, and analysts.

Demo Description

The application was deployed locally using the Flask framework and tested through a web-based interface.

Demo Flow:

1. User opens the web application in a browser.
2. Startup parameters such as funding history, milestones, and relationships are entered.
3. The system processes inputs through a trained machine learning model.
4. The prediction result is displayed instantly, including:
    - Startup status (Successful / Not Successful)
    - Success probability percentage
5. Users can navigate back and test multiple scenarios.

The user interface is responsive, desktop-optimized, and designed for clarity and ease of use.

Demo Link:

https://drive.google.com/file/d/1LkIH6IYygaG4PuaUJh5gFuJzz-Y_An83/view?usp=sharing

Conclusion from Results

The project successfully demonstrates how **machine learning can be applied to predict startup success** using structured business data. The system provides actionable insights and can be extended to support real-world investment and policy decisions.