

Project Documentation format

1. Introduction

- **Project Title:** Prosperity Prognosticator: Machine Learning for Startup Success Prediction
- **Team Members:**
 - Team Leader : Koyya Ramesh - Project & Data Collection & Preprocessing Lead
 - Team member : Sampath Budugula - Machine Learning Engineer
 - Team member : Satya Durga Prasad Kadali – Frontend Developer
 - Team member : Yaswanth Chowdary Nallapati – Backend Developer & Testing and Documentation Lead

2. Project Overview

- **Purpose:** This project uses machine learning to predict the success of startup companies based on various features such as funding, location, industry, and team size. This goal is to help investors and entrepreneurs make more informed decisions about which startups to invest in or launch or startup will be successful or not.

- **Features:**

1. Intelligent Predictive Modelling

The system leverages advanced supervised machine learning algorithms to predict startup success probability based on historical data patterns. It transforms complex business indicators into actionable predictions.

2. Robust Data Preprocessing Pipeline

Includes comprehensive data cleaning, missing value handling, categorical encoding, feature scaling, and normalization to ensure high-quality model input and improved accuracy.

3. Performance Optimization & Hyperparameter Tuning

Applies hyperparameter tuning techniques such as Grid Search or Random Search to optimize model performance and reduce overfitting.

4. Comprehensive Evaluation Metrics

Evaluates models using advanced performance metrics including:

- Accuracy

- Precision
- Recall
- F1-Score
- Confusion Matrix
- ROC-AUC Curve

This ensures reliable and validated prediction performance.

5. Scalable & Modular Architecture

Designed with a modular structure allowing easy integration with web applications, cloud platforms, or enterprise systems for real-world deployment.

6. Risk Assessment & Decision Support

Provides data-driven insights that help investors, venture capitalists, and entrepreneurs minimize financial risk and improve strategic decision-making.

7. Real-World Business Applicability

The system can be extended to investment analysis platforms, startup incubators, fintech solutions, and venture funding ecosystems.

8. Extensibility with Future Technologies

The framework can be enhanced with:

- Real-time data integration
- Deep Learning models
- Cloud-based deployment
- AI-driven recommendation systems

3. Architecture :

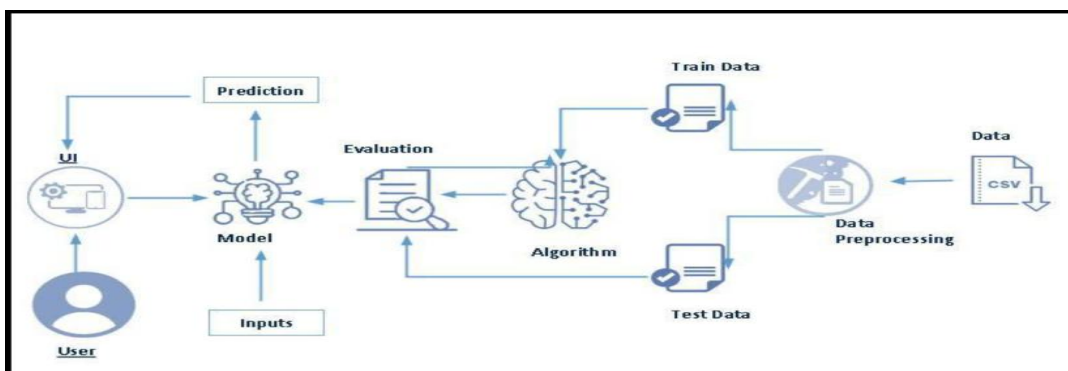


Fig.1:- Technical Architecture

Overview Of Technical Architecture :

1.Data Layer

The system begins with startup data stored in **CSV format**.
This dataset contains historical information required for predicting startup success.

2. Data Preprocessing Module

The raw dataset undergoes preprocessing to prepare it for model training. This includes:

- Cleaning inconsistent or missing values
- Encoding categorical attributes
- Feature transformation and scaling

This ensures structured and model-ready input data.

3. Training and Testing Split

The processed data is divided into:

- **Training Data** – Used to train the model
- **Testing Data** – Used to evaluate performance

This separation prevents overfitting and ensures reliable validation.

4. Machine Learning Algorithm Layer

The training dataset is passed to the selected machine learning algorithm (Random Forest Classifier as per your description).

The algorithm:

- Learns patterns from historical startup data
- Builds the predictive model

5. Model Evaluation Module

The trained model is evaluated using the testing dataset.

Performance metrics are computed to assess:

- Prediction accuracy
- Model reliability
- Generalization capability

This step validates the effectiveness of the trained model.

6. User Interface (UI) Layer

A web-based interface allows users to:

- Enter startup-related inputs
- Submit data for prediction

The UI communicates with the trained model.

7. Prediction Module

Once user input is provided:

- The model processes the input
- Generates prediction output
- Displays whether the startup is likely to be **Successful (Acquired)** or **Unsuccessful (Closed)**

Architecture Flow (As per Diagram)

CSV Data → Data Preprocessing → Train/Test Split → Algorithm → Model → Evaluation → UI → Prediction

4. Setup Instructions

• Prerequisites :

To complete this project, we require following software's, concepts and packages :

- **Anaconda Navigator**
- **Visual Studio**

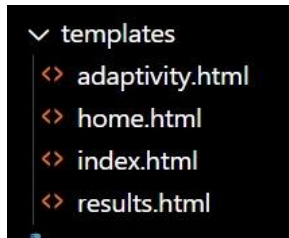
• Installation:

Installation of python packages :

- Open VS Code terminal or Command Prompt as administrator
- Type “pip install numpy” and press Enter.
- Type “pip install pandas” and press Enter.
- Type “pip install scikit-learn” and press Enter.
- Type “pip install matplotlib” and press Enter.
- Type “pip install scipy” and press Enter.
- Type “pip install joblib” and press Enter.
- Type “pip install seaborn” and press Enter.
- Type “pip install Flask” and press Enter.

5. Folder Structure

• Client:



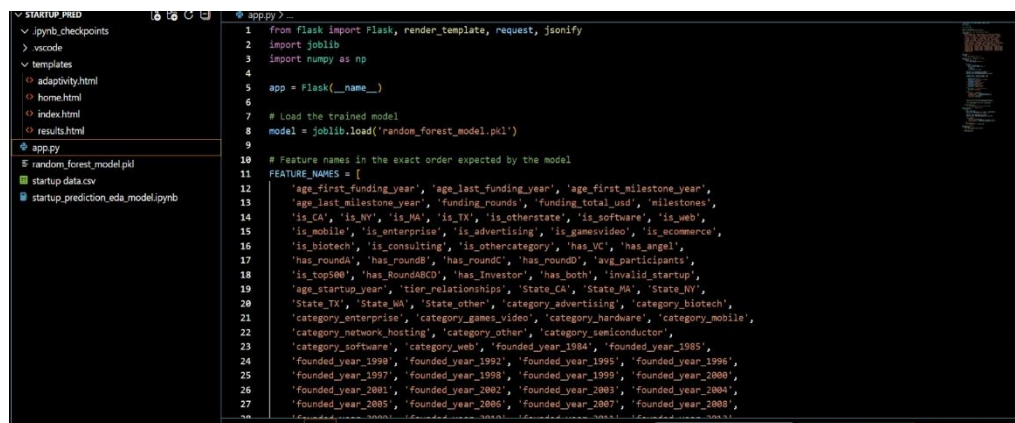
Frontend (Client-Side) Description :

The frontend of the project is developed using **HTML templates integrated with Flask**. All client-side pages are stored inside the templates folder.

- **home.html** – Landing page introducing the project.
- **index.html** – Contains the input form where users enter startup details.
- **results.html** – Displays the prediction output (Successful/Unsuccessful).
- **adaptivity.html** – Handles additional dynamic or adaptive content.

The frontend collects user input, sends it to the Flask backend for processing, and displays the prediction results in a user-friendly interface.

• Server:



Server-Side Architecture :

The backend of the project is developed using the Flask framework in Python. It handles model loading, request processing, and prediction generation.

- The trained Random Forest model (random_forest_model.pkl) is loaded using joblib.

- Feature names are defined in a fixed order to match the trained model input format.
- User input from the frontend is received using Flask's request method.
- The input data is converted into a NumPy array and passed to the model.
- The model generates a prediction (Successful/Unsuccessful).
- The result is returned to the results.html page for display.

Server Flow:

User Input → Flask Backend → Model Prediction → Send Result to Frontend

6. Running the Application

1. Importing Libraries and Reading the dataset :

```
#IMPORTING THE LIBRARIES
import numpy as np # linear algebra
import pandas as pd # data processing
pd.set_option('display.max_columns', None)
import os
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
```

HEAD DATASET :

Startup ID	state	city	founders	employees	total_funding	seed_funding	pre_seed_funding	post_seed_funding	total_funding_usd	seed_funding_usd	pre_seed_funding_usd	post_seed_funding_usd	total_funding_usd_m	seed_funding_usd_m	pre_seed_funding_usd_m	post_seed_funding_usd_m
0	CA	San Francisco	1	1	1000000	100000	900000	1000000	1000000	100000	900000	1000000	1000000	100000	900000	1000000
1	CA	San Francisco	1	1	1000000	100000	900000	1000000	1000000	100000	900000	1000000	1000000	100000	900000	1000000
2	CA	San Francisco	1	1	1000000	100000	900000	1000000	1000000	100000	900000	1000000	1000000	100000	900000	1000000
3	CA	San Francisco	1	1	1000000	100000	900000	1000000	1000000	100000	900000	1000000	1000000	100000	900000	1000000
4	CA	San Francisco	1	1	1000000	100000	900000	1000000	1000000	100000	900000	1000000	1000000	100000	900000	1000000
5	CA	San Francisco	1	1	1000000	100000	900000	1000000	1000000	100000	900000	1000000	1000000	100000	900000	1000000
6	CA	San Francisco	1	1	1000000	100000	900000	1000000	1000000	100000	900000	1000000	1000000	100000	900000	1000000
7	CA	San Francisco	1	1	1000000	100000	900000	1000000	1000000	100000	900000	1000000	1000000	100000	900000	1000000
8	CA	San Francisco	1	1	1000000	100000	900000	1000000	1000000	100000	900000	1000000	1000000	100000	900000	1000000
9	CA	San Francisco	1	1	1000000	100000	900000	1000000	1000000	100000	900000	1000000	1000000	100000	900000	1000000

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used startup .csv data. This data is downloaded from kaggle.com.

2.Univariate analysis

Univariate analysis is the process of analysing **individual features** in the dataset to understand

their distribution, frequency, and overall behaviour. In this activity, each variable is analysed

independently to gain insights into startup characteristics.

- In this project, univariate analysis is performed using **pie charts, bar plots, and distribution plots.**

State Distribution Analysis: The pie chart represents the distribution of startups across

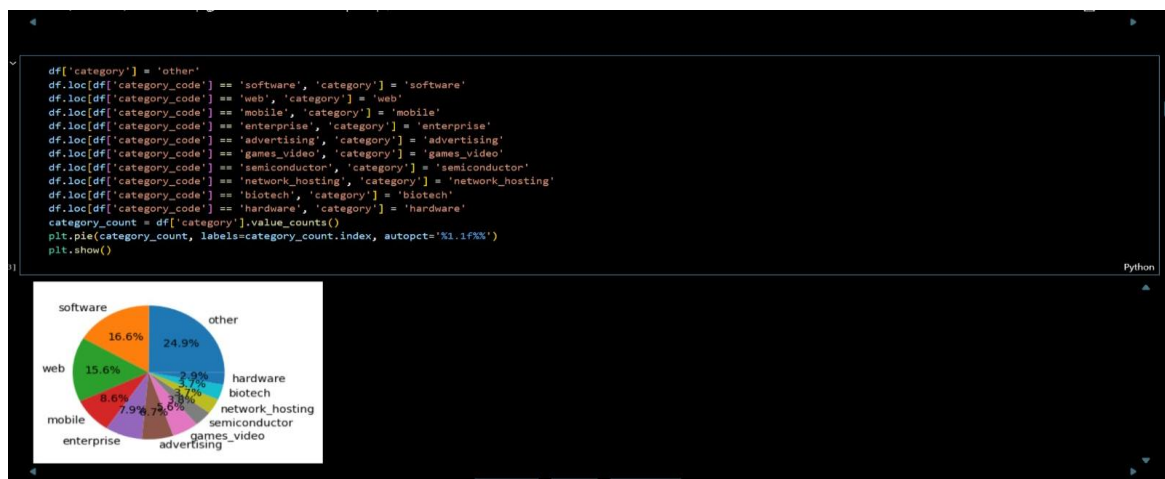
different states. It is observed that a majority of startups are located in California and New

York, indicating strong startup ecosystems in these regions.



Category Distribution Analysis:

The given code snippet performs a univariate analysis. It assigns the value 'other' to the 'category' column for all rows in the 'data' Data Frame. Then, specific rows matching certain 'category_code' values are assigned corresponding categories in the 'category' column. The 'value_counts()' method is used to count the occurrences of each unique category in the 'category' column, and a pie chart is created to visualize the distribution of categories.



Startup Status Distribution: The overall distribution of startup outcomes is analysed using a bar plot showing Acquired and Closed startups. From the bar graph, it can be observed that the number of acquired startups is higher than closed startups, indicating a considerable success rate among startups in the dataset.

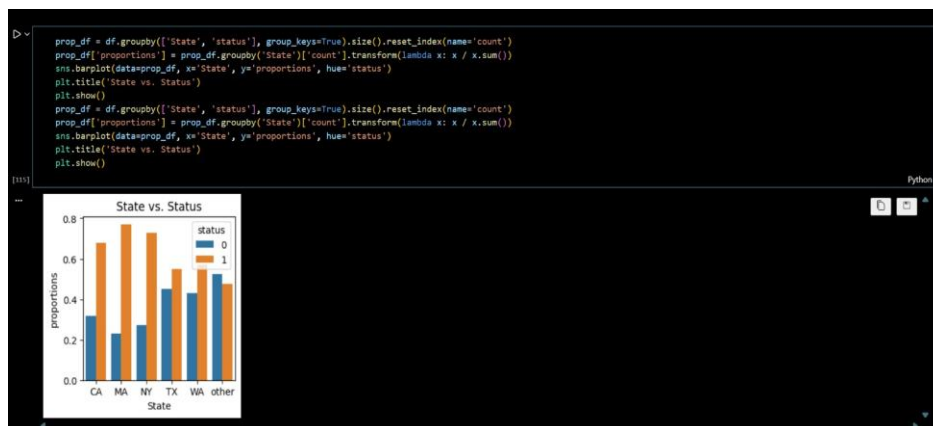


State vs Startup Status

The relationship between startup location and startup status is analyzed

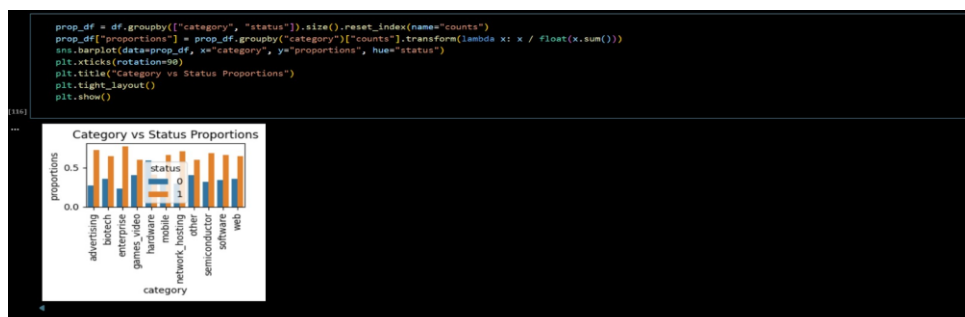
using a grouped bar

chart. The analysis indicates that startups located in California and New York have a higher proportion of acquired startups compared to other states. This shows that geographical location plays a significant role in startup success.



Category vs Startup Status

From the analysis, it is observed that software and web-based startups show a higher success rate when compared to other categories. Certain categories also show a higher closure rate, indicating higher business risk..

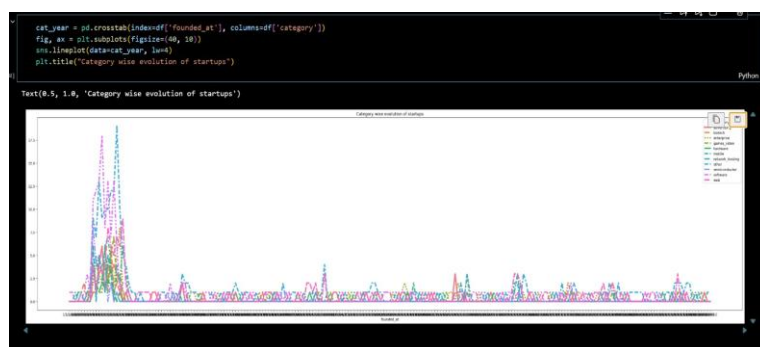


Multivariate analysis

In simple words, multivariate analysis is to find the relation between multiple features. In this project, multivariate analysis is performed to study how different startup attributes such as category, funding, milestones, and year of establishment collectively influence startup behaviour.

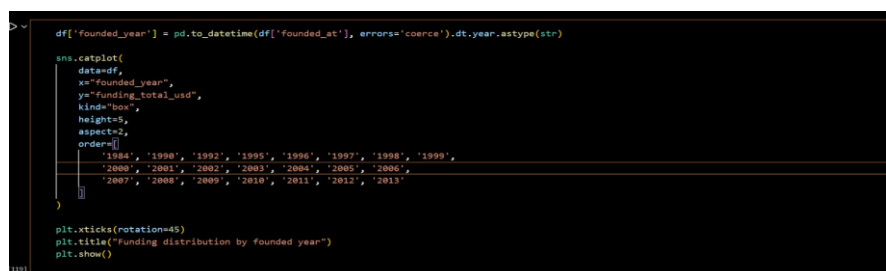
Category Evolution by Year

To understand how startup categories have evolved over time, a line plot is used to analyse the growth of different startup categories across founding years.



Funding Distribution by Founded Year

The distribution of total funding amount with respect to the founding year is analyzed using a box plot.



3.Data Pre-processing

Data preprocessing is an important step in machine learning. Raw data may contain missing values, irrelevant features, inconsistencies, or imbalanced classes. Therefore, preprocessing is performed to clean and prepare the dataset before model training.

In this project, preprocessing includes:

- Checking for null values
- Converting date features
- Feature selection
- Handling target variable
- Splitting dataset into training and testing sets

```
import numpy as np

conditions = [
    (df['relationships'] <= 9),
    (df['relationships'] > 9 & (df['relationships'] <= 18)),
    (df['relationships'] > 18 & (df['relationships'] <= 16)),
    (df['relationships'] > 16)
]

values = ['4', '3', '2', '1']

df['tier_relationships'] = np.select(conditions, values, default='0')

df['tier_relationships'] = df.tier_relationships.astype(int)

df = df.drop(['state_code'], axis=1)
df = df.drop(['id'], axis=1)
df = df.drop(['Unnamed: 0'], axis=1)
df = df.drop(['category_code'], axis=1)
df = df.drop(['object_id'], axis=1)
df = df.drop(['zip_code'], axis=1)
df = df.drop(['founded_at'], axis=1)
df = df.drop(['closed_at'], axis=1)
df = df.drop(['first_funding_at'], axis=1)
df = df.drop(['last_funding_at'], axis=1)
df = df.drop(['name'], axis=1)
df = df.drop(['Unnamed: 0'], axis=1)
df = df.drop(['latitude', 'longitude'], axis=1)
df = df.drop(['age_class_score'], axis=1)
df = df.drop(['relationships'], axis=1)
```

4.Model Building

The model building process uses a Random Forest Classifier, which is an ensemble learning algorithm that combines multiple decision trees to improve prediction accuracy and reduce overfitting through a bagging technique. The model is trained using the training dataset (X_{train} , Y_{train}) and evaluated on both training and testing data to check for overfitting or underfitting. Performance is assessed using multiple evaluation metrics including accuracy, confusion matrix, classification report, ROC-AUC, and precision-recall curve. The confusion matrix provides detailed insight into true positives, true negatives, false positives, and false negatives, while the classification report summarizes precision, recall, F1-score, and support. The ROC curve and AUC measure the model's ability to distinguish between classes, and the precision-recall curve is particularly useful for handling imbalanced datasets. Together, these evaluation techniques ensure that the model's predictive performance is reliable, balanced, and generalizable.

```
MODEL BUILDING:

from sklearn.metrics import auc, precision_recall_curve, f1_score
model = RandomForestClassifier()
model.fit(X_train, y_train)

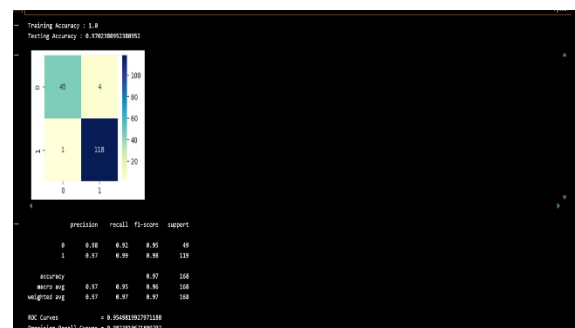
y_pred_rf = model.predict(X_test)

print("Training Accuracy :", model.score(X_train, y_train))
print("Testing Accuracy :", model.score(X_test, y_test))
model_cm = confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize=(10, 10))
sns.heatmap(model_cm, annot=True, cmap='YlGnBu', fmt='.1g')
plt.show()

model_cr = classification_report(y_test, y_pred_rf)
print(model_cr)

false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred_rf)
roc_auc = auc(false_positive_rate, true_positive_rate)
print("ROC Curve :")
print(roc_auc)

precision, recall, thresholds = precision_recall_curve(y_test, y_pred_rf)
f1 = f1_score(y_test, y_pred_rf)
Precision_Recall_f1 = auc(recall, precision)
print("Precision-Recall Curves :", Precision_Recall_f1)
```



5. Server-Side Application :

This code defines the Flask route for the /predict endpoint, which handles both GET and POST requests. When a GET request is received, it simply renders the home.html form page. When a POST request is submitted, the function collects all input features from the form based on the predefined FEATURE_NAMES list, replaces empty values with 0, converts them to float, and stores them in a list. The features are then converted into a NumPy array and reshaped to match the model's expected input format before being passed to the trained Random Forest model for prediction. The model returns both the predicted class and class probabilities, from which success and failure confidence percentages are calculated. Based on the success confidence ($\geq 80\%$, $\geq 60\%$, $\geq 40\%$, or below 40%), the code assigns a success level label and corresponding colour class for UI styling. Finally, all prediction details—including prediction result, confidence scores, success level, and colour class—are stored in a dictionary called result, which is later sent to the results page for display.

```
33 def home():
34     return render_template('index.html')
35
36 @app.route('/predict', methods=['GET', 'POST'])
37 def predict():
38     if request.method == 'GET':
39         return render_template('home.html')
40
41     try:
42         features = []
43         for feature in FEATURE_NAMES:
44             value = request.form.get(feature, 0)
45             if value == '':
46                 value = 0
47             features.append(float(value))
48
49         feature_array = np.array(features).reshape(1, -1)
50         prediction = model.predict(feature_array)[0]
51         prediction_proba = model.predict_proba(feature_array)[0]
52
53         confidence_success = prediction_proba[1] * 100
54         confidence_failure = prediction_proba[0] * 100
55
56         if confidence_success >= 80:
57             success_level = "Highly Likely to Succeed"
58             color_class = "high-success"
59         elif confidence_success >= 60:
60             success_level = "Likely to Succeed"
61             color_class = "medium-success"
62         elif confidence_success >= 40:
63             success_level = "Moderate Chance"
64             color_class = "moderate-success"
65         else:
66             success_level = "Low Success Probability"
67             color_class = "low-success"
68
69         result = {
70             'prediction': int(prediction),
71             'confidence_success': round(confidence_success, 2),
72             'confidence_failure': round(confidence_failure, 2),
73             'success_level': success_level,
74             'color_class': color_class
75         }
```

7. API Documentation :

Endpoint	Method	Description
/	GET	Loads main input page
/predict	GET	Loads home page
/predict	POST	Returns prediction via form submission
/api/predict	POST	Returns prediction in JSON format
/adaptivity	GET	Loads adaptivity page

API Documentation

1. Home Route

Endpoint: /

Method: GET

Description: Loads the main landing page.

Response: Renders index.html

2. Prediction Page (Web Form)

► GET /predict

Method: GET

Description: Displays the prediction input form.

Response: Renders home.html

► POST /predict

Method: POST

Description: Receives form data, makes prediction using Random Forest model, and displays results.

Parameters:

All features listed in FEATURE_NAMES (sent via form data).

Example parameters:

- age_first_funding_year
- funding_rounds
- funding_total_usd
- is_CA
- has_VC
- category_software
- founded_year_2010
- etc.

Response: Renders results.html with:

```
{  
  
  "prediction": 1,  
  
  "confidence_success": 78.45,  
  
  "confidence_failure": 21.55,  
  
  "success_level": "Likely to Succeed",
```

```
"color_class": "medium-success",

"input_data": { ...all feature values... }

}
```

3. API Prediction Endpoint (JSON API)

Endpoint: /api/predict

Method: POST

Description: Accepts JSON input and returns prediction in JSON format.

Request Body (JSON Example):

```
{

"age_first_funding_year": 2,

"funding_rounds": 3,

"funding_total_usd": 5000000,

"is_CA": 1,

"has_VC": 1

}
```

(Missing fields default to 0)

Response Example:

```
{

"prediction": 1,

"success_probability": 78.45,

"failure_probability": 21.55

}
```

4. Adaptivity Page

Endpoint: /adaptivity

Method: GET

Description: Loads adaptivity information page.

Response: Renders adaptivity.html

8. User Interface



Fig.2 :- Home Page Interface

The image displays the input page of the application, which is organized into three main sections: 'Company Information', 'Funding Information', and 'Milestones'. Each section contains several input fields for data entry. The 'Company Information' section includes fields for 'Founded Year', 'Age of Startup (Years)', 'State Code', and 'Category Code'. The 'Funding Information' section includes fields for 'Total Funding (USD)', 'Number of Funding Rounds', 'Age at First Funding (Years)', 'Age at Last Funding (Years)', and 'Average Participants per Round'. The 'Milestones' section includes fields for 'Number of Milestones', 'Age at First Milestone (Years)', and 'Age at Last Milestone (Years)'. Each field has a placeholder text indicating the expected format of the input.

Fig.3:- Input Page

9. Testing

1. Testing Strategy

0.Functional Testing

- Verified all routes (/ , /predict, /api/predict, /adaptivity) work correctly.
- Checked form submission and result rendering.
- Ensured correct model prediction output is displayed.

1. API Testing

- Tested /api/predict using JSON input.
- Verified correct success and failure probability responses.
- Checked error handling for invalid or missing inputs.

2. Model Validation Testing

- Confirmed that input features match FEATURE_NAMES order.
- Tested prediction consistency with sample data.

3. Error Handling Testing

- Tested empty fields (defaulted to 0).
- Checked exception handling in prediction block.

2. Tools Used

- **Manual Testing** – Browser testing for UI routes.
- **Postman / Thunder Client** – For testing /api/predict endpoint.
- **Flask Debug Mode** – To detect runtime errors.
- **NumPy & Joblib** – To validate model input/output behaviour.

10. Screenshots or Demo

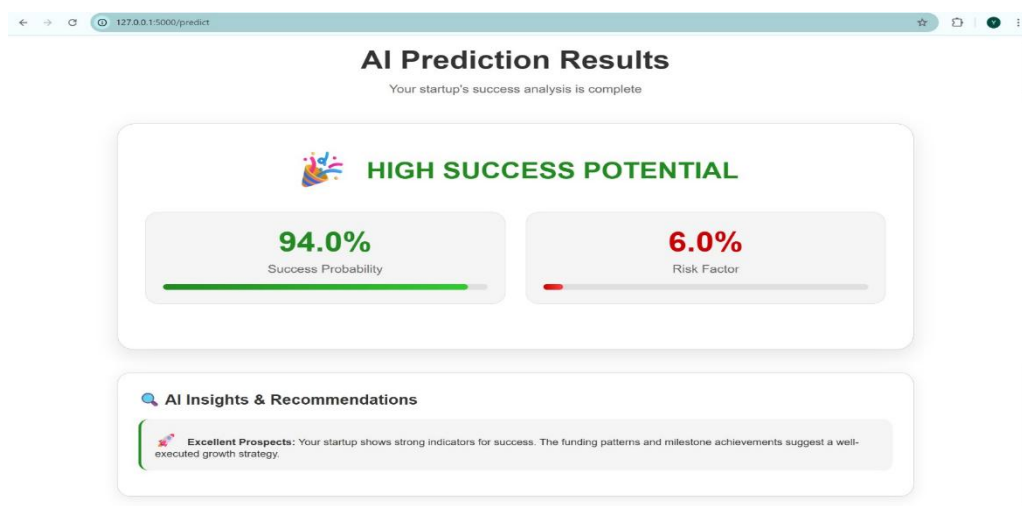


Fig.4 :- Results Page

Remaining Demo pages (Fig.2 and Fig.3) are mentioned in the above UI Interface Section.

11. Known Issues

- **Data Scarcity:** Limited info on early-stage startups.
- **Low Accuracy:** High failure rates make prediction difficult.
- **Historical Bias:** Penalizes underrepresented founders.
- **Survivorship Bias:** Ignores failed companies not in datasets.
- **Black Box:** Logic behind predictions is often unclear.
- **Intangibles:** Cannot measure founder "grit" or team chemistry.
- **Market Shifts:** Cannot predict "black swan" economic events.

12. Future Enhancements

- Expand data sources to include revenue, patents, team skills, and textual company descriptions.
- Apply NLP techniques to analyse mission statements and customer pain points.
- Use advanced success metrics such as ROI and long-term growth instead of simple IPO/acquisition labels.
- Support policy-making by identifying sectors needing funding or innovation support.
- Incorporate causal analysis to understand the real impact of funding and grants on startup success.