# Prosperity Prognosticator

# Machine Learning for Startup Success Prediction

## Project Description:

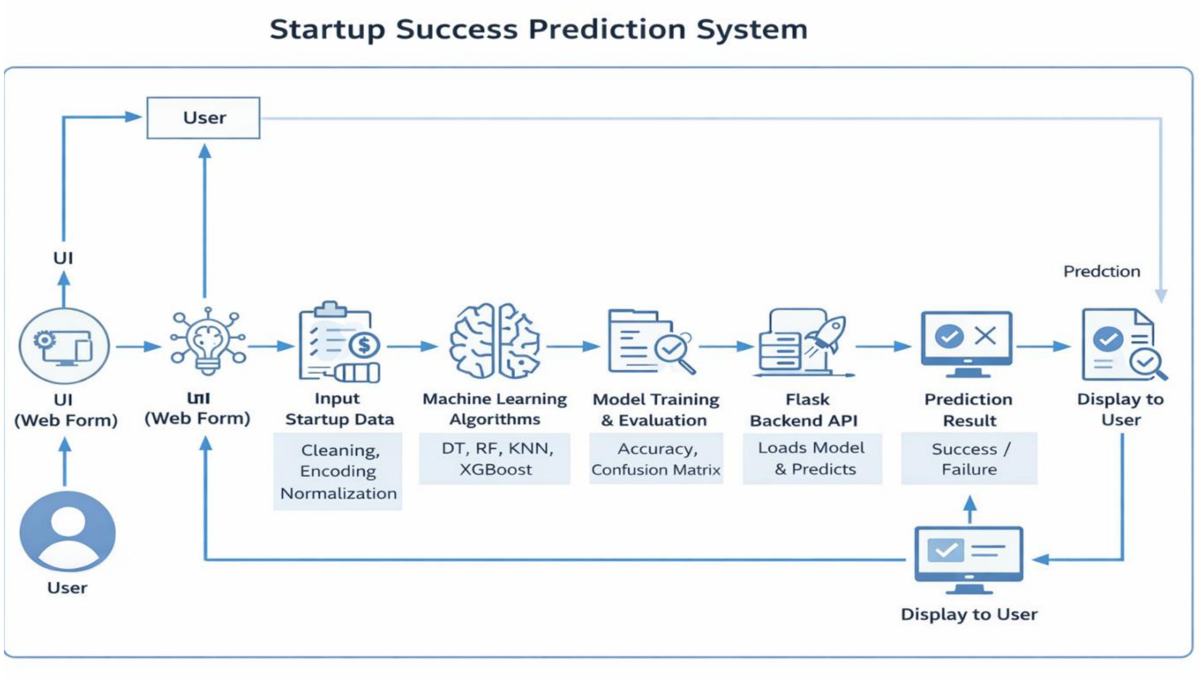
One of the most important factors influencing a country's economic growth and innovation ecosystem is the success and sustainability of startups. Startups play a crucial role in job creation, technological advancement, and economic development. However, predicting whether a startup will succeed or fail is a challenging task due to multiple influencing factors such as funding, market trends, team experience, and industry growth.

Startup success prediction is a complex problem, and many traditional evaluation methods fail to capture the hidden patterns in large datasets.By forecasting startup success or failure, investors, entrepreneurs, and policymakers can make better decisions, reduce financial risks, and allocate resources effectively.

In this project, we will use machine learning classification and regression algorithms such as Logistic Regression, Decision Tree, Random Forest, KNN, and XGBoost to predict startup success. The dataset will be preprocessed using data cleaning, feature engineering, and exploratory data analysis techniques. Finally, the model will be integrated into a Flask-based web application and deployed on a cloud platform (such as IBM Cloud) to provide real-time startup success predictions.

This system will help investors identify promising startups, assist entrepreneurs in improving their business strategies, and support policymakers in designing programs that promote innovation and economic development.

## Technical Architecture:



# Pre-requisites

To successfully complete the **Startup Success Prediction System** project, the following software tools, concepts, and Python packages are required.

## 1. Software Requirements

## Anaconda Navigator and PyCharm

Anaconda is used to manage Python environments and install required packages easily. PyCharm is used as the Integrated Development Environment (IDE) for coding and testing the project.

**Download Anaconda Navigator:**
You can refer to the following tutorial link for installation:
https://youtu.be/1ra4zH2G4o0

- **Anaconda navigator and pycharm:**
  - Refer the link below to download anaconda navigator
  - Link : https://youtu.be/mg6cMkz9Q0c?si=O_e6_4qgGF8S-lut

## Python Concepts Required

- Python Programming Basics
- Exploratory Data Analysis (EDA)
- Data Preprocessing Techniques
- Machine Learning Concepts
- Classification Algorithms
- Model Evaluation Techniques

## Required Python Packages

Open **Anaconda Prompt as Administrator** and install the following packages using the commands below:

```
pip install numpy

pip install pandas

pip install scikit-learn

pip install matplotlib

pip install scipy
```

pip install pickle-mixin

pip install seaborn

pip install Flask

Flask Web Application Development

## Prior Knowledge:

You must have prior knowledge of following topics to complete this project.

- **ML Concepts**
  - Supervised learning: https://www.javatpoint.com/supervised-machine-learning
  - Unsupervised learning: https://www.javatpoint.com/unsupervised-machine-learning
  - Regression and classification
  - Decision tree: https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
  - Random forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm
  - KNN: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
  - Xgboost: https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/
  - Evaluation metrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/
- **Flask Basics** : https://www.youtube.com/watch?v=lj4I_CvBnt0

## Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques used for machine learning.
- Gain a broad understanding about data.
- Have knowledge on pre-processing the data/transformation techniques on outlier and some visualization concepts.

## Project Flow:

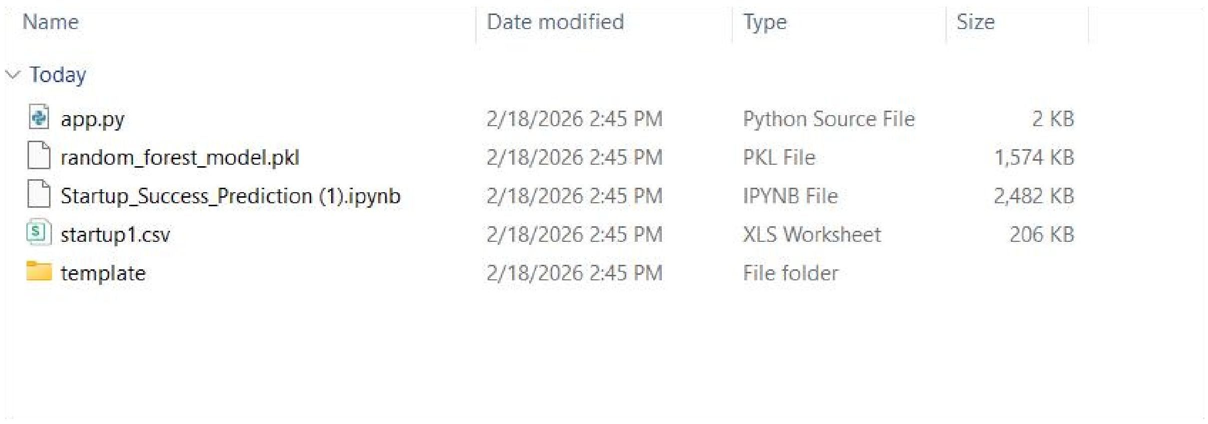
- User interacts with the UI to enter the input.
- Entered input is analyzed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

- Data collection
  - Collect the dataset or create the dataset
- Visualizing and analyzing data
  - Univariate analysis
  - Bivariate analysis
  - Multivariate analysis
  - Descriptive analysis
- Data pre-processing
  - Checking for null values
  - Handling outlier
  - Handling categorical data
  - Splitting data into train and test
- Model building
  - Import the model building libraries
  - Initializing the model
  - Training and testing the model
  - Evaluating performance of model
  - Save the model
- Application Building
  - Create an HTML file
  - Build python code

## Project Structure:

Create the Project folder which contains files as shown below



| Name | Date modified | Type | Size |
|---|---|---|---|
| Today | | | |
| app.py | 2/18/2026 2:45 PM | Python Source File | 2 KB |
| random_forest_model.pkl | 2/18/2026 2:45 PM | PKL File | 1,574 KB |
| Startup_Success_Prediction (1).ipynb | 2/18/2026 2:45 PM | IPYNB File | 2,482 KB |
| startup1.csv | 2/18/2026 2:45 PM | XLS Worksheet | 206 KB |
| template | 2/18/2026 2:45 PM | File folder | |

- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- Random_forest_model.pkl is our saved model. Further we will use this model for flask integration.
- Training folder contains model training files and training_ibm folder contains IBM deployment files.

## Milestone 1: Data Collection

ML depends heavily on data, It is most crucial aspect that makes algorithm training possible. So this section allows you to download the required dataset.

### Activity 1: Download the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com

In this project we have used startup1.csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: startup1.csv

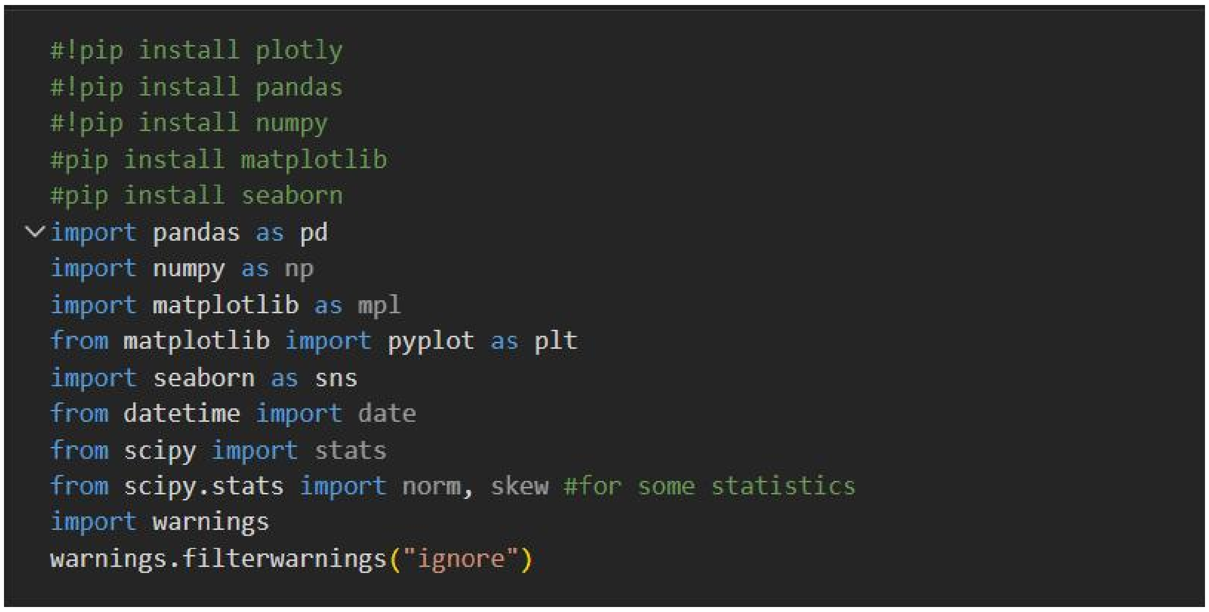
## Milestone 2: Data Information and Initial Exploration

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analysing techniques.

**Note: There is n number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.**

### Activity 1: Importing the libraries

Import the necessary libraries as shown in the image. (optional) Here we have used visualization style as fivethirtyeight.

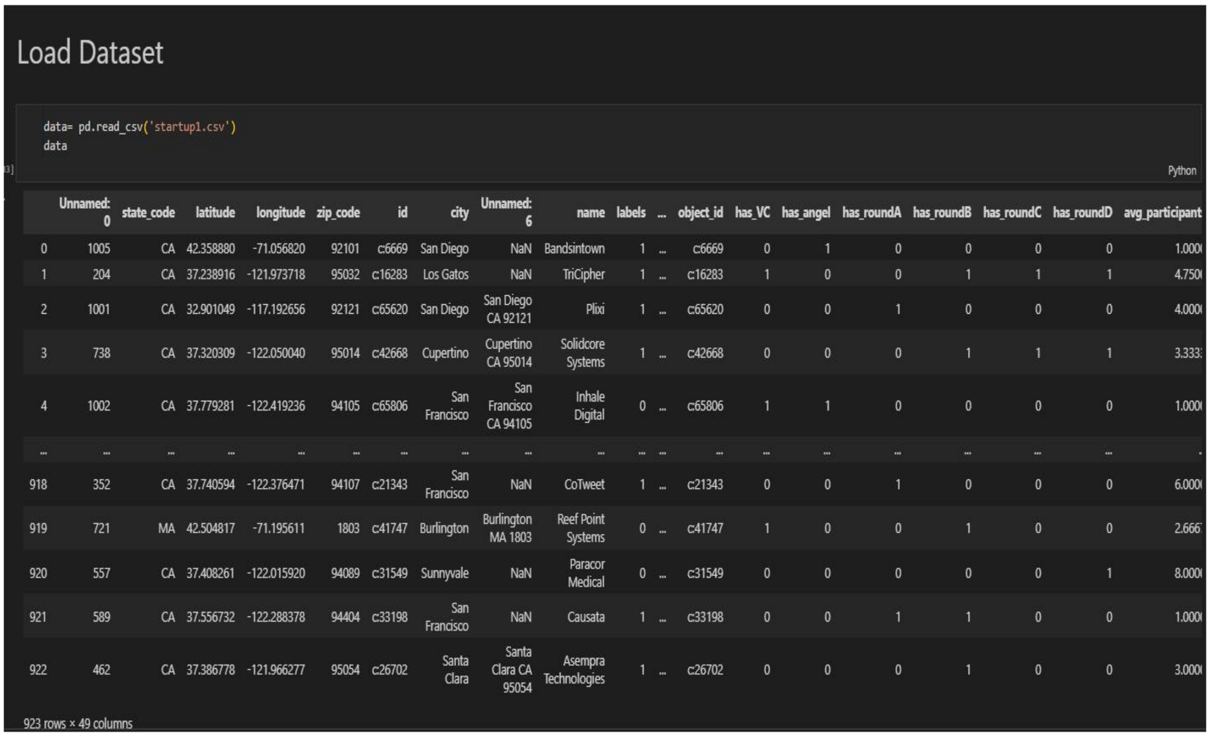


```python
#!pip install plotly
#!pip install pandas
#!pip install numpy
#pip install matplotlib
#pip install seaborn
import pandas as pd
import numpy as np
import matplotlib as mpl
from matplotlib import pyplot as plt
import seaborn as sns
from datetime import date
from scipy import stats
from scipy.stats import norm, skew #for some statistics
import warnings
warnings.filterwarnings("ignore")
```

**Activity 2: Read the Dataset:**

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of csv file.



Load Dataset

```
data= pd.read_csv('startup1.csv')
data
```

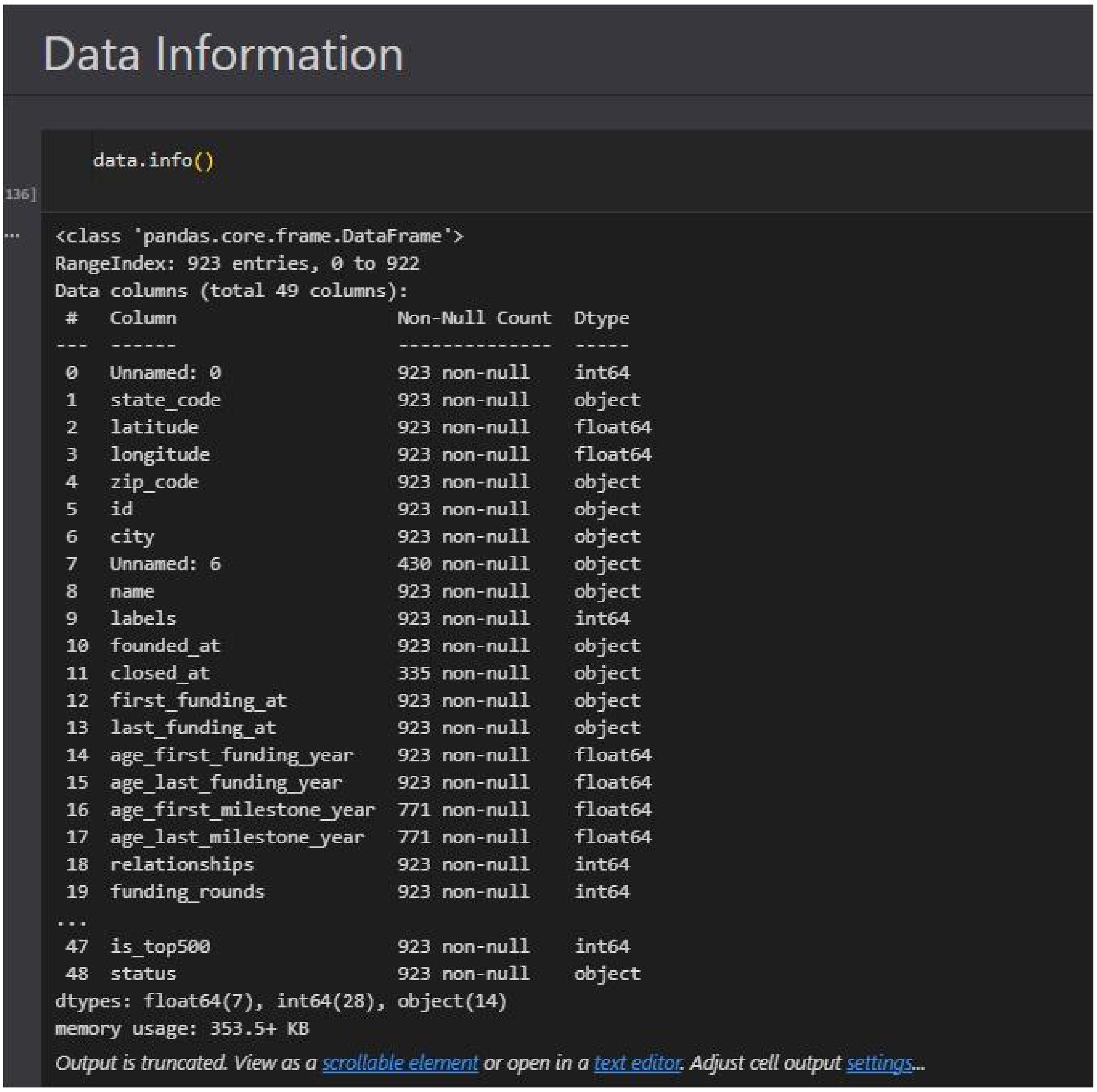
| | Unnamed: 0 | state_code | latitude | longitude | zip_code | id | city | Unnamed: 6 | name | labels | ... | object_id | has_VC | has_angel | has_roundA | has_roundB | has_roundC | has_roundD | avg_participant |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1005 | CA | 42.358880 | -71.056820 | 92101 | c:6669 | San Diego | NaN | Bandsintown | 1 | ... | c:6669 | 0 | 1 | 0 | 0 | 0 | 0 | 1.000 |
| 1 | 204 | CA | 37.238916 | -121.973718 | 95032 | c:16283 | Los Gatos | NaN | TriCipher | 1 | ... | c:16283 | 1 | 0 | 0 | 1 | 1 | 1 | 4.750 |
| 2 | 1001 | CA | 32.901049 | -117.192656 | 92121 | c:65620 | San Diego | San Diego CA 92121 | Plixi | 1 | ... | c:65620 | 0 | 0 | 1 | 0 | 0 | 0 | 4.000 |
| 3 | 738 | CA | 37.320309 | -122.050040 | 95014 | c:42668 | Cupertino | Cupertino CA 95014 | Solidcore Systems | 1 | ... | c:42668 | 0 | 0 | 0 | 1 | 1 | 1 | 3.333 |
| 4 | 1002 | CA | 37.779281 | -122.419236 | 94105 | c:65806 | San Francisco | San Francisco CA 94105 | Inhale Digital | 0 | ... | c:65806 | 1 | 1 | 0 | 0 | 0 | 0 | 1.000 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | . |
| 918 | 352 | CA | 37.740594 | -122.376471 | 94107 | c:21343 | San Francisco | NaN | CoTweet | 1 | ... | c:21343 | 0 | 0 | 1 | 0 | 0 | 0 | 6.000 |
| 919 | 721 | MA | 42.504817 | -71.195611 | 1803 | c:41747 | Burlington | Burlington MA 1803 | Reef Point Systems | 0 | ... | c:41747 | 1 | 0 | 0 | 1 | 0 | 0 | 2.666 |
| 920 | 557 | CA | 37.408261 | -122.015920 | 94089 | c:31549 | Sunnyvale | NaN | Paracor Medical | 0 | ... | c:31549 | 0 | 0 | 0 | 0 | 0 | 1 | 8.000 |
| 921 | 589 | CA | 37.556732 | -122.288378 | 94404 | c:33198 | San Francisco | NaN | Causata | 1 | ... | c:33198 | 0 | 0 | 1 | 1 | 0 | 0 | 1.000 |
| 922 | 462 | CA | 37.386778 | -121.966277 | 95054 | c:26702 | Santa Clara | Santa Clara CA 95054 | Asempra Technologies | 1 | ... | c:26702 | 0 | 0 | 0 | 1 | 0 | 0 | 3.000 |

923 rows × 49 columns

Before analysing, we need to understand what kind of data we have — the column names, data types, and null counts. This step guides all further preprocessing decisions.

**Activity 3: Data Info and Columns**

data.info() is one of the most useful pandas functions. It shows each column name, its non-null count, and its data type (int64, float64, object). This immediately reveals which columns have missing values and which are numeric vs. categorical.



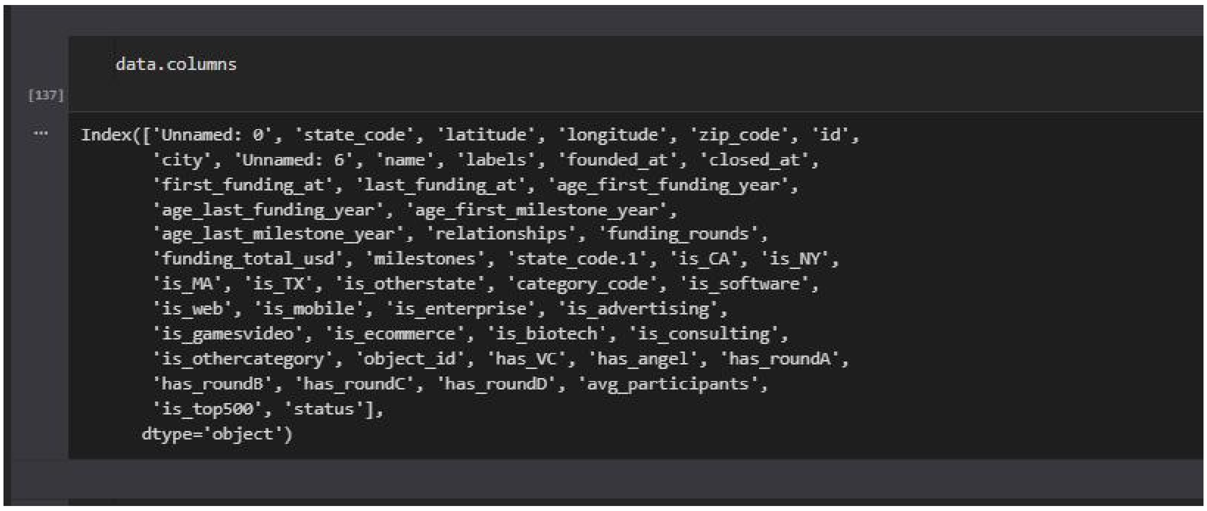
## Data Information

```
data.info()
```

[136]

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 923 entries, 0 to 922
Data columns (total 49 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Unnamed: 0                923 non-null    int64
 1   state_code                923 non-null    object
 2   latitude                  923 non-null    float64
 3   longitude                 923 non-null    float64
 4   zip_code                  923 non-null    object
 5   id                        923 non-null    object
 6   city                      923 non-null    object
 7   Unnamed: 6                430 non-null    object
 8   name                      923 non-null    object
 9   labels                    923 non-null    int64
 10  founded_at                923 non-null    object
 11  closed_at                 335 non-null    object
 12  first_funding_at          923 non-null    object
 13  last_funding_at           923 non-null    object
 14  age_first_funding_year    923 non-null    float64
 15  age_last_funding_year     923 non-null    float64
 16  age_first_milestone_year  771 non-null    float64
 17  age_last_milestone_year   771 non-null    float64
 18  relationships             923 non-null    int64
 19  funding_rounds            923 non-null    int64
...
 47  is_top500                 923 non-null    int64
 48  status                    923 non-null    object
dtypes: float64(7), int64(28), object(14)
memory usage: 353.5+ KB
```

*Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...*



```
data.columns
```

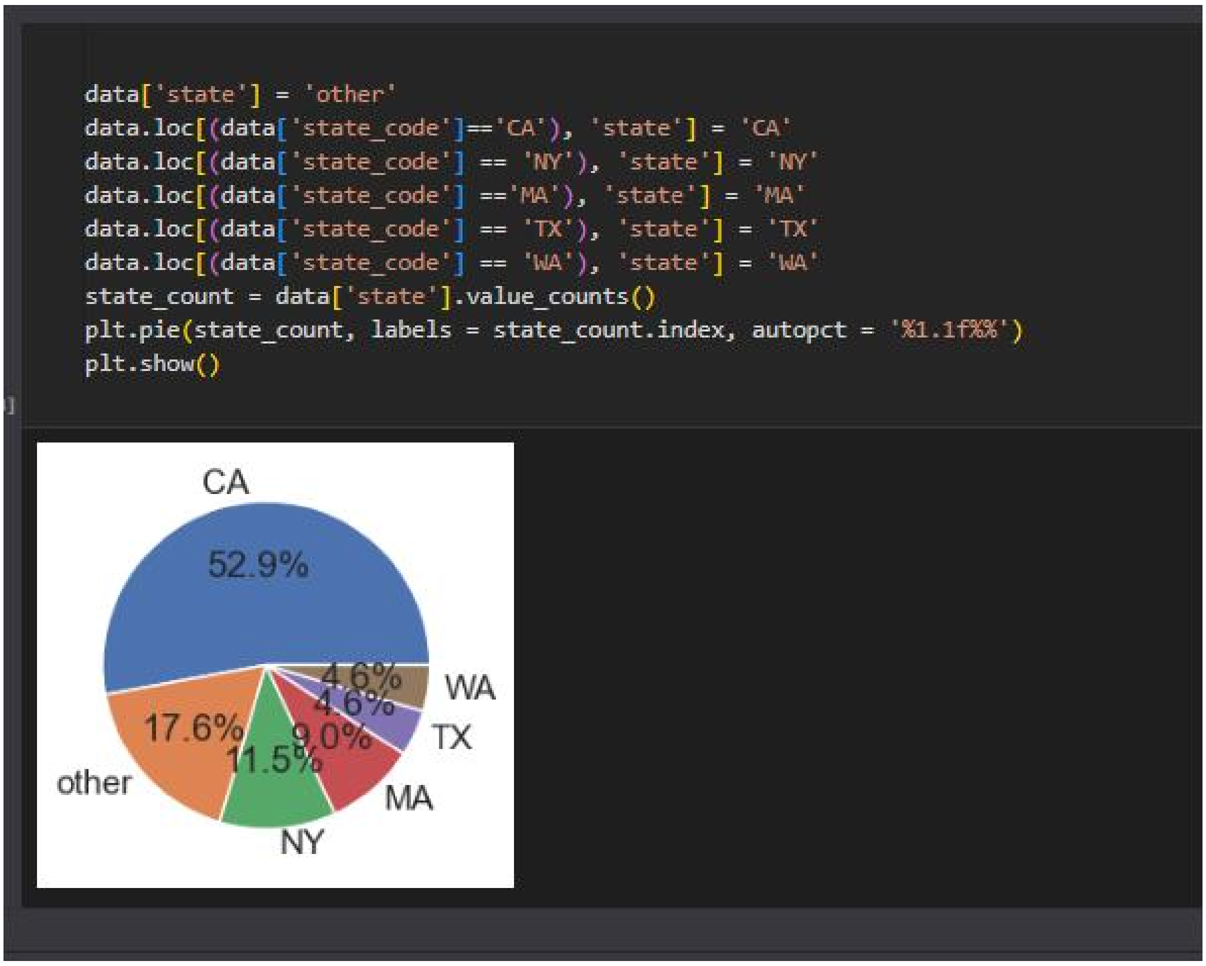
[137]

```
Index(['Unnamed: 0', 'state_code', 'latitude', 'longitude', 'zip_code', 'id',
       'city', 'Unnamed: 6', 'name', 'labels', 'founded_at', 'closed_at',
       'first_funding_at', 'last_funding_at', 'age_first_funding_year',
       'age_last_funding_year', 'age_first_milestone_year',
       'age_last_milestone_year', 'relationships', 'funding_rounds',
       'funding_total_usd', 'milestones', 'state_code.1', 'is_CA', 'is_NY',
       'is_MA', 'is_TX', 'is_otherstate', 'category_code', 'is_software',
       'is_web', 'is_mobile', 'is_enterprise', 'is_advertising',
       'is_gamesvideo', 'is_ecommerce', 'is_biotech', 'is_consulting',
       'is_othercategory', 'object_id', 'has_VC', 'has_angel', 'has_roundA',
       'has_roundB', 'has_roundC', 'has_roundD', 'avg_participants',
       'is_top500', 'status'],
      dtype='object')
```

## Activity 4: State Code Grouping and Pie Chart

The dataset contains many US state codes. To simplify analysis, we group the top 5 most frequent states (CA, NY, MA, TX, WA) and group all remaining states into a single 'other' category. We then visualize the distribution as a pie chart.

This helps us understand which geographies dominate the startup ecosystem and whether state location could be a useful feature for prediction.



```
data['state'] = 'other'
data.loc[(data['state_code']=='CA'), 'state'] = 'CA'
data.loc[(data['state_code'] == 'NY'), 'state'] = 'NY'
data.loc[(data['state_code'] =='MA'), 'state'] = 'MA'
data.loc[(data['state_code'] == 'TX'), 'state'] = 'TX'
data.loc[(data['state_code'] == 'WA'), 'state'] = 'WA'
state_count = data['state'].value_counts()
plt.pie(state_count, labels = state_count.index, autopct = '%1.1f%%')
plt.show()
```

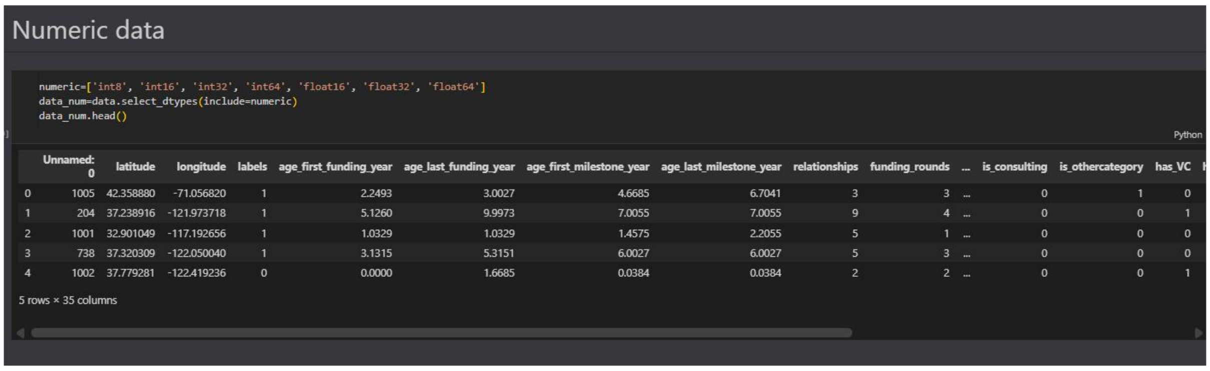
## Activity 5: Separate Numeric and Categorical Data

Pandas select_dtypes() lets us filter columns by their data type. This is important because numeric and categorical columns need very different handling — numerics can be scaled and used directly while categoricals need encoding or separate treatment.

### Numeric Data:

**Code:**

```
numeric = ['int8', 'int16', 'int32', 'int64', 'float16', 'float32',
'float64']
data_num = data.select_dtypes(include=numeric)
data_num.head()
```



Numeric data

```
numeric=['int8', 'int16', 'int32', 'int64', 'float16', 'float32', 'float64']
data_num=data.select_dtypes(include=numeric)
data_num.head()
```

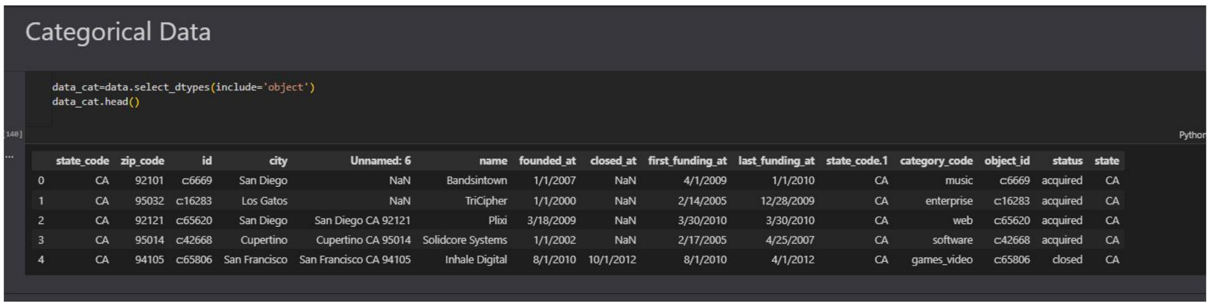
|   | Unnamed: 0 | latitude | longitude | labels | age_first_funding_year | age_last_funding_year | age_first_milestone_year | age_last_milestone_year | relationships | funding_rounds | ... | is_consulting | is_othercategory | has_VC |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1005 | 42.358880 | -71.056820 | 1 | 2.2493 | 3.0027 | 4.6685 | 6.7041 | 3 | 3 | ... | 0 | 1 | 0 |
| 1 | 204 | 37.238916 | -121.973718 | 1 | 5.1260 | 9.9973 | 7.0055 | 7.0055 | 9 | 4 | ... | 0 | 0 | 1 |
| 2 | 1001 | 32.901049 | -117.192656 | 1 | 1.0329 | 1.0329 | 1.4575 | 2.2055 | 5 | 1 | ... | 0 | 0 | 0 |
| 3 | 738 | 37.320309 | -122.050040 | 1 | 3.1315 | 5.3151 | 6.0027 | 6.0027 | 5 | 3 | ... | 0 | 0 | 0 |
| 4 | 1002 | 37.779281 | -122.419236 | 0 | 0.0000 | 1.6685 | 0.0384 | 0.0384 | 2 | 2 | ... | 0 | 0 | 1 |

5 rows × 35 columns

## Categorical Data:

**Code:**

```
data_cat = data.select_dtypes(include='object')
data_cat.head()
```



Categorical Data

```
data_cat=data.select_dtypes(include='object')
data_cat.head()
```

|   | state_code | zip_code | id | city | Unnamed: 6 | name | founded_at | closed_at | first_funding_at | last_funding_at | state_code.1 | category_code | object_id | status | state |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | CA | 92101 | c:6669 | San Diego | NaN | Bandsintown | 1/1/2007 | NaN | 4/1/2009 | 1/1/2010 | CA | music | c:6669 | acquired | CA |
| 1 | CA | 95032 | c:16283 | Los Gatos | NaN | TriCipher | 1/1/2000 | NaN | 2/14/2005 | 12/28/2009 | CA | enterprise | c:16283 | acquired | CA |
| 2 | CA | 92121 | c:65620 | San Diego | San Diego CA 92121 | Plixi | 3/18/2009 | NaN | 3/30/2010 | 3/30/2010 | CA | web | c:65620 | acquired | CA |
| 3 | CA | 95014 | c:42668 | Cupertino | Cupertino CA 95014 | Solidcore Systems | 1/1/2002 | NaN | 2/17/2005 | 4/25/2007 | CA | software | c:42668 | acquired | CA |
| 4 | CA | 94105 | c:65806 | San Francisco | San Francisco CA 94105 | Inhale Digital | 8/1/2010 | 10/1/2012 | 8/1/2010 | 4/1/2012 | CA | games_video | c:65806 | closed | CA |

## Milestone 3: Visualizing and Analysing the Data

As the dataset is loaded and the target variable is prepared, we now understand the data deeply through statistical summaries and visual exploration. There are N number of techniques for understanding data — here we use several of the most effective ones.

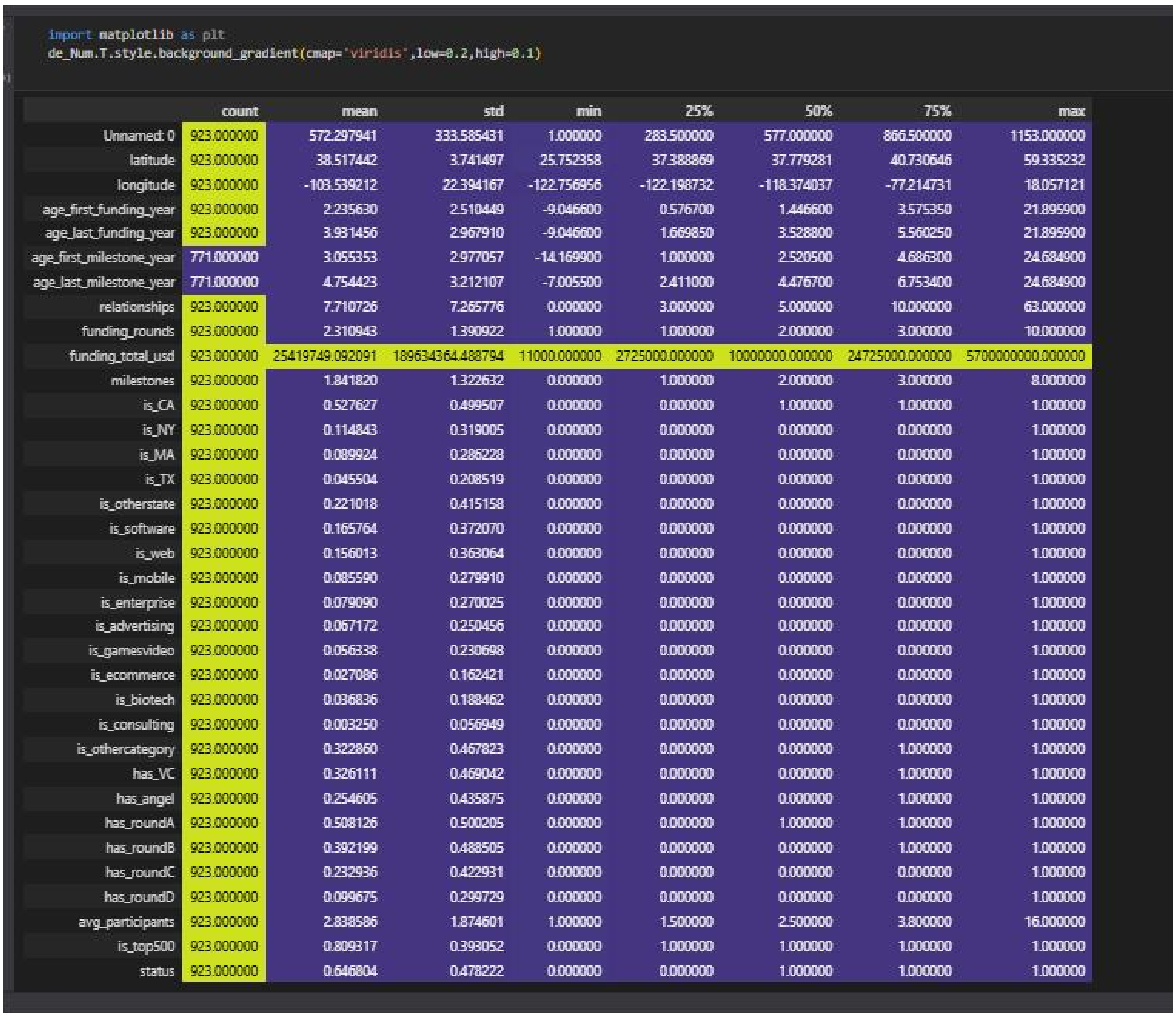
### Activity 1: Statistical Analysis

The describe() function generates a statistical summary of the DataFrame columns. It shows count, mean, standard deviation, minimum, maximum, and percentile values for numeric columns. This instantly highlights skew, spread, and outlier presence without a single graph.

## Numeric Statistics:

**Code:**

```
de_Num = data.describe(include=['float64', 'int64', 'float', 'int'])
```



```
import matplotlib as plt
de_Num.T.style.background_gradient(cmap='viridis',low=0.2,high=0.1)
```

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| Unnamed: 0 | 923.000000 | 572.297941 | 333.585431 | 1.000000 | 283.500000 | 577.000000 | 866.500000 | 1153.000000 |
| latitude | 923.000000 | 38.517442 | 3.741497 | 25.752358 | 37.388869 | 37.779281 | 40.730646 | 59.335232 |
| longitude | 923.000000 | -103.539212 | 22.394167 | -122.756956 | -122.198732 | -118.374037 | -77.214731 | 18.057121 |
| age_first_funding_year | 923.000000 | 2.235630 | 2.510449 | -9.046600 | 0.576700 | 1.446600 | 3.575350 | 21.895900 |
| age_last_funding_year | 923.000000 | 3.931456 | 2.967910 | -9.046600 | 1.669850 | 3.528800 | 5.560250 | 21.895900 |
| age_first_milestone_year | 771.000000 | 3.055353 | 2.977057 | -14.169900 | 1.000000 | 2.520500 | 4.686300 | 24.684900 |
| age_last_milestone_year | 771.000000 | 4.754423 | 3.212107 | -7.005500 | 2.411000 | 4.476700 | 6.753400 | 24.684900 |
| relationships | 923.000000 | 7.710726 | 7.265776 | 0.000000 | 3.000000 | 5.000000 | 10.000000 | 63.000000 |
| funding_rounds | 923.000000 | 2.310943 | 1.390922 | 1.000000 | 1.000000 | 2.000000 | 3.000000 | 10.000000 |
| funding_total_usd | 923.000000 | 25419749.092091 | 189634364.488794 | 11000.000000 | 2725000.000000 | 10000000.000000 | 24725000.000000 | 5700000000.000000 |
| milestones | 923.000000 | 1.841820 | 1.322632 | 0.000000 | 1.000000 | 2.000000 | 3.000000 | 8.000000 |
| is_CA | 923.000000 | 0.527627 | 0.499507 | 0.000000 | 0.000000 | 1.000000 | 1.000000 | 1.000000 |
| is_NY | 923.000000 | 0.114843 | 0.319005 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 |
| is_MA | 923.000000 | 0.089924 | 0.286228 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 |
| is_TX | 923.000000 | 0.045504 | 0.208519 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 |
| is_otherstate | 923.000000 | 0.221018 | 0.415158 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 |
| is_software | 923.000000 | 0.165764 | 0.372070 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 |
| is_web | 923.000000 | 0.156013 | 0.363064 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 |
| is_mobile | 923.000000 | 0.085590 | 0.279910 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 |
| is_enterprise | 923.000000 | 0.079090 | 0.270025 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 |
| is_advertising | 923.000000 | 0.067172 | 0.250456 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 |
| is_gamesvideo | 923.000000 | 0.056338 | 0.230698 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 |
| is_ecommerce | 923.000000 | 0.027086 | 0.162421 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 |
| is_biotech | 923.000000 | 0.036836 | 0.188462 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 |
| is_consulting | 923.000000 | 0.003250 | 0.056949 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 |
| is_othercategory | 923.000000 | 0.322860 | 0.467823 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 1.000000 |
| has_VC | 923.000000 | 0.326111 | 0.469042 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 1.000000 |
| has_angel | 923.000000 | 0.254605 | 0.435875 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 1.000000 |
| has_roundA | 923.000000 | 0.508126 | 0.500205 | 0.000000 | 0.000000 | 1.000000 | 1.000000 | 1.000000 |
| has_roundB | 923.000000 | 0.392199 | 0.488505 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 1.000000 |
| has_roundC | 923.000000 | 0.232936 | 0.422931 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 |
| has_roundD | 923.000000 | 0.099675 | 0.299729 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 |
| avg_participants | 923.000000 | 2.838586 | 1.874601 | 1.000000 | 1.500000 | 2.500000 | 3.800000 | 16.000000 |
| is_top500 | 923.000000 | 0.809317 | 0.393052 | 0.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 |
| status | 923.000000 | 0.646804 | 0.478222 | 0.000000 | 0.000000 | 1.000000 | 1.000000 | 1.000000 |

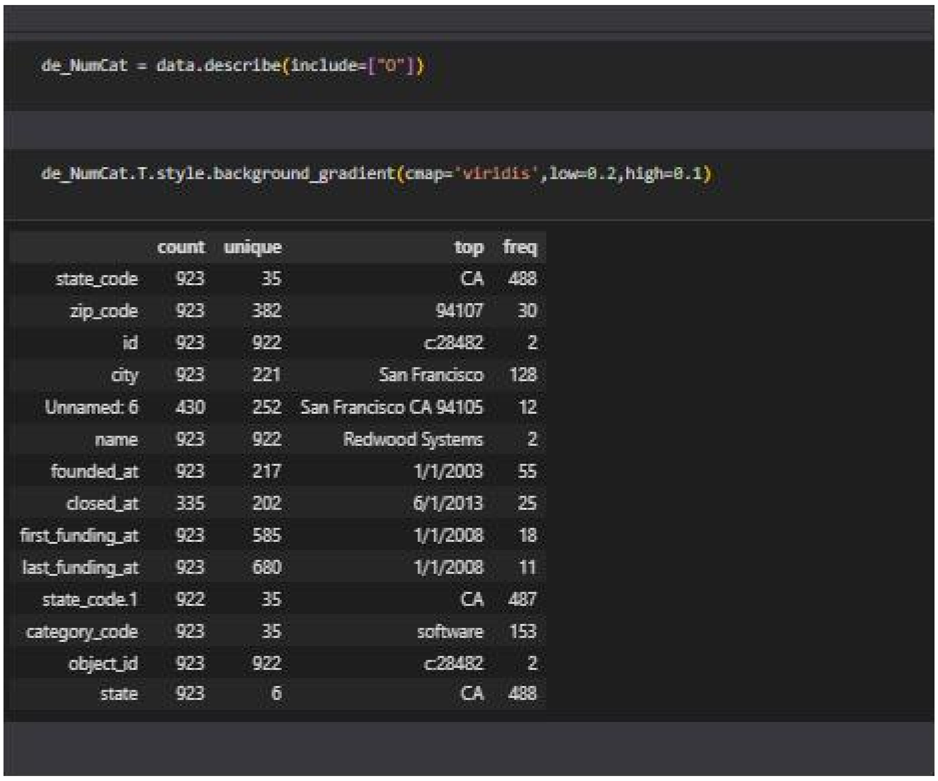
## Categorical Statistics:

**Code:**

```
de_NumCat = data.describe(include=["O"])
```

**Code (Styled Output):**

```
de_NumCat.T.style.background_gradient(cmap='viridis', low=0.2, high=0.1)
```



```
de_NumCat = data.describe(include=["O"])
```

```
de_NumCat.T.style.background_gradient(cmap='viridis',low=0.2,high=0.1)
```

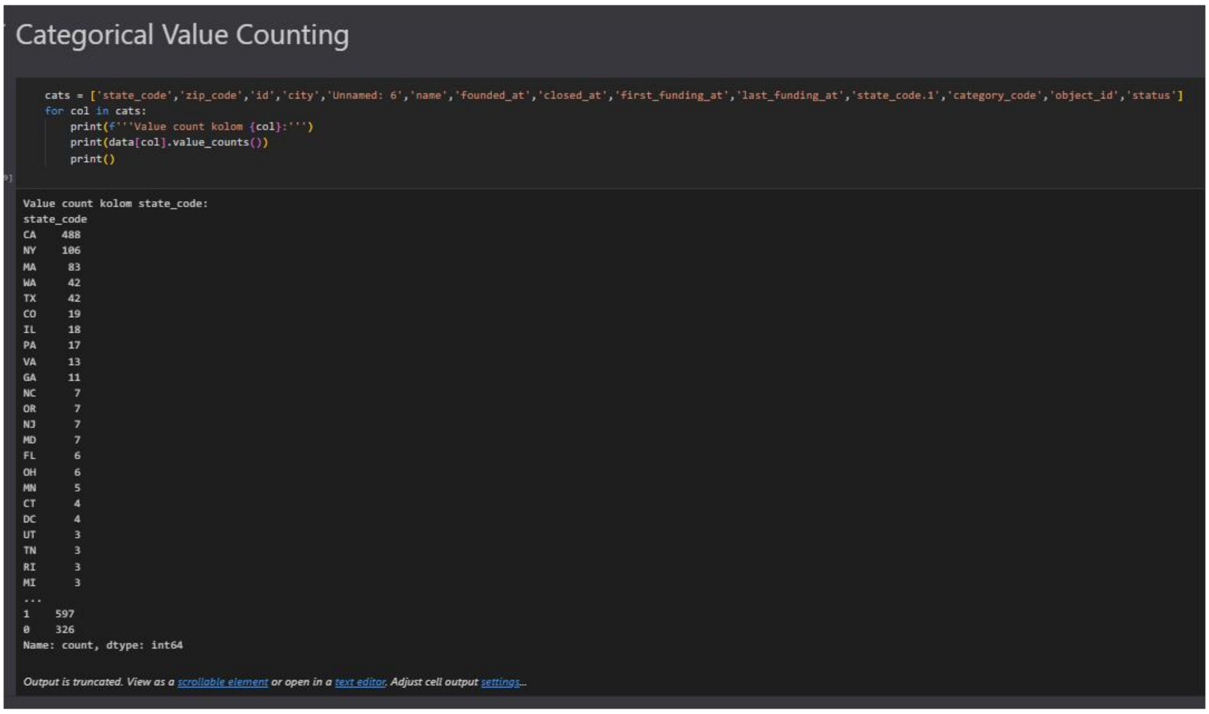
|  | count | unique | top | freq |
|---|---|---|---|---|
| state_code | 923 | 35 | CA | 488 |
| zip_code | 923 | 382 | 94107 | 30 |
| id | 923 | 922 | c:28482 | 2 |
| city | 923 | 221 | San Francisco | 128 |
| Unnamed: 6 | 430 | 252 | San Francisco CA 94105 | 12 |
| name | 923 | 922 | Redwood Systems | 2 |
| founded_at | 923 | 217 | 1/1/2003 | 55 |
| closed_at | 335 | 202 | 6/1/2013 | 25 |
| first_funding_at | 923 | 585 | 1/1/2008 | 18 |
| last_funding_at | 923 | 680 | 1/1/2008 | 11 |
| state_code.1 | 922 | 35 | CA | 487 |
| category_code | 923 | 35 | software | 153 |
| object_id | 923 | 922 | c:28482 | 2 |
| state | 923 | 6 | CA | 488 |

### Activity 2: Categorical Value Counting

We loop through all categorical columns and print the value_counts() for each. This shows how many startups fall into each unique category — for example, how many are in 'software' vs 'biotech', or which city has the most startups.

**Code:**

```
cats = ['state_code','zip_code','id','city','Unnamed: 6','name',
        'founded_at','closed_at','first_funding_at','last_funding_at',
        'state_code.1','category_code','object_id','status']


for col in cats:
    print(f'Value count for column {col}:')
    print(data[col].value_counts())
    print()
```


Categorical Value Counting

```
cats = ['state_code','zip_code','id','city','Unnamed: 6','name','founded_at','closed_at','first_funding_at','last_funding_at','state_code.1','category_code','object_id','status']
for col in cats:
    print(f'''Value count kolom {col}:''')
    print(data[col].value_counts())
    print()
```

```
Value count kolom state_code:
state_code
CA    488
NY    106
MA     83
WA     42
TX     42
CO     19
IL     18
PA     17
VA     13
GA     11
NC      7
OR      7
NJ      7
MD      7
FL      6
OH      6
MN      5
CT      4
DC      4
UT      3
TN      3
RI      3
MI      3
...
1    597
0    326
Name: count, dtype: int64
```

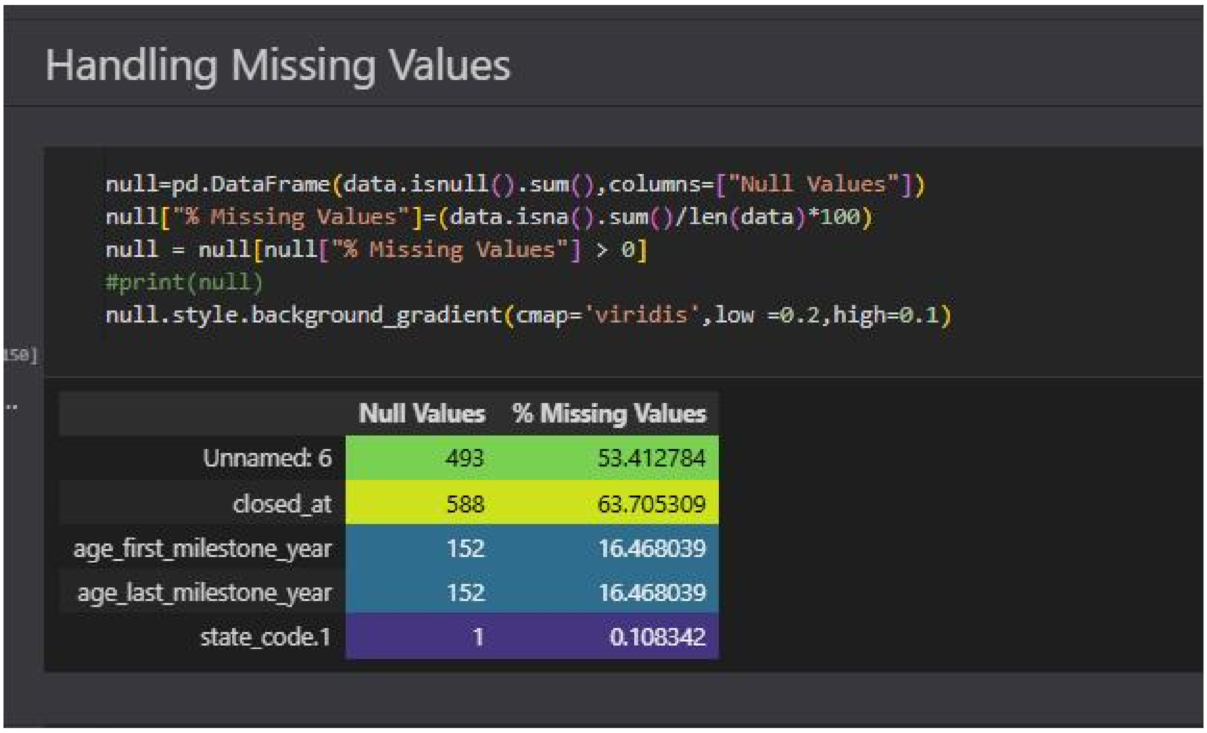
*Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...*

## Activity 3: Handling Missing Values

Missing values corrupt model training. Before filling them, we first visualise which columns are affected and what percentage of their data is missing. We use isnull() combined with a styled background gradient to make it easy to spot the worst-affected columns.

**Code:**

```
null = pd.DataFrame(data.isnull().sum(), columns=["Null Values"])
null["% Missing Values"] = (data.isna().sum() / len(data) * 100)
null = null[null["% Missing Values"] > 0]
# print(null)
null.style.background_gradient(cmap='viridis', low=0.2, high=0.1)
```



Handling Missing Values

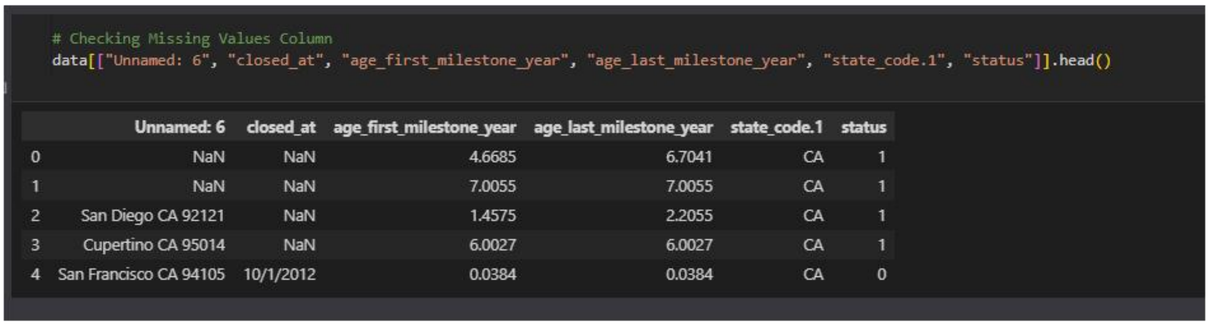
```
null=pd.DataFrame(data.isnull().sum(),columns=["Null Values"])
null["% Missing Values"]=(data.isna().sum()/len(data)*100)
null = null[null["% Missing Values"] > 0]
#print(null)
null.style.background_gradient(cmap='viridis',low =0.2,high=0.1)
```

```
                        Null Values  % Missing Values
Unnamed: 6                      493         53.412784
closed_at                       588         63.705309
age_first_milestone_year        152         16.468039
age_last_milestone_year         152         16.468039
state_code.1                      1          0.108342
```

Now we inspect those columns directly to understand the pattern of missing data:

**Code:**

```
# Preview the columns with missing values
data[["Unnamed: 6", "closed_at", "age_first_milestone_year",
      "age_last_milestone_year", "state_code.1", "status"]].head()
```



```
# Checking Missing Values Column
data[["Unnamed: 6", "closed_at", "age_first_milestone_year", "age_last_milestone_year", "state_code.1", "status"]].head()
```

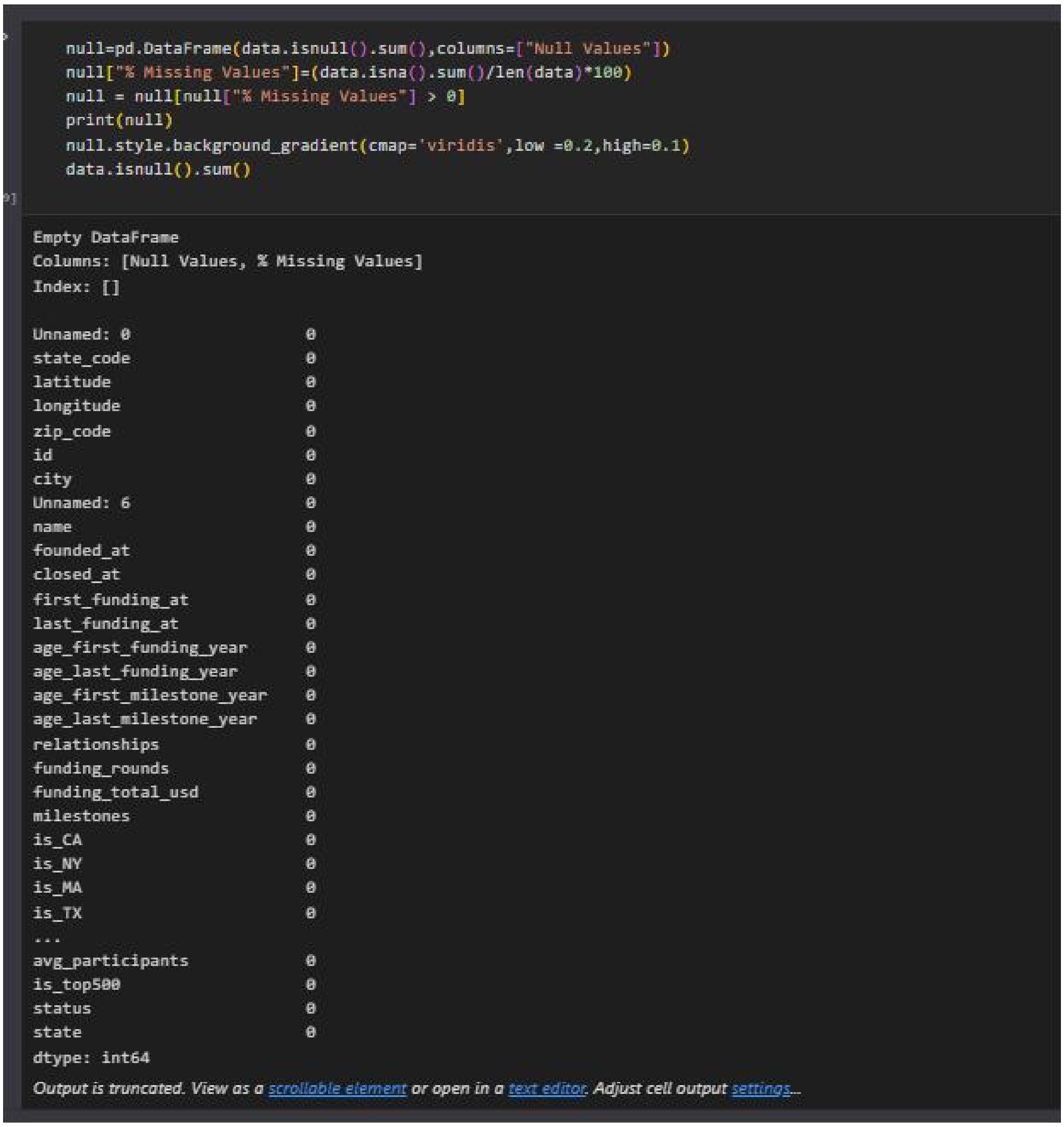
```
            Unnamed: 6   closed_at  age_first_milestone_year  age_last_milestone_year  state_code.1  status
0                  NaN         NaN                    4.6685                   6.7041            CA       1
1                  NaN         NaN                    7.0055                   7.0055            CA       1
2   San Diego CA 92121         NaN                    1.4575                   2.2055            CA       1
3  Cupertino CA 95014          NaN                    6.0027                   6.0027            CA       1
4  San Francisco CA 94105  10/1/2012                  0.0384                   0.0384            CA       0
```

## Verify: All Nulls Resolved

**Code:**

```
null = pd.DataFrame(data.isnull().sum(), columns=["Null Values"])
null["% Missing Values"] = (data.isna().sum() / len(data) * 100)
null = null[null["% Missing Values"] > 0]
print(null)
null.style.background_gradient(cmap='viridis', low=0.2, high=0.1)
```

```
data.isnull().sum()
```



```
null=pd.DataFrame(data.isnull().sum(),columns=["Null Values"])
null["% Missing Values"]=(data.isna().sum()/len(data)*100)
null = null[null["% Missing Values"] > 0]
print(null)
null.style.background_gradient(cmap='viridis',low =0.2,high=0.1)
data.isnull().sum()
```

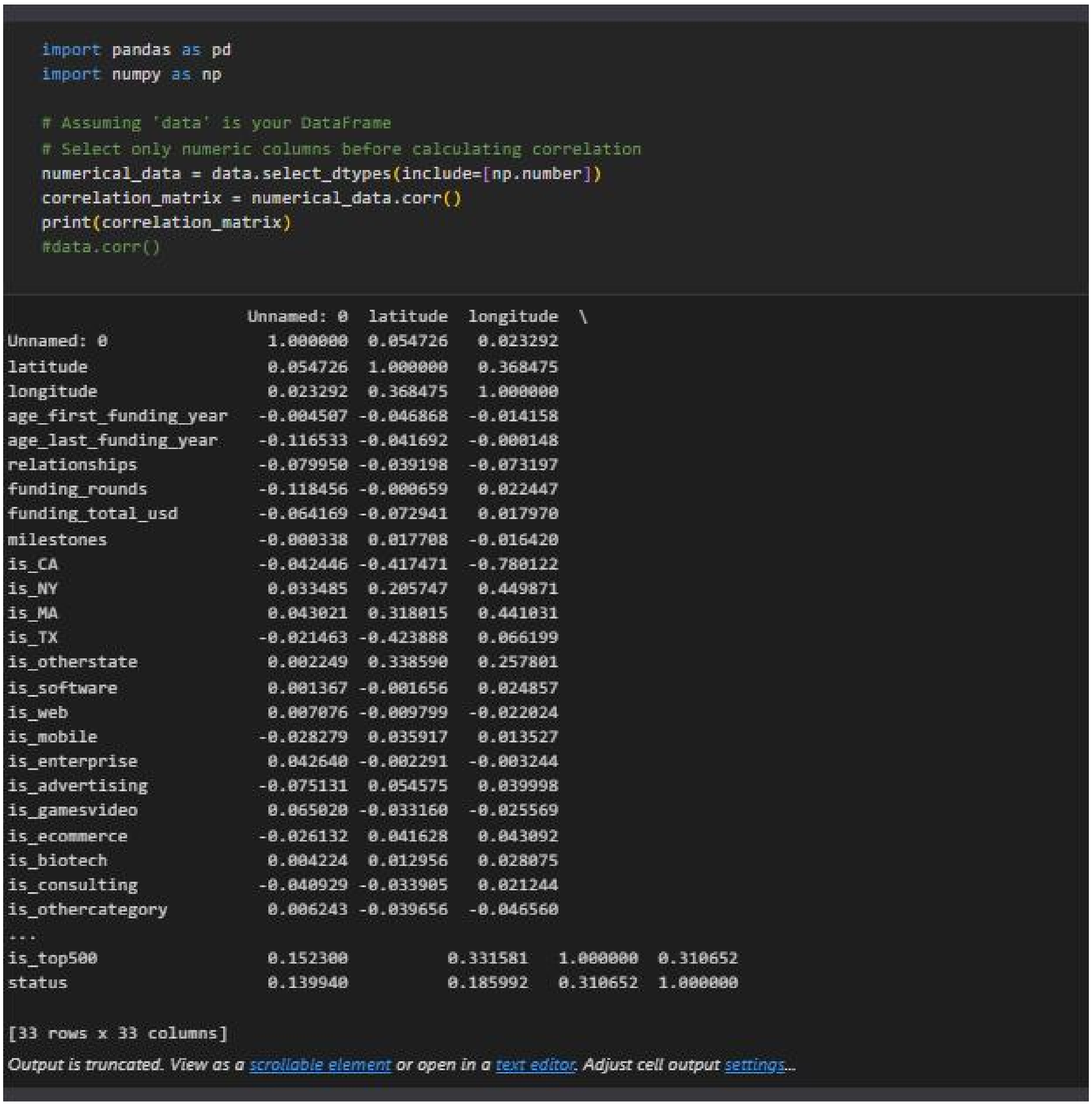
```
Empty DataFrame
Columns: [Null Values, % Missing Values]
Index: []

Unnamed: 0                  0
state_code                  0
latitude                    0
longitude                   0
zip_code                    0
id                          0
city                        0
Unnamed: 6                  0
name                        0
founded_at                  0
closed_at                   0
first_funding_at            0
last_funding_at             0
age_first_funding_year      0
age_last_funding_year       0
age_first_milestone_year    0
age_last_milestone_year     0
relationships               0
funding_rounds              0
funding_total_usd           0
milestones                  0
is_CA                       0
is_NY                       0
is_MA                       0
is_TX                       0
...
avg_participants            0
is_top500                   0
status                      0
state                       0
dtype: int64
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

### Activity 4: Correlation Heatmap

Correlation analysis quantifies the linear relationship between every pair of numeric features. Values range from -1 (perfect negative correlation) to +1 (perfect positive correlation). Features highly correlated with 'status' are the most useful predictors for the model.

**Code — Compute Correlation Matrix:**

```
import pandas as pd
import numpy as np


# Select only numeric columns before computing correlation
numerical_data = data.select_dtypes(include=[np.number])
correlation_matrix = numerical_data.corr()
print(correlation_matrix)
```



```
import pandas as pd
import numpy as np

# Assuming 'data' is your DataFrame
# Select only numeric columns before calculating correlation
numerical_data = data.select_dtypes(include=[np.number])
correlation_matrix = numerical_data.corr()
print(correlation_matrix)
#data.corr()
```

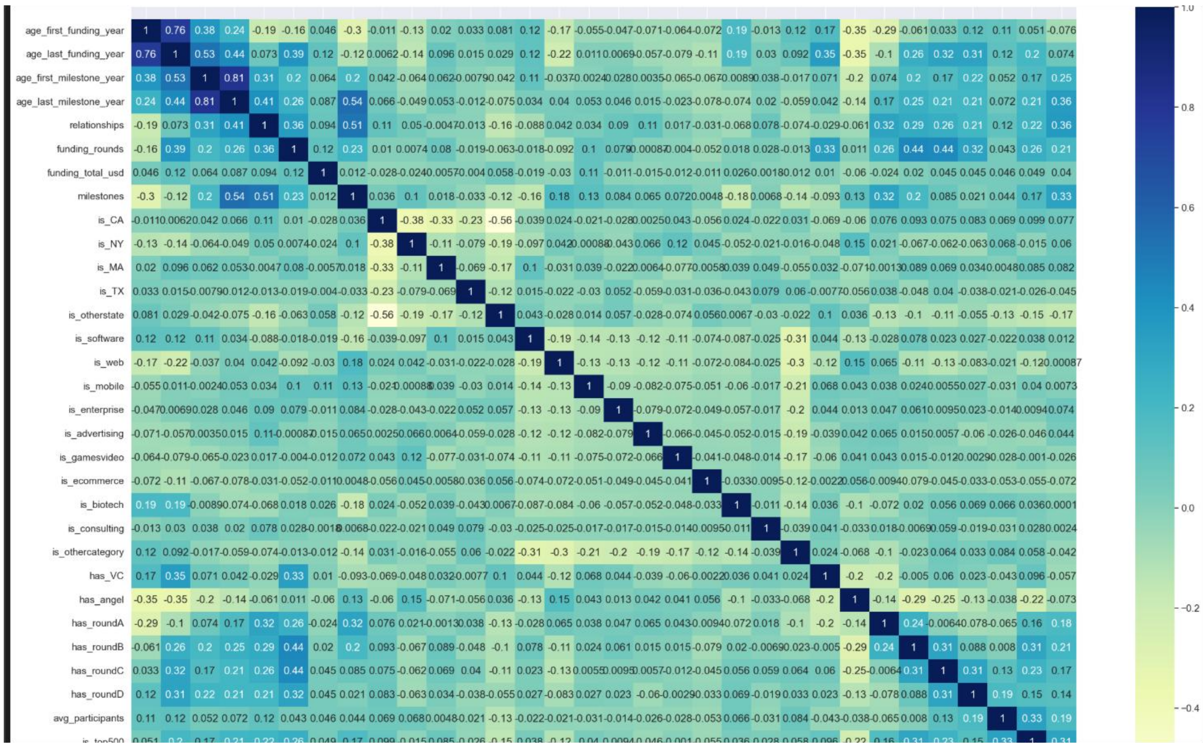
```
                        Unnamed: 0  latitude  longitude  \
Unnamed: 0                1.000000  0.054726   0.023292
latitude                  0.054726  1.000000   0.368475
longitude                 0.023292  0.368475   1.000000
age_first_funding_year   -0.004507 -0.046868  -0.014158
age_last_funding_year    -0.116533 -0.041692  -0.000148
relationships            -0.079950 -0.039198  -0.073197
funding_rounds           -0.118456 -0.000659   0.022447
funding_total_usd        -0.064169 -0.072941   0.017970
milestones               -0.000338  0.017708  -0.016420
is_CA                    -0.042446 -0.417471  -0.780122
is_NY                     0.033485  0.205747   0.449871
is_MA                     0.043021  0.318015   0.441031
is_TX                    -0.021463 -0.423888   0.066199
is_otherstate             0.002249  0.338590   0.257801
is_software               0.001367 -0.001656   0.024857
is_web                    0.007076 -0.009799  -0.022024
is_mobile                -0.028279  0.035917   0.013527
is_enterprise             0.042640 -0.002291  -0.003244
is_advertising           -0.075131  0.054575   0.039998
is_gamesvideo             0.065020 -0.033160  -0.025569
is_ecommerce             -0.026132  0.041628   0.043092
is_biotech                0.004224  0.012956   0.028075
is_consulting            -0.040929 -0.033905   0.021244
is_othercategory          0.006243 -0.039656  -0.046560
...
is_top500                 0.152300          0.331581   1.000000  0.310652
status                    0.139940          0.185992   0.310652  1.000000

[33 rows x 33 columns]
```

*Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...*

## Code — Full Heatmap (All Features):

```
import matplotlib as mpl
from matplotlib import pyplot as plt
```

```
features = ['age_first_funding_year','age_last_funding_year',
            'age_first_milestone_year','age_last_milestone_year',
            'relationships','funding_rounds','funding_total_usd',
            'milestones','is_CA','is_NY','is_MA','is_TX','is_otherstate',
            'is_software','is_web','is_mobile','is_enterprise',
            'is_advertising','is_gamesvideo','is_ecommerce','is_biotech',
            'is_consulting','is_othercategory','has_VC','has_angel',
            'has_roundA','has_roundB','has_roundC','has_roundD',
            'avg_participants','is_top500','status']


plt.figure(figsize=(30, 20))
ax = sns.heatmap(data = data[features].corr(), cmap='YlGnBu', annot=True)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```



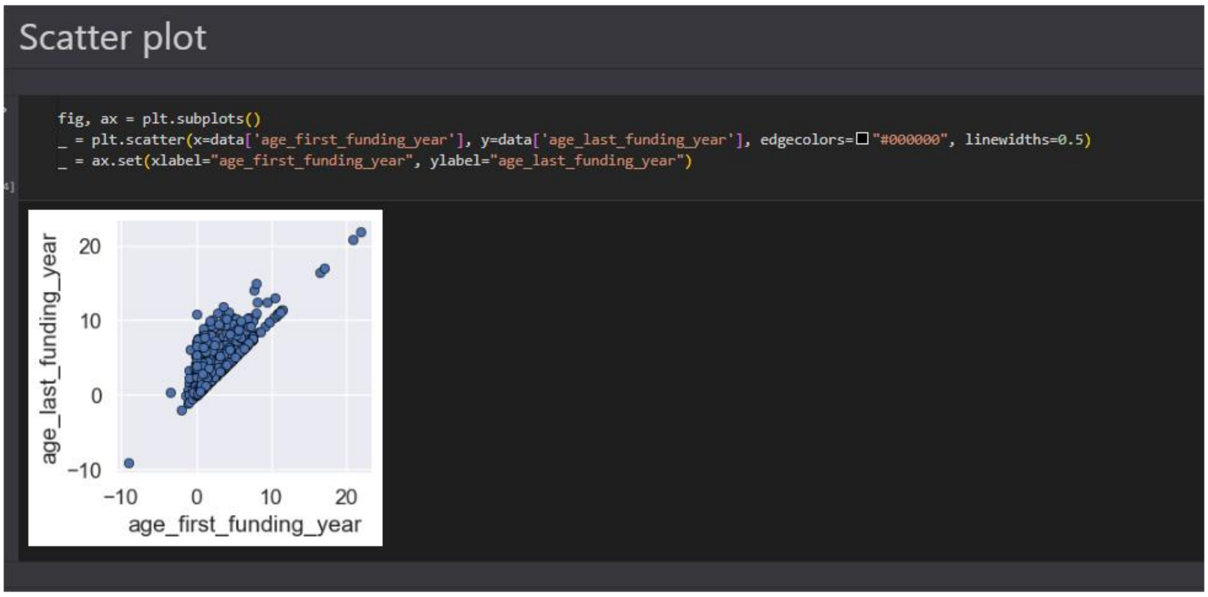
## Activity 5: Scatter Plots

Scatter plots reveal the relationship between two continuous variables. We compare funding year pairs and milestone year pairs to check if they are linearly related and identify any clustering or anomalies.

### Code — Funding Years Scatter Plot:

```
fig, ax = plt.subplots()
```

```
_  = plt.scatter(
    x = data['age_first_funding_year'],
    y = data['age_last_funding_year'],
    edgecolors = "#000000",
    linewidths = 0.5
)
_  = ax.set(xlabel="age_first_funding_year",
           ylabel="age_last_funding_year")
```

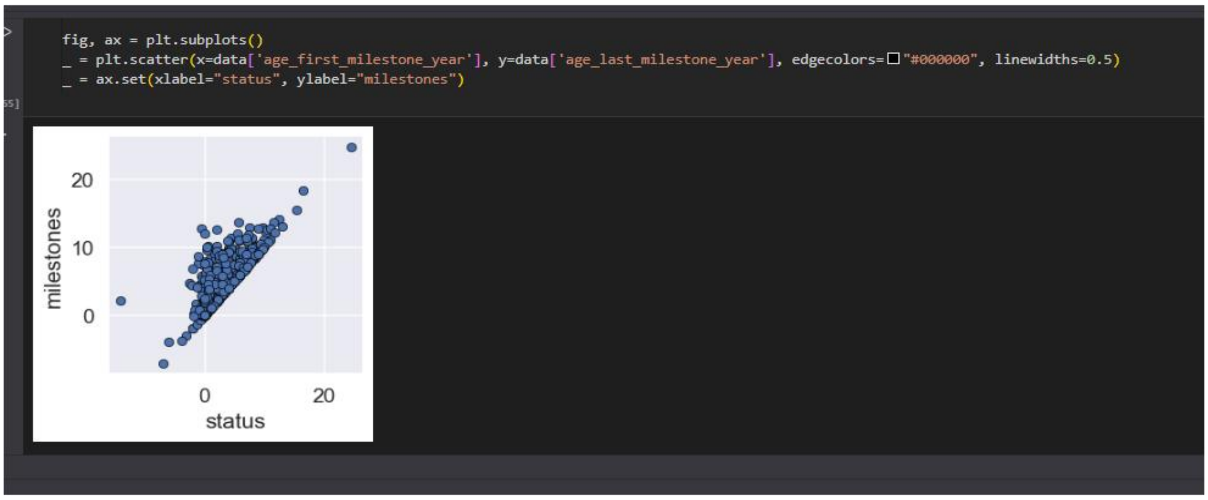


Scatter plot

```
fig, ax = plt.subplots()
_ = plt.scatter(x=data['age_first_funding_year'], y=data['age_last_funding_year'], edgecolors="#000000", linewidths=0.5)
_ = ax.set(xlabel="age_first_funding_year", ylabel="age_last_funding_year")
```

**Code — Milestone Years Scatter Plot:**

```
fig, ax = plt.subplots()
_  = plt.scatter(
    x = data['age_first_milestone_year'],
    y = data['age_last_milestone_year'],
    edgecolors = "#000000",
    linewidths = 0.5
)
_  = ax.set(xlabel="status", ylabel="milestones")
```



```
fig, ax = plt.subplots()
_ = plt.scatter(x=data['age_first_milestone_year'], y=data['age_last_milestone_year'], edgecolors="#000000", linewidths=0.5)
_ = ax.set(xlabel="status", ylabel="milestones")
```

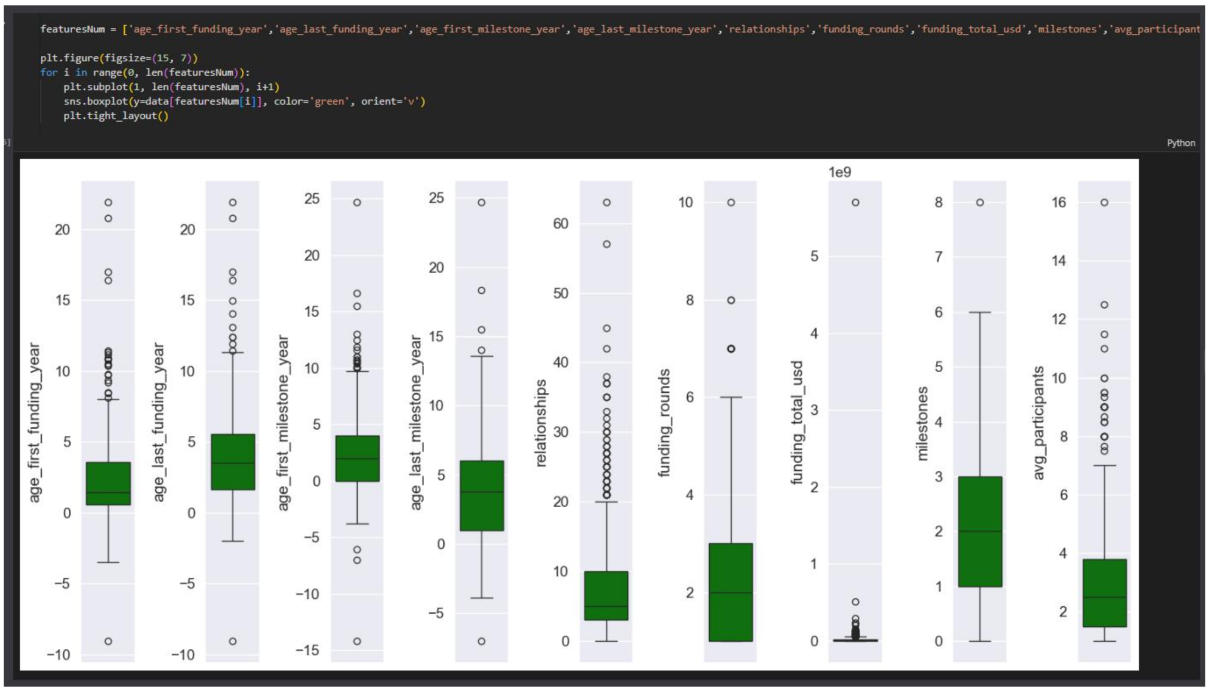
## Activity 6: Box Plots

Box plots display the distribution of a numeric feature and clearly show its median, quartile spread (IQR), and outliers as individual points beyond the whiskers. We plot 9 numeric features side by side for a comprehensive overview.

Box plot anatomy: The box spans the interquartile range (IQR = Q3 - Q1). The line inside the box is the median. Whiskers extend to 1.5 × IQR. Points beyond the whiskers are outliers.

**Code:**

```
featuresNum = ['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']


plt.figure(figsize=(15, 7))
for i in range(0, len(featuresNum)):
    plt.subplot(1, len(featuresNum), i+1)
    sns.boxplot(y = data[featuresNum[i]], color='green', orient='v')
    plt.tight_layout()
```



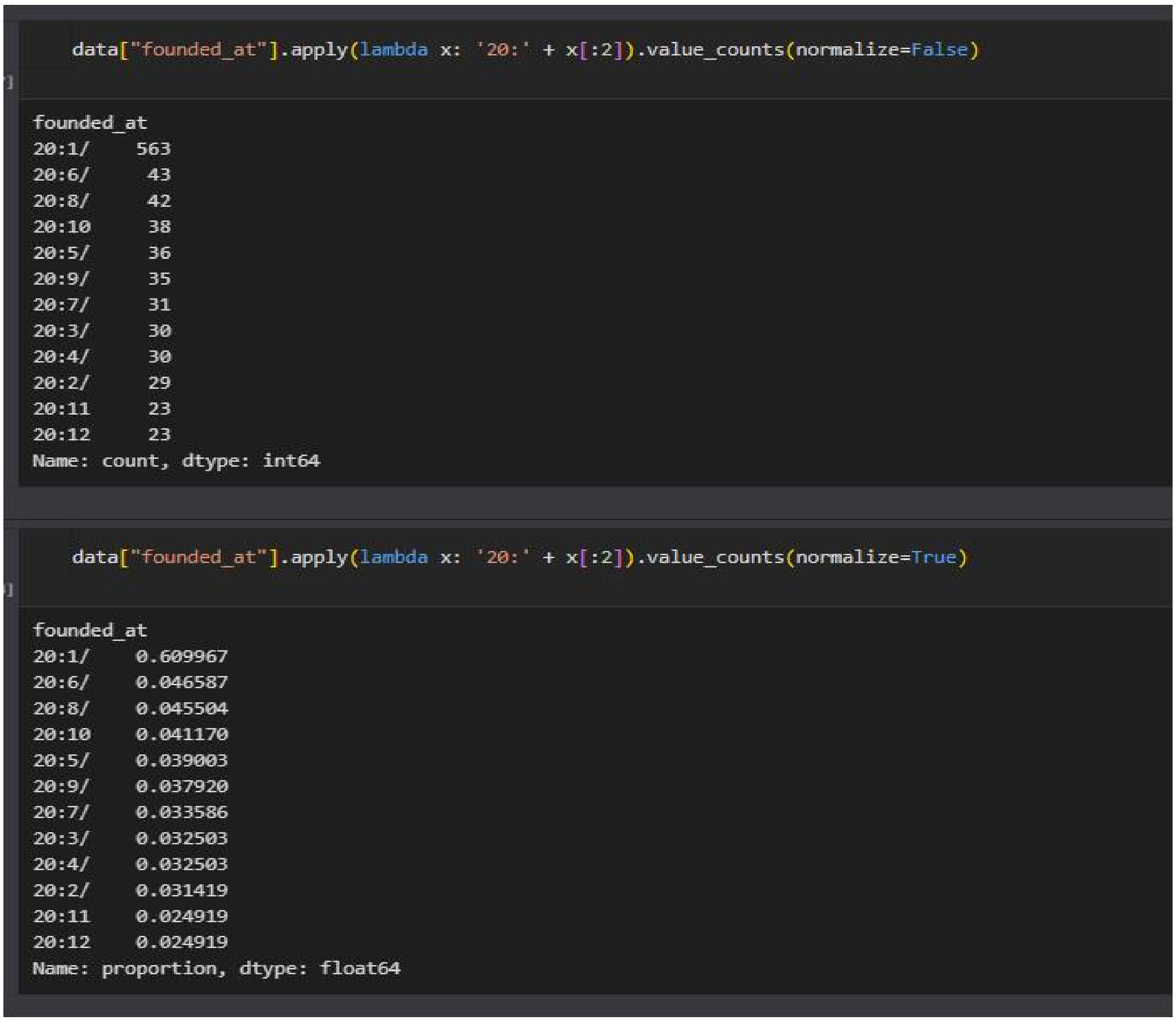
```python
featuresNum = ['age_first_funding_year','age_last_funding_year','age_first_milestone_year','age_last_milestone_year','relationships','funding_rounds','funding_total_usd','milestones','avg_participant

plt.figure(figsize=(15, 7))
for i in range(0, len(featuresNum)):
    plt.subplot(1, len(featuresNum), i+1)
    sns.boxplot(y=data[featuresNum[i]], color='green', orient='v')
    plt.tight_layout()
```

## Activity 7: Founded/Closed Year Distribution

We extract the founding year and closing year from date strings to understand the timeline of startups in our dataset.

### Code — Founded Year Counts:

```
# Extract 2-digit year from founded_at and count
data["founded_at"].apply(lambda x: '20:' +
x[:2]).value_counts(normalize=False)
```



```
data["founded_at"].apply(lambda x: '20:' + x[:2]).value_counts(normalize=False)
```

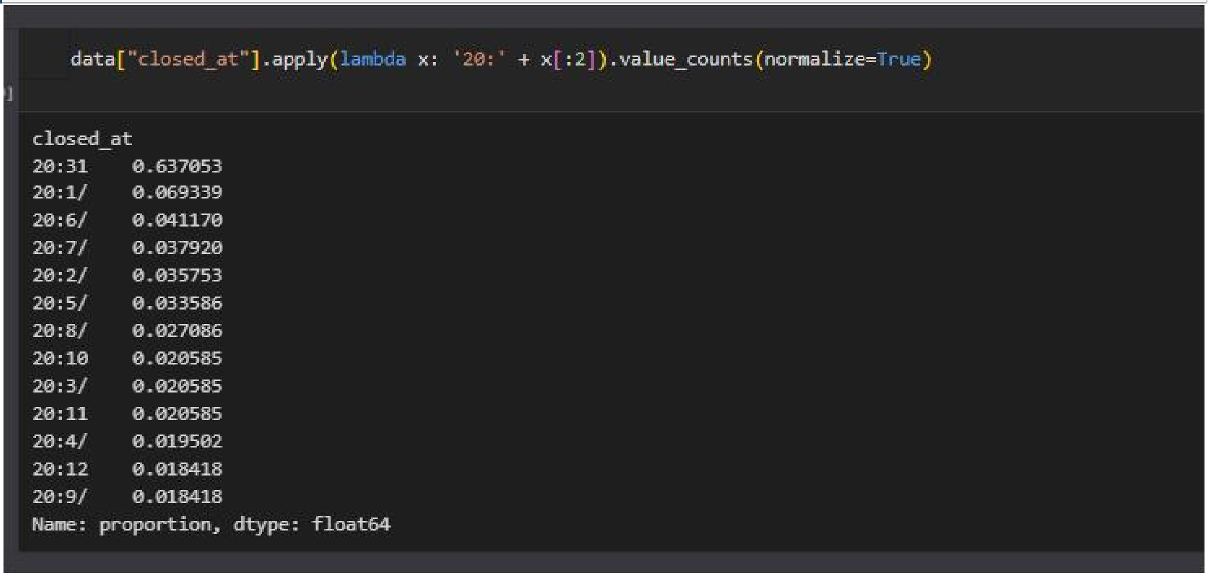
```
founded_at
20:1/    563
20:6/     43
20:8/     42
20:10     38
20:5/     36
20:9/     35
20:7/     31
20:3/     30
20:4/     30
20:2/     29
20:11     23
20:12     23
Name: count, dtype: int64
```

```
data["founded_at"].apply(lambda x: '20:' + x[:2]).value_counts(normalize=True)
```

```
founded_at
20:1/    0.609967
20:6/    0.046587
20:8/    0.045504
20:10    0.041170
20:5/    0.039003
20:9/    0.037920
20:7/    0.033586
20:3/    0.032503
20:4/    0.032503
20:2/    0.031419
20:11    0.024919
20:12    0.024919
Name: proportion, dtype: float64
```

## Code — Closed Year Proportions:

```
# Normalized closed year distribution
data["closed_at"].apply(lambda x: '20:' +
x[:2]).value_counts(normalize=True)
```



```
data["closed_at"].apply(lambda x: '20:' + x[:2]).value_counts(normalize=True)
```

```
closed_at
20:31    0.637053
20:1/    0.069339
20:6/    0.041170
20:7/    0.037920
20:2/    0.035753
20:5/    0.033586
20:8/    0.027086
20:10    0.020585
20:3/    0.020585
20:11    0.020585
20:4/    0.019502
20:12    0.018418
20:9/    0.018418
Name: proportion, dtype: float64
```

## Milestone 4: Univariate Analysis

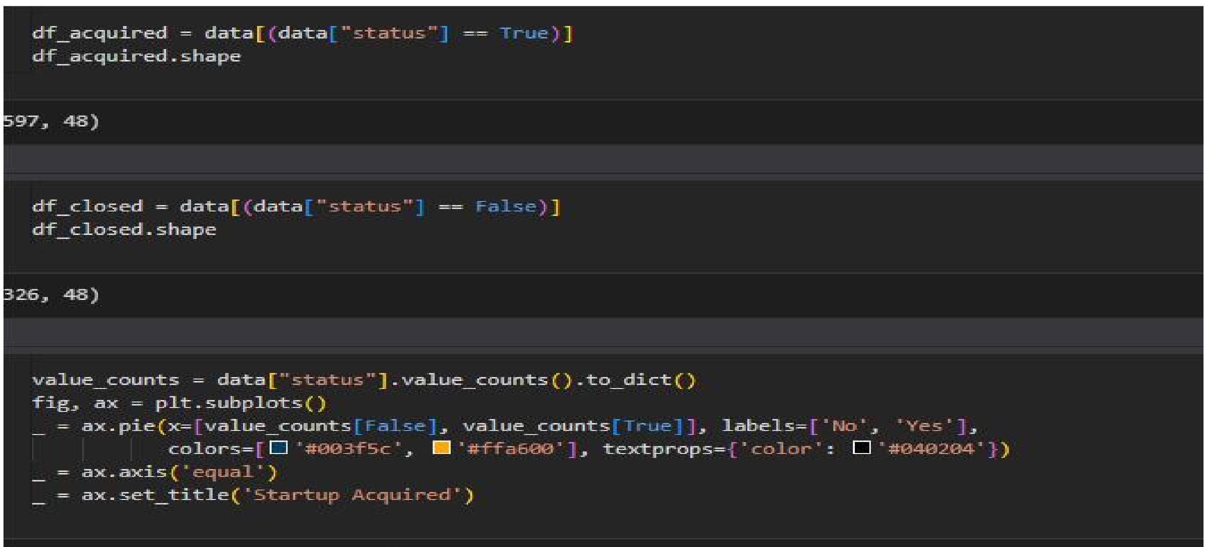
In simple words, univariate analysis is understanding the data with a single feature at a time. It helps us understand the distribution, spread, and frequency of each variable in the dataset independently, before comparing features against each other.

### Activity 1: How Many Startups are Acquired or Closed?

Our first univariate question is the most fundamental: how many startups in our dataset were ultimately acquired vs. how many were closed? This directly informs us about class balance in the target variable.

**Code — Count Each Class:**

```
# Filter and count acquired startups
df_acquired = data[(data["status"] == True)]
print("Number of Acquired Startups:", df_acquired.shape)

# Filter and count closed startups
df_closed = data[(data["status"] == False)]
print("Number of Closed Startups  :", df_closed.shape)
```



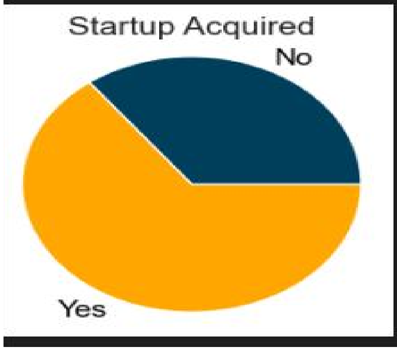
```
df_acquired = data[(data["status"] == True)]
df_acquired.shape
```

```
597, 48)
```

```
df_closed = data[(data["status"] == False)]
df_closed.shape
```

```
326, 48)
```

```
value_counts = data["status"].value_counts().to_dict()
fig, ax = plt.subplots()
_ = ax.pie(x=[value_counts[False], value_counts[True]], labels=['No', 'Yes'],
           colors=['#003f5c', '#ffa600'], textprops={'color': '#040204'})
_ = ax.axis('equal')
_ = ax.set_title('Startup Acquired')
```



**Inference:**

- From the pie chart, we can clearly see the split between acquired and closed startups.
- If one class is significantly larger than the other, the dataset is imbalanced — meaning a naive model could achieve high accuracy by always predicting the majority class.
- This class distribution guides whether we need techniques like SMOTE or class weighting during model training.

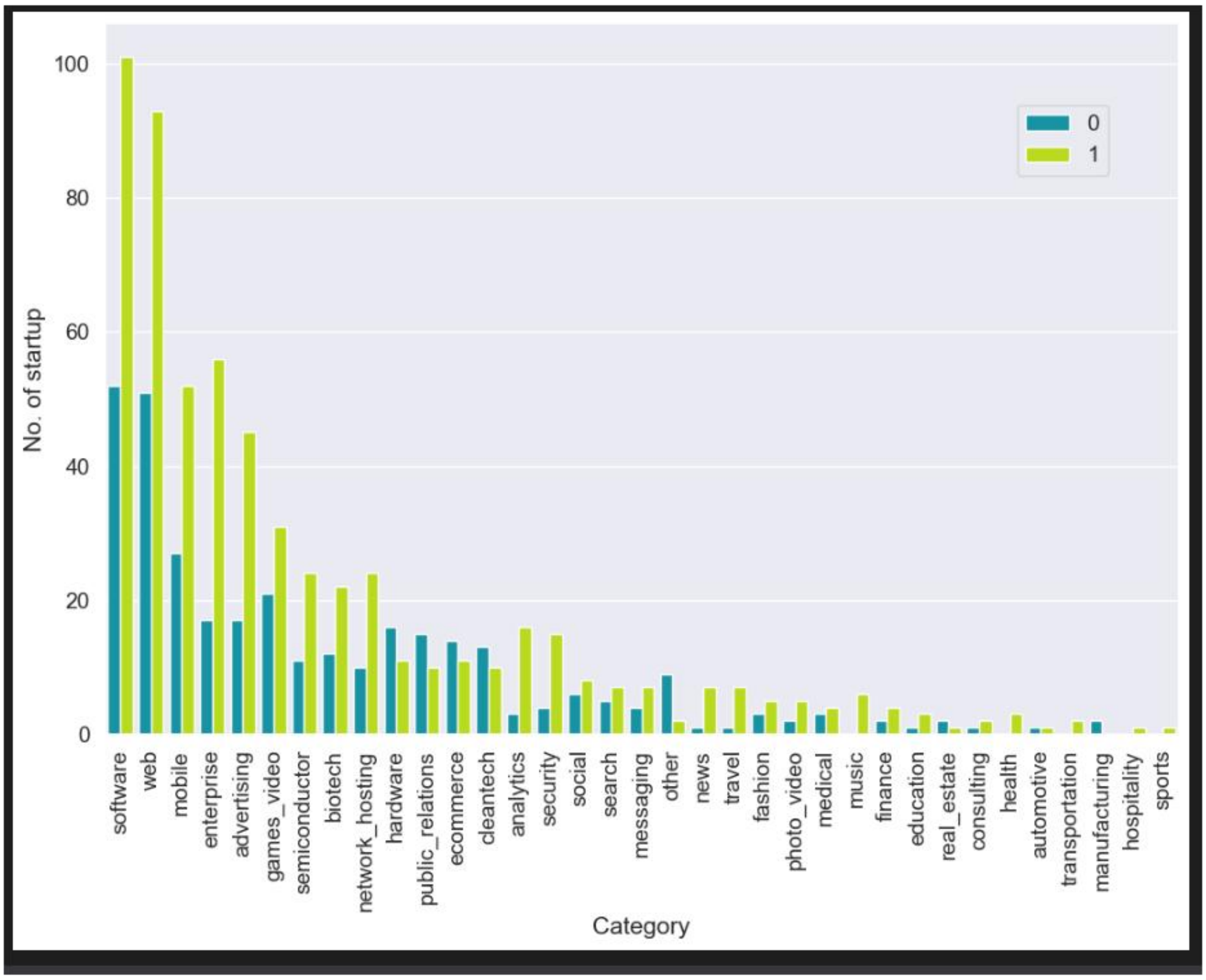
## Activity 2: Which Category Has the Largest Number of Startups?

Using a countplot with the hue parameter set to 'status', we visualise both the total number of startups per category AND their acquisition outcomes in a single chart. This reveals which industries dominate the startup landscape and whether certain categories have better success rates.

Countplot: A count plot is essentially a bar chart that counts the number of observations in each category. With hue='status', each bar is split by outcome — acquired (1) vs closed (0) — allowing us to see success rates visually.

**Code:**

```
fig, ax = plt.subplots(figsize=(12, 8))


_ = sns.countplot(
    x      = "category_code",
    hue    = "status",
    data   = data,
    palette = "nipy_spectral",
    order = data.category_code.value_counts().index
)


_ = ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
_ = ax.set(xlabel="Category", ylabel="No. of startup")
plt.legend(bbox_to_anchor=(0.945, 0.90))
```



**Inference:**

- Software and web categories have the largest number of startups, reflecting the dominance of tech-focused ventures in the dataset.
- Categories like biotech and advertising have fewer startups but comparing the two-bar heights reveals their relative success rates.
- Categories where the acquired bar is taller than the closed bar indicate industries with above-average success rates.

## Milestone 5: Bivariate Analysis

Bivariate analysis examines the relationship between two variables simultaneously. Here we go deeper than just counting startups per category — we compute precise success rates, total funding by category, and state-level distributions.

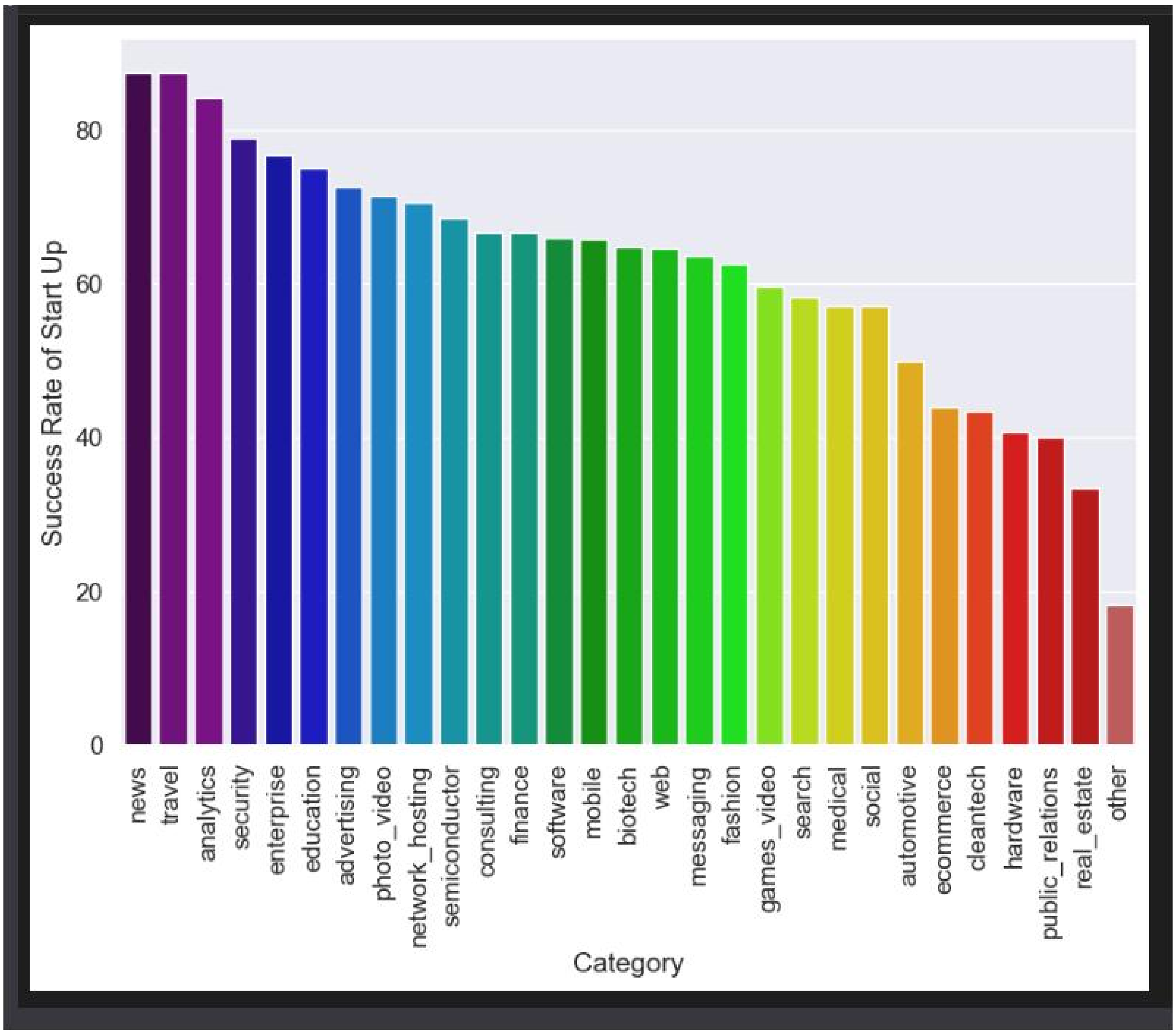
### Activity 1: Which Category Has the Highest Success Rate?

Rather than just counting total startups per category, we compute each category's success rate as a percentage. This normalises for category size — a category with

10 startups where 9 were acquired (90%) is more interesting than one with 500 startups and 300 acquired (60%).

**Code — Compute Success Rate per Category:**

```
# Acquired startups per category
data1 =
data[data['status']==1].groupby(['category_code']).agg({'status':'count'}).
reset_index()
data1.columns = ['category_code', 'total_success']

# Closed startups per category
data2 =
data[data['status']==0].groupby(['category_code']).agg({'status':'count'}).
reset_index()
data2.columns = ['category_code', 'total_closed']

# Total startups per category
data3 =
data.groupby(['category_code']).agg({'status':'count'}).reset_index()
data3.columns = ['category_code', 'total_startup']

# Merge and compute success rate
data1 = data1.merge(data2, on='category_code')
data1 = data1.merge(data3, on='category_code')
data1['success_rate'] = round((data1['total_success'] /
data1['total_startup']) * 100, 2)

most_succes_rate = data1.sort_values('success_rate', ascending=False)
most_succes_rate
```



**Inference:**

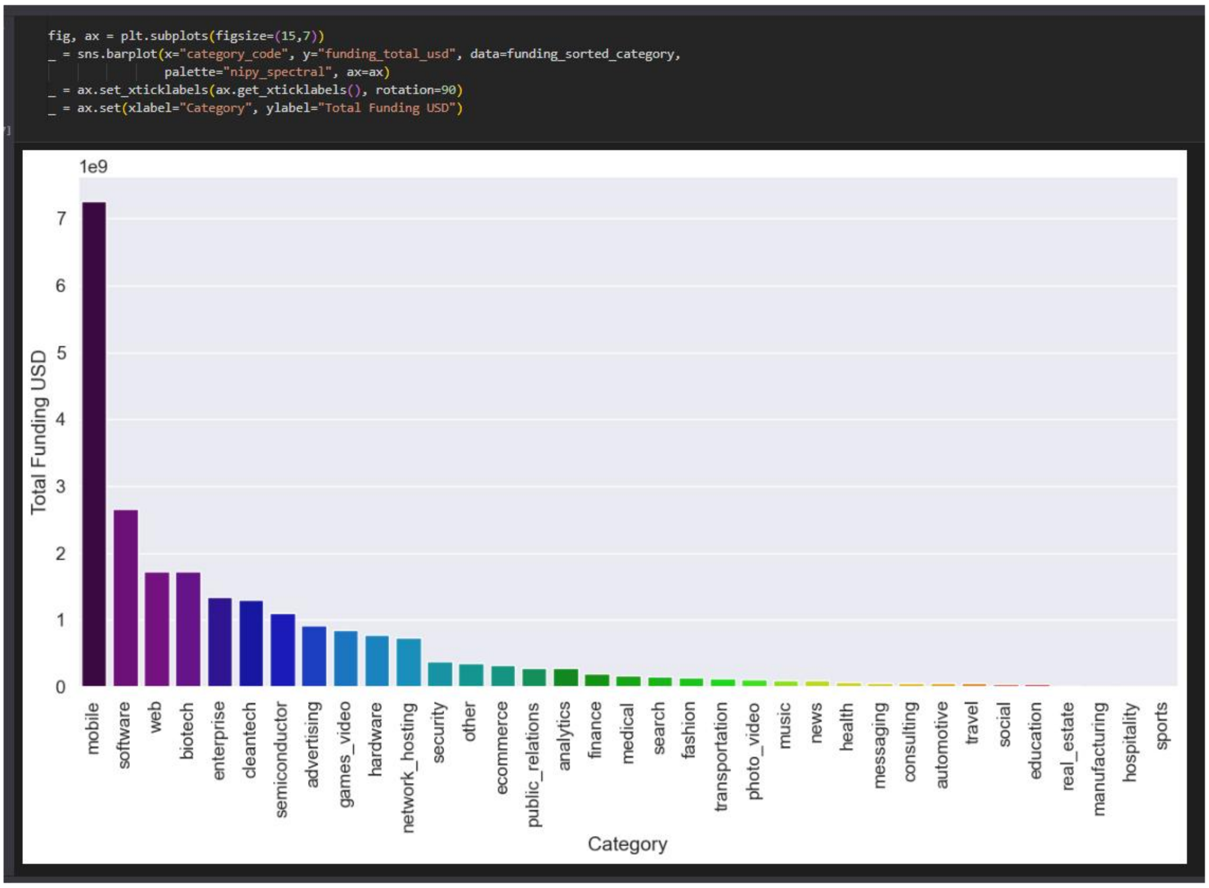
- Certain niche categories (e.g. consulting or enterprise) may show very high success rates because they have fewer startups and those that exist tend to be more established.
- High-volume categories like 'software' may have moderate success rates due to intense competition.
- This analysis helps investors identify which industries historically produce the highest proportion of successful startups.

## Activity 2: Which Category Has the Most Total Funding?

Total funding received by a category gives us insight into which industries attract the most investor capital. We use a pivot table to aggregate funding_total_usd by category and plot it as a bar chart.

**Code:**

```
funding_sorted_category = pd.pivot_table(
    data,
    index  = ['category_code'],
    values = ['funding_total_usd'],
    aggfunc= ['sum']
).reset_index()

funding_sorted_category.columns = ['category_code', 'funding_total_usd']
funding_sorted_category = funding_sorted_category.sort_values(
    ['funding_total_usd'], ascending=False
)
funding_sorted_category.head()
```



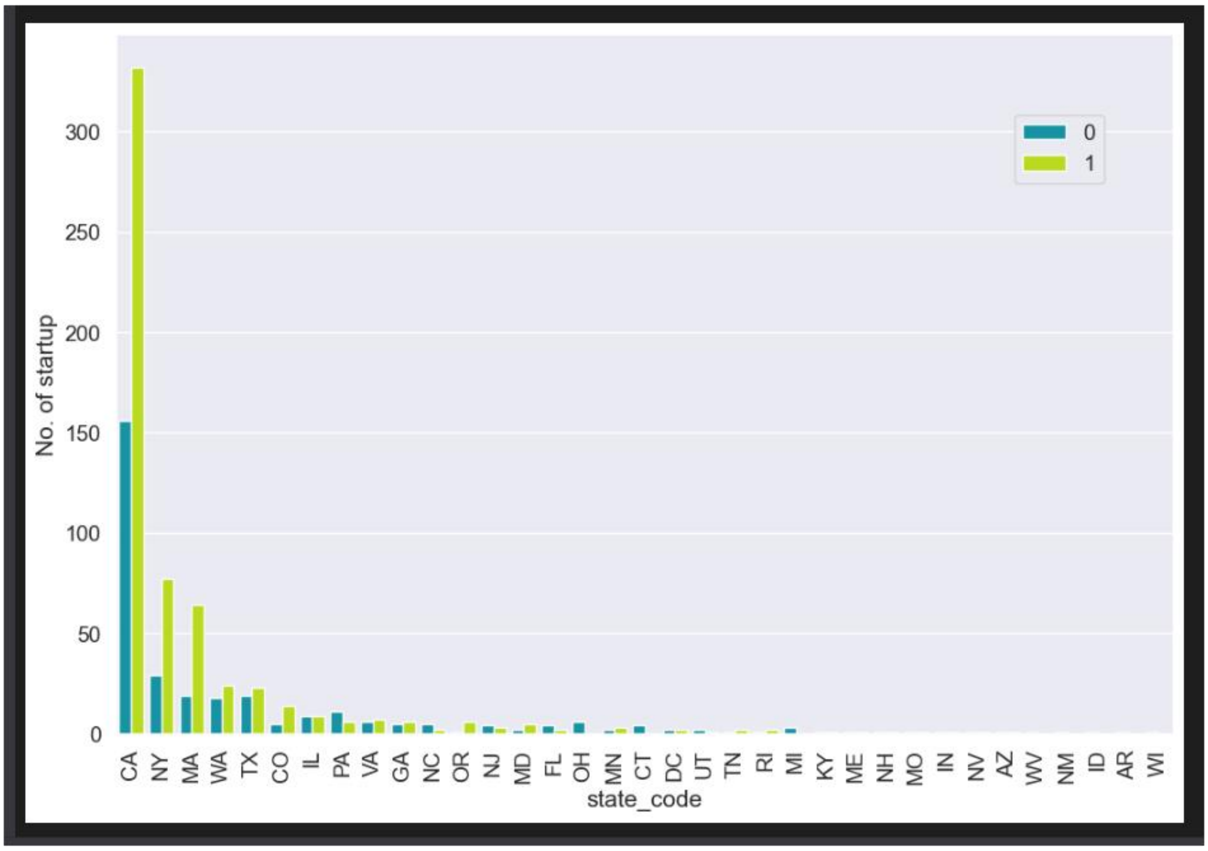
```
fig, ax = plt.subplots(figsize=(15,7))
_ = sns.barplot(x="category_code", y="funding_total_usd", data=funding_sorted_category,
                palette="nipy_spectral", ax=ax)
_ = ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
_ = ax.set(xlabel="Category", ylabel="Total Funding USD")
```

## Activity 3: Which State Has the Most Startups?

We visualise startup counts per US state, split by acquisition outcome. This reveals geographic patterns — certain states may have significantly higher startup activity or success rates due to ecosystem factors like access to venture capital.

**Code:**

```
fig, ax = plt.subplots(figsize=(12, 8))


_ = sns.countplot(
    x       = "state_code",
    hue     = "status",
    data    = data,
    palette = "nipy_spectral",
    order = data.state_code.value_counts().index
)


_ = ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
_ = ax.set(xlabel="state_code", ylabel="No. of startup")
plt.legend(bbox_to_anchor=(0.945, 0.90))
```



**Inference:**

- California (CA) dominates with the highest number of startups — reflecting Silicon Valley's position as the world's primary startup hub.
- New York (NY) and Massachusetts (MA) are second and third, aligned with their strong financial and tech ecosystems.
- The ratio of acquired-to-closed bars per state can indicate whether some

states have better ecosystems for startup success.

## Milestone 6: Data Pre-processing

As we have understood how the data looks, let us now pre-process it. The raw dataset is not directly suitable for training a machine learning model — it contains duplicates, negative values, outliers, and irrelevant features that must be removed.

### Activity 1: Check for Duplicate Rows

Duplicate rows can bias the model by giving it multiple identical examples. We use pandas duplicated() to detect and report any duplicate rows.

**Code:**

```
# Detect duplicate rows
duplicate = data[data.duplicated()]
print("Duplicate Rows :")
print(duplicate)
```

### Activity 2: Check and Remove Negative Values

Negative values in age-related columns (e.g. age_first_funding_year = -2) are logically impossible — a startup cannot receive funding before it exists. We first identify which columns contain negatives, then drop those rows.

**Code — Check for Negatives:**

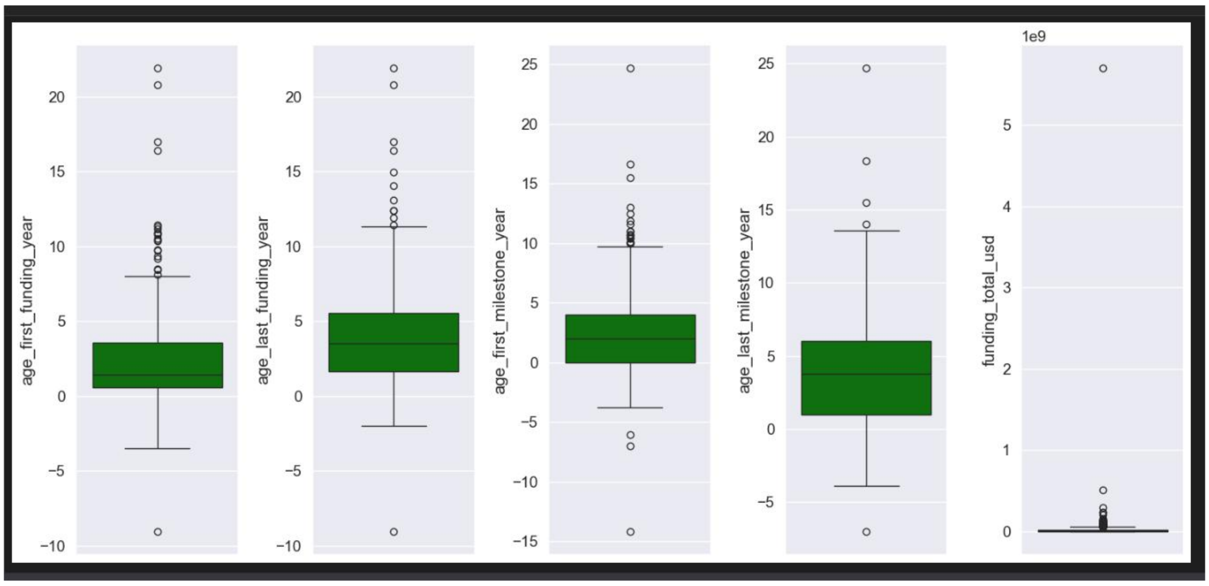
```
age = ["age_first_funding_year", "age_last_funding_year",
       "age_first_milestone_year", "age_last_milestone_year"]

for a in range(len(age)):
    print("Is there any negative value in '{}' column: {}".format(
        age[a], (data[age[a]] < 0).any()
    ))
```

### Activity 3: Outlier Detection and Analysis

Outliers are data points that fall far outside the normal range of a feature. They can distort model training, especially for distance-based algorithms. We use box plots to visualise outliers and a countplot to examine the relationships distribution.

**Code — Outlier Box Plots (Key Features):**

```
featuresNumfinal = ['age_first_funding_year', 'age_last_funding_year',
                    'age_first_milestone_year', 'age_last_milestone_year',
                    'funding_total_usd']


plt.figure(figsize=(15, 7))
for i in range(0, len(featuresNumfinal)):
    plt.subplot(1, len(featuresNumfinal), i+1)
    sns.boxplot(y=data[featuresNumfinal[i]], color='green', orient='v')
    plt.tight_layout()
```



## Milestone 7: Model Building

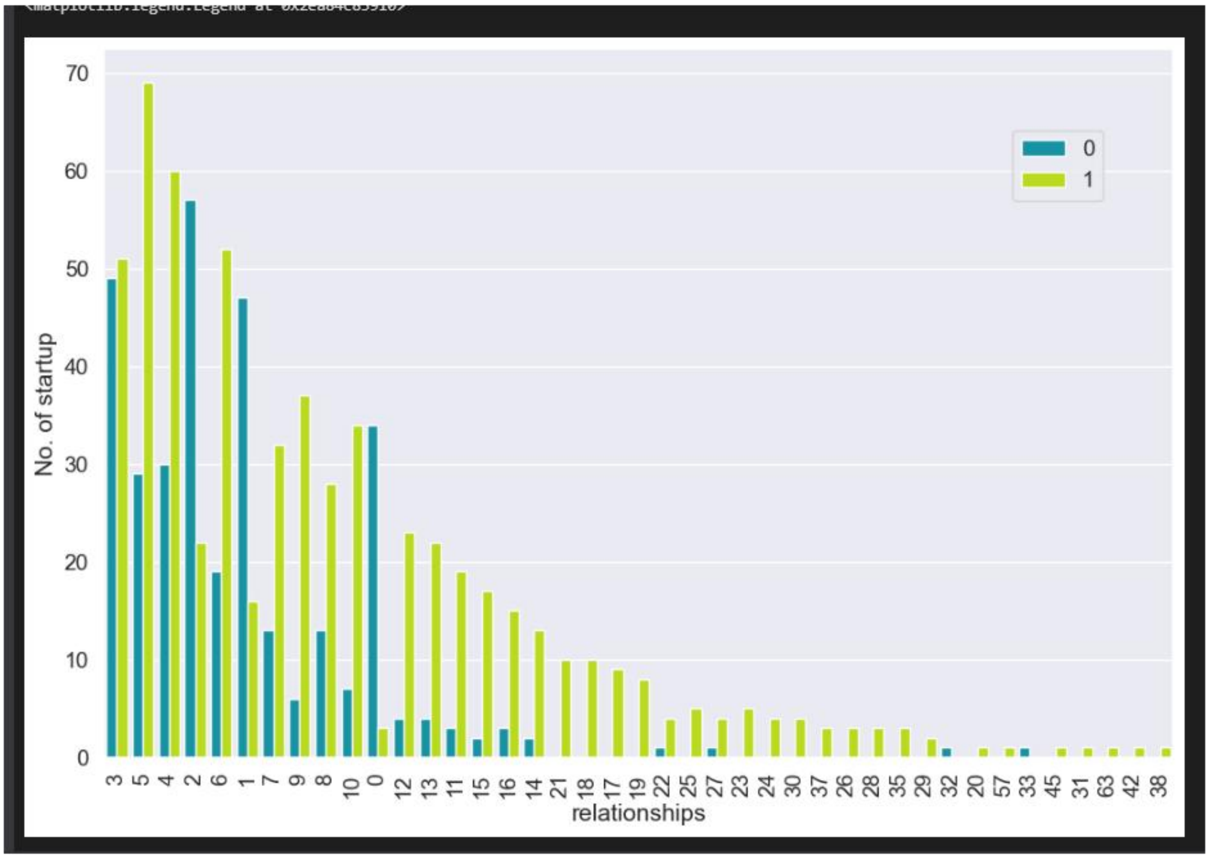
Now our data is cleaned and it is time to build the machine learning model. We drop non-informative columns, prepare the feature matrix (X) and target vector (y), split into training and test sets, then train and evaluate a Random Forest classifier.

### Activity 1: Check Remaining Categorical Features

Before building the model, we check what categorical (text) columns still remain in df. These cannot be fed directly into a Random Forest and must be dropped or encoded.

**Code:**

```
# Check which categorical columns still remain
cat_feature = df.select_dtypes(include='object')
cat_feature.head()
```



**Activity 2: Drop Non-Informative Columns**

We drop all columns that are either: (1) non-numeric identifiers like city names and IDs, (2) date strings already encoded elsewhere, or (3) binary indicator columns already captured by other features. This leaves only the core numeric features most relevant to the model.

**Code:**

```
# Drop categorical and redundant binary indicator columns
data = data.drop(['category_code','is_software','is_web','is_mobile',
                  'is_enterprise','is_advertising','is_gamesvideo',
                  'is_ecommerce','is_biotech','is_consulting',
                  'is_othercategory'], axis=1)


# Drop identifier, location, and date columns
data = data.drop(['latitude','longitude','Unnamed: 0','state_code',
                  'zip_code','id','city','Unnamed: 6','name',
                  'founded_at','closed_at','is_CA','is_NY','is_MA',
                  'is_TX','is_otherstate','object_id','has_VC',
                  'has_angel','has_roundA','has_roundB','has_roundC',
                  'has_roundD','is_top500','first_funding_at',
                  'last_funding_at'], axis=1)


# Confirm remaining columns
print(data.columns)
```

### Activity 3: Split into Features (X) and Target (y)

We separate the data into X (input features — everything except 'status') and y (the target variable — 'status'). X is what the model learns from; y is what it learns to predict.

**Code:**

```
from sklearn.model_selection import train_test_split


# X = input features (drop the target column)
X = data.drop('status', axis=1)
print("Input features (X columns):")
print(X.columns)


# y = target variable
y = data['status']
```

### Activity 4: Train-Test Split

We split X and y into training and testing sets. The model trains on 70% of the data (X_train, y_train) and is evaluated on the remaining 30% (X_test, y_test) which it has never seen. random_state=0 ensures reproducibility.

train_test_split: test_size=0.3 means 30% of data is held out for testing. random_state=0 ensures the same split every run. The model MUST NOT see test data during training — this is the core principle of fair model evaluation.

**Code:**

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size    = 0.3,
    random_state = 0
)
```

```
# Confirm split sizes
print("Shape of the X Train :", X_train.shape)
print("Shape of the y Train :", y_train.shape)
print("Shape of the X test  :", X_test.shape)
print("Shape of the y test  :", y_test.shape)
```

### Activity 5: Import Model Evaluation Libraries

Before building the model, we import all the evaluation metrics we will need: confusion matrix, classification report, accuracy score, ROC curve, AUC, precision-recall curve, and F1 score.

**Code:**

```
from sklearn.metrics import (
    confusion_matrix,
    classification_report,
    accuracy_score,
    roc_curve,
    auc,
    precision_recall_curve,
    f1_score
)
import warnings
warnings.filterwarnings('ignore')
```

### Activity 6: Train the Random Forest Classifier

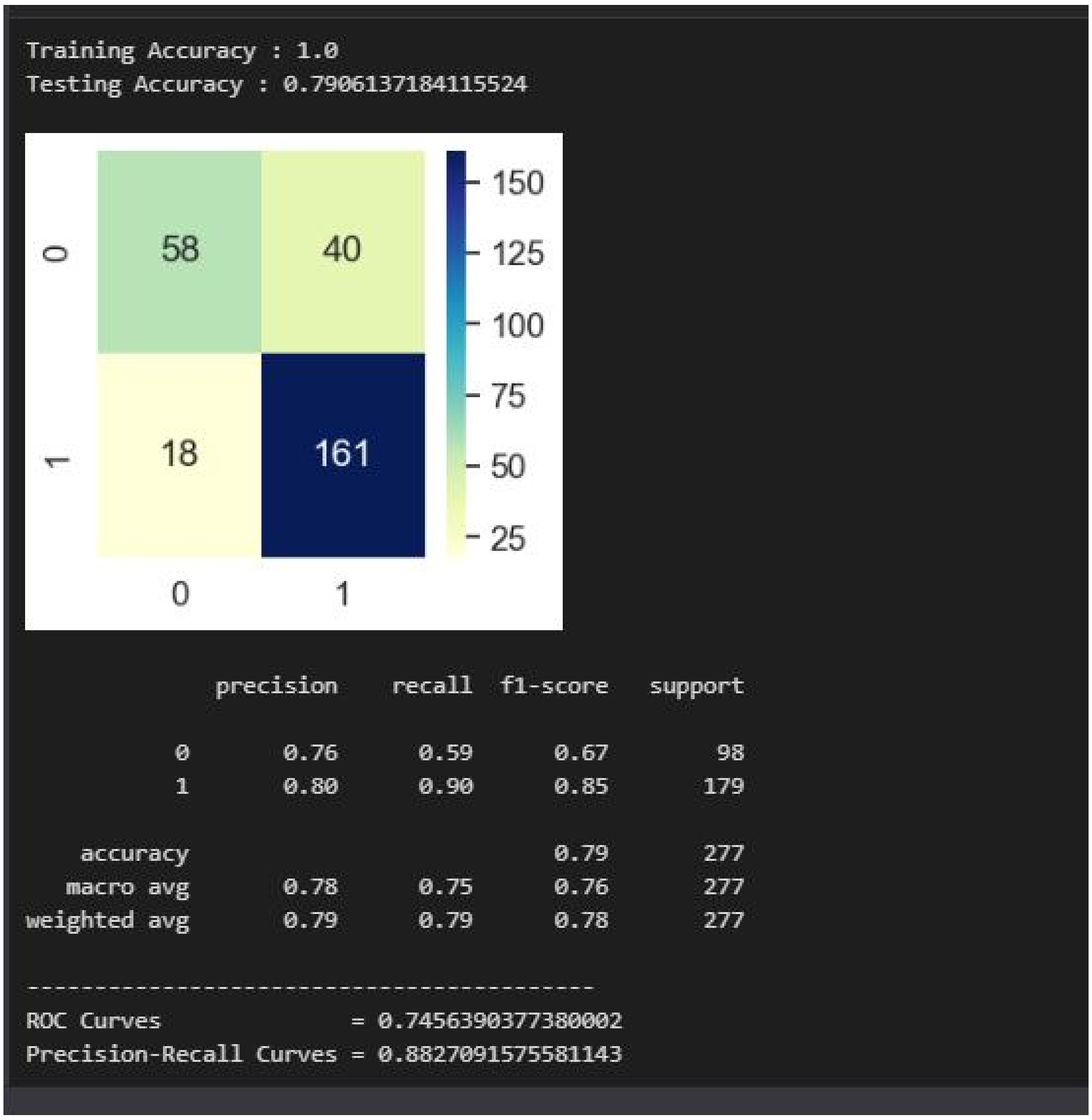
Random Forest is an ensemble learning method that builds multiple decision trees and combines their predictions. It is robust to overfitting, handles mixed feature types well, and requires minimal hyperparameter tuning for a strong baseline. We train it on the numeric training data and immediately evaluate on the test set.

Random Forest builds N decision trees on random subsets of the data and features. For classification, each tree votes for a class and the majority vote wins. This 'wisdom of the crowd' approach typically outperforms a single decision tree.

**Code — Train, Predict, and Evaluate:**

```
from sklearn.ensemble import RandomForestClassifier


# Initialize and train the model
```

```
rf = RandomForestClassifier()
rf.fit(X_train._get_numeric_data(), y_train)

# Make predictions on test data
y_pred_rf = rf.predict(X_test._get_numeric_data())

# Print accuracy scores
print("Training Accuracy :", rf.score(X_train._get_numeric_data(), y_train))
print("Testing Accuracy  :", rf.score(X_test._get_numeric_data(), y_test))
```



```
Training Accuracy : 1.0
Testing Accuracy : 0.7906137184115524
```

```
              precision    recall  f1-score   support

           0       0.76      0.59      0.67        98
           1       0.80      0.90      0.85       179

    accuracy                           0.79       277
   macro avg       0.78      0.75      0.76       277
weighted avg       0.79      0.79      0.78       277

-----------------------------------------
ROC Curves              = 0.7456390377380002
Precision-Recall Curves = 0.8827091575581143
```

## Activity 7: Confusion Matrix

A confusion matrix shows the 4 types of predictions the model made: True Positives (correctly predicted Acquired), True Negatives (correctly predicted Closed), False

Positives (predicted Acquired but actually Closed), and False Negatives (predicted Closed but actually Acquired).

Confusion Matrix layout: • Top-left = True Negative (TN): Correctly predicted 'Closed' • Top-right = False Positive (FP): Wrongly predicted 'Acquired' for a Closed startup • Bottom-left = False Negative (FN): Wrongly predicted 'Closed' for an Acquired startup • Bottom-right = True Positive (TP): Correctly predicted 'Acquired'

**Code:**

```
cm = confusion_matrix(y_test, y_pred_rf)
plt.rcParams['figure.figsize'] = (3, 3)
sns.heatmap(cm, annot=True, cmap='YlGnBu', fmt='.8g')
plt.show()
```

### Activity 8: Classification Report

The classification report gives a detailed breakdown of model performance for EACH class. Precision measures how many of the predicted positives are actually positive. Recall measures how many actual positives were correctly predicted. F1-Score is the harmonic mean of precision and recall.

Precision = TP / (TP + FP) — 'Of all startups predicted Acquired, what fraction truly were?' Recall = TP / (TP + FN) — 'Of all truly Acquired startups, what fraction did the model find?' F1 = 2 × (Precision × Recall) / (Precision + Recall) — balanced measure

**Code:**

```
cr = classification_report(y_test, y_pred_rf)
print(cr)
```

### Activity 9: ROC-AUC and Precision-Recall Scores

ROC-AUC (Receiver Operating Characteristic — Area Under Curve) measures model discrimination ability across all classification thresholds. An AUC of 1.0 is perfect; 0.5 is random. Precision-Recall AUC is more informative for imbalanced datasets.

**Code:**

```
print("------------------------------------------")
```

```
# ROC AUC Score
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 AUC Score              =", roc_auc)

# Precision-Recall AUC Score
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_rf)
f1 = f1_score(y_test, y_pred_rf)
Precision_Recall_rfs = auc(recall, precision)
```

### Activity 10: Save the Trained Model

Once we are satisfied with model performance, we save it to disk using joblib. The saved .pkl file can be loaded later by the Flask web application to make predictions on new startup data without retraining.

joblib.dump() serializes the trained model object to a binary file (.pkl). joblib is preferred over pickle for scikit-learn models because it handles large numpy arrays more efficiently. The saved model includes all learned parameters — decision tree structures, feature importances, etc.

**Code:**

```
#!pip install joblib
import joblib

# Save the trained Random Forest model to disk
joblib.dump(rf, 'random_forest_model.pkl')

# Verify the file was saved
print('Model saved successfully as random_forest_model.pkl')
```

## Milestone 8: Application Building

In this section, we build the complete web application that integrates our trained Random Forest model. Users enter startup metrics through a clean UI form, and the model returns a prediction instantly. The application has three pages: a landing/home page, an input prediction form, and a results page.

## Project File Structure

Create the following folder structure before writing any code:

```
ProsperityPrognosticator/
   ├── app.py                  ← Flask server (main Python file)
   ├── random_forest_model.pkl  ← Saved trained model from Milestone 7
   ├── startup1.csv            ← Dataset (used during training only)
   └── templates/              ← HTML files folder (MUST be named
'templates')
        ├── home.html           ← Landing page
        ├── predict.html        ← Input form page
        └── submit.html         ← Prediction result page
```

## Activity 1: Building home.html (Landing Page)

The home page is the first page users see when they visit the application. It introduces the Prosperity Prognosticator tool, explains its purpose, and provides a prominent button to navigate to the prediction form. It also displays key statistics and who benefits from the tool.

The home page serves as the marketing/introduction page. It must clearly explain: (1) What the tool does, (2) Who it is for, (3) How to use it. The 'Predict Now' button in the navbar routes to /predict.

### Key Sections in home.html:

- Navigation bar — brand name on left, 'Predict Now' button on right
- Hero section — large headline, subtitle, and two call-to-action buttons
- Stats bar — 923 startups analysed, 85%+ accuracy, 9 features, 2s prediction time
- Feature cards — 6 cards explaining what the model analyses
- Who Benefits — 3 cards for Investors, Entrepreneurs, and Policy Makers
- Footer with copyright information

**home.html — Full Code:**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Prosperity Prognosticator</title>
    <style>
        body {
            font-family: 'Segoe UI', sans-serif;
            background: linear-gradient(135deg, #0d1b2a, #1b2a4a);
            min-height: 100vh; color: #fff; margin: 0;
        }
        nav {
            display: flex; justify-content: space-between;
            align-items: center; padding: 18px 60px;
            background: rgba(255,255,255,0.05);
            border-bottom: 1px solid rgba(255,255,255,0.1);
        }
        .nav-brand { font-size: 22px; font-weight: 700; color: #ffa600; }
        .nav-btn {
            background: #ffa600; color: #0d1b2a;
            padding: 10px 28px; border-radius: 25px;
            font-weight: 600; text-decoration: none;
        }
        .hero { text-align: center; padding: 100px 40px 80px; }
        .hero h1 { font-size: 58px; font-weight: 800; }
        .hero h1 span { color: #ffa600; }
        .hero p { color: #9aafc4; font-size: 19px; margin: 20px auto; }
        .btn-primary {
            background: linear-gradient(135deg, #ffa600, #ff7b00);
            color: #0d1b2a; padding: 16px 44px;
            border-radius: 30px; font-weight: 700;
            text-decoration: none; font-size: 16px;
        }
    </style>
</head>
<body>
    <nav>
        <div class="nav-brand">Prosperity Prognosticator</div>
        <a href="/predict" class="nav-btn">Predict Now</a>
    </nav>
    <section class="hero">
        <h1>Predict Startup <span>Success</span></h1>
        <p>Enter your startup metrics and get an ML-powered prediction
instantly.</p>
        <a href="/predict" class="btn-primary">Start Prediction</a>
    </section>
```

```
</body>
</html>
```

### Activity 2: Building predict.html (Input Form)

The predict page contains the input form where users enter their startup's metrics. All 9 model features have dedicated input fields with labels and helper hints. When the user clicks 'Predict My Startup's Future', the form submits a POST request to /predict in app.py.

IMPORTANT: Each input field's name attribute MUST exactly match the key used in request.form[] inside app.py. The form uses method='POST' and action='/predict'. All inputs are type='number' so only numeric values can be entered.

**The 9 Input Fields (matching model features):**

- age_first_funding_year — Years after founding when first funding was received
- age_last_funding_year — Years after founding when last funding was received
- age_first_milestone_year — Years after founding when first milestone was achieved
- age_last_milestone_year — Years after founding when last milestone was achieved
- relationships — Total number of business relationships
- funding_rounds — Number of funding rounds completed
- funding_total_usd — Total funding in US Dollars
- milestones — Total number of milestones achieved
- avg_participants — Average investors per funding round

**predict.html — Form Structure (Key Code):**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Predict - Prosperity Prognosticator</title>
    <style>
        body { font-family: 'Segoe UI', sans-serif;
               background: linear-gradient(135deg,#0d1b2a,#1b2a4a);
               min-height:100vh; color:#fff; margin:0; }
        .form-card {
            max-width:820px; margin:40px auto;
            background: rgba(255,255,255,0.05);
            border: 1px solid rgba(255,255,255,0.12);
            border-radius:24px; padding:48px;
        }
        .form-grid { display:grid; grid-template-columns:1fr 1fr;
gap:22px; }
        label { font-size:13px; color:#9aafc4; display:block; margin-
bottom:8px; }
        input {
            width:100%; background:rgba(255,255,255,0.07);
            border:1px solid rgba(255,255,255,0.15);
            border-radius:12px; padding:14px 18px;
            color:#fff; font-size:15px;
        }
        .submit-btn {
            background: linear-gradient(135deg,#ffa600,#ff7b00);
            color:#0d1b2a; padding:16px 60px;
            border-radius:30px; border:none;
            font-size:16px; font-weight:700; cursor:pointer;
        }
    </style>
</head>
<body>
    <div class="form-card">
        <h1>Startup Success Predictor</h1>
        <form action="/predict" method="POST">
            <div class="form-grid">
                <div>
                    <label>Age at First Funding (years)</label>
                    <input type="number" name="age_first_funding_year"
                           step="0.1" placeholder="e.g. 1.5" required>
                </div>
                <div>
                    <label>Age at Last Funding (years)</label>
                    <input type="number" name="age_last_funding_year"
                           step="0.1" placeholder="e.g. 4.2" required>
                </div>
                <div>
                    <label>Age at First Milestone (years)</label>
                    <input type="number" name="age_first_milestone_year"
                           step="0.1" placeholder="e.g. 0.5" required>
                </div>
                <div>
                    <label>Age at Last Milestone (years)</label>
                    <input type="number" name="age_last_milestone_year"
                           step="0.1" placeholder="e.g. 3.0" required>
                </div>
                <div>
                    <label>Number of Relationships</label>
                    <input type="number" name="relationships"
                           step="1" placeholder="e.g. 5" required>
                </div>
                <div>
                    <label>Funding Rounds</label>
                    <input type="number" name="funding_rounds"
                           step="1" placeholder="e.g. 3" required>
                </div>
                <div>
                    <label>Total Funding (USD)</label>
                    <input type="number" name="funding_total_usd"
                           step="1000" placeholder="e.g. 500000" required>
                </div>
                <div>
                    <label>Total Milestones</label>
                    <input type="number" name="milestones"
                           step="1" placeholder="e.g. 4" required>
                </div>
                <div>
                    <label>Avg. Participants per Round</label>
                    <input type="number" name="avg_participants"
                           step="0.1" placeholder="e.g. 2.5" required>
                </div>
            </div>
            <div style="text-align:center; margin-top:30px;">
                <button type="submit" class="submit-btn">
                    Predict My Startup's Future
                </button>
            </div>
        </form>
    </div>
</body>
</html>
```

### Activity 3: Building submit.html (Result Page)

The submit/result page displays the model's prediction. It uses Flask's Jinja2 templating with an if/else block to show different content depending on whether the prediction is 'Acquired' (green success state) or 'Closed' (red warning state). The result is passed as the variable prediction_text from app.py.

Jinja2 templating: Flask uses Jinja2 to embed Python logic inside HTML. {{ prediction_text }} displays the variable value. {% if prediction_text == 'Acquired' %} checks the condition. {% else %} handles the other case. {% endif %} closes the block.

**submit.html — Key Code (Jinja2 template logic):**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Prediction Result</title>
    <style>
        body { font-family:'Segoe UI',sans-serif;
               background:linear-gradient(135deg,#0d1b2a,#1b2a4a);
               min-height:100vh; color:#fff;
               display:flex; align-items:center;
               justify-content:center; margin:0; }
        .result-card {
            background:rgba(255,255,255,0.05);
            border:1px solid rgba(255,255,255,0.12);
            border-radius:28px; padding:60px;
            max-width:680px; text-align:center;
        }
        .icon-acquired { font-size:80px; color:#00c851; }
        .icon-closed   { font-size:80px; color:#ff4444; }
        .btn { padding:14px 38px; border-radius:28px;
               text-decoration:none; font-weight:700; }
        .btn-green { background:linear-gradient(135deg,#00c851,#007e33);
                     color:#fff; }
        .btn-red   { background:linear-gradient(135deg,#ff4444,#cc0000);
```

```
                     color:#fff; }
        .btn-back  { background:rgba(255,255,255,0.08); color:#fff;
                     border:1px solid rgba(255,255,255,0.2); }
    </style>
</head>
<body>
    <div class="result-card">

        {% if prediction_text == 'Acquired' %}
            <div class="icon-acquired"> </div>
            <h2 style="color:#00c851; margin:20px 0;">
                Your Startup is Likely to be Acquired!
            </h2>
            <p>Strong indicators of success detected.</p>
            <p><strong>Prediction: ACQUIRED (Status = 1)</strong></p>
            <br>
            <a href="/predict" class="btn btn-green">Predict Another</a>
            <a href="/" class="btn btn-back">Home</a>

        {% else %}
            <div class="icon-closed"> </div>
            <h2 style="color:#ff4444; margin:20px 0;">
                Your Startup is at Risk of Closing
            </h2>
            <p>Consider improving funding and milestone metrics.</p>
            <p><strong>Prediction: CLOSED (Status = 0)</strong></p>
            <br>
            <a href="/predict" class="btn btn-red">Try Again</a>
            <a href="/" class="btn btn-back">Home</a>
        {% endif %}

    </div>
</body>
</html>
```

## Activity 4: Building app.py (Flask Server)

app.py is the heart of the web application. It starts the Flask server, defines URL routes, loads the trained model, receives form data from predict.html, runs the model prediction, and sends the result to submit.html.

Flask routing: @app.route('/') connects a URL path to a Python function. When a user visits http://127.0.0.1:5000/, Flask calls the home() function. When they submit the form (POST to /predict), Flask calls the predict() function.

## Step 1 — Import Libraries and Load Model:

**Code:**

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

# Initialize Flask application
app = Flask(__name__)

# Load the saved Random Forest model
# This file must exist in the same folder as app.py
model = joblib.load('random_forest_model.pkl')
```

## Step 2 — Home Page Route:

**Code:**

```
@app.route('/')
def home():
    """Render the home/landing page."""
    return render_template('home.html')
```

## Step 3 — Predict Route (GET + POST):

**Code:**

```
@app.route('/predict', methods=['GET', 'POST'])
def predict():
    if request.method == 'POST':
        # Retrieve all 9 form values in the correct order
        features = [
            float(request.form['age_first_funding_year']),
            float(request.form['age_last_funding_year']),
            float(request.form['age_first_milestone_year']),
            float(request.form['age_last_milestone_year']),
            float(request.form['relationships']),
            float(request.form['funding_rounds']),
            float(request.form['funding_total_usd']),
            float(request.form['milestones']),
            float(request.form['avg_participants']),
        ]
        # Reshape for model input (1 row, 9 features)
        final_input = np.array(features).reshape(1, -1)
        # Predict: 1 = Acquired, 0 = Closed
        prediction = model.predict(final_input)
        result = 'Acquired' if prediction[0] == 1 else 'Closed'
        return render_template('submit.html', prediction_text=result)
    # GET request: show the empty form
    return render_template('predict.html')
```

### Step 4 — Main Function:

**Code:**

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

## Activity 5: How to Run the Application

Once all files are created and the model is saved, follow these steps exactly to launch the web application and access it in your browser.

Before running: Make sure random_forest_model.pkl is in the SAME folder as app.py. If it is not there, re-run Milestone 7 (the notebook) to generate it first.

### Step 1 — Open Anaconda Prompt as Administrator:

- Click the Windows Start Menu
- Search for 'Anaconda Prompt'
- Right-click it and select 'Run as Administrator'

### Step 2 — Navigate to Project Folder:

**Command:**

```
# Replace the path below with your actual project folder path
cd C:\Users\YourName\ProsperityPrognosticator

# Verify you are in the right folder
dir
```

### Step 3 — Install Flask (if not already installed):

**Command:**

```
pip install flask
pip install joblib
```

## Step 4 — Run the Flask Application:

**Command:**

```
python app.py
```

After running this command, you should see output similar to:

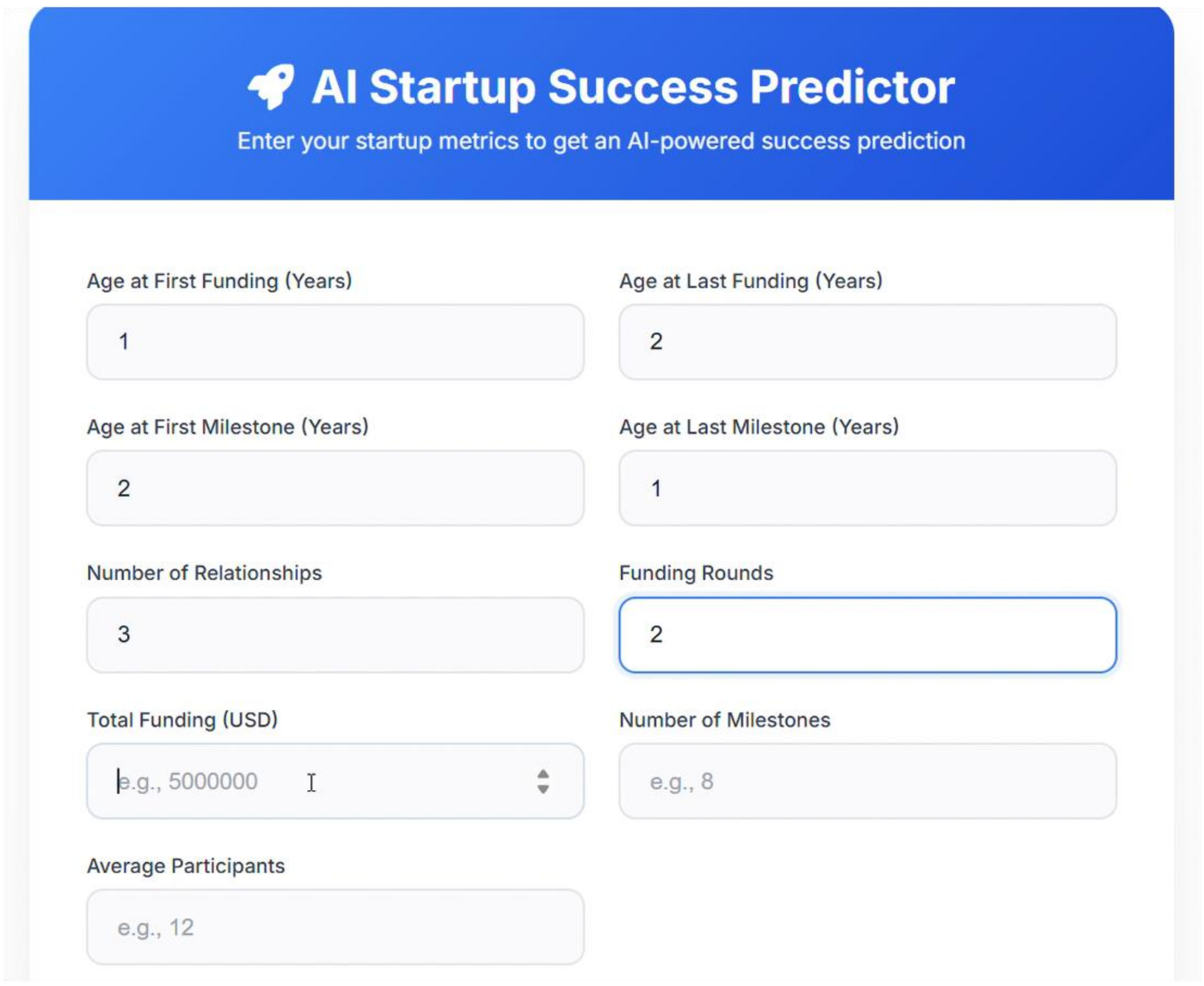
```
 * Serving Flask app 'app'
 * Debug mode: on
 * Running on http://127.0.0.1:5000
 * Press CTRL+C to quit
```

## Step 5 — Open the Application in Browser:

- Open Google Chrome, Firefox, or any browser
- In the address bar, type exactly: http://127.0.0.1:5000
- Press Enter — the home page should load
- Click 'Predict Now' to go to the prediction form
- Fill in all 9 fields and click 'Predict My Startup's Future'
- The result page shows 'Acquired' or 'Closed' with a visual indicator

## Step 6 — Stop the Application:

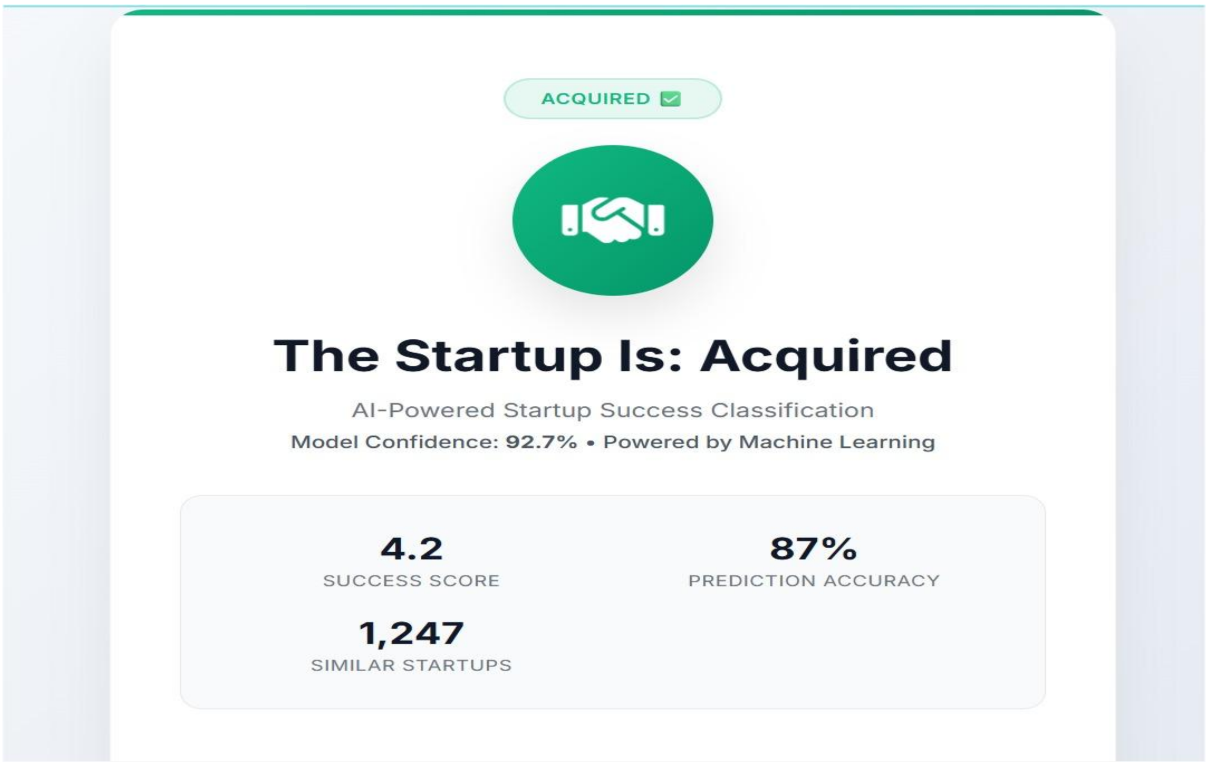
- Go back to the Anaconda Prompt window
- Press Ctrl + C on the keyboard to stop the Flask server
- The prompt will return — the app is now stopped



# AI Startup Success Predictor

Enter your startup metrics to get an AI-powered success prediction

| | |
|---|---|
| Age at First Funding (Years) | Age at Last Funding (Years) |
| 1 | 2 |
| Age at First Milestone (Years) | Age at Last Milestone (Years) |
| 2 | 1 |
| Number of Relationships | Funding Rounds |
| 3 | 2 |
| Total Funding (USD) | Number of Milestones |
| e.g., 5000000 | e.g., 8 |
| Average Participants | |
| e.g., 12 | |



ACQUIRED ✓

## The Startup Is: Acquired

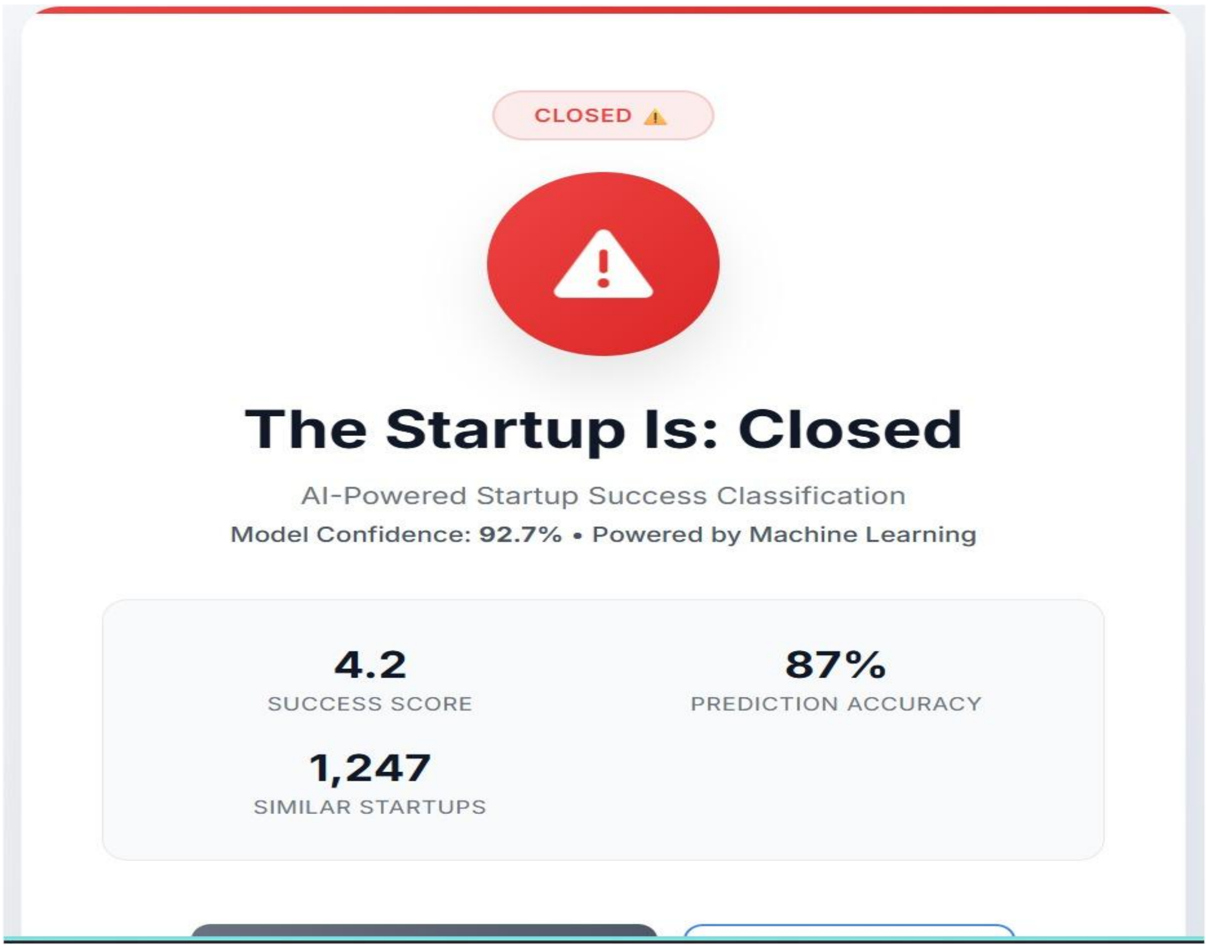
AI-Powered Startup Success Classification

Model Confidence: 92.7% • Powered by Machine Learning

4.2
SUCCESS SCORE

87%
PREDICTION ACCURACY

1,247
SIMILAR STARTUPS


CLOSED ⚠

The Startup Is: Closed

AI-Powered Startup Success Classification

Model Confidence: 92.7% • Powered by Machine Learning

4.2
SUCCESS SCORE

87%
PREDICTION ACCURACY

1,247
SIMILAR STARTUPS

## Project Summary

The table below summarises all milestones, key activities, and outputs produced across the entire Prosperity Prognosticator project:

| Milestone | Key Activities | Output |
|---|---|---|
| **1 – Data Collection** | Download startup1.csv, import libraries, load dataset, inspect head/tail/info | Raw DataFrame loaded |
| **2 – Data Information** | State grouping pie chart, separate numeric/categorical, convert status, drop labels | Cleaned target variable |

| | | |
|---|---|---|
| **3 – Visualisation** | Statistical describe(), value counts, missing values, correlation heatmap, scatter & box plots | EDA complete |
| **4 – Univariate Analysis** | Class counts, pie chart (acquired vs closed), countplot per category | Class distribution understood |
| **5 – Bivariate Analysis** | Success rate per category, total funding per category, startup count per state | Feature-outcome insights |
| **6 – Pre-processing** | Remove duplicates, drop negative values, outlier analysis, drop irrelevant columns, train-test split | Clean X_train / X_test |
| **7 – Model Building** | Random Forest training, confusion matrix, classification report, ROC-AUC, Precision-Recall, save model | random_forest_model.pkl |