Prosperity Prognosticator: Startup Success Prediction Using Machine Learning

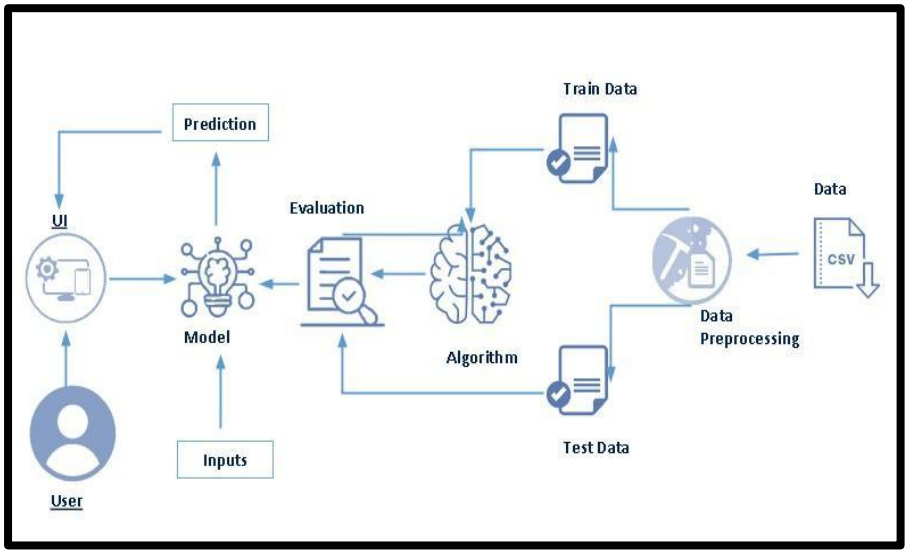
Project Description:

Startups play a crucial role in economic growth and innovation. However, not all startups succeed—many fail due to insufficient funding, lack of milestones, weak networks, or poor strategic decisions. Predicting the success or failure of startups at an early stage is a challenging but important task for investors, entrepreneurs, and financial institutions.

This project, Prosperity Prognosticator, aims to predict whether a startup will be Acquired (Successful) or Closed (Unsuccessful) using Machine Learning techniques. By analyzing funding history, milestones, relationships, and participation metrics, the model helps in understanding the factors influencing startup success.

Machine learning classification algorithms are used to train and test the dataset. Among them, Random Forest Classifier is selected due to its high accuracy and robustness. The trained model is deployed using Flask, allowing users to input startup details through a web interface and receive predictions with confidence scores.

Technical Architecture:



Pre requisites:

To complete this project, you must required following software's, concepts and packages

- **Anaconda navigator and Visual Studio Code:**

- Refer the link below to download anaconda navigator
- Link : https://youtu.be/1ra4zH2G4o0
- Refer the link below to download Visual Studio Code
- Link: https://code.visualstudio.com/

- **Python packages:**
  - Open VS Code terminal or Command Prompt as administrator
  - Type “pip install numpy” and click enter.
  - Type “pip install pandas” and click enter.
  - Type “pip install scikit-learn” and click enter.
  - Type ”pip install matplotlib” and click enter.
  - Type ”pip install scipy” and click enter.
  - Type ”pip install joblib” and click enter.
  - Type ”pip install seaborn” and click enter.
  - Type “pip install Flask” and click enter.

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 : https://en.wikipedia.org/wiki/Regression_analysis
  - Classification: https://en.wikipedia.org/wiki/Statistical_classification
  - Decision tree: https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
  - Random forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm
  - 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:

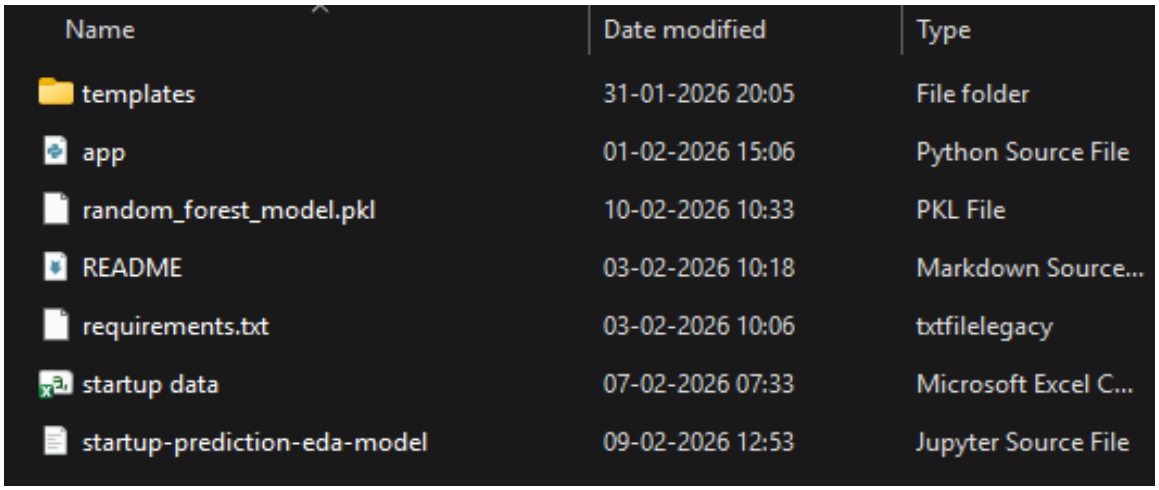
- User interacts with the UI to enter the input.
- Entered input is analyzed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

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



- The data obtained is in csv files,for training and testing.
- We are building a Flask application which will require the html files to be stored in the templates folder.
- app.py file is used for routing purposes using scripting.
- random_forest_model.pkl is the saved model. This will further be used in the Flask integration.

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, UCI repository, etc.

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

Link: https://www.kaggle.com/datasets/manishkc06/startup-success-prediction

Milestone 2: Visualizing and analysing the data

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.

```
In [1]: 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.model_selection import GridSearchCV
        from sklearn.metrics import classification_report
```

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.

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

```
data.head()
```

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

Activity 3: Univariate analysis

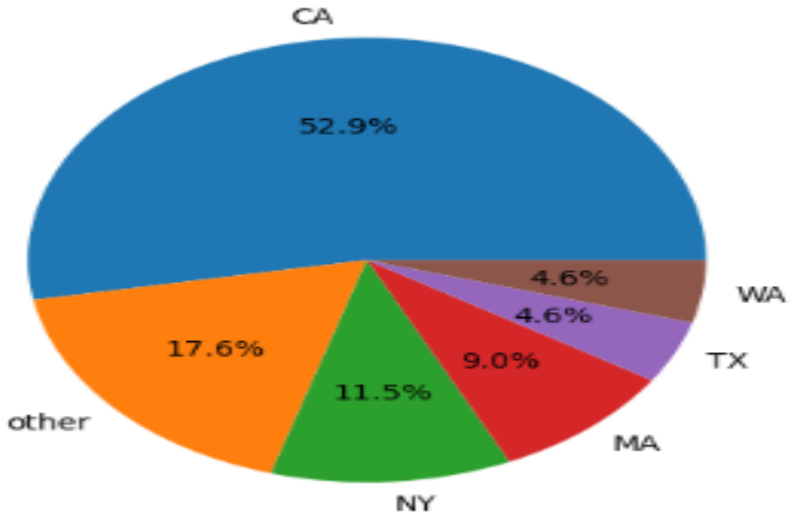
Univariate analysis is the process of analyzing **individual features** in the dataset to understand their distribution, frequency, and overall behavior.In this activity, each variable is analyzed independently to gain insights into startup characteristics.

- 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.

```
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'
```

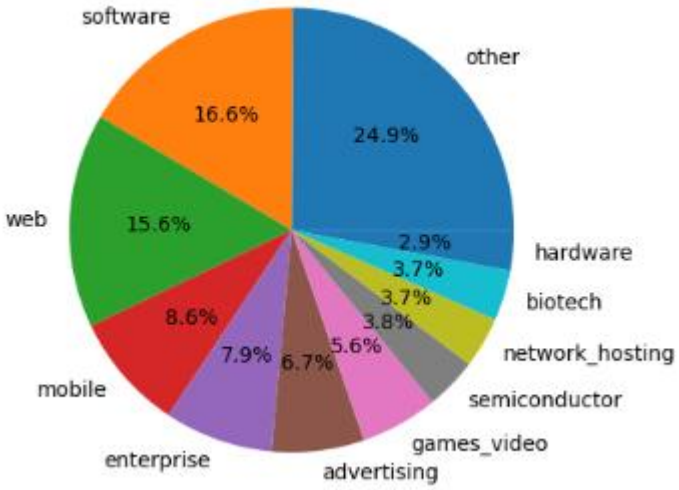
```
state_count = data['State'].value_counts()
plt.pie(state_count, labels = state_count.index, autopct = '%1.1f%%')
plt.show()
```



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' DataFrame. 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.

```
data['category'] = 'other'
data.loc[(data['category_code'] == 'software'), 'category'] = 'software'
data.loc[(data['category_code'] == 'web'), 'category'] = 'web'
data.loc[(data['category_code'] == 'mobile'), 'category'] = 'mobile'
data.loc[(data['category_code'] == 'enterprise'), 'category'] = 'enterprise'
data.loc[(data['category_code'] == 'advertising'), 'category'] = 'advertising'
data.loc[(data['category_code'] == 'games_video'), 'category'] = 'games_video'
data.loc[(data['category_code'] == 'semiconductor'), 'category'] = 'semiconductor'
data.loc[(data['category_code'] == 'network_hosting'), 'category'] = 'network_hosting'
data.loc[(data['category_code'] == 'biotech'), 'category'] = 'biotech'
data.loc[(data['category_code'] == 'hardware'), 'category'] = 'hardware'
```

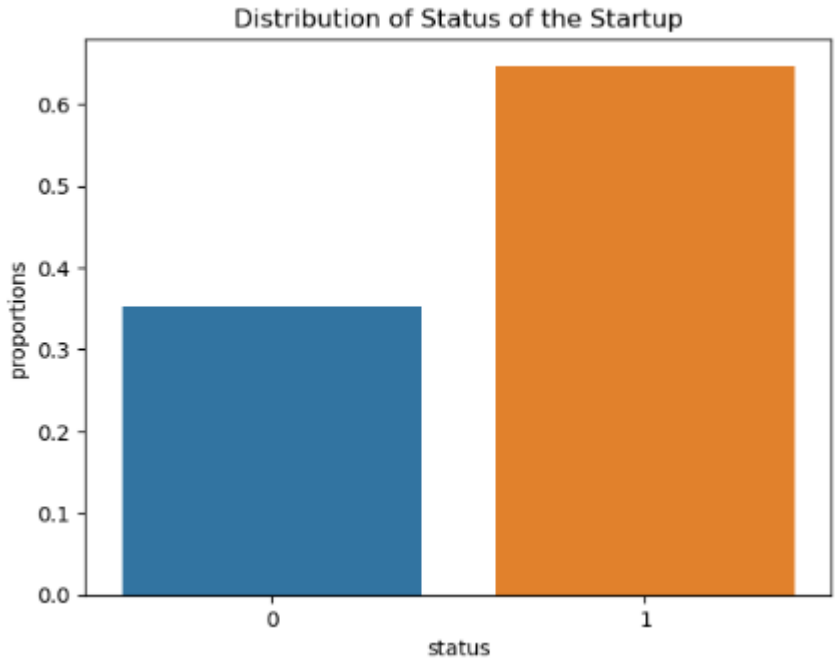
```
category_count = data['category'].value_counts()
plt.pie(category_count, labels = category_count.index, autopct = '%1.1f%%')
plt.show()
```



Startup Status Distribution: The overall distribution of startup outcomes is analyzed 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.

```
prop_df = data.groupby('status').size().reset_index(name = 'counts')
prop_df['proportions'] = prop_df['counts']/prop_df['counts'].sum()
```

```
sns.barplot(data = prop_df, x = 'status', y = 'proportions')
plt.title('Distribution of Status of the Startup')
```



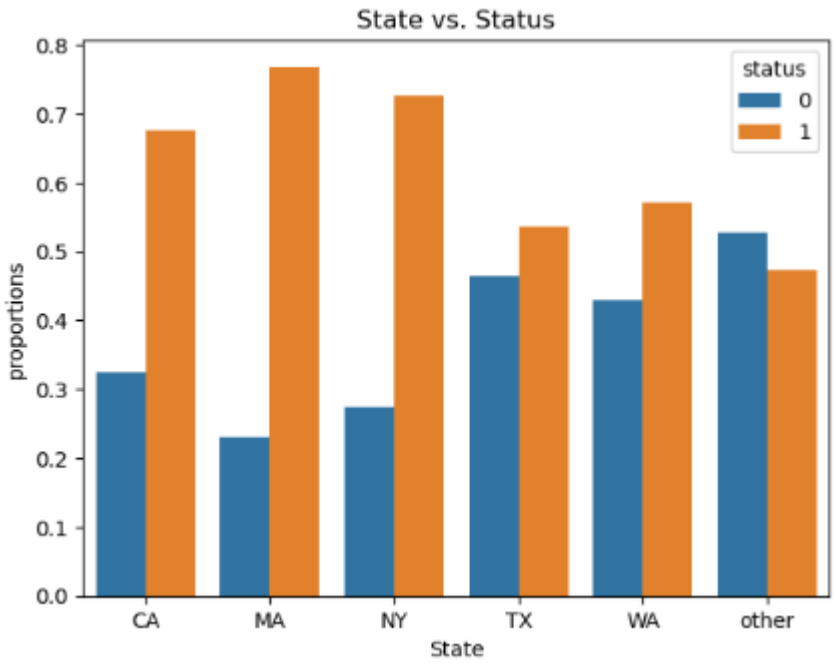
Activity 4: Bivariate analysis

Bivariate analysis is the process of analyzing the relationship between two variables to identify patterns and dependencies. In this project, bivariate analysis is performed to study how state and category influence the startup status.

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.

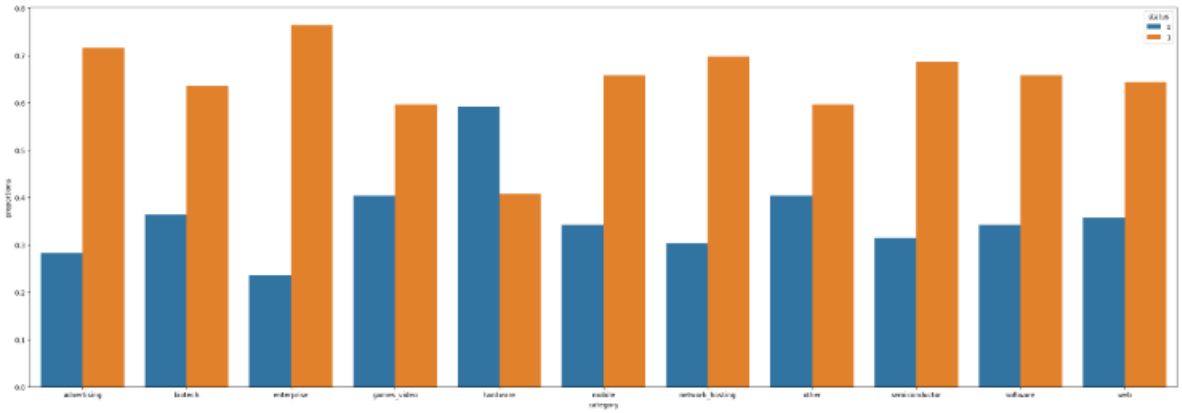
```
prop_df = data.groupby(['State','status'],group_keys = True).size().reset_index(name='count')
prop_df['proportions'] = prop_df.groupby('State')['count'].apply(lambda x: x/x.sum())
sns.barplot(data = prop_df, x = 'State', y = 'proportions',hue = 'status')
plt.title('State vs. Status')
```



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..

```
fig, ax = plt.subplots(figsize = (30,10))
prop_df = data.groupby(['category','status']).size().reset_index(name='counts')
prop_df['proportions'] = prop_df.groupby('category')['counts'].apply(lambda x: x/float(x.sum()))
sns.barplot(data = prop_df, x = 'category', y = 'proportions',hue = 'status')
```



Activity 5: Multivariate analysis

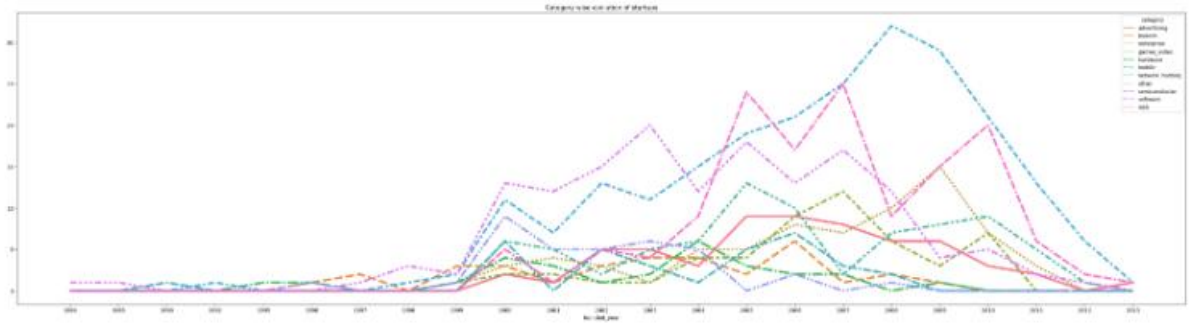
In simple words, multivariate analysis is to find the relation between multiple features. In this project, multivariate analysis is performed to study how different startup attributes such as **category, funding, milestones, and year of establishment** collectively influence startup behavior.

Category Evolution by Year

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

```
cat_year = pd.crosstab(index = data['founded_year'], columns = data['category'])
```

```
fig, ax = plt.subplots(figsize=(40,10))
sns.lineplot(data = cat_year, lw = 4)
plt.title('Category wise evolution of startups')
```

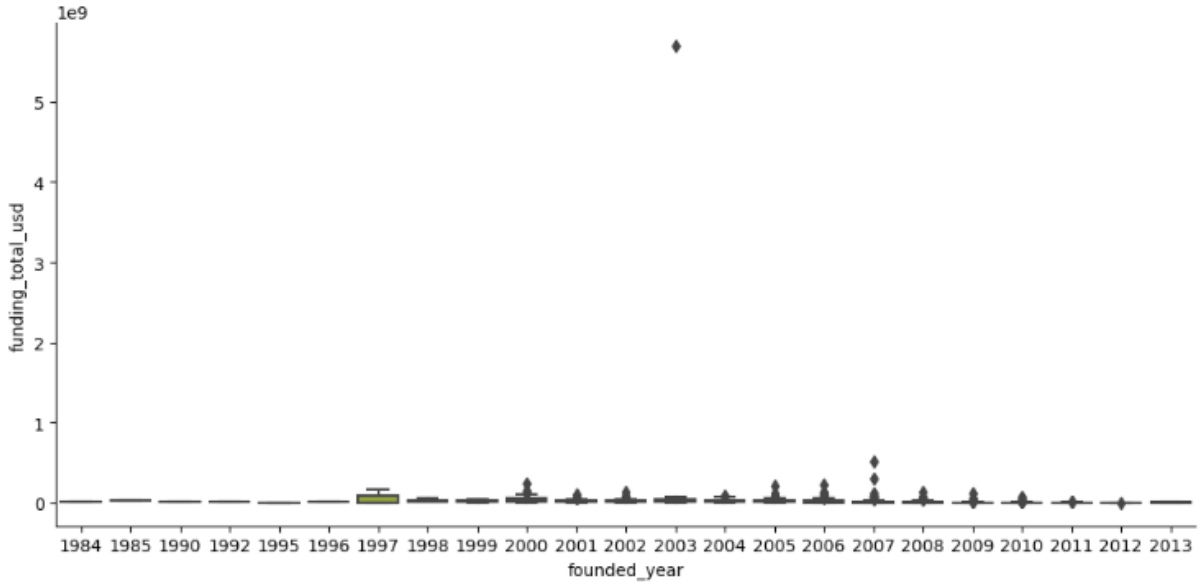


From the visualization, it can be observed that **software and web-based startups** show consistent growth over the years. Other categories show fluctuations, indicating varying levels of interest and investment trends over time.

Funding Distribution by Founded Year

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

```
sns.catplot(data=data, x="founded_year", y="funding_total_usd",
            kind="box",height=5, aspect=2,order = ['1984','1985','1990','1992','1995',
                                                   '1996','1997','1998','1999','2000',
                                                   '2001','2002','2003','2004','2005',
                                                   '2006','2007','2008','2009','2010',
                                                   '2011','2012','2013'])
```



Activity 6: Descriptive analysis

Descriptive analysis is used to summarize the **basic statistical characteristics** of the dataset. It helps in understanding the **central tendency, spread, and range** of numerical features and the overall structure of the data before preprocessing and model building.

```
data.describe()
```
✓ 0.3s

	Unnamed: 0	latitude	longitude	labels	founded_at	age_first_funding_year	age_last_funding_year	age_first_
count	923.000000	923.000000	923.000000	923.000000	869	923.000000	923.000000	
mean	572.297941	38.517442	-103.539212	0.646804	2005-07-25 19:39:50.333716992	2.235630	3.931456	
min	1.000000	25.752358	-122.756956	0.000000	1984-01-01 00:00:00	-9.046600	-9.046600	
25%	283.500000	37.388869	-122.198732	0.000000	2003-01-01 00:00:00	0.576700	1.669850	
50%	577.000000	37.779281	-118.374037	1.000000	2006-01-01 00:00:00	1.446600	3.528800	
75%	866.500000	40.730646	-77.214731	1.000000	2008-01-01 00:00:00	3.575350	5.560250	
max	1153.000000	59.335232	18.057121	1.000000	2013-02-05 00:00:00	21.895900	21.895900	
std	333.585431	3.741497	22.394167	0.478222	NaN	2.510449	2.967910	

Milestone 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

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Activity 1: Reducing the Number of Categories

In the dataset, the state_code variable contains multiple categories representing different states. Initially, a comparison is performed between the state_code and state_code.1 columns to verify whether both columns contain the same data.

During the comparison, it is observed that one row contains a missing value in the state_code.1 column. Therefore, the state_code column is considered the correct and reliable column for further analysis.
After analyzing the frequency distribution of the state_code variable, it is found that the top

five states — **CA, NY, MA, TX, and WA** — cover more than 80% of the total dataset. Hence, to simplify the model and reduce complexity, all remaining states are grouped under the category **"Other"**.

```
print(data['state_code'].equals(data['state_code.1']))
```

```
False
```

```
df = data.loc[data['state_code'] != data['state_code.1']]
df.style.set_properties(**{'background-color': 'yellow'}, subset=['state_code','state_code.1'])
```

	state_code	city	labels	founded_at	closed_at	first_funding_at	last_funding_at	age_first_funding_year	age_last_funding_year
515	CA	Menlo Park	0	1/1/2005	9/1/2010	3/1/2007	4/15/2008	2.161600	3.287700

```
state = data['state_code'].value_counts().to_frame()
state['proportion'] = state['state_code']/sum(state['state_code'])*100
state
```

Activity 2: Dropping the Irrelevant Columns

Dropping irrelevant columns reduces unnecessary data, improves computational efficiency, and ensures that the model focuses only on meaningful features.

```
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)
```

Activity 3: Target Variable Encoding

The target variable status is converted into numerical format:

- **Acquired → 1**
- **Closed → 0**

This conversion is necessary because machine learning algorithms require numerical input.

- Encoding the target variable makes the dataset compatible with classification algorithms.

Activity 4: Feature Selection

Important numerical features such as funding details, milestones, relationships, and participation metrics are selected for model training. Irrelevant and redundant columns are excluded.

Activity 5: Splitting data into train and test

The dataset is divided into input features (X) and the target variable (y). The target variable 'status' is assigned to y, while the selected numerical features are assigned to X. The data is then split into training and testing sets using train_test_split(), with 70% used for training and 30% used for testing.

```
#split train and test set
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size =0.3, random_state = 0)
```

Milestone 4: Model Building

After completing data preprocessing, the next step is to build a machine learning model to predict startup success. Since the target variable consists of two categories — **Acquired** and **Closed** — this problem is treated as a **binary classification problem**.

In this project, the **Random Forest Classifier** algorithm is used to build the prediction model because of its high accuracy and ability to handle structured data efficiently.

Activity 1: Random forest model

A function named randomForest is created and train and test data are passed as the parameters. Inside the function, RandomForestClassifier algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.

```
#rf = RandomForestClassifier()
param_grid = {'n_estimators':[100,200,300],
              'max_depth':[10,20,30],
              'min_samples_split':[2,4,6],
              'min_samples_leaf':[1,2,3],
              'bootstrap':[True,False]}

grid_search = GridSearchCV(estimator=rf, param_grid = param_grid, cv = 5, n_jobs = -1, verbose = False)
grid_search.fit(x_train, y_train)

print('Best parameters:', grid_search.best_params_)
```

```
Best parameters: {'bootstrap': False, 'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 100}
```

Activity2: Model Training

The dataset is divided into training and testing sets.

- 70% of the data is used for training
- 30% of the data is used for testing

The training data is fed into the Random Forest model using the .fit() method. During this process, the algorithm learns patterns and relationships between startup features and their corresponding outcomes.

Activity3: Model Prediction

After training, the model predicts startup outcomes on the testing dataset using the .predict() method. These predictions are compared with actual values to evaluate model performance.

Activity 4: Model Evaluation

The performance of the model is evaluated using the following metrics:

Accuracy Score

Accuracy measures the percentage of correctly predicted instances out of the total predictions.

```
#applying Random forest classifier
mode=RandomForestClassifier()
model.fit(x_train, y_train)
y_pred_test=model.predict(x_test)
y_pred_train=model.predict(x_train)
```

```
#checking accuracy
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
test_acc = accuracy_score(y_test,y_pred_test)
train_acc = accuracy_score(y_train,y_pred_train)
print('test_acc: ', test_acc)
print('train_acc: ', train_acc)
```

```
test_acc:  0.8
train_acc:  1.0
```

Classification Report

The classification report provides an evaluation of the model's performance. For class 0, the precision is 0.59, recall is 0.78, and F1-score is 0.67. For class 1, the precision is 0.92, recall is 0.82, and F1-score is 0.87. The overall accuracy of the model is 0.81. The macro average of precision, recall, and F1-score is 0.75, 0.80, and 0.77, respectively. The weighted average of precision, recall, and F1-score is 0.84, 0.81, and 0.82, respectively.

```
#rf = RandomForestClassifier(n_estimators=100,bootstrap=False,max_depth=20,min_samples_leaf=2, min_samples_split=2)
model_rf = rf.fit(x_train,y_train)
y_pred_rf = model_rf.predict(x_test)
cr_rf = classification_report(y_pred_rf, y_test)
print(cr_rf)
```

```
              precision    recall  f1-score   support

           0       0.59      0.78      0.67        68
           1       0.92      0.82      0.87       207

    accuracy                           0.81       275
   macro avg       0.75      0.80      0.77       275
weighted avg       0.84      0.81      0.82       275
```

Activity 5: Saving the model

After successful evaluation, the trained Random Forest model is saved in .pkl format using the joblib library. This allows the model to be reused without retraining and enables integration with the Flask web application.

.

```
# Save the trained model to a file
joblib.dump(model, 'random_forest_model.pkl')
```

```
['random_forest_model.pkl']
```

Milestone 5: Application Building

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building the Server-Side Script

Activity1: Building Html Pages:

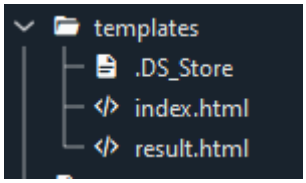
For this project, we create two HTML files namely

• Index.html

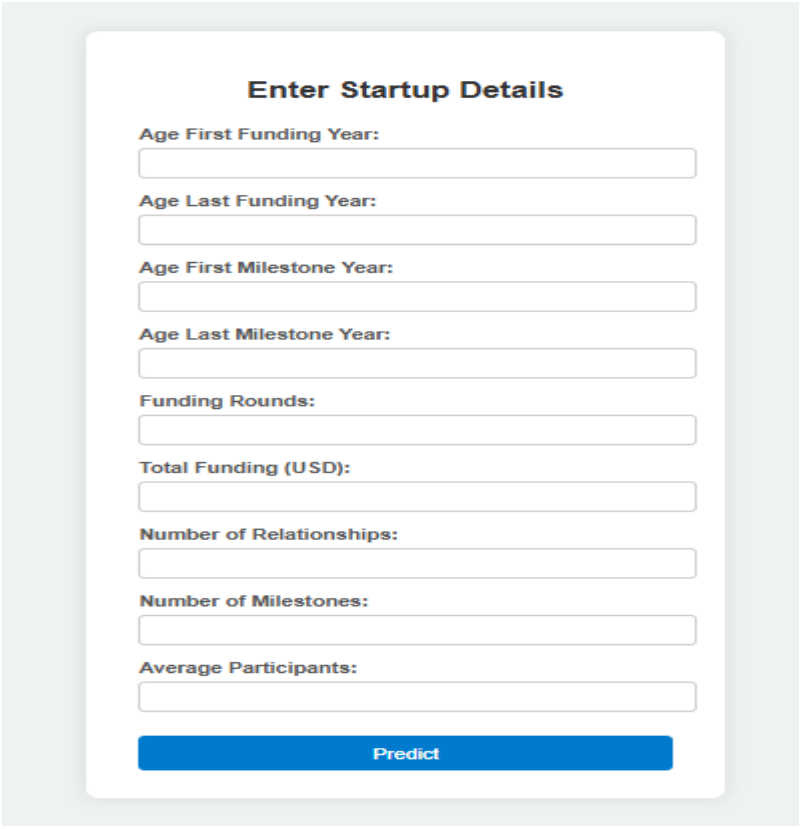
• Result.html

• home.html

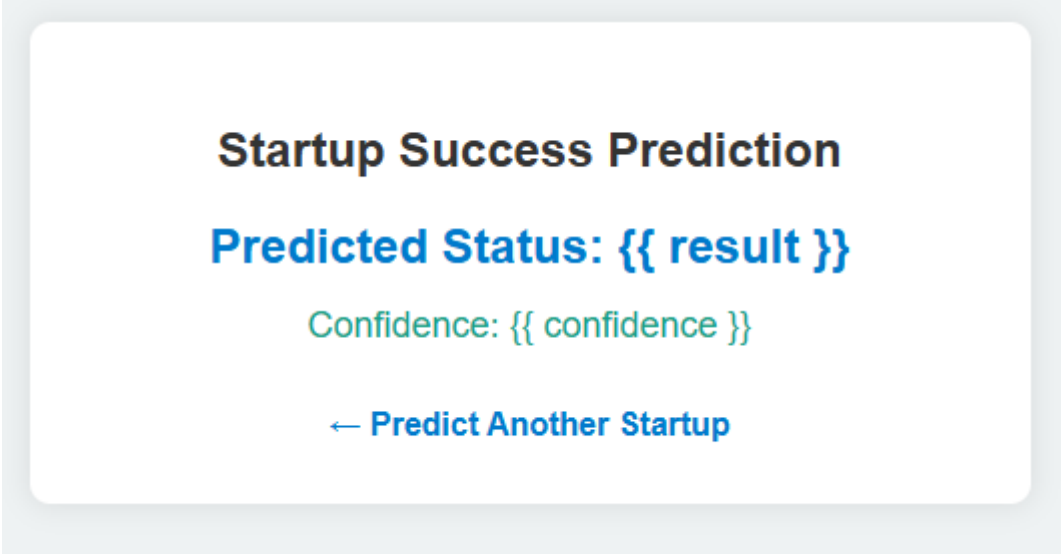
And we will save them in the templates folder.



Let’s see how our index.html page looks like:



Lets look how our result.html file looks like:



Activity 2: Build Python code:

Import the libraries

```
from flask import Flask, render_template, request
import joblib
```

Load the saved model. Importing flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument.

```
app = Flask(__name__)
model = joblib.load('random_forest_model.pkl')
```

Render HTML page:

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

Here we will be using declared constructor to route to the HTML page which we have created earlier.

In the above example, '/' URL is bound with home.html function. Hence, when the home page of the web server is opened in browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

Retrieves the value from UI:

```
@app.route('/predict', methods=['POST'])
def predict():
    # Get the input values from the form
    age_first_funding_year = float(request.form['age_first_funding_year'])
    age_last_funding_year = float(request.form['age_last_funding_year'])
    age_first_milestone_year = float(request.form['age_first_milestone_year'])
    age_last_milestone_year = float(request.form['age_last_milestone_year'])
    relationships = float(request.form['relationships'])
    funding_rounds = float(request.form['funding_rounds'])
    funding_total_usd = float(request.form['funding_total_usd'])
    milestones = float(request.form['milestones'])
    avg_participants = float(request.form['avg_participants'])
```

```
# Create a list with the input values
input_data = [
    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
]

# Make a prediction using the loaded model
prediction = model.predict([input_data])[0]

# Map the predicted label to a meaningful output
if prediction == 1:
    result = 'Acquired'
else:
    result = 'Closed'

# Render the prediction result
return render_template('result.html', result=result)
```

Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will rendered to the text that we have mentioned in the submit.html page earlier.

Main Function:

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

Activity 3: Run the application

When you run the “app.py” file this window will open in the console or output terminal. Copy the URL given in the form http://127.0.0.1:5000 and paste it into the browser.

```
PS C:\Users\gunna\OneDrive\Desktop\aiml\startup_pred> py app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 312-134-863
```

- When we paste this URL into the browser it will redirect us to home.html page,.