

Prosperity Prognosticator: Machine Learning for Startup

Success Prediction

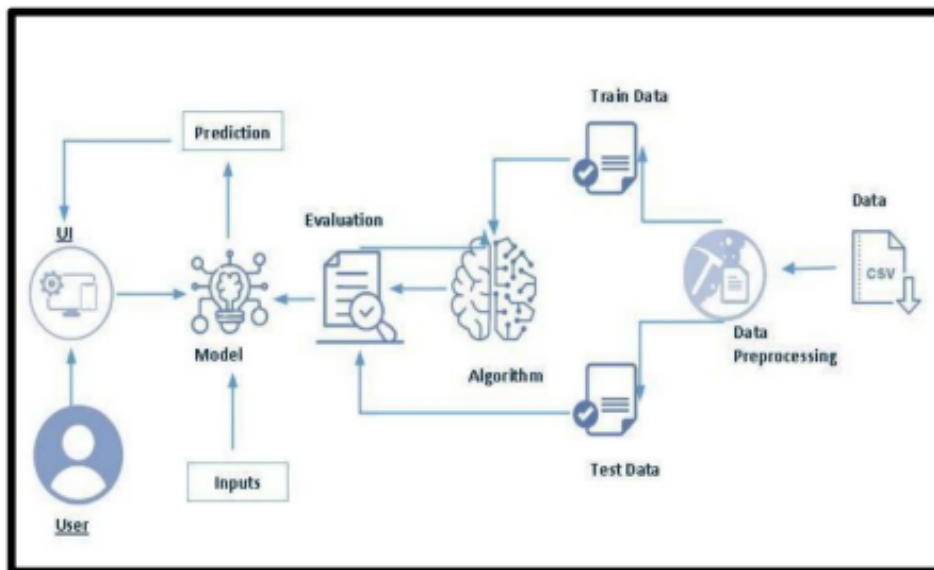
Develop a machine learning model for predicting the success of startups. By analyzing startup characteristics, market trends, and funding data, this project aims to provide insights into the potential success of new ventures, aiding investors, entrepreneurs, and policymakers in decision-making and resource allocation.

Scenario 1 (Investors): Utilize the prosperity prognosticator model to evaluate startup investment opportunities and assess the potential for returns. Improve investment decision-making, optimize portfolio management, and maximize investment profitability by identifying promising startups with high growth potential.

Scenario 2 (Entrepreneurs): Incorporate success prediction insights into business planning and strategy formulation for new ventures. Identify key success factors, mitigate risks, and enhance startup viability and sustainability through data-driven decision-making and resource allocation.

Scenario 3 (Policy Makers): Utilize machine learning predictions to inform entrepreneurship policies and support initiatives aimed at fostering startup growth and innovation. Identify factors influencing startup success, design targeted support programs, and stimulate economic development through the promotion of entrepreneurship and innovation ecosystems.

Technical Architecture:



Project Flow

- The user is shown the Home page. The user will browse through the Home page and go to predict my adaptivity and enter the specified engagement metrics.
- After clicking the Predict button the user will be directed to the Results page where the model will analyze the inputs given by the user and showcase the prediction of the Adaptivity level.
- To accomplish this we have to complete all the activities listed below:
 - Data Collection and Preparation
 - Collect the dataset
 - Data Preparation
 - Exploratory Data Analysis
 - Descriptive statistical
 - Visual Analysis
 - Model Building
 - Creating a function for evaluation
 - Training and testing the Models using multiple algorithms
 - Performance Testing & Hyperparameter Tuning
 - Testing model with multiple evaluation metrics
 - Comparing model accuracy before & after applying hyperparameter tuning
 - Comparing model accuracy for different numbers of features.
 - Building a model with appropriate features.

- Model Deployment
 - Save the best model
 - Integrate with Web Framework

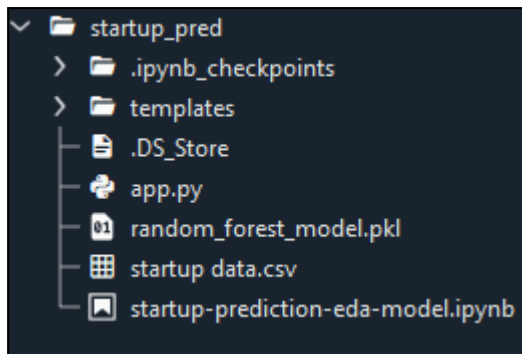
Prior Knowledge

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

- ML Concepts:
 - Supervised learning: <https://www.javatpoint.com/supervised-machine-learning>
 - Linear Regression: <https://www.javatpoint.com/linear-regression-in-machine-learning>
 - SVM: <https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm>
 - 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/>
 - Regularisation: <https://www.javatpoint.com/regularization-in-machine-learning>
- Flask Basics: https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Structure

Create 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 & Preparation

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

Activity 1: Collect the dataset

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

In this project we have used .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>

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

Note: There are a 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 1.1: Importing the libraries

Import the necessary libraries as shown in the image.

```
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 1.2: 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 the csv file.

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

data.head()

0.0s

Python

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	
0	1005	CA	42.358880	-71.056820	92101	c6669	San Diego	NaN	Bandsintown	1	1/1/2007	NaN	4/1/2009	1/1/2010	2.2493	3.0027
1	204	CA	37.238916	-121.973718	95032	c16283	Los Gatos	NaN	TriCipher	1	1/1/2000	NaN	2/14/2005	12/28/2009	5.1260	9.9973
2	1001	CA	32.901049	-117.192656	92121	c65620	San Diego	San Diego CA 92121	Plix	1	3/18/2009	NaN	3/30/2010	3/30/2010	1.0329	1.0329
3	738	CA	37.320309	-122.050040	95014	c42668	Cupertino	Cupertino CA 95014	Solidcore Systems	1	1/1/2002	NaN	2/17/2005	4/25/2007	3.1315	5.3151
4	1002	CA	37.779281	-122.419236	94105	c65806	San Francisco	San Francisco CA 94105	Inhale Digital	0	8/1/2010	10/1/2012	8/1/2010	4/1/2012	0.0000	1.6685

Activity 2: Data Preparation

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so

much randomness so we need to clean the dataset properly in order to fetch good results. This

activity includes the following steps.

- Handling missing values
- Handling Outliers

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 2.1: Handling missing values

For checking the null values, `df.isna().any()` function is used. To sum those null values

We use `.sum()` function.

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

✓ 0.2s

	null_count	% 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

Here we got some missing values in the columns like unnamed6,closed-at,age_first_milestone_year,age_last_milestone_year,and state_code.1

Now we have to handle the missing values

For the unnamed 6 we combine the zip_code,city,and state_code to fill the column and made it free of missing values

```
data['Unnamed: 6']=data.apply(lambda row: (row.city)+" "+(row.state_code)+" "+(row.zip_code),axis=1)
data.head()
```

✓ 0.0s

	Unnamed: 0	state_code	latitude	longitude	zip_code	id	city	Unnamed: 6	name
0	1005	CA	42.358880	-71.056820	92101	c6669	San Diego	San Diego CA 92101	Bandsintown
1	204	CA	37.238916	-121.973718	95032	c16283	Los Gatos	Los Gatos CA 95032	TriCipher
2	1001	CA	32.901049	-117.192656	92121	c65620	San Diego	San Diego CA 92121	Plixi
3	738	CA	37.320309	-122.050040	95014	c42668	Cupertino	Cupertino CA 95014	Solidcore Systems
4	1002	CA	37.779281	-122.419236	94105	c65806	San Francisco	San Francisco CA 94105	Inhale Digital

The above code is for achieve it

For the closed_at column we fill them with a random date.

```
data['closed_at']=data['closed_at'].fillna(value="31/12/2013")
```

For the columns like age_first_milestone_year and age_last_milestone_year
We fill the columns with a value of zero

```
data['age_first_milestone_year']=data['age_first_milestone_year'].fillna(value="0")  
data['age_last_milestone_year']=data['age_last_milestone_year'].fillna(value="0")
```

Here is the code to fill the above columns

And finally

We check and identify rows where 'state_code' and 'state_code.1' don't match — useful for spotting data inconsistencies.

Later we drop the state_code.1 column

```
data.drop(['state_code.1'],axis=1,inplace=True)
```

By doing all the above things we came to a missing value free data.

Activity 2.2: Handling Negative values

Before that we have to change the type of column to as float datatype

```
data['age_first_milestone_year']=data.age_first_milestone_year.astype(float)  
data['age_last_milestone_year']=data.age_last_milestone_year.astype(float)
```

Now check for the negative values

```
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],len(data[data[age[a]]<0])))
```

```
is there any negative value in 'age_first_funding_year' column : 46  
is there any negative value in 'age_last_funding_year' column : 13  
is there any negative value in 'age_first_milestone_year' column : 46  
is there any negative value in 'age_last_milestone_year' column : 12
```

We got negative values in the above columns. we clean the data by dropping rows with invalid (negative) age values in funding or milestone years.

```
data=data.drop(data[data.age_first_funding_year<0].index)
data=data.drop(data[data.age_last_funding_year<0].index)
data=data.drop(data[data.age_first_milestone_year<0].index)
data=data.drop(data[data.age_last_milestone_year<0].index)
```

After we again check for negative values were present in the data

```
✓ for a in range(len(age)):
    | print("is there any negative value in '{}' column : {}".format(age[a],len(data[data[age[a]]<0])))
✓ 0.0s
```

```
is there any negative value in 'age_first_funding_year' column : 0
is there any negative value in 'age_last_funding_year' column : 0
is there any negative value in 'age_first_milestone_year' column : 0
is there any negative value in 'age_last_milestone_year' column : 0
```

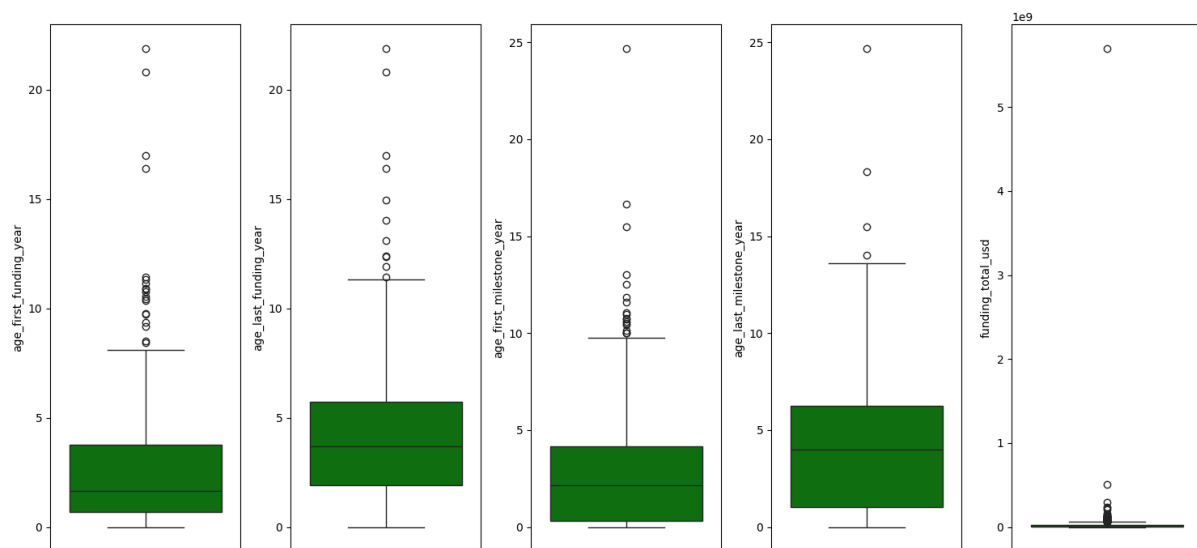
Activity 2.3: Handling Outliers

With the help of boxplot, outliers are visualized. And here we are going to find upper bound

and lower bound of

age_first_funding_year,age_last_funding_year,age_first_milestone_year,age_last_milestone_year,and total funding in usd feature with some mathematical formula.

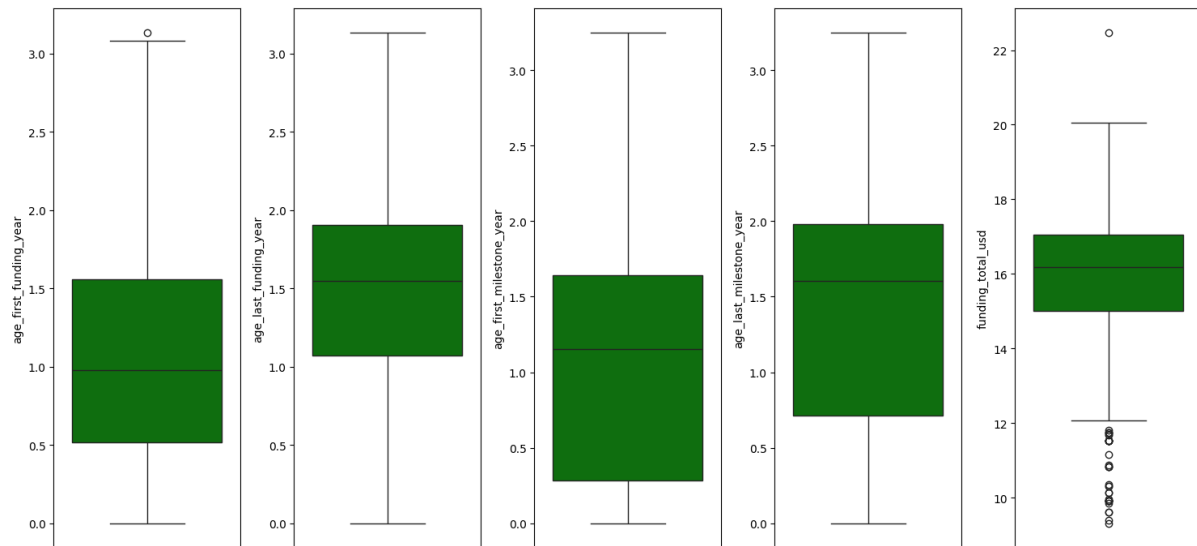
· From the below diagram, we could visualize that



age_first_funding_year,age_last_funding_year,age_first_milestone_year,age_last_milestone_year,and total funding in usd feature has outliers. Boxplot from seaborn library is used here.

```
data["age_first_funding_year"] = np.log1p(data["age_first_funding_year"])
data["age_last_funding_year"] = np.log1p(data["age_last_funding_year"])
data["age_first_milestone_year"] = np.log1p(data["age_first_milestone_year"])
data["age_last_milestone_year"] = np.log1p(data["age_last_milestone_year"])
data["funding_total_usd"] = np.log1p(data["funding_total_usd"])
```


It reduces skewness and handles large ranges in the data, making it more suitable for modeling or visualization — especially for features like funding amounts and age metrics.



Milestone 2: Exploratory Data Analysis

Activity 1: Descriptive statistical

Descriptive analysis is to study the basic features of data with the statistical process.

Here

pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find

mean, std, min, max and percentile values of continuous features.

```
describeNum= data.describe(include=['float64','int64','float','int'])
describeNum.T.style.background_gradient(cmap='viridis',low=0.2,high=0.1)
```

	count	mean	std	min	25%	50%	75%	max
Unnamed: 0	840.000000	570.323810	333.965519	1.000000	278.500000	575.000000	865.250000	1153.000000
latitude	840.000000	38.410522	3.689577	25.752358	37.386778	37.779281	40.730646	59.335232
longitude	840.000000	-103.766293	21.862099	-122.756956	-122.187927	-118.355788	-77.381456	18.057121
age_first_funding_year	840.000000	1.015180	0.647373	0.000000	0.519701	0.976890	1.559591	3.130958
age_last_funding_year	840.000000	1.464845	0.609871	0.000000	1.070830	1.549008	1.904969	3.130958
age_first_milestone_year	840.000000	1.059334	0.766397	0.000000	0.284276	1.151521	1.641789	3.245903
age_last_milestone_year	840.000000	1.352650	0.824112	0.000000	0.715821	1.606141	1.978262	3.245903
relationships	840.000000	7.752381	7.280069	0.000000	3.000000	5.500000	10.000000	63.000000
funding_rounds	840.000000	2.320238	1.404427	1.000000	1.000000	2.000000	3.000000	10.000000
funding_total_usd	840.000000	15.835517	1.786871	9.305741	15.001589	16.170264	17.059044	22.463732
milestones	840.000000	1.820238	1.326298	0.000000	1.000000	2.000000	3.000000	8.000000
is_CA	840.000000	0.532143	0.499263	0.000000	0.000000	1.000000	1.000000	1.000000

Activity 2: Visual analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed

Activity 2.1:Univariate Analysis

In simple words, univariate analysis is understanding the data with single feature.

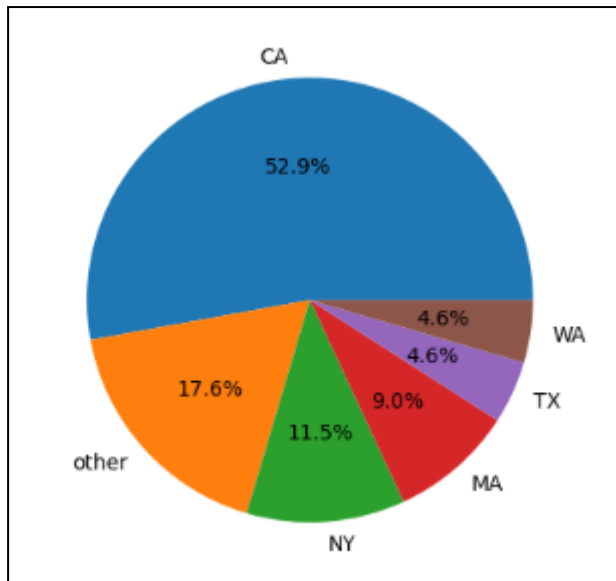
Here we have displayed two different graphs such as Piechart and countplot.

Seaborn package provides a wonderful function countplot. It is more useful for categorical features. With the help of countplot, we can Number of unique values in the feature.

```
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()
```

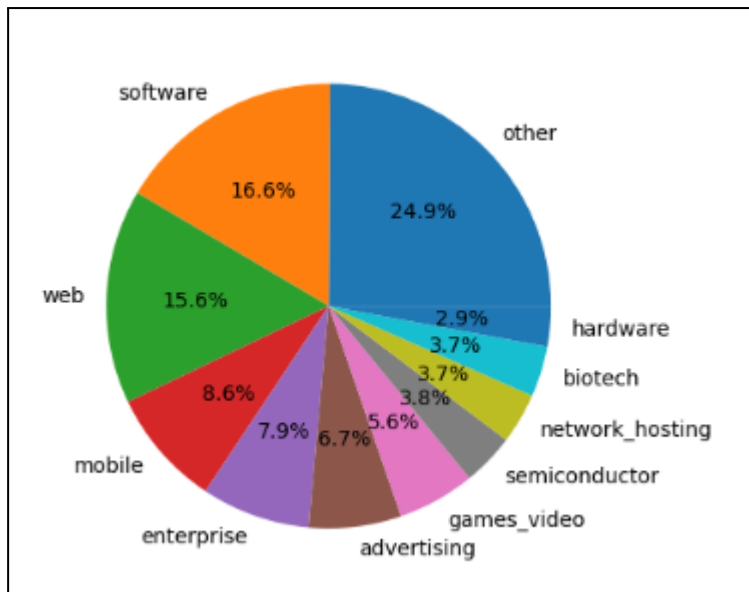
It assigns the value 'other' to the 'State' column for all rows in the 'data' DataFrame. Then, specific rows where the 'state_code' matches certain values (CA, NY, MA, TX, WA) are assigned their respective state names in the 'State' column. The 'value_counts()' method is used to count the occurrences of each unique value in the 'State' column, and a pie chart is created to visualize the distribution of states.



```
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()
```

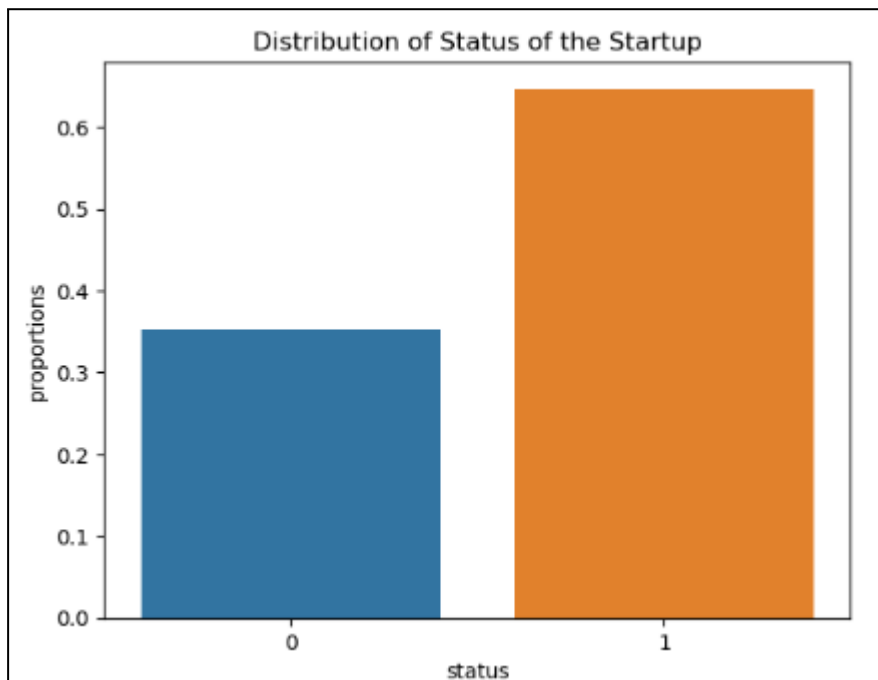
The given code snippet performs a univariate analysis. It assigns the value 'other' to the 'category' column for all rows in the 'data' DataFrame. Then, specific rows matching certain 'category_code' values are assigned corresponding categories in the 'category' column. The 'value_counts()' method is 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.



- **Distribution of Status of Startup**

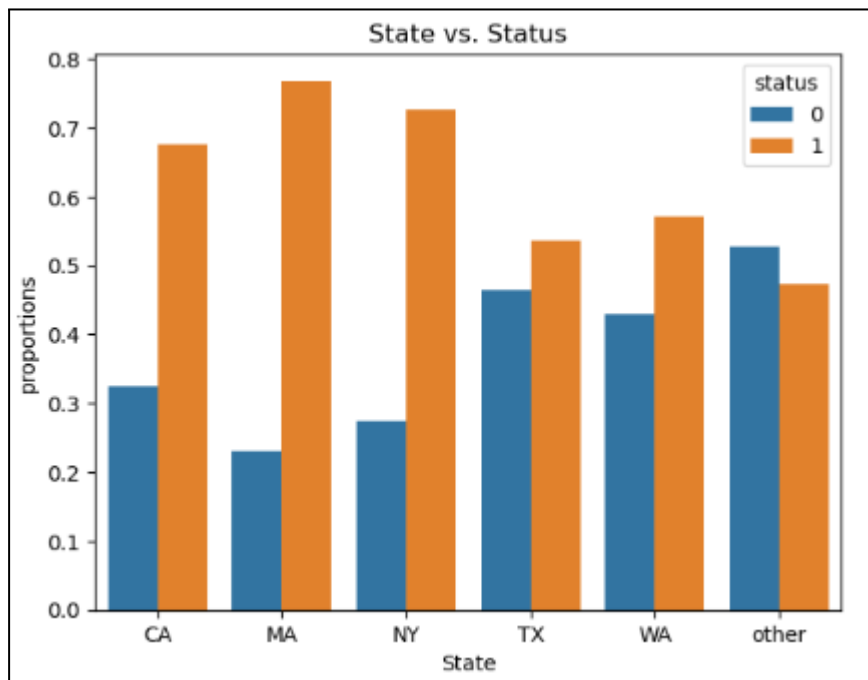
```
prop_df = data.groupby('status').size().reset_index(name = 'counts')
prop_df['proportions'] = prop_df['counts']/prop_df['counts'].sum()

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



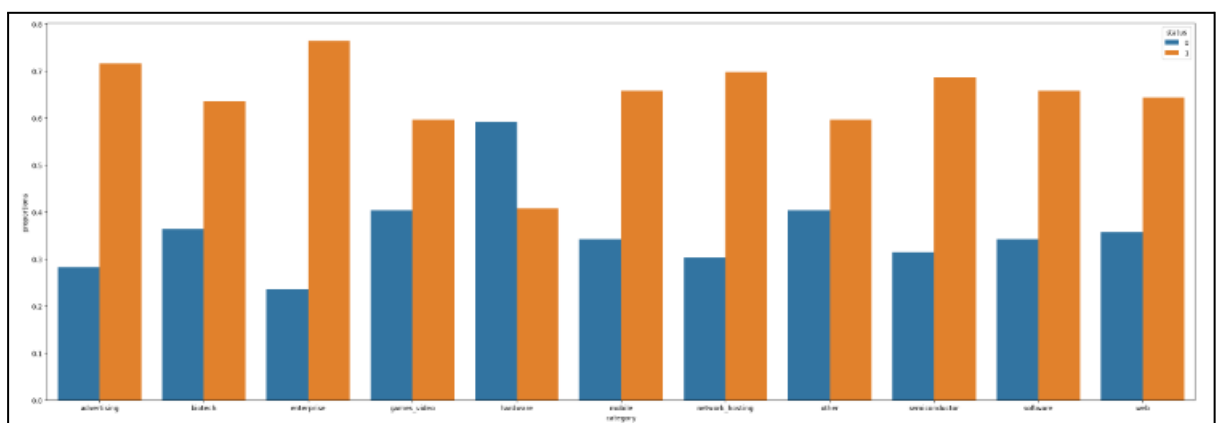
- **State Vs. Staus**

```
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')
```



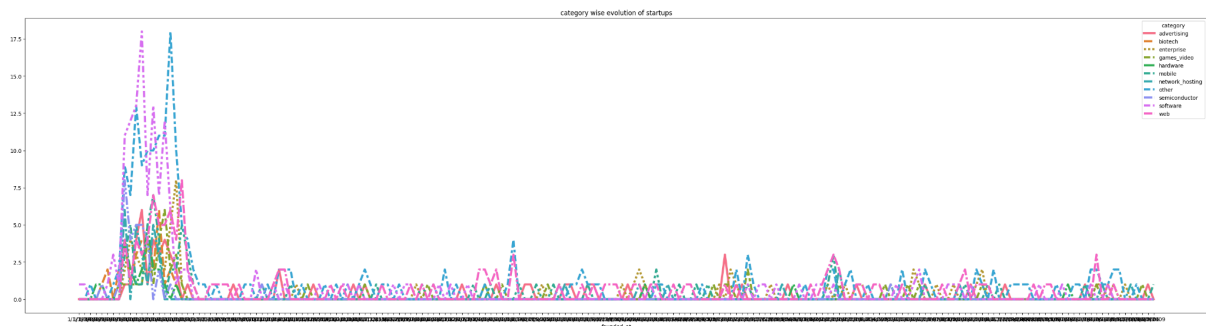
• State Vs. Category

```
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')
```



• Category Vs. Founded- Year

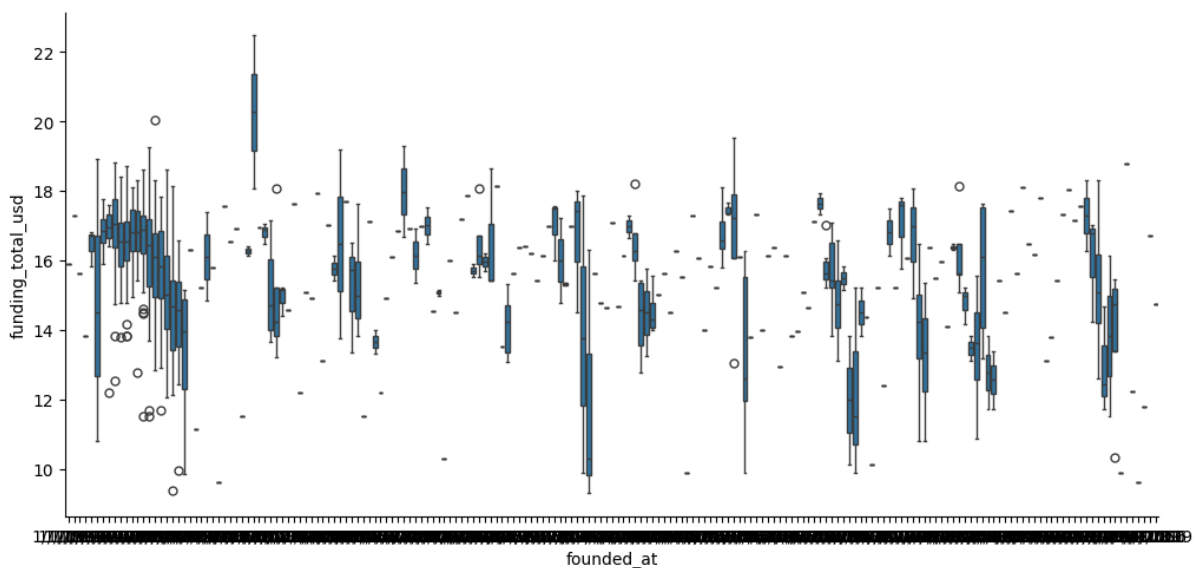
```
fig, ax = plt.subplots(figsize=(40, 10))
sns.lineplot(data=cat_year, lw=4, )
plt.title('category wise evolution of startups')
plt.show()
```



● Founded- Year Vs. Total Funding

```
# Use the actual unique values in 'founded_at' for the order
order = sorted(data['founded_at'].unique())
sns.catplot(data=data, x='founded_at', y='funding_total_usd', kind='box', height=5, aspect=2, order=order)
```

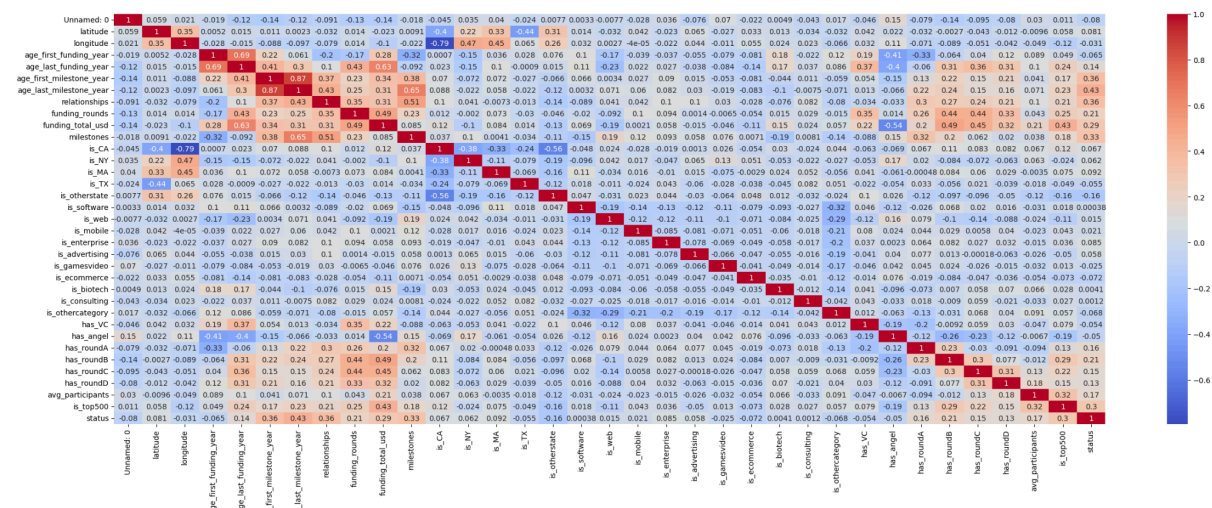
✓ 4.2s



Activity 2.3: Multivariate analysis

In simple words, multivariate analysis is to find the relation between multiple features.

Here we have used heatmap from seaborn package.



The heatmap reveals how various startup-related features—such as funding amounts, milestone ages, and other operational metrics—interrelate. Strong positive correlations between features like **age_first_funding_year** and **age_last_funding_year**, or between milestone ages, suggest that startups with early funding tend to progress through milestones in a predictable timeline. Similarly, a high correlation between **funding_total_usd** and milestone-related features may indicate that well-funded startups are more likely to reach key milestones faster.

These relationships are crucial for predictive modeling:

- 🔍 **Highly correlated features** can improve model accuracy when predicting success outcomes like IPO, acquisition, or long-term survival.
- ⚠️ **Redundant or overly correlated variables** might need dimensionality reduction (e.g., PCA) to avoid overfitting.
- 📊 **Weak or negative correlations** may highlight features that contribute less to success or even signal risk factors.

In short, understanding these correlations helps refine feature selection and build more robust models for forecasting startup trajectories.

Activity 3: data pre-processing

Activity 3.1: Reducing the number of columns

By combining the founded_at and closed_at columns into age_closed_startup

```
data['age_closed_startup'] = data.apply(lambda row: (row.closed_at - row.founded_at), axis=1)
```

Activity 3.2: Dropping the irrelevant 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)
```

The code snippet drops multiple columns ('category_code', 'is_software', 'is_web', 'is_mobile', 'is_enterprise', 'is_advertising', 'is_gamesvideo', 'is_ecommerce',

'is_biotech', 'is_consulting', 'is_othercategory') from the 'data' DataFrame, effectively removing those columns from the dataset.

```
data = data.drop(['id', 'name', 'city', 'state_code', 'zip_code', 'founded_at', 'first_funding_at', 'last_funding_at', 'Unnamed: 6', 'object_id', 'Unnamed: 0', 'closed_at', 'latitude', 'longitude', 'age_closed_startup', 'state', 'category', 'latitude', 'longitude', 'age_closed_startup', 'unnamed 0', 'unnamed 6', 'object_id'])
```

The code snippet drops multiple columns ('id', 'name', 'city', 'state_code', 'zip_code', 'founded_at', 'first_funding_at', 'last_funding_at', 'closed_at', 'state', 'category', 'latitude', 'longitude', 'age_closed_startup', 'unnamed 0', 'unnamed 6', 'object_id') from the 'data' DataFrame, effectively removing those columns from the dataset.

End with of 22 columns

Milestone 3: Model Building

Activity 3.1: split the data for training and testing

The code splits the DataFrame df into input features X and the target variable y. The target variable is assigned to y, while the input features are assigned to X after removing the 'Adaptivity Level' column using the drop() function. The data is then split into training and testing sets using the train_test_split() function, with 80% of the data allocated for training (X_train, y_train) and 20% for testing (X_test, y_test), ensuring reproducibility with a random state of 42

```
# define feature matrix `x` and target `y`
# prefer `selected_features` if it's defined in a later cell, otherwise fall back to the expected numeric feature list
x = data.drop('status', axis=1)

y = data['status']

# split train and test set (stratify to preserve class proportions)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

Activity 3.2: Random Forest Classifier

- The code snippet performs a grid search using the GridSearchCV class to find the best combination of hyperparameters for a Random Forest classifier. The parameter grid is defined with different values for 'n_estimators', 'max_depth', 'min_samples_split', 'min_samples_leaf', and 'bootstrap'. The grid search is performed using 5-fold cross-validation (cv=5) and parallelized (-1 for n_jobs). After fitting the grid search object to the training data, it prints the best parameters found based on the evaluation of different parameter combinations.


```
#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}
```

First Random Forest Model is imported from sklearn Library then RandomForestClassifier

algorithm is initialised and training data is passed to the model with .fit() function. Test

data is predicted with .predict() function and saved in a new variable. We can find the Train and Test accuracy by X_train and X_test.

- The _scale function takes in training and validation data along with a list of features. It applies the StandardScaler to standardize the numerical features in the training data. It then uses the computed scaler to transform both the training and validation data. The function returns the updated training and validation data with the scaled features. This ensures that the features have zero mean and unit variance, which can be beneficial for certain machine learning algorithms.

```
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(confusion_matrix(y_pred_rf,y_test))
print(cr_rf)
```

✓ 2.4s

```
[ 21   9]
[ 28 110]]
```

	precision	recall	f1-score	support
0	0.43	0.70	0.53	30
1	0.92	0.80	0.86	138
accuracy			0.78	168
macro avg	0.68	0.75	0.69	168
weighted avg	0.84	0.78	0.80	168

The classification report provides an evaluation of the model's performance. For class 0, the precision is 0.43, recall is 0.70, and F1-score is 0.53. For class 1, the precision is 0.92, recall is 0.80, and F1-score is 0.86. The overall accuracy of the

model is 0.78. The macro average of precision, recall, and F1-score is 0.68, 0.75, and 0.69, respectively. The weighted average of precision, recall, and F1-score is 0.84, 0.78, and 0.80, respectively

Milestone 4: Performance Testing

Activity 4.1: Analyzing Accuracy

The code snippet performs evaluation and saves the results of a model. It imports several metrics from `sklearn.metrics`, including `accuracy_score`, `classification_report`, `confusion_matrix`, `roc_curve`, and `roc_auc_score`. The model's predictions are compared with the true labels (`y_test`), and the accuracy score is calculated. Additionally, a classification report is generated using the predicted labels and the true labels. Finally, the results, including the predictions, accuracy, and classification report, are stored in a dictionary called "results."

```
#applying Random forest classifier
model=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 accuracy: 0.7797619047619048
Train accuracy: 1.0
```

The Random Forest classifier achieved an accuracy of 77% on the test dataset (`test_acc`). This means that the model correctly predicted the target variable for 77% of the test instances. However, the model achieved a perfect accuracy of 100% on the training dataset (`train_acc`), indicating that it may be overfitting the training data. Overfitting occurs when a model performs exceptionally well on the training data but fails to generalize well to unseen data. Further evaluation and tuning may be necessary to address the potential overfitting issue.

Activity 4.2 Testing The Model

Here we have tested with Random Forest Classifier. You can test . With the help of `predict()` function.

```
for successful

print(model_rf.predict([[1.178440, 1.386969, 1.734925, 2.041753, 3, 3, 12.834684, 3, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1.0000, 0]]))
✓ 0.0s

[1]
```

```
for unsuccessful

print(model_rf.predict([[0.000000, 0.981517, 0.037681, 0.037681, 2, 2, 14.077876, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1.0000, 1]]))
✓ 0.0s

[0]
```

Milestone 5: Model Deployment

Activity 1: Save the best model

Saving the best model after comparing its performance using different evaluation metrics

means selecting the model with the highest performance. This can be useful in avoiding

the need to retrain the model every time it is needed and also to be able to use it in the future.

```
%pip install joblib
import joblib

# save the trained model to a file
joblib.dump(model_rf, 'random_forest_model.pkl')
✓ 4.6s
```

The provided code snippet demonstrates the usage of joblib library for training and saving a Random Forest Classifier model. First, the Random Forest Classifier is trained using the training data `X_train` and target labels `y_train`. The trained model is then saved to a file named `'random_forest_model.pkl'` using the `joblib.dump()` function.

Later, to make predictions on new data, the saved model is loaded from the file using the `joblib.load()` function. The loaded model can

then be used to make predictions on the test data `X_test` by calling the `model.predict()` method, which returns an array of predicted labels.

We save the model using the pickle library into a file named `random_forest_model.pkl`

Activity 2: Integrate with Web Framework

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

Building HTML Pages

Building server-side script

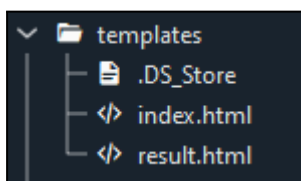
Run the web application

Activity 2.1: Building Html Page:

For this project, we create two HTML files namely

- `Index.html`
- `Results.html`
- `home.html`
- `adaptivity.html`

And we will save them in the templates folder.



Activity 2.2: Build Python code:

Create a new `app.py` file which will be stored in the Flask folder.

- **Import the necessary Libraries.**

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

- This code uses the joblib module to load a saved machine-learning model and a saved preprocessing object. The joblib.load function is used to load the model object from a file called model.pkl and the scaler object from a file called random_forest_model.pkl. These objects are then stored in the model and scaler variables, respectively. The loaded model object can be used to make predictions on new data and the loaded scaler object can be used to preprocess the data in the same way as it was done during training. This process of loading saved models and preprocessing objects can save time and resources when working on a new project or dataset.

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

- This code creates a new instance of a Flask web application using the Flask class from the Flask library. The __name__ argument specifies the name of the application's module or package.

```
app = Flask(__name__)
```

- The code snippet defines a route '/' for the Flask application. When a user accesses the homepage of the application, it calls the 'home' function. This function renders the 'index.html' template, which will be displayed as the homepage of the application.

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

- The code snippet defines a route '/predict' for the Flask application, which expects a POST request. When this route is accessed, the 'predict' function is called. Inside the function, it retrieves input values from a form submitted with the POST request. The values are converted to float and stored in respective variables for further processing.

```

app.route('/predict', methods=['POST'])
def predict():
    # Extract features from the form
    age_first_funding_year = float(request.form.get('feature1', 0))
    age_last_funding_year = float(request.form.get('feature2', 0))
    age_first_milestone_year = float(request.form.get('feature3', 0))
    age_last_milestone_year = float(request.form.get('feature4', 0))
    relationship = float(request.form.get('feature5', 0))
    funding_rounds = float(request.form.get('feature6', 0))
    funding_total_usd = float(request.form.get('feature7', 0))
    milestones = float(request.form.get('feature8', 0))
    canada = float(request.form.get('feature9', 0))
    newyork = float(request.form.get('feature10', 0))
    ma = float(request.form.get('feature11', 0))
    texas = float(request.form.get('feature12', 0))
    other_states = float(request.form.get('feature13', 0))
    vc = float(request.form.get('feature14', 0))
    angel = float(request.form.get('feature15', 0))
    roundA = float(request.form.get('feature16', 0))
    roundB = float(request.form.get('feature17', 0))
    roundC = float(request.form.get('feature18', 0))
    roundD = float(request.form.get('feature19', 0))
    avg_participants = float(request.form.get('feature20', 0))
    top500 = float(request.form.get('feature21', 0))
    # Create a list of input features

```

- The code snippet creates a list called 'input_data' containing the input values retrieved from the form. It then uses a loaded model to make a prediction based on the input data. The predicted label is mapped to a meaningful output ('Successful' or 'Unsuccessful'). Finally, the 'result' variable is passed to the 'result.html' template for rendering the prediction result on a webpage.

```

input_features = [age_first_funding_year, age_last_funding_year, age_first_milestone_year, age_last_milestone_year, relationship, funding_rounds, funding_total_usd, milestones, canada, newyork, ma, texas, other_states, vc, angel, roundA, roundB, roundC, roundD, avg_participants, top500]
#input_features = [age_first_funding_year, age_last_funding_year, age_first_milestone_year, age_last_milestone_year, relationship, funding_rounds, funding_total_usd, milestones, avg_participants]
# Make prediction
prediction = model.predict([input_features])[0]
# map the prediction to a human-readable label
if prediction == 1:
    result = 'Successful'
else:
    result = 'Unsuccessful'
# Render the result template with the prediction
return render_template('result.html', result=result)

```

Main Function:

This code runs the Flask application if the script is being executed directly (i.e. not imported as a module in another script). The `if __name__ == '__main__':` line checks if the script is the main module being executed and if so, runs the Flask application using the `app.run()` method. This method starts the Flask development

server, allowing the application to be accessed via a web browser at the appropriate URL.

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

Activity 2.3: Run the web application

- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

```
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!
```

Now, Go to the web browser and write the localhost url (<http://127.0.0.1:5000>) to get the below result

Welcome to Startup Success Prediction

This web application predicts the success of a startup based on various features.

Age at First Funding Year:

Age at Last Funding Year:

Age at First Milestone Year:

Age at Last Milestone Year:

Number of Relationships:

Startup Success Prediction

127.0.0.1:5000

12.834604

Number of Milestones:

3

is_CANADA:

1

is_NEWYORK:

0

is_MA:

0

is_Texas:

0

Startup Success Prediction

127.0.0.1:5000

is_otherstate:

0

has_VC:

0

has_angel:

1

has_roundA:

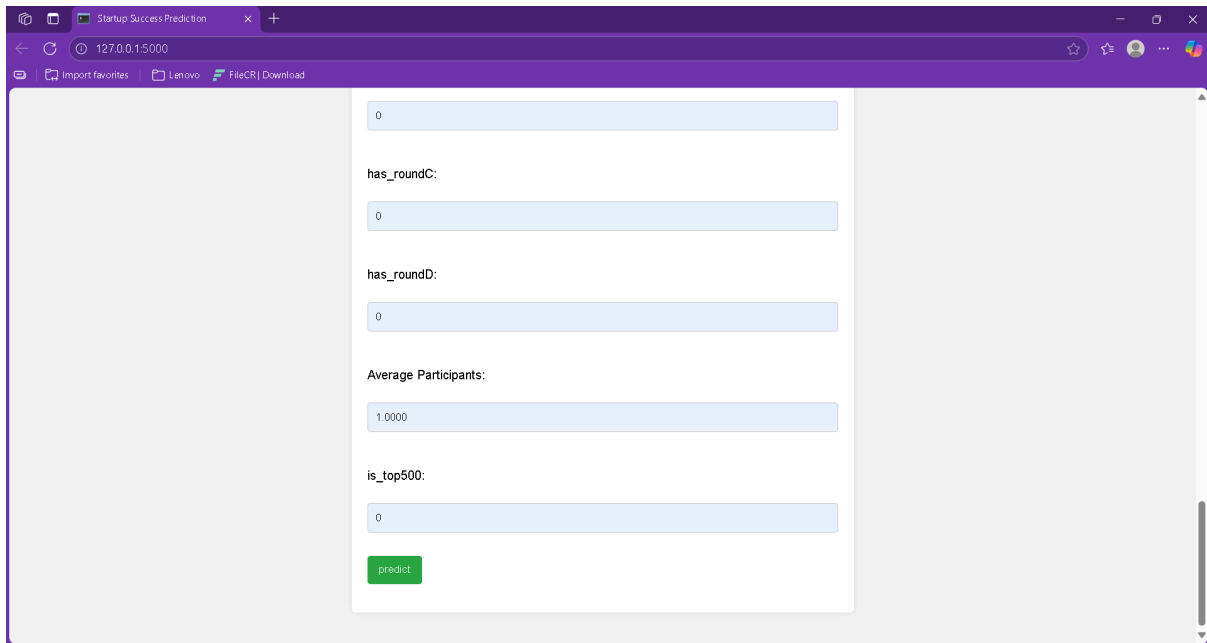
0

has_roundB:

0

has_roundC:

0



Startup Success Prediction

127.0.0.1:5000

0

has_roundC:

0

has_roundD:

0

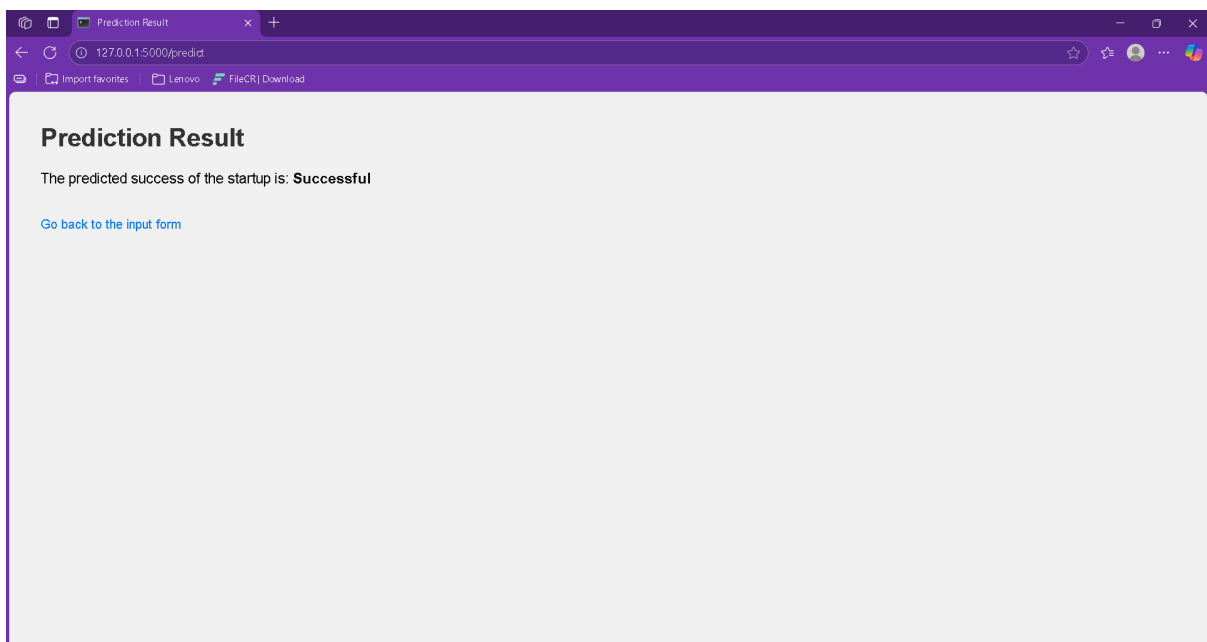
Average Participants:

1.0000

is_top500:

0

predict



Prediction Result

The predicted success of the startup is: **Successful**

[Go back to the input form](#)

Milestone 6: Project Demonstration & Documentation

Below mentioned deliverables to be submitted along with other deliverables

Activity 1:- Record explanation Video for project end to end solution

Activity 2:- Project Documentation-Step by step project development procedure