Prosperity Prognosticator: Machine Learning for Startup Success Prediction

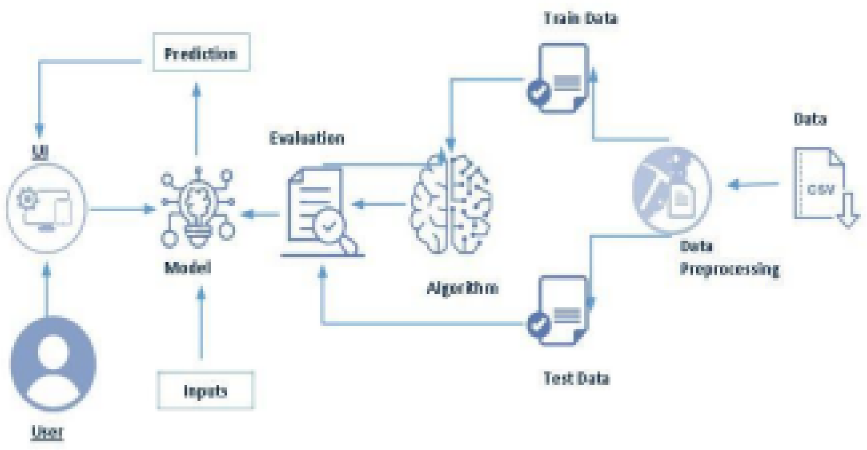
Project Description:

One of the most important factors that influences a country's economy and financial ecosystem is the growth and sustainability of startups. The process of startup success evaluation is gaining recognition among investors and entrepreneurs across the globe. As we know, startup success prediction is very crucial, and a variety of techniques are used to calculate the probability of growth or failure. In addition, evaluating startup performance has become one of the main functions in modern investment and business decision-making.

The prediction of startup success or failure is one of the difficult tasks for stakeholders. By forecasting whether a startup may be acquired or closed, investors can reduce financial losses and avoid unproductive investments. This helps in improving strategic planning and ensures better allocation of resources. This makes the study of startup success prediction important. Machine Learning techniques are very useful and crucial in predicting these types of outcomes.

We will be using classification algorithms such as Logistic Regression, Decision Tree, Support Vector Machine, and Random Forest. We will train and test the dataset using these algorithms. From this, the best model will be selected and saved in pkl format. We will be doing flask integration and deployment.

Technical Architecture:



Pre requisites:

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

- **VS Code and Python:**

o Refer the link below to download and install VS Code
o Link : https://code.visualstudio.com/

- **Python packages:**
  - o Open command prompt as administrator
  - o Navigate to your project folder (optional).
  - o Type “pip install numpy” and click enter.
  - o Type “pip install pandas” and click enter.
  - o Type “pip install scikit-learn” and click enter.
  - o Type ”pip install matplotlib” and click enter.
  - o Type ”pip install scipy” and click enter.
  - o Type ”pip install pickle-mixin” and click enter.
  - o Type ”pip install seaborn” and click enter.
  - o Type “pip install Flask” and click enter.

Prior Knowledge:

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

- **ML Conccepts**
  - o SSupervised learning: https://www.javatpoint.com/supervised-machine-learning
  - o Unsupervised learning: https://www.javatpoint.com/unsupervised-machine-learning
  - o Regression and classification
  - o Logistic Regression: https://www.javatpoint.com/logistic-regression-in-machine-learning
  - o Decision tree: https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
  - o Support Vector Machine: https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm
  - o Random forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm
  - o 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.

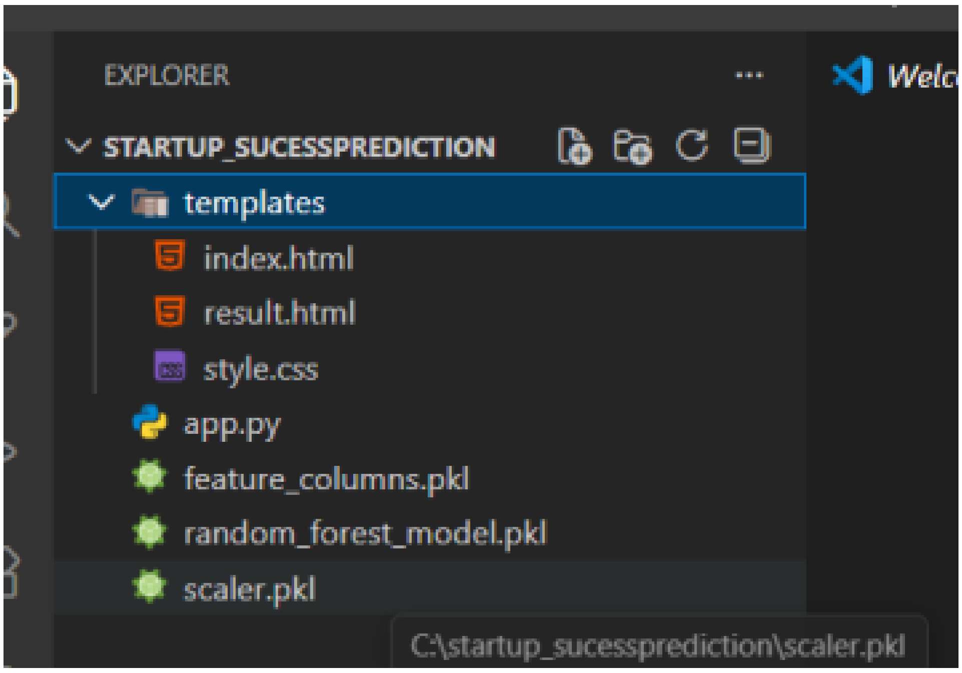
Project Flow:

- 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



```
templates
    index.html
    result.html
    style.css
app.py
feature_columns.pkl
random_forest_model.pkl
scaler.pkl
C:\startup_sucessprediction\scaler.pkl
```

• 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 deployment folder contains application 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, UCI repository, etc.

In this project we have used startup data.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

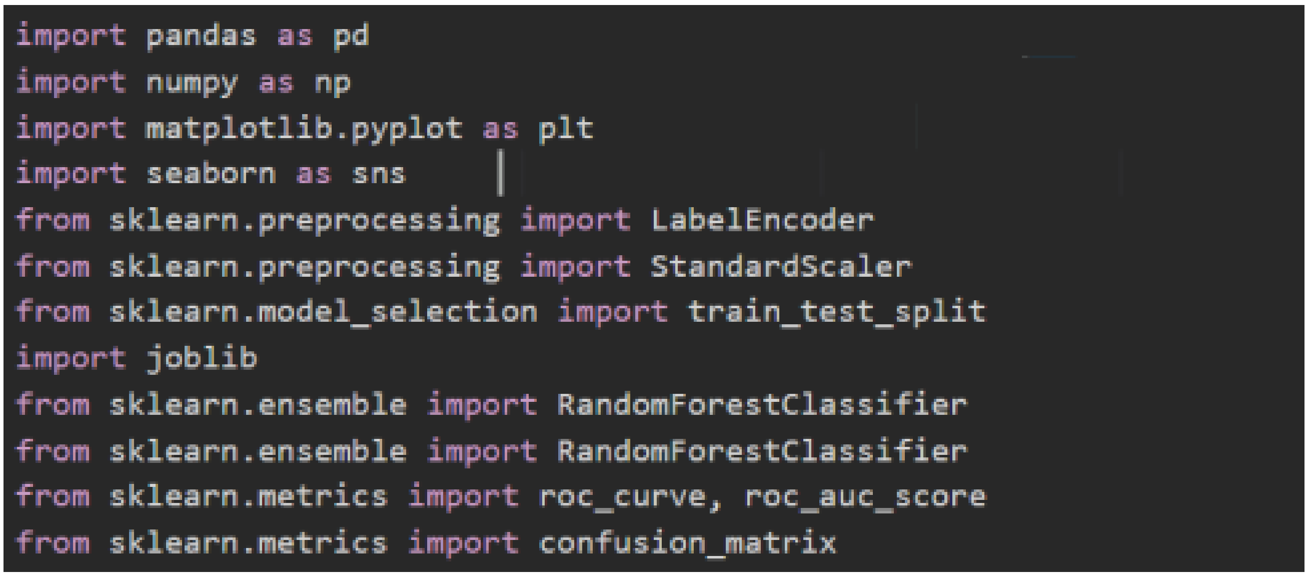
Milestone 2: Visualizing and analysing the data

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.

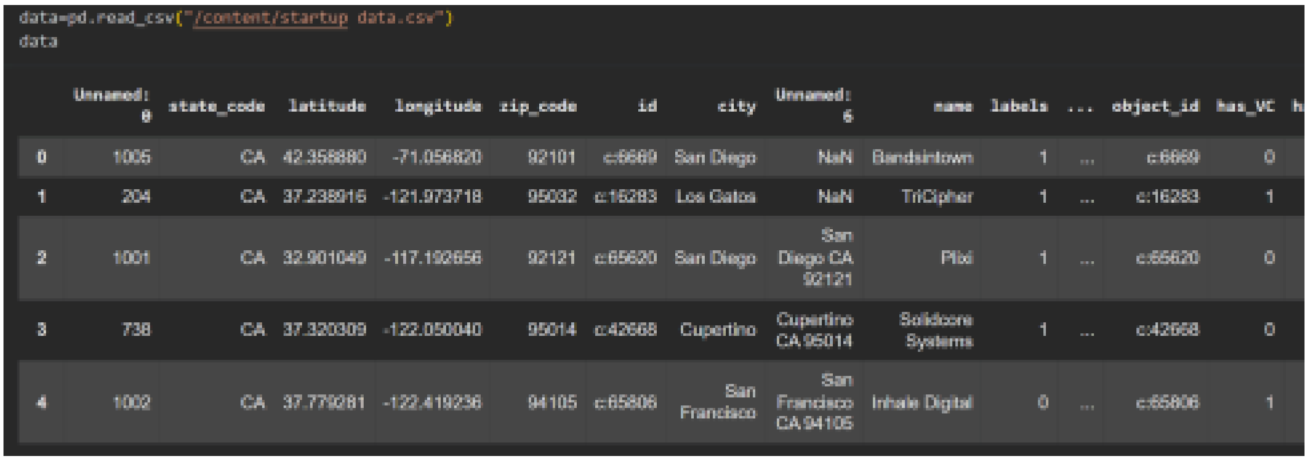


```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import joblib
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.metrics import confusion_matrix
```

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.



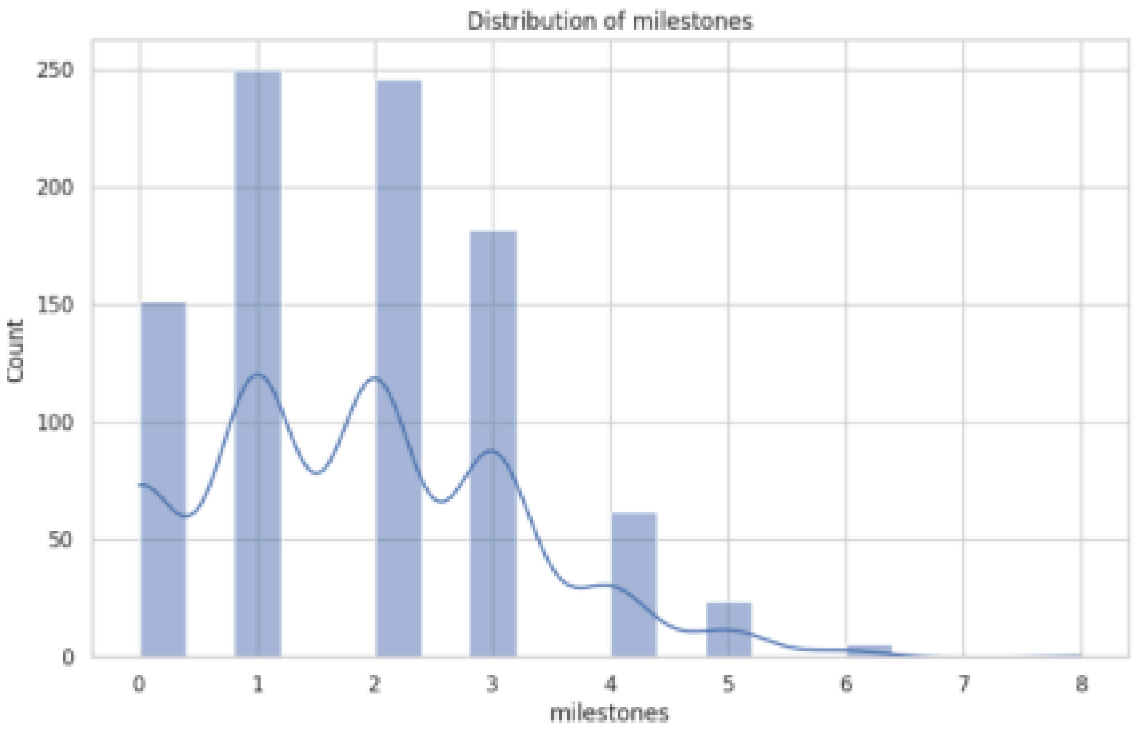
```python
data=pd.read_csv("/content/startup data.csv")
data
```

	Unnamed: 0	state_code	latitude	longitude	zip_code	id	city	Unnamed: 6	name	labels	...	object_id	has_VC	h
0	1005	CA	42.358880	-71.056820	92101	c:6669	San Diego	NaN	Bandsintown	1	...	c:6669	0	
1	204	CA	37.238916	-121.973718	95032	c:16283	Los Gatos	NaN	TriCipher	1	...	c:16283	1	
2	1001	CA	32.901049	-117.192656	92121	c:65620	San Diego	San Diego CA 92121	Plixi	1	...	c:65620	0	
3	738	CA	37.320309	-122.050040	95014	c:42668	Cupertino	Cupertino CA 95014	Solidcore Systems	1	...	c:42668	0	
4	1002	CA	37.779281	-122.419236	94105	c:65806	San Francisco	San Francisco CA 94105	Inhale Digital	0	...	c:65806	1	

Activity 3: Univariate analysis

In simple words, univariate analysis is understanding the data using a single feature. Here different graphs such as histogram plots with KDE and countplot were used to analyze individual features

- Seaborn's histplot function was used to visualize the distribution of important numerical features such as funding_total_usd, milestones, relationships and funding_rounds. The KDE curve was included to observe the density distribution of these features.



- The categorical target variable status was analyzed using a countplot to understand the number of startups in each category. The percentage distribution of each class was also calculated to examine class balance.
- From the plots it was observed that funding-related features show skewed distributions and the status variable contains two categories representing different startup outcomes.

Countplot:-

A count plot can be thought of as a histogram across a categorical variable instead of a quantitative variable. It helps in comparing counts across categories.

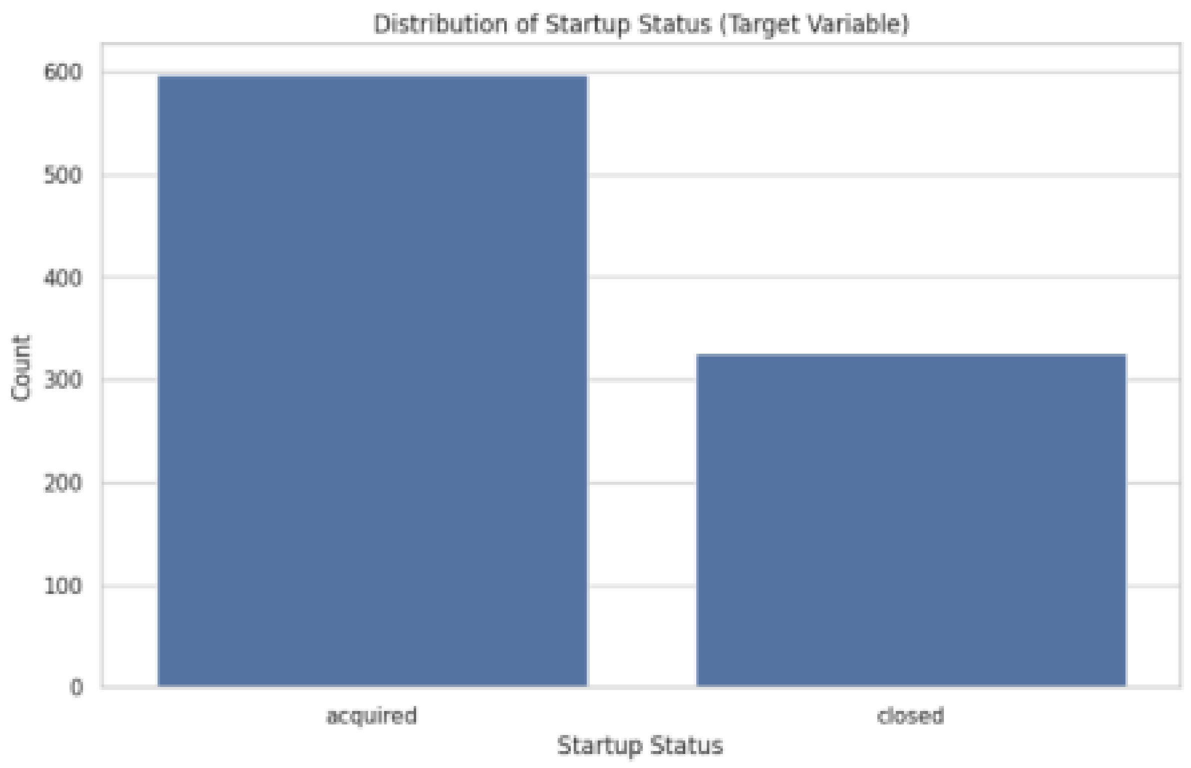
From the graph we can infer the analysis such as

- Understanding the distribution of startup status in the dataset.

● Observing patterns in funding_total_usd and funding_rounds in relation to startup outcomes. ● Identifying how milestones and relationships contribute to startup performance.

.

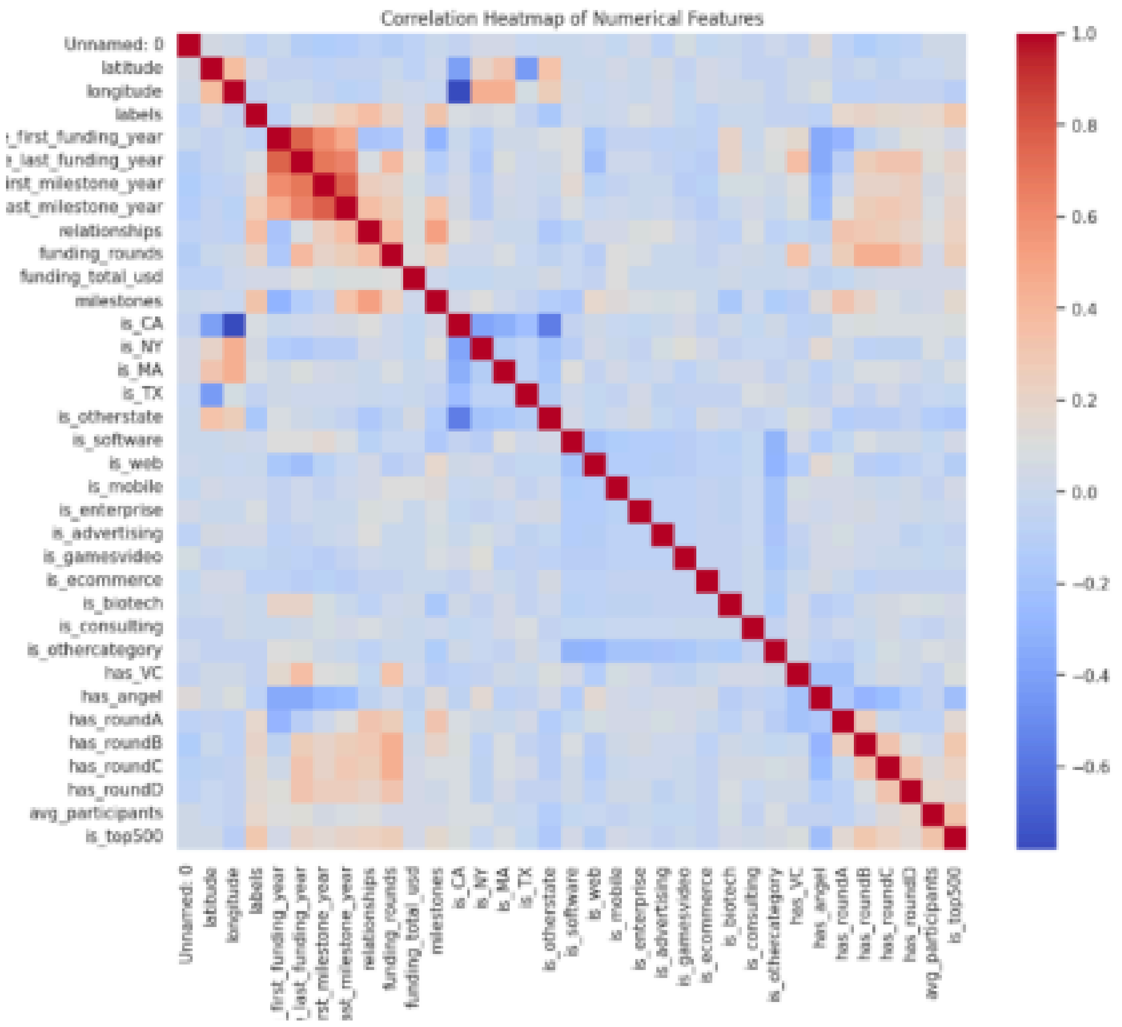
Activity 4: Bivariate analysis



```
status
acquired    64.68039
closed      35.31961
Name: proportion, dtype: float64
```

Countplot:-

A count plot can be thought of as a histogram across a categorical variable instead of a quantitative variable. It helps in comparing counts across categories.



From the graph we can infer the analysis such as

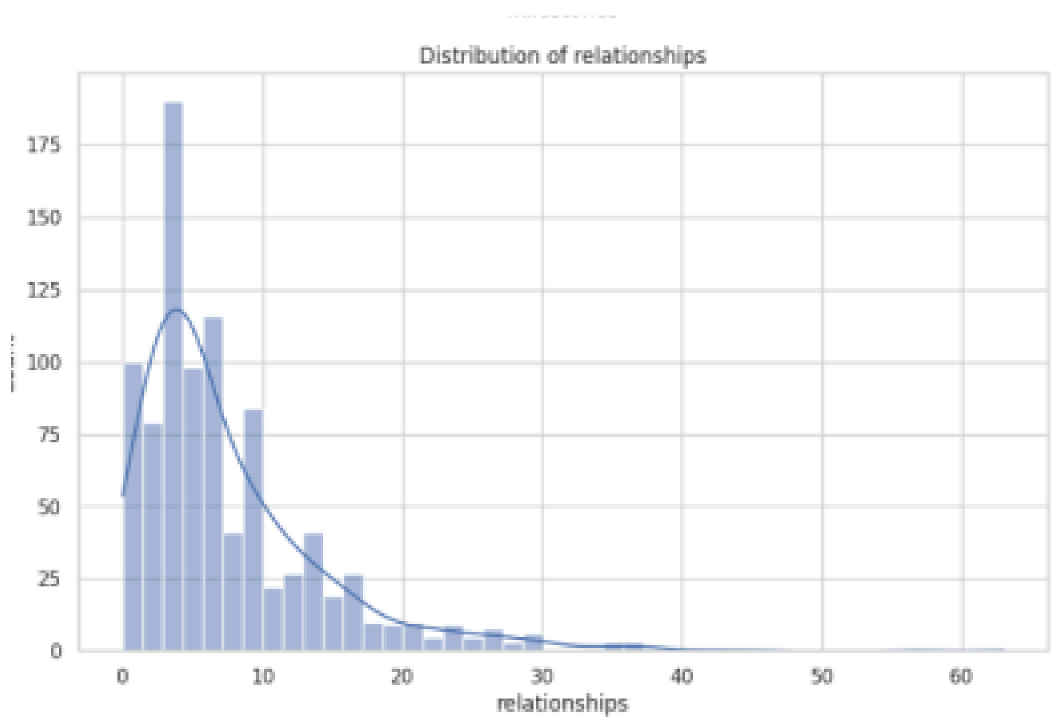
● Understanding the distribution of startup status in the dataset.
● Observing patterns in funding_total_usd and funding_rounds in relation to startup outcomes. ● Identifying how milestones and relationships contribute to startup performance.

This analysis helps in understanding the relationship between features and the target

variable. **Activity 5: Multivariate analysis**

In simple words, multivariate analysis is used to find the relationship between multiple

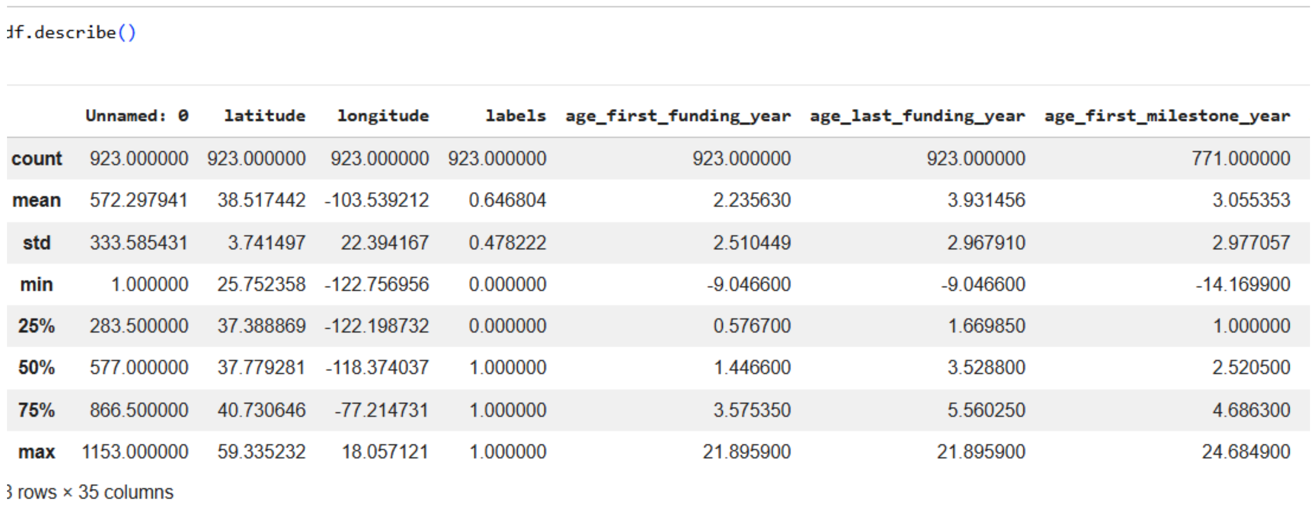
features. Here a correlation heatmap from seaborn package was used.



From the heatmap we analyzed the relationship between numerical features such as funding_total_usd, milestones, relationships and funding_rounds. The heatmap shows positive and negative correlations among these variables.

Activity 6: Descriptive analysis

Descriptive analysis is used to study the basic features of data using statistical methods. The pandas describe() function was used to understand count, mean, standard deviation, minimum, maximum and percentile values of numerical features.



```
df.describe()
```

	Unnamed: 0	latitude	longitude	labels	age_first_funding_year	age_last_funding_year	age_first_milestone_year
count	923.000000	923.000000	923.000000	923.000000	923.000000	923.000000	771.000000
mean	572.297941	38.517442	-103.539212	0.646804	2.235630	3.931456	3.055353
std	333.585431	3.741497	22.394167	0.478222	2.510449	2.967910	2.977057
min	1.000000	25.752358	-122.756956	0.000000	-9.046600	-9.046600	-14.169900
25%	283.500000	37.388869	-122.198732	0.000000	0.576700	1.669850	1.000000
50%	577.000000	37.779281	-118.374037	1.000000	1.446600	3.528800	2.520500
75%	866.500000	40.730646	-77.214731	1.000000	3.575350	5.560250	4.686300
max	1153.000000	59.335232	18.057121	1.000000	21.895900	21.895900	24.684900

3 rows × 35 columns

Milestone 3: Data Pre-processing

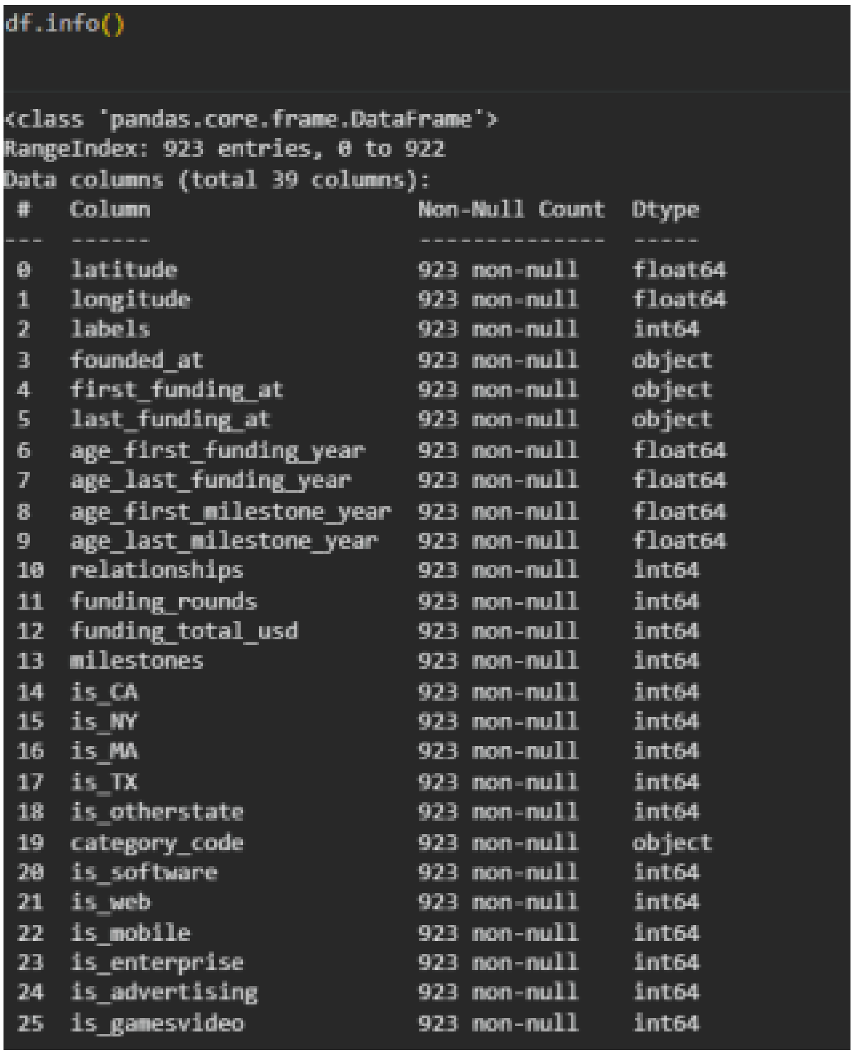
As we have understood how the data is lets pre-process the collected data.
The download data set is not suitable for training the machine learning model as it might have so much of randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling categorical data
- Handling outliers
- Scaling Techniques
- Splitting dataset into training and test set

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.

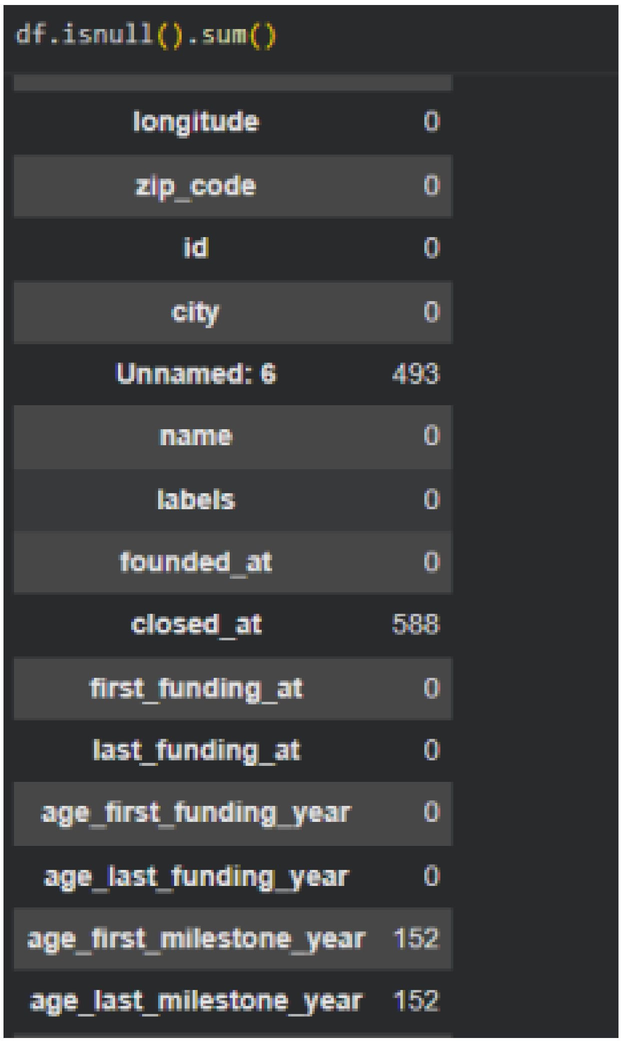
Activity 1: Checking for null values

- Let's find the shape of our dataset first, To find the shape of our data, df.shape method is used. To find the data type, df.info() function is used.



```
df.info()


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 923 entries, 0 to 922
Data columns (total 39 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   latitude                 923 non-null    float64
 1   longitude                923 non-null    float64
 2   labels                   923 non-null    int64
 3   founded_at               923 non-null    object
 4   first_funding_at         923 non-null    object
 5   last_funding_at          923 non-null    object
 6   age_first_funding_year   923 non-null    float64
 7   age_last_funding_year    923 non-null    float64
 8   age_first_milestone_year 923 non-null    float64
 9   age_last_milestone_year  923 non-null    float64
 10  relationships            923 non-null    int64
 11  funding_rounds           923 non-null    int64
 12  funding_total_usd        923 non-null    int64
 13  milestones               923 non-null    int64
 14  is_CA                    923 non-null    int64
 15  is_NY                    923 non-null    int64
 16  is_MA                    923 non-null    int64
 17  is_TX                    923 non-null    int64
 18  is_otherstate            923 non-null    int64
 19  category_code            923 non-null    object
 20  is_software              923 non-null    int64
 21  is_web                   923 non-null    int64
 22  is_mobile                923 non-null    int64
 23  is_enterprise            923 non-null    int64
 24  is_advertising           923 non-null    int64
 25  is_gamesvideo            923 non-null    int64
```

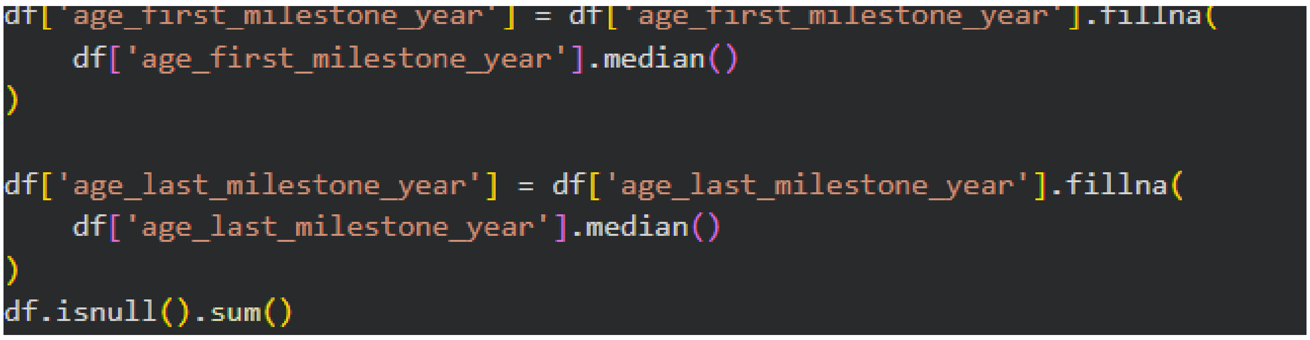
- For checking the null values, df.isnull() function is used. To sum those null values we use .sum() function to it. From the below image we found that there are no null values present in our dataset. So we can skip handling of missing values step.



```
df.isnull().sum()
```

```
longitude                    0
zip_code                     0
id                           0
city                         0
Unnamed: 6                 493
name                         0
labels                       0
founded_at                   0
closed_at                  588
first_funding_at             0
last_funding_at              0
age_first_funding_year       0
age_last_funding_year        0
age_first_milestone_year   152
age_last_milestone_year    152
```

From the above code of analysis, we can infer that columns such as Unnamed: 6, closed_at, age_first_milestone_year, and age_last_milestone_yearare having the



```
df['age_first_milestone_year'] = df['age_first_milestone_year'].fillna(
    df['age_first_milestone_year'].median()
)

df['age_last_milestone_year'] = df['age_last_milestone_year'].fillna(
    df['age_last_milestone_year'].median()
)
df.isnull().sum()
```

missing values, we need to treat them in a required way.

We will fill the missing values in numeric data type using mean value of that particular column and categorical data type using the most repeated value.

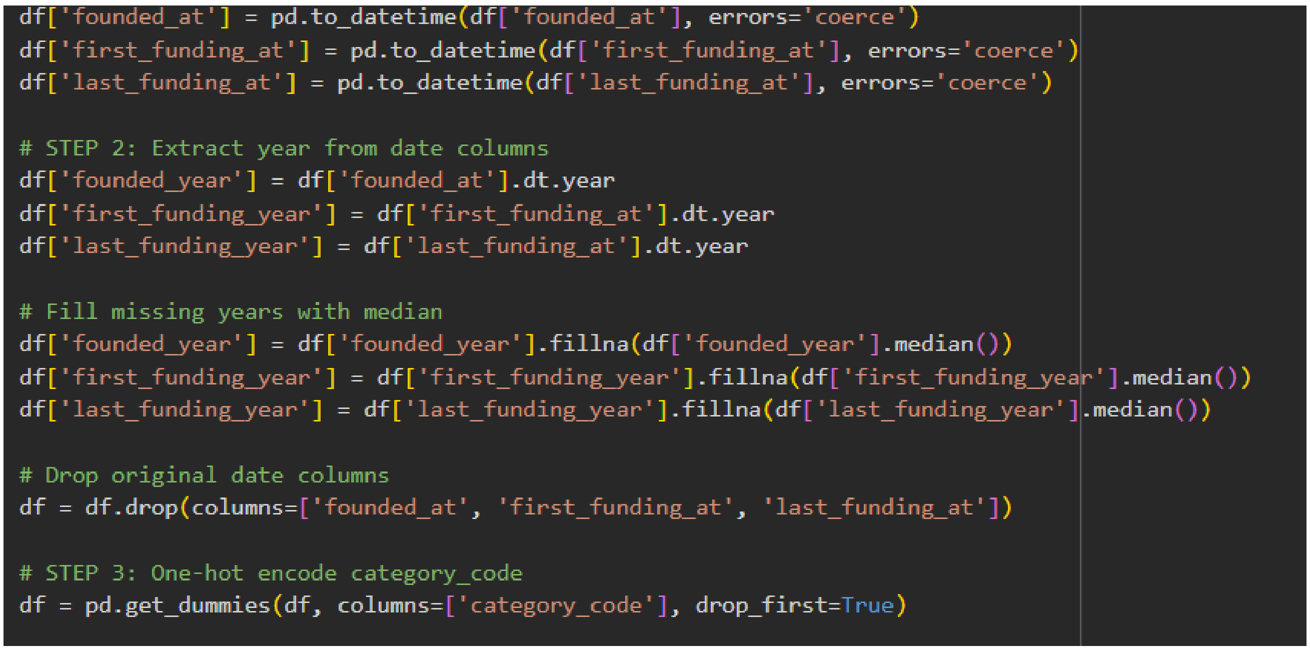
Activity 2: Handling Categorical Values

As we can see our dataset has date columns, we must convert the date columns to **datetime format** first and then extract relevant features such as year. To extract the year from the date columns we use dt.yearfunction.

- In our project ,founded_at,first_funding_at,last_funding_at, are converted to datetime and then the years are extracted.

- Missing years are filled with **median values** to handle nulls properly.

- Original date columns are then dropped after feature extraction.

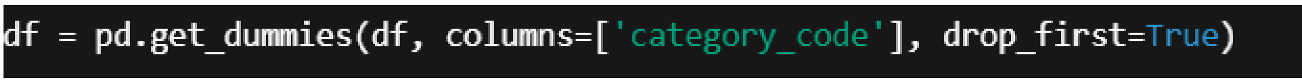


```
df['founded_at'] = pd.to_datetime(df['founded_at'], errors='coerce')
df['first_funding_at'] = pd.to_datetime(df['first_funding_at'], errors='coerce')
df['last_funding_at'] = pd.to_datetime(df['last_funding_at'], errors='coerce')

# STEP 2: Extract year from date columns
df['founded_year'] = df['founded_at'].dt.year
df['first_funding_year'] = df['first_funding_at'].dt.year
df['last_funding_year'] = df['last_funding_at'].dt.year

# Fill missing years with median
df['founded_year'] = df['founded_year'].fillna(df['founded_year'].median())
df['first_funding_year'] = df['first_funding_year'].fillna(df['first_funding_year'].median())
df['last_funding_year'] = df['last_funding_year'].fillna(df['last_funding_year'].median())

# Drop original date columns
df = df.drop(columns=['founded_at', 'first_funding_at', 'last_funding_at'])

# STEP 3: One-hot encode category_code
df = pd.get_dummies(df, columns=['category_code'], drop_first=True)
```

Activity 3: Balancing the dataset:

atOur dataset has categorical data in `category_code` which must be converted to numerical features. We are using **one-hot encoding** with `pd.get_dummies()` to transform categorical features into numerical features for modeling.

- In our project, `category_code` column is one-hot encoded, dropping the first category to avoid dummy variable trap.



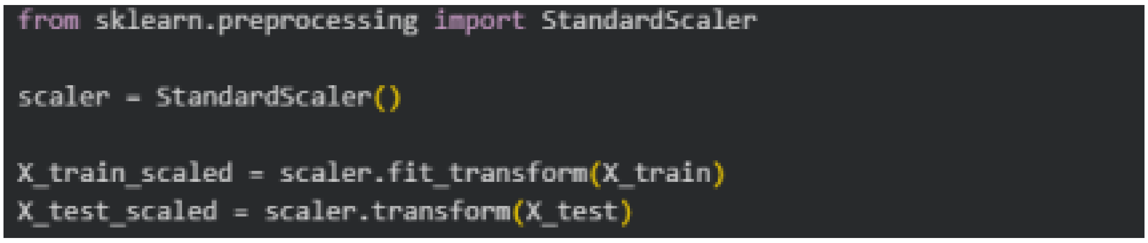
```
df = pd.get_dummies(df, columns=['category_code'], drop_first=True)
```

From the above picture, we can infer that ,previously our dataset is having 492 class 1, and 192 class items, after applying smote technique on the dataset the size has been changed for minority class.

Activity 4: Scaling the Data

Scaling is one the important process, we have to perform on the dataset, because of data measures in different ranges can leads to mislead in prediction

Models such as SVM, Logistic regression need scaled data, as they follow distance based method and Gradient Descent concept.



```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

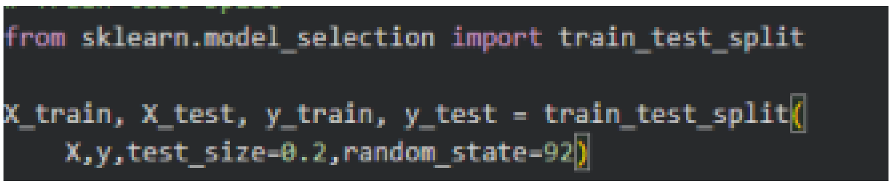
We will perform scaling only on the input values
Once the dataset is scaled, it will be converted into array and we need to convert it back to dataframe.

Activity : Splitting data into train and test

Now let’s split the Dataset into train and test sets

Changes: first split the dataset into x and y and then split the data set

Here x and y variables are created. On x variable, df is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.



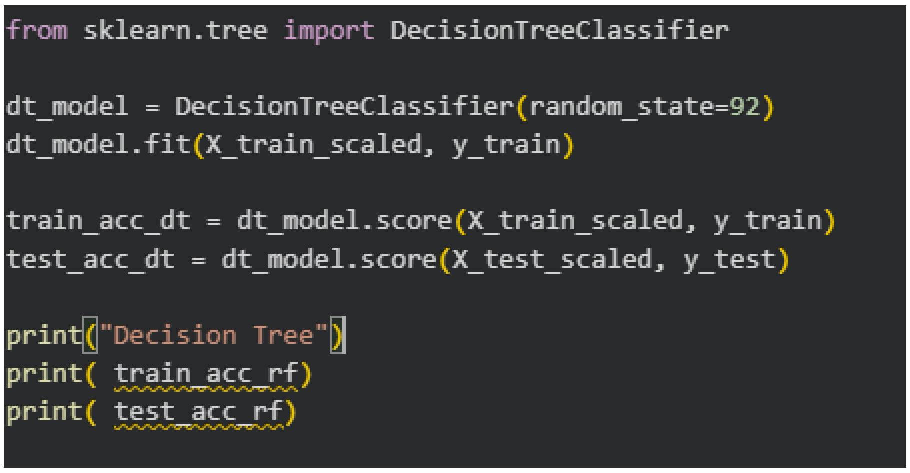
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X,y,test_size=0.2,random_state=92)
```

Milestone 4: Model Building

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. for this project we are applying four classification algorithms. The best model is saved based on its performance.

Activity 1: Decision tree model

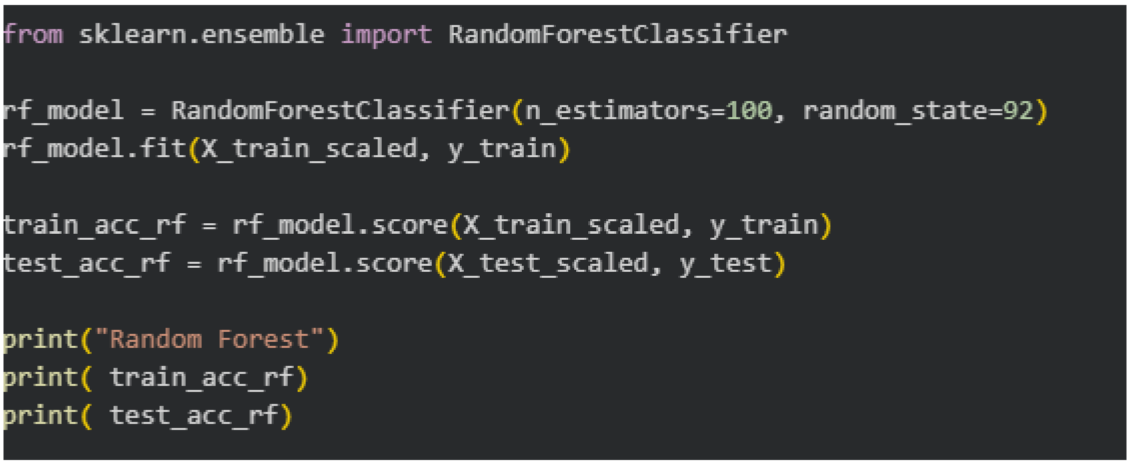
A function named decisionTree is created and train and test data are passed as the parameters. Inside the function, DecisionTreeClassifier algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable.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



```
from sklearn.tree import DecisionTreeClassifier

dt_model = DecisionTreeClassifier(random_state=92)
dt_model.fit(X_train_scaled, y_train)

train_acc_dt = dt_model.score(X_train_scaled, y_train)
test_acc_dt = dt_model.score(X_test_scaled, y_test)

print("Decision Tree")
print( train_acc_rf)
print( test_acc_rf)
```

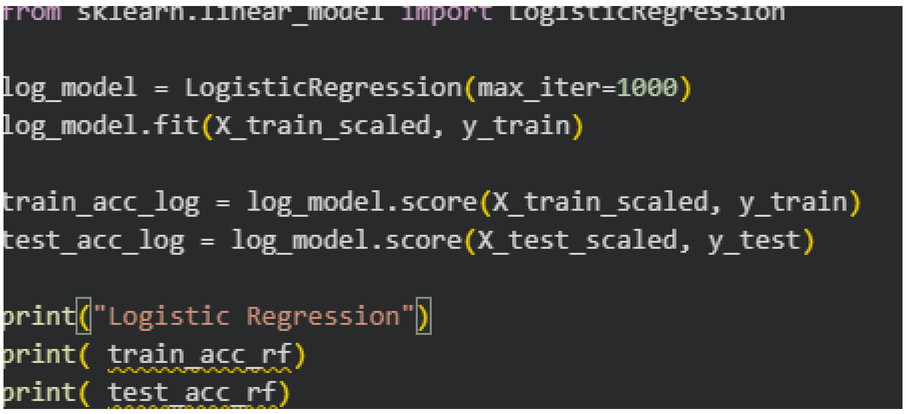
Activity 2: Random forest model

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.


```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(n_estimators=100, random_state=92)
rf_model.fit(X_train_scaled, y_train)

train_acc_rf = rf_model.score(X_train_scaled, y_train)
test_acc_rf = rf_model.score(X_test_scaled, y_test)

print("Random Forest")
print( train_acc_rf)
print( test_acc_rf)
```

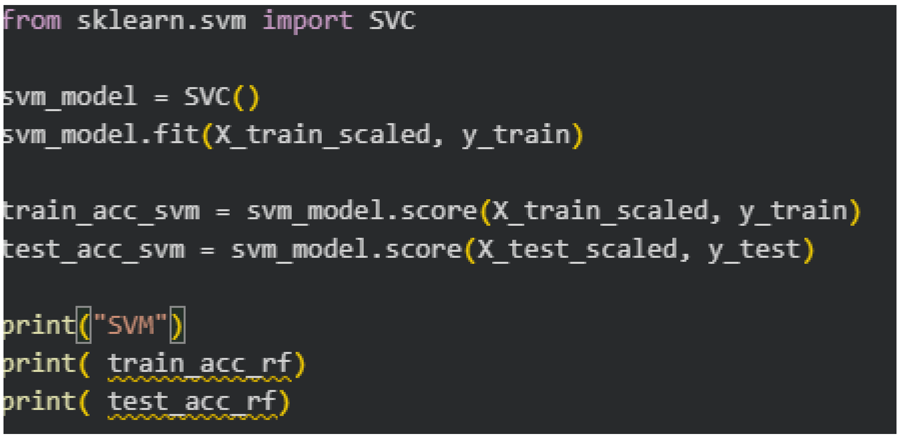
Activity 3: Logistic Regression model

A function named Logistic Regression is created and the train and test data are passed as parameters. The LogisticRegression algorithm is initialized and trained using the `.fit()` function. The test data is predicted using the `.predict()` function. Finally, the model is evaluated using a confusion matrix and classification report.


```
from sklearn.linear_model import LogisticRegression

log_model = LogisticRegression(max_iter=1000)
log_model.fit(X_train_scaled, y_train)

train_acc_log = log_model.score(X_train_scaled, y_train)
test_acc_log = log_model.score(X_test_scaled, y_test)

print("Logistic Regression")
print( train_acc_rf)
print( test_acc_rf)
```

Activity 4: SVM model

A function named SVM is created and the train and test data are passed as parameters. Inside the function, the Support Vector Machine (SVC) algorithm is initialized and the training data is fitted using the `.fit()` function. The test data is predicted using the `.predict()` function and stored in a new variable. Finally, the model is evaluated using a confusion matrix and classification report.

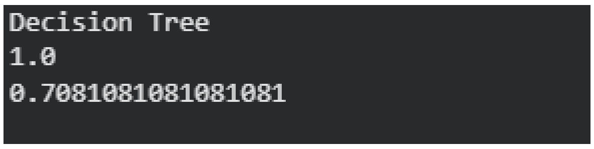


```
from sklearn.svm import SVC

svm_model = SVC()
svm_model.fit(X_train_scaled, y_train)

train_acc_svm = svm_model.score(X_train_scaled, y_train)
test_acc_svm = svm_model.score(X_test_scaled, y_test)

print("SVM")
print( train_acc_rf)
print( test_acc_rf)
```

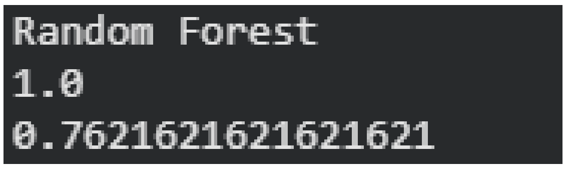
Now let's see the performance of all the models and save the best model

Activity 5: Compare the model

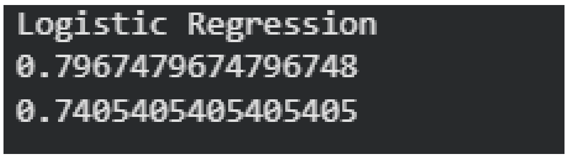
For comparing the above four models compare Modelfunction is defined.



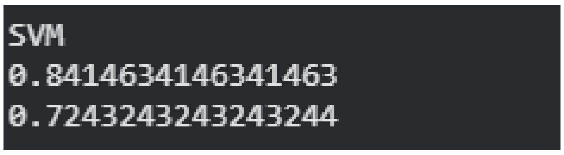
```
Decision Tree
1.0
0.7081081081081081
```



```
Random Forest
1.0
0.7621621621621621
```



```
Logistic Regression
0.7967479674796748
0.7405405405405405
```

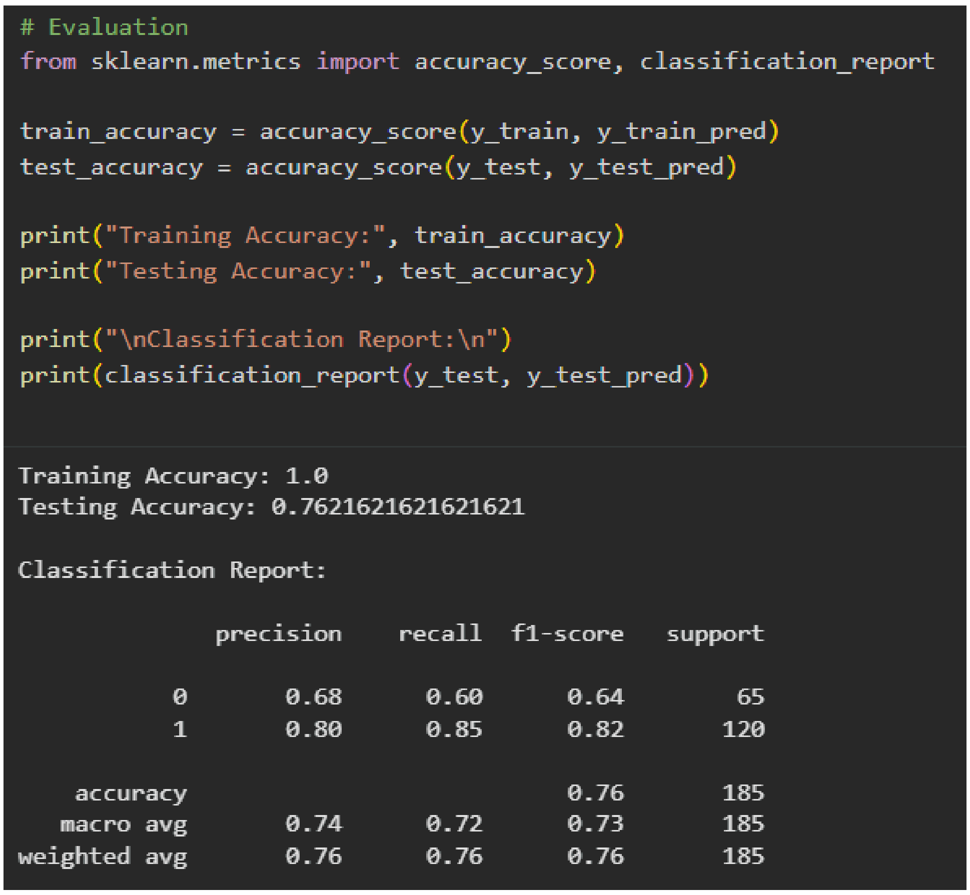


```
SVM
0.8414634146341463
0.7243243243243244
```

After calling the function, the results of models are displayed as output. From the four model RandomForest is performing well. From the below image, We can see the accuracy of the model. RandomForest is giving the accuracy of 100% with training data , 76% accuracy for the testing data.so we considering RandomForest and deploy this model.

Activity 6: Evaluating performance of the model and saving the model

To evaluate the performance of the model, accuracy_score and classification_report are imported from sklearn.metrics. The training accuracy is calculated using the predicted values of the training dataset, and the testing accuracy is calculated using the predicted values of the testing dataset. Both training and testing accuracy scores are printed to analyze the model's performance. Finally, a classification report is generated to evaluate precision, recall, f1-score, and support for the test dataset.



```
# Evaluation
from sklearn.metrics import accuracy_score, classification_report

train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)

print("Training Accuracy:", train_accuracy)
print("Testing Accuracy:", test_accuracy)

print("\nClassification Report:\n")
print(classification_report(y_test, y_test_pred))
```

```
Training Accuracy: 1.0
Testing Accuracy: 0.7621621621621621

Classification Report:

              precision    recall  f1-score   support

           0       0.68      0.60      0.64        65
           1       0.80      0.85      0.82       120

    accuracy                           0.76       185
   macro avg       0.74      0.72      0.73       185
weighted avg       0.76      0.76      0.76       185
```

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

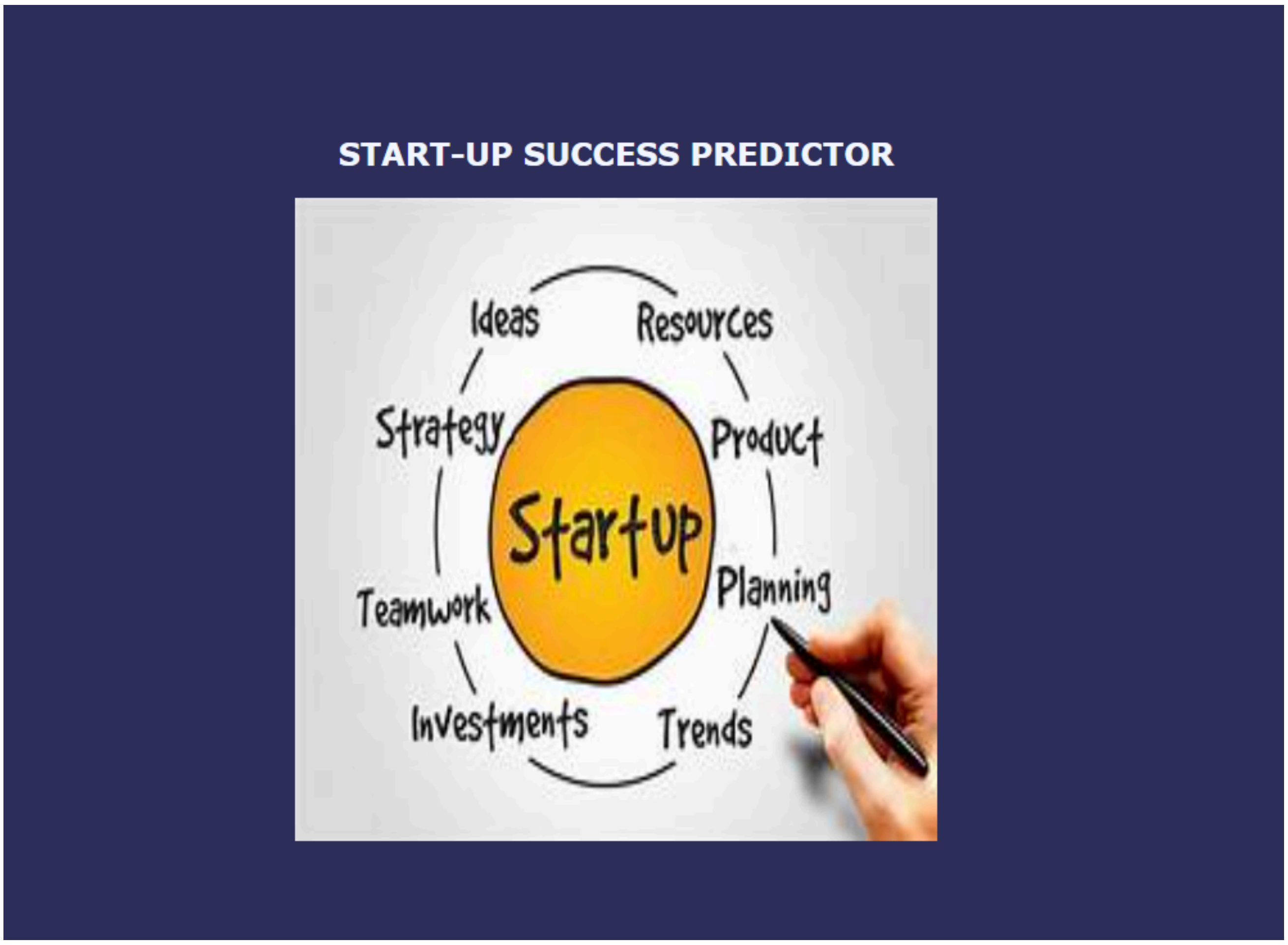
- Building HTML Pages
- Building serverside script

Activity1: Building Html Pages:

For this project create three HTML files namely

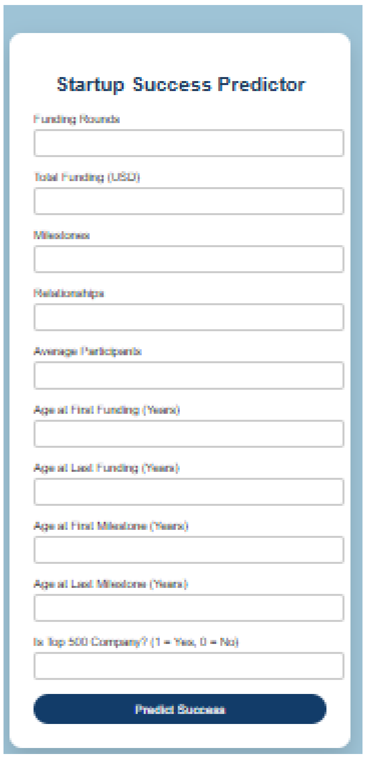
- index.html
- predict.html
- results.html

and save them in templates folder.



Let's see how our home.html page looks like:
Now when you click on predict button from top right corner you will get redirected to predict.html

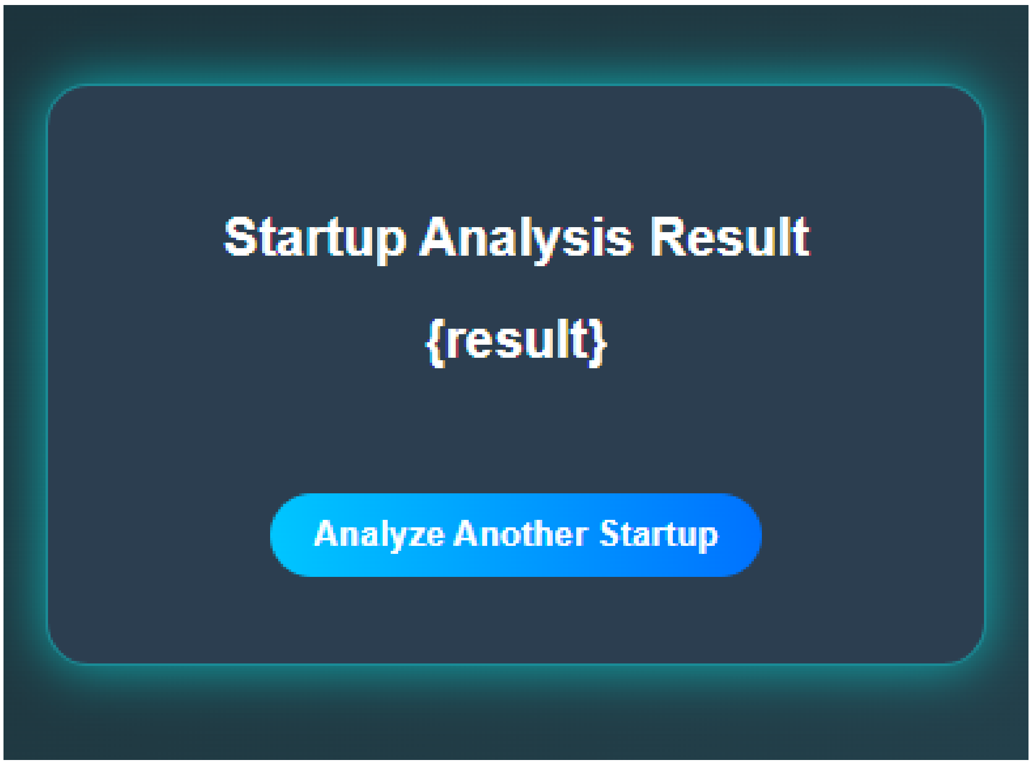
Lets look how our predict.html file looks like:



Startup Success Predictor
Funding Rounds
Total Funding (USD)
Milestones
Relationships
Average Participants
Age at First Funding (Years)
Age at Last Funding (Years)
Age at First Milestone (Years)
Age at Last Milestone (Years)
Is Top 500 Company? (1 = Yes, 0 = No)
Predict Success

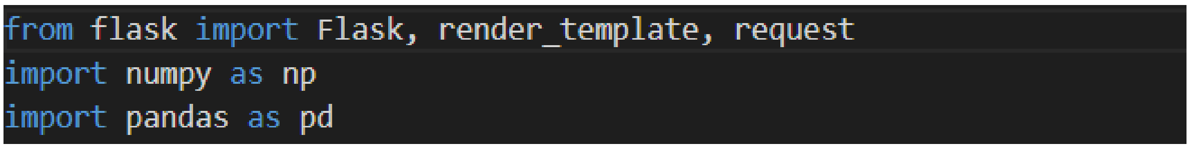
Now when you click on submit button from left bottom corner you will get redirected to submit.html

Lets look how our results.html file looks like:



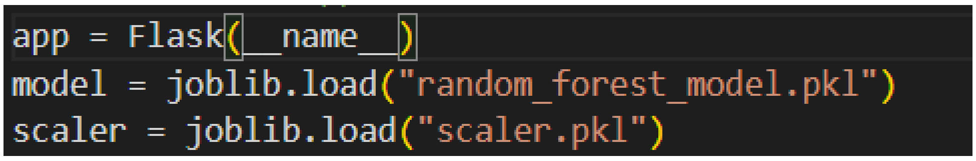
Activity 2: Build Python code:

Import the libraries



```python
from flask import Flask, render_template, request
import numpy as np
import pandas as pd
```

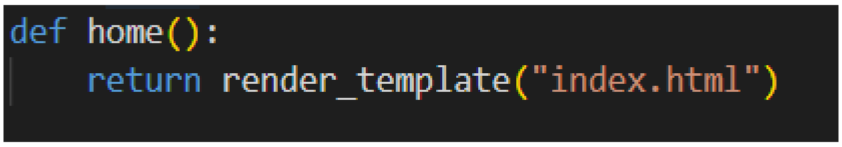
Load the saved model. Importing flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument



```python
app = Flask(__name__)
model = joblib.load("random_forest_model.pkl")
scaler = joblib.load("scaler.pkl")
```

.

Render HTML page:

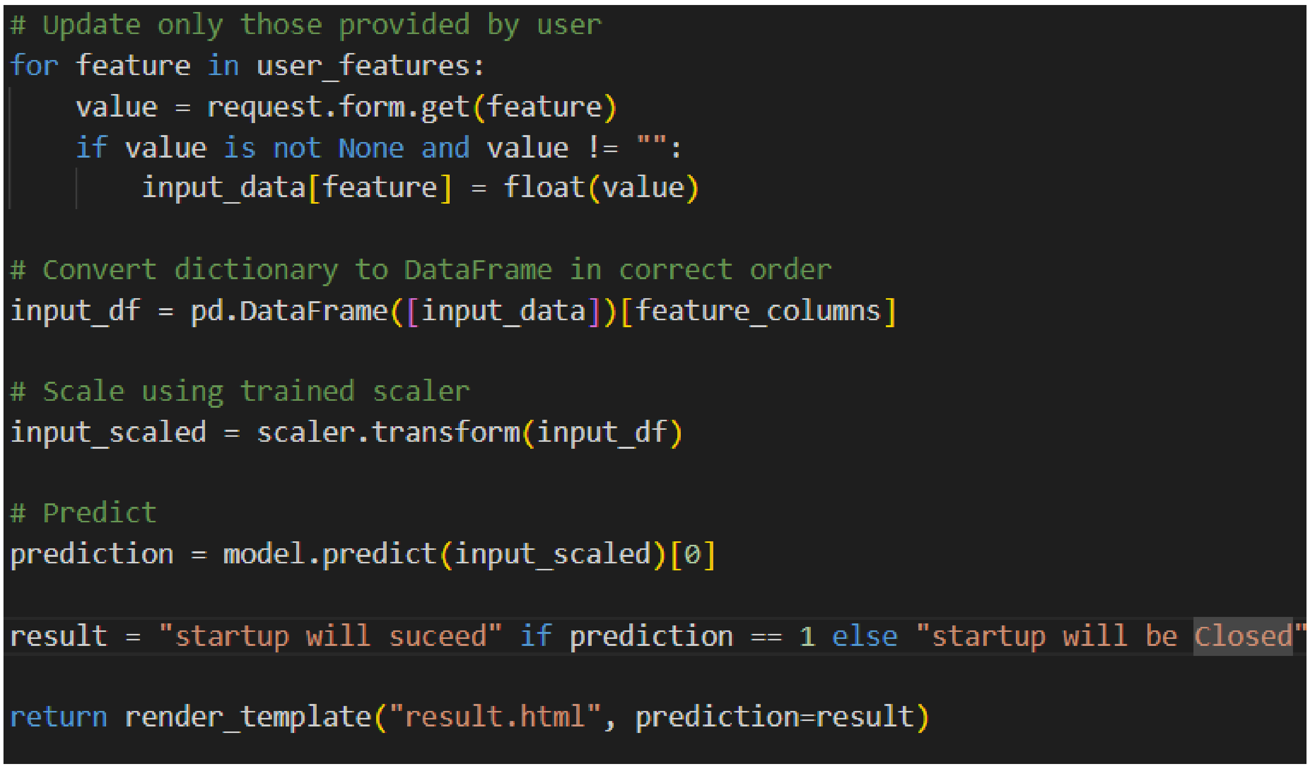


```python
def home():
    return render_template("index.html")
```

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 index.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.

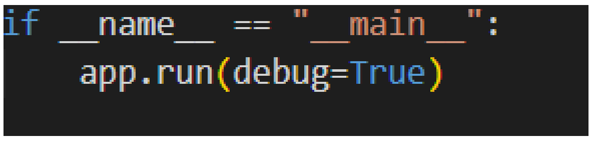
Retrieves the value from UI:



```python
# Update only those provided by user
for feature in user_features:
    value = request.form.get(feature)
    if value is not None and value != "":
        input_data[feature] = float(value)

# Convert dictionary to DataFrame in correct order
input_df = pd.DataFrame([input_data])[feature_columns]

# Scale using trained scaler
input_scaled = scaler.transform(input_df)

# Predict
prediction = model.predict(input_scaled)[0]

result = "startup will suceed" if prediction == 1 else "startup will be Closed"

return render_template("result.html", prediction=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 results.html page earlier.

Main Function:

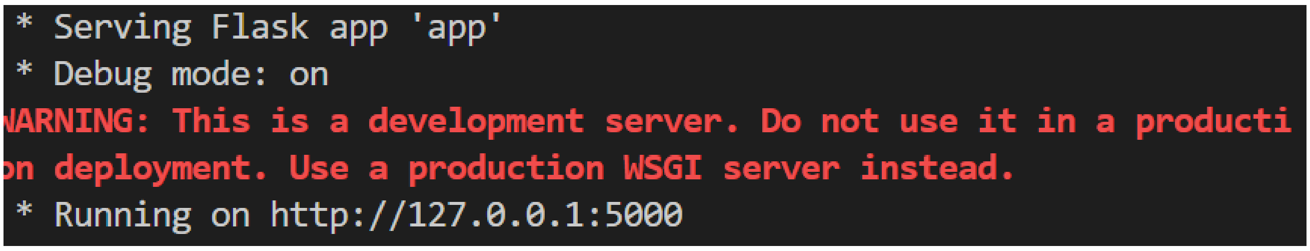


```python
if __name__ == "__main__":
    app.run(debug=True)
```

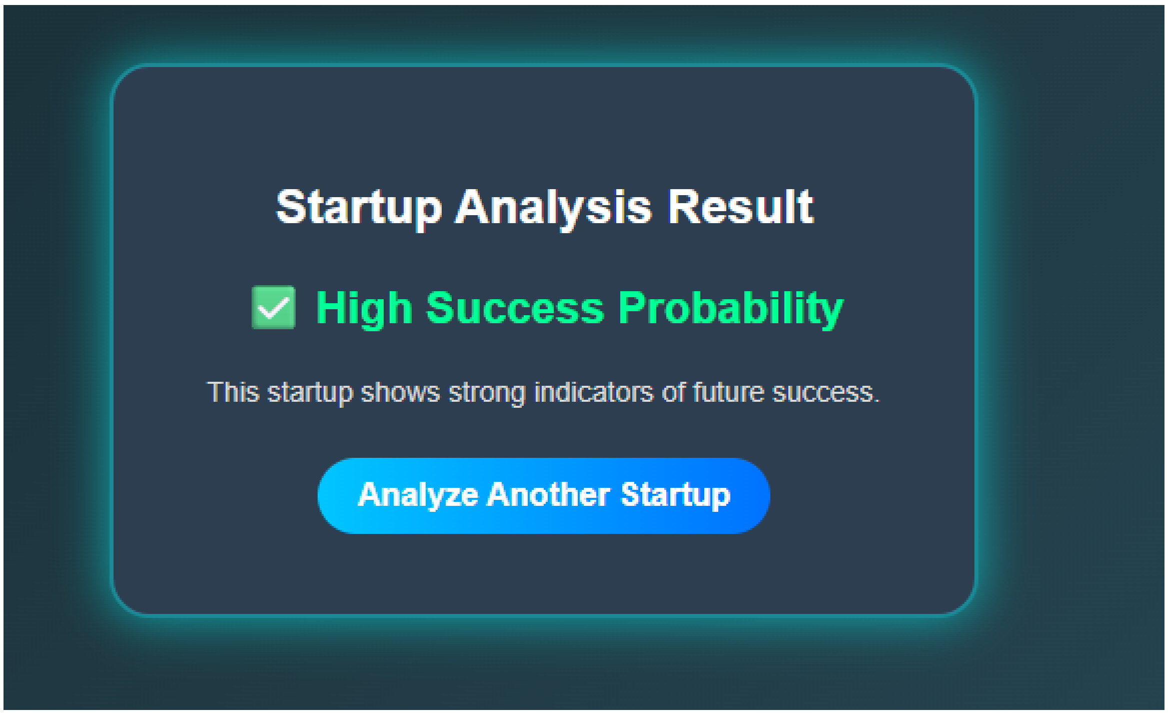
Activity 3: Run the application

- Open Visual Studio Code from the Start menu.
- Open the folder where your Python script is located using File → Open Folder.
- Open the integrated terminal and type **python app.py** to run the application.

- Navigate to the localhost where you can view your web page.
- Click on the predict button from top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.



```
 * Serving Flask app 'app'
 * Debug mode: on
ARNING: This is a development server. Do not use it in a producti
on deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
```



Startup Analysis Result

✅ High Success Probability

This startup shows strong indicators of future success.