Project Development Phase

Project Manual

Date	5 November 2023
Team ID	Team - 593188
Project Name	STARTUP PROPHET: HARNESSING AI TO DIVINE THE FUTURE OF STARTUP SUCCESS

Milestone 1: Data Collection

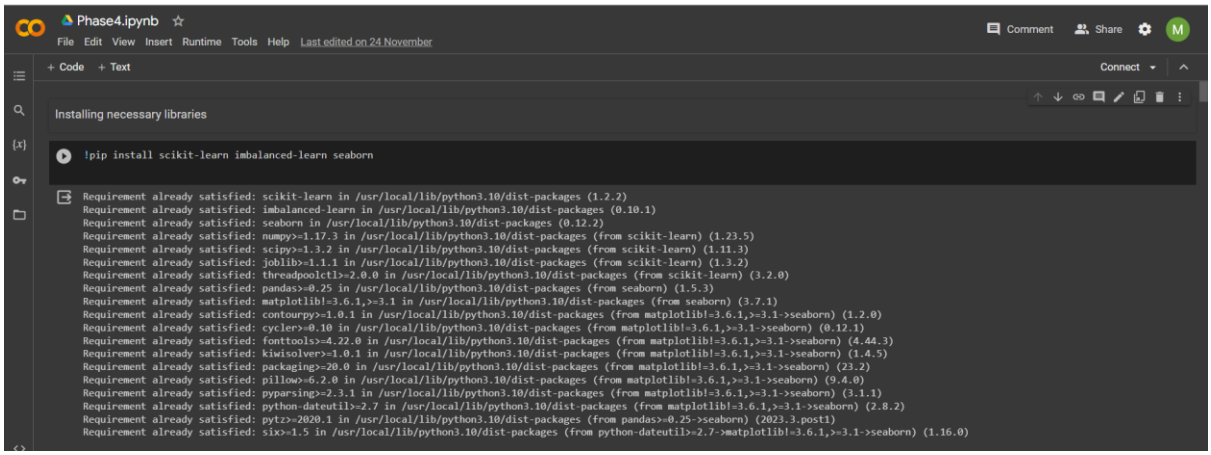
ML depends heavily on data, It is most crucial aspect that makes algorithm training possible.

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.

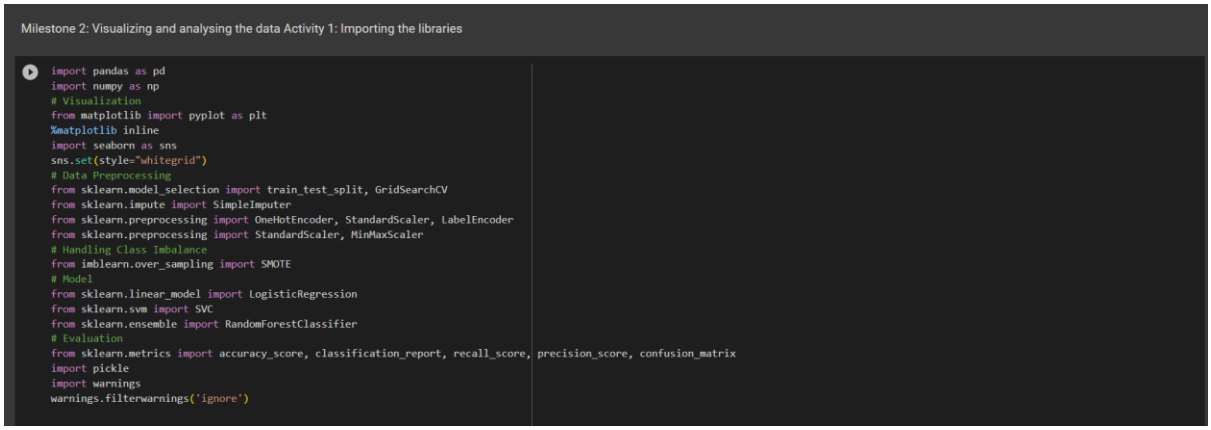
Eg: kaggle.com, UCI repository, etc.



Milestone 2: Visualizing and analyzing 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.

Activity 1: Importing the libraries



Activity 2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt , 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 name of csv file.

Activity 2: Read the Dataset

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

	Unnamed: 0	state_code	latitude	longitude	zip_code	id	city	Unnamed: 6	name	labels	...	object_id	has_VC	has_angel	has_roundA	has_roundB	has_roundC	has_roundD	avg_participants	is_top500	status
0	1005	CA	42.358880	-71.056820	92101	c:6669	San Diego	NaN	Bandsintown	1	...	c:6669	0	1	0	0	0	0	1.0000	0	acquired
1	204	CA	37.238916	-121.973718	95032	c:16283	Los Gatos	NaN	TriCipher	1	...	c:16283	1	0	0	1	1	1	4.7500	1	acquired
2	1001	CA	32.901049	-117.192656	92121	c:65620	San Diego	San Diego CA 92121	Plixi	1	...	c:65620	0	0	1	0	0	0	4.0000	1	acquired
3	738	CA	37.320309	-122.050040	95014	c:42668	Cupertino	Cupertino CA 95014	Solidcore Systems	1	...	c:42668	0	0	0	1	1	1	3.3333	1	acquired
4	1002	CA	37.779281	-122.419236	94105	c:65806	San Francisco	San Francisco CA 94105	Inhale Digital	0	...	c:65806	1	1	0	0	0	0	1.0000	1	closed
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
918	352	CA	37.740594	-122.376471	94107	c:21343	San Francisco	NaN	CoTweet	1	...	c:21343	0	0	1	0	0	0	6.0000	1	acquired
919	721	MA	42.504817	-71.195611	1803	c:41747	Burlington	Burlington MA 1803	Reef Point Systems	0	...	c:41747	1	0	0	1	0	0	2.6667	1	closed
920	557	CA	37.408261	-122.015920	94089	c:31549	Sunnyvale	NaN	Paracor Medical	0	...	c:31549	0	0	0	0	0	1	8.0000	1	closed
921	589	CA	37.556732	-122.288378	94404	c:33198	San Francisco	NaN	Causata	1	...	c:33198	0	0	1	1	0	0	1.0000	1	acquired
922	462	CA	37.386778	-121.966277	95054	c:26702	Santa Clara	Santa Clara CA 95054	Asempra Technologies	1	...	c:26702	0	0	0	1	0	0	3.0000	1	acquired

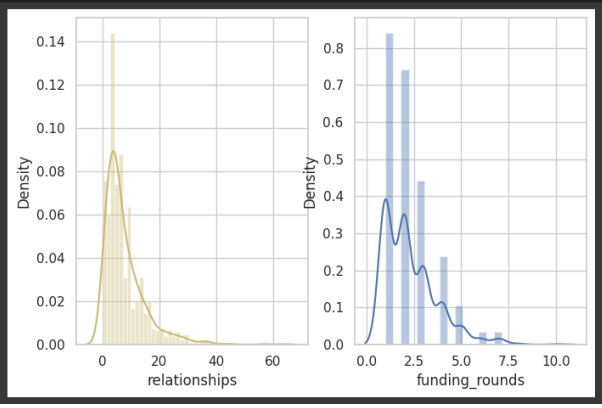
923 rows × 49 columns

Activity 3: Univariate analysis

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

Activity 3: Univariate analysis

```
[ ] plt.figure(figsize=(12,5))
    plt.subplot(131)
    sns.distplot(df["relationships"],color="y")
    plt.subplot(132)
    sns.distplot(df["funding_rounds"])
    plt.show()
```



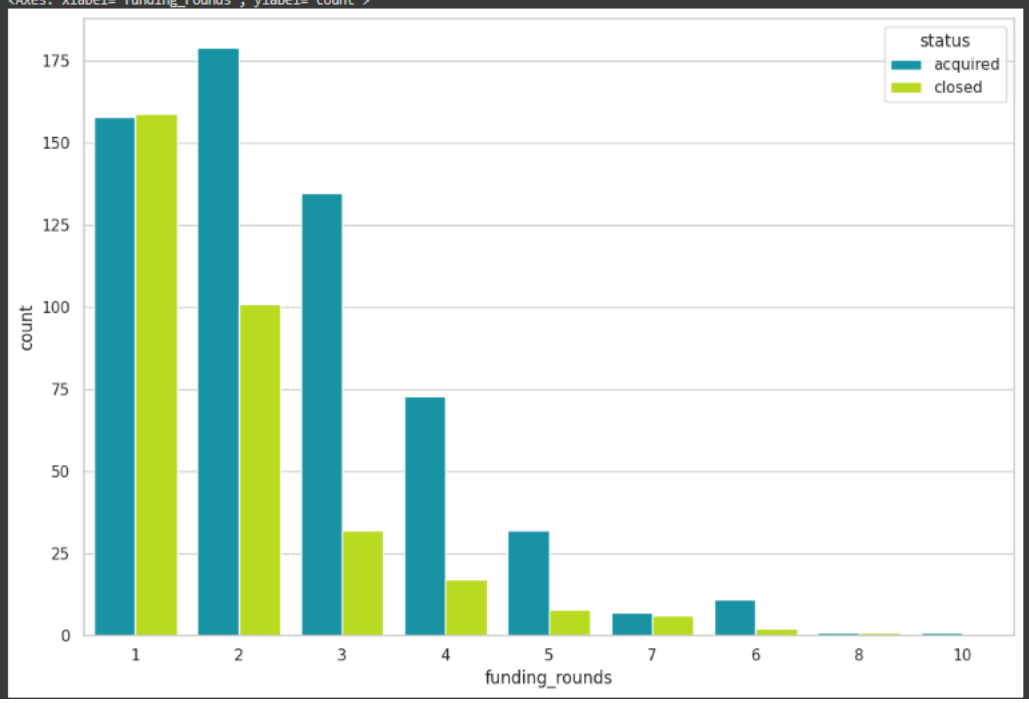
Activity 4: Bivariate analysis

To find the relation between two features we use bivariate analysis. Here we are visualizing the relationship between two features.

Activity 4: Bivariate analysis

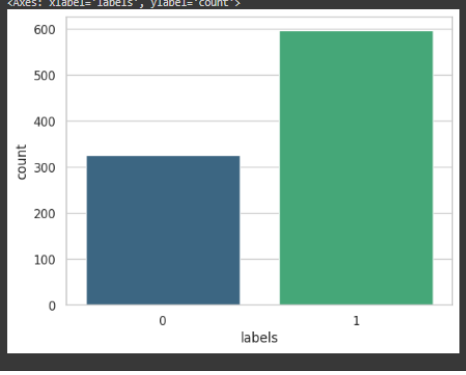
```
[ ] fig, ax = plt.subplots(figsize=(12,8))
    sns.countplot(x="funding_rounds", hue="status", data=df, palette="nipy_spectral", order=df.funding_rounds.value_counts().index)
    #plt. Legend(bbox_to_anchor=(0.945, 0.90))
```

```
<Axes: xlabel='funding_rounds', ylabel='count'>
```



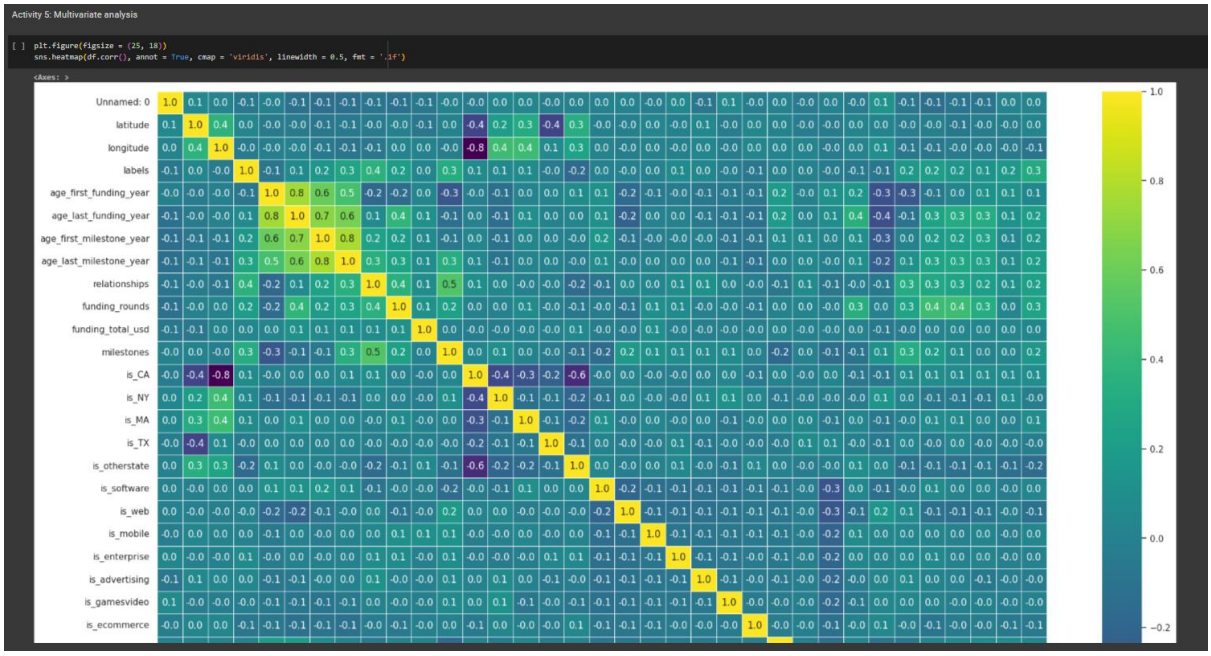

```
[ ] sns.countplot(x = df['labels'], palette = 'viridis')
```

```
<Axes: xlabel='labels', ylabel='count'>
```



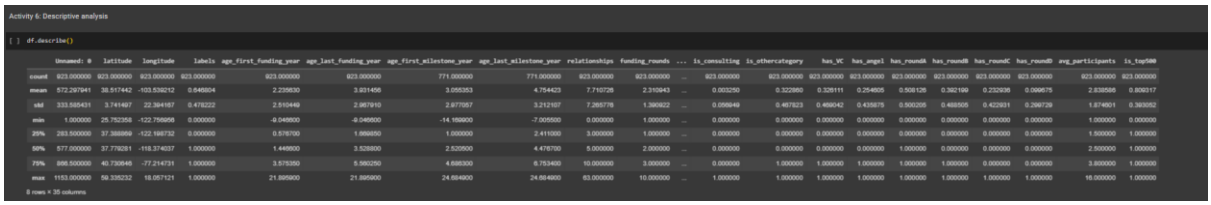
Activity 5: Multivariate analysis

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



Activity 6: Descriptive analysis

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe.



Milestone 3: Data Pre-processing

As we have understood how the data is lets pre-process the collected data.

Activity 1: Checking for null values and drop unwanted cols

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.

Milestone 3: Data Pre-processing

Activity 1: Checking for null values and drop unwanted cols

```
[ ] df.shape
```

```
(923, 49)
```

```
[ ] df.info
```

```
<bound method DataFrame.info of      Unnamed: 0 state_code   latitude   longitude zip_code       id  \
0          1005         CA  42.358880  -71.056820    92101   c:6669
1           204         CA  37.210916 -121.973718    95032  c:16283
2          1001         CA  32.901049 -117.192656    92121  c:65620
3           738         CA  37.320309 -122.050040    95014  c:42668
4          1002         CA  37.779281 -122.419236    94105  c:65806
..          ...        ...        ...        ...      ...      ...
918         352         CA  37.740594 -122.376471    94107  c:21343
919         721         MA  42.504817  -71.195611     1803  c:41747
920         557         CA  37.408261 -122.015920    94089  c:31549
921         589         CA  37.556732 -122.288378    94404  c:33198
922         462         CA  37.360778 -121.966277    95054  c:26702

                city              Unnamed: 6                 name  labels  ...  \
0          San Diego                     NaN           Bandsintown       1  ...
1          Los Gatos                     NaN             TriCipher       1  ...
2          San Diego       San Diego CA 92121                 Plixi       1  ...
3          Cupertino       Cupertino CA 95014      Solidcore Systems       1  ...
4     San Francisco  San Francisco CA 94105         Inhale Digital       0  ...
..              ...                     ...                    ...     ...  ...
918   San Francisco                     NaN               CoTweet       1  ...
919       Burlington      Burlington MA 1803     Reef Point Systems       0  ...
920        Sunnyvale                     NaN          Paracor Medical       0  ...
921   San Francisco                     NaN                 Causata       1  ...
922     Santa Clara    Santa Clara CA 95054  Asempra Technologies       1  ...

     object_id has_VC has_angel has_roundA  has_roundB  has_roundC  has_roundD  \
0     c:6669      0        1          0           0           0           0
1    c:16283      1        0          0           1           1           1
2    c:65620      0        0          1           0           0           0
3    c:42668      0        0          0           1           1           1
4    c:65806      1        1          0           0           0           0
..       ...    ...      ...        ...         ...         ...         ...
918  c:21343      0        0          1           0           0           0
919  c:41747      1        0          0           1           0           0
920  c:31549      0        0          0           0           0           1
921  c:33198      0        0          1           1           0           0
922  c:26702      0        0          0           1           0           0

     avg_participants  is_top500    status
0              1.0000          0  acquired
1              4.7500          1  acquired
2              4.0000          1  acquired
3              3.3333          1  acquired
4              1.0000          1    closed
..                ...        ...       ...
918            6.0000          1  acquired
919            2.6667          1    closed
920            8.0000          1    closed
921            1.0000          1  acquired
922            3.0000          1  acquired

[923 rows x 49 columns]>
```

```
[ ] df.isna().sum()
```

```
Unnamed: 0                     0
state_code                     0
latitude                       0
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
relationships                  0
funding_rounds                 0
funding_total_usd              0
milestones                     0
state_code.1                   1
is_CA                          0
is_NY                          0
is_MA                          0
is_TX                          0
is_otherstate                  0
category_code                  0
is_software                    0
is_web                         0
is_mobile                      0
is_enterprise                  0
is_advertising                 0
is_gamesvideo                  0
is_ecommerce                   0
is_biotech                     0
is_consulting                  0
is_othercategory               0
object_id                      0
has_VC                         0
has_angel                      0
has_roundA                     0
has_roundB                     0
has_roundC                     0
has_roundD                     0
avg_participants               0
is_top500                      0
status                         0
dtype: int64
```

Activity 2: Handling Categorical Values

As we can see our dataset has no categorical data so no need to do anything .

```
[ ] df.shape

    (923, 49)
```

Activity 2: Handling Categorical Values

```
[ ] df.dtypes

    Unnamed: 0                   int64
    state_code                  object
    latitude                   float64
    longitude                  float64
    zip_code                    object
    id                          object
    city                        object
    Unnamed: 6                  object
    name                        object
    labels                       int64
    founded_at                  object
    closed_at                   object
    first_funding_at            object
    last_funding_at             object
    age_first_funding_year     float64
    age_last_funding_year      float64
    age_first_milestone_year   float64
    age_last_milestone_year    float64
    relationships                int64
    funding_rounds               int64
    funding_total_usd            int64
    milestones                   int64
    state_code.1                object
    is_CA                        int64
    is_NY                        int64
    is_MA                        int64
    is_TX                        int64
    is_otherstate                int64
    category_code               object
    is_software                  int64
    is_web                       int64
    is_mobile                    int64
    is_enterprise                int64
    is_advertising               int64
    is_gamesvideo                int64
    is_ecommerce                 int64
    is_biotech                   int64
    is_consulting                int64
    is_othercategory             int64
    object_id                   object
    has_VC                       int64
    has_angel                    int64
    has_roundA                   int64
    has_roundB                   int64
    has_roundC                   int64
    has_roundD                   int64
    avg_participants           float64
    is_top500                    int64
    status                      object
    dtype: object
```

```
import pandas as pd

# Assuming df is your DataFrame
# Extract numerical and categorical columns
numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns
categorical_cols = df.select_dtypes(include=['object']).columns

# Separate numerical and categorical data
X_numerical = df[numerical_cols]
X_categorical = df[categorical_cols]

# Display the numerical and categorical data
print("Numerical Data:")
print(X_numerical.head())

print("\nCategorical Data:")
print(X_categorical.head())
```

```
Numerical Data:
   Unnamed: 0   latitude   longitude  labels  age_first_funding_year  \
0        1005  42.358880  -71.056820       1                  2.2493
1         204  37.238916 -121.973718       1                  5.1260
2        1001  32.901049 -117.192656       1                  1.0329
3         738  37.320309 -122.050040       1                  3.1315
4        1002  37.779281 -122.419236       0                  0.0000

   age_last_funding_year  age_first_milestone_year  age_last_milestone_year  \
0                 3.0027                    4.6685                   6.7041
1                 9.9973                    7.0055                   7.0055
2                 1.0329                    1.4575                   2.2055
3                 5.3151                    6.0027                   6.0027
4                 1.6685                    0.0384                   0.0384

   relationships  funding_rounds  ...  is_consulting  is_othercategory  \
0              3               3  ...              0                 1
1              9               4  ...              0                 0
2              5               1  ...              0                 0
3              5               3  ...              0                 0
4              2               2  ...              0                 0

   has_VC  has_angel  has_roundA  has_roundB  has_roundC  has_roundD  \
0       0          1           0           0           0           0
1       1          0           0           1           1           1
2       0          0           1           0           0           0
3       0          0           0           1           1           1
4       1          1           0           0           0           0

   avg_participants  is_top500
0            1.0000          0
1            4.7500          1
2            4.0000          1
3            3.3333          1
4            1.0000          1

[5 rows x 35 columns]

Categorical Data:
  state_code zip_code       id           city             Unnamed: 6  \
0         CA    92101  c:6669      San Diego                    NaN
1         CA    95032 c:16283      Los Gatos                    NaN
2         CA    92121 c:65620      San Diego      San Diego CA 92121
3         CA    95014 c:42668      Cupertino      Cupertino CA 95014
4         CA    94105 c:65806  San Francisco  San Francisco CA 94105

                name founded_at  closed_at first_funding at last_funding at  \
0        Bandsintown   1/1/2007        NaN         4/1/2009        1/1/2010
1          TriCipher   1/1/2000        NaN        2/14/2005       12/28/2009
2               Plixi  3/18/2009        NaN        3/30/2010        3/30/2010
3  Solidcore Systems   1/1/2002        NaN        2/17/2005        4/25/2007
4     Inhale Digital   8/1/2010  10/1/2012         8/1/2010         4/1/2012

  state_code.1 category_code object_id    status
0           CA         music    c:6669  acquired
1           CA    enterprise   c:16283  acquired
2           CA           web   c:65620  acquired
3           CA      software   c:42668  acquired
4           CA  games_video   c:65806    closed
```

Activity 3: Scaling Techniques

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.

```
import pandas as pd
from sklearn.impute import SimpleImputer

# Assuming you have a DataFrame named df

# Separate numerical and categorical columns
numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns
categorical_cols = df.select_dtypes(include=['object']).columns

# Impute missing values for numerical columns
numeric_imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median' or 'most_frequent'
df[numeric_cols] = numeric_imputer.fit_transform(df[numeric_cols])

# Impute missing values for categorical columns
categorical_imputer = SimpleImputer(strategy='most_frequent')  # You can use other strategies like 'constant' with fill_value='missing'
df[categorical_cols] = categorical_imputer.fit_transform(df[categorical_cols])
```

Activity 3: Scaling Techniques

```
[ ] X = df.drop(['labels'], axis = 1)
    y = df['labels']
```

```
[ ] X
```

	Unnamed: 0	state_code	latitude	longitude	zip_code	id	city	Unnamed: 6	name	founded_at	...	object_id	has_VC	has_angel	has_roundA	has_roundB	has_roundC	has_roundD	avg_participants	is_top500	status
0	1005.0	CA	42.358880	-71.056820	92101	c:6669	San Diego	San Francisco CA 94105	Bandsintown	1/1/2007	...	c:6669	0.0	1.0	0.0	0.0	0.0	0.0	1.0000	0.0	acquired
1	204.0	CA	37.238916	-121.973718	95032	c:16283	Los Gatos	San Francisco CA 94105	TriCipher	1/1/2000	...	c:16283	1.0	0.0	0.0	1.0	1.0	1.0	4.7500	1.0	acquired
2	1001.0	CA	32.901049	-117.192656	92121	c:65620	San Diego	San Diego CA 92121	Plixi	3/18/2009	...	c:65620	0.0	0.0	1.0	0.0	0.0	0.0	4.0000	1.0	acquired
3	738.0	CA	37.320309	-122.050040	95014	c:42668	Cupertino	Cupertino CA 95014	Solidcore Systems	1/1/2002	...	c:42668	0.0	0.0	0.0	1.0	1.0	1.0	3.3333	1.0	acquired
4	1002.0	CA	37.779281	-122.419236	94105	c:65806	San Francisco	San Francisco CA 94105	Inhale Digital	8/1/2010	...	c:65806	1.0	1.0	0.0	0.0	0.0	0.0	1.0000	1.0	closed
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
918	352.0	CA	37.740594	-122.376471	94107	c:21343	San Francisco	San Francisco CA 94105	CoTweet	1/1/2009	...	c:21343	0.0	0.0	1.0	0.0	0.0	0.0	6.0000	1.0	acquired
919	721.0	MA	42.504817	-71.195611	1803	c:41747	Burlington	Burlington MA 1803	Reef Point Systems	1/1/1998	...	c:41747	1.0	0.0	0.0	1.0	0.0	0.0	2.6667	1.0	closed
920	557.0	CA	37.408261	-122.015020	94089	c:31549	Sunnyvale	San Francisco CA 94105	Paracor Medical	1/1/1999	...	c:31549	0.0	0.0	0.0	0.0	0.0	1.0	8.0000	1.0	closed
921	589.0	CA	37.556732	-122.288378	94404	c:33198	San Francisco	San Francisco CA 94105	Causata	1/1/2009	...	c:33198	0.0	0.0	1.0	1.0	1.0	0.0	1.0000	1.0	acquired
922	462.0	CA	37.388778	-121.966277	95054	c:26702	Santa Clara	Santa Clara CA 95054	Asempra Technologies	1/1/2003	...	c:26702	0.0	0.0	0.0	1.0	0.0	0.0	3.0000	1.0	acquired

923 rows × 48 columns

```
[ ] x=pd.DataFrame(X)
```

Activity 3: Handling Class Imbalance

Our data is imbalance so by using SMOTE technique to balance the data . SMOTE (Synthetic Minority Over-sampling Technique) is a technique used to address class imbalance in machine learning datasets, particularly in classification tasks. It aims to balance the distribution of classes by generating synthetic samples for the minority class. SMOTE creates these synthetic samples by interpolating between existing instances of the minority class.

Activity 4: Splitting data into train and test

After the SMOTE our data is balance now,For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing x_bal, y_bal, test_size, random_state

Activity 3: Handling Class Imbalance

```
[ ] from sklearn.impute import SimpleImputer
    from imblearn.over_sampling import SMOTE
    from sklearn.model_selection import train_test_split

    # Assuming you have a DataFrame named df with 'labels' as your target column
    # Assume 'labels' is binary (0 or 1)

    # Extract numeric columns
    numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns
    X_numeric = df[numeric_cols]

    # Impute missing values
    imputer = SimpleImputer(strategy='mean')
    X_numeric_imputed = imputer.fit_transform(X_numeric)

    # Extract the target variable
    y = df['labels']

    # Split the dataset into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X_numeric_imputed, y, test_size=0.3, random_state=42)

    # Apply SMOTE to balance the classes using only the numeric features
    smote = SMOTE(sampling_strategy='auto', random_state=42)
    X_train_bal, y_train_bal = smote.fit_resample(X_train, y_train)

    # Print the shapes to verify the balance
    print("Before SMOTE - Class 0 count:", sum(y_train == 0), "Class 1 count:", sum(y_train == 1))
    print("After SMOTE - Class 0 count:", sum(y_train_bal == 0), "Class 1 count:", sum(y_train_bal == 1))

    Before SMOTE - Class 0 count: 226 Class 1 count: 420
    After SMOTE - Class 0 count: 420 Class 1 count: 420
```

Activity 4: Splitting data into train and test

```
[ ] from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

    # Create a Logistic Regression model
    logistic_model = LogisticRegression()
    logistic_model.fit(X_train, y_train)

    # Predict on the test set
    y_pred = logistic_model.predict(X_test)

    # Evaluate the model
    accuracy = accuracy_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)
    class_report = classification_report(y_test, y_pred)

    print("Accuracy:", accuracy)
    print("\nConfusion Matrix:\n", conf_matrix)
    print("\nClassification Report:\n", class_report)
```

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: Logistic Regression

We create a LogisticRegression model .The model is trained on the training data using the fit method.Predictions are made on the test data using the predict method

```
Accuracy: 0.63898916967509 02

Confusion Matrix:
 [[  0 100]
 [  0 177]]

Classification Report:
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00       100
         1.0       0.64      1.00      0.78       177

    accuracy                           0.64       277
   macro avg       0.32      0.50      0.39       277
weighted avg       0.41      0.64      0.50       277
```

Milestone 4: Model Building

Activity 1: Logistic Regression

```
[ ] from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

    # Create a Logistic Regression model
    logistic_model = LogisticRegression()
    logistic_model.fit(X_train, y_train)

    # Predict on the test set
    y_pred = logistic_model.predict(X_test)

    # Evaluate the model
    accuracy = accuracy_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)
    class_report = classification_report(y_test, y_pred)

    print("Accuracy:", accuracy)
    print("\nConfusion Matrix:\n", conf_matrix)
    print("\nClassification Report:\n", class_report)
```

```
Accuracy: 0.63898916967509 02

Confusion Matrix:
 [[  0 100]
 [  0 177]]

Classification Report:
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00       100
         1.0       0.64      1.00      0.78       177

    accuracy                           0.64       277
   macro avg       0.32      0.50      0.39       277
weighted avg       0.41      0.64      0.50       277
```

Activity 2: Support vector machine

We create code trains a Support Vector Machine (SVM) classifier with a radial basis function (RBF) kernel on a balanced dataset and then evaluates its performance using a confusion matrix and a classification report.

Activity 2: Support vector machine

```
[ ] from sklearn.svm import SVC
    from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

    # Create an SVM classifier with RBF kernel
    svm_classifier = SVC(kernel='rbf', random_state=42)

    # Train the SVM classifier on the balanced dataset
    svm_classifier.fit(X_train, y_train)

    # Predict on the test set
    y_pred_svm = svm_classifier.predict(X_test)

    # Evaluate the SVM classifier
    accuracy_svm = accuracy_score(y_test, y_pred_svm)
    conf_matrix_svm = confusion_matrix(y_test, y_pred_svm)
    class_report_svm = classification_report(y_test, y_pred_svm)

    # Print the evaluation metrics
    print("SVM Accuracy:", accuracy_svm)
    print("\nSVM Confusion Matrix:\n", conf_matrix_svm)
    print("\nSVM Classification Report:\n", class_report_svm)
```

```
SVM Accuracy: 0.6389891696750902

SVM Confusion Matrix:
 [[  0 100]
 [  0 177]]

SVM Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       100
           1       0.64      1.00      0.78       177

    accuracy                           0.64       277
   macro avg       0.32      0.50      0.39       277
weighted avg       0.41      0.64      0.50       277
```

Activity 2: Random forest model

We create a RandomForest model named RF is created and Inside the function, RandomForestClassifier algorithm is initialized and training data is passed to the model

Activity 2: Random forest model

```
[ ] from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

    # Create a Random Forest classifier
    RF = RandomForestClassifier(random_state=42)

    # Train the Random Forest classifier on the balanced dataset
    RF.fit(X_train, y_train)

    # Predict on the test set
    y_pred_rf = RF.predict(X_test)

    # Evaluate the Random Forest classifier
    accuracy_rf = accuracy_score(y_test, y_pred_rf)
    conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
    class_report_rf = classification_report(y_test, y_pred_rf)

    # Print the evaluation metrics
    print("Random Forest Accuracy:", accuracy_rf)
    print("\nRandom Forest Confusion Matrix:\n", conf_matrix_rf)
    print("\nRandom Forest Classification Report:\n", class_report_rf)
```

```
Random Forest Accuracy: 1.0

Random Forest Confusion Matrix:
 [[100   0]
 [  0 177]]

Random Forest Classification Report:
               precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       100
         1.0       1.00      1.00      1.00       177

    accuracy                           1.00       277
   macro avg       1.00      1.00      1.00       277
weighted avg       1.00      1.00      1.00       277
```

Activity 3: Testing the model

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

Activity 4: Pickle Model save

Activity 3:Testing the model

```
[ ] # Assuming you have a trained Random Forest model named RF
    # Assuming X_train has 35 features

    # New data for testing with 35 features
    new_data = [[3, 3, 3, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]

    # Predict with the trained Random Forest model
    prediction = RF.predict(new_data)

    # Display the prediction
    print("Model Prediction:", prediction)
```

```
Model Prediction: [0.]
```

Activity 4: Pickle Model save

```
[ ] import pickle

    # Assuming RF is your trained Random Forest model
    # You should have trained and saved RF before running this code

    # Save the Random Forest model to a file
    with open('random_forest_model.pkl', 'wb') as file:
        pickle.dump(RF, file)
```

After the pickle model save, we have to download and save in project Folder.

Milestone 5: Application Building

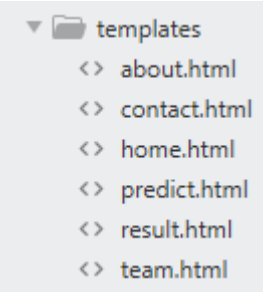
In this section, we will be building a web application that is integrated to the model we built.A UI is provided for the uses where he has to enter the values for predictions.

Activity1: Building Html Pages:

For this project create these HTML files namely

- home.html
- predict.html
- result.html
- about.html
- team.html

and save them in templates folder in project folder.



Activity 2: Build Python code:

Import the libraries

```
from flask import Flask, render_template, request, redirect
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
import pandas as pd
from datetime import datetime
from sklearn.impute import SimpleImputer
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__)

# Load the model
model = joblib.load("random_forest_model.pkl")

# Save the model
joblib.dump(model, "random_forest_model.pkl")
```

Render HTML pages:

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

@app.route('/about')
def about():
    return render_template('about.html')

@app.route('/team')
def team():
    return render_template('team.html')

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

```python
@app.route('/predict', methods=['GET', 'POST'])
def predict():

    columns_after_one_hot_encoding = ['latitude', 'longitude', 'zip_code', 'city', 'name', 'founded_at', 'closed_at',
                                      'first_funding_at', 'last_funding_at', 'age_first_funding_year',
                                      'age_last_funding_year', 'age_first_milestone_year',
                                      'age_last_milestone_year', 'relationships', 'funding_rounds',
                                      'funding_total_usd', 'milestones', 'has_VC', 'has_angel',
                                      'has_roundA', 'has_roundB', 'has_roundC', 'has_roundD',
                                      'avg_part', 'top500', 'state_code_CA', 'state_code_MA',
                                      'state_code_NY', 'state_code_TX', 'state_code_other',
                                      'category_advertisement', 'category_biotechnology',
                                      'category_consulting', 'category_e-commerce', 'category_enterprise',
                                      ]

    if request.method == 'POST':
        # Get form data
        state_code = request.form['state_code']
        latitude = float(request.form['latitude'])
        longitude = float(request.form['longitude'])
        zip_code = int(request.form['zip_code'])
        city = request.form['city']
        name = request.form['name']
        founded_at = datetime.strptime(request.form['founded_at'], '%Y-%m-%d')
        closed_at = datetime.strptime(request.form['closed_at'], '%Y-%m-%d') if request.form['closed_at'] else None
        first_funding_at = datetime.strptime(request.form['first_funding_at'], '%Y-%m-%d')
        last_funding_at = datetime.strptime(request.form['last_funding_at'], '%Y-%m-%d')
        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 = int(request.form['relationships'])
        funding_rounds = int(request.form['funding_rounds'])
        funding_total_usd = float(request.form['funding_total_usd'])
        milestones = int(request.form['milestones'])
        category = request.form['category']
        has_VC = int(request.form.get('has_VC', 0))
```

```python
        has_angel = int(request.form.get('has_angel', 0))
        has_roundA = int(request.form.get('has_roundA', 0))
        has_roundB = int(request.form.get('has_roundB', 0))
        has_roundC = int(request.form.get('has_roundC', 0))
        has_roundD = int(request.form.get('has_roundD', 0))
        avg_part = float(request.form['avg_part'])
        top500 = int(request.form.get('top500', 0))

        # Encode categorical variables
        label_encoder = LabelEncoder()

        city_encoded = label_encoder.fit_transform([city])
        name_encoded = label_encoder.fit_transform([name])

            # One-hot encode state_code and category
        state_code_encoded = pd.get_dummies([state_code], prefix='state_code')
        category_encoded = pd.get_dummies([category], prefix='category')
```

```python
    state_code_encoded = state_code_encoded.reindex(columns=state_code_columns, fill_value=0)
    category_encoded = category_encoded.reindex(columns=category_columns, fill_value=0)

    founded_at_timestamp = founded_at.timestamp()
    closed_at_timestamp = closed_at.timestamp() if closed_at else None
    first_funding_at_timestamp = first_funding_at.timestamp()
    last_funding_at_timestamp = last_funding_at.timestamp()

    # Prepare the input data for prediction
    input_data = [[latitude, longitude, zip_code, city_encoded[0], name_encoded[0],
                   founded_at_timestamp, closed_at_timestamp,
                   first_funding_at_timestamp, last_funding_at_timestamp,
                   age_first_funding_year, age_last_funding_year,
                   age_first_milestone_year, age_last_milestone_year,
                   relationships, funding_rounds,
                   funding_total_usd, milestones, has_VC, has_angel,
                   has_roundA, has_roundB, has_roundC, has_roundD,
                   avg_part, top500, *state_code_encoded.values.tolist()[0],
                   *category_encoded.values.tolist()[0]]]

    # Create a DataFrame from the input data

    input_df = pd.DataFrame(input_data, columns=['latitude', 'longitude', 'zip_code', 'city', 'name', 'founded_at', 'closed_at',
                                                 'first_funding_at', 'last_funding_at', 'age_first_funding_year',
                                                 'age_last_funding_year', 'age_first_milestone_year',
                                                 'age_last_milestone_year', 'relationships', 'funding_rounds',
                                                 'funding_total_usd', 'milestones', 'has_VC', 'has_angel',
                                                 'has_roundA', 'has_roundB', 'has_roundC', 'has_roundD',
                                                 'avg_part', 'top500', *state_code_encoded.columns.tolist(),
                                                 *category_encoded.columns.tolist()])
```

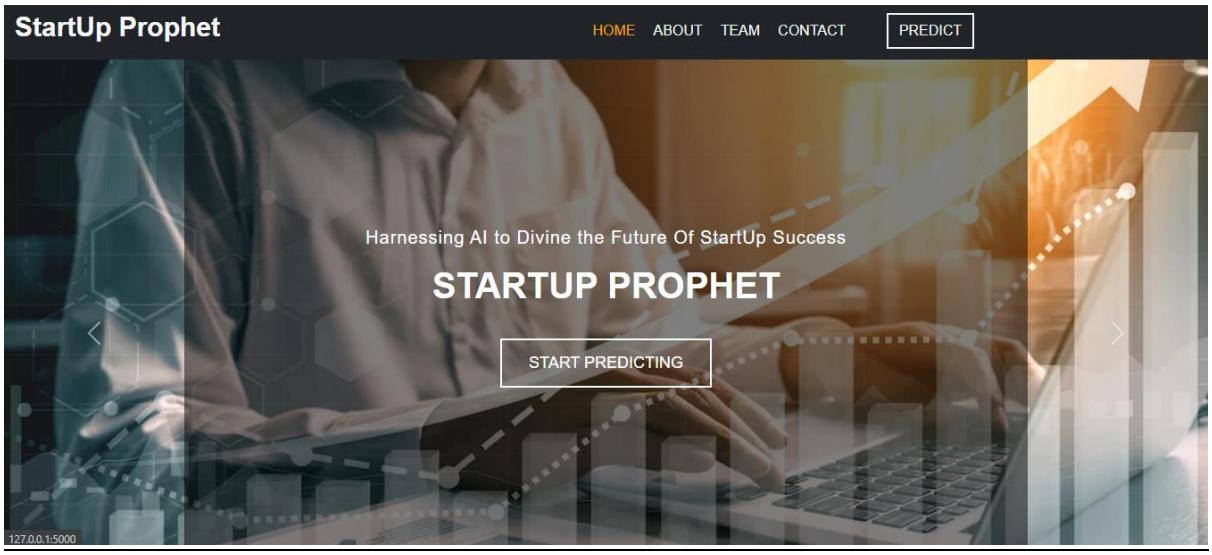
```python
        # Use the model to make predictions
        prediction = model.predict(input_df_imputed)

        # Determine the result (you need to adjust this based on your model output)
        result = "Successful" if prediction == 1 else "Not Successful"

        # Redirect to the result page with the result
        return redirect(f'/result?result={result}')

    return render_template('predict.html')

@app.route('/result')
def result():
    result = request.args.get('result', 'Unknown')
    return render_template('result.html', result=result)

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

Activity 3: Run the application

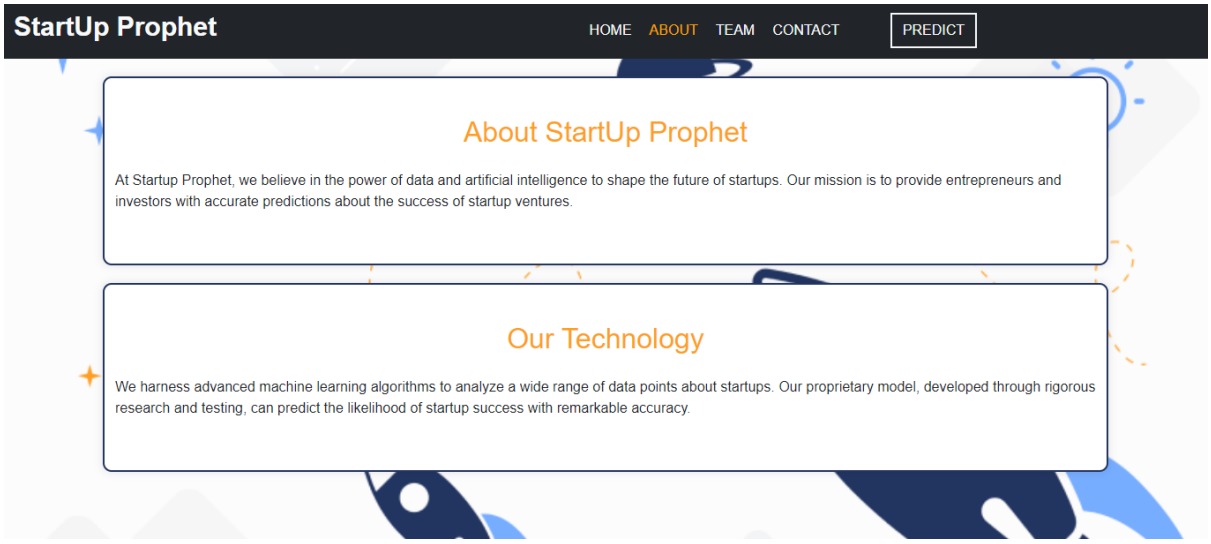
- Open git bash from the start menu
- Navigate to the folder where your python script is.
- Now type “flask run” command
- Navigate to the localhost where you can view your web page.

```
(virt)
HP@DESKTOP-UISA9GE MINGW64 ~/OneDrive/Documents/Startup Prophet/Project Developm
ent/StartUp Prophet
$ flask run
 * Debug mode: off
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
```

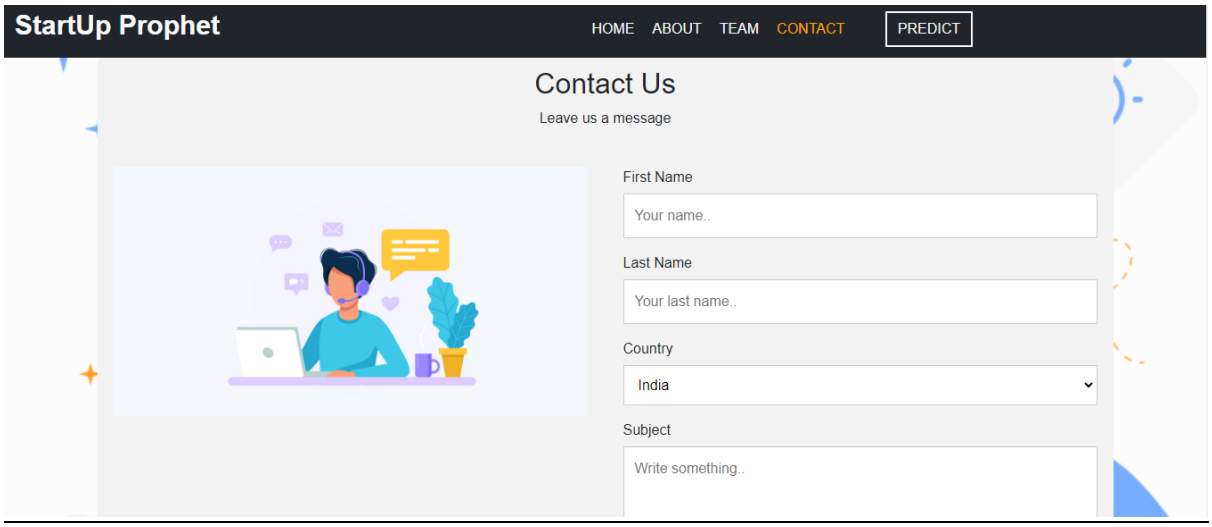
Home Page



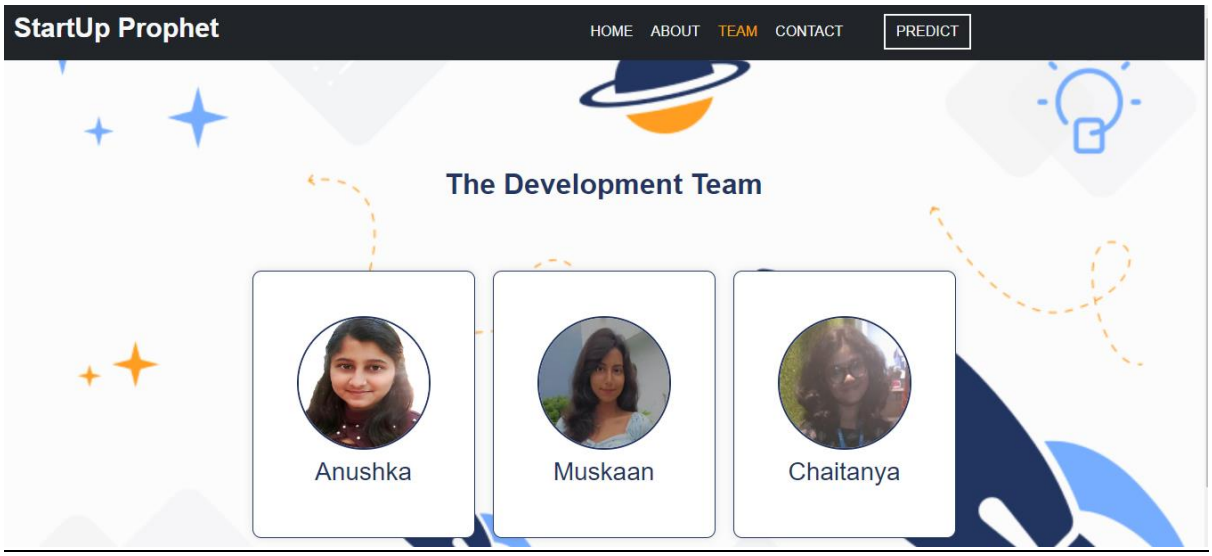
About Page



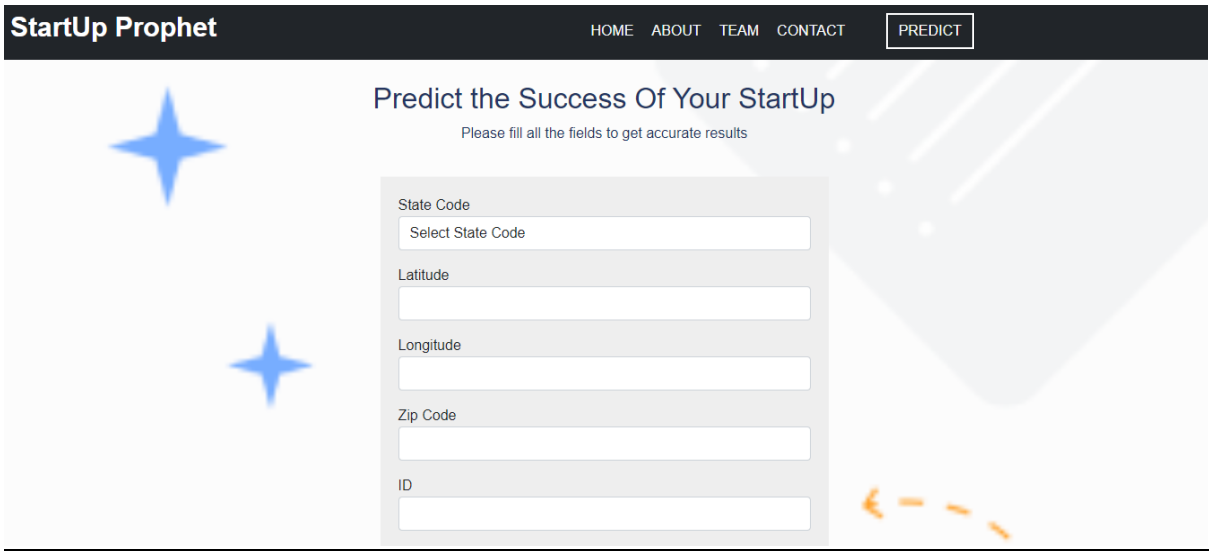
Contact Page

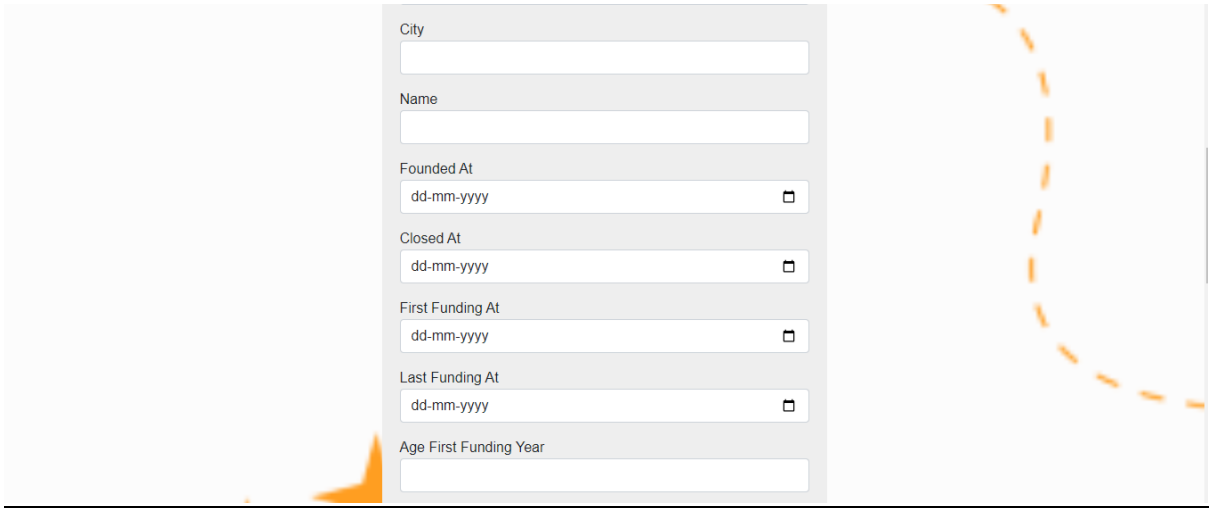


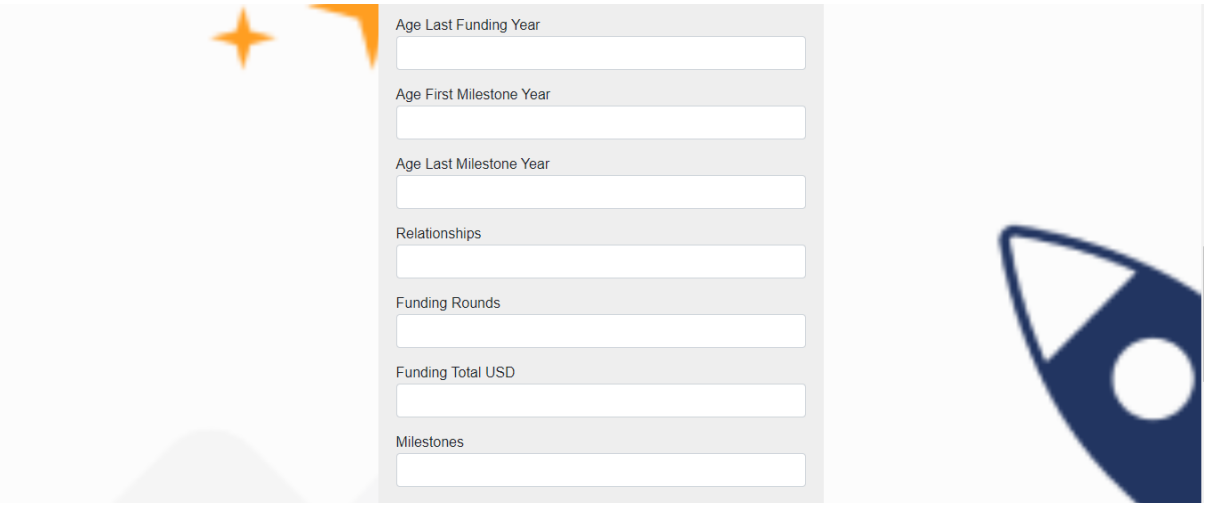
Team Page

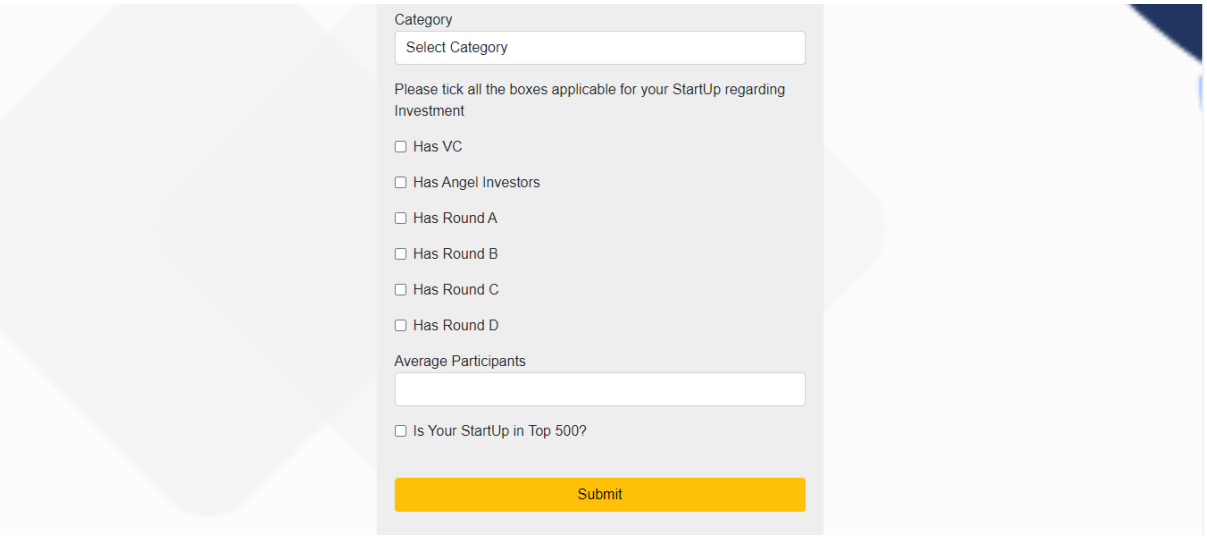


Predict Page

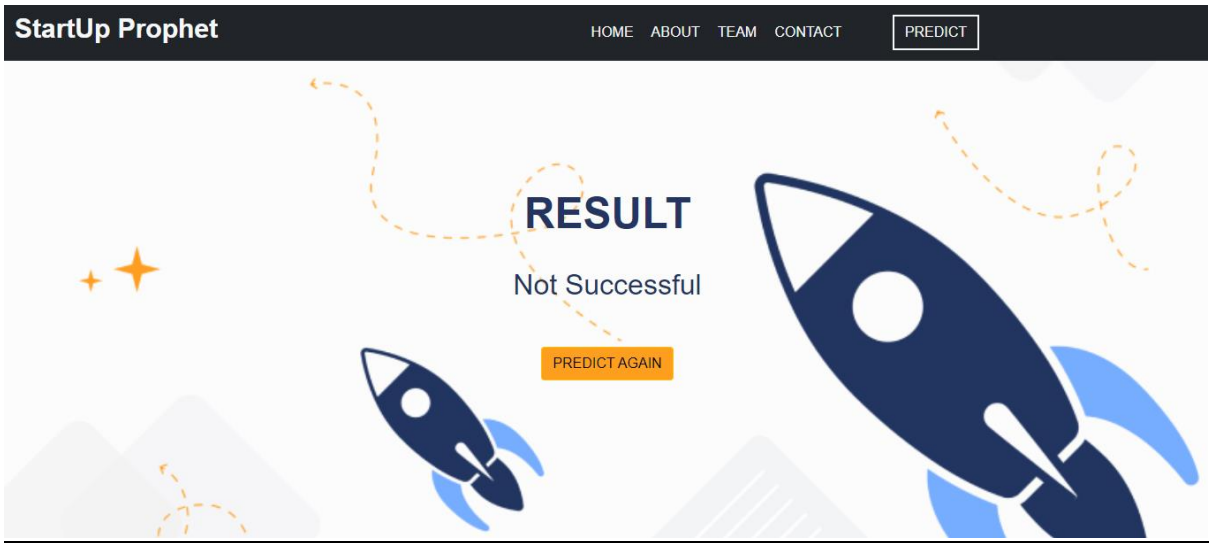





Age Last Funding Year
Age First Milestone Year
Age Last Milestone Year
Relationships
Funding Rounds
Funding Total USD
Milestones


Category
Select Category
Please tick all the boxes applicable for your StartUp regarding Investment
Has VC
Has Angel Investors
Has Round A
Has Round B
Has Round C
Has Round D
Average Participants
Is Your StartUp in Top 500?
Submit

Result Page


StartUp Prophet
HOME
ABOUT
TEAM
CONTACT
PREDICT
RESULT
Not Successful
PREDICT AGAIN