SpaceX Falcon 9 First Stage Landing Success Predictor

Project Description:

The SpaceX Falcon 9 First Stage Landing Success Predictor project aims to develop a predictive model that can forecast the success of first-stage landings for Falcon 9 rocket launches. The project involves gathering and preprocessing data related to SpaceX launches, including factors such as launch site coordinates, weather conditions, launch outcomes, and possibly other relevant parameters. The goal is to create a reliable model that can predict whether the first stage of the Falcon 9 rocket will successfully land, based on historical data and real-time inputs.

Project Scenario:

Scenario 1: Mission Planning for Satellite Deployment

Background: A satellite company plans to deploy a constellation of communication satellites using SpaceX's Falcon 9 rocket. The success of each satellite deployment mission relies on the precise first stage landing of the Falcon 9.

Use Case:

1. **Mission Parameters Input:** Input launch site coordinates, scheduled launch dates for each satellite, and specific payload details.
2. **Weather Assessment:** Evaluate real-time weather forecasts at launch sites to assess wind speeds, visibility, and other factors affecting launch and landing conditions.
3. **Predictive Analysis:** Utilize the machine learning model to predict the likelihood of successful first stage landings based on historical data and current environmental factors.
4. **Decision Support:** Optimize mission timelines and adjust launch schedules based on predictive insights to ensure reliable satellite deployment and minimize launch risks.
5. **Outcome:** Enhance mission planning efficiency and success rates, ensuring reliable satellite deployment and operational continuity for communication services.

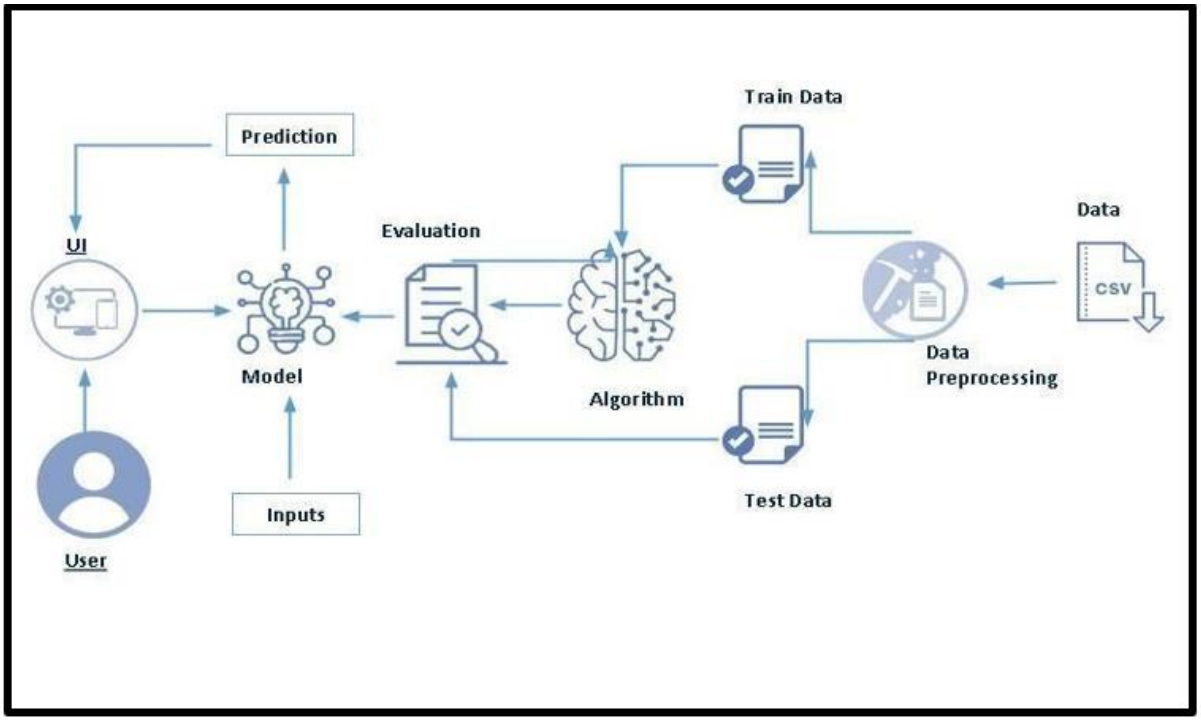
Scenario 2: Crewed Mission to the International Space Station (ISS)

Background: SpaceX conducts a crewed mission to the ISS using the Falcon 9 rocket, with the safety and success of crew transportation contingent on the Falcon 9's first stage landing.

Use Case:

1. **Mission Details Input:** Input launch coordinates, crewed mission schedule, and specific safety requirements for crew transportation to the ISS.
2. **Safety Assessment:** Evaluate real-time safety parameters and mission constraints to ensure crew safety during launch and first stage landing phases.
3. **Predictive Modeling:** Employ the machine learning model to predict the probability of successful first stage landings, considering crew safety protocols and mission criticality.
4. **Decision Making:** Implement risk mitigation strategies and adjust mission procedures based on predictive insights to ensure safe and successful crew transportation.
5. **Outcome:** Facilitate reliable crew transportation to the ISS, supporting ongoing space exploration and ensuring crew safety in critical missions.

Technical Architecture:



Project Flow: SpaceX Falcon 9 First Stage Landing Success Predictor

1. Project Initiation

- **Project Objective:** The objective of the SpaceX Falcon 9 First Stage Landing Success Predictor project is to develop a machine learning model that predicts the success of Falcon 9 first stage landings based on various input factors. The model will be integrated into a Flask web application to provide real-time predictions through a user-friendly interface

2. Data Collection and Integration

- **Collect the Dataset:** Gather historical SpaceX launch data from sources like Kaggle, ensuring it includes features such as flight number, payload mass, orbit, launch site, weather conditions, and first stage landing outcomes.

3. Data Preprocessing

- **Load Data:**
- Load the collected datasets into a Pandas DataFrame for initial inspection and processing.
- **Handle Missing Values:**
- Identify and handle missing values by filling with appropriate statistics (mean/median/mode) or using advanced techniques like interpolation.
- **Remove Duplicates:**
- Identify and remove duplicate rows to ensure data integrity.
- **Feature Engineering:**
- Create new features that might help improve the model's performance, such as launch time of day or weather conditions during launch.
- Encode categorical variables using techniques like one-hot encoding to convert them into numerical format.

4. Exploratory Data Analysis (EDA)

- **Statistical Analysis**: Analyze data distributions, and summary statistics.
- **Visualization**: Use visual tools to understand data patterns and relationships (e.g., countplot, barplots, heatmaps).

5. **Model Development**

- **Algorithm Selection**: Choose appropriate machine learning algorithms (e.g., logistic regression, decision trees, random forests).
- **Model Training**: Train models using the training dataset.
- **Model Tuning**: Optimize hyperparameters using techniques such as grid search or random search.
- **Model Evaluation**: Evaluate models using metrics such as accuracy, precision, recall, F1-score, and ROC-AUC.

6. **Model Validation**

- **Cross-Validation**: Perform cross-validation to ensure model robustness.

- **Performance Comparison**: Compare performance of different models to select the best one.

7.**Model Deployment**

- **Integration:** Deploy the selected model into a web application for easy accessibility.
- **API Development:** Develop APIs to integrate the model with other systems for real-time data input and predictions.
- **Real-Time Predictions:** Enable real-time prediction of Falcon 9 first stage landing success.

8.**User Interaction Design**

- **User-Friendly Interface:** Create an intuitive web interface for users to input launch parameters and receive predictions.
- **Visualization:** Display prediction results and key insights visually for better understanding.

9.**Monitoring and Maintenance**

- **Performance Monitoring:** Continuously monitor the model's performance and the accuracy of predictions.
- **Feedback Loop:** Collect user feedback to refine and improve the model and the user interface.

10.**Reporting and Insights**

- **Dashboard Creation:** Develop dashboards to visualize key metrics, such as success rates and influential factors.
- **Stakeholder Reporting:** Provide regular reports to stakeholders on model performance and the impact of predictions on decision-making.

prior Knowledge:

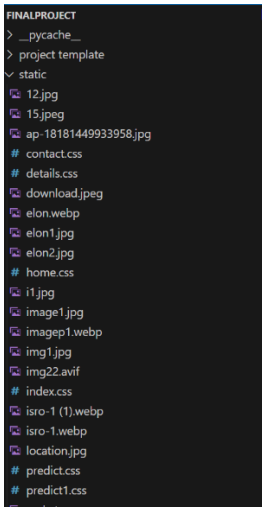
You must have prior knowledge of the following topics to complete this project.

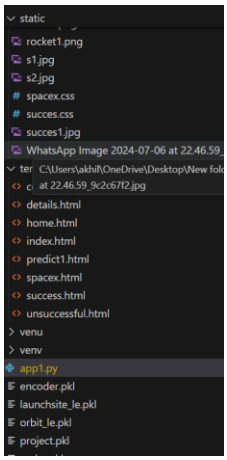
- ML Concepts

- Supervised learning: https://www.javatpoint.com/supervised-machine-learning
- Unsupervised learning: https://www.javatpoint.com/unsupervised-machine-learning
- Decision tree: https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
- Random forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm
- Hyper parameter Tuning : https://www.geeksforgeeks.org/hyperparameter-tuning/
- Flask Basics: https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Structure:

Create the Project folder which contains files as shown below





- We are building a Flask application that needs HTML pages stored in the templates folder and a Python script app.py for scripting.
- model.pkl is our saved model. Further, we will use this model for 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: SpaceX_Falcon9_Launch_Data (kaggle.com)

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

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

Activity 2: Importing the libraries

Import the necessary libraries as shown in the image

```
#Importing Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score,classification_report,roc_auc_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
```

```
import pickle
```

Activity 3: Read the Dataset

Our dataset format might be in .csv, excel files,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.

```
df=pd.read_csv("/content/spacex_launch_data.csv")
```

Activity 4: 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 to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling categorical data
- 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 5: Handling missing values

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

```
#handling missing values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 90 entries, 0 to 89
Data columns (total 17 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   FlightNumber    90 non-null     int64
 1   Date            90 non-null     object
 2   BoosterVersion  90 non-null     object
 3   PayloadMass     90 non-null     float64
 4   Orbit           90 non-null     object
 5   LaunchSite      90 non-null     object
 6   Outcome         90 non-null     object
 7   Flights         90 non-null     int64
 8   GridFins        90 non-null     bool
 9   Reused          90 non-null     bool
 10  Legs            90 non-null     bool
 11  LandingPad      64 non-null     object
 12  Block           90 non-null     float64
 13  ReusedCount     90 non-null     int64
 14  Serial          90 non-null     object
 15  Longitude       90 non-null     float64
 16  Latitude        90 non-null     float64
dtypes: bool(3), float64(4), int64(3), object(7)
memory usage: 10.2+ KB
```

For checking the null values, data.isnull() function is used. To sum those null values we use .sum() function. From the below image, we found that there are null values present in our dataset.

```
df.isnull().sum()/df.count()*100
```

```
FlightNumber       0.000
Date               0.000
BoosterVersion     0.000
PayloadMass        0.000
Orbit              0.000
LaunchSite         0.000
Outcome            0.000
Flights            0.000
GridFins           0.000
Reused             0.000
Legs               0.000
LandingPad        40.625
Block              0.000
ReusedCount        0.000
Serial             0.000
Longitude          0.000
Latitude           0.000
dtype: float64
```

As we can see our dataset has missing values.

There is no use of LandingPad in dataset then we dropping this column

```
df.drop(["LandingPad"],axis=1,inplace=True)
```

We can proceed with it.

Feature Engineering

1 Check for data types to find out categorical and numerical data

```
df.dtypes
```

```
FlightNumber       int64
Date              object
BoosterVersion    object
PayloadMass      float64
Orbit             object
LaunchSite        object
Outcome           object
Flights            int64
GridFins            bool
Reused              bool
Legs                bool
Block            float64
ReusedCount        int64
Serial            object
Longitude        float64
Latitude         float64
dtype: object
```

2. Apply value counts on features to check which one should be label encoded.

i.) Calculate the number of launches on each site

The data contains several Space X launch facilities:

CCAFS SLC 40: Cape Canaveral Space Launch Complex 40

VAFB SLC 4E: Vandenberg Air Force Base Space Launch Complex 4E,

KSC LC 39A: Kennedy Space Center Launch Complex 39A.

The location of each Launch Is placed in the column LaunchSite

Next, let's see the number of launches for each site.

Use the method value_counts() on the column LaunchSite to determine the number of launchs

```
df["LaunchSite"].value_counts()
```

```
LaunchSite
CCAFS SLC 40    55
KSC LC 39A      22
VAFB SLC 4E     13
Name: count, dtype: int64
```

ii.) Calculate the number and occurrence of each orbit

Use the method .value_counts() to determine the number and occurrence of each orbit in the column Orbit

```
df["Orbit"].value_counts()
```

```
Orbit
GTO      27
ISS      21
VLEO     14
PO        9
LEO       7
SSO       5
MEO       3
ES-L1     1
HEO       1
SO        1
GEO       1
Name: count, dtype: int64
```

iii.) Calculate the number of longitudes and latitudes used

```
3] df["Longitude"].value_counts()
```

```
Longitude
-80.577366     55
-80.603956     22
-120.610829    13
Name: count, dtype: int64
```

```
4] df["Latitude"].value_counts()
```

```
Latitude
28.561857    55
28.608058    22
34.632093    13
Name: count, dtype: int64
```

iv.)Calculate the block versions used

```
[15] df["Block"].value_counts()
```

```
Block
5.0    39
1.0    19
3.0    15
4.0    11
2.0     6
Name: count, dtype: int64
```

---

v.) Calculate the number of times block has been reused

```
df["ReusedCount"].value_counts()
```

```
ReusedCount
0    30
1    24
3    12
5    11
2     9
4     4
Name: count, dtype: int64
```

vi.) Calculate number of flights with a particular core

```
df["Flights"].value_counts(
```

```
Flights
1    53
2    19
3     8
4     6
5     2
6     2
Name: count, dtype: int64
```

3.Creating target column class to classify launches

i.) Calculate the number and occurrence of mission outcome per orbit type

Use the method .value_counts() on the column Outcome to determine the number of landing_outcomes. Then assign it to a variable landing_outcomes.

```
landing_outcomes=df["Outcome"].value_counts()
landing_outcomes
```

```
Outcome
True ASDS      41
None None      19
True RTLS      14
False ASDS      6
True Ocean      5
False Ocean     2
None ASDS       2
False RTLS      1
Name: count, dtype: int64
```

- **True Ocean** means the mission outcome was successfully landed to a specific region of the ocean.
- **False Ocea**n means the mission outcome was unsuccessfully landed to a specific region of the ocean.
- **True RTLS** means the mission outcome was successfully landed to a ground pad.
- **False RTLS** means the mission outcome was unsuccessfully landed to a ground pad.
- **True ASDS** means the mission outcome was successfully landed to a drone ship.
- **False ASDS** means the mission outcome was unsuccessfully landed to a drone ship.
- **None ASDS and None None** these represent a failure to land.

ii.) Create a set of outcomes where the second stage did not land successfully:

```
for i,outcome in enumerate(landing_outcomes.keys()):
  print(i,outcome)
```

```
0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 False Ocean
6 None ASDS
7 False RTLS
```

```
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
bad_outcomes
```

```
{'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

iii.) Create a landing outcome label from Outcome column

Using the Outcome, create a list where the element is zero if the corresponding row in Outcome is in the set bad_outcome; otherwise, it's one. Then assign it to the variable landing_class:

```
landing_class=[0 if i in set(bad_outcomes) else 1 for i in df["Outcome"]]
```

```
[ ] df["class"]=landing_class
    df[["class"]].head(8)
```

	class
0	0
1	0
2	0
3	0
4	0
5	0
6	1
7	1

Milestone 2: Exploratory Data Analysis

Activity 1: Descriptive statistics

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

```
df.describe()
```

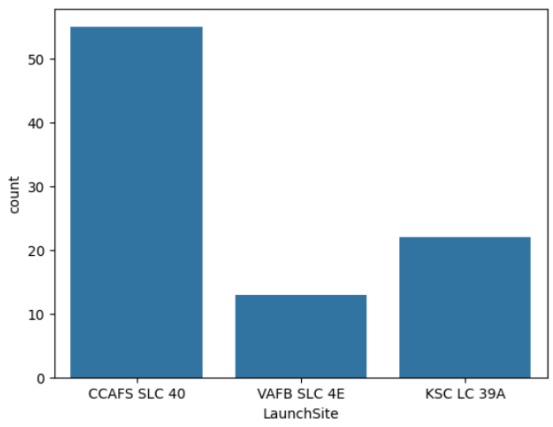
	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Longitude	Latitude	class
count	90.000000	90.000000	90.000000	90.000000	90.000000	90.000000	90.000000	90.000000
mean	45.500000	6104.959412	1.788889	3.500000	1.655556	-86.366477	29.449963	0.666667
std	26.124701	4694.671720	1.213172	1.595288	1.710254	14.149518	2.141306	0.474045
min	1.000000	350.000000	1.000000	1.000000	0.000000	-120.610829	28.561857	0.000000
25%	23.250000	2510.750000	1.000000	2.000000	0.000000	-80.603956	28.561857	0.000000
50%	45.500000	4701.500000	1.000000	4.000000	1.000000	-80.577366	28.561857	1.000000
75%	67.750000	8912.750000	2.000000	5.000000	3.000000	-80.577366	28.608058	1.000000
max	90.000000	15600.000000	6.000000	5.000000	5.000000	-80.577366	34.632093	1.000000

Activity 2: Visual analysis

Visual analysis is 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 decisions

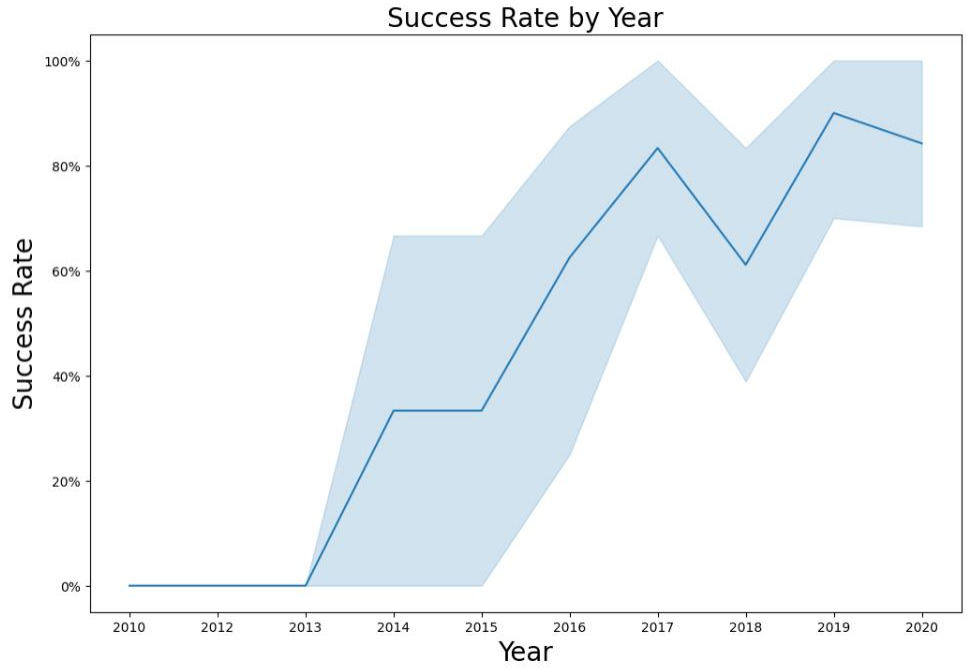
```
sns.countplot(x="LaunchSite",data=df)
```

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



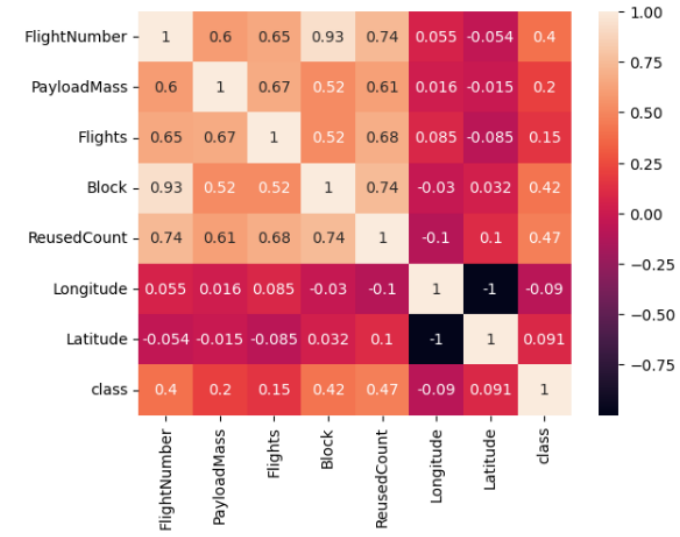
```
[30] year=[]
     def Extract_year(date):
       for i in df["Date"]:
         year.append(i.split("-")[0])
       return year
```

```
df["year"]=Extract_year(df["Date"])
fig, ax=plt.subplots()
fig.set_size_inches(12,8)
sns.lineplot(x="year",y="class",data=df)
plt.xlabel("Year",fontsize=20)
plt.ylabel("Success Rate",fontsize=20)
ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, _: "{:.0f}%".format(x*100)))
plt.title("Success Rate by Year",fontsize=20)
plt.show()
```




```
[32] # Step 1: Select only numerical columns
     df1 = df.select_dtypes(include=[int, float])
     sns.heatmap(df1.corr(),annot=True)
```

```
<Axes: >
```



Activity 3: Splitting data into train and test

Now let's split the Dataset into train and test sets. First, split the dataset into x and y and then split the data set, here x and y variables are created. On the x variable, data is passed by dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using the train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.

```
x=df[["PayloadMass","Orbit","LaunchSite","Longitude","Latitude","GridFins","Legs","Block","Flights","ReusedCount"]]
x.head()
```

	PayloadMass	Orbit	LaunchSite	Longitude	Latitude	GridFins	Legs	Block	Flights	ReusedCount
0	6104.959412	5	0	2	0	0	0	1.0	1	0
1	525.000000	5	0	2	0	0	0	1.0	1	0
2	677.000000	4	0	2	0	0	0	1.0	1	0
3	500.000000	7	2	0	2	0	0	1.0	1	0
4	3170.000000	2	0	2	0	0	0	1.0	1	0

```
[43] y=df["class"]
     y
```

```
0     0
1     0
2     0
3     0
4     0
     ..
85    1
86    1
87    1
88    1
89    1
Name: class, Length: 90, dtype: int64
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

Scaling all features

```
Scaler=preprocessing.StandardScaler()
x_train=Scaler.fit_transform(x_train)
x_test=Scaler.transform(x_test)
```

The code imports the StandardScaler module for sci-kit-learn for feature scaling. It then scales the features in the input data 'x' using standardization and splits the dataset into training and testing sets with a test size of 20% and a random state of 42.

Milestone 3: Model Building

Activity 1: Training the model in multiple algorithms

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 regression algorithms. The best model is saved based on its performance.

Model 1 :

Logistic Regression

A function named lr is created and train and test data are passed as the parameters. Inside the function, LogisticRegression algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable.

Logistic Regression Model

```
[115] lr=LogisticRegression()
      lr.fit(x_train,y_train)
      lr_pred=lr.predict(x_test)
```

```
[116] lr_accuracy=accuracy_score(y_test,lr_pred)
      lr_precision=precision_score(y_test,lr_pred)
      lr_recall=recall_score(y_test,lr_pred)
      lr_f1_score=f1_score(y_test,lr_pred)
      lr_auc_score=roc_auc_score(y_test,lr.predict_proba(x_test)[:,1])
```

```
lr_accuracy
```

```
0.9444444444444444
```

```
d=[[3170.000000,2,0,2,0,0,0,1.0,1,0]]
input_data_scaled=Scaler.transform(d)
y_pred=lr.predict(input_data_scaled)
print(y_pred)
```

```
[0]
```

Apply logistic regression to predict SpaceX Falcon 9 first stage landing success based on weather conditions, payload characteristics, geographical factors (launch site coordinates), and historical mission outcomes. Collect and preprocess data, train the model, evaluate with

accuracy metrics, and integrate for real-time decision support, aiming to optimize mission planning and success rates

Model 2:

Decision Tree

Apply decision tree modeling to predict SpaceX Falcon 9 first stage landing success using weather conditions, payload characteristics, geographical factors (launch site coordinates), and historical mission outcomes. Prepare data, train the model to learn decision rules from features, evaluate using metrics like accuracy and F1-score, and deploy for real-time decision-making to enhance mission planning and success rates.

```
[119] dt=DecisionTreeClassifier()
      dt.fit(x_train,y_train)
      dt_pred=dt.predict(x_test)
```

```
dt_accuracy = accuracy_score (y_test, dt_pred)
dt_precision=precision_score (y_test, dt_pred)
dt_recall=recall_score(y_test, dt_pred)
dt_f1_score=f1_score(y_test, dt_pred)
dt_auc_score=roc_auc_score (y_test, dt.predict_proba (x_test) [:, 1])
```

```
[40] dt_accuracy
```

```
0.9444444444444444
```

```
d=[[3170.000000,2,0,2,0,0,0,1.0,1,0]]
input_data_scaled=Scaler.transform(d)
y_pred=dt.predict(input_data_scaled)
print(y_pred)
```

```
[0]
```

Model 3:

Random Forest

Utilize Random Forest modeling to predict SpaceX Falcon 9 first stage landing success, incorporating weather conditions, payload characteristics, geographical factors (launch site coordinates), and historical mission outcomes. Prepare data, train the ensemble model of

decision trees, evaluate performance metrics such as accuracy and F1-score, and deploy for real-time decision support to optimize mission planning and success rates.

Random Forest Model

```
[122] rf=RandomForestClassifier()
      rf.fit(x_train, y_train)
      rf_pred= rf.predict(x_test)
```

```
[123] rf_accuracy = accuracy_score (y_test, rf_pred)
      rf_precision = precision_score (y_test, rf_pred)
      rf_recall = recall_score (y_test, rf_pred)
      rf_f1_score = f1_score (y_test, rf_pred)
      rf_auc_score =roc_auc_score (y_test, rf.predict_proba (x_test) [:, 1])
```

```
[142] rf_accuracy
```

```
0.9444444444444444
```

```
d=[[3170.000000,2,0,2,0,0,0,1.0,1,0]]
input_data_scaled=Scaler.transform(d)
y_pred=rf.predict(input_data_scaled)
print(y_pred)
```

```
[0]
```

Model 4:

K-Nearest Neighbors (KNN)

Apply K-Nearest Neighbors (KNN) to predict SpaceX Falcon 9 first stage landing success using weather conditions, payload characteristics, geographical factors (launch site coordinates), and historical mission outcomes. Prepare and normalize data, train the KNN model by identifying patterns from nearest neighbors, evaluate using accuracy and F1-score, and deploy for real-time decision support to enhance mission planning and success rates.

```
[121] knn = KNeighborsClassifier()
      knn.fit(x_train, y_train)
      knn_pred= knn.predict(x_test)
      knn_accuracy = accuracy_score (y_test, knn_pred)
      knn_precision= precision_score (y_test, knn_pred)
      knn_recall = recall_score (y_test, knn_pred)
      knn_f1_score=f1_score(y_test,knn_pred)
      knn_auc_score = roc_auc_score (y_test, knn.predict_proba (x_test) [:, 1])
```

```
[144] dt_accuracy

0.9444444444444444
```

```
d=[[3170.000000,2,0,2,0,0,0,1.0,1,0]]
input_data_scaled=Scaler.transform(d)
y_pred=knn.predict(input_data_scaled)
print(y_pred)

[0]
```

```
[133] metrics=pd.DataFrame({ 'Model': ['Logistic Regression', 'Decision Tree', 'KNN', 'Random Forest'],
      'Accuracy': [lr_accuracy, dt_accuracy, knn_accuracy, rf_accuracy],
      'AUC Score': [lr_auc_score, dt_auc_score, knn_auc_score, rf_auc_score],
      'Precision': [lr_precision, dt_precision, knn_precision, rf_precision],
      'Recall': [lr_recall, dt_recall, knn_recall, rf_recall],
      'F1 Score' : [lr_f1_score, dt_f1_score, knn_f1_score, rf_f1_score]})

      print (metrics)
```

	Model	Accuracy	AUC Score	Precision	Recall	F1 Score
0	Logistic Regression	0.944444	1.000000	0.933333	1.000000	0.965517
1	Decision Tree	0.944444	0.964286	1.000000	0.928571	0.962963
2	KNN	0.944444	0.991071	1.000000	0.928571	0.962963
3	Random Forest	0.944444	0.982143	1.000000	0.928571	0.962963

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 and saving its weights and configuration. 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.

```
filename="project.pkl"
pickle.dump(lr,open(filename,'wb'))
```

```
[138] pickle.dump(Scaler, open('scaler.pkl', 'wb'))
```

This code exports the trained machine learning model 'model1' using pickle serialization and saves it as a file named "project. pkl". This allows for the model to be easily stored, shared, and loaded for future use without needing to retrain it.

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 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 server-side script
- Run the web application

Activity 3: Building Html Pages:

Activity 3: Building Html Pages:

For this project create one HTML file namely

- index.html

and save them in the templates folder.

Activity 4: Build Python code:

Import the libraries

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

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

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

In the below example, the '/' URL is bound with the home.html function. Hence, when the home page of the web server is opened in the 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:

```
flask_app=Flask(__name__)
@flask_app.route("/")
def spacex():
  return render_template('spacex.html')
@flask_app.route("/about")
def about():
  return render_template('home.html')
@flask_app.route("/details")
def details():
  return render_template('details.html')
@flask_app.route("/contact")
def contact():
  return render_template('contact.html')
@flask_app.route("/predict",methods=['POST',"GET"])
def predict():
  return render_template('predict1.html')
@flask_app.route('/submit', methods=['POST', 'GET'])
```

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

```
longitudes = {
    -120.610829: 0,
    -80.603956: 1,
    -80.577366: 2
}
latitudes = {
    28.561857: 0,
    28.608058: 1,
    34.632093: 2
}
with open("project.pkl",'rb') as m:
  model=pickle.load(m)
with open("scaler.pkl",'rb') as s:
  scaler=pickle.load(s)
with open("orbit_le.pkl",'rb') as o:
  orbit_le=pickle.load(o)
with open("launchsite_le.pkl",'rb') as ls:
  launchsite_le=pickle.load(ls)
#create flask app
flask_app=Flask(__name__)
@flask_app.route("/")
def spacex():
  return render_template('spacex.html')
@flask_app.route("/about")
def about():
  return render_template('home.html')
@flask_app.route("/details")
def details():
  return render_template('details.html')
@flask_app.route("/contact")
```

```
    return render_template('predict1.html')
@flask_app.route('/submit', methods=['POST', 'GET'])
def submit():
    payload_mass = float(request.form['PayloadMass'])
    orbit = orbit_le.transform([str(request.form.get('orbit'))])[0]
    launch_site = launchsite_le.transform([str(request.form.get('launch_site'))])[0]
    longitude = longitudes[float(request.form.get('longitude'))]
    latitude = latitudes[float(request.form.get('latitude'))]
    grindfins=0 if request.form.get("Grindfins") =="False" else 1
    legs=0 if request.form.get("Legs") =="False" else 1
    core_block_version = float(request.form.get('core_block_version'))
    flights_with_core = float(request.form['Flights_With_That_Core'])
    core_reused_count = float(request.form['Core_Reused_Count'])
    input_data = [payload_mass, orbit, launch_site, longitude, latitude, grindfins, legs, core_block_version, flights
    arr = np.array(input_data)
    print(arr)
    print(input_data)
    scaled_arr = scaler.transform([arr])
    print(scaled_arr)
    prediction = model.predict(scaled_arr)
    if prediction[0] == 1:
      return render_template('success.html')
    else:
      return render_template('unsuccessful.html')
```

Main Function:

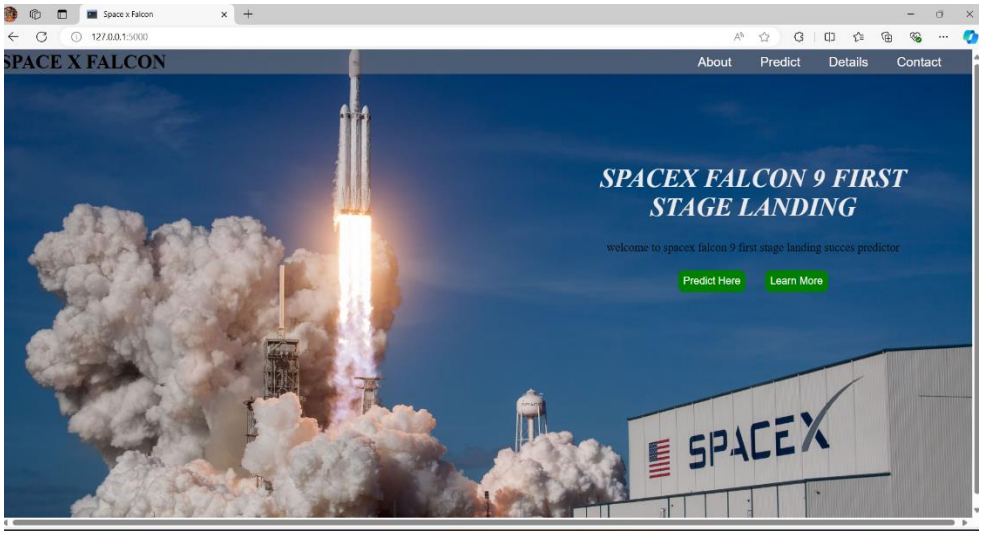
```
if __name__=="__main__":
  flask_app.run(debug=True)
```

Activity 5: Run the web application

- Open the anaconda prompt from the start menu
- Navigate to the folder where your Python script is.
- Now type the “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.

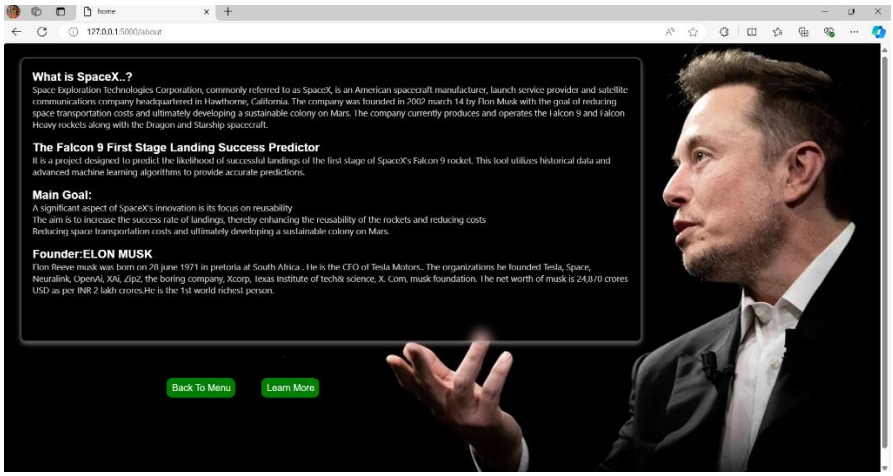
```
WARNING: This is a development server.
WSGI server instead.
 * Running on http://127.0.0.1:5000
```

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

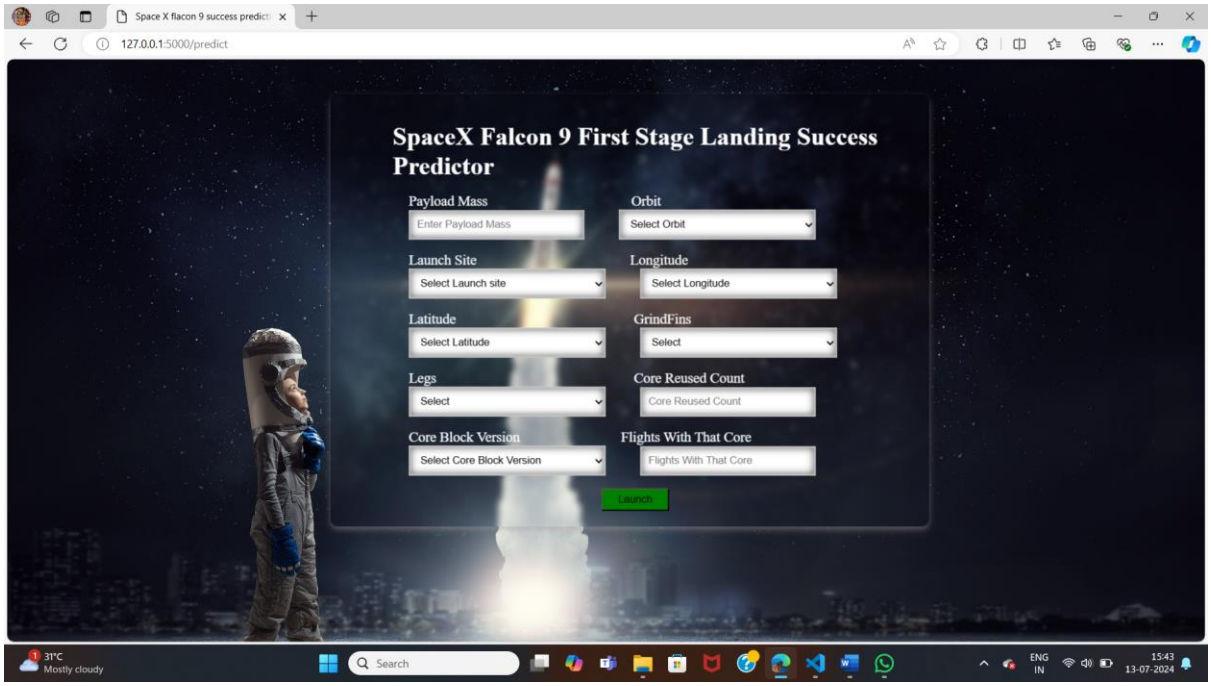


The above UI page is the spacex page. In the spacex page, it contains about, predict,details,contact

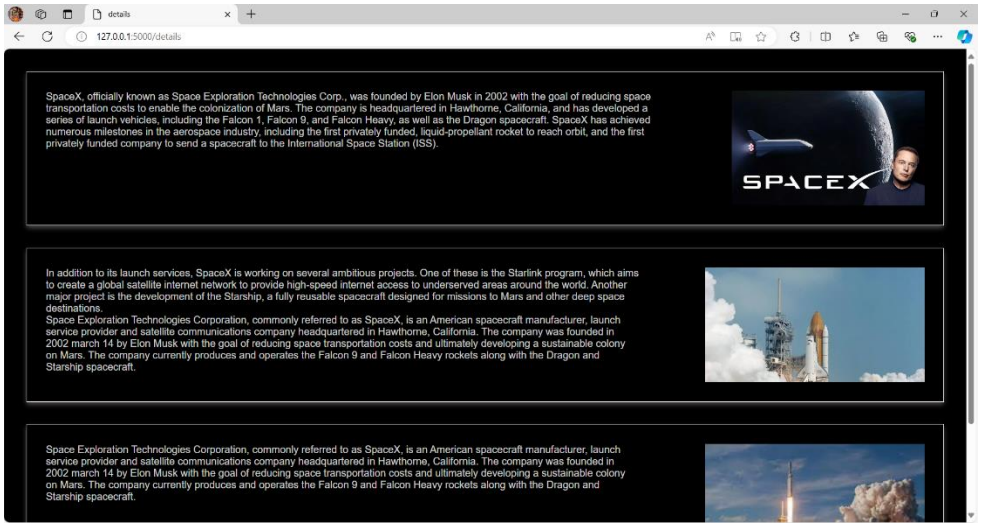
On the spacex page we have Get about the button click on it. So, that it will be redirected to the inner page.

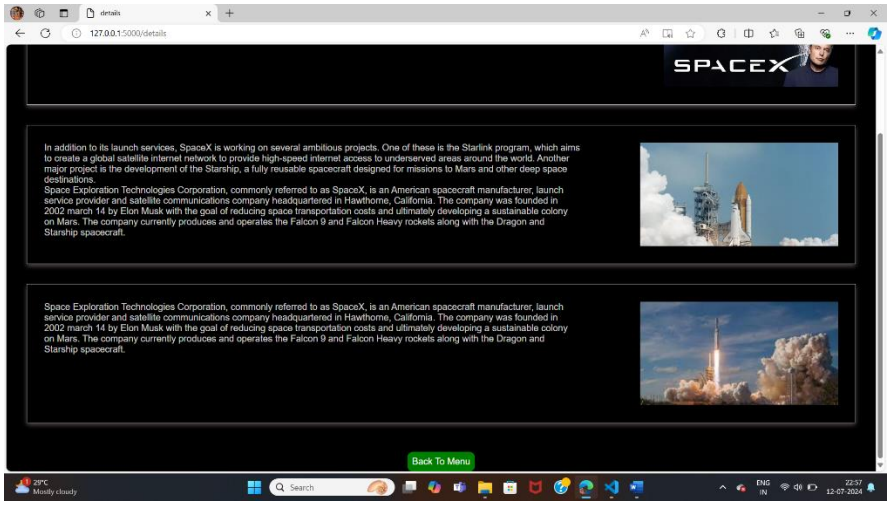


On the spacex page we have Get predict the button click on it. So, that it will be redirected to the inner page.

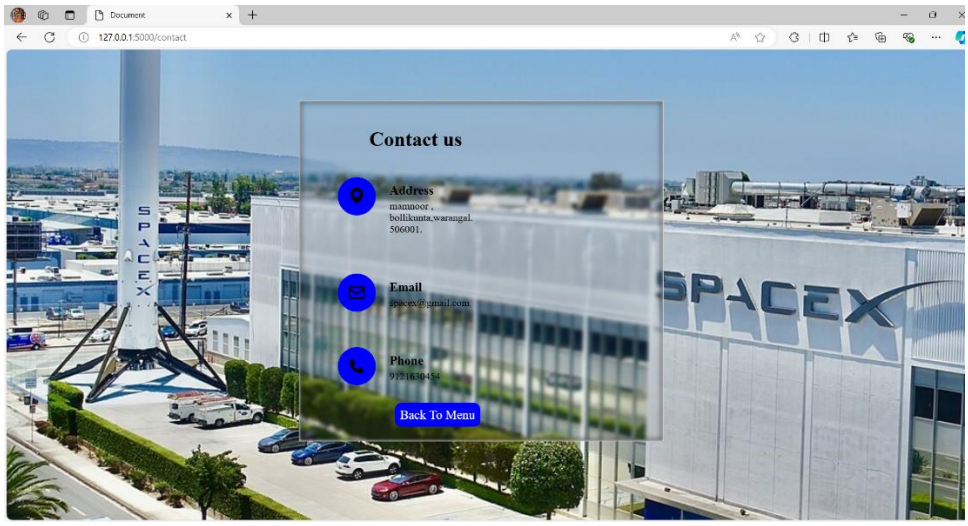


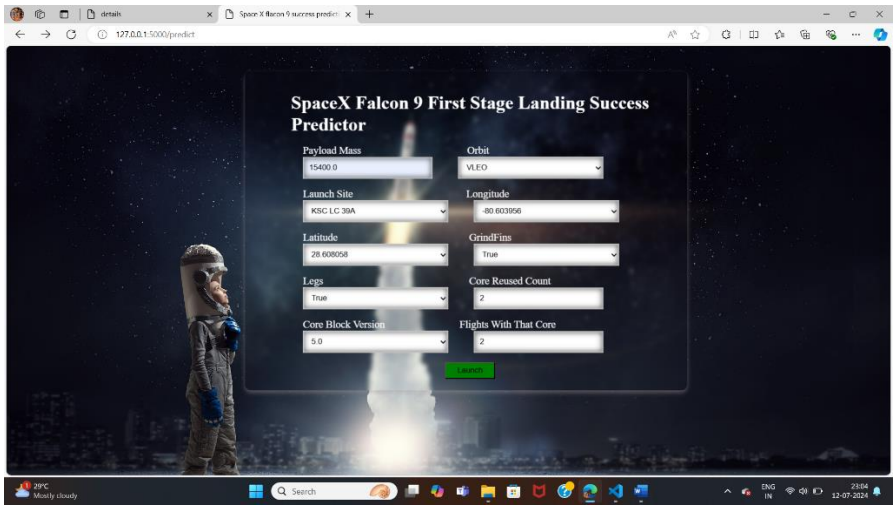
On the spacex page we have Get Details the button click on it. So, that it will be redirected to the inner page.



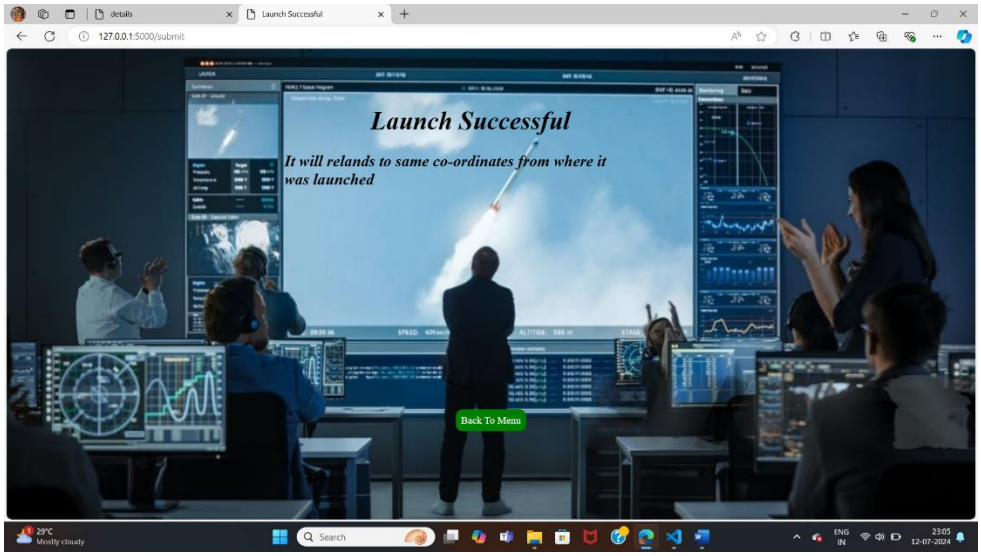


On the spacex page we have Get contact the button click on it. So, that it will be redirected to the inner page.





Output :



It is given as launch successful predicting that spaceX falcon 9 first stage landing success predicting

For launch unsuccessful prediction