CSE 587

DATA INTENSIVE COMPUTING

PROJECT

Phase – 3

Team members

Sujith Varma Vatsavai

UBID: 50442317

sujithva@buffalo.edu

Jahnavi Rudraraju

UBID: 50464467

jahnavir@buffalo.edu

Dataset Link: https://www.kaggle.com/datasets/sameepvani/nasa-nearest-earth-objects

PROJECT REPORT

PHASE – 3

Title: DETECTION OF IMPACT ZONE ON EARTH BY SPACE OBJECTS

Recap from Phase 1:

During phase 1 we have preprocessed the code using data cleaning methods such as Outliers, splitting columns, Renaming (or) clear formatting, Duplicate Entries, Missing values, Inconsistent values, Data transformation. Exploratory Data Analysis (EDA) is used for visualization of the dataset.

Recap from Phase 2:

We used a data cleaning method during model training on the hazardous column the method is to assign 0 and 1 to the values of the data-frame column, after that we split the total data into training and testing data. We used various algorithms for classification of the data. We have used K-nearest neighbors (KNN), Logistic regression, Naive Bayes, Decision Tree, Neural networks. Among the algorithms we have used for our dataset we found Decision Tree is more effective with highest accuracy.

Problem Statement:

The main problem we are going to focus is the detection of objects hitting earth, some object with diameter of at-least 5kms can be detected by using 'Radar tracking data', 'NEOWISE spacecraft' to analyze the orbit of the particles, our problem statement will focus on small particles of a diameter below 1-2 kms. We analyze the data given by NEO observers.

Analyzing whether the object is on collision course with earth, the possible impact zone, and the power of impact of all the space objects based on their rarity, last closest approach to earth, magnitude, and diameter of the object.

Dataset Description:

Dataset contains all the objects that passed by the earth, described with 90836 rows (Number of objects) and 10 columns which includes the name of the object, closest approach data, the distance between earth and the object, the speed of the object, size of the object and is the object dangerous based on the magnitude.

Dataset Link: https://www.kaggle.com/datasets/sameepvani/nasa-nearest-earth-objects

After phase 2 the following are the accuracy of the algorithms we have used:

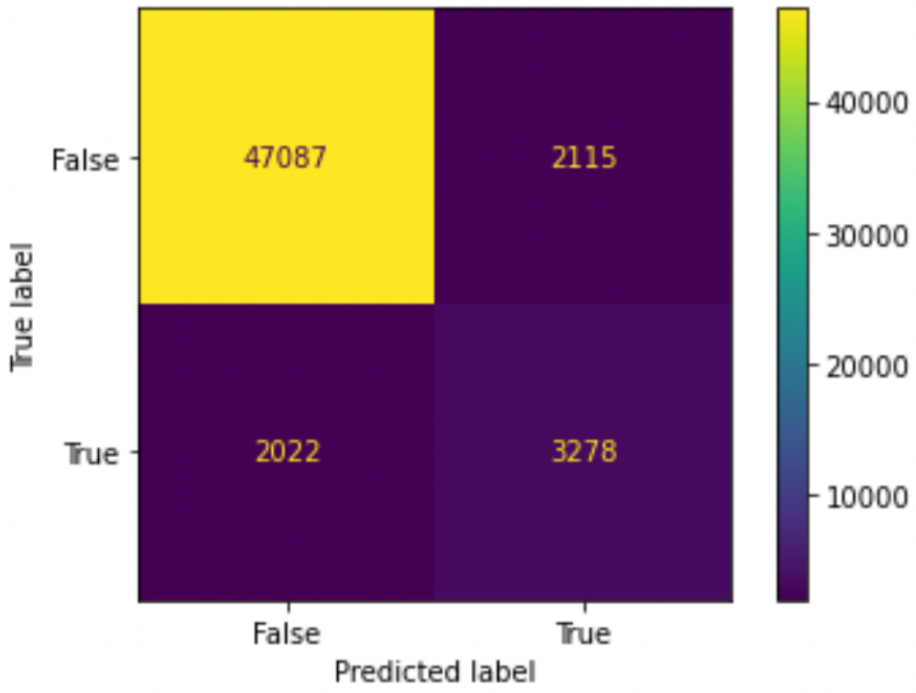
S.No	Model	Accuracy
1	K-nearest neighbors (KNN)	88.69
2	Logistic regression	90.18
3	Naive Bayes	88.63
4	Decision Tree	92.40
5	Neural networks	90.27

Among the above algorithms, we can observe that decision tree has highest accuracy. So, we have used Decision Tree to predict that if the object is going to hit the earth and either it is going to be hazardous or not.

Explanation:

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree- structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules, and each leaf node represents the outcome. We perform split based on a particular attribute at each node. The leaf doesn't split, and they are decision nodes. An optimal decision is made at each node to select a condition and a feature to perform the split.

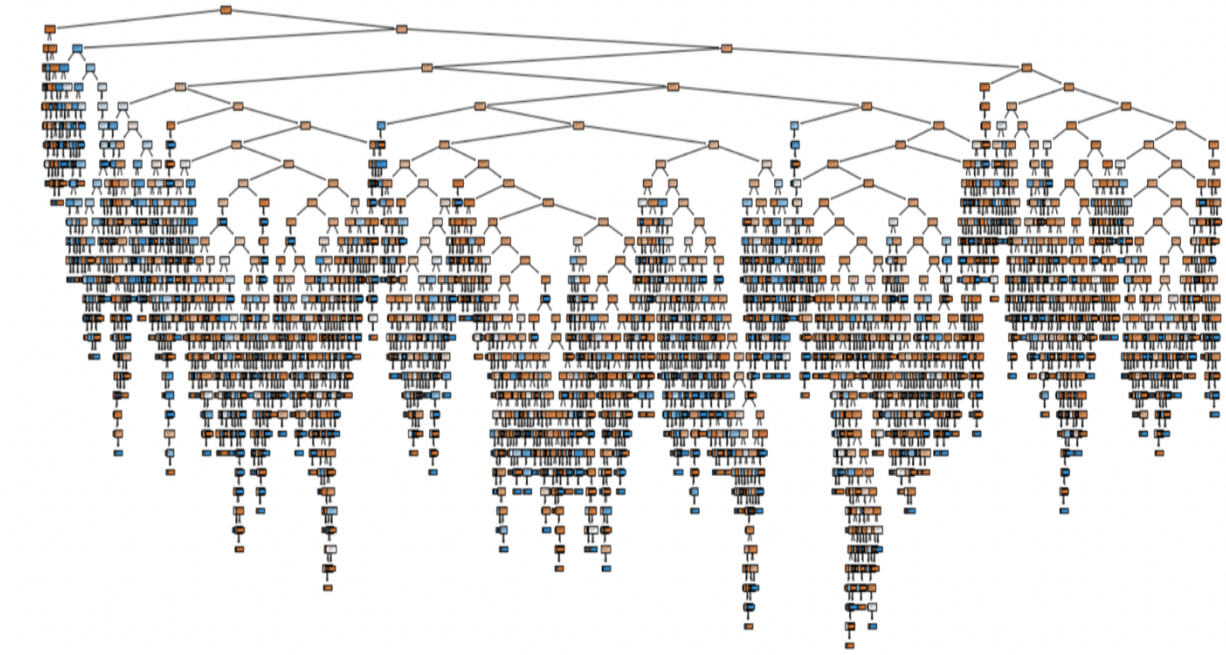
Confusion Matrix:



Evaluation metrics:

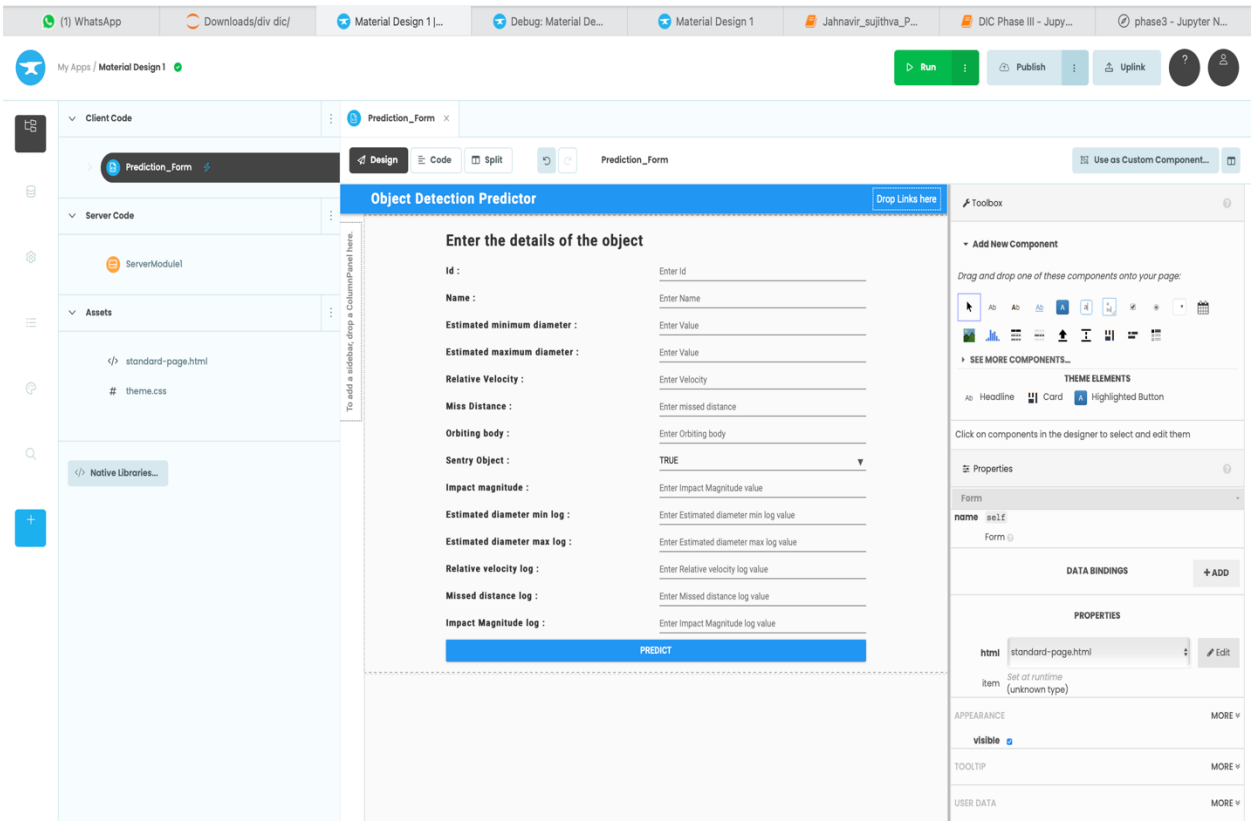
```
Accuracy: 92.40945286411508
Precision: 92.46934630532473
Recall: 92.40945286411508
F1 score: 92.43887117204078
```

Decision tree:



Web App / UI

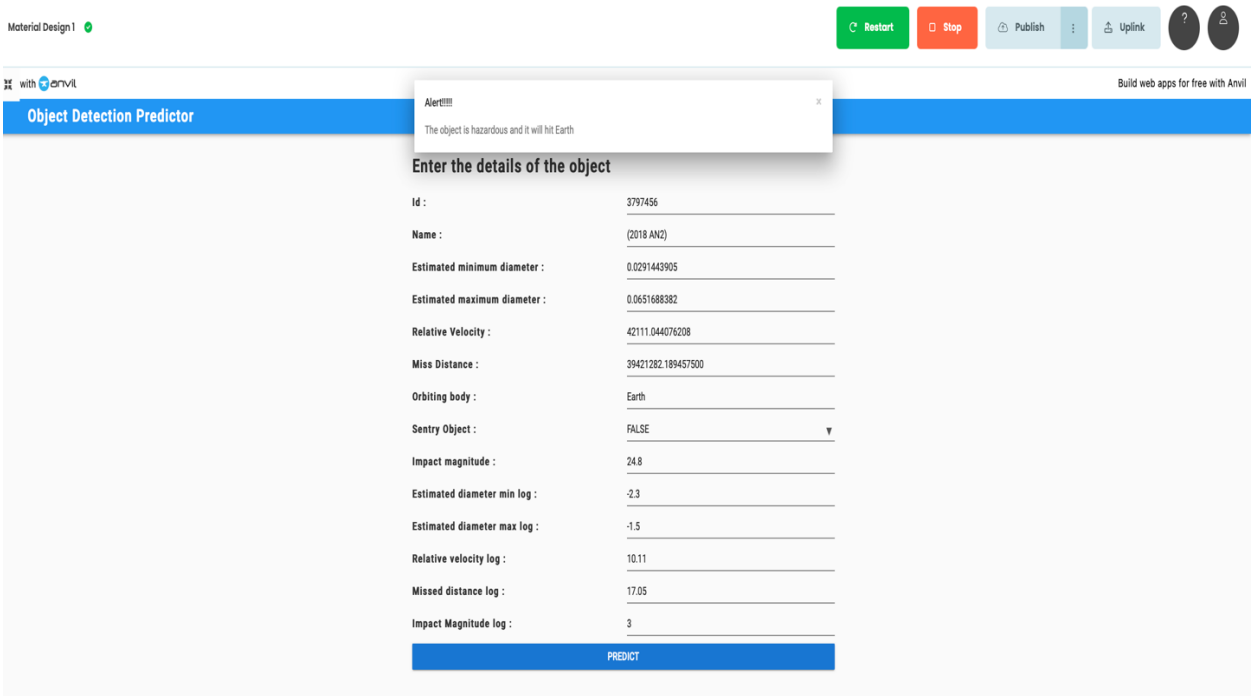
- ➔ We have built a web application which interacts with Jupyter notebook which uses as a back end.
- ➔ For front end, we have used Anvil which is a cloud-based app development platform that enables developers to quickly a create and deploy web applications by using python without the need for server setup or backend programming.
- ➔ Anvil is a drag and drop web app which can built it in python and can be published with a link or URL.



The above form will be available for end users which can be used to predict if any object is on the collision course with earth and if it is hazardous or not. When the formed is filled and the predict button is clicked all the values given by the user will be stored in dictionary (collide) which represents a row in the dataset. The anvil calls the collide function and the prediction is returned as output. Based on the output the notification will be displayed.

There are two outcomes if it is 1 “The object is hazardous, and it will hit Earth” and if it 0 “The object is not hazardous, and it will not hit Earth”.

If the outcome is 1, the outcome will be as in the below snippet (front end:

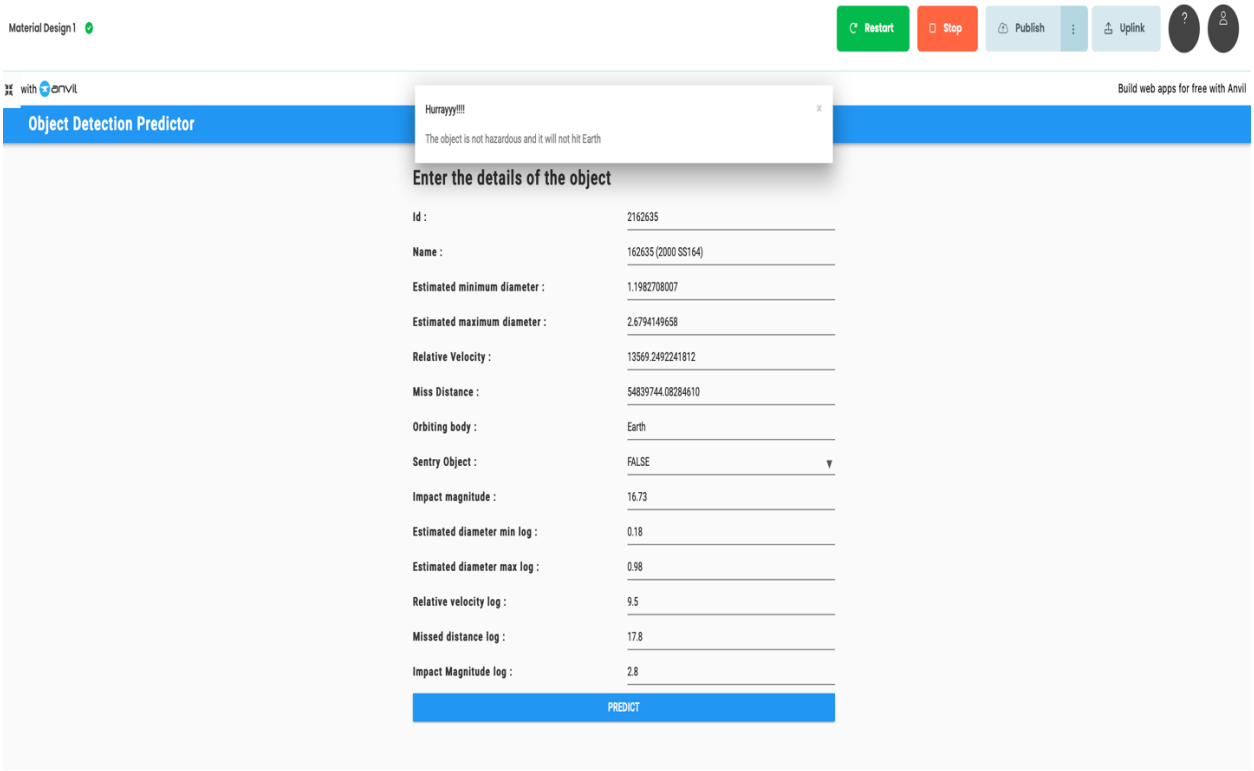


This will be the output in the jupyter notebook(backend):

```
In [*]: #@anvil.server.callable can be accessed by clients connected to an Anvil app server.
        @anvil.server.callable
        def collide(features):
            print("The inputs given by the end user are : , %s!" % features)
            model_inputs = generateListforFeatures(features)
            #print([np.array(model_inputs)])

            ### loading pickle file model
            finalized_model = pickle.load(open(filename,'rb'))
            output = finalized_model.predict([np.array(feature_List)])
            return(output[0])

        #anvil.server.wait_forever() is used to keep the Anvil server running indefinitely,
        #waiting for incoming requests to the predict function.
        anvil.server.wait_forever()
```

```
The inputs given by the end user are : , {'missed_distance_log': '17.05', 'relative_velocity_log': '10.11', '
Impact_Magnitude': '24.8', 'relative_velocity': '42111.044076208', 'obj_id': '3797456', 'Est_dia_MIN_log': '-
2.3', 'orbiting_body': 'Earth', 'obj_name': '(2018 AN2)', 'Est_dia_MAX_log': '-1.5', 'estimation_diameter_min
': '0.0291443905', 'Impact_Magnitude_log': '3', 'sentry_object': 'FALSE', 'estimation_diameter_max': '0.06516
88382', 'miss_distance': '39421282.189457500'}!
```

If the outcome is 0, the outcome will be as in the below snippet (front end):



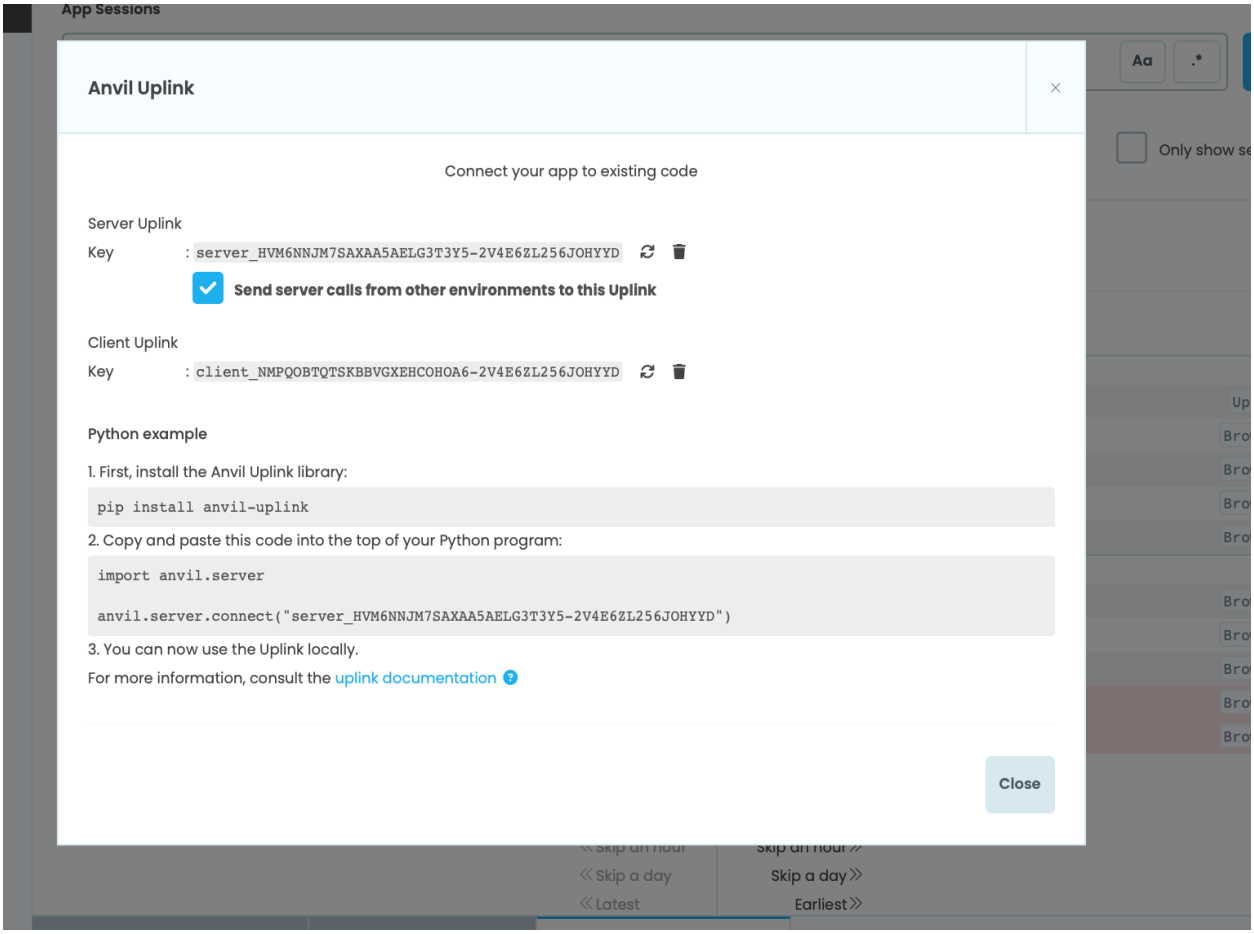
This will be the output in the jupyter notebook(backend):

```
In [*]: #@anvil.server.callable can be accessed by clients connected to an Anvil app server.
        @anvil.server.callable
        def collide(features):
            print("The inputs given by the end user are : , %s!" % features)
            model_inputs = generateListforFeatures(features)
            #print([np.array(model_inputs)])

            ### loading pickle file model
            finalized_model = pickle.load(open(filename,'rb'))
            output = finalized_model.predict([np.array(feature_List)])
            return(output[0])

        #anvil.server.wait_forever() is used to keep the Anvil server running indefinitely,
        #waiting for incoming requests to the predict function.
        anvil.server.wait_forever()
        .0, 3.0]
        The inputs given by the end user are : , {'missed_distance_log': '17.8', 'relative_velocity_log': '9.5', 'Imp
        act_Magnitude': '16.73', 'relative_velocity': '13569.2492241812', 'obj_id': '2162635', 'Est_dia_MIN_log': '0.
        18', 'orbiting_body': 'Earth', 'obj_name': '162635 (2000 SS164)', 'Est_dia_MAX_log': '0.98', 'estimation_diam
        eter_min': '1.1982708007', 'Impact_Magnitude_log': '2.8', 'sentry_object': 'FALSE', 'estimation_diameter_max'
        : '2.6794149658', 'miss_distance': '54839744.08284610'}!
```

Connection between front end and backend:

Front End:

- ➔ We have installed anvil-uplink by using pip install anvil-uplink in Jupyter notebook.
- ➔ We can now get unique uplink key from Anvil which will be connecting to our jupyter code.



By using this uplink key, we can connect to jupyter

```
import anvil.server

anvil.server.connect("server_HVM6NNJM7SAXAA5AELG3T3Y5-2V4E6ZL256JOHYYD")
```

```
Connecting to wss://anvil.works/uplink
Anvil websocket open
Connected to "Default Environment" as SERVER
```

Back End:

As we have chosen Decision tree as our best model, we will store that model in a pickle file (pickling is the process of serializing objects, so they can be saved to a file, sent over network, or saved in a database. They can be trained once and then saved to a pickle file for later user without the need to retrain them).

```
In [115]: ### creating a pickle file
          import pickle
          filename = 'DecisionTree'
          pickle.dump(trees,open(filename,'wb'))
```

Collide function:

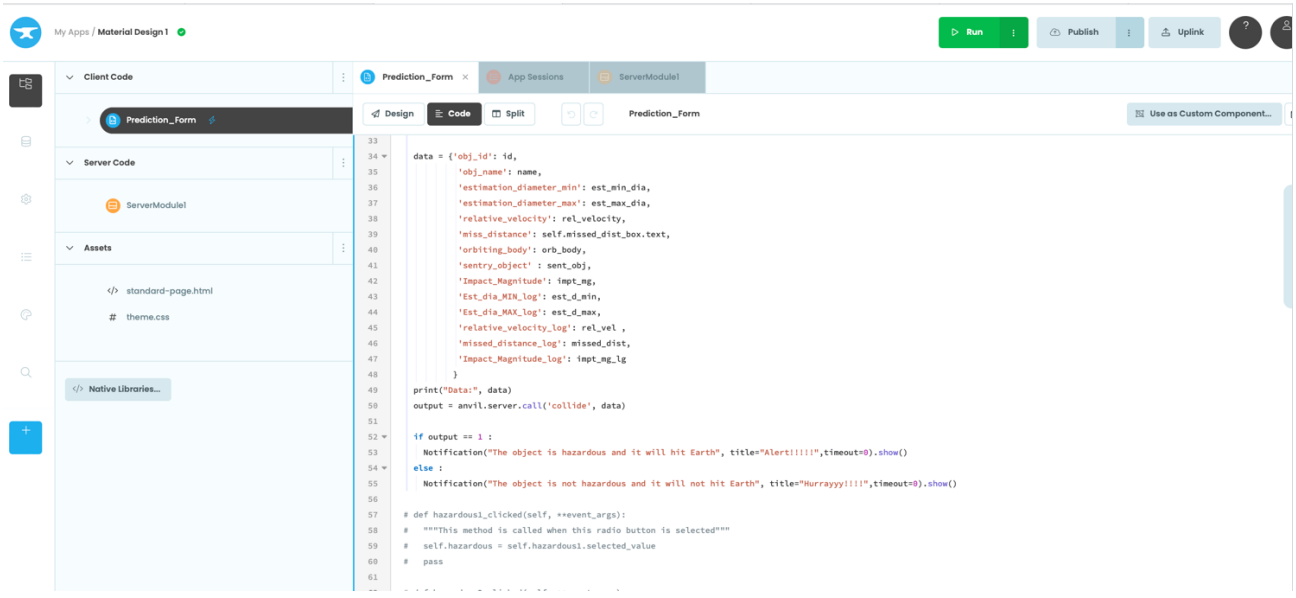
```
In [*]: #@anvil.server.callable can be accessed by clients connected to an Anvil app server.
        @anvil.server.callable
        def collide(features):
            print("The inputs given by the end user are : , %s!" % features)
            model_inputs = generateListforFeatures(features)
            #print([np.array(model_inputs)])

            ### loading pickle file model
            finalized_model = pickle.load(open(filename,'rb'))
            output = finalized_model.predict([np.array(feature_List)])
            return(output[0])

        #anvil.server.wait_forever() is used to keep the Anvil server running indefinitely,
        #waiting for incoming requests to the predict function.
        anvil.server.wait_forever()
```

```
The inputs given by the end user are : , {'missed_distance_log': '17.05', 'relative_velocity_log': '10.11', '
Impact_Magnitude': '24.8', 'relative_velocity': '42111.044076208', 'obj_id': '3797456', 'Est_dia_MIN_log': '-
2.3', 'orbiting_body': 'Earth', 'obj_name': '(2018 AN2)', 'Est_dia_MAX_log': '-1.5', 'estimation_diameter_min
': '0.0291443905', 'Impact_Magnitude_log': '3', 'sentry_object': 'FALSE', 'estimation_diameter_max': '0.06516
88382', 'miss_distance': '39421282.189457500'}!
The inputs given by the end user are : , {'missed_distance_log': '17.05', 'relative_velocity_log': '10.11', '
Impact_Magnitude': '24.8', 'relative_velocity': '42111.044076208', 'obj_id': '3797456', 'Est_dia_MIN_log': '-
2.3', 'orbiting_body': 'Earth', 'obj_name': '(2018 AN2)', 'Est_dia_MAX_log': '-1.5', 'estimation_diameter_min
': '0.0291443905', 'Impact_Magnitude_log': '3', 'sentry_object': 'FALSE', 'estimation_diameter_max': '0.06516
88382', 'miss_distance': '39421282.189457500'}!
```

➔ '@anvil.server.callable' is a decorator in the Anvil platform's server module which is used to define a callable function that can be called from the client-side code of an Anvil web application.

➔ Collide function is decorated with @anvil.server.callable, it becomes available as a server function that can be called from the client-side code.

➔ The client-side code can call this server function using the anvil.server.call()method, passing arguments to the function.



Working Instructions

Step 1:

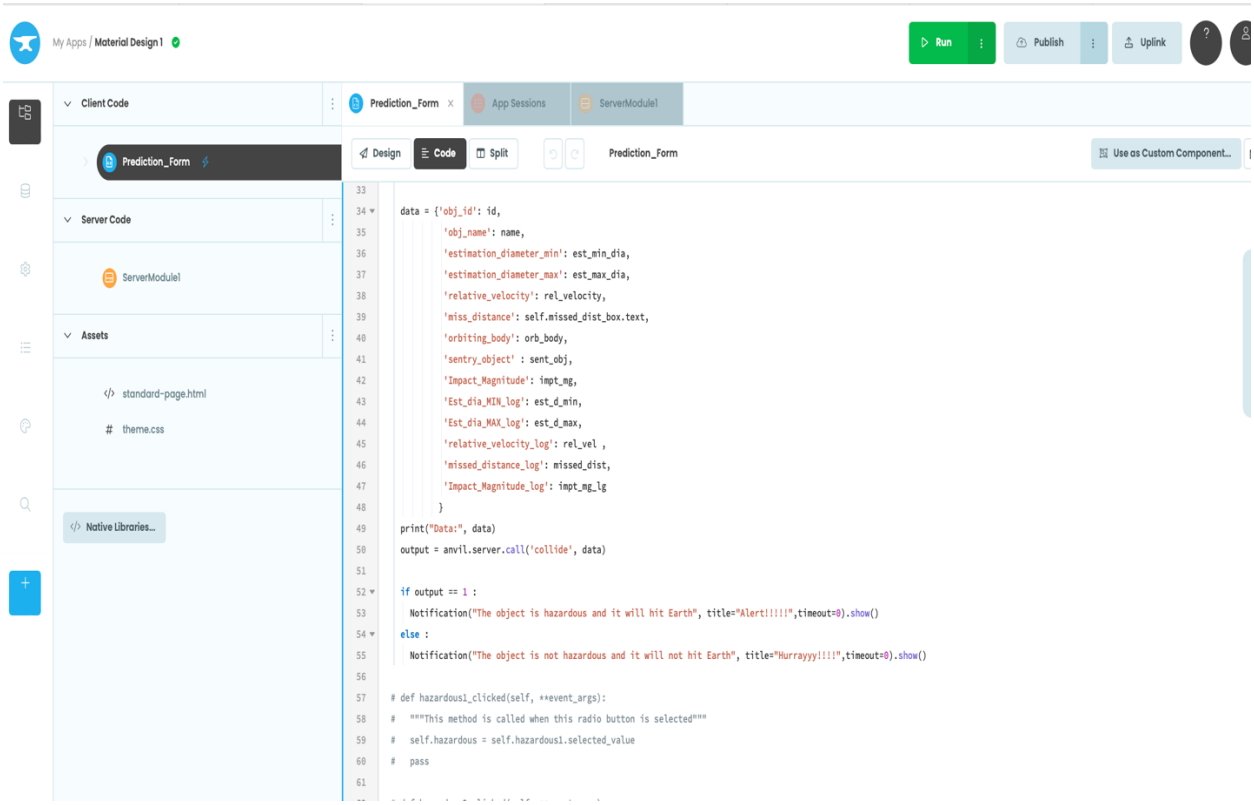
Firstly, open Jupiter notebook and run all the cells.

Note : If anvil-uplink is not installed in your system please make sure you have installed it pip install anvil-uplink.

Step 2:

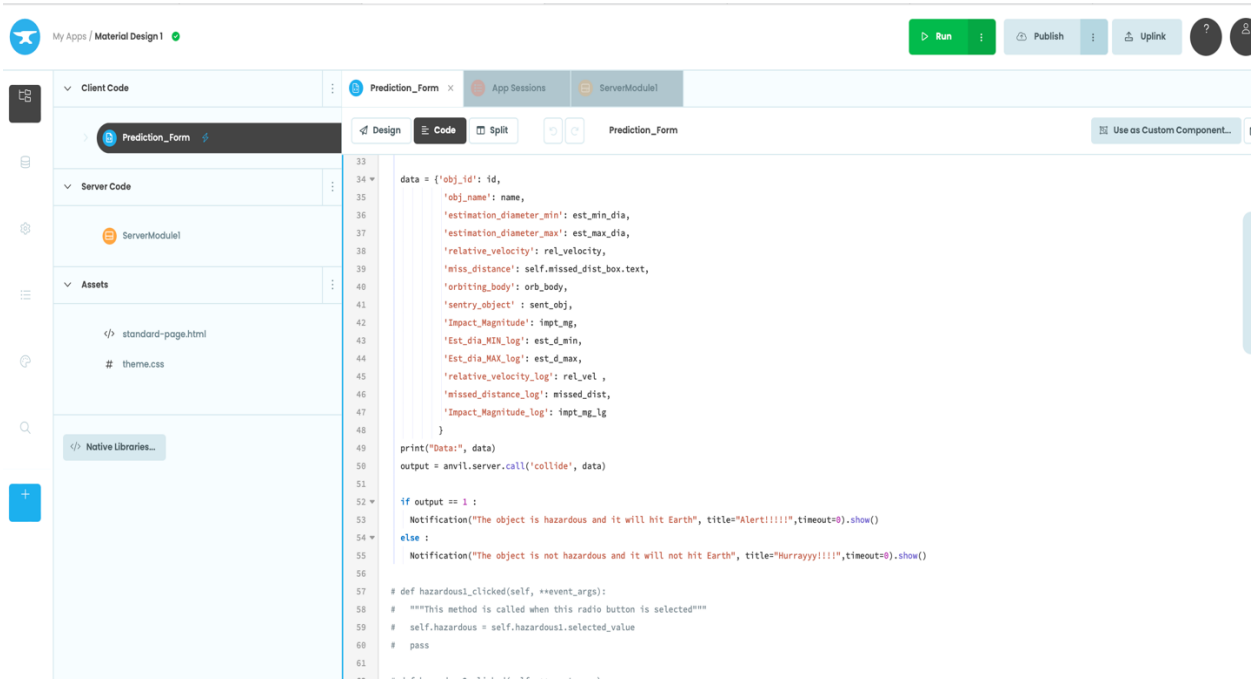
➔ Use this link in any browser : https://space-objects-hazardous.anvil.app

➔ After that it will navigate to the following web page :



Step 3:

Fill all the input fields and click on highlighted predict button.



Recommendations :

The main problem we are going to focus is the detection of objects hitting earth, we focus on small particles of a diameter below 1-2 kms. We analyze the data given by NEO observers. Analyzing whether the object is on collision course with earth, the possible impact zone, and the power of impact of all the space objects based on their rarity, last closest approach to earth, magnitude, and diameter of the object.

- ➔ We predict that if the object which is going to hit the Earth is hazardous or not. The end users who are going to use our product will come to know that if the object which is on collision course with Earth is hazardous or not.
- ➔ It alerts end users to stay in safe zones and avoid damage that may be caused by the incoming object.

Future Scope:

- ➔ This model can be further developed to provide real-time monitoring of hazardous objects, enabling users to receive immediate alerts and take necessary precautions.
- ➔ This model can be integrated with disaster management systems to help officials make informed decisions in the event of an incoming hazardous object. This could potentially save lives and minimize damage.
- ➔ This model can be expanded to predict the hazardousness of objects that are on a collision course with other celestial bodies, such as the moon or other planets.
- ➔ Our product can be expanded to provide additional information about hazardous objects, such as their size, speed, and trajectory, to enable users to make more informed decisions about how to respond.

Running instructions:

- ➔ First you need to install anvil-uplink (pip install anvil-uplink). If it is already installed in your system kindly avoid it.
- ➔ Then run every sell in the jupyter notebook which includes code of all the phases.

References :

- ➔ https://scikit-learn.org/stable/modules/tree (for decision tree)
- ➔ https://youtu.be/yh0B4HjQxOU (for Anvil Web App)
- ➔ https://youtu.be/J63TAIbggZk (for Anvil Web App)