

IMPLEMENTATION PART:-

Launch VS Code.

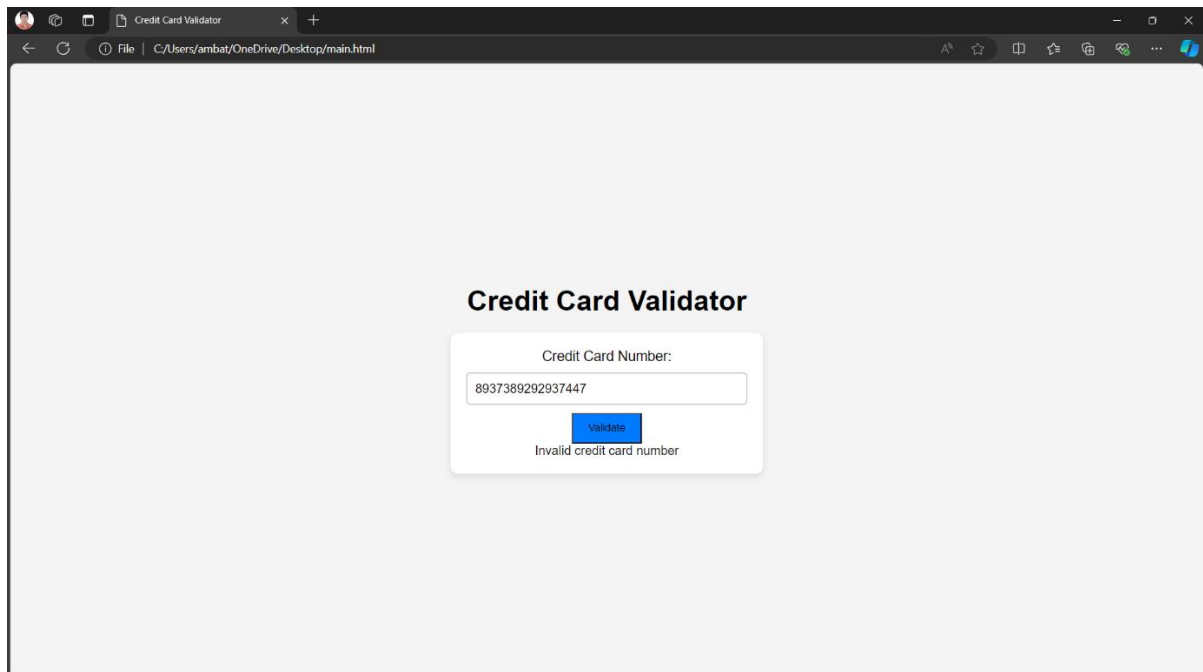
Open the project folder by selecting File > Open Folder and choosing the folder containing your downloaded files.

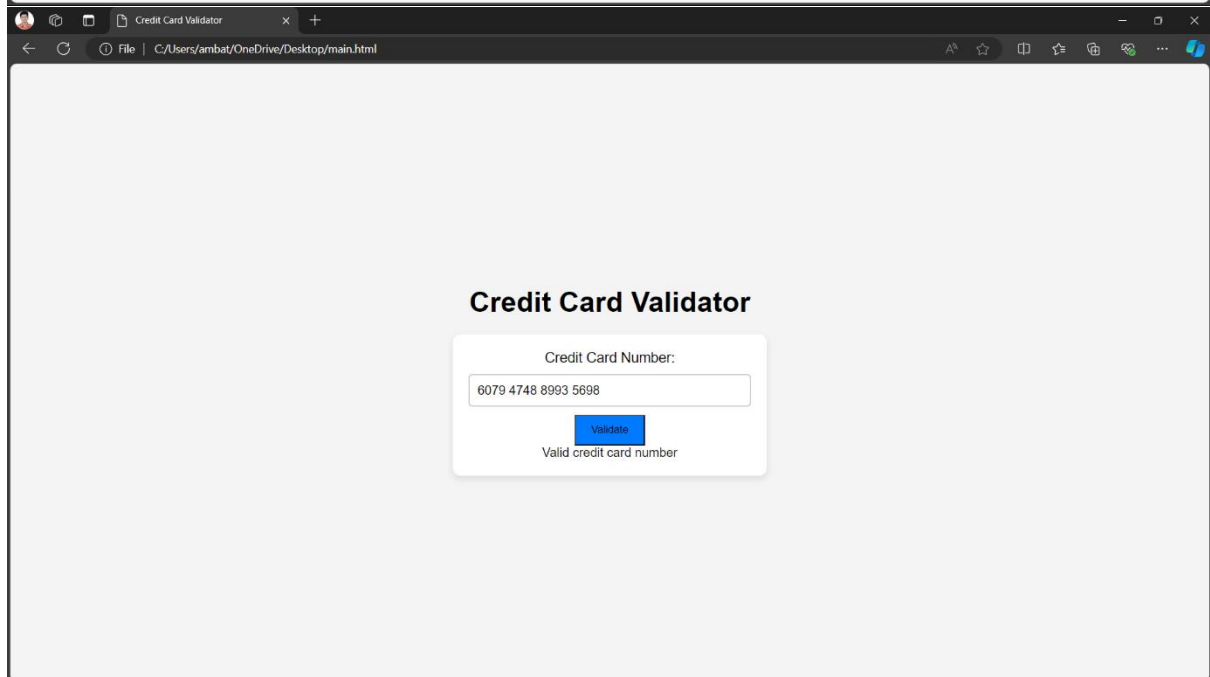
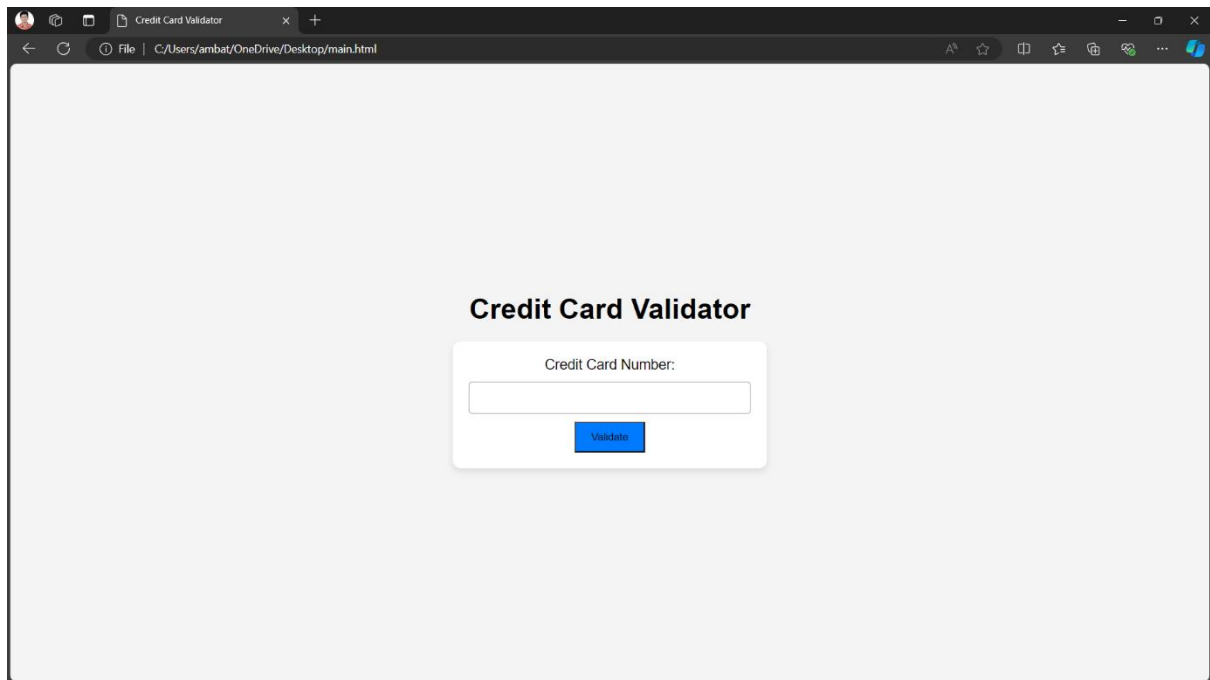
Open index.html in VS Code:

Locate the index.html file in the VS Code file explorer.

Right-click on index.html and Run it by clicking start debugging it will open in a web browser you can take any card number and check the validation.

Now we have implemented this code using Luhns algorithm . This algorithm is considered

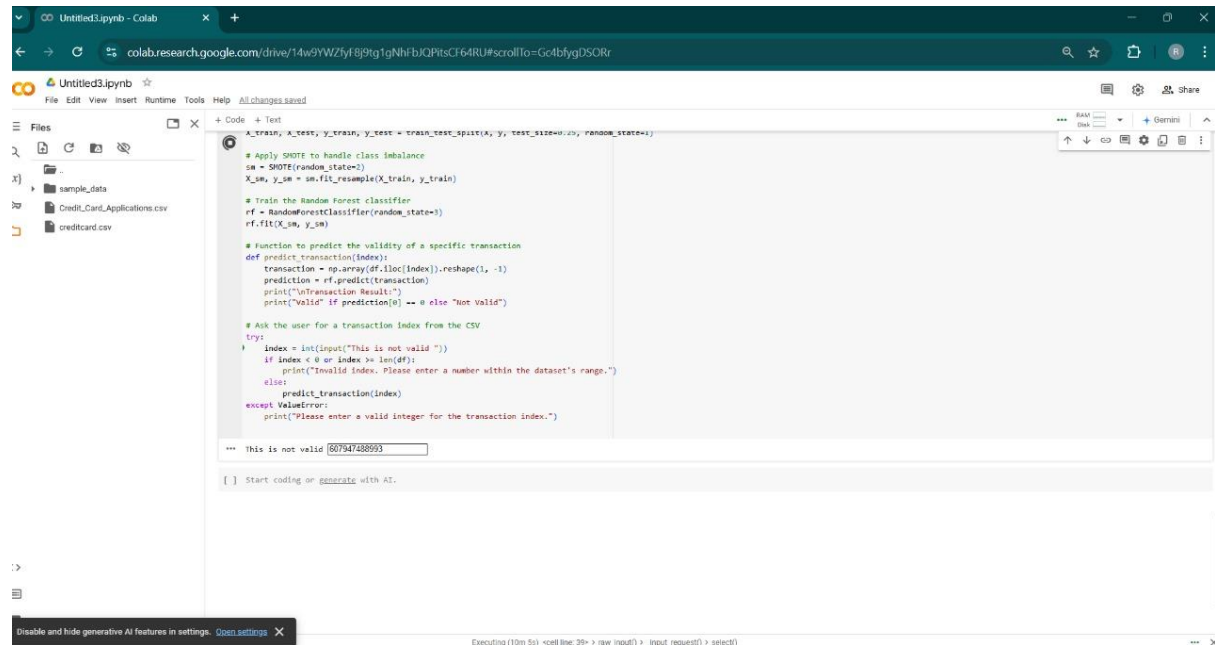




Now we will solve this using Random forest algorithm:-

In Google colab upload the csv file and and run the code.

You will get following outputs.



The screenshot shows a Google Colab notebook titled 'Untitled3.ipynb'. The left sidebar displays the file explorer with 'sample_data' containing 'Credit_Card_Applications.csv' and 'creditcard.csv'. The main code area contains the following Python code:

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

# Apply SMOTE to handle class imbalance
sm = SMOTE(random_state=1)
X_sm, y_sm = sm.fit_resample(X_train, y_train)

# Train the Random Forest classifier
rf = RandomForestClassifier(random_state=3)
rf.fit(X_sm, y_sm)

# Function to predict the validity of a specific transaction
def predict_transaction(index):
    transaction = np.array(df.iloc[index]).reshape(1, -1)
    prediction = rf.predict(transaction)
    print("Transaction Result:")
    print("Valid" if prediction[0] == 0 else "Not Valid")

# Ask the user for a transaction index from the CSV
try:
    index = int(input("This is not valid "))
    if index < 0 or index > len(df):
        print("Invalid index. Please enter a number within the dataset's range.")
    else:
        predict_transaction(index)
except ValueError:
    print("Please enter a valid integer for the transaction index.")
```

Below the code, there is an input field with the text "This is not valid" and a text box containing the value "607947488993". At the bottom of the notebook, a status bar indicates "Executing (10m 5s) <cell line: 36> > raw_input() > _input_request() > select()".

Implementation

HTML (Structure)

The HTML file sets up the structure for the Credit Card Validator. Here's a breakdown:

- The `<!DOCTYPE html>` and `<html lang="en">` tags define the document type and language.
- In the `<head>` section, the page title is set as "Credit Card Validator," and a link to the `style.css` file is included to apply styling.
- The main content is wrapped in a `<div>` with the class `container`, containing a form with:
 - A label and input field (`<input type="text">`) where users can enter a credit card number.
 - A submit button to trigger the validation.
- A `<div>` with the ID `result` is included to display the validation outcome (either "Valid credit card number" or "Invalid credit card number").
- The `<script src="script.js">` tag at the bottom includes the JavaScript file to enable validation functionality.

CSS (Styling)

The CSS styles make the interface visually appealing and responsive:

- A reset style is applied to remove default margins and padding, ensuring uniform appearance.
- `body` is styled with a background color and centered layout, using flex properties to vertically and horizontally center the content.
- The `.container` `div` is styled to appear as a card with rounded corners, a shadow, and padding, enhancing visual appeal.
- The form elements (label, input, and button) are styled for readability and usability:
 - Labels and input fields are sized and spaced for ease of use.
 - The button has a distinct blue background color, giving it a clickable appearance.

JavaScript (Functionality)

The JavaScript file (`script.js`) handles the core functionality of credit card validation:

- **Function `validateCreditCardNumber(ccNum)`:** This function takes a credit card number as input, removes any non-numeric characters, and uses the Luhn algorithm to determine its validity:
 - The number is reversed and iterated over.
 - Every second digit (starting from the right) is doubled, and if this results in a two-digit number, 9 is subtracted.
 - All resulting digits are summed, and if the total is divisible by 10, the card number is valid.

- **Event Listener on Form Submit:** When the form is submitted:
 - `e.preventDefault()` prevents page reload.
 - The `validateCreditCardNumber` function is called with the entered number.
 - The `resultDiv` displays either "Valid credit card number" or "Invalid credit card number" based on the validation result.