**Scripts / Model Execution**

[**1. How is our solution delivered 1**](#_n5l8v33wwhij)

[**2. How to run our code using Github 1**](#_bxd2k1imn10d)

[**3. How to run our code on Azure 3**](#_42rvnwx8stin)

[**4. Script description: 3**](#_vba9jsb8ze32)

[**5. Troubleshooting/ Where to get help ? 4**](#_ci3dj9rqq7bq)

[5.1. General Setup Issues 4](#_wdfc9padjppi)

[5.2. Local Code Execution Issues 5](#_xrrh7frqkmv4)

[5.3. Azure Deployment or Execution Issues 5](#_5l6kvhpuvaam)

[5.4. Data-Related Issues 6](#_mx80evkmwzq9)

[5.5. Where to Get Help 6](#_qhv84kr3ybui)

[**6. End-Users 6**](#_1mr4qs8s8lu9)

[6.1. Business Analysts and Data Scientists 7](#_qdfyr5fx5vqu)

[6.2. Data Engineers and IT Operations Teams 8](#_98ya69rfu1xo)

[6.3. Business Stakeholders and Decision-Makers 8](#_573dok2h1kaw)

[6.4. Product Managers / Project Managers 9](#_mc8soazyw49)

[6.5. Compliance and Audit Teams 9](#_3ej10bxv98wx)

[6.6. External Clients or Partners (Optional) 10](#_hvn12qp3tbky)

[**7. Website 10**](#_xpuu6mg0n9hs)

[7.1. How the website works (back end & front end) 10](#_jbkhi3jtgmsl)

[7.2. How to Use the Website 11](#_ts1x6o17yjdn)

[7.3. Functionality 13](#_1ackbp9id8nq)

[7.4. Troubleshooting (In Case the Website Doesn't Function) 14](#_ausnn3o4crmr)

# **1. How is our solution delivered**

Our solution is delivered through a combination of GitHub and Azure services to ensure seamless development, deployment, and execution. Here’s how the process works:

1. GitHub as the Central Repository:

* We use GitHub to host both the codebase and documentation. It acts as the central repository for our development team.
* GitHub enables version control, allowing our team to track changes, collaborate efficiently, and maintain an organised workflow.
* ~~Furthermore,~~ GitHub plays a crucial role in automating the deployment of our infrastructure and scripts to Azure, making the process streamlined and efficient.

1. Development and Testing:

* While Azure serves as the remote environment where we run our code, all development and testing of the Python scripts are carried out locally. This allows for a faster development cycle, ensuring that we refine the scripts before deployment.
* Once the scripts are thoroughly tested and ready, they are hosted on Azure to interact with the data stored in Azure Data Lake Gen2.

1. Azure as the Execution Environment:

* Azure is utilised as a remote server where we run our finalised scripts.
* The raw data is hosted on Azure Data Lake Gen2, and Azure Functions are employed to run Python scripts that clean, enrich, and synthesise the data.
* After the pipeline is executed, the final processed dataset is produced and stored on Azure.

In summary, GitHub serves as the foundation for development, documentation, and deployment, while Azure provides the execution environment to process and generate the final dataset. This approach ensures that both code and data are handled securely and efficiently, with proper version control and automation in place.

# **2. How to run our code using Github**

To run the code from our GitHub repository, follow these steps:

**Step 1: Clone the Repository**

First, you will need to clone the repository to your local machine. Use the following command in your terminal:

`git clone https://github.com/AlanDataPortfolio/ey-azure-fn-pipeline.git`

This will download the entire codebase to your local environment.

**Step 2: Run the Code**

There are two methods to execute the code:

**Method ONE: Run All Scripts with a Single Command (in development)**

We have provided a master script that automates the process by executing all the other scripts in the correct sequence. This is the simplest and fastest method to run the pipeline.

To run the master script, navigate to the project directory and execute the following command:

`bash run\_all.sh` for Unix

`pwsh run\_all.ps1` for Windows

This script will automatically run all the necessary scripts to set up the infrastructure, clean the data, enrich it, synthesise the results, and merge them into the final dataset.

### 

**Method TWO: Run Each Script Individually**

***SO WHAT THE MASTER SCRIPT IS DOING, THEY CAN DO THIS MANUALLY.***

Alternatively, you can run each script manually if you prefer more control over the execution process. To do this, navigate to the src folder and run each script in the order below:



This method gives us the flexibility to review and monitor each step individually. This method will run some of the scripts in parallel. We can refer them as stages as seen in the flowchart. In stage 1, the cleaning scripts for the 3 datasets can run at the same time. After cleaning the datasets, we can move on to stage 2 where we enrich the 3 datasets. The enriching process is essentially refining the cleaned datasets. The enriching scripts can also run in parallel. Then, we move on to stage 3 where we merge dataset 1 and 2. The merging is done so that data can be synthesised based on datasets 1 and 2. In this stage, dataset 1 and dataset 2 is combined together. This script cannot run parallel to other scripts. Stage 4 is where data synthesisation takes place. Data is synthesised based on the merged dataset. Two synthesisation methods are used and the scripts for these two methods can run parallel to each other. Finally, we move on to the last stage which is stage 5. Here, all of the datasets are merged together to have dataset consisting of 20,900 rows approximately. In this stage, the 3 enriched datasets and the 2 synthesised datasets are combined together to create a comprehensive dataset.

# **3. How to run our code on Azure**

Our scripts are hosted on Azure Functions and can be triggered by making a POST request to a private endpoint secured with an authorisation key. There are two ways you can run the scripts via Azure:

**Method TWO: Make a Pull Request to Activate GitHub Actions (in development)**

Another way to run the code on Azure is to make a pull request (PR) to the GitHub repository. This will automatically trigger a GitHub Actions workflow that will make the POST request to Azure Functions on your behalf. Here's how you can do it:

Make changes to the code (or leave a comment) and create a pull request in the repository.

Once the PR is submitted, GitHub Actions will be activated, and the necessary POST request will be made to trigger the Azure Functions pipeline.

This method allows for fully automated script execution through GitHub's continuous integration workflow.

# 

# **4. Script description:**



Our scripts will run in stages. In stage 1, the raw datasets are loaded and are being cleaned. Universal paths are used in all of the scripts to ensure that the code runs in any environment. In stage 1, missing values are filled in using logic and imputation methods. The **columns named are** renamed using camelCase for consistency across the datasets. In this stage, it is also ensured that ‘None’ is not treated as null. After data cleaning is complete, three csv files are created which will then be used in stage 2.

In stage 2, our data enrichment process takes place. The cleaned datasets are loaded in from stage 1 and the datasets are refined. In this stage, data transformation takes place such as feature engineering and data distribution modelling. Key columns are adjusted as well like the insurancePremium columns are multiplied by 12 to convert it to an annual amount. Binary categorical columns such as Yes and No are converted to 1 and 0 respectively. For example, policeReportBool which has ‘YES’ and ‘NO’ outputs are converted to ‘1’ and ‘0’ respectively. In this stage, some columns are created based on the distribution and logic. For example, the vehicleAge column was created by subtracting the vehicleYear from 2015 which was the year the dataset was based on. Finally, this stage also filters out the columns that we deemed were unnecessary for our model. This script will create 3 csv files and hence will complete our enriching process.

In stage 3, we merge enriched datasets 1 and 2 to combine them for data synthesisation. For synthesisation, we do not use dataset 3 as it has a lot of columns missing compared to our base dataset which is dataset 1. This script essentially loads both dataset 1 and 2 and uses the concat function from pandas to combine both the datasets and creates a csv file containing 2575 rows of data. An index is also created to keep track of the specific rows.

In stage 4, our data synthesisation takes place. The synthesisation scripts can run parallel to each other. In our synthesisation process, we use datasets 1 and 2 to synthesise more rows. We use two methods to synthesise the data. The first method directly samples the data from the original dataset’s distribution and some noise is added to ensure variance. The second method uses two neural networks to synthesise the data. This model is called a CTGAN model where synthetic data is created by learning the underlying distribution of the datasets. The two methods creates 4000 rows of data each which adds up to 8000 rows of synthesised data. The two scripts will create two csv files also with an indexed column to keep track of the rows.

In our final stage, all of our datasets are combined together. Enriched dataset 1, 2, and 3 are combined with synthesised methods 1 and 2 to create 20877 rows of data which is our final dataset. Like stage 3, we also use the concat function from pandas to combine the five datasets. This script also handles any potential issues regarding indexing. It ensures that the index starts from 1 and does not reset. This script will create a csv file named 'merged\_20000\_rows.csv' and this is the final dataset that will be used to train the AI model.

# 

# **5. Troubleshooting/ Where to get help ?**

## 5.1. General Setup Issues

* **Git Clone Error**:
  + **Issue**: Unable to clone the GitHub repository.
  + **Solution**: Make sure Git is installed on your machine. Use the command git --version to verify the installation. Ensure you have internet access and correct permissions for the repository.
* **Python or Module Not Found Error**:
  + **Issue**: Error related to missing Python packages like pandas or numpy.
  + **Solution**: Ensure Python is installed and accessible via the command line. Use python --version to confirm.
  + Install any missing packages using:  
    Code: pip install -r requirements.txt
  + If you encounter permission issues, try:  
    Code : pip install --user -r requirements.txt

## 5.2. Local Code Execution Issues

* **Master Script (run\_all.sh or run\_all.ps1) Errors**:
  + **Issue**: Scripts are not running correctly or terminating early.
  + **Solution**:
    - Make sure you have the necessary permissions to run shell or PowerShell scripts (use chmod +x run\_all.sh on Unix).
    - Verify that all required Python modules are installed.
    - Check for typos in file paths and ensure that paths match your operating system’s format (e.g., forward vs backward slashes).
* **Missing Permissions for PowerShell Scripts on Windows**:
  + **Issue**: PowerShell scripts are blocked by default.

**Solution**: Enable script execution with the following command:  
powershell  
Code: Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser

* **Parallel Execution Errors**:
  + **Issue**: Some datasets aren't processed correctly during parallel execution.
  + **Solution**: Verify that all input files are correctly formatted and placed in the expected directories. If necessary, run each stage individually to identify the point of failure.

## 

## 5.3. Azure Deployment or Execution Issues

* **Azure Function Not Triggering**:
  + **Issue**: The Azure function isn’t executing after a pull request or POST request.
  + **Solution**:
    - Ensure the authorisation key is valid and correctly included in the POST request.
    - Check the Azure Function logs for any errors using the Azure portal.
    - Verify that the function app service is running and that the private endpoint is properly configured.
* **GitHub Actions Not Triggering**:
  + **Issue**: The PR doesn’t activate the GitHub Action workflow.
  + **Solution**:
    - Ensure that the workflow file exists in .github/workflows/ and follows proper syntax.
    - Make sure the repository settings allow actions and the triggers (like pull requests) are correctly set.

## 

## 5.4. Data-Related Issues

* **Data Not Found or Incorrect Paths**:
  + **Issue**: Scripts can’t locate the raw or processed datasets.
  + **Solution**: Ensure the data is placed in the correct directory and the paths match what is specified in the scripts. Use relative paths wherever possible to avoid OS-specific issues.
* **Unexpected Values in Processed Data**:
  + **Issue**: The enriched or synthesised data contains unexpected values.
  + **Solution**: Review the cleaning and enriching logic. Ensure that missing values are handled properly and categorical columns are converted consistently.

## 

## 5.5. Where to Get Help

* **Documentation**:
  + Review the README file and documentation on the GitHub repository for detailed instructions and usage guidelines.
  + The documentation provides examples and troubleshooting steps for common issues.
* **GitHub Issues**:
  + If the issue persists, raise an issue in the GitHub repository. Provide as much detail as possible, including error messages and screenshots, if applicable.
* **Azure Logs**:
  + Use the Azure Portal to access logs for the Azure Functions to identify runtime errors and misconfigurations.
* **Stack Overflow / Online Forums**:
  + For general programming issues or package-related errors, search on Stack Overflow or consult Python package documentation.

# 

# **6. End-Users**

**User Manual Overview**

| **End User** | **Key Responsibilities** | **Primary Tools** | **Where to Get Help** |
| --- | --- | --- | --- |
| **Business Analysts/ Data Scientists** | Analyse datasets, generate insights , build models | Python, Power BI | GitHub Issues, Data Engineers |
| **Data Engineers** | Maintain and troubleshoot the pipeline | Github, Azure Portal | Logs, Github Issues, Azure Logs |
| **Business Stakeholders** | Use data for decision making | Power BI, Dashboards | Project Manager, IT Team |
| **Product/project Managers** | Monitor progress, coordinate code changes | GitHub Actions, Azure Portas | GitHuns PRs, Status Report |
| **Compliance Teams** | Ensure Compliance and data governance | Github Logs, Azure Logs | Data Engineers, Audit Reports. |
| **External Clients/Partners** | Integrate processed datasets into their systems | CSV/JSON Files, Pythos | Project POC, Data Engineers |

## 

## **6.1. Business Analysts and Data Scientists**

**Purpose:**

To generate insights, build predictive models, and analyse the final datasets.

### 

**How to Use the Dataset**

1. **Accessing the Dataset:**
   * Download the final processed datasets from **Azure Data Lake Gen2** using authorised credentials.
   * Navigate through Azure’s Storage Explorer or Data Factory to locate files named, for example, final\_dataset.csv.
2. **Software Requirements:**
   * Python (with packages like pandas, numpy installed).
3. **Steps to Analyse the Data:**
   * Load the dataset into Python:  
     Code:

import pandas as pd

df = pd.read\_csv('path\_to\_final\_dataset.csv')

print(df.head())

* + Use the enriched columns (e.g., vehicleAge, insurancePremium) to perform descriptive analysis.

1. **Best Practices:**
   * Validate column names and value consistency (e.g., camelCase format).
   * Refer to the project documentation for dataset definitions and data quality checks.
2. **Troubleshooting:**
   * If datasets are missing or incomplete, contact the **Data Engineers** or refer to the GitHub repository’s **Issues** section.

## 

## **6.2. Data Engineers and IT Operations Teams**

**Purpose:**

To maintain, troubleshoot, and improve the data pipeline while managing the infrastructure.

**Pipeline Execution Guide:**

1. **Pre-requisites:**
   * Ensure access to the **GitHub repository** and **Azure portal**.
   * Install required packages using:
   * Code: pip install -r requirements.txt
2. **How to Run the Code Locally:**
   * Clone the repository:
   * Code: git clone <https://github.com/AlanDataPortfolio/ey-azure-fn-pipeline.git>

Execute the **master script** to run the entire pipeline:  
 Code:   
 bash run\_all.sh # Unix

pwsh run\_all.ps1 # Windows

1. **How to Monitor the Azure Functions:**
   * Log in to the **Azure portal**.
   * Navigate to the **Function App** and check the **Monitor** tab for logs and errors.
   * Ensure proper authorisation keys are in place for triggering the function.
2. **Handling Failures:**
   * If any script fails, manually run individual stages to debug:  
     Code: python src/clean\_dataset1.py
   * Check the Azure **Activity Log** and GitHub Actions logs for detailed error messages.

## 

## **6.3. Business Stakeholders and Decision-Makers**

**Purpose:**

To consume high-level insights from the final dataset to inform & support strategic decisions.

**How to Access and Use the Data**

1. **Where to Find the Data:**
   * The final dataset is hosted on **Azure Data Lake Gen2**.
   * IT Operations will provide access links or share reports derived from the dataset.
2. **How to Use the Data for Decision-Making:**
   * Review the summary metrics, trends, and key indicators provided in reports or dashboards.
   * Use the enriched and synthesised data (e.g., vehicle age, insurance premiums) to assess risk, performance, or market opportunities.
3. **Requesting Additional Data:**
   * If further data processing or custom reports are needed, request specific modifications through the **Project Manager** or raise a ticket with the **Data Engineering Team**.

## 

## **6.4. Product Managers / Project Managers**

**Purpose:**

To monitor the project progress, ensure deliverables align with business needs, and manage code changes.

**Managing GitHub Workflow and Deployment**

1. **Monitoring GitHub Actions:**
   * Navigate to the repository’s **Actions** tab on GitHub.
   * Check the status of automated workflows triggered by pull requests.
   * If a job fails, review logs to identify the issue.
2. **How to Manage Code Changes:**
   * Ensure all code changes follow the required pull request (PR) review process.
   * Coordinate with developers to ensure all stages in the pipeline are functional before merging code into the main branch.
3. **Accessing Azure Logs and Reports:**
   * Use the **Azure portal** to monitor pipeline executions and view status logs.
   * Schedule regular status reports with the Data Engineering team to ensure smooth operations.

## **6.5. Compliance and Audit Teams**

**Purpose:**

To ensure the data processing pipeline adheres to regulatory standards and is auditable.

**How to Audit the Data Pipeline**

1. **Access to Logs and Execution Reports:**
   * Review the **GitHub Actions logs** for version control and code change history.
   * Use the **Azure portal** to access the logs for each function execution.
2. **Verify Data Lineage:**
   * Confirm that the raw data from Azure Data Lake Gen2 matches the cleaned and enriched datasets.
   * Review pipeline documentation to ensure transformations and enrichments are documented and traceable.
3. **Compliance Checklist:**
   * Ensure all personal or sensitive data is handled per regulatory guidelines (e.g., GDPR, CCPA).
   * Validate that **access control** policies are enforced, with authorised access only.

## **6.6. External Clients or Partners (Optional)**

**Purpose:**

To consume processed datasets for integration with their systems or analysis.

**How to Access the Datasets**

1. **Data Access:**
   * IT will share secure links or credentials to access **Azure Data Lake Gen2**.
   * Datasets will be available in formats like **CSV** or **JSON**.
2. **How to Integrate the Data:**
   * Use standard data loading processes to import datasets into your systems. Example for Python:
   * Code: import pandas as pd

data = pd.read\_csv('https://secure\_link\_to\_dataset.csv')

* + Ensure your system supports the column naming conventions (e.g., camelCase).

1. **Where to Report Issues:**
   * Contact the project’s point of contact (POC) or raise an issue with the Data Engineering team for support with integration.

# **7. Website**

## 7.1. How the website works (back end & front end)

The website is built using Next.js, a React framework that allows for server-side rendering and API integration. The back end of the website consists of two main components:

* Serverless API endpoints that handle the fetching and updating of claim data from a CSV file.
* Front-end user interface implemented with React, which displays the form and handles user interactions.

Back End (Server-Side Logic)

* Data Storage: The data is stored in a CSV file named claims.csv, which includes fields for claim information such as claim status, claim outcome, description, and other details.
* APIs:
  + /api/getClaim: This API fetches the first open and pending claim from the CSV file and returns it to the front end for display. It uses the csv-parser library to read the CSV file.
  + /api/updateClaim: This API updates the claim status and outcome in the CSV file. It first reads the CSV file to locate the specified claim, modifies the fields, and writes the updated data back to the file using the json2csv library.

Front End (React Components)

* The front end is built with React components that include form elements for claim information, claim outcome, fraud evaluation, and related actions.
* State Management: The application uses React's useState for managing the state of form fields such as claim details, fraud risk score, and claim description.
* Button Functionality:
  + "Get Claim" Button: Fetches the first open and pending claim using the /api/getClaim endpoint.
  + "Close Case" Button: Updates the claim outcome to "approved" or "denied," and sets the claim status to "closed" via the /api/updateClaim endpoint.
  + "Escalate to Manager" Button: Updates the claim outcome to "escalated" while leaving the claim status as "open."
  + "Check Fraud" Button (Not Functional Yet): Intended to run a fraud detection check on the current claim using the /api/checkFraud endpoint, returning a fraud risk score and analysis summary.

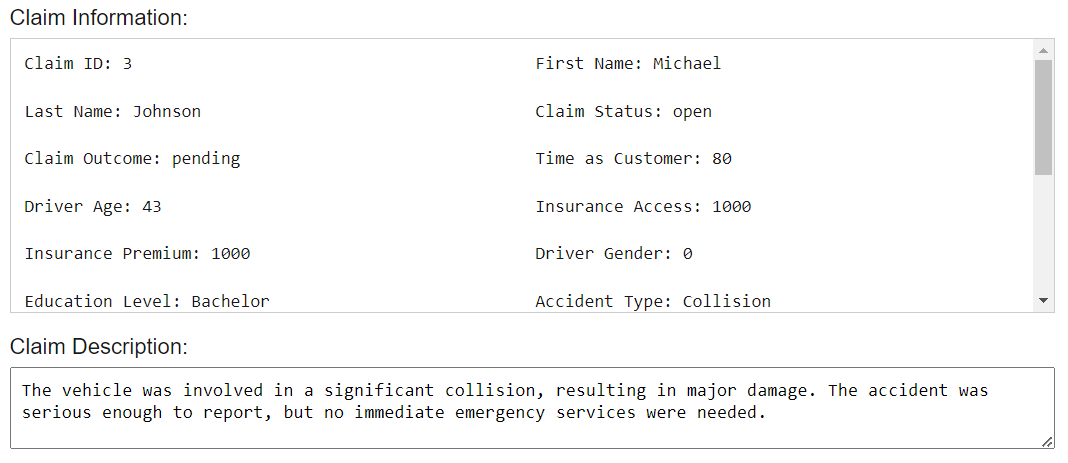
## 

## 7.2. How to Use the Website

1. Accessing the Website:
   * Navigate to the website URL or open the local development server (http://localhost:3000) when running locally.
2. Fetching a Claim:
   * Click on the "Get Claim" button to load the first open and pending claim from the database (CSV file). The claim details will automatically populate the form fields.



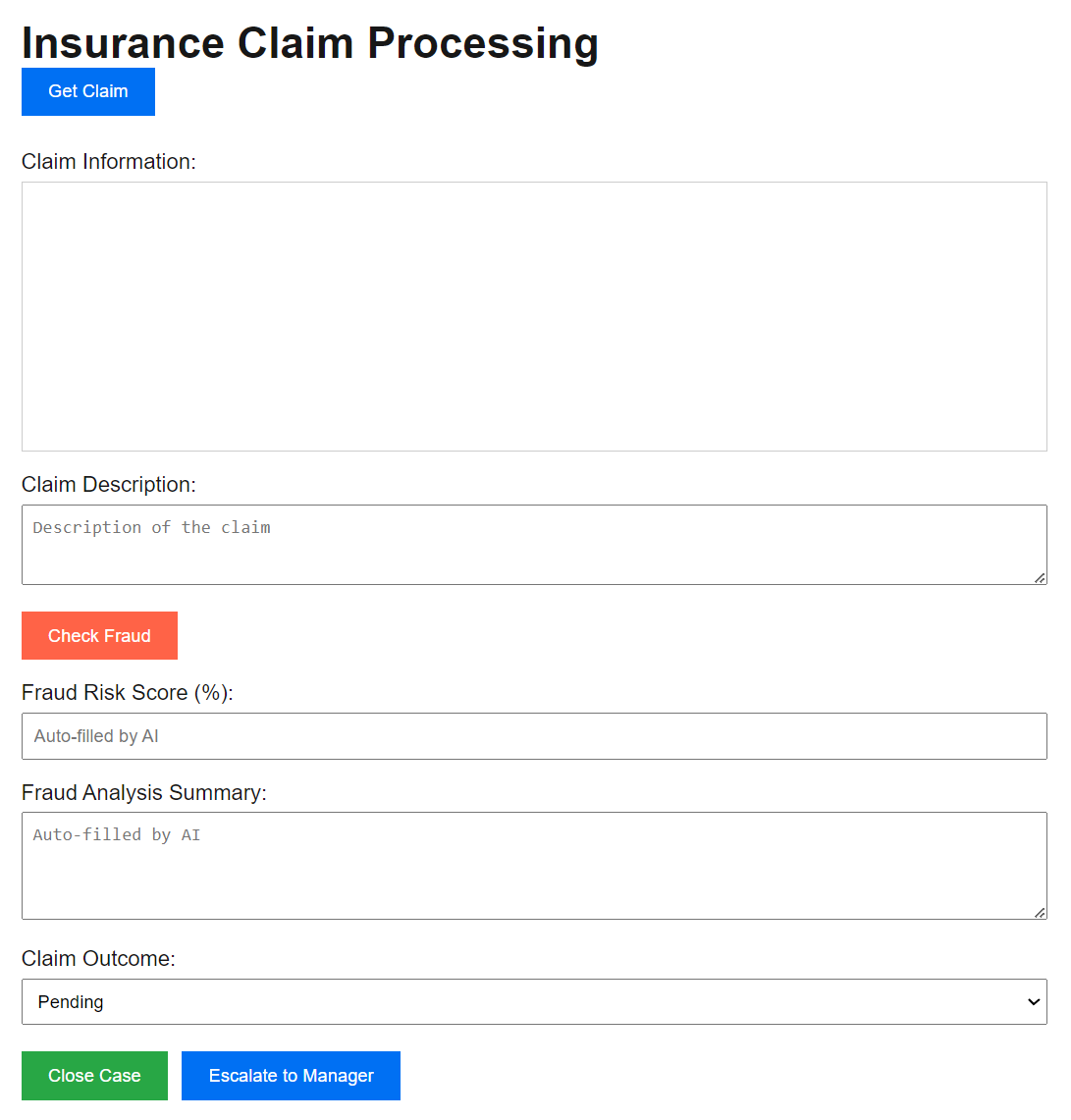
1. Viewing Claim Information:
   * The fetched claim information will be displayed in a formatted text area, including details such as Claim ID, customer information, and claim description.
   * The Claim Outcome dropdown will be populated based on the fetched claim.



1. Checking for Fraud (Not Functional Yet):
   * Click the "Check Fraud" button to run a fraud detection check on the current claim.
   * This will return a fraud risk score (%) indicating the likelihood of fraud, along with a brief analysis summary explaining the key factors influencing the score, such as unusual claim patterns or inconsistencies.

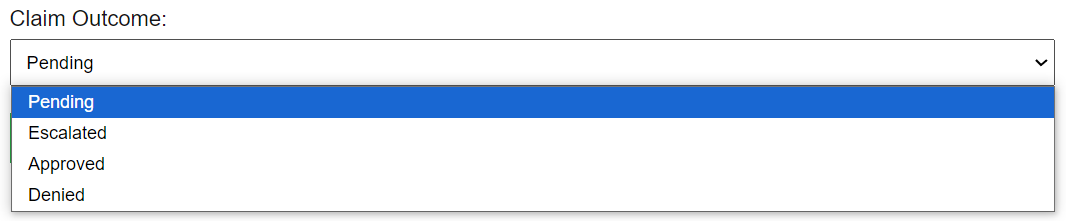


1. Updating the Claim:
   * Choose an option from the Claim Outcome dropdown (Pending, Approved, Denied, or Escalated).
   * Click on "Close Case" to mark the claim as resolved, or "Escalate to Manager" to flag the claim for further review.
2. Clearing the Form:
   * After updating the claim (closing or escalating), the form fields will be automatically cleared for the next operation.



## 7.3. Functionality

* Claim Management: The website allows users to manage insurance claims by updating claim outcomes and statuses.
* Automated Field Population: The form auto-populates when a claim is fetched, reducing manual input and potential errors.
* Data Persistence: Changes to the claims are saved back to the CSV file, ensuring data consistency.
* Fraud Detection): A "Check Fraud" button is included to eventually allow automated fraud risk analysis. (This feature is currently under development).
* Flexible Claim Actions:
  + Closing the Case: Automatically sets the claim to "closed" if the outcome is "approved" or "denied."
  + Escalation: Marks the claim as "escalated" while keeping it open for further processing.





## 7.4. Troubleshooting (In Case the Website Doesn't Function)

1. Website Not Loading:
   * Ensure the development server is running by executing npm run dev in the project directory.
   * Confirm that Node.js is properly installed and configured on your machine.



1. Buttons Not Responding:
   * Check the console for any JavaScript errors that might indicate a problem with the button's event handler.
   * Verify that the API endpoints (/api/getClaim and /api/updateClaim) are reachable by testing them directly in the browser or using a tool like Postman.
2. Claim Not Fetching or Updating:
   * Make sure the CSV file (claims.csv) is located in the correct directory and is not open in another program that may lock the file.
   * Ensure the CSV file has the correct format with all required columns (ClaimID, ClaimStatus, ClaimOutcome, etc.).
   * Restart the development server to clear any temporary issues related to file reading or writing.
3. Data Not Displaying Correctly in Form:
   * Check if the state variables (e.g., claimDetails, claimOutcome) are being correctly set by logging their values.
   * Confirm that the API responses contain the expected data format, and adjust the front-end code accordingly.

Claim.csv data description

| Column Name | Data Type | Description |
| --- | --- | --- |
| ClaimID | String/Integer | A unique identifier for each insurance claim. |
| timeAsCustomer | Integer | The number of years the claimant has been a customer. |
| driverAge | Integer | The age of the driver involved in the claim. |
| insuranceAccess | Integer | The monetary amount or level of access to the insurance coverage. |
| insurancePremium | Integer | The annual cost of the insurance policy. |
| driverGender | String | The gender of the driver, typically "M"/"F" or "0"/"1". |
| educationLevel | String | The highest level of education attained by the claimant (e.g., "High School", "Bachelor"). |
| accidentType | String | The type of accident or incident (e.g., "Collision", "Theft"). |
| incidentSeverity | String | The severity of the incident (e.g., "Minor", "Major", "Severe"). |
| authoritiesInvolved | String | Indicates if authorities were involved (e.g., "Yes" or "No"). |
| incidentTime | String | The time of the incident (e.g., "14:30" for 2:30 PM). |
| numVehiclesInvolved | Integer | The number of vehicles involved in the incident. |
| numBodilyInjuries | Integer | The number of bodily injuries reported. |
| policeReportBool | Boolean/String | Indicates if a police report was available (e.g., "Yes"/"No" or true/false). |
| totalClaimAmount | Integer | The total amount claimed in the insurance request. |
| vehicleAge | Integer | The age of the vehicle in years at the time of the incident. |
| driverExperience | Integer | The number of years the driver has had a valid license. |
| licenseType | String | The type of driver's license (e.g., "Full", "Provisional", "Learner"). |
| firstName | String | The first name of the person submitting the claim. |
| lastName | String | The last name of the person submitting the claim. |
| claimStatus | String | The current status of the claim ("open" or "closed"). |
| claimOutcome | String | The outcome of the claim ("pending", "approved", "denied", or "escalated"). |
| claimDescription | String | A description provided by the claimant, explaining the details of the incident. |