**Scripts / Model Execution**

[**1. How is our solution delivered 1**](#_z69jq6f8h50k)

[**2. How to run our code using Github 1**](#_e0f9joahez4k)

[**3. How to run our code on Azure 3**](#_xg3ghqr1k8ru)

[**4. Script description: 3**](#_h68qeg5bqirz)

[**5. Troubleshooting/ Where to get help ? 4**](#_lzkwx1hwt23d)

[5.1. General Setup Issues 4](#_2wuxuzomdp9y)

[5.2. Local Code Execution Issues 5](#_ups520hswc8u)

[5.3. Azure Deployment or Execution Issues 5](#_px02angu82ke)

[5.4. Data-Related Issues 6](#_xj2vxkaeg7xc)

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

[**6. End-Users 6**](#_2m0dnztto7bj)

[6.1. Business Analysts and Data Scientists 7](#_qcidi39aizj7)

[6.2. Data Engineers and IT Operations Teams 8](#_1py629phq3ha)

[6.3. Business Stakeholders and Decision-Makers 8](#_ebjrnser5cv8)

[6.4. Product Managers / Project Managers 9](#_of9mrvxkbqjx)

[6.5. Compliance and Audit Teams 9](#_v3r22p950sw2)

[6.6. External Clients or Partners (Optional) 10](#_e0yr6s6wuayt)

[**7. Website 10**](#_fsj1l118itlm)

[7.1. How the website works (back end & front end) 10](#_aqha6s82q4j3)

[7.2. How to Use the Website 11](#_hwl446gj5dr2)

[7.3. Functionality 13](#_v3li46ruq7nz)

[7.4. Troubleshooting (In Case the Website Doesn't Function) 14](#_d94qs7mxjfvo)

# **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.
* 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 masters 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 to 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.

## **2.1. 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. Thecolumns 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.

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

## 4.1. General Setup Issues

* **Erorr in Github cloning:** 
  + **Issue:** User has difficulties in cloning the Github repository.
  + **Solution:** Ensure the Github is installed on your machine. Utilise the command *‘git–version’* to verify the installation. Additionally, verify you have stable internet access and correct permissions for the repository,
* **Python or Module Not Found Error**:
  + **Issue**: User receives an error relating to the absence of Python packages such as *pandas* or *numpy*
  + **Solution**: Verify that correct Python is installed and can be accessed using the command line *python–version* to confirm. Follow-up steps include:
    - Instal missing packages with *pip install -r requirements.txt*
    - Handle permission issues with *pip install --user -r requirements.txt*

## 4.2. Local Code Execution Issues

* **Master Script *(run\_all.sh or run\_all.ps1)* Errors**:
  + **Issue**: User has issues as scripts are terminating early or not running correctly.
  + **Solution**:
    - Ensure you have the correct permissions to run shell or PowerShel scripts. Use *chmod +x run\_all.sh* on Unix.
    - Confirm that the required python modules are installed.
    - Examine for any typos in file paths and ensure that the paths match your operating system’s format. For example forward or backward slashes.
* **Missing Permissions for PowerShell Scripts on Windows**:
  + **Issue**: By default, Powershell scripts are disabled.
  + **Solution**: Enable script execution with the following command:
    - Powershell Code: *Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser*
* **Parallel Execution Errors**:
  + **Issue**: During parallel execution some datasets arent processed correctly.
  + Some datasets aren't processed correctly during parallel execution.
  + **Solution**: Ensure that all input files are properly formatted and located in the correct directories. If needed, execute each stage separately to pinpoint the source of the error.

## 

## 4.3. Azure Deployment or Execution Issues

* **Azure Function Not Triggering**:
  + **Issue**: user is unable to execute Azure Functions after a pull request or POST request.
  + **Solution**:
    - Inspect the logs on Azure Functions to check for any errors that occured when using the Azure portal
    - Verify that authorisation key is accurately incorporated into the post request
    - Ensure that the private end-point is correctly configured and the function app service is functioning properly.
* **GitHub Actions Not Triggering**:
  + **Issue**: User’s pull request does not initiate the GitHub Action workflow
  + **Solution**:
    - Verify that the workflow file can be found in ‘*.github/workflows/’* and follows correct syntax.
    - Esure that the repository settings that authorise actions and triggers are properly configured.

## 

## 4.4. Data-Related Issues

* **Data Not Found or Incorrect Paths**:
  + **Issue**: Scrips fail to locate the processed or raw datasets.
  + **Solution**: Make sure the data is located in the correct directory, and that the paths align with those specified in the scripts. Whenever possible, use relative paths to prevent OS-related issues.
* **Unexpected Values in Processed Data**:
  + **Issue**: The synthesised or enriched data contains unexpected values.
  + **Solution**: Review the data cleaning and enrichment logic. Verify that missing values are appropriately addressed and categorical columns are consistently converted.

## 4.5. Where to Get Help?

* **Documentation**:
  + Refer to the README file and documentation in the GitHub repository for comprehensive instructions and usage guidelines.
  + This documentation will ensure that you have some examples and troubleshooting steps for common issues.
* **GitHub Issues**:
  + If the problem continues, open an issue in the GitHub repository. Include detailed information, such as error messages and screenshots, if relevant.
* **Azure Logs**:
  + When issues relating to runtime mistakes and misconfigurations are found utilise the Azure Portal to access logs for Azure Functions.
* **Stack Overflow / Online Forums**:
  + When general programming issues or errors that are package-related arise, look on the Stack Overflow or seek information from the Python package documentation

# **5. Website**

## **5.1. 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.

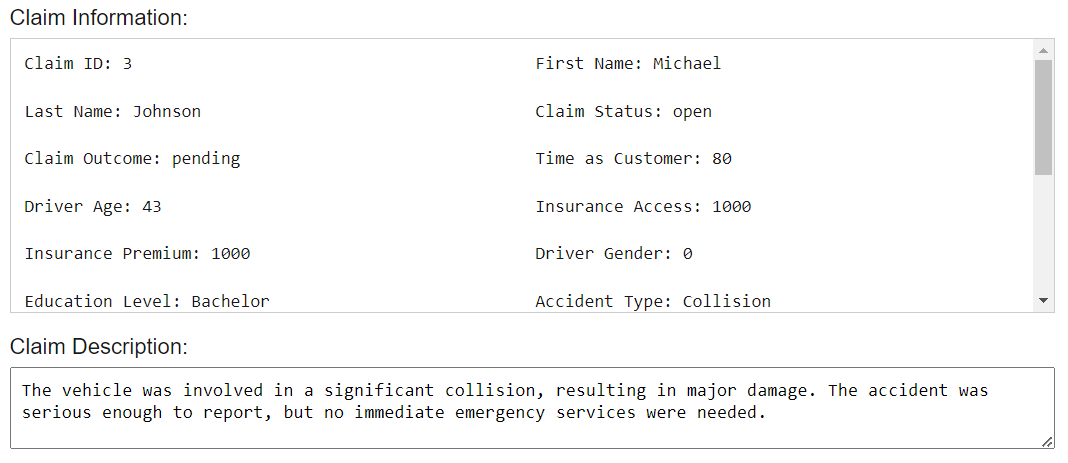
## 

## **5.2. 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.



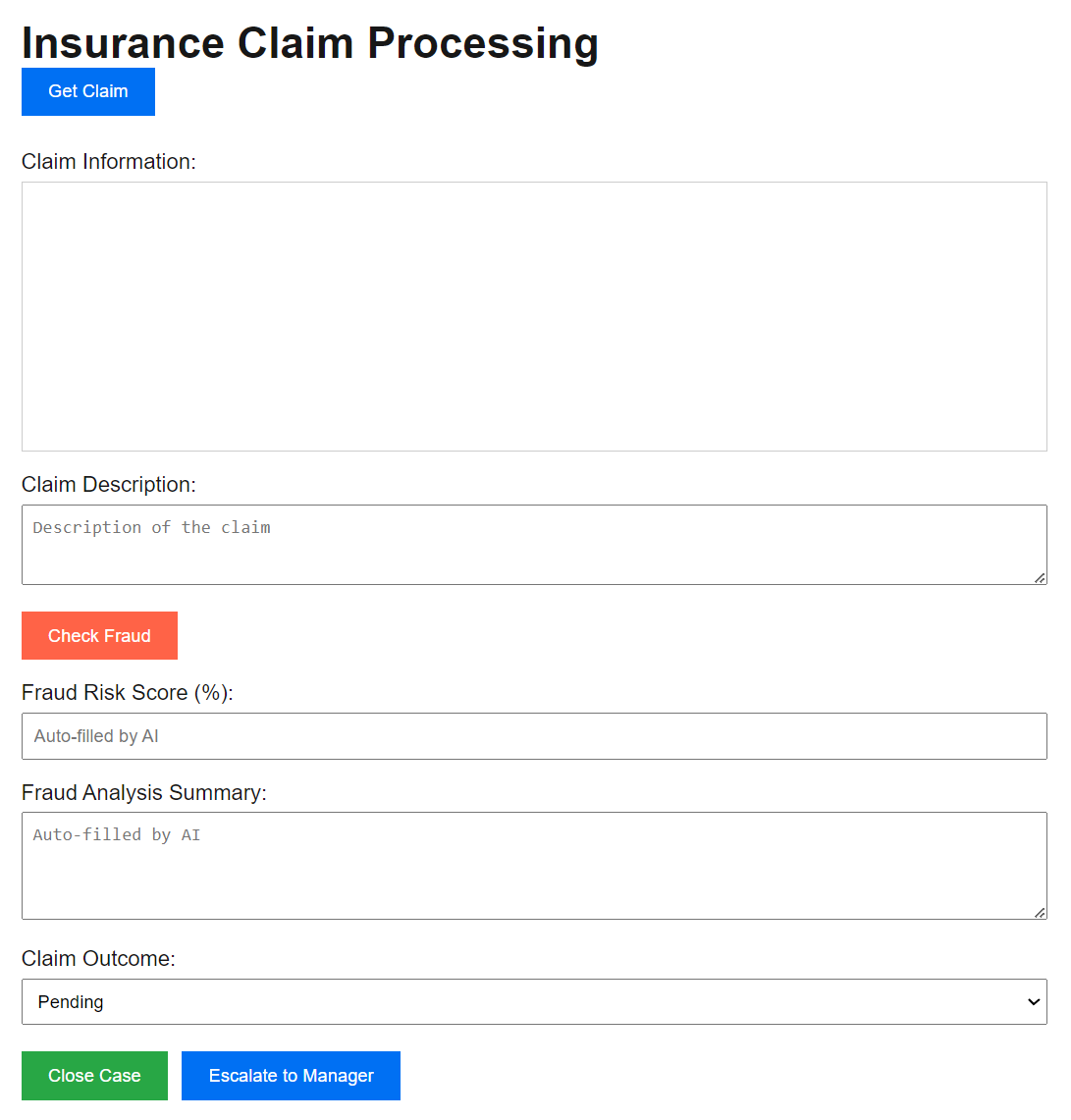
1. Viewing Claim Information:
   * The fetched claim information will be displayed in a formatted text area, including details such as Claim ID, customer information, and claim description.
   * The Claim Outcome dropdown will be populated based on the fetched claim.



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.

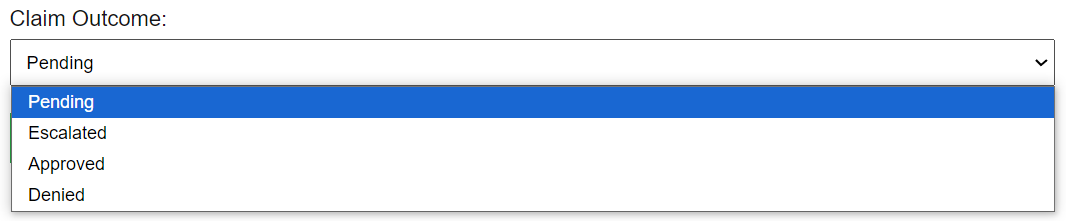


1. Updating the Claim:
   * Choose an option from the Claim Outcome dropdown (Pending, Approved, Denied, or Escalated).
   * Click on "Close Case" to mark the claim as resolved, or "Escalate to Manager" to flag the claim for further review.
2. Clearing the Form:
   * After updating the claim (closing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



## **5.3. 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 (Not Functional Yet): 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.





## **5.4. 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. |

# **6. End-Users**

**User Manual Overview**

| **End User** | **Key Responsibilities** | **Primary Tools** | **Where to Get Help** |
| --- | --- | --- | --- |
| **Claims Agent** | * Analyse pending claims and decide on claim status * Ensure data accuracy and consistency | * Fraud detection website * Claim management page | * User manual (5.1.) * System administrators (5.4.) * Website Manual () |
| **Supervisor/Manager** | * Review escalated claims * Monitor trends in claims for fraud patterns * Ensure proper outcomes are decided for claims | * Fraud detection website * Escalated claims page | * User manual () * System Administrators () * Website Manual () |
| **Business Analyst** | * Interpret insights for Power BI and fraud reports * Using data advise NRMA on strategic decisons | * Power BI dashboard * Fraud detection website | * User manual () * Data engineering team () * System Administrators () |
| **System Administrators** | * Monitor the technical infrastructure * Oversee APIs and maintain the functionality of the system * Search for and issues and errors with the system | * node.js * API management tools * Log monitoring systems | * User manual () * Data engineering team () * Website Manual () |
| **Data Engineers** | * Monitor and improve the data pipeline * Troubleshoot error to ensure the integrity of the data | * GitHub * Azure | * User manual () * GitHub Documentation () * Troubleshooting () |
| **AI Engineers** | * Enhance and maintain the AI model * Update model * Monitor performance metrics of the model | * CI/CD tools * API endpoints | * User manual () * Data engineers () * Troubleshooting () |

## 

## **6.1. Claims Agent**

**Purpose**:

The claims agent central role is to analyse the received claims and determine whether they should be approve, deny or escalate for further review by a supervisor.

**How to Use the Website:**

1. **Accessing the System:**
   * Open the website by clicking on the provided URL or it can be opened locally using (http://localhost:3000).
2. **Fetching and Reviewing a Claim:**
   * Click "Get Claim" to retrieve the next pending claim from the database
   * Review details of the claim such as, claim ID, claim amount, type of accident, etc. match the accident desciption.
3. **Managing Claim Status:**
   * Click "Check Fraud" to prompt the AI to analyse the claim
   * Examine the given ‘Fraud Risk Score’ and ‘Fraud Analysis Summary’
   * Select the appropriate outcome from the ‘Claim Outcome’ dropdown which includes, Pending, Approved, Denied, and Escalated.
   * Click “Escalate to Manager” is the claim requires further review and notify the respective manager
   * Click "Close Case" to conclude the claim analysis and progress onto the next claim.
4. **Clearing the Form:**
   * After the claim has been completed, the form with reset to the default empty form and once "Get Claim" is clicked a new pending claim will be received.
5. **Data Entry Guidelines:**
   * Review updates to the data are consistent and accurate to prevent issues that arise with inconsistencies in data.

## **6.2. Supervisors/Managers**

**Purpose:**

The main role of the Supervisor is to manage escalated claims and ensure they are resolved appropriately.

**How to Use the System:**

1. **Accessing Escalated Claims:**
   * Log in to the system and retrieve escalated claims by clicking the “Get Claim” button.
2. **Reviewing and Updating Claims:**
   * Examine the claim details and decide on the appropriate resolution. Choose an option from the Claim Outcome dropdown, then click “Close Case” to finalise the review.
3. **Monitoring Claim Trends:**
   * Monitor claim statuses and identify any suspicious patterns that may need further investigation.

## **6.3. Business Analyst**

**Purpose:**

The main role of the Business Analyst is to interpret high-level insights from the fraud detection system to advise the company on strategic decisions with the support these insights.

**How to Access and Use the Data**

1. **Where to Find the Data:**
   * The website will have a page for reports on insights made by the fraud detection model thus, the navigation bar can be used to find the page called ‘Fraud Insights’
   * Clicking on this page will lead to a Power BI dashboard that can be used to find accessible and digestible insights.
2. **How to Use the Data for Decision-Making:**
   * Review the dashboard and reports for key indicators, trends, and summary metrics.
3. **Requesting Additional Data:**
   * If there is a need for more specific reports or information they can create new Power BI dashboards or raise a ticket with the Data Engineering Team to create a more precise dataset.

## **6.4. System Administrators**

**Purpose:**

After the project is handed over to NRMA the System Administrators will oversee the technical infrastructure for the fraud detection website, APIs, and data files.

**How to Maintain the System:**

1. **Setting Up the Website:**
   * Verify that Node.js is installed and the server is running by executing *npm run dev*
   * Ensure the CSV file (Claims.csv) is correctly located in the directory.
2. **Managing APIs:**
   * Test the/api/getClaim and /api/updateClaim endpoints to confirm smoot data transfer between the website and backed.
3. **Troubleshooting Issues:**
   * Monitor system logs for errors and take corrective action to address server downtime or connectivity problems.
   * Check performissions to guarantee that the website can read from and write to the CSV file.
4. **Data Backup and Security:**
   * Ensure that regular backups are scheduled of claims.csv and verify that it is compliant with data security policies.

## **6.5. Data Engineers**

**Purpose:**

Once we hand this project over to NRM the data engineers will have to ensure they maintain, troubleshoot, and enhance the data pipeline while overseeing the infrastructure.

**Pipeline Execution Guide:**

1. **Pre-requisites:**
   * Ensure you have access to the Github repository and Azure portal.
   * Install the necessary packages using:
   * *pip install -r requirements.txt*
2. **How to Run the Code Locally:**
   * Clone the repository:
   * *git clone* [*https://github.com/AlanDataPortfolio/ey-azure-fn-pipeline.git*](https://github.com/AlanDataPortfolio/ey-azure-fn-pipeline.git)
   * Execute the **master script** to run the entire pipeline:

*bash run\_all.sh # Unix*

*pwsh run\_all.ps1 # Windows*

1. **How to Monitor the Azure Functions:**
   * Log in to the **Azure portal**.
   * Go to the Function App and check the monitor tab for logs and errors.
   * Ensure that all the appropriate authorisation keys are set for triggering the function.
2. **Handling Failures:**
   * If any script encounters an error, run an individual stages manually to troubleshoot:
   * *python src/clean\_dataset1.py*
   * Check the Azure activity log and Github Actions logs for any comprehensive error messages.

## **6.6. AI Engineers**

**Purpose**: After the project is handed over to NRMA, the role of AI engineers is crucial in maintaining, troubleshooting and enhancing the machine learning models integrated into the fraud detection system, They ensure the models continue to operate effectively , adapt to changing data patterns and provide valuable insights to claim analysis.

**Model management Guide:**

1. **Pre-requisites:**
   * Ensure you have access to the Github repository and Azure portal.
   * Install the necessary packages using:
   * *pip install -r requirements.txt*
2. **How to Run the Code Locally:**
   * Clone the repository:
   * *git clone* [*https://github.com/AlanDataPortfolio/ey-azure-fn-pipeline.git*](https://github.com/AlanDataPortfolio/ey-azure-fn-pipeline.git)
   * Execute the **master script** to run the entire pipeline:

*bash run\_all.sh # Unix*

*pwsh run\_all.ps1 # Windows*

1. **Monitoring model performance**
   * Log into the Azure portal.
   * Navigate to the relevant Azure Machine Learning Service.
   * Check performance metrics and logs for anomalies or degradation in model accuracy
2. **Handling model failures:**
   * If a model fails to execute, run individual components to diagnose the issue:
   * *Python src/run\_model.py*
   * Review azure activity logs and Github action logs for detailed error messages.
3. **Model Retraining:**
   * Locate fresh datasets in the Azure Data Lake storage account
   * Ensure data is in expected format for retraining
   * Schedule retraining by setting up a schedule using Azure Data Factory or a cron job.
   * Execute the retraining script:
   * *python src/retrain\_model.py*
   * Validaate the updated model against a holdout dataset
   * Use the following command to deploy the updated model:
   * *az ml model deploy --name <model\_name> --model <model\_path>*

# **7. Frequently Asked Questions (FAQs)**

1. **How can we deploy the fraud detection system if new data sources are added in the future?**

In the documentation we have provided instructions for deployment which you can find above. To accommodate the addition of new data the ingestion pipeline can altered by updating scripts or functions in Azure Data Factory or Functions. Additionally, you can refer to the AI Team’s user manual for the AI model for retraining steps due to the addition of new datasets

1. **What do we do if the model starts showing lower accuracy over time?**

If you observe a decrease in performance, it could signal concept drift, which refers to shifts in fraud patterns. Refer to the training manual for guidelines on retraining the model with the latest data. Additionally, you can utilise the automatic retraining process to address this issue.

1. **How do we troubleshoot if the data pipeline fails during execution?**

Azure will generate comprehensive error logs. We suggest reviewing the logs in Azure functions or Data factory for specific error codes. For guidance on restarting the pipeline and trouble shooting common issues, please consult the user manual. If necessary, reach out to our team during the designated support period.

1. **Can the fraud detection system be integrated with our existing CRM or claim processing system?**

Absolutely, the fraud detection model can interface with other systems through API endpoints included in the deployment. Additionally, your team should coordinate with the CRM team to ensure proper alignment of the data flow.

1. **What steps should we follow if Azure services experience downtime?**

If the Azure services experience downtime then make sure the Data Engineering team is monitoring the Azure tools that provide alerts. Next, the user should follow the Azure recovery plan in the documentation provided. Furthermore, remember to set-up a back-up storage system to prevent the event of data loss if this instance occurs.

1. **How do we update libraries or dependencies for automation scripts?**

All provided scripts come with a requirements.txt file listing the essential libraries. We recommend using a virtual environment or containerization (Such as Docker) to handle dependencies effectively.

1. **How frequently should the AI model be retrained?**

We would suggest retraining the model every 3 to 6 months, based on the influx of new claims and evolving fraud trends. Additionally, regular performance assessments such as accuracy and recall should be performed to evaluate whether retraining is needed sooner.

1. **What support will be available post-handover?**

After the handover we will be providing minimal support for any issues and troubleshooting as we expect that the team over at NRMA or a contracted third party will provide support.

1. **How do we ensure data security and privacy when using this system?**

All data utilised in this system adheres to NRMA’s security protocols and Azures data protection policies. Role-based access control (RBAC) has been implemented to manage access. It is advisable to regularly review roles and monitor data access logs to ensure compliance.

1. **What tools are available to interpret and visualise the model’s outputs?**

For the handover period no visualisation model will have been built due to constrictions with the deadline. Thus, will the agreement with NRMA it has been decided that they will continue the development of the fraud detection system to include the existence of visualisation

1. **What if fraudulent claims are missed by the model?**

Although the model is engineered to reduce false negatives, no system can achieve perfection. We suggest conducting regular manual reviews of both flagged and non flagged claims to uncover any gaps. Enhancements to the model can be achieved by retraining ot with additional fraudulent data as more cases are discovered.

1. **How is data security ensured on this website?**

To ensure the security of the website as it contain private information that should only be view by authorised individuals we have include a authentication system and access controls that prevent unauthorised individuals from accessing sensitive information