Loan Data Anomaly Detection

A project created for the technical discussions between Nimesh Acharya(author) and Exaloan. The project aims to get the anomalities in the provided loan data so as to ease the task of analyst to certain extent.

Technical Details

- File structure:

```
Exaloan
│   .env
│    requirements.txt
└──── Project
    │   detect_anomaly.py
    │   detect_anomaly_vector.py
    │   main.py
    │   naive_approach.ipynb
    │   session_state.py
    │   utils.py
    └──uploaded_files
        test.xlsx
```

The file structure of the project is as above. `main.py` is the main file to run the script, once running UI will automatically opened on browser. The file `detect_anomaly.py` has the functions that uses naive approach(iterating through every rows), whereas file `detect_anomaly_vector.py` has the functions that uses masking approach(iterating through only the masked data). File `naive_approach.ipynb` is a junk-file where my approach while creating this projects were tried. `session_state.py` just has one function that initializes the session and `utils.py` has the functions to help on UI, with motive of keeping `main.py` clean and modular. The frontend was created using streamlit, and python was used as main language for performing analysis.

- Install requirements.txt

```
pip install -r requirements.txt
```

- Run streamlit

```
streamlit run main.py
```

Now you can upload the files, which is internally saved inside `uploaded_files` folder.

User flow:

The user has a file in excel format which one can pass to script and script should automatically return the data with respective LoanID and their status along with why they were the reason of being flagged. User has the option to analyse whether using normal(naive) approach or vectorized approach. For the ease of the user experience, the UI is created for handling all the processes, which also shows time consumed by Vector and Normal method.

Product Details

Goal of the task and their solutions

- ### Data Ingestion & Parsing:

  Datasets were read using pandas and kept as dataframe on local session once updated. For ensuring the missing columns try-catch block was added when performing the main calculations so as to catch any KeyError. The error would be notified to user via toast notification. All the date formats were read using pandas to_datetime to make consistent format across datasets.

- ### Anomaly Detection:

  Different types of anomalities were tried to catch as follows:

  - **Detecting payment ability anomalities**: The main idea of writing this function was whether the borrower has the ability to pay back monthly installments or not.

  - **Detecting re-payment anomalities**: The idea of writing this function was to detect how earlier or later the actual repayment was made than that of expected repayment date. In case of earlier, the lender may suffer loss, as enough interest is not gained. In case of later, the lender may have less or no profit at all.

  - **Detecting payment history anomalities**: Reason to write this function is to know how consistent was the borrower in past in terms of paying in time. Two factors are considered in this function the first is `Days late` column where just the threshold of 90 days is checked. The second is going through `payments` to analyse payments history and counting if most of the payment state were either "pending late" or "paid with delay" and flagging them accordingly.

  - **Data validation anomalies**: Reason to write this function is to check if the data are valid in terms of real-life condition or not, for example: interest rate ranging between 0-100, loan term as positive integer, credit score in certain range, etc. Further the JSON data was also checked and flagged if required, in case they weren't on valid format (for payment history column)

  All the anomalies were detected and then kep in dict along with the reason of detection.

• Design for Scale and Robustness:

Initially the naive/normal approach was used to solve the problem, but later on new file was created detect_anomaly_vector.py where the idea was to reduce the cost. The way I thought of doing this solution was masking the condition, so that the iterator doesn't have to pass through those data that were not masked(non-anomaly data),i.e. less iteration ofcourse reduces cost. **Note**: In case of payment history, one still needs to iterate through every row as it is in JSON format, and cannot be easily masked. Though there is difference in computation time between normal and vector method I feel this can further be made in a way that saves cost.(Future possibilities [v])

For handling the unreadable lines or data errors, as mentioned earlier the try catch block wraps the main operation which shows the error in console, and also in UI as a toast notification.

• Reporting and Outputs:

The saved dictionary of section anomaly detection was now converted to datafram and shown in UI. Further, the table can be downloaded from the UI in CSV format.

• Use of an LLM:

Not performed in this project. As I felt there are a lot of things to play with data before getting to this step.

Future Possibilities

i. The first thing that one can do is calculate the Interest rate anomality. By looking at the trend of actual repayment date vs expected repayment date. So, the lender will not suffer from early repayment or late repayment.

ii. In terms of individual borrower, using all the data related to family, spouse, DTI, last debt payment, etc. to evaluate risk factor.

iii. In terms of business borrower, using the information about business, collateral, appraisal,etc to evaluate risk factor.

iv. Leveraging the use of LLM maybe to parse the company description, and know more about it. In case of real data we can maybe automate passing about the individual/business to search engine/LLM and get insights of public on those factors.

v. Managing the payment history, so that the cost can be saved, and JSON parsing error can be mnimized. Rather than storing on JSON format creating different table for it and connecting with main data using LoanID/Borrower ID.

vi. Using Machine Learning to train the old loan cases, and creating a model so that the new analysis can be done.

vii. Using statistical methods, maybe calculating how deviated are the data from mean of the data.

viii. Maybe using rule based methods as did in this project but also for other variables like creditscore, transaction frequency, etc.