FRAUDULENT TRANSACTION DATA ANALYSIS

IMPLEMENTATION REPORT

YEDHU PRASAD

Contents

INTRODUCTION

Electronic transactions have become increasingly prevalent today, making life more convenient for people worldwide. However, as the use of online transactions grows, so does the possibility of fraudulent activity. Fraudulent operations can result in substantial financial losses, particularly in the banking industry, which mainly relies on electronic transactions.

In this project, I have been tasked with developing a fraud detection system for a bank that issues credit cards. The system must be accurate, efficient, and effective in detecting fraudulent behaviours to activate appropriate mechanisms for protecting the bank's clients. To achieve this goal, I will pre-process the data, and create new features. The ultimate objective is to develop a system that can distinguish between fraudulent and non-fraudulent activities, protect the bank's clients from financial losses, and ensure the smooth running of the bank's operations.

PROBLEM ANALYSIS

The task at hand is to create a fraud detection system for a credit card issuing bank. Using the bank's customers' purchasing transaction data, the system should be able to detect fraudulent transactions in real-time. A dataset module, a distance module, and a statistics module will be required for the solution. A main module will also be required to provide a user interface for querying and interacting with the other modules' operations.

The dataset module will retrieve user transactions from specified datasets and provide a dictionary including user and transaction data. This module will do this task by utilising Python file objects.

The distance module will include two functions: one for determining the distance between any two given user transactions, and another for computing the distance between any two given user transactions.

The statistics module will include 12 functions for calculating fundamental statistics on user transactions. The average, mode, and median of transactions, as well as the interquartile range, location centroid, and standard deviation, will be computed. The module will also include features for detecting fraudulent transactions, recognising odd transactions, calculating Z scores, calculating transaction frequencies at specified locations, and finding outliers and transaction percentiles.

The main module, also known as the test module, will serve as a user interface for querying and interacting with the functions in the other modules. The main module will provide users with access to the features in the distance and statistics modules.

Throughout the development process, I will be ensuring that my code is well-documented and organized, utilizing appropriate programming concepts and principles.

Dataset

The dataset contains a total of 1100 transactions made by 10 users. Each transaction is represented by a row of data. The dataset includes seven columns, which contain information about each transaction. The first column displays the user ID, the second column the transaction ID, and the third column the transaction description. The fourth column displays the transaction's value, while the fifth and sixth columns give the x and y coordinates of each transaction, respectively. Finally, the seventh column is a Boolean label that indicates whether the transaction is fraudulent or not, where 'false' means the transaction is not fraudulent and 'true' means the transaction is fraudulent.

Also, we have been provided with two other files along with the dataset. The description.txt, which contains all the transaction descriptions as well as the fraud-description.txt which contains the description of the fraudulent transaction.

SOLUTION REQUIREMENTS

- Implementation of Dataset module
    - Develop a dataset module that retrieves data from the provided dataset using Python file objects.
    - The retrieved data should be in the form of a nested dictionary containing user transactions.
- Implementation of Distance module
    - Implement a function that calculates the distance between any two transactions of a user.
    - Implement a function that calculates the distance between transactions of any two users.
- Implementation of Statistics module
    - Develop 12 functions to compute basic statistics on transactions of any user or of all users.
    - Functions should compute :
        - Average transactions
        - Mode
        - Median
        - Interquartile range
        - Location centroid
        - Standard deviation
        - Fraud detection
        - Abnormal transactions
        - Z score
        - Transaction frequencies
        - Outliers
        - nth Percentiles.
- Implementation of Main/test module:
    - Implement the user interface function that enables users to query and interact with all functions in the distance and statistics modules.

IMPLEMENTATION OF SOLUTION

For the implementation of the dataset module, there is a function 'dataset()' that retrieves a nested dictionary containing user transactions, including the location and amount of money spent in each transaction. The module uses Python file objects to read and process the data. The function handles and addresses all possible errors and exceptions.

In the second module, there are two functions, SameUserDistance() and TwoUserDistance(). The SameUserDistance() function takes three inputs: id, an integer value representing the user ID; tran1, tran1: integer value representing the first transaction ID; and tran2, an integer value representing the second transaction ID. and returns the Euclidean distance between two transactions for a given user. If the input IDs are valid and exist in the dataset, the function returns the distance rounded to two decimal places. Otherwise, it raises a key error. The second function (TwoUserDistance()) calculates the Euclidean distance between two transactions for two given users. It accepts 4 inputs (id1, id2, tran1, tran2).

The statistics_module contains 12 functions to compute basic statistics on transactions. The functions use the data retrieved in the dataset_module to compute the statistics. The functions are:

- Mean()
  The Mean() takes an optional parameter id. If the id is not passed, it returns the average of all transactions for all users. If the id is passed, it returns the average of all transactions for that specific user.
- Mode()
  The Mode() takes an optional parameter id. If the id is not passed, it returns the mode of all transactions for all users. If the id is passed, it returns the mode of all transactions for that specific user.
- Median()
  The Median() takes an optional parameter id. If the id is not passed, it returns the median of all transactions for all users. If the id is passed, it returns the median of all transactions for that specific user.
- IQR()
  The IQR() also takes an optional parameter id. It creates a list of all transaction amounts for the given user or all users. It then sorts the list of transaction amounts and calculates the first quartile (Q1) and third quartile (Q3) by indexing the list at n/4 and 3n/4, respectively, where n is the length of the list. Finally, it calculates the IQR as the difference between Q3 and Q1 and returns this value.
- LocCentroid()
  The LocCentroid() takes id as a parameter and returns the location centroid of the user, based on their transaction locations. It calculates the centroid of the user's transaction locations using the formula: (sum of x coordinates) / (number of transactions) and (sum of y coordinates) / (number of transactions).
- SD()
  This function computes the standard deviation of transactions for a specific user by first getting the transaction amounts for the user from the dataset. It then calculates the mean of the transaction amounts and uses it to calculate the variance of the transaction amounts. It returns the square root of the variance, which is the standard deviation. The function takes one argument, which is the ID of the user whose transactions will be used to calculate the standard deviation. It returns a float value that represents the standard deviation of the user's transactions.
- fraudOrNot()
  The fraudOrNot() takes a transaction ID as input and searches for that ID in the dataset. If the ID exists in the dataset, the function checks whether the transaction is fraudulent or not. If the transaction is fraudulent, the function returns a message indicating that the transaction is fraudulent and provides details of the transaction, including the user ID, transaction ID, and the rest of the transaction details. If the transaction is not fraudulent, the function simply returns a message indicating that the transaction is not fraudulent.
- abnormal()
  The abnormal()takes a user ID as input and returns the abnormal transactions that exceed the threshold. The function calculates the abnormal transaction threshold for the given user by adding three standard deviations to the median of their transaction amounts. It then iterates through each transaction of the user and checks if the transaction amount is greater than the threshold. If a transaction amount is greater than the threshold, it is marked as abnormal and its details are added to a string called abnormal_transition. Finally, the function returns the abnormal threshold and prints out the details of any abnormal transactions for the user.
- ZScore()
  This ZScore computes the Z score of transactions for either a specific user or all users in a dataset. If the ID parameter is not provided, the function calculates the Z score for all users in the dataset. It iterates through each transaction for each user, calculates the variance and stores these values in a list, and then calculates the standard deviation for the list. Next, it iterates

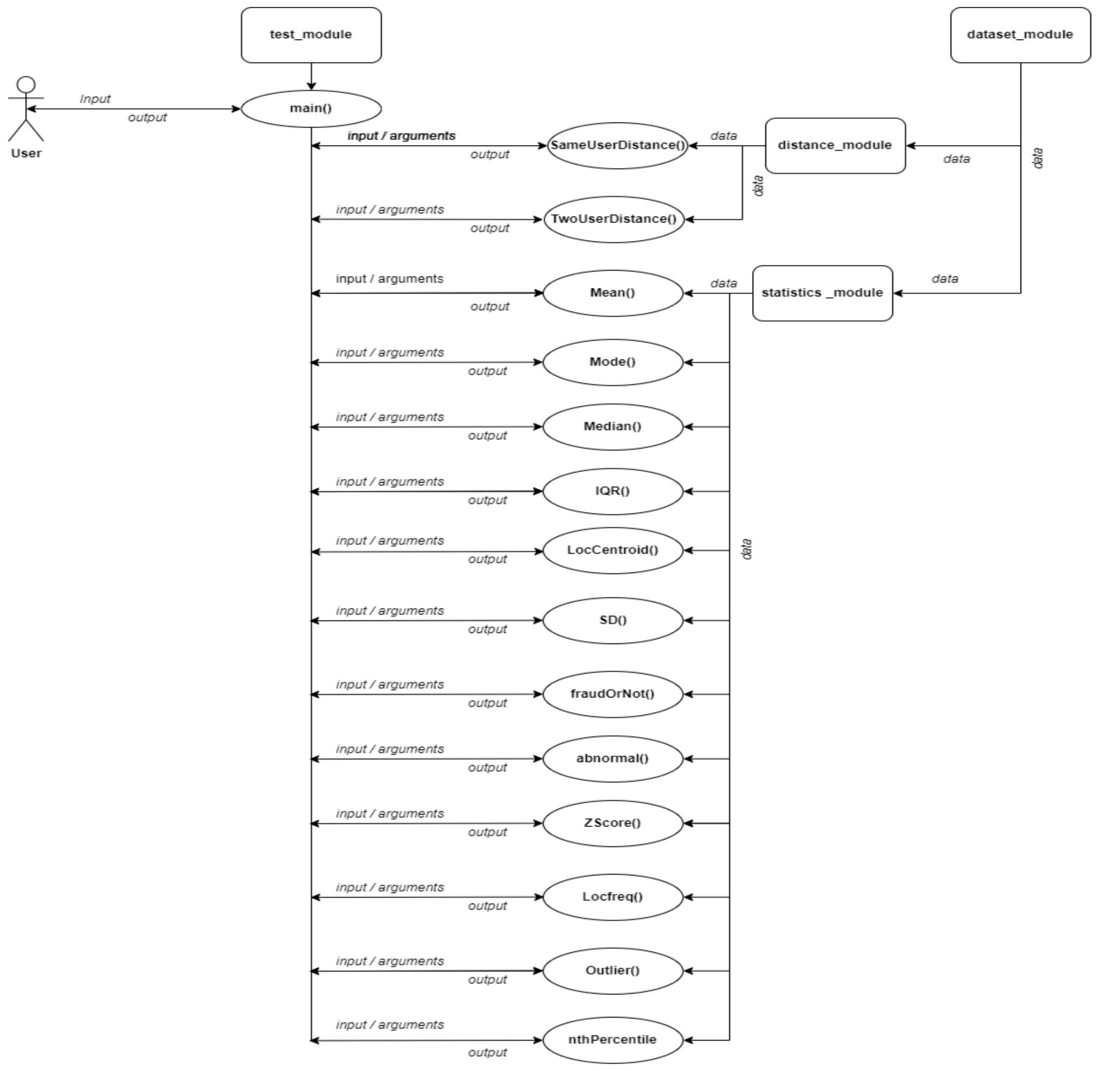
through each transaction for each user again, calculates the Z score for each transaction using the formula Z = variance / SD and stores these values in the Z string. If the 'id' parameter is provided, the function calculates the Z score only for transactions associated with that specific user. It retrieves the standard deviation for that user using the SD() function and then iterates through each transaction for that user to calculate the Z score for each transaction using the same formula as before.

- Locfreq()
  Locfreq() calculates the frequency of transactions at a given location (specified by the x and y coordinates). For each transaction, it checks if the transaction's x and y coordinates match the input coordinates. If they do, it increments the count variable. Finally, the function returns the count of transactions at the given location.
- Outlier()
  To calculate the outliers, it first gets the centroid of the user's location and assigns it to variables x1 and y1. Then, it calculates the distance between each transaction and the centroid and stores it in the list distance. After that, it calculates the mean distance and the standard deviation for the distances from the centroid and calculates the median of the distances. Using the median and the standard deviation, it calculates the outlier threshold using the equation median + 3*SD. It then checks if the distance between each transaction and the centroid is greater than the outlier threshold. If it is, it adds the transaction to the list of outlier locations.
- nthPercentile()
  The nthPercentile function calculates the nth percentile of all transactions in the dataset or for a specific user. It takes a percentile value and an optional user ID as input and returns the nth percentile value and its index in the sorted list of all transactions. The function calculates the index of the value that corresponds to the nth percentile using the formula: (p/100) * n, where n is the total number of transactions. If the index is an integer, the function takes the average of the index and index-1th values to calculate the nth percentile. If the index is not an integer, the function rounds up to the nearest integer and returns the corresponding value as the nth percentile.

PROGRAM EXECUTION

The programme consists of four modules: dataset_module for data retrieval, distance_module to calculate the Euclidean distance between two transactions, statistics_module for data analysis with 12 statistical functions, and main/test_module as the user interface. The first three modules are imported into the test_modules. It has a primary function with a list of alternatives that is driven by a menu, and when a user chooses an option, that function is called by their selection.

SYSTEM DIAGRAM



REFLECTION

The goal of this project is to create a system for the bank to identify fraud. To activate the proper safeguards for the bank's customers' protection. The ultimate goal is to distinguish between fraudulent and lawful activity, safeguard clients from financial harm, and ensure the efficient operation of the bank. The main tasks are the creation of the modules. The first is for the data retravel and the second module is for the calculation of the distance between users' transactions. For me the first two modules were easy-going and the only thing that challenged me was the data retrieval into a nested dictionary. The statistics module was so complicated for me at the first glance and the question was so confusing, after spending a whole lot of time understanding the questions after that I started coding the module. Creating each function manually took me a lot of time. The coding was fun I learned a new thing while searching for answers.

APPENDIX

Dataset module

```
FUNCTION dataset(data_path):
    user ← EMPTY DICTIONARY
    OPEN file f with data_path
    FOR each line in f DO
        columns ← SPLIT line by ':'
        user_id ← FIRST element of columns
        transaction_id ← SECOND element of columns
        description ← THIRD element of columns
        amount ← FOURTH element of columns
        x_coordinate ← FIFTH element of columns
        y_coordinate ← SIXTH element of columns
        fraud ← SEVENTH element of columns
        transaction ← DICTIONARY with description, x_coordinate, y_coordinate, amount, and fraud as
                        keys and their respective values as values
        IF user_id is in user THEN
            ADD transaction with transaction_id as key to user[user_id]
        ELSE
            CREATE a new dictionary with transaction_id as key and transaction as value and add it to user[user_id]
        ENDIF
    ENDFOR
    RETURN user
ENDFUNCTION
```

Distance Module

```
FUNCTION SameUserDistance(id, tran1, tran2):
    user ← dataset()
    x1 ← user[id][tran1]['x_coordinate']
    y1 ← user[id][tran1]['y_coordinate']
    x2 ← user[id][tran2]['x_coordinate']
    y2 ← user[id][tran2]['y_coordinate']
    distance ← square root of (((x2 - x1) ^ 2) + ((y2 - y1) ^ 2))
    RETURN distance rounded to 2 decimal places
FUNCTION TwoUserDistance(id1, id2, tran1, tran2):
    user ← dataset('/content/drive/MyDrive/Colab Notebooks/ASS_1/Transaction.txt')
    x1 ← user[id1][tran1]['x_coordinate']
    y1 ← user[id1][tran1]['y_coordinate']
    x2 ← user[id2][tran2]['x_coordinate']
    y2 ← user[id2][tran2]['y_coordinate']
    dis_Bw_2 ← square root of (((x2 - x1) ^ 2) + ((y2 - y1) ^ 2))
    RETURN dis_Bw_2 rounded to 2 decimal places
ENDFUNCTION
```

Statistics Module

Mean()

```
BEGIN function Mean(id=None)
  IF id is None THEN
      count ← 0
      amount ← 0
      FOR each i in user DO
          FOR each j in user[i] DO
              amount ← amount + user[i][j]['amount']
              count ← count + 1
         ENDFOR
      ENDFOR
      avg ← amount / count
  ELSE
      count ← 0
      amount ← 0
      FOR each i in user[id] DO
          amount ← amount + user[id][i]['amount']
```

```
        count ← count + 1
      ENDFOR
      avg ← amount / count
    ENDIF
    RETURN avg rounded to 2 decimal places
END function
```

Mode()

```
BEGIN function Mode(id=None)
    IF id is None THEN
      user ← dataset()
      mode_dic ← empty list
      FOR each i in user DO
        FOR each j in user[i] DO
          mode_dic.append(user[i][j]['amount'])
        ENDFOR
      ENDFOR
      mode ← the value that appears most frequently in mode_dic
    ELSE
      user ← dataset()
      IF id not in user THEN
        RETURN None
      ENDIF
      mode_dic ← empty list
      FOR each i in user[id] DO
        mode_dic.append(user[id][i]['amount'])
      ENDFOR
      mode ← the value that appears most frequently in mode_dic
    ENDIF
    RETURN mode
END function
```

Median()

```
BEGIN function Median(id=None)
    user ← dataset()
    IF id == None THEN
        median_dic ← empty list
        FOR each i in user DO
            FOR each j in user[i] DO
                append user[i][j]['amount'] to median_dic
            ENDFOR
        ENDFOR
        sort median_dic in ascending order
        n ← length of median_dic
        IF n % 2 == 0 THEN
            median ← (median_dic[n//2-1] + median_dic[n//2])/2
        ELSE
            median ← median_dic[n//2]
        ENDIF
    ELSE
        median_dic ← empty list
        FOR each i in user[id] DO
            append user[id][i]['amount'] to median_dic
        ENDFOR
        sort median_dic in ascending order
        n ← length of median_dic
        IF n == 0 THEN
            median ← 0
        ELSE IF n % 2 == 0 THEN
            median ← (median_dic[n//2-1] + median_dic[n//2])/2
        ELSE
            median ← median_dic[n//2]
        ENDIF
    ENDIF
    RETURN round(median,2)
END function
```

IQR()

```
BEGIN function IQR(id=None)
    user ← dataset()
    amounts ← empty list
    IF id is None THEN
        FOR each i in user DO
            FOR each j in user[i] DO
                append user[i][j]['amount'] to amounts
            ENDFOR
        ENDFOR
    ELSE
        FOR each i in user[id] DO
            append user[id][i]['amount'] to amounts
        ENDFOR
    ENDIF
    sort amounts in ascending order
    n ← length of amounts
    q1 ← amounts[int(n/4)]
    q3 ← amounts[int(3*n/4)]
    iqr ← q3 - q1
    RETURN iqr
END function
```

LocCentroid()

```
BEGIN function LocCentroid(id)
    user ← dataset()
    loc_x ← empty list
    loc_y ← empty list
    FOR each i in user[id] DO
        append user[id][i]['x_coordinate'] to loc_x
        append user[id][i]['y_coordinate'] to loc_y
    ENDFOR
    x_center ← sum of loc_x / length of loc_x
    y_center ← sum of loc_y / length of loc_y
    RETURN round(x_center, 2), round(y_center, 2)
END function
```

SD()

```
BEGIN function SD(id)
    user ← dataset()
    amount ← empty list
    FOR each i in user[id] DO
        append user[id][i]['amount'] to amount
    ENDFOR
    n ← length of amount
    amount_mean ← Mean(id)
    sd_list ← empty list
    FOR each i in user[id] DO
        sd_list.append(((user[id][i]['amount']) - amount_mean) ** 2)
    ENDFOR
    amt_SD ← square root of (sum(sd_list) / (n-1))
    RETURN round(amt_SD, 2)
END function
```

fraudOrNot()

```
BEGIN function fraudOrNot(tid)
    user ← dataset()
    fruad_str ← ""
    FOR each i in user DO
```

```
        IF tid is in user[i] THEN
            IF user[i][tid]['fraud'] is equal to 'true' THEN
                PRINT "Fraudulent"
                PRINT "Transaction ID:", tid
                PRINT "User ID:", i
                PRINT "Details: "
                PRINT str(user[i][tid])
                RETURN "Fraudulent"
            ELSE IF user[i][tid]['fraud'] is equal to 'false' THEN
                RETURN "Not Fraudulent"
            ELSE
                RETURN "Invalid input"
            ENDIF
        ENDIF
    ENDFOR
    RETURN "Transaction not found"
END function
```

abnormal()

```
BEGIN function abnormal(id)
    user ← dataset()
    abnormal_transition ← empty string
    abnormal_threshold ← Median(id) + 3 * SD(id)
    FOR each tid in user[id] DO
        IF user[id][tid]['amount'] > abnormal_threshold THEN
            append tid and transaction details to abnormal_transition
        ENDIF
    ENDFOR
    print abnormal_transition
    RETURN abnormal_threshold
END function
```

ZScore()

```
BEGIN function ZScore(id=None)
   user ← dataset()
   Z ← empty string
   sd_lst ← empty list
   IF id is None THEN
      FOR each i in user DO
         FOR each j in user[i] DO
            append ((user[i][j]['amount']) - Mean()) ** 2 to sd_lst
         ENDFOR
      ENDFOR
      n ← length of sd_lst
      stdv ← (sum(sd_lst) / (n-1)) ** 0.5

      FOR each i in user DO
         FOR each j in user[i] DO
            z_score ← round((((user[i][j]['amount']) - Mean()) / stdv), 3)
            concatenate str(j), ":\t", str(z_score), "\n" to Z
         ENDFOR
      ENDFOR
   ELSE
      sd ← SD(id)
      FOR each tid in user[id] DO
         z_score ← round((((user[id][tid]['amount']) - Mean(id)) / sd), 3)
         concatenate str(tid), ":\t", str(z_score), "\n" to Z
      ENDFOR
   ENDIF
   PRINT Z
END function
```

Locfreq()

```
BEGIN function Locfreq(x, y)
```

```
    user ← dataset()
    count ← 0
    FOR each i in user DO
        FOR each j in user[i] DO
            IF user[i][j]['x_coordinate'] is equal to x AND user[i][j]['y_coordinate'] is equal to y THEN
                count ← count + 1
            ENDIF
        ENDFOR
    ENDFOR
    RETURN count
END function
```

Outlier()

```
BEGIN function Outlier(id)
    user ← dataset()
    (x1, y1) ← LocCentroid(id)
    distance ← empty list
    FOR each tid in user[id] DO
        x2 ← user[id][tid]['x_coordinate']
        y2 ← user[id][tid]['y_coordinate']
        distance.append(sqrt((x2 - x1)**2 + (y2 - y1)**2))
    ENDFOR
    n ← length of distance
    mean_distance ← sum of distance / n
    sd_lst ← empty list
    FOR each distance[i] in distance DO
        sd_lst.append((distance[i] - mean_distance) ** 2)
    ENDFOR
    SD ← square root of (sum of sd_lst / (n - 1))
    distance.sort()
    IF n is even THEN
        median ← (distance[n//2 - 1] + distance[n//2]) / 2
    ELSE
        median ← distance[n//2]
    ENDIF
    outlier_threshold ← median + 3 * SD
    outlier_location ← empty string
    FOR each tid in user[id] DO
        x2_check ← user[id][tid]['x_coordinate']
        y2_check ← user[id][tid]['y_coordinate']
        distance_check ← sqrt((x2_check - x1)**2 + (y2_check - y1)**2)
        IF distance_check > outlier_threshold THEN
            append x2_check and y2_check to outlier_location
        ENDIF
    ENDFOR
    print outlier_location
    RETURN
END function
```

nthPercentile()

```
BEGIN function nthPercentile(p, id=None)
    user ← dataset()
    amount ← empty list
    IF id is not None AND id exists in user THEN
        FOR each transaction in user[id] DO
            append transaction amount to amount list
        ENDFOR
    ELSE
        FOR each user in user DO
            FOR each transaction in user[user] DO
                append transaction amount to amount list
            ENDFOR
        ENDFOR
    ENDIF
    sort amount list in ascending order
    n ← length of amount list
    n_idx ← p/100 * n
    IF n_idx is an integer THEN
        n_idx_low ← n_idx - 1
        n_idx_high ← n_idx
        nth ← (amount[n_idx_low] + amount[n_idx_high]) / 2
    ELSE
        nth ← amount[integer(n_idx)]
    ENDIF
    RETURN round(nth, 2), integer(n_idx)
END function
```