FRAUD DETECTION SYSTEM

Implementation Report

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

We shall begin this report by examining the problem that has to be defined and addressed. We will go through the two portions of the necessary programme and the various similarity algorithms. We will establish the programme needs based on this analysis. Following that, we will go through the implementation process, covering the programming ideas that were employed during development. We will also give instructions for running the programme correctly. A flowchart diagram will be produced to demonstrate the general flow of the programme from beginning to end. We have given pseudocode for the two modules in Appendix A because there is no clear breakdown of the logic of the modules in the flowchart. This pseudocode only illustrates sections of the module code that are not discussed in the flowchart and does not include all of the code from the modules. In the final section of the report, we will reflect on the process and examine which aspects went well and which did not. We will also make ideas for enhancements or alternative methods that may have been adopted.

CHAPTER 2 PROBLEM ANALYSIS

The challenge is to create a system that allows a major bank that provides credit cards to detect fraudulent behaviour in real time by analysing its clients' purchasing transaction data. The system should be able to detect fraudulent transactions rapidly and trigger relevant safeguards to protect the bank's clients. Based on the concepts and principles presented in this lesson, the system should be created.

The following are the primary challenges in this problem:

1. The massive amount of transaction data that must be analysed in real time.
2. The requirement to correctly discern between fraudulent and non-fraudulent transactions.
3. The requirement to provide suitable systems to protect the bank's clients in the event of fraudulent transactions.
4. The requirement to create a scalable system capable of handling a growing number of transaction data over time.

To address this issue, we must examine the existing data sets for patterns and features that distinguish fraudulent transactions from non-fraudulent ones. To safeguard the bank's clients, we must build and deploy a system that can process transaction data in real time, identify fraudulent activities, and take relevant responses. In addition, the system should be built to manage a growing number of transaction data over time. Finally, we must assess the system's performance and constantly enhance it to ensure its efficacy in identifying fraudulent behaviour.

FOR DISTANCE MODULE

Method	Min. Possible Result Value	Max. Possible Result Value	Result Information
Distance between any two transactions of same user	0	∞	If the result is larger, it means the distance is greater between the 2 transaction locations and vice versa.
Distance between any two transactions of different user	0	∞	If the result is larger, it means the distance is greater between the points and vice versa.

FOR STATISTICS MODULE

Method	Min. Possible Result Value	Max. Possible Result Value	Result Information
Average	0	∞	The larger the average means larger is the Transaction amount

Mode	0	∞	Larger the Mode represents more same transactions done repeatedly
Median	0	∞	Calculates the median value by sorting the data to give an insight into the transactions
Interquartile Range	0	∞	It measures the spread of the data and helps identify the outliers. Larger value represents outliers.
Location Centroid	Latitude=0 Longitude=0	Latitude=∞ Longitude=∞	Larger the location centroid represents the transaction locations are clustered together and vice versa
Standard Deviation	0	∞	Larger the standard deviation larger represents larger spread and vice versa
Is Fraudulent	False	True	True represents a transaction is fraudulent and False represents normal.
Abnormal Transaction	0	∞	Larger the abnormal represents transactions could be fraud.
Z score	-1	∞	The larger the z score are fraudulent transactions.
Transaction Frequency	0	∞	Larger frequency represents more transactions on same location.
Outliers	0	∞	These are the potential fraud transactions
Nth Percentile(0 - 100)	0	∞	Indicates the value below which a certain percentage of observations in a dataset fall.

CHAPTER 3 SOLUTION REQUIREMENTS

- **Implement a module that contains a function which retrieves the user transaction details**.
- **Create a distance module with 2 functions:**
  - A function which calculates the distance between any 2 transactions of same user.
  - A function which calculates the distance between any 2 transactions of different users.
- **Implement a statistic module that contains the following 12 functions:**
  - A function that returns the average transactions of any user and all users.
  - A function that returns the mode of transactions of any user and that of all users.
  - A function that returns the median of all transactions of a user and that of all users.
  - A function that returns the interquartile range of any user's transactions and of all users.
  - A function that returns the location centroid of any user.
  - A function that computes the standard deviation of any user's transactions.
  - A function that determines whether a transaction is fraudulent or not
  - A function that returns an abnormal transaction for any given user.
  - A function that computes the Z score of transactions of a user and all transactions.
  - A function that computes those frequencies of transactions at any given location.
  - A function that returns the outlier of any location and of any user.
  - A function that returns the nth percentiles of transactions of any user and of all users.
- **Implement a test module that has a user interface function which interacts with other modules.**

CHAPTER 4 IMPLEMENTATION OF SOLUTION

The aim of the project is to develop a fraud detection system that analyzes users' transaction data and identifies fraudulent transactions. Here we implement four modules to perform the action, dataset_module, distance module, statistics_module and test_module.

I. Dataset Module: The dataset_module is used for retrieving transaction attributes and features and then returns the dictionary with dataset. The module uses Python file functions and exception handling for the task. The dictionary contains the Transaction id, User id, Transaction location, Transaction amount and Transaction description.

II. Distance Module: The distance_module contains 2 functions, first one calculates the distance between the transaction locations of a single user while the second function calculates the distance between 2 transaction locations of 2 different users. Here the Euclidean distance formula is used distance = $\sqrt{((x2 - x1)^2 + (y2 - y1)^2)}$ .Here type error, key error and value error exception handling are used.

III. Statistics Module: The statistics module contains several functions used to find the estimates of the dataset 'transaction.txt'. Among these functions are: **Average Transaction**: This function calculates the average transaction amount for any and all users. It employs the formula: (total transaction amounts) **/** total number of transactions. **Transaction Mode**: This method returns the transaction mode for any and all users (most occurring amount). **Transaction Median**: This method returns the median of transactions for any and all users. It employs the median formula, which takes the average of the ordered transaction amounts. **Interquartile Range**: This method returns the interquartile range of any user's transactions as well as the interquartile range of all users. It employs the following formula: Q3 - Q1, where Q1 is the first quartile and Q3 is the third. **Location Centroid**: This method returns the location centroid. It employs the centroid formula, which is the average of all transaction sites latitude and longitude coordinates. **Standard Deviation**: This function computes the standard deviation of any particular user's transactions using the formula: square root of the variance, where variance is the sum of the squared differences between each transaction amount and the average transaction amount divided by the number of transactions. **Fraud Detection**: This function determines whether a transaction is fraudulent and provides the information. **Abnormal Transaction**: For each given user, this method returns an abnormal transaction and uses the 3 sigma outlier rule to determine if a transaction is an outlier. **Z-score**: Using the formula: (transaction amount - average transaction amount) / standard deviation, this function computes the Z score of any user's transactions and all users' transactions. **nth percentile:** Uses the formula (n/100) * (N+1), where n is the desired percentile and N is the size of the dataset to find the desired percentile.

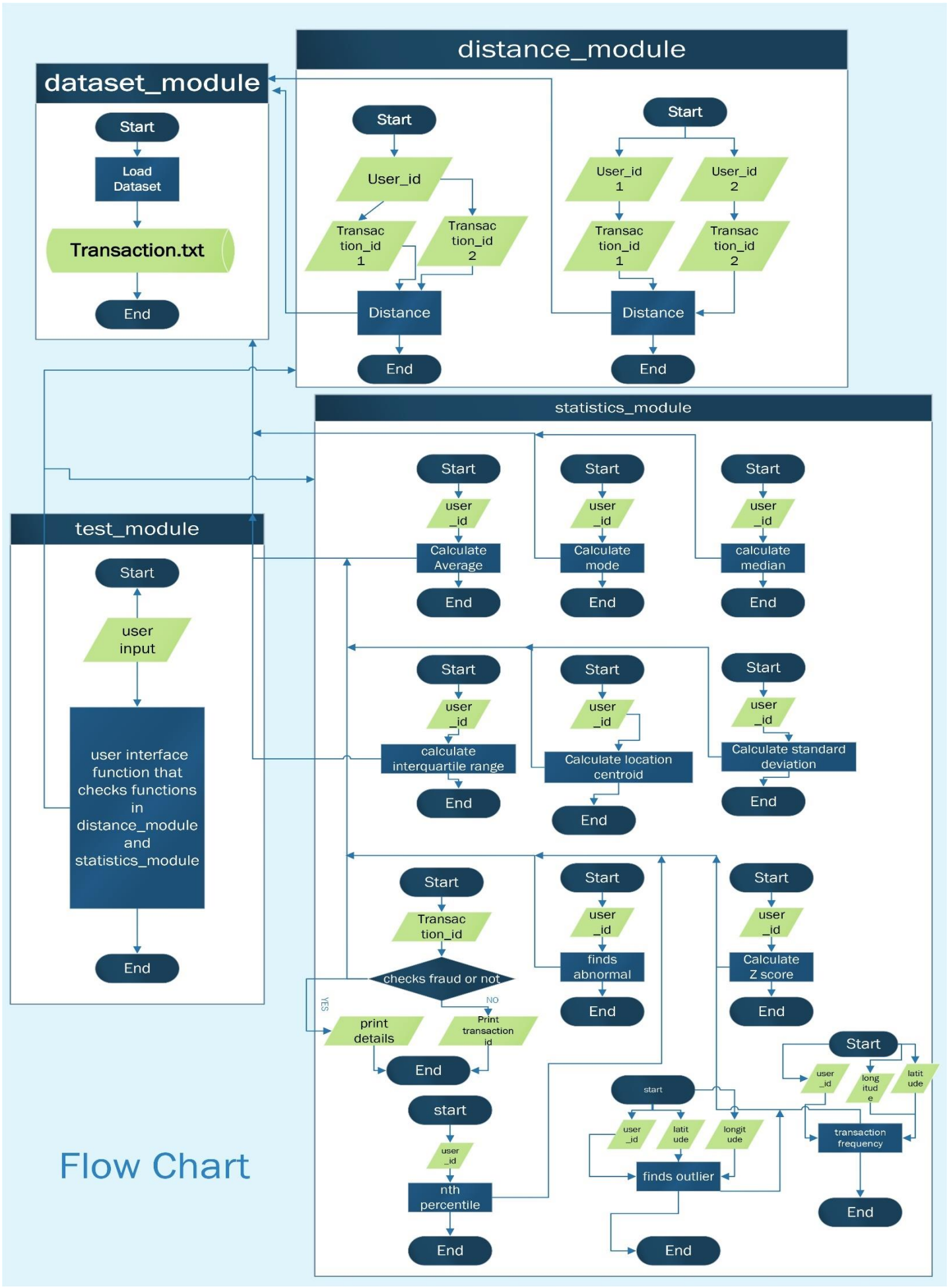
IV. Test Module: The test_module contains the function that implements the user interface through which users can query and interact with all functions in the distance_module and statistics_module. The module handles all possible errors and exceptions.

Finally, exception handling was utilised in the implementation of the four modules to prevent the programme from crashing or delivering unexpected results in the event of errors or incorrect inputs. For example, exception handling has been implemented in the dataset_module to capture issues that may occur when reading the input file, such as file not found or improper file format. The functions in the distance_module have been developed to handle scenarios where the inputs are invalid or not in the anticipated format. Similarly, exception handling has been implemented in the statistics_module to capture mistakes that may arise during computations, such as division by zero. Finally, exception handling has been implemented in the test_module to handle issues such as improper input formats or unexpected user actions.

CHAPTER 5 PROGRAM EXECUTION

Some further details on the program's execution are provided below. The flowchart in the next section of the report depicts the general flow of the complete programme from beginning to end. **Appendix A** contains the pseudocode for the two modules. This covers elements of the module code that are not depicted in depth in the flowchart, but does not include all of the code that makes up the module files. Here the transaction dataset "Transaction.txt" is initially uploaded into a file named "PCPAssignment1.ipynb". Then the "dataset_module" is created and uploaded into the ipynb file as "dataset_module.py". After that the "distance_module" with 2 functions are created and uploaded as "distance_module.py". Similarly, "statistics_module" and "test_module" are created as "statistics_module.py" and "test_module.py". Here each module has to be imported by giving the exact name since python is case sensitive. The test module imports both distance and statistics module where as the distance and statistics modules import dataset module to implement functions.

CHAPTER 6 PROGRAM STRUCTURE FLOWCHART



CHAPTER 7 REFLECTION

In today's data-driven world, the importance of analysing and understanding transaction data cannot be overstated. This is especially true in the financial sector, where the detection of fraudulent transactions can have a significant impact on an organization's bottom line. To facilitate this process, a software system can be developed, consisting of three main modules: dataset, distance, and statistics. The dataset module provides a retrieving function that loads all the data from the dataset, including attributes such as user_id, transaction_id, description, amount, location, and a Boolean label that represents the status of each transaction in terms of fraudulent.

The distance module, with its two main functions, can be used for various applications such as fraud detection. The first function calculates the distance between any two given transactions of a user, which can be used to identify anomalous transactions or transactions that are significantly different from the usual spending pattern of a user. This function can be used in fraud detection systems to flag potentially fraudulent transactions based on how different they are from the user's typical spending behaviour. The second function calculates the distance between transactions of different users, which can be used to group users with similar spending patterns together or recommend products or services that are frequently purchased by users who have similar transaction histories.

The statistics module provides a comprehensive set of functions for analysing and understanding transaction data. The module enables users to gain insights into their transaction data, identify patterns, and detect potential anomalies or fraudulent transactions. These functions include the average, mode, median, interquartile range, location centroid, standard deviation, fraud detection, abnormal transaction, Z score, frequency of transactions, outlier, and percentile functions.

To make the software more accessible to non-technical users and improve adoption and usage, a user interface can be implemented in the test module. By aggregating the functions from the distance and statistics modules into a single user interface, the software can be more easily controlled and managed, reducing the complexity of the overall system and making it easier to maintain and debug. By combining the functions from the distance and statistics modules into a single user interface, users can perform more complex queries and analyses that would not be possible with the individual modules alone.

By separating the user interface from the underlying functions, the software can be more modular and reusable. This means that the user interface can be easily replaced or updated without affecting the underlying functionality, and the same functions can be used across multiple user interfaces or applications. The implementation of a user interface in the test module can thus significantly improve the software's usability and accessibility, making it more useful to a wider range of users.

In conclusion, the development of a software system that includes a dataset module, a distance module, a statistics module, and a user interface can significantly enhance the analysis and understanding of transaction data, especially in the financial sector. By implementing user-friendly interfaces, users can interact with the functions more easily and intuitively, improving adoption and usage. Furthermore, by aggregating the functions into a single user interface, the software can be more easily controlled and managed, making it more modular and reusable. Overall, the software system can improve the detection of fraudulent transactions, enhance decision-making processes, and lead to better financial outcomes for organizations.

APPENDIX A MODULE FILES PSEUDOCODE

Module dataset_module

```
Define function get_transaction_data_set():
  try

    trans <- empty dictionary

    file_data <- open_file("transaction.txt")

    for each line in file_data:

      data <- line.split(":") and strip()

      is_fraudulent status from the data list
      user_id <- int(data[0])
      transaction_id <- int(data[1])
      description <- data[2]
      amount <- float(data[3])
      latitude <- float(data[4])
      longitude <- float(data[5])
      is_fraudulent <- data[6].lower() == "true"


      if user_id not in trans:
        trans[user_id] <- empty dictionary

      trans[user_id][transaction_id] <- {
        "description": description,
        "amount": amount,
        "latitude": latitude,
        "longitude": longitude,
        "is_fraudulent": is_fraudulent
      }


  except filenotfounderror:
    print "transaction data file not found."

  Except:print "an error occurred while reading transaction data.
  return trans
```

Module distance_module

```
Define function single_user_transaction():
  try:
    print "enter the user id:"
    get user_id<-get the user id
    print "enter the transaction id 1:<-get transaction_id1
    print "enter the transaction id 2:"<-get transaction_id2
    set data to get_transaction_data_set()
    set transaction1 to data[user_id][transaction_id1]
```

```
    set transaction2 to data[user_id][transaction_id2]
    set x1, y1 to transaction1['latitude'], transaction1['longitude']
    set x2, y2 to transaction2['latitude'], transaction2['longitude']
    set distance to square_root((x2 - x1)^2 + (y2 - y1)^2)
    print "the distance is", distance
  except valueerror:<-print "invalid input. please enter a valid integer."
  except keyerror:<-print "invalid user id or transaction id. please try again."
  except typeerror:<-print "invalid transaction data. please check the data types of latitude and longitude values."
  except zerodivisionerror:<-print "division by zero occurred during distance calculation."
  except:<-print "sorry, a different kind of error occurred!"

define function different_user_transaction():
  try:
    print "enter user id 1:"<-get user1
    print "enter user id 2:"<-get user2
    print "enter transaction id 1:"<-get transaction_id1
    print "enter transaction id 2:"<-get transaction_id2
    set data to get_transaction_data_set()<-set transaction1 to data[user1].get(transaction_id1)
    set transaction2 to data[user2].get(transaction_id2)
    if not transaction1 or not transaction2:
      return print "transactions are in the same user id please use single_user_transaction()."
    set x1, y1 to transaction1["latitude"], transaction1["longitude"]
    set x2, y2 to transaction2["latitude"], transaction2["longitude"]
    set distance to square_root((x2 - x1)^2 + (y2 - y1)^2)<-print "the distance is",<- distance
  except value error:<-print "invalid input. please enter a valid integer."
  except key error:<-print "invalid user id or transaction id. please try again."
  except typeerror:<-print "invalid transaction data. please check the data types of latitude and longitude values."
  except zerodivisionerror:<-print "division by zero occurred during distance calculation."
  except:<-print "sorry, a different kind of error occurred!"
```

Module statistics_module

```
Define function get_average():
data <- get_Transaction_data_set()
try:
  id <- int(input("Enter the user_id: "))
except ValueError:
  print("User ID must be a valid integer")
  return

user_amounts <- empty dictionary
all_amount <- empty list

for the user_id in data:

  transactions <- data[user_id]
    amounts <- empty list
  for transaction in transactions.values():
    amounts.append(transaction["amount"])
    all_amount.append(transaction["amount"])
    if len(amounts) = 0:
   then  user_amounts[user_id] <- 0
```

```
    else:
        average_amount <- sum(amounts) / len(amounts)
        user_amounts[user_id] <- average_amount

try:
    ave_user <- user_amounts[id]
except KeyError:
    print("User ID does not exist in the transaction data")
    return
ave_total <- sum(all_amount) / len(all_amount)
print("Total Transaction Average is: ave_total)

try:
    ave_user <- user_amounts[id]
except KeyError:
    print("User ID  does not exist in the transaction data")
    return
except ZeroDivisionError:
    print("User ID has no transactions")
    return

print("The average transactions of user is %f" % (id, ave_user))
return

define function: mode(values)
    mode <- no
    freq_dict <- empty the dictionary
    for each value in values:
        if the value in freq_dict:
            freq_dict[value] += 1
        else:
            freq_dict[value] <- 1
        if the mode is none or freq_dict[value] > freq_dict[mode]:
            mode <- value
    if the  freq_dict[mode] == 1:
        return non
         return mode

define function: get_the_mode()
    data <- get_transaction_dataset()
    try:
        when id <- ask the user to enter the user_id as integer
        if the id is not integer:
            print("user id must be a valid integer")
            return
    except valueerror:
        print("user id must be a valid integer")
        return
    user_amounts <- empty dictionary
    all_amount <- empty list
    for user_id in data:
        transactions <- data[user_id]
        amounts <- empty list
        for transaction in transactions.values():
```

```
            amounts.append(transaction["amount"])
            all_amount.append(transaction["amount"])
        mode_amount <- call the mode function with amounts list
        user_amounts[user_id] <- mode_amount
    try:
        mod_user <- user_amounts[id]
    except keyerror:
        print("user id %s does not exist in the transaction data" % id)
        return
    except typeerror:
        print("user id %s has no mode for transaction amounts" % id)
        return
    mod_total <- call the mode function with all_amount list
    print("total transaction mode is: %s" % (str(mod_total) if mod_total is not none else "not available"))
    print("the mode of user %s is: %s" % (id, str(mod_user) if mod_user is not none else " not available
"))


define median(values):
sort(values)    length <-len(values)
    if length % 2-> 0:
        mid <- length // 2
        median <-(values[mid - 1] + values[mid]) / 2
    else:
        mid <- length // 2
        median <-values[mid]
    return median

define get_median():
    data <-get_Transaction_data_set()
    user_amounts = {}
    all_amount = []
    for user_id in data:
        transactions = data[user_id]
        amounts = []
        for transaction in transactions.values():
            amounts.append(transaction["amount"])
            all_amount.append(transaction["amount"])
        median_amount <- median(amounts)
        user_amounts[user_id] <- median_amount
    median_user <- user_amounts[id]
    median_total <- median(all_amount)
    print(Total Transaction median)
    print(The median of user )

define function get_interquartile_range():
    id <-prompt user to input a user id and convert to integer
    data <- call get_transaction_data_set() to get transaction data set
    user_amounts <- create empty dictionary
    all_amount <- create empty list
    for each user in data:
        transactions <- get transactions for user from data
        amounts <- create empty list
        for each transaction in transactions:
```

```
            add amount for transaction to amounts list
            add amount for transaction to all_amount list
        median_amount <- calculate median amount for the user using amounts list
        add median_amount to user_amounts dictionary for user
    sorted_all_amount <- sort all_amount list in ascending order
    n <- get length of sorted_all_amount
    q1_index <- calculate index of the first quartile (q1)
    q1 <- get value of the first quartile (q1)
    q3_index <- calculate index of the third quartile (q3)
    q3 <- get value of the third quartile (q3)
    iqr <- calculate interquartile range (iqr) for all users
    print interquartile range for all users
    if id exists in user_amounts:
        user_transactions <- get transactions for user from data
        user_amounts <- create empty list
        for each transaction in user_transactions:
            append amount for transaction to user_amounts list
        sorted_user_amounts <- sort user_amounts list in ascending order
        n <- get length of sorted_user_amounts
        q1_index <- calculate index of the first quartile (q1) for the user
        q1 <- get value of the first quartile (q1) for the user
        q3_index <-calculate index of the third quartile (q3) for the user
        q3 <- get value of the third quartile (q3) for the user
        user_iqr <- calculate interquartile range (iqr) for the user
        return interquartile range for the user
    else:
        return user not found

Define function get_location_centroid():
    data <- get_Transaction_data_set()
    id <- integer value of input('Enter the user_id: ')
    user_transactions <- data.get(id)
    if user_transactions is None:
        print("User %s not found" % id)
        return
    latitudes <- empty list
    longitudes <- empty list
    for transaction in user_transactions.values():
        add transaction["latitude"] to latitudes list
        add transaction["longitude"] to longitudes list
    centroid_latitude <- sum of latitudes / length of latitudes
    centroid_longitude <- sum of longitudes / length of longitudes
    print("Centroid latitude”)
    print("Centroid longitude”)

def get_standard_deviation():
            data <- get_Transaction_data_set()
                    id <- int(input('Enter the user_id: '))
                    user_transactions <- data.get(id)
                    if user_transactions is None:
                    print("User not found”)
                    return
                    amounts <- []
                    for transaction in user_transactions.values():
```

```
                amounts.add(transaction["amount"])
                mean_price <- sum(amounts) / len(amounts)
                variance <- 0
                for amount in amounts:
                variance += (amount - mean_price) ** 2
                variance /= len(amounts)
                standard_deviation <-variance ** 0.5
        Return (Standard deviation of amounts for user))

define function is_fraudulent():
   data <- get_Transaction_data_set()
   id <- input('Enter the Transaction_id: ')
   transaction <- None
   for user_transactions in data.values():
      if id in user_transactions:
         transaction <- user_transactions[id]
         break
   if transaction is None:
      print("Transaction  not found”)
      return
   is_fraudulent <- transaction.get("is_fraudulent")
   if is_fraudulent is None:
      print("is_fraudulent not found in transaction”)
      return
   if is_fraudulent:
      print("Transaction is fraudulent")
      print("Details of transaction:")
      print(transaction)
   else:
      print("Transaction is not fraudulent")

define function get_abnormal_transaction():
   id <- prompt user to enter user id as integer

   data <- call get_transaction_data_set() function

   user_transactions <- get transactions for the specified user from the data dictionary

   if user_transactions is none:
      print "user not found"
      return

   amounts <- create a list of all transaction amounts for the specified user

   average_amount <- calculate the average transaction amount for the specified user

   std_dev <- calculate the standard deviation of transaction amounts for the specified user

   abnormal_transactions <- create an empty list to store abnormal transactions

   for each transaction for the specified user:
      if the transaction amount is an outlier:
         append the transaction id and transaction details to abnormal_transactions list
```

```
    if abnormal_transactions is not empty:
        print abnormal transactions along with their ids
    else:
        print "no abnormal transactions found for user"

function z_scores():
    try:
        data <- get_transaction_data_set()
        id <- prompt user for user_id as integer
        all_transactions <- empty list
        user_z_scores <- empty list
        all_z_scores <- empty list
        user_transactions <- data.get(id)
        if user_transactions is not empty:
            amounts <- empty list
            for transaction in user_transactions.values():
                add transaction["amount"] to amounts
            mean_amount <- calculate mean of amounts
            std_amount <- calculate standard deviation of amounts
            for transaction in user_transactions.values():
                z_score <- calculate z-score for transaction
                add z_score to user_z_scores
            print "z scores for user id:", user_z_scores
        else:
            print "user id not found"
        for user_transactions in data.values():
            amounts <- empty list
            for transaction in user_transactions.values():
                add transaction["amount"] to amounts
            mean_amount <-calculate mean of amounts
            std_amount <- calculate standard deviation of amounts
            for transaction in user_transactions.values():
                z_score <- calculate z-score for transaction
                add z_score to all_z_scores
        print "z scores for all users:", all_z_scores
    except:
            handle exceptions and errors during the execution of the function.

Define function get_location_frequency():
    data <- get_Transaction_data_set()
    latitude <- input("Enter latitude: ")
    longitude <- input("Enter longitude: ")
    location <- (latitude, longitude)
    total_frequency <- 0
    for user_id in data:
        for transaction_id in data[user_id]:
            transaction <- data[user_id][transaction_id]
            if (transaction["latitude"], transaction["longitude"]) <- location:
                total_frequency += 1
    print("The total transaction frequency at location”)

define function get_nth_percentile():
    data <- get_transaction_data_set()
```

```
    try:
        user_id <- ask for input("enter the user_id: ")
        n <- ask for input("enter the percentile (0-100): ")
    except valueerror:
        print("user id and percentile must be valid integers")
        return

    if user_id in data then
        transactions <- data[user_id]
        amounts <- [transaction["amount"] for transaction in transactions.values()]

        if len(amounts) <- 0 then
            print("user id has no transactions”)
            return

        amounts.sort()
        index <- int(n/100 * len(amounts))
        percentile <- amounts[index]

        print("the percentile of user))
    else:
        print("user id does not exist in the transaction data")

    all_amounts <- [transaction["amount"] for transactions in data.values() for transaction in
transactions.values()]

    if len(all_amounts) <- 0 then
        print("no transactions found in the dataset")
        return

    all_amounts.sort()
    index <- int(n/100 * len(all_amounts))
    percentile <- all_amounts[index]

    print("the percentile of all transactions))
end function

define function outlier
            input<- none

            id <- user input as integer ("enter the user_id")

            data <- call get_transaction_data_set()

            user_transactions <- retrieve transactions for id from data

            if user_transactions is none then
                        print "user not found"
                         return none

                        latitudes = empty list
                        longitudes = empty list

                        for each transaction in user_transactions.values():
```

```
                append transaction["latitude"] to latitudes
                 append transaction["longitude"] to longitudes

                lat_user <- calculate the centroid latitude by taking the average of the latitudes
                long_user <- calculate the centroid longitude by taking the average of the longitudes

                print "centroid latitude:", lat_user
                print "centroid longitude:", long_user

                distance_dict <- empty dictionary
                distance_ls <- empty list

        for each user_id in data:
         transactions <- retrieve transactions for user_id from data

        if id <- user_id then
          for each i in transactions.keys():
                latitude <- retrieve the latitude of the transaction
                longitude <- retrieve the longitude of the transaction
                distance <- calculate the distance between the transaction and the user's centroid
         append distance to distance_ls
        add distance to distance_dict with i as the key

index1 <- calculate the index of the first quartile
q1 <- calculate the first quartile
index3 <- calculate the index of the third quartile
q3 <- calculate the third quartile
iqr <- calculate the interquartile range
outlierlocations <- empty list

for each i in distance_ls:
   if i > (q3 + (1.5 * iqr)) or i < (q1 - (1.5*iqr)):
      append i to outlierlocations

keys <- empty list

for each value in distance_dict.values():
   if value is in outlierlocations:
      for each key, val in distance_dict.items():
         if val is equal to value:
            append key as a string to keys

print "the outlier transactions ids are :"
return none
```

Module test_module

```
import distance_module as mod
import statistics_module as modu
function user_interface():
   while true:
      display "dear customer welcome to our transaction analysis tool"
      display "-------------------------------------------------------"
      display "please choose from the below given choices,thank you."
```

```
display ""
display "distance"
display "1. calculate distance between two transactions of a single user"
display "2. calculate distance between two transactions of different users"
display ""
display "transaction"
display "3. calculate the average transactions of given user_id and all user_ids."
display "4. calculate the mode of transactions of the given user_id and all user_ids"
display "5. calculate the median of transactions of the given user_id and all user_ids"
display "6. calculate the interquartile range of transactions of the given user_id and all user_ids"
display "7. calculate the location centroid of the given user_id, based on their transaction locations"
display "8. calculate the standard deviation of transactions of the given user_id."
display "9. detect whether the given transaction_id is fraudulent or not and also specify the details of that transaction_id"
display "10. find the abnormal transactions for the given user_id"
display "11. compute the z score of transactions of the given user_id and all user_ids"
display "12. computes the frequencies of transactions at any given location"
display "13. calculate the nth percentiles of transactions of any user and all of users"
display "14. quit"
choice <- get_input("enter your choice: ")
        If choice <- 1
            Mod(single_user_transaction())

        else if choice <- 2
            mod(different_user_transaction())

        else if choice <- 3
           modu(get_average())

         else if choice <- 4
            modu(get_mode())

         else if choice <- 5
            modu(get_median())

         else if choice <- 6
            modu(get_interquartile_range())

         else if choice <-7
            modu(get_location_centroid())

         else if choice <-8
            modu(get_standard_deviation())

         else if choice <- 9
            modu(is_fraudulent())

          else if choice <- 10
              modu(get_abnormal_transaction())

          else if choice <- 11
               modu(z_scores())
          elif choice <- 12
```

```
    modu(get_location_frequency())
 elif choice <- 13
    modu(get_nth_percentile())
elif choice <- 14
   print("Goodbye!")
   break
else:
     return Sorry dear customer,Invalid choice, please try again.
```