Fraud Detection System

Implementation Report

Table of Contents

[CHAPTER 1 INTRODUCTION 3](#_Toc132477552)

[CHAPTER 2 PROBLEM ANALYSIS 3](#_Toc132477553)

[CHAPTER 3 SOLUTION REQUIREMENTS 4](#_Toc132477554)

[CHAPTER 4 IMPLEMENTATION OF SOLUTION 4](#_Toc132477555)

[CHAPTER 5 PROGRAM EXECUTION 5](#_Toc132477556)

[CHAPTER 6 PROGRAM STRUCTURE FLOWCHART 6](#_Toc132477557)

[CHAPTER 7 REFLECTION 7](#_Toc132477558)

[APPENDIX A MODULE FILES PSEUDOCODE 8](#_Toc132477559)

[Module dataset\_module 8](#_Toc132477560)

[Module distance\_module 8](#_Toc132477561)

[Module statistic\_module 11](#_Toc132477562)

[Module test\_module 18](#_Toc132477563)

# CHAPTER 1 INTRODUCTION

This report begins with an analysis of the proposed and implemented problem and provides a brief overview of the three parts of the program and the required similarity methods. The software requirements are also outlined in this solution. The following implementation is discussed, including a mention of significant programming concepts used in the software development process. Following this section is the information the user needs to run the program correctly. That is pursued by a program flow chart showing the overall flow of the entire program from start to finish. Since the flowchart does not provide a detailed overview of the logic of the modules, the pseudocode for the three modules is contained Appendix. It only shows the parts of the module code not described in the flowchart, so it does not include the entire module code. The final part of the report deals with a personal reflection on what parts of the process went well and what did not, and suggestions for improvements or alternative ways the solutions could have been implemented.

# CHAPTER 2 PROBLEM ANALYSIS

This project brief involves designing and developing an intelligent recommendation engine for a bank's online platform. The focus of the project is on analyse information from user purchasing transactions.

The first part of the program developed concentrates on retrieving data from provided datasets. The data to be retrieved is a nested dictionary containing user transaction, i.e. , users and transaction they have performed (including the location and amount of money spent in that transaction). In its module function also handle and address all possible errors and exceptions.

The second part of the program focuses on computing the distance between any two transactions of any user, as well as a function for computing the distance between transactions of any two given users. This is a crucial step in determining the similarity between users' transaction patterns and making recommendations based on that information. The program imports the third-party Numpy module and uses it to calculate the square root in the Euclidean distance formula within the module. Additionally, some decorator functions are provided to handle input exceptions such as incorrect user ID type, incorrect transaction ID type, or non-existent user ID or transaction ID. For example, if the user inputs alphabet characters ‘abc8’, the program will display an error message indicating that the input is the wrong type of user ID or transaction ID. Similarly, if the user inputs an incorrect user ID or transaction ID, the program will display an error notification stating that the user ID or transaction ID does not exist.

The third part of the program involves designing and implementing 18 functions that use the retrieved data to compute basic statistics on transactions for any user or all users. This enables program users to analyse the basic parameters of transactions and monitor their status. The information gathered from these statistics can be used to gain insights into user behaviour, preferences, and spending patterns, ultimately leading to personalized recommendations based on that data. Additionally, the program reuses the decorator function from the second part and develops new decorator functions to catch invalid input before executing several statistic functions. The program imports third party libraries such as Numpy and Statistics to provide a convenient way to work with arrays and perform various mathematical operations on them. By using these libraries to calculate operations, the program can improve the speed and accuracy of its calculations, making it more efficient and reliable.

# CHAPTER 3 SOLUTION REQUIREMENTS

In dataset\_module:

* Implement a function that retrieves the attributes/features of data and returns a nested dictionary.

In distance\_module:

* Implement a function that computes the distance between any two given transactions of a user.
  + Each should accept parameters of user ID, transaction ID 1, and transaction ID 2.
* Implement a function that computes the distance of transactions any two users.
  + Each should accept parameters of user ID 1, transaction ID 1, and user ID 2, transaction ID 2.

In statistic\_module:

* Implement a function that returns the average, mode, median, interquartile, standard deviation, z score transactions, and nth percentile of transactions of any user
  + Each should accept parameters of user ID
* Implement functions that returns the average, mode, median, interquartile range, Z score, nth percentiles transactions of all user
* Implement a function that returns the location centroid of any user, based on their transaction locations.
  + Each should accept parameters of user ID
* Implement a function that determines whether a transaction is fraudulent or not. Then it provides details of such transactions.
  + Each should accept parameters of transaction ID
* Implement a function that computes those frequencies of transactions at any given location.
  + Each should accept parameters of x, y points
* Implement a function that returns the nth percentiles of transactions of all users.

In the test\_module:

* Implement a function that contains the user interface through which users can query and interact with all functions in distance\_module and statistic\_module above.

# CHAPTER 4 IMPLEMENTATION OF SOLUTION

The program uses exception handling mainly focused on potential file handling errors. The program only catches IO errors while checking file presence and readability. This is better than catching all possible exceptions, as it avoids misleading error messages and helps the user identify the issue more easily.

When user input an argument value, ‘get\_x’ functions are used to seek that ‘x’ value in the dictionary. After the user enter a value, the program will immediately verify the value by the decorator functions before executing any other distance and statistic function. By this way, determining user and transaction in this stage help prevent future errors and exceptions, therefore, requesting handling errors and exceptions also decrease in other parts in the program.

The program has implemented a function for option menus, which display specific menu options to the user and receive their choice, simplifying the process of accessing available options. The function uses a dictionary with keys is choice numbers and the value of these key contain mean of its statistic and its statistic function name. For example, “{ 1: {'statistic\_name': 'Distance of a user', 'option': distance\_of\_any\_two\_given\_transaction\_of\_a\_user}}” with ‘1’ is choice number, ‘Distance of a user’ is statistic name and ‘distance\_of\_any\_two\_given\_transaction\_of\_a\_user’ is statistic function. In addition, these functions can be reused throughout the program, avoiding the repetition of code. The menu choices are represented by numbers for simplicity, and the program will display an error message then loop back to the user can re-enter a new value if the user inputs an incorrect choice.

As well as functions, parameter are used to pass standard data types such as numerical data. Specially, in the test\_module, the program will not pass any parameters to call the function because before these statistic function executed, these parameters will be entered, check the validation then override these parameters to the statistic function via decorator functions. By this way, the statistic function will only be called only if every parameter is valid, and as referred it will reduce some future program problems.

An exit method will be called whenever user input wrong menu choice as well as argument values and when user enter all valid input and then the program print out results the screen. Therefore, if user want to stop the program, they can enter ‘yes’ or ‘y’. The uppercase for this choice also accepts without shown any error message because it always is converted to lowercase to simplify corresponding selection statements. Conversely, the user can type any characters to loop back the menu to input new statistic choice then enter argument values.

The program uses global variables that are declared at the beginning of the code. The advantage of using parameters over global variables is that the data can be protected from unknown persistent changes that may occur within the function.

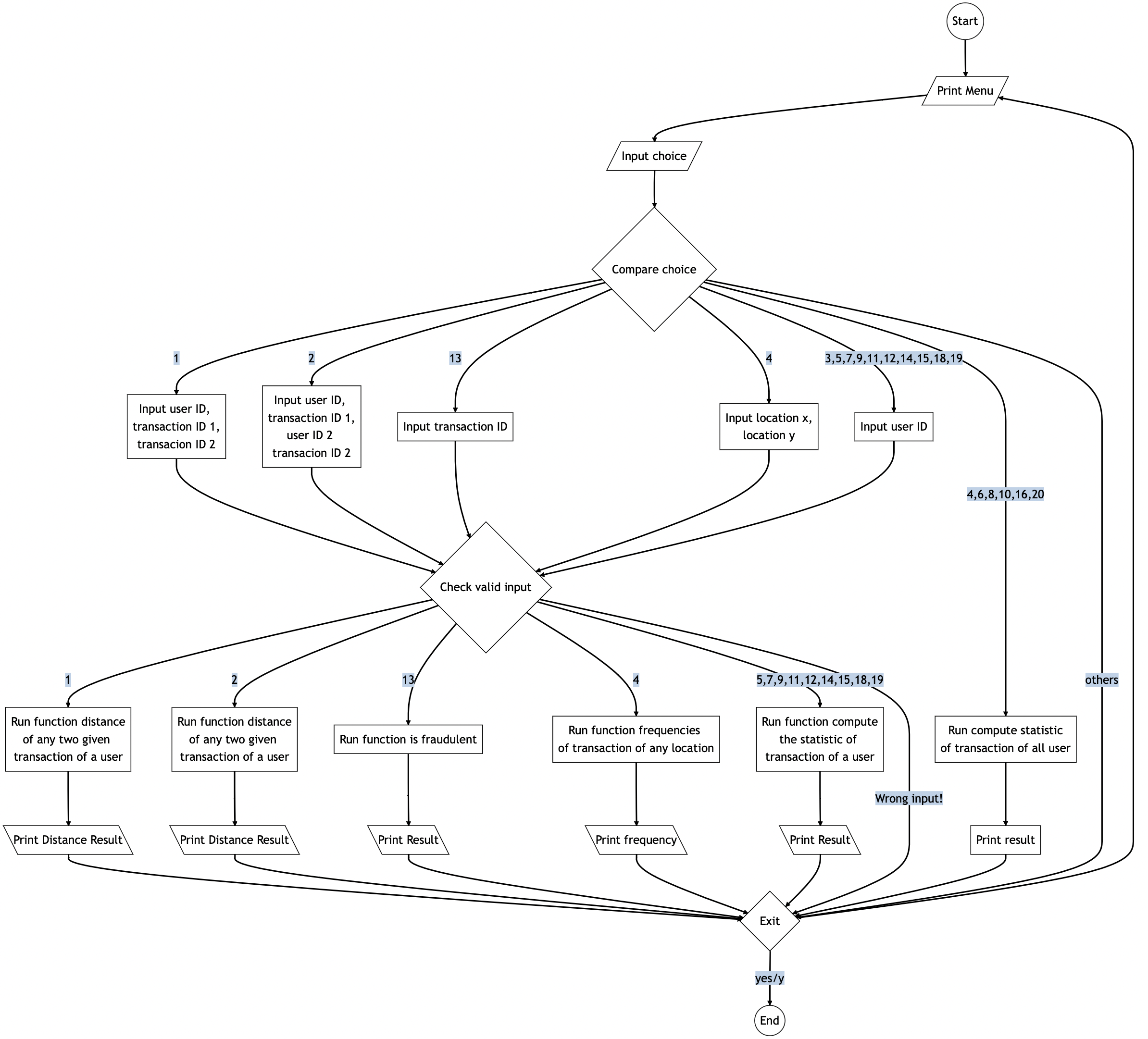
# CHAPTER 5 PROGRAM EXECUTION

Below are some additional details about how the program execute. The flowchart in the upcoming section of the report provides an overview of the program's start-to-finish execution flow. Pseudocode for the three modules can be found in Appendix A, which includes portions of the module code that are not explicitly depicted in the flowchart. However, the pseudocode does not include all the code that is part of the module files.

In order for the program to function properly, the notebook file named "test\_module.ipynb" and the three module files named "dataset\_module.py", "statistic\_module.py", and "statistics\_module.py" must be placed in the same folder as the data file named "transaction.txt". All of these files must be located within the same directory for the program to execute correctly.

The program's menu options can be accessed by entering a corresponding number when prompted. This means that the user can simply enter the number of the option they wish to select rather than typing out the description of the option. For example, if the menu presents the options "4. Compute the average" and "6. Compute standard deviation", the user can simply enter "4" or "6" to select the desired option.

# CHAPTER 6 PROGRAM STRUCTURE FLOWCHART



# CHAPTER 7 REFLECTION

With regarding to finding abnormal transactions or outliers in the third part of the program, there are three popular approaches such as using Turkey algorithm based on IQR (Interquartile range) method, using statistic method (i.e. Z-score) base on the difference of a value with mean and standard deviation in data set or using machine learning. The best approach is using clustering to find outlier in data set because it helps group similar data points together, make it easier to identify groups that contain outliers as well as detect non-uniform clusters, which may contain outlier data points. Using this approach, the clustering method can quickly and accurately identify outlier data points which may be more effective than others. However, this method required big data set as well as the requirement in this program does not refer the accuracy of output, so the program can use this method to optimize its output in future.

On the other hand, IQR method and Z-score are other method which help approaching easier. The IQR method uses the interquartile range (IQR) to identify outliers. The IQR is calculated as the difference between the third quartile and the first quartile of the data. Any data point that falls below the first quartile minus 1.5 times the IQR or above the third quartile plus 1.5 times the IQR is considered an outlier. Z-score, statistic method uses the z-score to identify outliers. The z-score measures how many standard deviations a data point is from the mean of the data set. Any data point that has a z-score greater than a certain threshold, typically 2.5 or 3, is considered an outlier. The table blow compares each of these method advantages and disadvantages:

|  |  |  |
| --- | --- | --- |
|  | IQR | Z-score |
| Pros | * Simple and easy to understand. * Suitable for datasets with non-normal distributions or significant outliers. * Fast and efficient calculation when used directly on the dataset. | * Allows for determination of the level of outlier for each data point. * Suitable for datasets with normal or near-normal distributions. * Helps differentiate between different types of outliers in the dataset. |
| Cons | * Does not work well with datasets with asymmetric or skewed distributions. * Cannot differentiate between different types of outliers in the dataset. * Does not allow for accurate determination of the level of outlier for each data point. | * Does not work well with datasets with asymmetric or skewed distributions. * Requires calculation of standard deviation and mean on the dataset, and therefore may be affected by outlier values. * When dealing with large datasets, calculating Z-scores can be time-consuming and computationally expensive. |

Despite the lack of information about the normality of the data distribution, IQR method is chosen to seek the abnormal outliers. Besides, IQR method is also easy to understand and fast to compute, therefore, it suits with big dataset and non-normal distributions. That’s why the program choose this method to seek abnormal transactions.

# APPENDIX A MODULE FILES PSEUDOCODE

## Module dataset\_module

function get\_data(filename=''):  
 try:  
 with open(filename, 'r') as file:  
 transaction = {}  
 for i in file:  
 split\_line = i.split(':')  
 mini\_diction = {'fraudulent': False if split\_line[6].strip() == 'false' else True,  
 'x': float(split\_line[4]),  
 'y': float(split\_line[5]),  
 'amount': float(split\_line[3]),  
 'description': split\_line[2]}  
 user\_id = int(split\_line[0])  
 transaction\_id = int(split\_line[1])  
 if user\_id not in transaction.keys():  
 transaction[user\_id] = {transaction\_id: mini\_diction}  
 else:  
 transaction[user\_id][transaction\_id] = mini\_diction  
 return transaction  
 except FileNotFoundError:  
 print(f'Sorry, the file {filename} does not exist')

endfunction

## Module distance\_module

from dataset\_module import get\_data

from math import sqrt

source\_data ← get\_data('Transaction.txt')

class InvalidInputError(Exception):

pass

function integer\_check(index):

"""

Function to input value

:param index: any given value

:return: an integer value

"""

value ← input(f'Please insert {index}: ')

if not value.isdigit():

raise InvalidInputError(f'Invalid {index}. Please input a number.')

endif

return int(value)

endfunction

function get\_value(index, data):

"""

Function to find input value in dataset

:param index: input value

:param data: dataset

:return: a correct value

"""

value ← integer\_check(index)

if value not in data:

raise InvalidInputError(f'{index} {value} does not exist')

endif

return value

endfunction

function get\_two\_transaction\_id\_and\_a\_user\_id(func):

"""

This is decorator to check two transaction ID of two user id are valid or not in source data

:param func:

:return: function with correct parameters

"""

function wrapper():

try:

user\_id ← get\_value('User\_id', source\_data)

trans\_1\_id ← get\_value('Transaction\_id\_1', source\_data[user\_id])

trans\_2\_id ← get\_value('Transaction\_id\_2', source\_data[user\_id])

return func(user\_id, trans\_1\_id, trans\_2\_id)

except InvalidInputError as e:

print(str(e))

endfunction

return wrapper

endfunction

function get\_two\_transaction\_id\_and\_two\_user\_id(func):

"""

This is decorator to check two transaction ID and two user id are valid or not in source data

"""

function wrapper():

try:

user\_id\_1 ← get\_value('User\_id\_1', source\_data)

trans\_1\_id ← get\_value('Transaction\_id\_1', source\_data[user\_id\_1])

user\_id\_2 ← get\_value('User\_id\_2', source\_data)

trans\_2\_id ← get\_value('Transaction\_id\_2', source\_data[user\_id\_2])

return func(user\_id\_1, trans\_1\_id, user\_id\_2, trans\_2\_id)

except InvalidInputError as e:

print(str(e))

endfunction

return wrapper

endfunction

@get\_two\_transaction\_id\_and\_a\_user\_id

function distance\_of\_any\_two\_given\_transaction\_of\_a\_user(user\_id, trans\_1\_id, trans\_2\_id):

"""

Compute distance of any two given transaction of a user

:param user\_id: a given user ID. e.g. 21

:param trans\_1\_id: a first transaction ID of the user ID. e.g. 500000

:param trans\_2\_id: a second transaction ID of the user ID. e.g. 500001

:return: a float value of distance

"""

x\_1 ← source\_data[user\_id][trans\_1\_id]['x']

y\_1 ← source\_data[user\_id][trans\_1\_id]['y']

x\_2 ← source\_data[user\_id][trans\_2\_id]['x']

y\_2 ← source\_data[user\_id][trans\_2\_id]['y']

print(f'Distance of any two given transaction of a user\_id {user\_id}:')

print(f'First is {source\_data[user\_id][trans\_1\_id]}')

print(f'Second is {source\_data[user\_id][trans\_2\_id]}')

return sqrt((x\_1 - x\_2) \*\* 2 + (y\_1 - y\_2) \*\* 2)

endfunction

@get\_two\_transaction\_id\_and\_two\_user\_id

function distance\_of\_any\_two\_given\_transaction\_of\_any\_user(user\_id\_1, trans\_1\_id, user\_id\_2, trans\_2\_id):

"""

Compute distance of any two given transaction of two user

:param user\_id\_1: a first user ID. e.g. 21

:param trans\_1\_id: a first transaction ID of the first user ID. e.g. 500000

:param user\_id\_2: a second user ID. e.g. 22

:param trans\_2\_id: a first transaction ID of the second user ID. e.g. 500200

:return: a float value of distance

"""

x\_1 ← source\_data[user\_id\_1][trans\_1\_id]['x']

y\_1 ← source\_data[user\_id\_1][trans\_1\_id]['y']

x\_2 ← source\_data[user\_id\_2][trans\_2\_id]['x']

y\_2 ← source\_data[user\_id\_2][trans\_2\_id]['y']

print(f'Information of any two given transaction of a user\_id {user\_id\_1} and transaction id {trans\_1\_id} is:')

print(f'First is {source\_data[user\_id\_1][trans\_1\_id]}')

print(f'Information of any two given transaction of a user\_id {user\_id\_2} and transaction id {trans\_2\_id} is:')

print(f'Second is {source\_data[user\_id\_2][trans\_2\_id]}')

return sqrt((x\_1 - x\_2) \*\* 2 + (y\_1 - y\_2) \*\* 2)

endfunction

## Module statistic\_module

from distance\_module import source\_data, get\_value, integer\_check, InvalidInputError

import statistics as stats

import numpy as np

function get\_valid\_user\_id(func):

"""

Decorator to find valid user Id input in dataset

:param func:

:return: function with a valid user Id parameter

"""

function wrapper():

try:

user\_id ← get\_value('User\_id', source\_data)

return func(user\_id)

except InvalidInputError as e:

print(str(e))

endfunction

return wrapper

endfunction

function get\_list\_amount\_of\_a\_user(user\_id):

"""

Get all transactions of a user according to a given user id

:param user\_id: a given user id

:return: array transactions of the user

"""

return np.array([v.get('amount') for k, v in source\_data[user\_id].items()])

endfunction

function get\_list\_amount\_of\_all\_user():

"""

Get all transactions

:return: array of amounts of all transactions

"""

return np.array([value['amount'] for user\_id in source\_data for transaction, value in source\_data[user\_id].items()])

endfunction

@get\_valid\_user\_id

function average\_of\_a\_user(user\_id):

"""

Calculate the average of transactions according to a given user id

:param user\_id: a given user id

:return: the average of the transactions

"""

amounts ← get\_list\_amount\_of\_a\_user(user\_id)

return amounts.mean()

endfunction

function average\_of\_all\_user():

"""

Calculate the average of transactions of all users

:return: list average of transactions of every user

"""

return get\_list\_amount\_of\_all\_user().mean()

endfunction

@get\_valid\_user\_id

function mode\_of\_a\_user(user\_id):

"""

Calculate the mode of transactions of any given user

:param user\_id: a given user id

:return: mode of the transaction

"""

amounts ← get\_list\_amount\_of\_a\_user(user\_id)

return stats.mode(amounts)

endfunction

function mode\_of\_all\_user():

"""

Calculate the mode of transactions of all user

:return: list mode of the user's transactions.

"""

return stats.mode(get\_list\_amount\_of\_all\_user())

endfunction

@get\_valid\_user\_id

function median\_of\_a\_user(user\_id):

"""

Calculate the median of transactions of any given user

:param user\_id: a given user id

:return: median of the transaction

"""

amounts ← get\_list\_amount\_of\_a\_user(user\_id)

return np.median(amounts)

endfunction

function median\_of\_all\_user():

"""

Calculate the median of transactions of all user

:return: list median of the user's transactions.

"""

return np.median(get\_list\_amount\_of\_all\_user())

endfunction

@get\_valid\_user\_id

function interquartile\_of\_a\_user(user\_id):

"""

Calculate the interquartile of transactions of any given user

:param user\_id: a given user id

:return: interquartile of the transaction

"""

amounts ← get\_list\_amount\_of\_a\_user(user\_id)

q\_3, q\_1 ← np.percentile(amounts, [75, 25])

iqr ← q\_3 - q\_1

return iqr

endfunction

function interquartile\_of\_all\_user():

"""

Calculate the interquartile of transactions of all user

:return: list interquartile of the user's transactions.

"""

amounts ← get\_list\_amount\_of\_all\_user()

q\_3, q\_1 ← np.percentile(amounts, [75, 25])

iqr ← q\_3 - q\_1

return iqr

endfunction

@get\_valid\_user\_id

function location\_centroid\_a\_user(user\_id):

"""

Calculate the location centroid of any user based on their transaction locations.

:param user\_id: a given user id

:return: x, y point of location centroid

"""

list\_of\_location ← np.array([[v.get('x'), v.get('y')] for k, v in source\_data[user\_id].items()])

return list\_of\_location.mean(axis=0)

endfunction

@get\_valid\_user\_id

function standard\_deviation\_of\_a\_user(user\_id):

"""

Calculate the standard deviation of any user.

:param user\_id: a given user id

:return: standard deviation of the user

"""

amounts ← get\_list\_amount\_of\_a\_user(user\_id)

return amounts.std()

endfunction

function standard\_deviation\_of\_all\_user():

"""

Calculate the standard deviation of all user.

:return: standard deviation of all user

"""

amounts ← get\_list\_amount\_of\_all\_user()

return amounts.std()

endfunction

function get\_valid\_integer\_type\_of\_transaction\_id(func):

function wrapper():

try:

nth ← integer\_check('transaction\_id')

return func(nth)

except InvalidInputError as e:

print(str(e))

endfunction

return wrapper

endfunction

@get\_valid\_integer\_type\_of\_transaction\_id

function is\_fraudulent(transaction\_id):

"""

Check a transaction is fraudulent or not

:param transaction\_id: a given transaction id

:return: list of status of fraudulent and the transaction information.

"""

list\_of\_fraudulent ← list(source\_data.values())

for index in list\_of\_fraudulent:

for key, value in index.items():

if key == transaction\_id:

return value['fraudulent'], value

endif

endfor

endfor

raise InvalidInputError('Transaction Id does not exist')

endfunction

@get\_valid\_user\_id

function abnormal\_transaction\_of\_a\_user(user\_id):

"""

Calculate abnormal transactions of a user

:param user\_id: a given user id

:return: list of abnormal transaction

"""

amounts ← np.array([[int(transaction\_id), v.get('amount')] for transaction\_id, v in source\_data[user\_id].items()])

q\_3\_x, q\_1\_x ← np.percentile(amounts[:, 1], [75, 25])

iqr\_x ← q\_3\_x - q\_1\_x

lower\_x ← q\_1\_x - 1.5 \* iqr\_x

upper\_x ← q\_3\_x + 1.5 \* iqr\_x

return [(int(transaction\_id), source\_data[user\_id][transaction\_id]) for transaction\_id, v in amounts if

v <= lower\_x or v >= upper\_x]

endfunction

@get\_valid\_user\_id

function z\_score\_of\_a\_user(user\_id):

"""

Calculate z\_score of a user

:param user\_id: a given user id

:return: list of z\_score

"""

amounts ← get\_list\_amount\_of\_a\_user(user\_id)

mean ← np.mean(amounts)

std ← np.std(amounts)

return [[x, (x - mean) / std] for x in get\_list\_amount\_of\_a\_user(user\_id)]

endfunction

function z\_score\_of\_all\_user():

"""

Calculate z\_score of transactions of all user

:return: list z\_score of the user's transactions.

"""

mean ← average\_of\_all\_user()

std ← standard\_deviation\_of\_all\_user()

return [[x, (x - mean) / std] for x in get\_list\_amount\_of\_all\_user()]

endfunction

function get\_location(func):

"""

Input value and check input exception

:param func:

:return: function with 2 correct parameters

"""

function wrapper():

try:

location\_x ← input('Please input x point: ')

if not location\_x.replace('.', '', 1).isdigit():

raise InvalidInputError(f'Invalid x coordinate {location\_x}. Please input a float number.')

endif

location\_y ← input('Please input y point: ')

if not location\_y.replace('.', '', 1).isdigit():

raise InvalidInputError(f'Invalid x coordinate {location\_y}. Please input a float number.')

endif

return func(float(location\_x), float(location\_y))

except InvalidInputError as e:

print(str(e))

endfunction

return wrapper

endfunction

@get\_location

function frequencies\_of\_transaction(location\_x, location\_y):

"""

Computes those frequencies of transactions at any given location.

:param location\_x: a given x point

:param location\_y: a given y point

:return: those frequencies of transactions based x, y given point.

"""

freq ← 0

for key\_1, item in source\_data.items():

for key\_2, item2 in item.items():

if item2['x'] == location\_x and item2['y'] == location\_y:

freq += 1

endif

endfor

endfor

return freq

endfunction

@get\_valid\_user\_id

function outlier\_any\_location\_of\_any\_user(user\_id):

"""

Computes outlier of any location and of any user.

:param user\_id: a given user id

:return: list of outlier

"""

location ← np.array([[v.get('x'), v.get('y')] for k, v in source\_data[user\_id].items()])

x ← location[:, 0:1]

y ← location[:, 1:2]

# calculate Q1 and Q3 of location of x coordinates

q\_3\_x, q\_1\_x ← np.percentile(x, [75, 25])

iqr\_x ← q\_3\_x - q\_1\_x

lower\_x ← q\_1\_x - 1.5 \* iqr\_x

upper\_x ← q\_3\_x + 1.5 \* iqr\_x

# calculate Q1 and Q3 of location of x coordinates

q\_3\_y, q\_1\_y ← np.percentile(y, [75, 25])

iqr\_y ← q\_3\_y - q\_1\_y

lower\_y ← q\_1\_y - 1.5 \* iqr\_y

upper\_y ← q\_3\_y + 1.5 \* iqr\_y

outlier ← [[l\_x, l\_y] for l\_x, l\_y in location if

(l\_x < lower\_x or l\_x > upper\_x or l\_y < lower\_y or l\_y > upper\_y)]

return outlier

endfunction

function get\_user\_id\_and\_nth(func):

"""

Decorator to check user id and nth value valid or not

:param func:

:return: function with two valid parameters

"""

function wrapper():

try:

user\_id ← integer\_check('user\_id')

nth ← integer\_check('nth')

return func(user\_id, nth)

except InvalidInputError as e:

print(str(e))

endfunction

return wrapper

endfunction

@get\_user\_id\_and\_nth

function nth\_percentile\_of\_a\_user(user\_id, n):

"""

Nth percentiles of transactions of any user.

:param user\_id: a given user id

:param n: nth percentiles

:return: the nth percentiles of transactions of the user

"""

amount ← get\_list\_amount\_of\_a\_user(user\_id)

return np.percentile(amount, n)

endfunction

function get\_valid\_nth(func):

function wrapper():

try:

nth ← integer\_check('nth')

return func(nth)

except InvalidInputError as e:

print(str(e))

endfunction

return wrapper

endfunction

@get\_valid\_nth

function nth\_percentile\_of\_all\_user(n):

"""

Nth percentiles of transactions of all user.

:param n: nth percentiles

:return: the nth percentiles of transactions of the user

"""

amount ← get\_list\_amount\_of\_all\_user()

return np.percentile(amount, n)

endfunction

## Module test\_module

from distance\_module import (

distance\_of\_any\_two\_given\_transaction\_of\_a\_user,

distance\_of\_any\_two\_given\_transaction\_of\_any\_user,

integer\_check,

InvalidInputError,

)

import statistics\_module as st

function main():

statistic\_functions ← {

1: {

'statistic\_name': 'Distance of a user',

'option': distance\_of\_any\_two\_given\_transaction\_of\_a\_user

},

2: {

'statistic\_name': 'Distance of two user',

'option': distance\_of\_any\_two\_given\_transaction\_of\_any\_user

},

3: {

'statistic\_name': 'Average of a user',

'option': st.average\_of\_a\_user

},

4: {

'statistic\_name': 'Average of all user',

'option': st.average\_of\_all\_user

},

5: {

'statistic\_name': 'Mode of a user',

'option': st.mode\_of\_a\_user

},

6: {

'statistic\_name': 'Mode of all user',

'option': st.mode\_of\_all\_user

},

7: {

'statistic\_name': 'Median of a user',

'option': st.median\_of\_a\_user

},

8: {

'statistic\_name': 'Median of all user',

'option': st.median\_of\_all\_user

},

9: {

'statistic\_name': 'Interquartile Range of a user',

'option': st.interquartile\_of\_a\_user

},

10: {

'statistic\_name': 'Interquartile Range of all user',

'option': st.interquartile\_of\_all\_user

},

11: {

'statistic\_name': 'Location Centroid',

'option': st.location\_centroid\_a\_user

},

12: {

'statistic\_name': 'Standard Deviation',

'option': st.standard\_deviation\_of\_a\_user

},

13: {

'statistic\_name': 'Check fraudulent',

'option': st.is\_fraudulent

},

14: {

'statistic\_name': 'Abnormal Transaction',

'option': st.abnormal\_transaction\_of\_a\_user

},

15: {

'statistic\_name': 'Z-score of a user',

'option': st.z\_score\_of\_a\_user

},

16: {

'statistic\_name': 'Z-score of all user',

'option': st.z\_score\_of\_all\_user

},

17: {

'statistic\_name': 'Frequency of transaction location',

'option': st.frequencies\_of\_transaction

},

18: {

'statistic\_name': 'Outliers of location',

'option': st.outlier\_any\_location\_of\_any\_user

},

19: {

'statistic\_name': 'Nth quartile of a user',

'option': st.nth\_percentile\_of\_a\_user

},

20: {

'statistic\_name': 'Nth quartile of all user',

'option': st.nth\_percentile\_of\_all\_user

}

}

message\_menu ← """Please choose one of these actions:

1. Compute the distance between any two given transactions of a user.

2. Compute the distance of transactions of any two users.

3. Compute the average transactions of any user

4. Compute the average transactions of all users.

5. Compute the mode of transactions of any user.

6. Compute the mode of transactions of all users.

7. Compute the median of all transactions of a user.

8. Compute the median of all transactions of all users.

9. Compute the interquartile range of any user’s transactions.

10. Compute the interquartile range of all user’s transactions.

11. Compute the location centroid of any user based on their transaction locations.

12. Compute the standard deviation of any specific user’s transaction.

13. Determine whether a transaction is fraudulent or not and provide details of such transactions.

14. Find an abnormal transaction for any given user.

15. Compute the Z-score of any user’s transactions.

16. Compute the Z-score for all users’ transactions.

17. Compute the frequencies of transactions at any given location.

18. Find the outlier of any location and of any user.

19. Compute the nth percentiles of transactions of any user

20. Compute the nth percentiles of transactions of all users.

"""

while True:

try:

print(message\_menu)

choice ← integer\_check('Choice')

if choice in statistic\_functions:

statistic\_name ← statistic\_functions[choice]['statistic\_name']

statistic\_fn ← statistic\_functions[choice]['option']

print(f"{statistic\_name} :", statistic\_fn())

else:

print('Input choice wrong!!!')

endif

except InvalidInputError as e:

print(str(e))

except Exception as e:

print(str(e))

finally:

continue\_type ← input('Do you want to exit: yes or no ')

if continue\_type.lower() == 'yes' or continue\_type.lower() == 'y':

print('Good bye! ')

break

else:

continue

endif

endwhile

endfunction

main()