Software Testing Report <Data visualization>

Shinzo TANIMOTO

Duwon HA

James SO

Table of Contents

1.0 Unit Tests

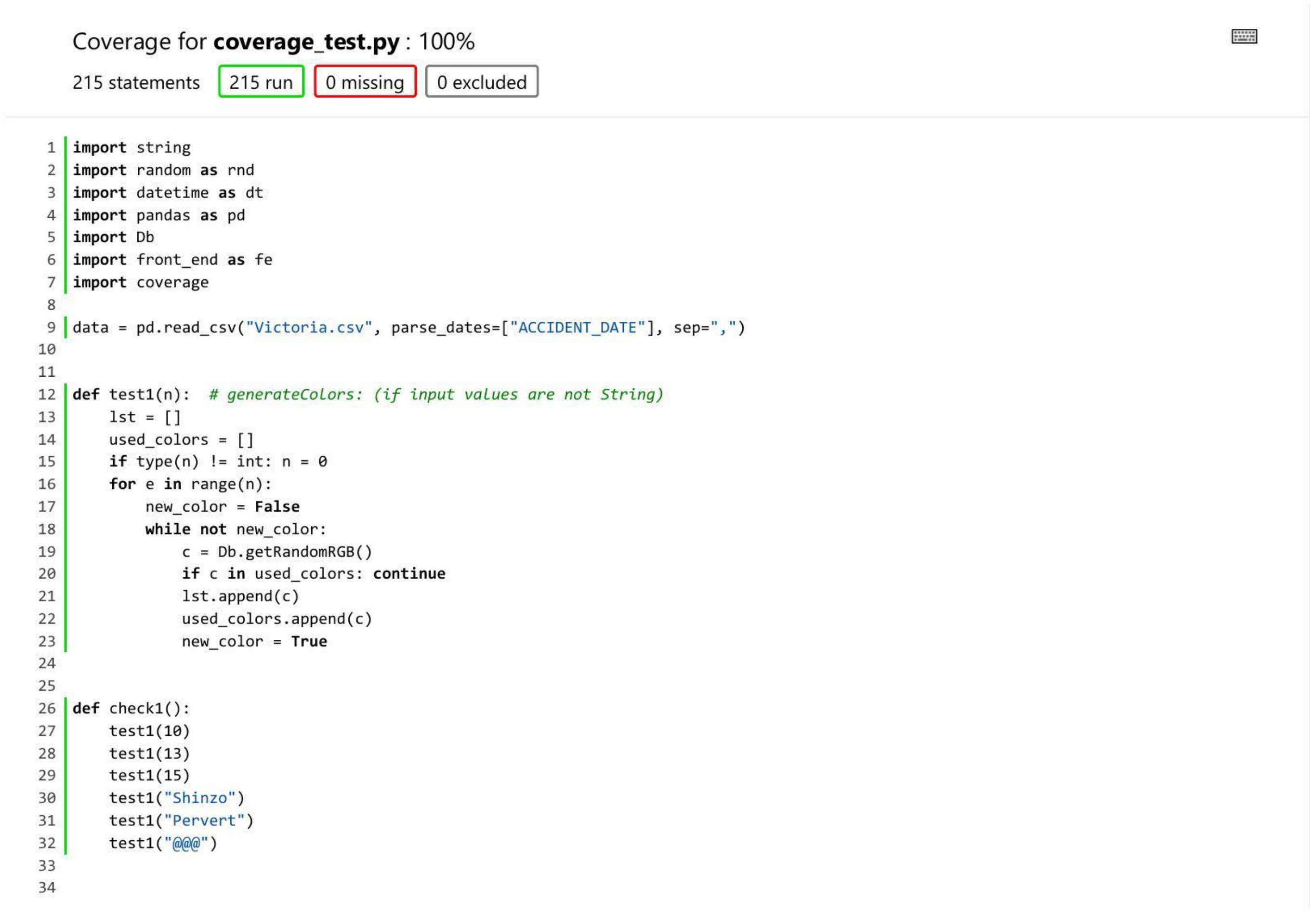
In unit tests, each function with parameter/s will be tested by intentionally causing errors and see what the actual results would be relative to expected results. Function codes are modified just for testing purposes because most of the functions are getting input values directly from GUI such as Date Picker and it makes it impossible to test with initial codes.

No	Test Case	Expected Results	Actual Results
1.0	Generate Color Function. n is string instead of integer	Return Empty List	Returned Empty List
2.0	getNumberOfDays. date_from is later than date_to	Return 0	Returned 0
2.1	getNumberOfDays. date format is wrong	Return 0	Returned 0
3.0	queryTime. classification column name doesn’t exist	Raise error	Raised error. Nothing else done.
4.0	periodByAccident. show_all is an integer instead of boolean	show_all considered as True	show_all was considered as True
5.0	getNumberOfAccidentsInHours. DataFrame is empty	Dictionary still created but all values are 0	Dictionary was still created and all values were 0
6.0	visualize. value is a string instead of VisualizeChart class instance	print error	error printed
7.0	getAccidentNumberOfAlcohol. The user-selected period does not have any matching information.	Return the Error message (ValueError)	Return the Error message (ValueError)

8.0	**ButtonPressed. If there is no matching information with the keyword**	**Shows empty chart**	**Shows empty chart**
8.1	**ButtonPressed** **Pressing search button with no matching the keyword**	**data.empty = True**	**data.empty = True**
8.2	**ButtonPressed** **Visualize button pressed without selecting visualization option.**	**Shows the error message “Please select the visualization option first!”**	**Shows the error message “Please select the visualization option first!”**

Unit-test code is included in the submission

2.0 Coverage Report



Coverage for **coverage_test.py** : 100%

215 statements 215 run 0 missing 0 excluded

```
import string
import random as rnd
import datetime as dt
import pandas as pd
import Db
import front_end as fe
import coverage

data = pd.read_csv("Victoria.csv", parse_dates=["ACCIDENT_DATE"], sep=",")


def test1(n):  # generateColors: (if input values are not String)
    lst = []
    used_colors = []
    if type(n) != int: n = 0
    for e in range(n):
        new_color = False
        while not new_color:
            c = Db.getRandomRGB()
            if c in used_colors: continue
            lst.append(c)
            used_colors.append(c)
            new_color = True


def check1():
    test1(10)
    test1(13)
    test1(15)
    test1("Shinzo")
    test1("Pervert")
    test1("@@@")


```

```python
def test2(date_from, date_to):  # getNumberOfDays (if date_from is bigger than date_to)
    try:
        d_f = date_from.split("-")
        d_t = date_to.split("-")
        if int("".join(d_f)) >= int("".join(d_t)):
            return_val = 0
        else:
            a = dt.date(int(d_f[0]), int(d_f[1]), int(d_f[2]))
            b = dt.date(int(d_t[0]), int(d_t[1]), int(d_t[2]))
            return_val = (b - a).days
    except Exception as e:
        print("ERROR:", e)
        return_val = 0


def check2():
    test2("2021-02-01", "2020-05-23")
    test2("2016-10-14", "2017-05-23")
    test2("2021-02-01", "2020-05-23")
    test2("20214-02-01", "20202-05-23")
    test2(12345, 22222)


def test3(date_from, date_to):  # getNumberOfDays (if user input date in wrong format)
    try:
        d_f = date_from.split("-")
        d_t = date_to.split("-")
        if int("".join(d_f)) >= int("".join(d_t)):
            return_val = 0
        else:
            a = dt.date(int(d_f[0]), int(d_f[1]), int(d_f[2]))
            b = dt.date(int(d_t[0]), int(d_t[1]), int(d_t[2]))
            return_val = (b - a).days
    except Exception as e:
        print("ERROR:", e)
        return_val = 0


def check3():
    test3("2019-05-22", "2020-07-28")
    test3("2025-05-22", "2020-07-28")
    test3("2019/05/22", "2020/07/28")
    test3("20190522", "20200728")
    test3("20194-05-22", "20204-07-28")
    test3("hello", "error")


def test4(col, cdn=''):  # queryTime. (If the classfication column does not exist)
    try:
        q = {
            1: "2021-02-01",
            2: "2022-05-23",
            0: '@q[1] <= ACCIDENT_DATE <= @q[2]'}
        cols = ['ACCIDENT_DATE', 'ACCIDENT_TYPE', 'REGION_NAME', 'ALCOHOLTIME', 'ACCIDENT_TIME']
        condition = cdn
        classification = col

        if classification not in cols and condition != '': cols.append(classification)
        d = data.query(q[0])[cols]
        if condition == '':
            pass  # return d
        else:
            d = d[d[classification].str.contains(pat=condition, na=False, case=False)]
    except Exception as e:
        print("ERROR!:", e)
        error_message = e


def check4():
    test4('ACCIDENT_RANDOM')
    test4(True)
    test4(1234, 'Hoi')
    test4("Empty_Column")
    test4("ACCIDENT_TYPE", 'Hi')
    test4("Error")


def test5(show):  # periodByAccident (if value show is not a boolean datatype)
    condition = ''
    classification = 'ACCIDENT_TYPE'
    show_all = show
    if show_all:
        col_names = list(data.columns)
    else:
        col_names = ['ACCIDENT_DATE', 'ACCIDENT_TYPE', 'REGION_NAME', 'ALCOHOLTIME', 'ACCIDENT_TIME']
    d = Db.queryTime({
        1: "2021-02-01",
        2: "2022-05-23",
        0: '@q[1] <= ACCIDENT_DATE <= @q[2]'
    }, col_names, condition, classification)


def check5():
    test5(False)
    test5(123)
    test5(555)
    test5("Hello")


def test6(date_from, date_to):  # getNumberOfAccidentsInHours (if there is no matching information, it will return the empty empty tuple
    try:
        a = date_from
        b = date_to
        d = Db.queryTime({
            1: a,
            2: b,
            0: '@q[1] <= ACCIDENT_DATE <= @q[2]'
        }, ['ACCIDENT_TIME'], '', 'ACCIDENT_TYPE')
        reference = d['ACCIDENT_TIME'].to_dict()
        output = {}  # output dictionary
        num_days = Db.getNumberOfDays(a, b)
        for i in range(24): output[i] = 0  # initialization
        for row in reference: output[int(reference[row][:2])] += 1
        for e in output:
            output[e] = round(output[e] / num_days, 3)
        x = list(output.keys())
        y = list(output.values())
    except Exception as e:
        print("ERROR!:", e)

```

```python

def check6():
    test6("2021-05-21", "2022-05-01")
    test6("2015-01-01", "2016-01-02")
    test6("2022-03-20", "2023-04-15")
    test6("2016-05-16", "2017-01-01")
    test6(True, False)
    test6(2015, 2018)


def test7(val):  # visualize (if user input wrong visualization option, it will print Error)
    values = val
    if type(values) == list or type(values) == Db.VisualizeChart:
        pass
    else:
        e = "Error! Type list or VisualizeChart is expected!"


def check7():
    test7("pie")
    test7("error")
    test7("line")
    test7(123)
    test7("Hello")
    test7(Db.VisualizeChart([['Hello', 'Good Bye', 'Good Night', 'Good Day', 'See You'], [1, 2, 2, 1, 3]]))


def test8(data_from, data_to):  # getAccidentNumberOfAlcohol.(if there is no matching information, it will return the empty dataframe)
    try:
        a = data_from
        b = data_to
        d = Db.queryTime({
            1: a,
            2: b,
            0: '@q[1] <= ACCIDENT_DATE <= @q[2]'
        }, ['SEVERITY', 'ALCOHOLTIME'], '', 'ACCIDENT_TYPE')

        alcohol = sorted(d['ALCOHOLTIME'].values)
        severity = d.groupby(['SEVERITY', 'ALCOHOLTIME']).groups
        keys = [('Fatal accident', 'No'), ('Fatal accident', 'Yes'), ('Serious injury accident', 'No'),
                ('Serious injury accident', 'Yes')]
        for k in keys:
            if k not in severity:
                severity[k] = []
        output = {
            'Alcohol (A)': len(alcohol) - alcohol.index('Yes'),
            'Non-Alcohol (B)': alcohol.index('Yes'),
            'Serious A': len(severity['Serious injury accident', 'Yes']),
            'Serious B': len(severity['Serious injury accident', 'No']),
            'Fatal A': len(severity['Fatal accident', 'Yes']),
            'Fatal B': len(severity['Fatal accident', 'No'])
        }  # output dictionary
    except Exception as e:
        print("ERROR!:", e)
        error_message = e


def check8():
    test8("2014-05-15", "2015")
    test8("2014", "2015")
    test8("2014", "2015-04-13")
    test8("", "")
    test8("2020-01-01", "2022-03-20")
    test8(2013, 2014)


def test9(date_f, date_t, key):  # button pressed (Even user input the wrong information, program will not be broken)
    date_from = fe.format_date(date_f)
    date_to = fe.format_date(date_t)
    keyword = str(key)
    classification = 'ACCIDENT_TYPE'
    fromCheck = int(date_from.replace('-', ''))
    toCheck = int(date_to.replace('-', ''))
    # Check if date to is behind date from
    if fromCheck >= toCheck:
        print("ERROR!: Date to can't be behind date from")  # fe.wx.MessageBox("date to can't be behind date from", "Checker")
    else:
        result = Db.VisualizeChart(
            values=Db.getNumberOfAccidentsInHours(date_from, date_to, keyword, classification),
            title="Average number of accidents in each hour of the day",
            xlabel="Each hour",
            ylabel="Average number of accidents",
            dtype='pie'
        )


def check9():
    test9(dt.date(2020, 9, 2), dt.date(2021, 9, 2), "ajsndjsnjd")
    test9(dt.date(2020, 9, 2), dt.date(2021, 9, 2), "Collision")
    test9(dt.date(2020, 9, 2), dt.date(2021, 9, 2), True)
    test9(dt.date(2017, 9, 2), dt.date(2018, 9, 2), "sydney")
    test9(dt.date(2020, 9, 2), dt.date(2021, 9, 2), "Brisbane")
    test9(dt.date(2022, 9, 2), dt.date(2021, 9, 2), 123421)


def test10(key):  # buttonPressed (if there is no matching information after pressing search button, it will return the empty dataframe)
    date_from = fe.format_date(dt.date(2019, 9, 2))
    date_to = fe.format_date(dt.date(2021, 9, 2))
    keyword = str(key)
    classification = 'ACCIDENT_TYPE'
    show_all = False
    data = Db.periodByAccident(date_from, date_to, keyword, classification, show_all)


def check10():
    test10("ufhiqbwe")
    test10("Pedestrian")
    test10("Tokyo")
    test10(False)
    test10(1252932)
    test10("Seoul")


def test11(selection):  # ButtonPressed, (it will always check the visualization option)
    date_from = fe.format_date(dt.date(2019, 9, 2))
    date_to = fe.format_date(dt.date(2021, 9, 2))
    keyword = "collision"
    classification = 'ACCIDENT_TYPE'
    user_selection = selection
    if user_selection == 'alcohol':
        pass
    elif user_selection in ['line', 'pie', 'bar']:
        pass
    else:
        error_message = "Please select the visualization option first!"


def check11():
    test11("line")
    test11("Brisbane")
    test11(1234)
    test11("alcohol")
    test11(9898)
    test11(True)


check1()
check2()
check3()
check4()
check5()
check6()
check7()
check8()
check9()
check10()
check11()
```

A description of the coverage of your unit tests, including how you evaluated coverage (function, statement, branch, condition)

Test1: Not much edited

Test2: To make the program work in any situation, we used try: and except: to get the error. When error happens, the function prints out the error and returns 0

Test3: Same as test2

Test4: To make the program work in any situation, we used try: and except: to get the error. When an error happens, this function prints out the error and returns an empty DataFrame.

Test5: show_all is tested here. Despite any circumstance, this function will work as python is flexible. If the string is fetched, it is True. For an integer, it checks 0 or 1.

Test6: To make the program work in any situation, we used try: and except: to get the error. When an error happens, this prints out the error and return a tuple with two empty strings.

Test7: visualize function is tested here. In order to check if the users input visualization option is in the visualization option, if condition is being used.

Test8: getAccidentNumberOfAlcohol function is tested. if the data type of the selected period is not a string or the selected period is in the wrong format, it will print the error message.

Test9: To make the program work in any situation, the keyword is always converted to str, and the function will only work when the date_from is behind date_to. Dates should always be date due to the GUI.

Test10: buttonPressed function is tested and if there is no matching information with keywords from users, the function will output empty dataframe.

Test11: buttonPressed function is tested. The difference between selection is what we want to see. selection is always converted to str and the if and else statement will check the selection.

3.0 Requirements Acceptance Testing

Software Requirement No	Test	Implemented (Full /Partial/ None)	Test Results (Pass/ Fail)	Comments (for partial implementation or failed test results)
1	search function. Show lists, return None.	Implemented	Pass	N/A
2	showTable function. Show DataFrame as table.	Implemented	Pass	N/A
3	visualize function. Chart will be shown. Gets the same data as the search function.	Implemented	Pass	N/A
4	printEmpty function. This happens when error occurs.	Partial	Pass	This function was not implemented as a function itself but implemented partially inside functions.
5	reset function. Resets the visualized table.	Implemented	Pass	N/A
6	destruct function from chartWindow class. Destructs the window.	None	N/A	To allow users to manage individual chart windows, the destruct function which deletes all windows at once was not implemented.