Software Testing Report

Victoria State Accident Data Analysis Tool

Jigme Yeshey

Harmol Kaur

Table of Contents

[1. Unit Tests 4](#_Toc147905755)

[2. Unit Test Code 6](#_Toc147905756)

[1.1 Test data file 6](#_Toc147905757)

[1.2 Test data format for selection 6](#_Toc147905758)

[1.3 Test retrieval of data with a specified parameter 7](#_Toc147905759)

[2.1 Test data file 8](#_Toc147905760)

[2.2 Test retrieval of data with specified parameter 8](#_Toc147905761)

[2.3 Test data retrieval for chart mapping 9](#_Toc147905762)

[3.1 Test data file 10](#_Toc147905763)

[3.2 Test retrieval of data with specified parameter 11](#_Toc147905764)

[3.3 Test data retrieved for chart mapping 12](#_Toc147905765)

[4.1 Test data file 13](#_Toc147905766)

[4.2 Test retrieval of data with specified parameter 13](#_Toc147905767)

[4.3 Test retrieval of data with test parameters 14](#_Toc147905768)

[5.1 Test data file 16](#_Toc147905769)

[5.2 Test retrieval of data with specified parameter 16](#_Toc147905770)

[3. Coverage Report 18](#_Toc147905771)

[4. Requirements Acceptance Testing 19](#_Toc147905772)

# Unit Tests

| **No** | **Test Case** | **Expected Results** | **Actual Results** |
| --- | --- | --- | --- |
| **1.0** | **Retrieving Accident Records in a Specified Date Range** | | |
| 1.1 | Test data file | Data file is correctly retrieved. | Data file is correctly retrieved. |
| 1.2 | Test data format for selection | Date from data file is correctly formatted. | Date from data file is correctly formatted. |
| 1.3 | Test retrieval of data with a specified parameter. | Data within the specified date ranges is retrieved correctly. | Data within the specified range is retrieved correctly. |
| **2.0** | **Showing Accident Trends during Alcohol Time** | | |
| 2.1 | Test data file | Data file is correctly retrieved. | Data file is correctly retrieved. |
| 2.2 | Test retrieval of data with a specified parameter. | Data for accidents during alcohol time is retrieved correctly. | All records for accidents during alcohol time were retrieved. |
| 2.3 | Test data retrieved for chart mapping | The retrieved data is pertinent to alcohol time. | All accident records were associated with alcohol time. |
| **3.0** | **Illustrating Accidents on a Specific Hour** | | |
| 3.1 | Test data file | Data file is correctly retrieved. | Data file is correctly retrieved. |
| 3.2 | Test retrieval of data with a specified parameter. | Hours from each record are in required format. | Hours from each record are extracted properly and in correct format. |
| 3.3 | Test data retrieved for chart mapping | The retrieved data is pertinent to selected hour. | All retrieved accidents occurred on the selected hour. |
| **4.0** | **Retrieving Accident Records by Accident Type** | | |
| 4.1 | Test data file | Data file is correctly retrieved. | Data file is correctly retrieved. |
| 4.2 | Test retrieval of data with a specified parameter. | All data should be retrieved by mentioning all accident types. | All records were retrieved. |
| 4.3 | Test retrieval of data with test parameters. | The retrieved data is pertinent to selected accident type. | All retrieved accidents show |
| **5.0** | **Illustrating Accident Type Frequencies over a Time Period** | | |
| 5.1 | Test data file | Data file is correctly retrieved. | Data file is correctly retrieved. |
| 5.2 | Test data retrieved for chart mapping | The retrieved data is pertinent to selected hour. | All retrieved accidents occurred on the selected hour. |

# Unit Test Code

### 1.1 Test data file

import csv  
  
def read\_csv\_file(file\_path):  
 data = []  
 with open(file\_path, "r") as file:  
 reader = csv.reader(file)  
 for row in reader:  
 data.append(row)  
 return data  
  
def test\_read\_victoria\_accident\_csv():  
 file = "victoria\_accident.csv"  
 data = read\_csv\_file(file)  
 assert len(data) > 0

### 1.2 Test data format for selection

import os  
import csv  
import re  
import csv  
  
def read\_csv\_file(file\_path):  
 data = []  
 if os.path.exists(file\_path):  
 with open(file\_path, "r") as file:  
 reader = csv.reader(file)  
 next(reader)  
 for row in reader:  
 data.append(row)  
 return data  
  
def test\_formatted\_date():  
 file\_name = "victoria\_accident.csv"  
  
 data = read\_csv\_file(file\_name)  
  
 for row in data:  
 date = row[4]  
 date\_components = date.split('/')  
 if len(date\_components) == 3:  
 formatted\_date = f"{date\_components[1]}/{date\_components[2]}"  
 assert re.match(r'\d{1,2}/\d{4}', formatted\_date) is not None

### 1.3 Test retrieval of data with a specified parameter

import os  
import csv  
import re  
  
def retrieve\_data(selected\_start\_date, selected\_end\_date):  
  
 start\_date = int(selected\_start\_date.split('/')[1] + selected\_start\_date.split('/')[0])  
 end\_date = int(selected\_end\_date.split('/')[1] + selected\_end\_date.split('/')[0])  
  
 selected\_data = []  
  
 with open("victoria\_accident.csv", "r") as file:  
 reader = csv.reader(file)  
 headers = next(reader)  
  
 for row in reader:  
  
 date\_parts = row[4].split('/')  
  
 date\_value = int(date\_parts[2] + date\_parts[1])  
  
 if start\_date <= date\_value <= end\_date:  
 selected\_data.append(row)  
  
 return selected\_data  
  
def test\_retrieve\_data():  
 selected\_start\_date = "7/2013"  
 selected\_end\_date = "8/2013"  
  
 selected\_data = retrieve\_data(selected\_start\_date, selected\_end\_date)  
  
 assert len(selected\_data) > 0

### 2.1 Test data file

import csv  
  
def read\_csv\_file(file\_path):  
 data = []  
 with open(file\_path, "r") as file:  
 reader = csv.reader(file)  
 for row in reader:  
 data.append(row)  
 return data  
  
def test\_read\_victoria\_accident\_csv():  
 file = "victoria\_accident.csv"  
 data = read\_csv\_file(file)  
 assert len(data) > 0

### 2.2 Test retrieval of data with specified parameter

import os  
import csv  
import re  
  
def retrieve\_data():  
  
 selected\_data = []  
  
 with open("victoria\_accident.csv", "r") as file:  
 reader = csv.reader(file)  
 headers = next(reader)  
  
 for row in reader:  
 date\_parts = row[4].split('/')  
  
 date\_value = int(date\_parts[2] + date\_parts[1])  
  
 if row[6] == 'Yes':  
 selected\_data.append(row)  
  
 return selected\_data  
  
def test\_alcohol\_time():  
  
 selected\_data = retrieve\_data()  
  
 for data in selected\_data:  
 assert data[6] == 'Yes'

### 2.3 Test data retrieval for chart mapping

import os  
import csv  
import re  
  
def retrieve\_data(selected\_start\_date, selected\_end\_date):  
  
 selected\_start\_date = "07/2013"  
 selected\_end\_date = "08/2013"  
  
 start\_date = int(selected\_start\_date.split('/')[1] + selected\_start\_date.split('/')[0])  
 end\_date = int(selected\_end\_date.split('/')[1] + selected\_end\_date.split('/')[0])  
  
 selected\_data = []  
  
 with open("victoria\_accident.csv", "r") as file:  
 reader = csv.reader(file)  
 next(reader)  
  
 for row in reader:  
 date\_parts = row[4].split('/')  
  
 date\_value = int(date\_parts[2] + date\_parts[1])  
  
 if row[6] == 'Yes' and start\_date <= date\_value <= end\_date:  
 selected\_data.append(row)  
  
 date\_counts = {}  
  
 for row in selected\_data:  
 date = row[4]  
 if date in date\_counts:  
 date\_counts[date] += 1  
 else:  
 date\_counts[date] = 1  
  
 sorted\_dates = sorted(date\_counts.keys(), key=lambda x: (datetime.datetime.strptime(x, "%d/%m/%Y").year,  
 datetime.datetime.strptime(x,  
 "%d/%m/%Y").month))  
  
 return sorted\_dates  
  
def test\_alcohol\_chart():  
  
 selected\_start\_date = "07/2013"  
 selected\_end\_date = "08/2013"  
  
 sorted\_dates = retrieve\_data(selected\_start\_date, selected\_end\_date)  
  
 for date in sorted\_dates:  
 matching\_rows = [row for row in selected\_data if row[4] == date] # Find rows with the same date  
 assert all(row[6] == 'Yes' for row in matching\_rows), f"Alcohol time is not 'Yes' for date {date}"

### 3.1 Test data file

import csv  
  
def read\_csv\_file(file\_path):  
 data = []  
 with open(file\_path, "r") as file:  
 reader = csv.reader(file)  
 for row in reader:  
 data.append(row)  
 return data  
  
def test\_read\_victoria\_accident\_csv():  
 file = "victoria\_accident.csv"  
 data = read\_csv\_file(file)  
 assert len(data) > 0

### 3.2 Test retrieval of data with specified parameter

import os  
import csv  
import re  
  
def accident\_hour():  
 hours = []  
 with open("victoria\_accident.csv", "r") as file:  
 reader = csv.reader(file)  
 headers = next(reader)  
  
 for row in reader:  
 hour\_value = row[5].split('.')[0]  
 hours.append(hour\_value)  
  
 return hours  
  
def test\_accident\_hour():  
 hours = accident\_hour()  
 valid\_hour\_values = ['00', '01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23']  
  
 for hour in hours:  
 assert hour in valid\_hour\_values

### 3.3 Test data retrieved for chart mapping

import os  
import csv  
import re  
  
def retrieve\_data(selected\_start\_date, selected\_end\_date, hour):  
  
 start\_date = int(selected\_start\_date.split('/')[1] + selected\_start\_date.split('/')[0])  
 end\_date = int(selected\_end\_date.split('/')[1] + selected\_end\_date.split('/')[0])  
  
 selected\_data = []  
  
 with open("victoria\_accident.csv", "r") as file:  
 reader = csv.reader(file)  
 next(reader)  
  
 for row in reader:  
 hour\_value = row[5].split('.')[0]  
 date\_parts = row[4].split('/')  
  
 date\_value = int(date\_parts[2] + date\_parts[1])  
  
 if hour\_value == hour and start\_date <= date\_value <= end\_date:  
 selected\_data.append(row)  
  
 date\_hour = {}  
  
 for row in selected\_data:  
  
 key = row[4]  
  
 if key in date\_hour:  
 date\_hour[key] += 1  
 else:  
 date\_hour[key] = 1  
  
 sorted\_dates = sorted(date\_hour.keys(), key=lambda x: (int(x.split('/')[2]), int(x.split('/')[1]), int(x.split('/')[0])))  
  
 return sorted\_dates  
  
def test\_alcohol\_chart():  
  
 selected\_start\_date = "07/2013"  
 selected\_end\_date = "08/2013"  
 hour = '03'  
  
 sorted\_dates = retrieve\_data(selected\_start\_date, selected\_end\_date, hour)  
  
 for date in sorted\_dates:  
 matching\_rows = [row for row in retrieve\_data(selected\_start\_date, selected\_end\_date, hour) if row[4] == date]  
 assert all(row[5].split('.')[0] == '03' for row in matching\_rows), f"Hour is not '03' for date {date}"

### 4.1 Test data file

import csv  
  
def read\_csv\_file(file\_path):  
 data = []  
 with open(file\_path, "r") as file:  
 reader = csv.reader(file)  
 for row in reader:  
 data.append(row)  
 return data  
  
def test\_read\_victoria\_accident\_csv():  
 file = "victoria\_accident.csv"  
 data = read\_csv\_file(file)  
 assert len(data) > 0

### 4.2 Test retrieval of data with specified parameter

import os  
import csv  
import re  
  
def accident\_type():  
  
 types = []  
  
 with open("victoria\_accident.csv", "r") as file:  
 reader = csv.reader(file)  
 headers = next(reader)  
  
 for row in reader:  
 accident\_type = row[7]  
 types.append(accident\_type)  
  
 return(types)  
  
def test\_accident\_type():  
  
 with open("victoria\_accident.csv", "r") as file:  
 reader = csv.reader(file)  
 headers = next(reader)  
  
 types = accident\_type()  
  
 for type in types:  
  
 assert type in ['Struck Pedestrian', 'Collision with vehicle', 'Collision with a fixed object', 'No collision and no object struck', 'Struck animal', 'Vehicle overturned (no collision)', 'collision with some other object', 'Fall from or in moving vehicle', 'Other accident']

### 4.3 Test retrieval of data with test parameters

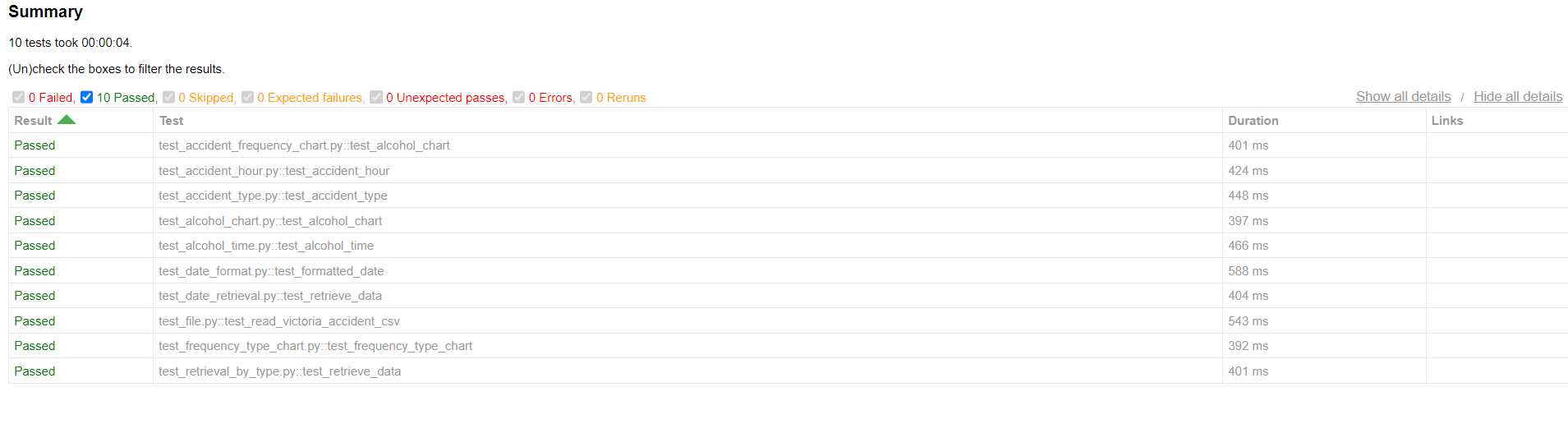
import os  
import csv  
import re  
  
def retrieve\_data(selected\_start\_date, selected\_end\_date):  
  
 start\_date = int(selected\_start\_date.split('/')[1] + selected\_start\_date.split('/')[0])  
 end\_date = int(selected\_end\_date.split('/')[1] + selected\_end\_date.split('/')[0])  
  
 selected\_data = []  
  
 with open("victoria\_accident.csv", "r") as file:  
 reader = csv.reader(file)  
 headers = next(reader)  
  
 for row in reader:  
 date\_parts = row[4].split('/')  
  
 date\_value = int(date\_parts[2] + date\_parts[1])  
  
 if start\_date <= date\_value <= end\_date:  
 selected\_data.append(row)  
  
 type\_counts = {}  
  
 for row in selected\_data:  
 type = row[7]  
 if type in type\_counts:  
 type\_counts[type] += 1  
 else:  
 type\_counts[type] = 1  
  
 return type\_counts  
  
def test\_frequency\_type\_chart():  
  
 selected\_start\_date = "07/2013"  
 selected\_end\_date = "08/2013"  
  
 type\_counts = retrieve\_data(selected\_start\_date, selected\_end\_date)  
  
 valid\_types = ['Struck Pedestrian', 'Collision with vehicle', 'Collision with a fixed object', 'No collision and no object struck', 'Struck animal', 'Vehicle overturned (no collision)', 'Collision with some other object', 'Fall from or in moving vehicle', 'Other accident']  
  
 for type in type\_counts:  
 assert type in valid\_types

### 5.1 Test data file

import csv  
  
def read\_csv\_file(file\_path):  
 data = []  
 with open(file\_path, "r") as file:  
 reader = csv.reader(file)  
 for row in reader:  
 data.append(row)  
 return data  
  
def test\_read\_victoria\_accident\_csv():  
 file = "victoria\_accident.csv"  
 data = read\_csv\_file(file)  
 assert len(data) > 0

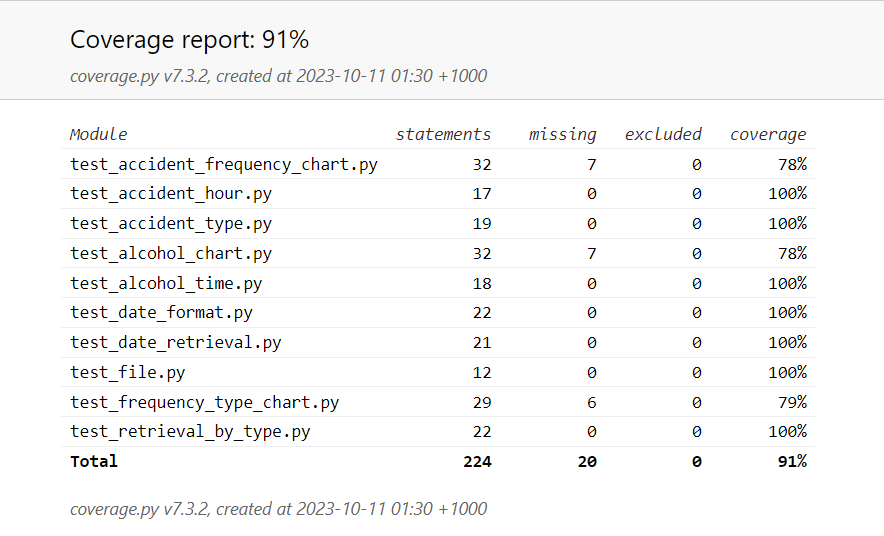
### 5.2 Test retrieval of data with specified parameter

import os  
import csv  
import re  
  
def retrieve\_data(selected\_start\_date, selected\_end\_date, accident\_type):  
 start\_date = int(selected\_start\_date.split('/')[1] + selected\_start\_date.split('/')[0])  
 end\_date = int(selected\_end\_date.split('/')[1] + selected\_end\_date.split('/')[0])  
  
 selected\_data = []  
  
 with open("victoria\_accident.csv", "r") as file:  
 reader = csv.reader(file)  
 headers = next(reader)  
  
 for row in reader:  
  
 date\_parts = row[4].split('/')  
  
 date\_value = int(date\_parts[2] + date\_parts[1])  
  
 if row[7] == accident\_type and start\_date <= date\_value <= end\_date:  
 selected\_data.append(row)  
  
 return selected\_data  
  
  
def test\_retrieve\_data():  
 selected\_start\_date = "7/2013"  
 selected\_end\_date = "8/2013"  
 accident\_type = 'Collision with vehicle'  
  
 selected\_data = retrieve\_data(selected\_start\_date, selected\_end\_date, accident\_type)  
  
 assert len(selected\_data) > 0



The test report above shows that all tests have run without any errors.

# Coverage Report



1. **Function Coverage**

Function coverage focuses on whether all functions have been executed. The coverage report does not show a coverage percent of 100% due to specific files. It is essential to note that all functions were executed without errors. Hence, the discrepancy in coverage likely arises from factors unrelated to the execution of functions. There is 100% function coverage.

1. **Statement Coverage**

Statement coverage measures whether each like of code in a program has been executed while testing. It checks whether the lines of code have been executed at least once. After careful analysis of the coverage discrepancy, it was found out that all lines of code have been executed more than one time. Hence, there is no problem with statement coverage.

1. **Branch Coverage**

Branch coverage focuses on decision points where the flow of code can branch out based on specified conditions. The coverage report shows that discrepancies are within the “if” statements. The coverage issue could have been caused because tests for scenarios where the conditions of “if” statements were not met have been examined properly. Hence, proper testing should be conducted to cover test cases for both true and fale conditions within the decision point.

1. **Condition Coverage**

Condition coverage pertains to all possible outcomes of conditions. It ensures that all possible outcomes of conditions are tested. The coverage discrepancy could have occurred because outcomes that did not fulfill the conditions were not adequately tested. The test cases could have overlooked these scenarios leading to untested behaviours.

# Requirements Acceptance Testing

| **Software  Requirement No** | **Test** | **Implemented (Full /Partial/ None)** | **Test Results (Pass/ Fail)** | **Comments (for partial implementation or failed test results)** |
| --- | --- | --- | --- | --- |
| 1 | The tool shall allow users to select a specific location for analysis. | None |  |  |
| 2 | Users shall be able to specify the time period for the analysis. | Full | Pass |  |
| 3 | User shall be able to apply filters for weather conditions, road types, vehicle types, and accident severity. | Partial | Pass | Data can be filtered by accident types and hour of accident but not with the parameters specified in the requirements. |
| 4 | The tool shall provide interactive maps displaying accident locations, hotspots, clusters with the chosen location and time frame. | None |  |  |
| 5 | Users shall be able to zoom in and out of the maps for detailed analysis. | None |  |  |
| 6 | Users shall have the option to click on accident markers to view detailed information about each accident. | None |  |  |
| 7 | The tools shall be able to produce the charts based on the average number of accidents that happen over the selected period. | Full | Pass |  |
| 8 | Users should be able to adjust the time granularity for the time series plot. | Partial | Pass | Users can view different results for different date parameters. |
| 9 | The tool shall enable users to compare accident data across different factors, such as, urban vs rural areas, time of the day, weekday, or weekend, licensed or unlicensed. | Partial | Pass | Users can compare accidents by their type. |
| 10 | Users will be able to analyse the effects of alcohol time on rate of accidents on daily basis. | Full | Pass |  |
| 11 | The tool shall be able to generate safety recommendations for improving road safety based on the analysis results. | None |  |  |
| 12 | The users shall be able to customise the visualisations by adjusting filters and parameters within each section. | None |  |  |
| 13 | The tool shall provide options to export visualisations and analysis results in various formats, such as PDF and CSV. | None |  |  |
| 14 | The tool’s interface shall be intuitive and easy to use and navigate providing a seamless user experience. | Full | Pass |  |