Software Testing Report

Victoria State Accident

Ben White - s2850345

Liam Preston - s5301986

Thomas Chapman - s5251549

Table of Contents

[1.0 Unit Tests 3](#_Toc49779837)

[2.0 Coverage Report 4](#_Toc49779838)

[3.0 Requirements Acceptance Testing 5](#_Toc49779839)

# Unit Tests

| **No** | **Test Case** | **Expected Results** | **Actual Results** |
| --- | --- | --- | --- |
| **1.0** | **Loading Data from Dataset (General Loading) - df = pd.read\_csv()** | | |
| 1.1 | Loaded data from a valid file path. | Data is successfully loaded into the system and functions flawlessly. | Data is successfully loaded into the system and functions flawlessly. |
| 1.2 | Loaded data from an invalid path. | A message prompts user indicating data failed to load. | An error message occurs stating incorrect file path. |
| 1.3 | Loaded all data into a table from the entire date range. | The data present is within the table | Successful, the data is present within the table. |
| **2.0** | **Display a table showing the data from a specific date range - on\_search\_q1() / update\_grid()** | | |
| 2.1 | Load data successfully from valid file path. | Data is successfully loaded into the system and functions flawlessly. | Data is successfully loaded into the system and functions flawlessly. |
| 2.2 | Load data unsuccessfully from invalid file path. | A message prompts user indicating data failed to load. | An error message occurs stating incorrect file path. |
| 2.3 | Data from selected date period is displayed to user. | The time period selected is presented correctly in the table. | Successful, the data is present within the table. |
| **3.0** | **Display a chart showing average number of accidents per hour - on\_search\_q2** | | |
| 3.1 | Load data successfully from valid file path. | Data is successfully loaded into the system and functions flawlessly. | Data is successfully loaded into the system and functions flawlessly. |
| 3.2 | Load data unsuccessfully from invalid file path. | A message prompts user indicating data failed to load. | An error message occurs stating incorrect file path. |
| 3.3 | Data from the selected date range displays hourly averages for number of accidents within valid date range. | The graph is correctly displayed matching users selected date period. | Successful, the data is present within the graph. |
| **4.0** | **Display table of keyword related accidents - on\_search\_q3** | | |
| 4.1 | Load data successfully from valid file path. | Data is successfully loaded into the system and functions flawlessly. | Data is successfully loaded into the system and functions flawlessly. |
| 4.2 | Load data unsuccessfully from invalid file path. | A message prompts user indicating data failed to load. | An error message occurs stating incorrect file path. |
| 4.3 | Table displays valid keyword related data from the dataset. | The correct table is present matching keyword inputted by the user | Successful, the data is present within the table. |
| **5.0** | **Display a chart of alcohol involved in combination with collision type chart – on\_search\_q4 / generate\_graph** | | |
| 5.1 | Load data successfully from valid file path. | Data is successfully loaded into the system and functions flawlessly. | Data is successfully loaded into the system and functions flawlessly. |
| 5.2 | Load data unsuccessfully from invalid file path. | A message prompts user indicating data failed to load. | An error message occurs stating incorrect file path. |
| 5.3 | Analysis of alcohol and collision type with valid selection | The correct chart is displayed matching the checkbox of whether alcohol was involved or not. | Partially, the data is present within the chart but doesn’t work accurately with DatePickerCtrl, |
| **6.0** | **Display charts of speed related incidents – on\_search\_q5 / generate\_graph\_q5 / generate\_injury\_graph** | | |
| 6.1 | Load data successfully from valid file path. | Data is successfully loaded into the system and functions flawlessly. | Data is successfully loaded into the system and functions flawlessly. |
| 6.2 | Load data unsuccessfully from invalid file path. | A message prompts user indicating data failed to load. | An error message occurs stating incorrect file path. |
| 6.3 | Charts present information selected from the drop-down menu. | The chart displayed matches the speed inputted by the user. | Successful, the data is present within the chart. |

|  |  |  |
| --- | --- | --- |
| **Unit Test Code Examples** | | |
| 1.1 | Load data from csv file | import pytest import pandas as pd   def test\_load\_csv\_file():  file\_path = 'CrashStatisticsVictoria.csv'   # Attempt to read the CSV file  try:  df = pd.read\_csv(file\_path, parse\_dates=['ACCIDENT\_DATE'], dayfirst=True)   # Check if the DataFrame is not empty  assert not df.empty, "DataFrame is empty"   # Check if 'ACCIDENT\_DATE' column is of datetime type  assert pd.api.types.is\_datetime64\_any\_dtype(df['ACCIDENT\_DATE']), "ACCIDENT\_DATE column is not of datetime type"   except FileNotFoundError:  pytest.fail(f"File {file\_path} not found")  except pd.errors.EmptyDataError:  pytest.fail(f"File {file\_path} is empty")  except pd.errors.ParserError:  pytest.fail(f"Error occurred while parsing {file\_path}")   def test\_invalid\_file\_path():  invalid\_file\_path = 'InvalidFilePath.csv'   with pytest.raises(None):  df = pd.read\_csv(invalid\_file\_path) |
| 1.2 | Load invalid data from incorrect csv file | import unittest import pandas as pd  class TestDateParsing(unittest.TestCase):   def test\_valid\_load(self):  file\_path = 'CrashStatisticsVictoria.csv'  df = pd.read\_csv(file\_path, parse\_dates=['ACCIDENT\_DATE'], dayfirst=True)  self.assertIsNotNone(df, "Data should not be None.")  self.assertIsInstance(df, pd.DataFrame, "Data should be a pandas DataFrame.")  self.assertFalse(df.empty, "DataFrame should not be empty.")  self.assertIn ('ACCIDENT\_DATE', df.columns, "ACCIDENT\_DATE column should exist.")   def test\_invalid\_load(self):  file\_path = 'invalidload.csv'  with self.assertRaises(FileNotFoundError):  df = pd.read\_csv(file\_path, parse\_dates=['ACCIDENT\_DATE'], dayfirst=True)  if \_\_name\_\_ == '\_\_main\_\_':  unittest.main() |

|  |  |  |
| --- | --- | --- |
| 2.3 | Display a table showing the data from a specific date range - on\_search\_q1() / update\_grid**()** | import pandas as pd import pytest  class TestClass:  def \_\_init\_\_(self):  pass   def m\_datePicker\_q1\_1(self):  return MockDatePicker('2023-01-01')   def m\_datePicker\_q1\_2(self):  return MockDatePicker('2023-02-01')   def m\_grid\_q1(self):  return MockGrid()   def update\_grid(self, grid, data):  grid.ClearGrid()  if grid.GetNumberRows() > 0:  grid.DeleteRows(0, grid.GetNumberRows(), True)  if grid.GetNumberCols() > 0:  grid.DeleteCols(0, grid.GetNumberCols(), True)  grid.AppendCols(data.shape[1])  grid.AppendRows(data.shape[0])   for col, colname in enumerate(data.columns):  grid.SetColLabelValue(col, colname)  grid.SetColSize(col, 120)  for row, value in enumerate(data[colname]):  grid.SetCellValue(row, col, str(value))   grid.AutoSizeColumns(False)   def on\_search\_q1(self, event, df):  start\_date = self.m\_datePicker\_q1\_1().GetValue().FormatISODate()  end\_date = self.m\_datePicker\_q1\_2().GetValue().FormatISODate()   start\_date = pd.to\_datetime(start\_date)  end\_date = pd.to\_datetime(end\_date)   mask = (pd.to\_datetime(df['ACCIDENT\_DATE']) >= start\_date) & (pd.to\_datetime(df['ACCIDENT\_DATE']) <= end\_date)  filtered\_df = df[mask]   self.update\_grid(self.m\_grid\_q1(), filtered\_df)   class MockDatePicker:  def \_\_init\_\_(self, date):  self.date = date   def GetValue(self):  return self   def FormatISODate(self):  return self.date   class MockGrid:  def ClearGrid(self):  pass   def GetNumberRows(self):  return 0   def DeleteRows(self, startRow, numRows, updateLabels):  pass   def GetNumberCols(self):  return 0   def DeleteCols(self, startCol, numCols, updateLabels):  pass   def AppendCols(self, numCols):  pass   def AppendRows(self, numRows):  pass   def SetColLabelValue(self, col, label):  pass   def SetColSize(self, col, size):  pass   def SetCellValue(self, row, col, value):  pass   def AutoSizeColumns(self, enable):  pass   def test\_on\_search\_q1():  test\_instance = TestClass()   df = pd.DataFrame({'ACCIDENT\_DATE': ['2023-01-15', '2023-01-20', '2023-02-10']})   event = None  test\_instance.on\_search\_q1(event, df)   if \_\_name\_\_ == '\_\_main\_\_':  pytest.main() |

|  |  |  |
| --- | --- | --- |
| 3.3 | Display a chart showing average number of accidents per hour - on\_search\_q2 | import pandas as pd import matplotlib.pyplot as plt import wx import pytest import math from matplotlib.backends.backend\_wxagg import FigureCanvasWxAgg as FigureCanvas  app = wx.App(False)  class TestClass:  def \_\_init\_\_(self):  pass   def m\_datePicker\_q2\_1(self):  return MockDatePicker('2023-01-01')   def m\_datePicker\_q2\_2(self):  return MockDatePicker('2023-02-01')   def m\_panel\_q2\_graph(self):  return wx.Panel(wx.Frame(None, wx.ID\_ANY, "Temp Frame"), wx.ID\_ANY)   def on\_search\_q2(self, event, df):  start\_date = self.m\_datePicker\_q2\_1().GetValue().FormatISODate()  end\_date = self.m\_datePicker\_q2\_2().GetValue().FormatISODate()   start\_date = pd.to\_datetime(start\_date)  end\_date = pd.to\_datetime(end\_date)   mask = (pd.to\_datetime(df['ACCIDENT\_DATE']) >= start\_date) & (pd.to\_datetime(df['ACCIDENT\_DATE']) <= end\_date)  filtered\_df = df[mask].copy()   filtered\_df['HOUR'] = pd.to\_datetime(filtered\_df['ACCIDENT\_TIME'], format='%H.%M.%S').dt.hour  avg\_accidents\_per\_hour = filtered\_df.groupby('HOUR').size() / filtered\_df['ACCIDENT\_DATE'].dt.date.nunique()   fig, ax = plt.subplots(figsize=(5, 3))  avg\_accidents\_per\_hour.plot(kind='bar', ax=ax)  ax.set\_title('Average Number of Accidents per Hour')  ax.set\_xlabel('Hour of the Day')  ax.set\_ylabel('Average Number of Accidents')  ax.set\_xticks(range(24))   if hasattr(self, 'canvas'):  self.canvas.Destroy()  self.canvas = FigureCanvas(self.m\_panel\_q2\_graph(), -1, fig)  sizer = self.m\_panel\_q2\_graph().GetSizer()  if sizer is not None:  sizer.Add(self.canvas, 1, wx.EXPAND)  self.m\_panel\_q2\_graph().Layout()   class MockDatePicker:  def \_\_init\_\_(self, date):  self.date = date   def GetValue(self):  return self   def FormatISODate(self):  return self.date   def test\_on\_search\_q2():  test\_instance = TestClass()   df = pd.DataFrame({  'ACCIDENT\_DATE': pd.to\_datetime(['2023-01-15', '2023-01-20', '2023-02-10']),  'ACCIDENT\_TIME': ['10.30.45', '15.45.30', '09.15.30']  })  event = None   test\_instance.on\_search\_q2(event, df)    if \_\_name\_\_ == '\_\_main\_\_':  pytest.main() |

|  |  |  |
| --- | --- | --- |
| 4.3 | Display table of keyword related accidents - on\_search\_q3 | import pandas as pd import wx import pytest  app = wx.App(False)  class TestClass:  def \_\_init\_\_(self):  pass   def m\_datePicker\_q3\_1(self):  return MockDatePicker('2023-01-01')   def m\_datePicker\_q3\_2(self):  return MockDatePicker('2023-02-01')   def m\_textCtrl\_q3(self):  return MockTextCtrl()   def m\_grid\_q3(self):  return MockGrid()   def on\_search\_q3(self, event, df):  start\_date = self.m\_datePicker\_q3\_1().GetValue().FormatISODate()  end\_date = self.m\_datePicker\_q3\_2().GetValue().FormatISODate()  keyword = self.m\_textCtrl\_q3().GetValue()   mask = (  (df['ACCIDENT\_DATE'] >= start\_date) &  (df['ACCIDENT\_DATE'] <= end\_date) &  df['ACCIDENT\_TYPE'].str.contains(keyword, case=False, na=False)  )  results = df[mask]   self.update\_grid(self.m\_grid\_q3(), results)   def update\_grid(self, grid, data):  grid.ClearGrid()  if grid.GetNumberRows() > 0:  grid.DeleteRows(0, grid.GetNumberRows(), True)  if grid.GetNumberCols() > 0:  grid.DeleteCols(0, grid.GetNumberCols(), True)  grid.AppendCols(data.shape[1])  grid.AppendRows(data.shape[0])   for col, colname in enumerate(data.columns):  grid.SetColLabelValue(col, colname)  grid.SetColSize(col, 120)  for row, value in enumerate(data[colname]):  grid.SetCellValue(row, col, str(value))   grid.AutoSizeColumns(False)  class MockDatePicker:  def \_\_init\_\_(self, date):  self.date = date   def GetValue(self):  return self   def FormatISODate(self):  return self.date  class MockTextCtrl:  def GetValue(self):  return "YourKeyword"  class MockGrid:  def \_\_init\_\_(self):  self.number\_of\_rows = 0  self.number\_of\_cols = 0   def ClearGrid(self):  pass   def DeleteRows(self, \*args, \*\*kwargs):  pass   def DeleteCols(self, \*args, \*\*kwargs):  pass   def AppendCols(self, \*args, \*\*kwargs):  pass   def AppendRows(self, \*args, \*\*kwargs):  pass   def SetColLabelValue(self, \*args, \*\*kwargs):  pass   def SetColSize(self, \*args, \*\*kwargs):  pass   def SetCellValue(self, \*args, \*\*kwargs):  pass   def AutoSizeColumns(self, \*args, \*\*kwargs):  pass   def GetNumberRows(self):  return self.number\_of\_rows   def GetNumberCols(self):  return self.number\_of\_cols  def test\_on\_search\_q3():  test\_instance = TestClass()   df = pd.DataFrame({  'ACCIDENT\_DATE': pd.to\_datetime(['2023-01-15', '2023-01-20', '2023-02-10']),  'ACCIDENT\_TYPE': ['Collision', 'Rollover', 'Collision']  })   event = None   test\_instance.on\_search\_q3(event, df)   if \_\_name\_\_ == '\_\_main\_\_':  pytest.main() |

|  |  |  |
| --- | --- | --- |
| 6.3 | Display charts of speed related incidents – on\_search\_q5 / generate\_graph\_q5 / generate\_injury\_graph | import pytest import pandas as pd import matplotlib.pyplot as plt import wx from matplotlib.backends.backend\_wxagg import FigureCanvasWxAgg as FigureCanvas  class TestClassMock:  def \_\_init\_\_(self, parent):  self.m\_choice1 = wx.Choice(parent, wx.ID\_ANY)  self.frame = parent   def on\_search\_q5(self, event):  global mask  selected\_speed = self.m\_choice1.GetString(self.m\_choice1.GetSelection())   df = pd.DataFrame({  'SPEED\_ZONE': ['40 km/hr', '50 km/hr', '60 km/hr', '70 km/hr', '80 km/hr', '90 km/hr', '100 km/hr',  '110 km/hr'],  'ACCIDENT\_DATE': pd.to\_datetime(['2023-01-15', '2023-01-20', '2023-02-10', '2023-03-15', '2023-04-20', '2023-05-10', '2023-06-15', '2023-07-20'])  })   df['SPEED\_ZONE'] = df['SPEED\_ZONE'].astype(str)   if '40 Km/h' in selected\_speed:  mask = (df['SPEED\_ZONE'] == '40 km/hr')  elif '50 Km/h' in selected\_speed:  mask = (df['SPEED\_ZONE'] == '50 km/hr')  elif '60 Km/h' in selected\_speed:  mask = (df['SPEED\_ZONE'] == '60 km/hr')  elif '70 Km/h' in selected\_speed:  mask = (df['SPEED\_ZONE'] == '70 km/hr')  elif '80 Km/h' in selected\_speed:  mask = (df['SPEED\_ZONE'] == '80 km/hr')  elif '90 Km/h' in selected\_speed:  mask = (df['SPEED\_ZONE'] == '90 km/hr')  elif '100 Km/h' in selected\_speed:  mask = (df['SPEED\_ZONE'] == '100 km/hr')  elif '110 Km/h' in selected\_speed:  mask = (df['SPEED\_ZONE'] == '110 km/hr')   filtered\_data = df[mask]   speeding\_counts = filtered\_data.groupby('ACCIDENT\_DATE').size().reset\_index(name='ACCIDENT\_COUNT')  speeding\_counts['DAY\_OF\_WEEK'] = pd.to\_datetime(speeding\_counts['ACCIDENT\_DATE']).dt.day\_name()   self.generate\_graph\_q5(speeding\_counts)   def generate\_graph\_q5(self, data):  panel\_width, panel\_height = self.frame.GetSize()   fig\_width, fig\_height = panel\_width / 100, panel\_height / 100   accidents\_by\_day = data.groupby('DAY\_OF\_WEEK').agg({'ACCIDENT\_COUNT': 'sum'}).reset\_index()   order\_of\_days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']  accidents\_by\_day['DAY\_OF\_WEEK'] = pd.Categorical(accidents\_by\_day['DAY\_OF\_WEEK'], categories=order\_of\_days,  ordered=True)  accidents\_by\_day = accidents\_by\_day.sort\_values('DAY\_OF\_WEEK')   fig, ax = plt.subplots(figsize=(fig\_width, fig\_height))   ax.bar(accidents\_by\_day['DAY\_OF\_WEEK'], accidents\_by\_day['ACCIDENT\_COUNT'], label='Number of Accidents',  color=['blue', 'orange', 'yellow', 'green', 'pink', 'blue', 'purple'])  ax.set\_title('Total Speed Related Accidents Per Day of Week')  ax.set\_xlabel('Day of the Week', fontsize=8)  ax.set\_ylabel('Number of Accidents', fontsize=8)   ax.tick\_params(axis='x', rotation=45, labelsize=7)   if hasattr(self, 'canvas\_q5\_1'):  self.canvas\_q5\_1.Destroy()   self.canvas\_q5\_1 = FigureCanvas(self.frame, -1, fig)   def test\_on\_search\_q5():  app = wx.App(False)  frame = wx.Frame(None, wx.ID\_ANY)   test\_instance = TestClassMock(frame)   test\_instance.m\_choice1.AppendItems(  ['40 Km/h', '50 Km/h', '60 Km/h', '70 Km/h', '80 Km/h', '90 Km/h', '100 Km/h', '110 Km/h']) |

|  |  |  |
| --- | --- | --- |
| **Further Test Code for GUI** | | |
| 1.1 | Full loading Application Test | import wx import pytest from accident\_analyzer import Frame  @pytest.fixture def wx\_app\_and\_frame():  app = wx.App(False)  frame = wx.Frame(None, wx.ID\_ANY)  return app, frame   def test\_page\_functionality(wx\_app\_and\_frame):  app, frame = wx\_app\_and\_frame   page\_instance = Frame(frame)   frame.Destroy() |

# Coverage Report

A black background with white text

Description automatically generated

Figure 1: Coverage Report

The coverage report shows how much of the code is run during the testing process:

* Function Coverage

The Function Coverage metrics focus on ensuring that each of the functions within the code are executed at least once during the testing process. Although not 100% coverage the application itself and functions all work flawless as designed. Understanding why these are 100% coverage could allow for more efficient code to be produced.

* Statement Coverage

Is about making sure that individual lines or statements within the code are being executed. Most of the files have high coverages besides test\_speed\_accidents however this does not affect its functionality. This means a small portion of the code is being tested.

* Branch Coverage

The reason some of the coverage files have a lower percentage is due to some branches within the files not properly being tested. Such as if and else statements not being properly noted. Examples being making sure a True and False conditions are always both met.

* Condition Coverage

Condition Coverage represents ‘AND’ and ‘OR’ or True and False statements. accident\_analyzer contains missed lines leading to the inadequacies. It is always very important to cater to all Boolean expressions and possible outcomes. However the code still functions without issues but its safe practice to implement fully developed case scenarios.

# Requirements Acceptance Testing

| **Software  Requirement No** | **Test** | **Implemented (Full /Partial/ None)** | **Test Results (Pass/ Fail)** | **Comments (for partial implementation or failed test results)** |
| --- | --- | --- | --- | --- |
| 1.1 | The system shall provide a user-friendly interface for easy navigation and interaction. | Full | Pass | This software requirement function as required |
| 1.2 | The system shall enable users to select specific date ranges for data retrieval. | Full | Pass | This software requirement function as required |
| 1.3 | The system shall display accident details for user-selected periods. | Full | Pass | This software requirement function as required |
| 1.4 | The system shall produce an hourly chart showcasing the average number of accidents for chosen periods. | Full | Pass | This software requirement function as required |
| 1.5 | The system shall allow users to input keywords and filter accident types accordingly. | Full | Pass | This software requirement function as required |
| 1.6 | The system shall offer dedicated analysis tools focusing on alcohol-related accidents. | Full | Pass | This software requirement function as required |
| 1.7 | The software shall provide an "insight tool" to automatically generate notable patterns or trends from the dataset. | Full | Pass | This software requirement function as required |
| 1.8 | The system shall ensure data privacy and prevent unauthorized access. | None | Fail | The system didn’t allow for users to have data privacy or prevent unauthorised access. This would need to be implemented to meet this requirement |
| 1.9 | The system shall support data input in different formats like CSV, Excel, and JSON for flexibility in data sourcing. | Partial | Pass | The system was capable of reading CSV but wasn’t rested with other data formats. |
| 1.10 | The software shall be compatible with various operating systems (e.g., Windows, macOS) to ensure broad user accessibility. | Partial | Pass | The software worked with current Windows and Mac operating systems but wasn’t tested on a full array of operating systems. |
| 1.11 | The system shall provide error messages in case of invalid date ranges or keyword inputs. | None | Fail | No error messages are prompted when an incorrect user selection is made. |
| 1.12 | The system shall ensure that the data presented is updated and synced with the original dataset at regular intervals. | Partial | Pass | The information is retrieved every time the application is used. This means it only refreshes once the software is relaunched. |
| 1.13 | The system shall be optimized for performance to handle large datasets without lag. | Full | Pass | This software requirement function as required |