**White-Box Testing**

**White-Box Testing is one of the ways to analyze how efficient and effective current test module is for the certain project. Unlike black-box method which examines input specification, white-box method examines source code to evaluate test data adequacy and it often refers to percentage coverage of source code. To efficiently analyze coverage, it often generates control flow graph from source code.**

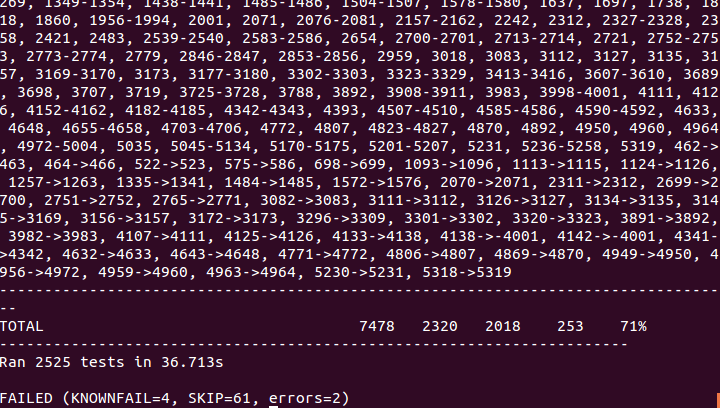
**Control flow graph is made by combination of nodes and edges and nodes like usual graph. Nodes represent routine like if statement or switch statement and edges represent control flow between nodes.**

**In white-box method, there is number of ways how to determine coverage percentage, which is called coverage technique. There are two basic structural coverage, which are Statement Coverage and Branch Coverage. Statement coverage only covers true condition, so we are going use Branch Coverage instead to achieve better coverage. Branch coverage aims for every possible edges at each decision point (IF, Switch ...) executed at least once. This ensures that every reachable point from source code can be tested at least once.**

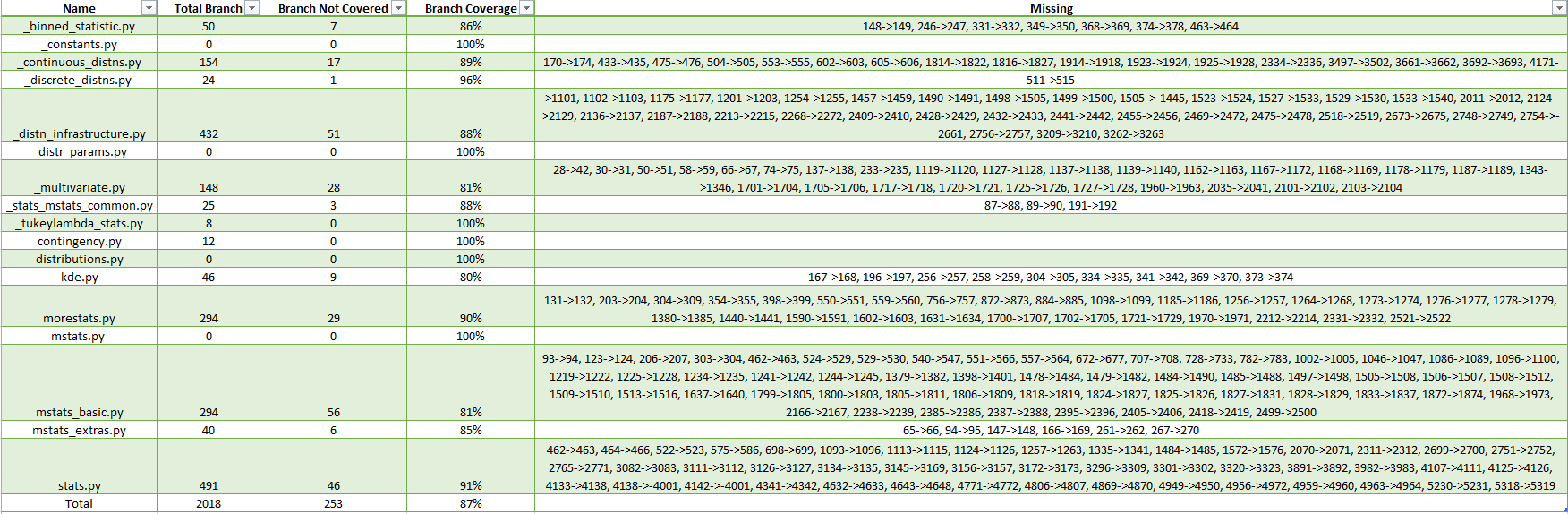
**Process**

**This will be overall process for our group to analyze Stats module in Scipy Library by White-Box Method**

1. **We first need to fully understand about project we are using and this will be most time consuming step** 
   1. **Since we need to understand how software works to analyze it perfectly**
2. **Create control flow graph by analyzing source code, so we can analyze how many routines and control flows are there by drawing nodes and edges.** 
   1. **We found that it is hard to find Control Flow Graph Generating Tool for Python, so we skipped this part**
3. **Run a test provided by Scipy Library to stats module and generated coverage test result**
   1. **sudo python runtests.py --coverage -s stats**
   2. **There were few skips and errors on test (See Figure 1)**
4. **Removed Statement coverage data and arranged branches into excel file**
   1. **See Figure 2**
5. **Moved excel file data into table including line number and solution**
6. **Analyze Test Data Adequacy for the project and provide some sample test cases that can be inserted into test module**



**Test Result (Figure 1)**



**Excel Data (Figure 2)**

**Test Result table**

* **Some duplicates are removed from the table**
* **Some uncovered branches are removed due to uncertainty of test result**
* **Some python files do not have any branch or any uncovered branch**
* **With else conditions, it is default else condition if statements to be executed == N/A**

|  |  |  |
| --- | --- | --- |
| **Branch Condition** | **Line #** | **Statements to be executed** |
| if floc is not None and fscale is not None: | **170 429** | raise ValueError("All parameters fixed. There is nothing to " |
| "optimize.") |
| if f0 is not None and f1 is not None: | **433** | raise ValueError("All parameters fixed. There is nothing to " |
| "optimize.") |
|  |
| if ier != 1: | **475 504** | raise FitSolverError(mesg=mesg) |
|
| elif (n == 3.0): | **555** | return where(b > 3, a\*(a+1.0)\*(a+2.0)/((b-3.0)\*(b-2.0)\*(b-1.0)),inf) |
| elif (n == 4.0): | **558** | return where(b > 4, a\*(a+1.0)\*(a+2.0)\*(a+3.0)/((b-4.0)\*(b-3.0) \* (b-2.0)\*(b-1.0)), inf) |
| else: | **562** | raise NotImplementedError |
|
| if 's' in moments: | **602** | g1 = sqrt(2)\*(12\*c\*c-9\*c\*k\*(c+2)+2\*k\*k\*(c\*(c+3)+3)) |
| g1 /= sqrt(c\*(c\*(k-2)+2\*k))\*(3\*c\*(k-2)+6\*k) |
| if 'k' in moments: | **605** | g2 = (c\*\*3\*(k-3)\*(k\*(3\*k-16)+24)+12\*k\*c\*c\*(k-4)\*(k-3)+ 6\*c\*k\*k\*(3\*k-14) + 12\*k\*\*3) |
| g2 /= 3\*c\*(c\*(k-2)+2\*k)\*\*2 |
| elif y > -3: | **1822** | x0 = exp(y/2.332) + 0.08661 |

**Look at StatsUncoveredBranchList.xlsx for full table**

**Recommended Test Cases to Be Inserted**

1. **Most uncovered branches are due to uncovered boundaries or uncovered cases**

|  |  |  |
| --- | --- | --- |
| if n == 1: | **354** | return kstat(data, n=2) \* 1.0/N |

**In morestats.py**

1. **Test for wrong data object to a function**

|  |  |  |
| --- | --- | --- |
| if not isinstance(sparams, tuple): | **550** | sparams = tuple(sparams) |

**In morestats.py**

1. **Test for many cases for different number of elements in array or list**

|  |  |  |
| --- | --- | --- |
| if N != 1 and N != 2: | **246** | xedges = yedges = np.asarray(bins, float)  bins = [xedges, yedges] |

**In \_binned\_statistic.py**

1. **Test for case that calls unimplemented method(function)**

|  |  |  |
| --- | --- | --- |
| if not callable(statistic) and statistic not in known\_stats: | **331** | raise ValueError('invalid statistic %r' % (statistic,)) |

**In \_binned\_statistic.py**

1. **Include non scalar value into number array or list**

|  |  |  |
| --- | --- | --- |
| for i in np.arange(D):   * if !(np.isscalar(bins[i])): | **377** | edges[i] = np.asarray(bins[i], float)  nbin[i] = len(edges[i]) + 1 # +1 for outlier bins |

**In \_binned\_statistic.py**

1. **Test if error handling works correctly**

|  |  |  |
| --- | --- | --- |
| if ier != 1: | **475 504** | raise FitSolverError(mesg=mesg) |

**In \_continuous\_distns.py**

1. **Passing non-ndarray**
2. **Testing for optional parameters**

|  |  |  |
| --- | --- | --- |
| if plot is not None: | **756** | plot.plot(svals, ppcc, 'x') |
| … |

**In morestats.py**

1. **There are also few cases that cannot be tested since that branch is for defensive programming(Ensure value is inserted correctly when it should be)**

|  |  |  |
| --- | --- | --- |
| if d != self.d: | **196** | if d == 1 and m == self.d: |
| …. |

**In kde.py**

**Conclusion**

**There were total 2018 branches for stats module and 253 uncovered branches by testing.**

1. **(2018 – 253) / 2018 = 87.5%**

**Overall, 87.5% is decent percentage for white-box test, so we can state that this project is well tested for branch coverage. However, there are still few test cases that can be inserted to eliminate uncovered branches. There are two common reasons to be uncovered, one is due to missing simple test conditions and the other is due to defensive programming that will not be executed forever. Mostly, uncovered branches can be covered by adding test cases for certain simple conditions.**