Test Strategies

cost_calculation

The test strategy that we have decided to deploy for cost calculation is condition and branch coverage. The reason for this is because there are multiple variables to consider when we are making the calculations for how much it costs to charge the vehicle. By using empty strings, we can see if the program will act appropriately and return an error instead of just continuing with the error. Finally, we use appropriate values that goes through both the if statements with 100% condition coverage to make sure it acts appropriately as well.

time_calculation

Just like the previous one, the test strategy that we have decided to use for time calculation is also condition and branch coverage. The reasoning behind this choice of our is because we also needed to consider when using empty string inputs for this function as well and see if it would spit out the error that we were expecting. Then when we input everything appropriately, we make sure that the output given to us is roughly in line to what we are expecting.

is_holiday

The next function would be in line with the previous strategies that we use. The strategies that we decided to use for the holiday function, where it returns a True or False, is also condition and branch coverage. This is because there are multiple if statements across the entire function. We need to make sure it goes through all the branches and make sure that the results output given to us is also in line with what we are expecting. By using this strategy, we are able to minimize the amount of test cases necessary instead of giving out 2^n test cases where n is the amount of variables.

is_peak

The strategy for the function where we check if it is peak hour uses the branch coverage. The reason why we deviated from the condition and branch coverage is because there is only one variable to consider in this function. Since there is only one variable, there is no reason to reduce the amount of test cases.

get_duration

The strategy that we have decided to use for the get duration function uses line coverage. This is because this function uses no if statements, therefore we only need to make sure that all the lines in this function is covered to make sure that everything has been run.

getID

The strategy we used for the function to get the ID uses condition coverage. This is because the function has two possible outcomes. The first outcome being where the ID given to it is not a valid ID, therefore spitting out an error. The second one being where it is a legitimate ID and returning it instead. By using this strategy, we make sure that the variable given to us is correct. Since there is only one variable, we cannot reduce the amount of test cases, which makes it unnecessary to use the branch and condition coverage.

get_sun_hour

The strategy used for getting the amount of time there is sun out uses line coverage. This is because there are no if statements. This is just used to make sure that all the lines are run and everything is working as expected.

get_day_light_length

The same can be said for the function that gets the daylight length. This is because, again, there are no if statements, it just needs to run through the whole function and there is only one outcome. So we just need to make sure that the function only works as it is intended. Making it a fairly simple testing strategy

get_cloud_cover

We cannot say the same for this function though. Since this function, of getting the amount of cloud cover, uses the API though, there is more than just one variable to consider as it might be due that whatever inputted may be wrong. Therefore, we need to use branch and condition coverage even if there is only one if statement. Making this the perfect strategy to use when considering

solar_energy

As for the solar energy, we decided to use line coverage as well. Even though there is a while loop that iterates through that section of code, we are sure that there would be no errors in that while loop. This is because there is only one possible outcome after going through the whole code as there is only one return statement and nothing is raised in that section of code.

provide_mean_sum

Finally, for the mean sum function, we have decided to use condition and branch coverage for this function. This is because, there are too many variables to consider for this function if we want to account for every single possible combination of variables. Therefore, the most logical way to do it is to by reducing the amount of test cases while also making sure that they cover every single possible branch and conditions. Even though there is only one return statement, the value can be altered in many ways based on the conditions and variables given to us.