Name: Aw Yong Jia Min Kellie
Student ID: 1005466

From the lesson on Blackbox Testing, the use case diagram is essential to be used as reference such that input space should be derived for each use case. From the input spaces derived, for each input space, an equivalence partition could then be done. Subsequently, for each equivalence partition, one can then proceed to choose a boundary and middle value.

Equivalence Classes

Using the idea that input in the same class should behave similarly in the program, for each of the more prominent use cases, I came up with equivalence classes such that the behaviour(output) is similar for each of the classes.

1. **Use Case: Parse through CSV file**
   This use case takes in a CSV file as input and produces entries for each line in the CSV file to be stored in a list as output.
   - Empty file:
     The output all instances of this equivalence class produces an empty list

   - Filled CSV file:
     The output of all instances of this equivalence class produces a list with all the entries in the CSV file.

2. **Use Case: Compare for differences**
   This use case takes in two lists as input and produces a list of the differences as output.
   - Two lists with equal values:
     The output of all instances of this equivalence class produces an empty list.

   - Two lists with different values:
     The output of all instances of this equivalence class produces a list with the differences found.

   - Two lists with one list being empty:
     The output of all instances of this equivalence class produces a list with all the entries in the filled list.

3. **Use Case: Store differences in another CSV file**
   This use case takes in a single list of all the differences found and produces another CSV file containing all the differences as output.
   - An empty list:
     The output of all instances of this equivalence class produces an empty CSV file.

   - A list with some blank entries:
     The output of all instances of this equivalence class produces a CSV file containing all entries in the list including the blank entries.

- A list with filled entries:
  The output of all instances of this equivalence class produces a CSV file with all the entries in the list.

Boundary Value Analysis

1. Use Case: Parse through CSV file
   - Empty file:
     As this is a special equivalence class where it is a singleton, it does not have any boundary or middle value.

   - Filled CSV file:
     The ***boundary value*** chosen for this would be a CSV file with one entry. One would only have to remove this one entry for it to become another equivalence class, an empty file.
     The ***middle value*** chosen for this would be a CSV file with several entries. One would then have to delete all entries for it to become another equivalence class, an empty file.

2. Use Case: Compare for differences
   - Two lists with equal values:
     The ***boundary value*** chosen for this would be two empty lists. Inherently, as both lists do not contain any entries at all, their values are considered to be equal hence no difference would be found. To make it become another equivalence class, two lists with different values, one could intentionally make one of the lists to appear empty by adding spaces in the entry instead.
     The ***middle value*** chosen for this would be two lists with the same values and in the same order. For these two lists, they are guaranteed to behave in the same way such that no difference would be found. To make them different, one has to either delete an entry or add an entry to one of the lists which are not slight changes. Thus, two lists with the same values in the same order serve as a middle value.

   - Two lists with different values:
     The ***boundary value*** chosen for this would be one of the lists having an additional intended blank entry at the bottom of the list. Inherently, it could be thought of two lists with different values since there is an intentional blank entry in one of the lists filled with spaces which is not seen in the other lists. However, by just removing that one blank entry, it would become another equivalence class, two lists with equal values especially if the two lists had equal values in the same order except for an additional blank entry at the last row for one of the lists.
     The ***middle value*** chosen for this would be two filled lists with values that differ for each row between the two lists. These two lists would then require all entries to be changed for either of the lists to match it with the other list in order for them to become two lists with equal values. Doing this would require great changes, making the two filled lists with values that differ for each row between the two lists as a middle value.

- Two lists with one list being empty:
  The ***boundary value*** chosen for this would be one of the lists having only one entry and the other list being an empty one. To make this into another equivalence class, two lists with same values, one only has to add the same entry into the empty list.
  The ***middle value*** chosen for this would be one of the lists having multiple entries while the other list is just an empty one. To make this into another equivalence class, two lists with the same values, one has to add many entries which are equal to the filled list onto the empty lists.

3. Use Case: Store differences in another CSV file
   - An empty list:
     As this is a special equivalence class where it is a singleton, it does not have any boundary or middle value.

   - A list with some blank entries:
     The ***boundary value*** chosen for this would be a list with a blank entry below filled entries. With a slight change where the blank entry is no longer an entry whereby both scenarios look similar, it would then become another equivalence class, a list with filled entries.
     The ***middle value*** chosen for this would be a list with blank entries in between the filled entries. This list is evidently a list that contains blank entries in between filled entries and thus to make it become another equivalence class, a list with filled entries, one has to either remove all the blank entries or to move all the blank entries to the bottom of the list. To also make it become an empty list, one has to remove all entries. Thus with all these changes which require more than just slight changes, a list with blank entries serve as the middle value.

   - A list with filled entries:
     The ***boundary value*** chosen for this would be a list with an intended blank entry at the last row. Such a list does serve as a filled entry as that entry could contain spaces, making the list generally, a filled one. However, by removing such spaces in the entry to intentionally make it seem blank, which is a slight change, that entry thus becomes an empty entry. This allows it to become another equivalence class, a list with some blank entries.
     The ***middle value*** chosen for this would be a list with no blank entries. Such a list would require all entries to be deleted in order for it to become another equivalence class, an empty list. It would also require certain entries to be deleted for it to become another equivalence class, a list with some blank entries. Hence, as all these changes are not slight changes, this list with no blank entries thus serves as a middle value.