Ian Eggleston

1. vec_2 <- vec_1==3
2. There are too many values in the original vector because we are running it 12,345 times. A program can be run to determine when 3 occurred much faster than we can visually check.
3. The number of times 3 is generated is random so this is expected to change each time vector 1 is rerun.
4. The size of the vector is too large to check each time by hand. Because we are only checking for one number, a logical test allows us to check just the number of times 3 occurs anywhere in the table. It also works for multiple types of vectors that change over time. A code does not have to be written for each specific vector.
5. Subsetting by hand creates problems from human error, if values are missed or miscounted for extra. It is also not very flexible when comparing different vectors and datasets. If a code is written that is general enough for multiple applications, then it speeds up analyses in a consistent manner. The code will be the same for anyone that uses it even for different data sets, so reproducibility is guaranteed.
6. ```
   for (i in 1:10)
   {
    print(paste("This is loop iteration: ", i))
   }
   ```
7. ```
   n=1234
   for (i in 1:n)
   {
    print(i)
   }
   ```
8. ```
   n=17
   vec_1 = sample(10, n, replace=TRUE)
   print(paste("## The element of vec_1 at index ", 1:n, " is ", vec_1))
   ```

   Alternative with for loop

   ```
   n=17
   vec_1 = sample(10, n, replace=TRUE)
   for (i in 1:n)
   {
   print(paste("## The element of vec_1 at index ", i, " is ", vec_1[i]))
   }
   ```

9. ```
   create_and_print_vec = function(n, min = 100,  max = 2000)
   {
    vec_1 = sample(min:max, n, replace=TRUE)
    print(paste("The element at index ", 1:n, " is ", vec_1))
   }
   ```

```
create_and_print_vec(10)
```

Alternative with for loop

```
create_and_print_vec = function(n, min = 100,  max = 2000)
{
  vec_1 = sample(min:max, n, replace=TRUE)
  for (i in 1:n)
  {
  print(paste("The element at index ", i, " is ", vec_1[i]))
  }
}
create_and_print_vec(10)
```