Pre-fix Sum Equals K :

if you ever get a pre-fix sum, where two values are the same, that means the value in between those two values sum to zero.

- $sum_i - sum_j = 0$

case 1

input: nums = [1,1,1], k = 2

output: 2

case 2

input: nums = [1,2,3], k = 3 (1 + 3 + 6)

output: 2

if: $sum_i - k = sum_j$ then, we know the difference between sum_i and sum_j is actually k.

deriving the idea → $sum_i - sum_j = k$
→ $sum_i = k + sum_j$
→ $sum_i - k = sum_j$

1 2 3 1 3 6 3
[1,1,1] , [1,2,3,-3]

Using a hashmap, we will store prefix_sum - k as the key and the # of times we've seen it as the value.

handling case 1: Assume we have an empty hashmap: {}

prefix_sum : [1,1,1]
1+1

↑ this is an answer, but we only add it if prefix_sum - k is already in our hashmap. So, we're not counting it towards our number of solutions.

intuition : the first time we see a contiguous subarray that equals to k, we won't count that solution.

To handle this case, we need to add {0:1} to account for the first solution.

pseudocode :

input : array of integers, nums and target value k
output : # of contiguous subarrays equal to k.

```
subarray(nums, k):

  prefix_sum, result = 0;
  prefix_hash = {0:1}

  for num in nums:
     prefix_sum += num;
     if prefix_sum - k in prefix_hash:
        result += prefix_hash[prefix_sum - k]
     if prefix_sum not in prefix_hash:
        prefix_hash[prefix_sum] += 1;
     else:
        prefix_hash[prefix_sum] = 1;
  return result;
```