Rust and it's ability to insert/replace string into another string onto specific position

Let's say you have a string you just loaded from a file. You walk thru that file content (with a regex match) and you want to substitude a precise location (the current match in your matches loop) for something different - a new string. You cannot use "replace all" or even "replace" approach because it might cause the source string will be modified on another places but your current match or you need to compute each substitution separately so you cannot generalize the process.

In another words imagine the operation as a simple text edit done by user in a text editor - you locate the pattern, you select it with your mouse, hit backspace and write in something else. The question is how to do that in Rust since `std` lib (any any other as I'm aware after my crates.io research) doesn't offer a function like `replace_at(self, start_pos, end_pos, substitution)`.

The solution is not a rocket science but requires some basic knowledge of bytes and strings in Rust.

1. string is a list of bytes (not chars)
2. each char in a `String` is represented by 1 byte as long as the character is ASCII up to 4 bytes otherwise (because of UTF-8)
3. indexing `String` is forbidden - index over bytes or over chars

In our example we work with `Match` from Regex library which provides `start()` and `end()` positions of the match (and even Range with `range()`) for such match. With such info we can cut off from original string and then perform insertion.

Fortunatelly Vec provides nifty method called splice which "replaces the specified range in the vector with the given iterator". That means you can perform "cut off" and "insert" operation in the same time - conveniently.

Here is the example code with some helpful debugging.

```
use std::mem;
use regex::Regex;

fn main() {
    let s = "Pavel X".to_string();

    println!("Bytes: {:?}", &s.bytes());
    println!("Length: {}", s.len());
    println!("Memory size: {}", mem::size_of_val(&s));

    let re = Regex::new("ave").unwrap();
    let find = re.find(&s).unwrap();
```

```
    let range = find.range();
    println!("Regex match: {:?}", &find);
    println!("Regex match bytes: {:?}", &s.as_bytes()[range.clone()]);

    let mut sb = s.into_bytes();
    {
        let removed: Vec<_> = sb.splice(range,
         ↪  "----------".bytes()).collect();
        println!("Removed from string:{:?}", &removed);
    }
    println!("Final string: {}", String::from_utf8(sb).unwrap());
}
```

Also sits on playground.