A value was moved out while it was still borrowed.

Erroneous code example:

```
struct Value {}

fn borrow(val: &Value) {}

fn eat(val: Value) {}

fn main() {
    let x = Value{};
    let _ref_to_val: &Value = &x;
    eat(x);
    borrow(_ref_to_val);
}
```

Here, the function `eat` takes ownership of `x`. However, `x` cannot be moved because the borrow to `_ref_to_val` needs to last till the function `borrow`. To fix that you can do a few different things:

- Try to avoid moving the variable.
- Release borrow before move.
- Implement the `Copy` trait on the type.

Examples:

```
struct Value {}

fn borrow(val: &Value) {}

fn eat(val: &Value) {}

fn main() {
    let x = Value{};

    let ref_to_val: &Value = &x;
    eat(&x); // pass by reference, if it's possible
    borrow(ref_to_val);
}
```

Or:

```
struct Value {}

fn borrow(val: &Value) {}

fn eat(val: Value) {}

fn main() {
    let x = Value{};

    let ref_to_val: &Value = &x;
    borrow(ref_to_val);
    // ref_to_val is no longer used.
    eat(x);
}
```

Or:

```
#[derive(Clone, Copy)] // implement Copy trait
struct Value {}

fn borrow(val: &Value) {}

fn eat(val: Value) {}

fn main() {
    let x = Value{};
    let ref_to_val: &Value = &x;
    eat(x); // it will be copied here.
    borrow(ref_to_val);
}
```

For more information on Rust's ownership system, take a look at the References & Borrowing section of the Book.