This error occurs when an attempt is made to move out of a value whose type implements the `Drop` trait.

Erroneous code example:

```
struct FancyNum {
    num: usize
}

struct DropStruct {
    fancy: FancyNum
}

impl Drop for DropStruct {
    fn drop(&mut self) {
        // Destruct DropStruct, possibly using FancyNum
    }
}

fn main() {
    let drop_struct = DropStruct{fancy: FancyNum{num: 5}};
    let fancy_field = drop_struct.fancy; // Error E0509
    println!("Fancy: {}", fancy_field.num);
    // implicit call to `drop_struct.drop()` as drop_struct goes out of scope
}
```

Here, we tried to move a field out of a struct of type `DropStruct` which implements the `Drop` trait. However, a struct cannot be dropped if one or more of its fields have been moved.

Structs implementing the `Drop` trait have an implicit destructor that gets called when they go out of scope. This destructor may use the fields of the struct, so moving out of the struct could make it impossible to run the destructor. Therefore, we must think of all values whose type implements the `Drop` trait as single units whose fields cannot be moved.

This error can be fixed by creating a reference to the fields of a struct, enum, or tuple using the `ref` keyword:

```
struct FancyNum {
    num: usize
}

struct DropStruct {
    fancy: FancyNum
}

impl Drop for DropStruct {
    fn drop(&mut self) {
```

```
        // Destruct DropStruct, possibly using FancyNum
    }
}

fn main() {
    let drop_struct = DropStruct{fancy: FancyNum{num: 5}};
    let ref fancy_field = drop_struct.fancy; // No more errors!
    println!("Fancy: {}", fancy_field.num);
    // implicit call to `drop_struct.drop()` as drop_struct goes out of scope
}
```

Note that this technique can also be used in the arms of a match expression:

```
struct FancyNum {
    num: usize
}

enum DropEnum {
    Fancy(FancyNum)
}

impl Drop for DropEnum {
    fn drop(&mut self) {
        // Destruct DropEnum, possibly using FancyNum
    }
}

fn main() {
    // Creates and enum of type `DropEnum`, which implements `Drop`
    let drop_enum = DropEnum::Fancy(FancyNum{num: 10});
    match drop_enum {
        // Creates a reference to the inside of `DropEnum::Fancy`
        DropEnum::Fancy(ref fancy_field) => // No error!
            println!("It was fancy-- {}!", fancy_field.num),
    }
    // implicit call to `drop_enum.drop()` as drop_enum goes out of scope
}
```