```
Data: Data frame input by user
Result: Interactive litre plot
/* Declare Shiny server
server ← function(input, output, session){
    /* User input options
    observeEvent(input$goButton, values$x ← values$x + 1)
    observeEvent(input$selPair, values$x ← 0)
    observeEvent(input$selMetric, values$x ← 0)
    observeEvent(input$selOrder, values$x ← 0)
    observeEvent(input$binSize, values$x ← 0)
    /* Create reactive expression of plotly background litre plot
    gP ← reactive(p ← ggplot(data); gP ← ggplotly(p))
    /* Declare shiny output litre plot
    output$litrePlot ← renderPlotly({
        /* Create reactive expression of plotly background litre plot
        plotlyLitre ← reactive(gP())
        /* Tailor interactivity of the plotly litre plot object using custom JavaScript
        plotlyLitre() %>% onRender("function(el, x, data){
            /* Read handle called 'points' to obtain variables sent from R into JavaScript
            Shiny.addCustomMessageHandler('points', function(drawPoints){
                /* Delete any old superimposed plotly geoms (dots)
                if (x.data.length > 0){Plotly.deleteTraces(el.id)}
                /* Create traces for selected gene IDs as points that state gene names upon hovering
                trace = {x: drawPoints.geneX, y: drawPoints.geneY, mode: 'markers', color: drawPoints.pointColor, size:
                  drawPoints.pointSize, text: drawPoints.geneID, hoverinfo: 'text'}
                /* Superimpose traces onto the plotly litre plot object
                Plotly.addTraces(el.id, trace)
            })
        }")
    })
    /* If the user changes their input, store information about new superimposed genes with a handle called 'points'.
      These values can then be sent from R to JavaScript
    observe({session$sendCustomMessage(type = "points", message=list(geneX=geneX, geneY=geneY, pointSize =
      pointSize, geneID=geneID, pointColor=pointColor))})
    /* Declare Shiny output boxplot
    output$boxPlot ← renderPlotly({
        /* Create reactive expression of plotly background boxplot BP ← reactive(ggplot() + geom_boxplot()) ggBP ←
          reactive(ggplotly(BP()))
        /* If the user changes their input, store information about new superimposed genes with a handle called 'lines'.
          These values can then be sent from R to JavaScript
        observe({session$sendCustomMessage(type = "lines", message=list(geneInfo=currGene(), geneID=geneID,
          pointColor=pointColor))})
        /* Tailor interactivity of the plotly boxplot object using custom JavaScript
        ggBP() %>% onRender("function(el, x, data){
            /* Read handle called 'lines' to obtain variables sent from R into JavaScript
            Shiny.addCustomMessageHandler('lines', function(drawLines){
                /* Delete any old superimposed plotly geoms (lines)
                Plotly.deleteTraces(el.id, traceLine)
                /* Create traces for selected gene IDs as lines that state gene names upon hovering
                traceLine = {x: drawLines.geneInfo, y: drawLines.geneInfo, mode: 'lines', color: drawLines.pointColor,
                  width: 2, opacity: 0.9; text: drawLines.geneID, hoverinfo: 'text'}
                /* Superimpose traces onto the plotly litre plot object
                Plotly.addTraces(el.id, traceLine)
            })
        })
    })
}
```

S2 Pseudocode: Pseudocode for interactive litre plot