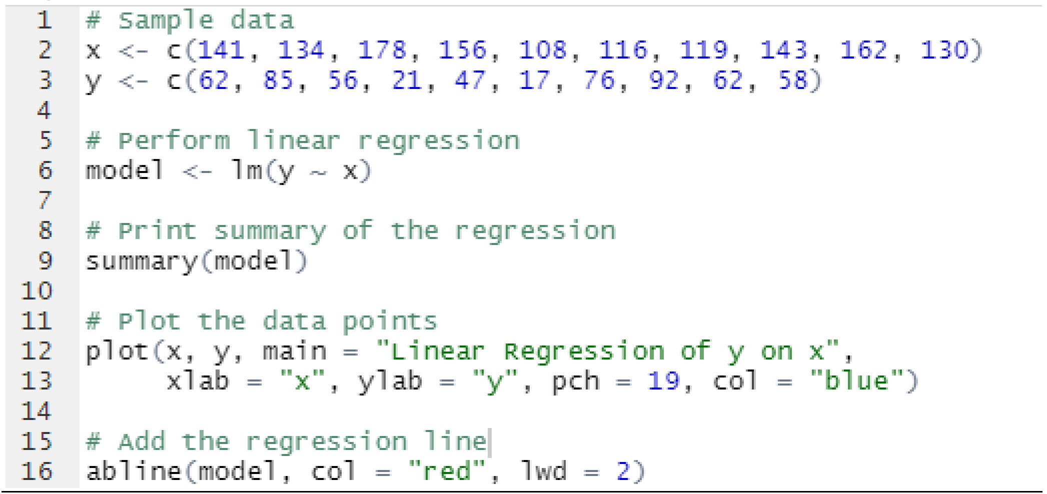
Write R program to find the following Regressions with the sample data and visualize the regressions graphically.

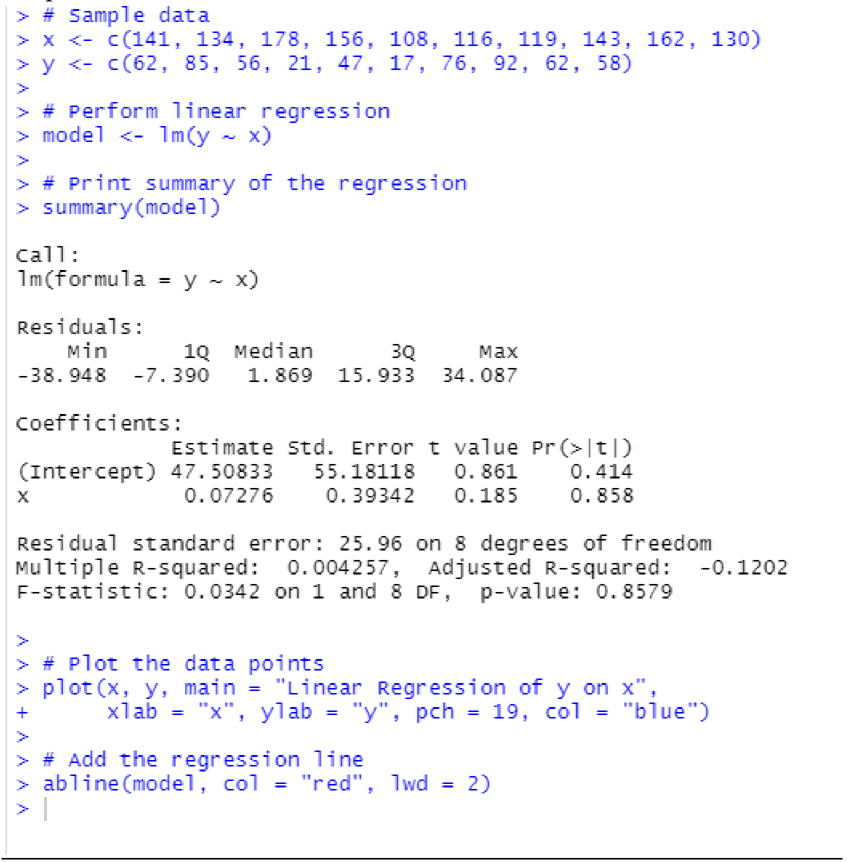
a) Linear Regression

Program:

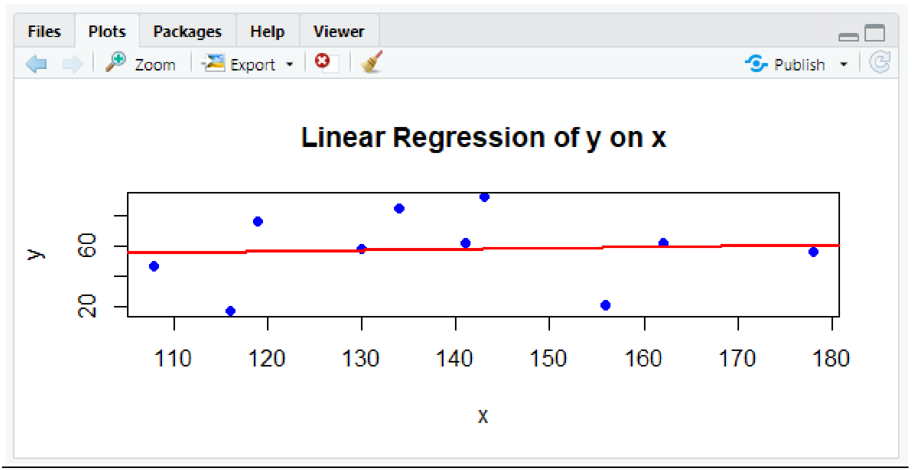


```r
# Sample data
x <- c(141, 134, 178, 156, 108, 116, 119, 143, 162, 130)
y <- c(62, 85, 56, 21, 47, 17, 76, 92, 62, 58)

# Perform linear regression
model <- lm(y ~ x)

# Print summary of the regression
summary(model)

# Plot the data points
plot(x, y, main = "Linear Regression of y on x",
     xlab = "x", ylab = "y", pch = 19, col = "blue")

# Add the regression line
abline(model, col = "red", lwd = 2)
```

Output:



```
> # Sample data
> x <- c(141, 134, 178, 156, 108, 116, 119, 143, 162, 130)
> y <- c(62, 85, 56, 21, 47, 17, 76, 92, 62, 58)
>
> # Perform linear regression
> model <- lm(y ~ x)
>
> # Print summary of the regression
> summary(model)

Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max 
-38.948  -7.390   1.869  15.933  34.087 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 47.50833   55.18118   0.861    0.414
x            0.07276    0.39342   0.185    0.858

Residual standard error: 25.96 on 8 degrees of freedom
Multiple R-squared:  0.004257,	Adjusted R-squared:  -0.1202 
F-statistic: 0.0342 on 1 and 8 DF,  p-value: 0.8579

>
> # Plot the data points
> plot(x, y, main = "Linear Regression of y on x",
+      xlab = "x", ylab = "y", pch = 19, col = "blue")
>
> # Add the regression line
> abline(model, col = "red", lwd = 2)
> 
```



Explanation:

x <- c(141, 134, 178, 156, 108, 116, 119, 143, 162, 130)

- **This line creates a numeric vector** x, containing the sample data points for the independent variable (predictor).

y <- c(62, 85, 56, 21, 47, 17, 76, 92, 62, 58)

- **This line creates another numeric vector** y, which holds the corresponding values of the dependent variable (response).

model <- lm(y ~ x)

- This line **fits a linear regression model** where *y is modeled as a function of x*.
- lm() stands for "linear model".
- The syntax y ~ x means "predict y using x".
- The result is stored in the object model.

summary(model)

- This line **prints a detailed summary** of the linear regression model.
- It shows important statistics like coefficients (slope and intercept), R-squared, p-values, residuals, etc.
- Useful for understanding how well the model fits the data.

```
plot(x, y, main = "Linear Regression of y on x",
    xlab = "x", ylab = "y", pch = 19, col = "blue")
```

- This creates a **scatter plot of the data points**.
- x and y are plotted on the horizontal and vertical axes, respectively.
- main sets the title of the plot.
- xlab and ylab label the x and y axes.
- pch = 19 sets the plot character to filled circles (better visibility).
- col = "blue" colors the points blue.

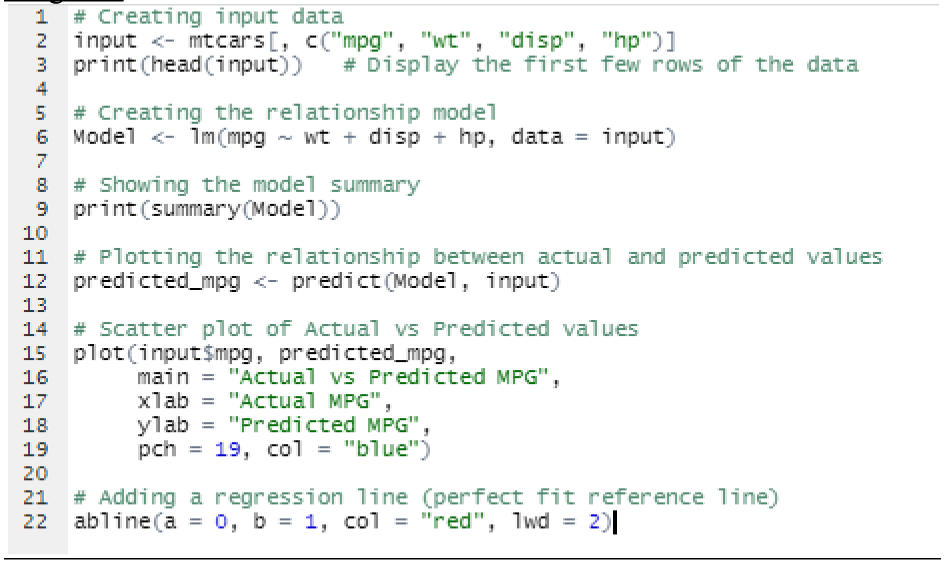
abline(model, col = "red", lwd = 2)

- This adds the **regression line** to the plot.
- abline() draws a straight line; here it uses the intercept and slope from model.
- col = "red" colors the regression line red.
- lwd = 2 sets the line width to 2, making it thicker and easier to see.

---

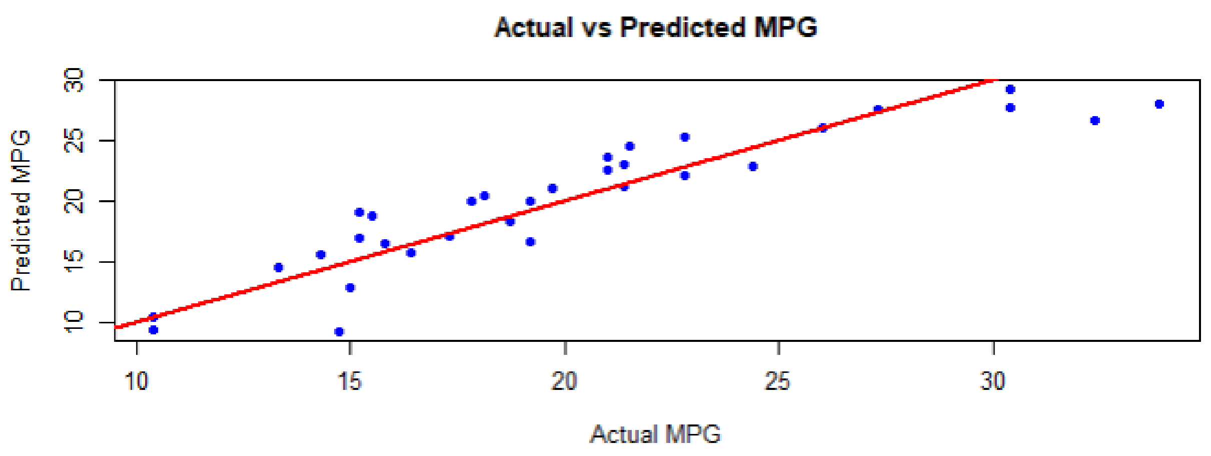
b) Multiple Regression

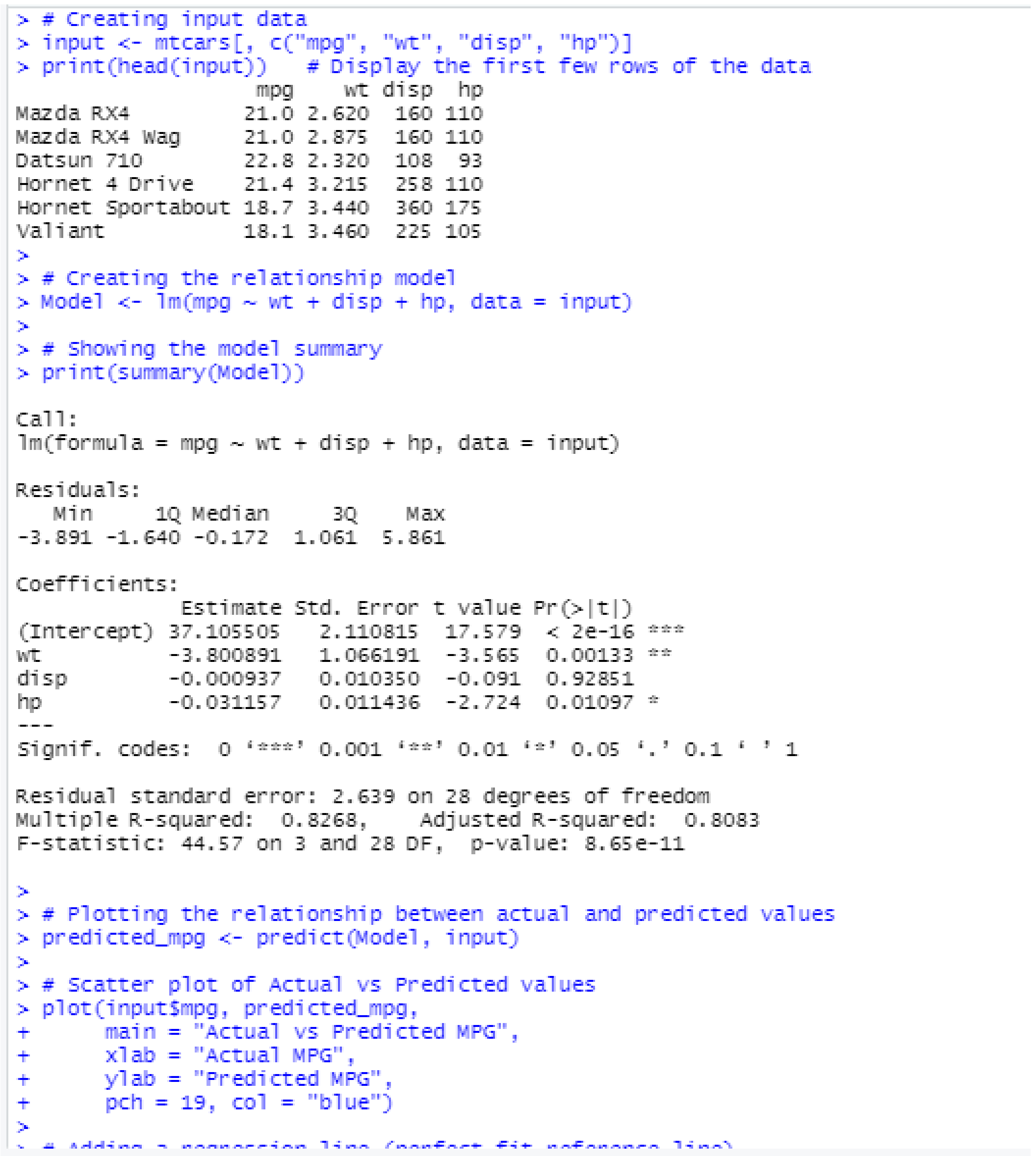
Program:



```
# Creating input data
input <- mtcars[, c("mpg", "wt", "disp", "hp")]
print(head(input))   # Display the first few rows of the data

# Creating the relationship model
Model <- lm(mpg ~ wt + disp + hp, data = input)

# Showing the model summary
print(summary(Model))

# Plotting the relationship between actual and predicted values
predicted_mpg <- predict(Model, input)

# Scatter plot of Actual vs Predicted values
plot(input$mpg, predicted_mpg,
     main = "Actual vs Predicted MPG",
     xlab = "Actual MPG",
     ylab = "Predicted MPG",
     pch = 19, col = "blue")

# Adding a regression line (perfect fit reference line)
abline(a = 0, b = 1, col = "red", lwd = 2)
```

Output:





```
> # Creating input data
> input <- mtcars[, c("mpg", "wt", "disp", "hp")]
> print(head(input))   # Display the first few rows of the data
                   mpg    wt disp  hp
Mazda RX4         21.0 2.620  160 110
Mazda RX4 Wag     21.0 2.875  160 110
Datsun 710        22.8 2.320  108  93
Hornet 4 Drive    21.4 3.215  258 110
Hornet Sportabout 18.7 3.440  360 175
Valiant           18.1 3.460  225 105
>
> # Creating the relationship model
> Model <- lm(mpg ~ wt + disp + hp, data = input)
>
> # Showing the model summary
> print(summary(Model))

Call:
lm(formula = mpg ~ wt + disp + hp, data = input)

Residuals:
   Min     1Q Median     3Q    Max 
-3.891 -1.640 -0.172  1.061  5.861 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept) 37.105505   2.110815  17.579  < 2e-16 ***
wt          -3.800891   1.066191  -3.565  0.00133 ** 
disp        -0.000937   0.010350  -0.091  0.92851    
hp          -0.031157   0.011436  -2.724  0.01097 *  
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.639 on 28 degrees of freedom
Multiple R-squared:  0.8268,	Adjusted R-squared:  0.8083 
F-statistic: 44.57 on 3 and 28 DF,  p-value: 8.65e-11

>
> # Plotting the relationship between actual and predicted values
> predicted_mpg <- predict(Model, input)
>
> # Scatter plot of Actual vs Predicted values
> plot(input$mpg, predicted_mpg,
+      main = "Actual vs Predicted MPG",
+      xlab = "Actual MPG",
+      ylab = "Predicted MPG",
+      pch = 19, col = "blue")
>
> # Adding a regression line (perfect fit reference line)
```

Explanation:

#Creating input data.
input <- mtcars[,c("mpg","wt","disp","hp")]

- This line **extracts specific columns** from the built-in dataset mtcars.
- mtcars[, c("mpg","wt","disp","hp")] selects only the columns "mpg", "wt", "disp", and "hp".
- The result is assigned to a new data frame called input.
- So, input now contains only those four variables from the mtcars dataset.

print(head(input))

- head(input) returns the **first 6 rows** of the input data frame.
- print() outputs these rows to the console.
- This lets you quickly check the first few data points in input.

```
Model <- lm(mpg ~ wt + disp + hp, data = input)
```

Now you're checking **how well your model predicts** the actual values of `mpg` (miles per gallon).

The goal of this section is to **compare the predicted vs actual values** of `mpg` visually.

```
predicted_mpg <- predict(Model, input)
```

What it does:

- `predict()` is an R function that takes a model (like `lm`) and a dataset, and **computes the predicted values** of the dependent variable based on the model equation.
- Here, it uses your trained regression model (`Model`) to predict the `mpg` for all cars in the `input` dataset.

```
plot(input$mpg, predicted_mpg,
     main = "Actual vs Predicted MPG",
     xlab = "Actual MPG",
     ylab = "Predicted MPG",
     pch = 19, col = "blue")
```

What it does:

This line creates a **scatter plot** comparing:

- **x-axis (`input$mpg`)** → actual observed values of `mpg`
- **y-axis (`predicted_mpg`)** → model-predicted values of `mpg`

Purpose:

- It helps you **visually assess how well the model fits**.
- If the model predictions are accurate, most points should lie **close to a straight line** (the 45° line where predicted = actual).

Parameter details:

- `main = "Actual vs Predicted MPG"` → Sets the **title** of the graph.
- `xlab = "Actual MPG"` → Label for the **x-axis**.
- `ylab = "Predicted MPG"` → Label for the **y-axis**.
- `pch = 19` → Defines the **plotting character**.
  `19` gives **solid circular points**.
- `col = "blue"` → Makes all the points **blue** in color.

So this line **plots a blue dot** for each car, where the x-position is the *actual mpg* and the y-position is the *predicted mpg*.

```
abline(a = 0, b = 1, col = "red", lwd = 2)
```

What it does:

This adds a **reference line** to your scatter plot.

- `abline()` adds a straight line to an existing plot.
- Here, you specify:
  - `a = 0` → the intercept of the line

    intercept = **5**.
    That's the point where the line cuts the **y-axis** (at y = 5).

  - `b = 1` → the slope of the line

  Slope = **2** -> This means: for every **1 unit increase in x**, `y` increases by **2 units**.

  Slope = **2** -> This means: for every **1 unit increase in x**, `y` decreases by **2 units**.

So the line drawn is the equation:
[
y = x
]
This is a **perfect fit line** — if the model were perfect, every predicted value would equal the actual value, and all blue points would lie exactly on this red line.

Parameter details:

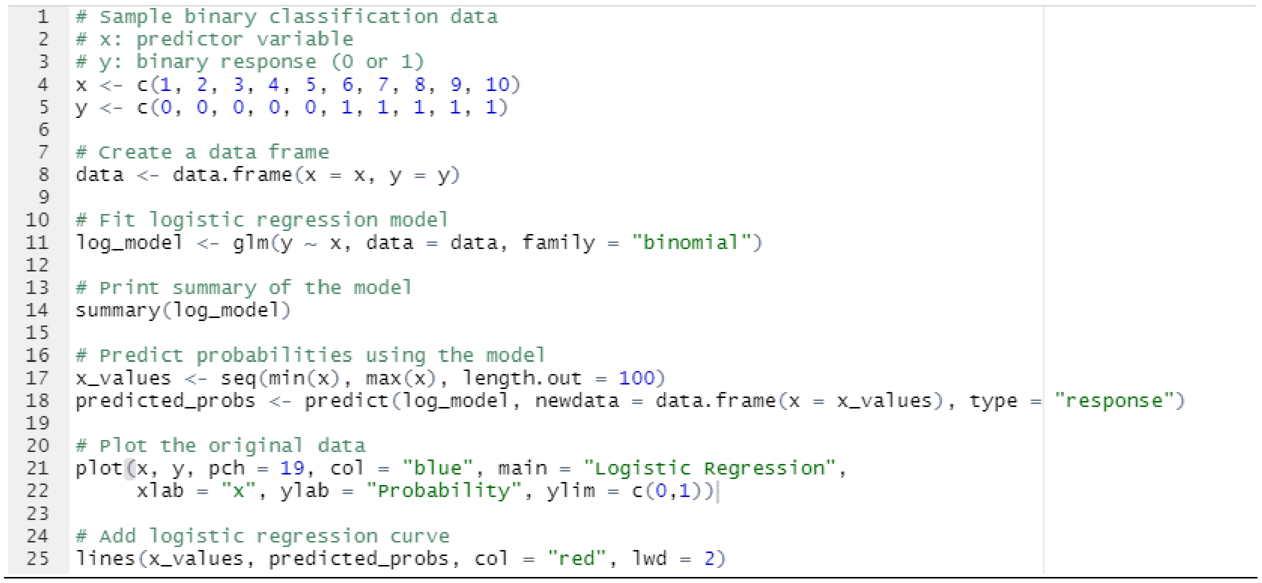
- `col = "red"` → makes the line **red**.
- `lwd = 2` → increases the **line width** for better visibility.

Interpretation:

- Points **on the red line** → predicted ≈ actual → good prediction.
- Points **above the line** → model **underestimated** the mpg (predicted < actual).
- Points **below the line** → model **overestimated** the mpg (predicted > actual).

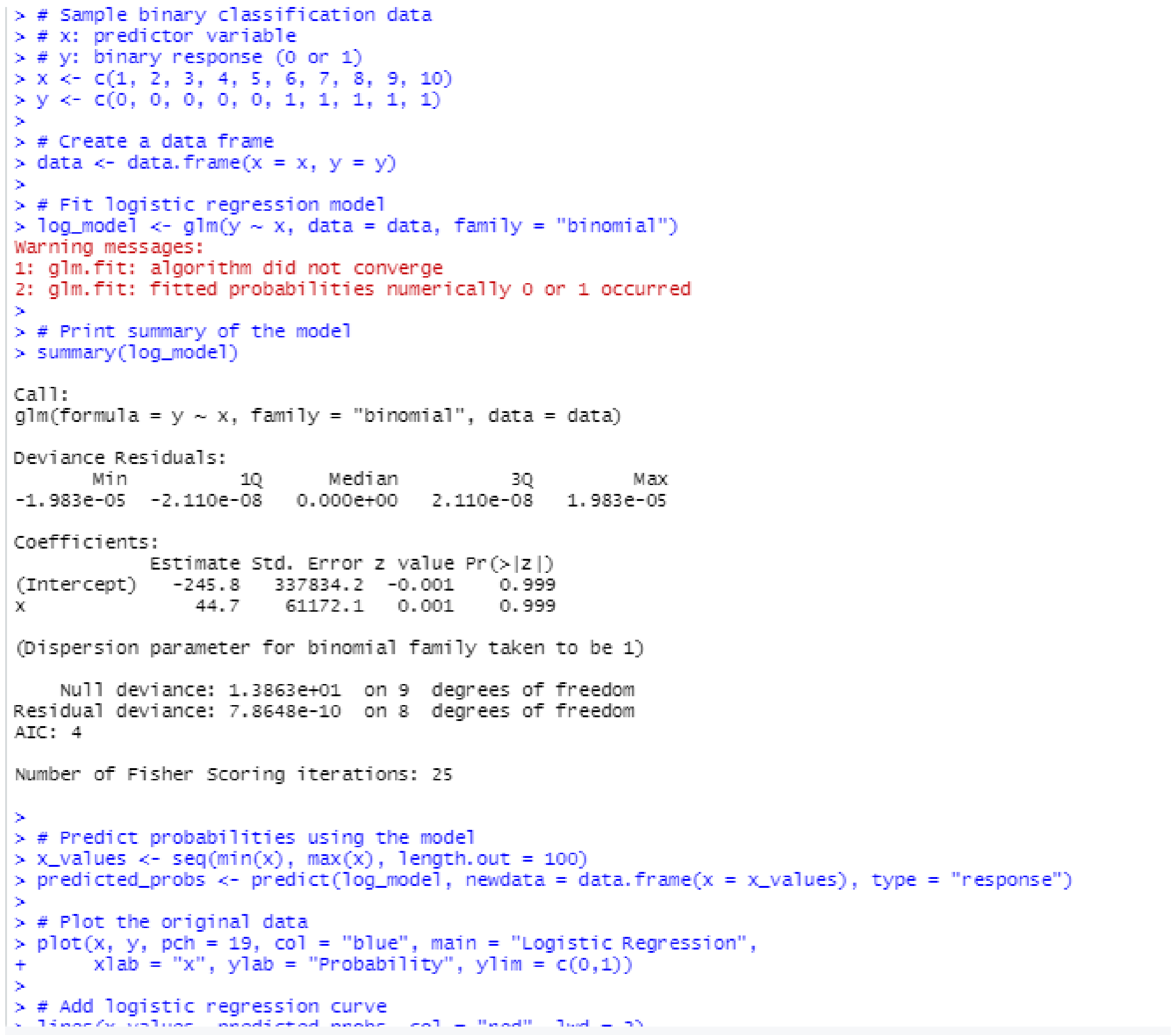
---

c) Logistic Regression
   Program:

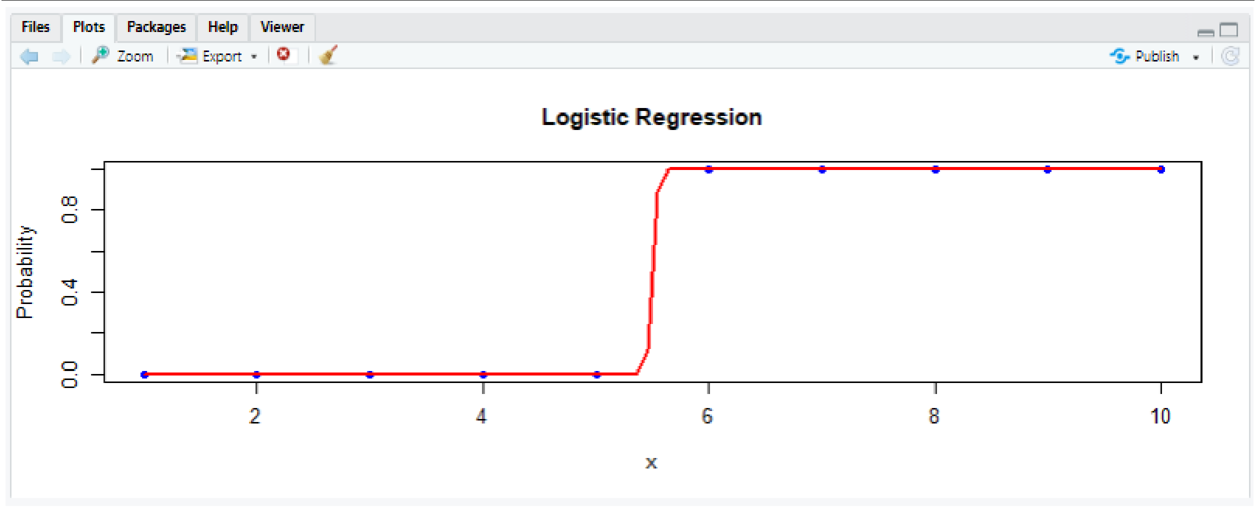


```r
# Sample binary classification data
# x: predictor variable
# y: binary response (0 or 1)
x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
y <- c(0, 0, 0, 0, 0, 1, 1, 1, 1, 1)

# Create a data frame
data <- data.frame(x = x, y = y)

# Fit logistic regression model
log_model <- glm(y ~ x, data = data, family = "binomial")

# Print summary of the model
summary(log_model)

# Predict probabilities using the model
x_values <- seq(min(x), max(x), length.out = 100)
predicted_probs <- predict(log_model, newdata = data.frame(x = x_values), type = "response")

# Plot the original data
plot(x, y, pch = 19, col = "blue", main = "Logistic Regression",
     xlab = "x", ylab = "Probability", ylim = c(0,1))

# Add logistic regression curve
lines(x_values, predicted_probs, col = "red", lwd = 2)
```

Output:



```
> # Sample binary classification data
> # x: predictor variable
> # y: binary response (0 or 1)
> x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
> y <- c(0, 0, 0, 0, 0, 1, 1, 1, 1, 1)
>
> # Create a data frame
> data <- data.frame(x = x, y = y)
>
> # Fit logistic regression model
> log_model <- glm(y ~ x, data = data, family = "binomial")
Warning messages:
1: glm.fit: algorithm did not converge
2: glm.fit: fitted probabilities numerically 0 or 1 occurred
>
> # Print summary of the model
> summary(log_model)

Call:
glm(formula = y ~ x, family = "binomial", data = data)

Deviance Residuals:
       Min          1Q      Median          3Q         Max
-1.983e-05  -2.110e-08   0.000e+00   2.110e-08   1.983e-05

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   -245.8   337834.2  -0.001    0.999
x               44.7    61172.1   0.001    0.999

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1.3863e+01  on 9  degrees of freedom
Residual deviance: 7.8648e-10  on 8  degrees of freedom
AIC: 4

Number of Fisher Scoring iterations: 25

>
> # Predict probabilities using the model
> x_values <- seq(min(x), max(x), length.out = 100)
> predicted_probs <- predict(log_model, newdata = data.frame(x = x_values), type = "response")
>
> # Plot the original data
> plot(x, y, pch = 19, col = "blue", main = "Logistic Regression",
+      xlab = "x", ylab = "Probability", ylim = c(0,1))
>
> # Add logistic regression curve
> lines(x_values, predicted_probs, col = "red", lwd = 2)
```



Explanation:

Sample binary classification data

```
# x: predictor variable
# y: binary response (0 or 1)
x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

- Creates a numeric vector x representing the predictor (independent variable) with values from 1 to 10.

```
y <- c(0, 0, 0, 0, 0, 1, 1, 1, 1, 1)
```

- Creates a numeric vector y representing the binary response variable with values 0 or 1, where the first five are 0 and the last five are 1.

```
# Create a data frame
data <- data.frame(x = x, y = y)
```

- Combines vectors x and y into a data frame named data. This structure is easier to work with in modeling functions. A **data frame** in R is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column. Data frames are used as the fundamental data structure by most of R's modeling software.

```
# Fit logistic regression model
log_model <- glm(y ~ x, data = data, family = "binomial")
```

- Fits a logistic regression model predicting y using x.
- glm() is the function for generalized linear models.
- family = "binomial" specifies logistic regression (for binary outcomes).
- The fitted model is stored in the variable log_model.
-

```
# Print summary of the model
summary(log_model)
```

- Prints a detailed summary of the logistic regression model.
- This includes coefficients, standard errors, z-values, p-values, and overall model fit statistics.

```
# Predict probabilities using the model
x_values <- seq(min(x), max(x), length.out = 100)
```

- Creates a sequence x_values from the minimum to maximum of x, divided into 100 evenly spaced points.
- This is for plotting a smooth logistic curve.

```
predicted_probs <- predict(log_model, newdata = data.frame(x = x_values), type = "response")
```

- Uses the model log_model to predict the probability of y = 1 for each value in x_values.
- type = "response" means the output is on the probability scale (0 to 1).

```
# Plot the original data
plot(x, y, pch = 19, col = "blue", main = "Logistic Regression",
    xlab = "x", ylab = "Probability", ylim = c(0,1))
```

- Plots the original binary data points.
- pch = 19 plots solid circles.
- col = "blue" colors the points blue.
- main sets the plot title.
- xlab and ylab label the axes.
- ylim = c(0,1) sets y-axis limits from 0 to 1 for probability scale.

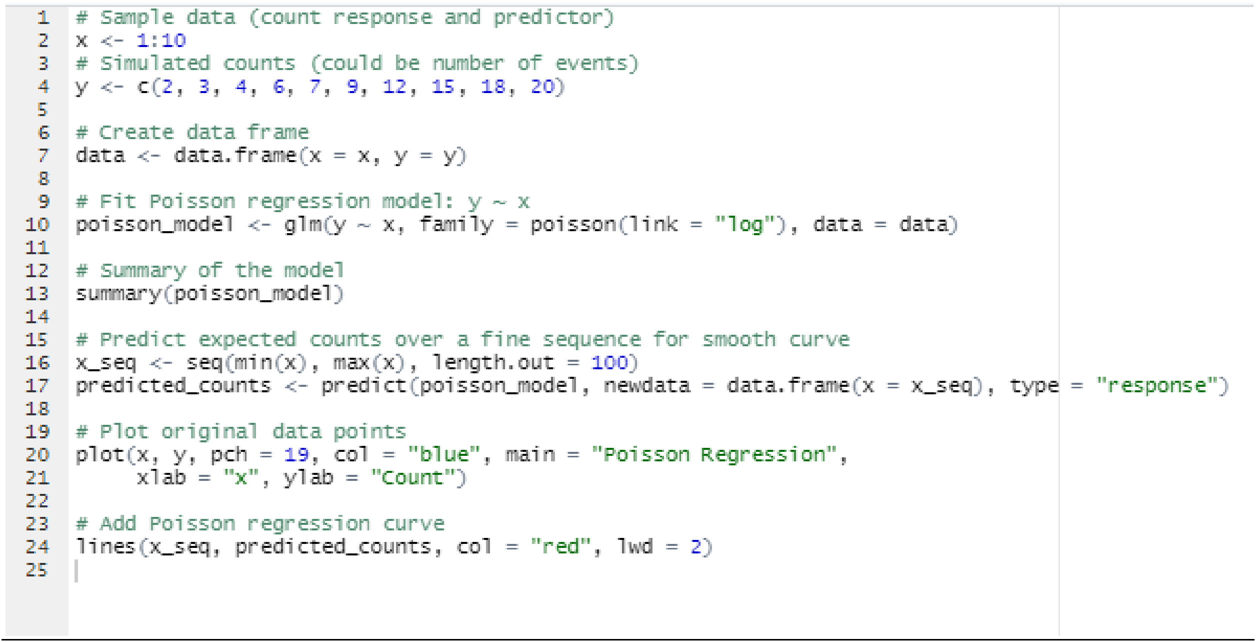
```
# Add logistic regression curve
lines(x_values, predicted_probs, col = "red", lwd = 2)
```

- Adds the logistic regression curve (predicted probabilities) to the plot.
- col = "red" colors the curve red.
- lwd = 2 makes the curve line thicker.

---

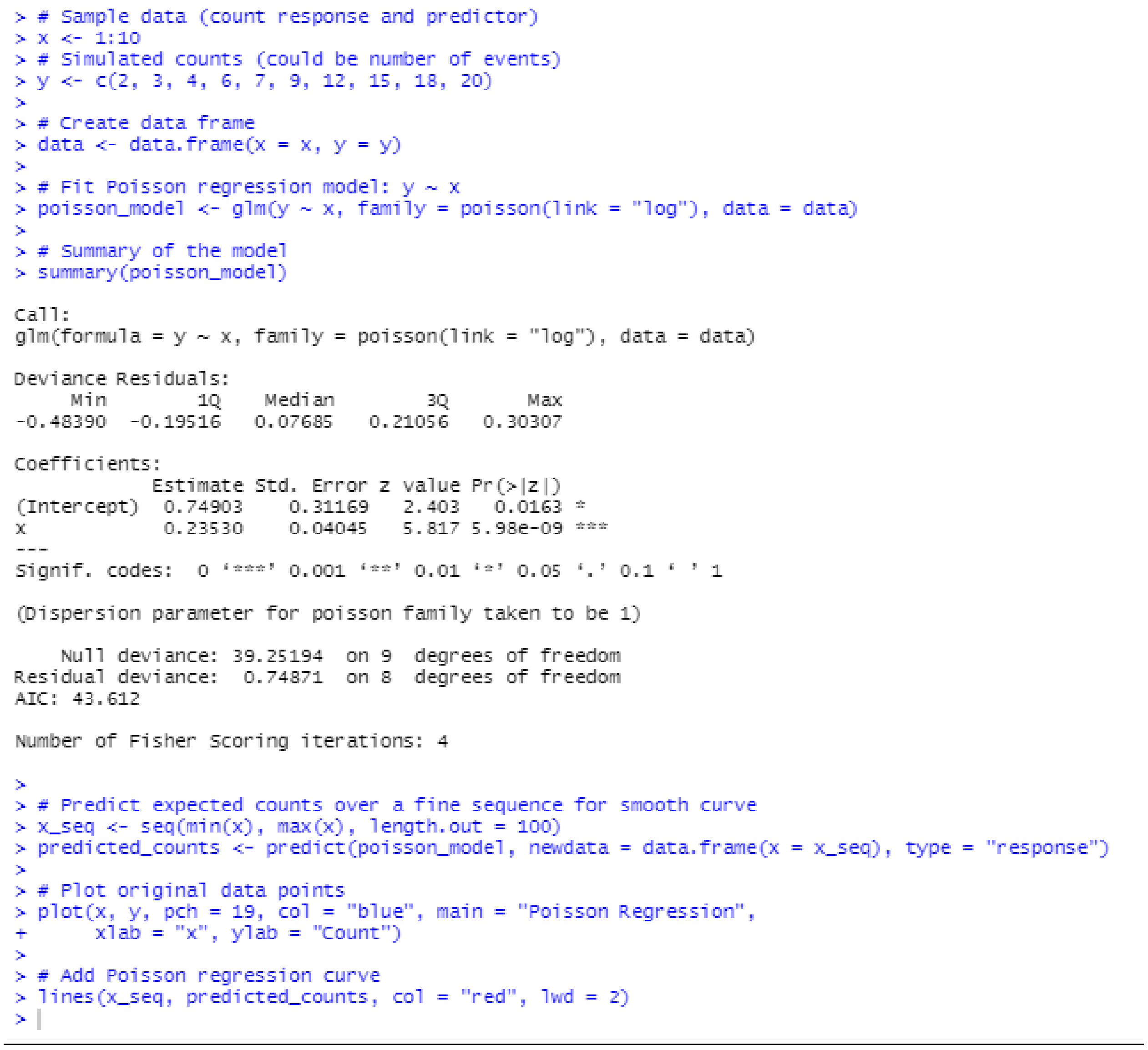
d) Poisson Distribution

Program:

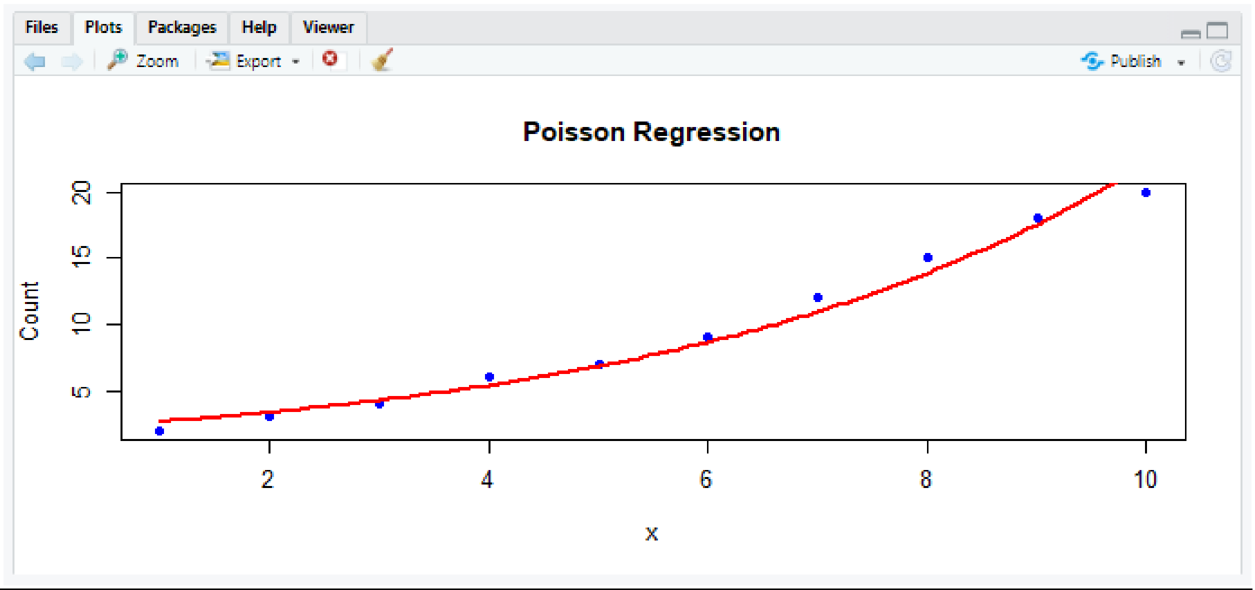


```
# Sample data (count response and predictor)
x <- 1:10
# Simulated counts (could be number of events)
y <- c(2, 3, 4, 6, 7, 9, 12, 15, 18, 20)

# Create data frame
data <- data.frame(x = x, y = y)

# Fit Poisson regression model: y ~ x
poisson_model <- glm(y ~ x, family = poisson(link = "log"), data = data)

# Summary of the model
summary(poisson_model)

# Predict expected counts over a fine sequence for smooth curve
x_seq <- seq(min(x), max(x), length.out = 100)
predicted_counts <- predict(poisson_model, newdata = data.frame(x = x_seq), type = "response")

# Plot original data points
plot(x, y, pch = 19, col = "blue", main = "Poisson Regression",
     xlab = "x", ylab = "Count")

# Add Poisson regression curve
lines(x_seq, predicted_counts, col = "red", lwd = 2)

```

Output:



```
> # Sample data (count response and predictor)
> x <- 1:10
> # Simulated counts (could be number of events)
> y <- c(2, 3, 4, 6, 7, 9, 12, 15, 18, 20)
>
> # Create data frame
> data <- data.frame(x = x, y = y)
>
> # Fit Poisson regression model: y ~ x
> poisson_model <- glm(y ~ x, family = poisson(link = "log"), data = data)
>
> # Summary of the model
> summary(poisson_model)

Call:
glm(formula = y ~ x, family = poisson(link = "log"), data = data)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-0.48390  -0.19516   0.07685   0.21056   0.30307

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.74903    0.31169   2.403   0.0163 *
x            0.23530    0.04045   5.817 5.98e-09 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 39.25194  on 9  degrees of freedom
Residual deviance:  0.74871  on 8  degrees of freedom
AIC: 43.612

Number of Fisher Scoring iterations: 4

>
> # Predict expected counts over a fine sequence for smooth curve
> x_seq <- seq(min(x), max(x), length.out = 100)
> predicted_counts <- predict(poisson_model, newdata = data.frame(x = x_seq), type = "response")
>
> # Plot original data points
> plot(x, y, pch = 19, col = "blue", main = "Poisson Regression",
+      xlab = "x", ylab = "Count")
>
> # Add Poisson regression curve
> lines(x_seq, predicted_counts, col = "red", lwd = 2)
> |
```



Explanation:

# Sample data (count response and predictor)
x <- 1:10

- Creates a numeric vector x containing integers from 1 to 10.
- This will be the predictor (independent variable).

# Simulated counts (could be number of events)
y <- c(2, 3, 4, 6, 7, 9, 12, 15, 18, 20)

- Creates a numeric vector y with count data (dependent variable).
- These could represent counts of events observed for each corresponding x.

# Create data frame
data <- data.frame(x = x, y = y)

- Combines vectors x and y into a data frame named data.
- Data frames are the standard format for modeling functions in R.

# Fit Poisson regression model: y ~ x
poisson_model <- glm(y ~ x, family = poisson(link = "log"), data = data)

- Fits a Poisson generalized linear model (GLM) predicting y using x.
- family = poisson(link = "log") specifies a Poisson distribution with a log link function, appropriate for count data.
- The fitted model is saved in the variable poisson_model.

# Summary of the model
summary(poisson_model)

- Displays a detailed summary of the Poisson regression model.

- Includes coefficient estimates, standard errors, z-values, p-values, and model fit statistics.

```
# Predict expected counts over a fine sequence for smooth curve
x_seq <- seq(min(x), max(x), length.out = 100)
```

- Creates a sequence `x_seq` of 100 evenly spaced values from the minimum to the maximum of `x`.
- Used to generate smooth predicted values for plotting the regression curve.

```
predicted_counts <- predict(poisson_model, newdata = data.frame(x = x_seq), type = "response")
```

- Uses the fitted model to predict expected counts (`y`) for each value in `x_seq`.
- `type = "response"` returns predictions on the original count scale (not on the log scale).

```
# Plot original data points
plot(x, y, pch = 19, col = "blue", main = "Poisson Regression",
     xlab = "x", ylab = "Count")
```

- Creates a scatter plot of the original data points.
- `pch = 19` sets the plotting character to solid circles.
- `col = "blue"` colors the points blue.
- `main`, `xlab`, and `ylab` set the plot title and axis labels.

```
# Add Poisson regression curve
lines(x_seq, predicted_counts, col = "red", lwd = 2)
```

- Adds a line showing the predicted Poisson regression curve.
- `col = "red"` colors the line red.
- `lwd = 2` makes the line thicker for better visibility.
