Statistics Software Lab Report - 9

Name of the Student: Shatansh Patnaik
Roll No: 20MA20067

IIT Kharagpur
Statistics Software Lab

Linear Regression

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship between the variables, aiming to find the best-fitting line that predicts the dependent variable based on the independent variables.

Algorithm to Compute Coefficients

The following algorithm outlines the steps to compute the coefficients of a linear regression model using Ordinary Least Squares (OLS) method:

Algorithm 1 Compute Coefficients of Linear Regression Model

procedure OLS($(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, n)
    Calculate the means: $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$, $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$
    Calculate the slope (coefficient): $\beta_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$
    Calculate the intercept (coefficient): $\beta_0 = \bar{y} - \beta_1\bar{x}$
    return β_0, β_1
end procedure

Analysis of the Algorithm

The algorithm employs the Ordinary Least Squares (OLS) method to estimate the coefficients of the linear regression model. It starts by calculating the means of the independent and dependent variables. Then, it computes the slope (coefficient) of the regression line by finding the ratio of the covariance of the variables to the variance of the independent variable. Finally, it calculates the intercept (coefficient) using the means and the slope. The resulting coefficients represent the parameters of the linear regression model.

Computation of Confidence Intervals of the Parameters of the Model

Confidence intervals for the parameters of a linear regression model provide a range of values within which we can be reasonably confident that the true value of the parameter lies. These intervals are essential for assessing the uncertainty associated with the estimated coefficients of the model. The computation involves estimating the standard error of the coefficients and then using critical values from the t-distribution to determine the bounds of the interval.

Algorithm for the Computation of Confidence Intervals of the Parameters of the Model

The following algorithm outlines the steps for computing confidence intervals for the parameters of a linear regression model:

Algorithm 2 Compute Confidence Intervals

```
procedure ConfidenceIntervals(β̂_0, β̂_1, ..., β̂_n, SE(β̂_0), SE(β̂_1), ..., SE(β̂_n), df)
    Set desired confidence level (e.g., 95%)
    Find critical value t* from t-distribution with df degrees of freedom
    for i ← 0 to n do
        Calculate margin of error ME_i = t* × SE(β̂_i)
        Compute confidence interval [β̂_i − ME_i, β̂_i + ME_i]
    end for
    return Confidence intervals for each parameter
end procedure
```

This algorithm provides a systematic approach to calculate confidence intervals for the parameters of a linear regression model, allowing researchers to make inferences about the true values of the coefficients with a specified level of confidence.

Significance Testing

Significance testing for the parameters of a linear regression model assesses whether the estimated coefficients are significantly different from zero. This helps determine the importance of each predictor variable in explaining the variation in the dependent variable. The computation involves comparing the t-statistic for each coefficient to a critical value from the t-distribution at a specified significance level (e.g., $\alpha = 0.05$).

Algorithm for performing significance testing

The following algorithm outlines the steps for conducting significance testing for the parameters of a linear regression model:

Algorithm 3 Significance Testing

```
procedure SignificanceTesting(β̂_0, β̂_1, ..., β̂_n, SE(β̂_0), SE(β̂_1), ..., SE(β̂_n), df, α)
    for i ← 0 to n do
        Calculate the t-statistic: t_i = β̂_i / SE(β̂_i)
        Find critical value t* from t-distribution with df degrees of freedom at significance level α
        if |t_i| > t* then
            β̂_i is statistically significant
        else
            β̂_i is not statistically significant
        end if
    end for
end procedure
```

This algorithm provides a systematic approach to conduct significance testing for the parameters of a linear regression model. It helps identify which predictors have a significant impact on the dependent variable, allowing researchers to make informed decisions about the model's explanatory power.

Lack of Fitness Test

The lack of fitness test, also known as lack-of-fit test, assesses whether a simple linear regression model adequately fits the observed data. It is used to determine whether there is significant evidence that the relationship between the dependent and independent variables is not adequately described by a linear model. The test compares the variation explained by the regression model to the residual variation not explained by the model.

Algorithm for performing lack of fitness test

The following algorithm outlines the steps for conducting the lack of fitness test for a simple linear regression model:

Algorithm 4 Lack of Fitness Test

procedure LackOfFitnessTest($\hat{y}_1, \hat{y}_2, ..., \hat{y}_n,\ y_1, y_2, ..., y_n$)
    Calculate the total sum of squares (TSS): $TSS = \sum_{i=1}^{n}(y_i - \bar{y})^2$
    Calculate the regression sum of squares (RSS): $RSS = \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2$
    Calculate the residual sum of squares (ESS): $ESS = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
    Calculate the lack of fit sum of squares (LFSS): $LFSS = ESS - RSS$
    Calculate the degrees of freedom for the lack of fit: $df_{LF} = n - 2$
    Calculate the mean square for lack of fit: $MS_{LF} = \frac{LFSS}{df_{LF}}$
    Calculate the mean square error: $MSE = \frac{ESS}{n-1}$
    Perform an F-test using $F = \frac{MS_{LF}}{MSE}$ and df_{LF} and $n - 1$ degrees of freedom
    if F is significant **then**
        Lack of fit is significant; the linear model does not adequately fit the data
    else
        Lack of fit is not significant; the linear model adequately fits the data
    end if
end procedure

Conclusion

The lack of fitness test is an important diagnostic tool for evaluating the adequacy of a simple linear regression model. By comparing the variation explained by the model to the residual variation, researchers can determine whether the linear model adequately captures the relationship between the variables. If the lack of fit is significant, it indicates that the model does not adequately explain the data, and alternative modeling approaches may be necessary.

Correlation analysis

Covariance analysis for a linear regression model assesses the relationships between variables and their contributions to predicting the dependent variable. It involves analyzing the covariance matrix to understand the relationships between predictors and to identify multicollinearity, which can affect the model's stability and interpretation.

The following algorithm outlines the steps for covariance analysis for a linear regression model:

Algorithm 5 Covariance Analysis for Linear Regression Model

```
procedure CovarianceAnalysis(X_1, X_2, ..., X_n)
    Compute covariance matrix Cov(X) from the predictor variables
    Compute correlation matrix Corr(X) from the covariance matrix
    for each pair of predictors (X_i, X_j) do
        if |Corr(X_i, X_j)| > threshold then
            Identify multicollinearity
        end if
    end for
    Analyze eigenvalues of Corr(X) to determine magnitude of multicollinearity
    Perform diagnostic tests and address multicollinearity if present
end procedure
```

Covariance analysis is an essential step in assessing the quality of a linear regression model. By examining the relationships between predictors and identifying multicollinearity, researchers can ensure the stability and reliability of their regression analysis.

Plotting of Scatter Plots for the Given Datasets

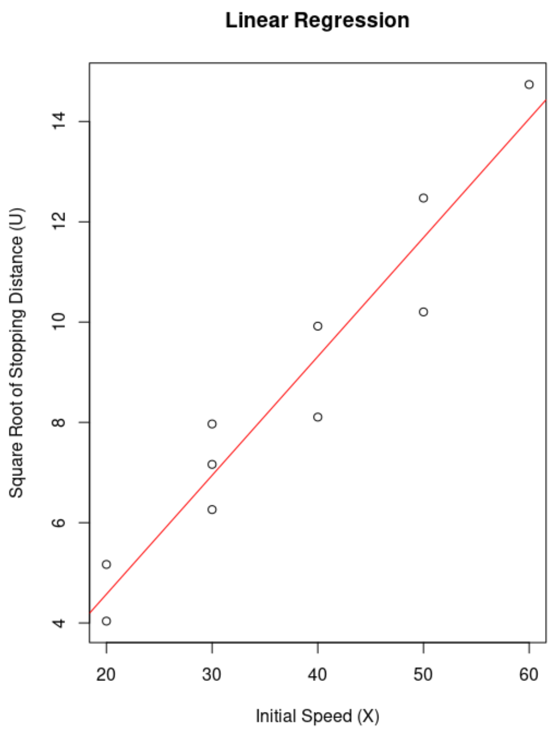

Figure 1: Scatter Plot for Y vs X

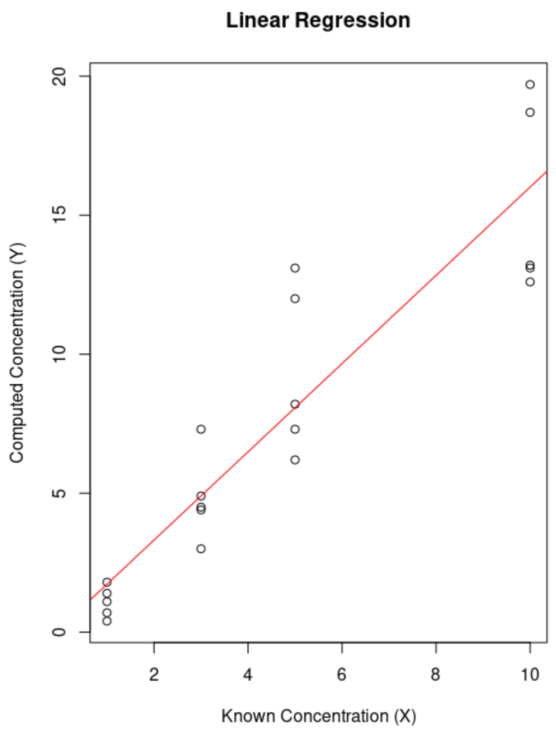

Figure 2: Scatter Plot for Y vs X

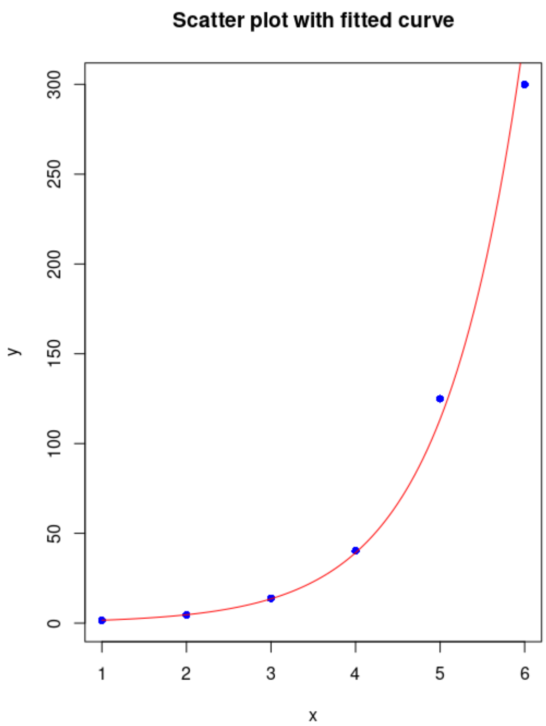

Figure 3: Scatter Plot for Y vs X

Implementation of the above algorithms using R

The following code demonstrates the implementation of computation of parameters of the model, computation of the respective confidence and prediction interval, significance testing and confidence interval testing.

Code for Exercise-1

```
# Exercise - 1
X <- c(20, 20, 30, 30, 30, 40, 40, 50, 50, 60)
y <- c(16.3, 26.7, 39.2, 63.5, 51.3, 98.4, 65.7, 104.1, 155.6, 217.2)

# a) Scatter plot
plot(X, y, xlab = "X", ylab = "y", main = "Scatter plot")

# b) Fitting a linear regression model
U <- sqrt(y)
meanX <- sum(X) / length(X)
meanU <- sum(U) / length(U)

```

```r
beta1 <- sum((X - meanX) * (U - meanU)) / sum((X - meanX)^2)
beta0 <- meanU - beta1 * meanX
UPredicted <- beta0 + beta1 * X
cat("Intercept (beta0):", beta0, "\n")
cat("Slope (beta1):", beta1, "\n")

plot(X, U, xlab = "Initial Speed (X)", ylab = "Square Root of Stopping
    Distance (U)", main = "Linear Regression")
abline(beta0, beta1, col = "red")

# c) Confidence Intervals for beta0, beta1 and sigma squared
n <- length(X)
df <- n - 2
RSS <- sum((U - UPredicted)^2)
sigma2 <- RSS / df
SESlope <- sqrt(sigma2 / sum((X - meanX)^2))
SEIntercept <- sqrt(sigma2 * (1 / n + meanX^2 / sum((X - meanX)^2)))

t <- qt(0.975, df)
CIForSlope <- c(beta1 - t * SESlope, beta1 + t* SESlope)
CIForIntercept <- c(beta0 - t * SEIntercept, beta0 + t * SEIntercept)

chiLower <- qchisq(0.025, df)
chiUpper <- qchisq(0.975, df)

CISigma2 <- c((df * sigma2) / chiUpper, (df * sigma2) / chiLower)

cat("95% Confidence Interval for Slope (beta1):", CIForSlope, "\n")
cat("95% Confidence Interval for Intercept (beta0):", CIForIntercept, "\n")
cat("95% Confidence Interval for Sigma^2:", CISigma2, "\n")

# d) Test of Significance for beta1 and beta0
# For Beta0:
alpha <- 0.05
SSE <- sum((U - UPredicted)^2)
StandardErrorBeta0 <- sqrt((SSE / df) * ((1 / n) + (meanX^2 / sum((X - meanX
    )^2))))
tObserved <- beta0 / StandardErrorBeta0
pValue <- 2 * pt(abs(tObserved), n-2, lower.tail = FALSE)

if (pValue < alpha) {
  cat("Reject null hypothesis: Beta0 is significant.\n")
} else {
  cat("Accept the null hypothesis: Beta0 is not significant.\n")
}

# For Beta1
alpha <- 0.05
df <- n-2
SSE <- sum((U - UPredicted)^2)
StandardErrorBeta1 <- sqrt((SSE / df) / sum((X - meanX)^2))
tObserved <- beta1 / StandardErrorBeta1
```

```
pValue <- 2 * pt(abs(tObserved), n-2, lower.tail = FALSE)

if (pValue < alpha) {
  cat("Reject null hypothesis: Beta1 is significant.\n")
} else {
  cat("Accept the null hypothesis: Beta1 is not significant.\n")
}

# e) Confidence Interval for mean of y given X
X0 <- 35
UPredictedForX0 <- beta0 + beta1 * X0
SEPred <- sqrt(sigma2 * ( 1 / n + (X0 - meanX)^2 / sum((X - meanX)^2)))
df <- length(X) - 2
t <- qt(0.975, df)

CIForU <- c(UPredictedForX0- t * SEPred, UPredictedForX0 + t * SEPred)
CIForY <- CIForU^2
cat("95% Confidence Interval for the expected stopping distance when the
    initial speed is 35:", CIForY, "\n")

# f) Prediction Interval for a new observation of y given X
X0 <- 35
UPredictedForX0 <- beta0 + beta1 * X0
SEPred <- sqrt(sigma2 * (1 + 1 / n + (X0 - meanX)^2 / sum((X - meanX)^2)))
df <- length(X) - 2
t <- qt(0.975, df)
PIForU <- c(UPredictedForX0- t * SEPred, UPredictedForX0 + t * SEPred)

PIForY <- PIForU^2
cat("95% Confidence Interval for the expected stopping distance when the
    initial speed is 35:", PIForY, "\n")

# g) Lack of fit analysis
n <- length(X)
m <- length(unique(X))
SSE <- sum((U - beta0 - beta1*X)^2)

SSPE <- 0
for(i in 1:m){
  SSPE <- SSPE + sum((sqrt(y[X==unique(X)[i]]) - mean(sqrt(y[X==unique(X)[i
      ]])))^2)
}

SSLOF <- SSE - SSPE

fStat <- (SSLOF/(m-2))/(SSPE/(n-m))
fCritical <- qf(0.95,m-2,n-m)

if(fStat > fCritical){
  cat("The model caught lacking","\n")
}else{
  cat("The model doesnt not lack in fitting the data","\n")
```

```
}
```

Code for Exercise-2

```
# Exercise - 2
n <- 20
X <- c(1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 5, 5, 5, 5, 5, 10, 10, 10, 10, 10)
y <- c(1.1, 0.7, 1.8, 0.4, 1.4, 3.0, 4.5, 4.9, 4.4, 7.3, 7.3, 8.2, 6.2,
    13.1, 12.0, 12.6, 13.2, 13.1, 18.7, 19.7)
plot(X, y, xlab = "X", ylab = "y", main = "Scatter plot")

# Fitting a linear regression model
meanX <- sum(X) / length(X)
meanY <- sum(y) / length(y)

beta1 <- sum((X - meanX) * (y - meanY)) / sum((X - meanX)^2)
beta0 <- meanY - beta1 * meanX
YPredicted <- beta0 + beta1 * X
cat("Intercept (beta0):", beta0, "\n")
cat("Slope (beta1):", beta1, "\n")

plot(X, y, xlab = "Known Concentration (X)", ylab = "Computed
    Concentration (Y)", main = "Linear Regression")
abline(beta0, beta1, col = "red")

# Lack of fit test
m <- length(unique(X))
SSE <- sum((y - beta0 - beta1*X)^2)

SSPE <- 0
for(i in 1:m){
  SSPE <- SSPE + sum(((y[X==unique(X)[i]]) - mean((y[X==unique(X)[i]])))
      ^2)
}

SSLOF <- SSE - SSPE

fStat <- (SSLOF/(m-2))/(SSPE/(n-m))
fCritical <- qf(0.95,m-2,n-m)

if(fStat > fCritical){
  cat("The model caught lacking","\n")
}else{
  cat("The model doesnt not lack in fitting the data","\n")
}

# Correlation Coefficient
correlation <- (n * sum(X * y) - sum(X) * sum(y)) / sqrt((n * sum(X^2) -
     sum(X)^2) * (n * sum(y^2) - sum(y)^2))
cat("Correlation Coefficient:", correlation, "\n")

```

```
# Test of significance for beta1 and beta0
# For Beta0:
alpha <- 0.05
df <- n-2
SSE <- sum((y - YPredicted)^2)
StandardErrorBeta0 <- sqrt((SSE / df) * ((1 / n) + (meanX^2 / sum((X -
   meanX)^2))))
tObserved <- beta0 / StandardErrorBeta0
pValue <- 2 * pt(abs(tObserved), n-2, lower.tail = FALSE)

if (pValue < alpha) {
  cat("Reject null hypothesis: Beta0 is significant.\n")
} else {
  cat("Accept the null hypothesis: Beta0 is not significant.\n")
}

# For Beta1
alpha <- 0.05
SSE <- sum((y - YPredicted)^2)
StandardErrorBeta1 <- sqrt((SSE / df) / sum((X - meanX)^2))
tObserved <- beta1 / StandardErrorBeta1
pValue <- 2 * pt(abs(tObserved), n-2, lower.tail = FALSE)

if (pValue < alpha) {
  cat("Reject null hypothesis: Beta1 is significant.\n")
} else {
  cat("Accept the null hypothesis: Beta1 is not significant.\n")
}

# Confidence Intervals for all the parameters
sigma2 <- SSE / df
SESlope <- sqrt(sigma2 / sum((X - meanX)^2))
SEIntercept <- sqrt(sigma2 * (1 / n + meanX^2 / sum((X - meanX)^2)))

t <- qt(0.975, df)
CIForSlope <- c(beta1 - t * SESlope, beta1 + t* SESlope)
CIForIntercept <- c(beta0 - t * SEIntercept, beta0 + t * SEIntercept)

chiLower <- qchisq(0.025, df)
chiUpper <- qchisq(0.975, df)

CISigma2 <- c((df * sigma2) / chiUpper, (df * sigma2) / chiLower)

cat("95% Confidence Interval for Slope (beta1):", CIForSlope, "\n")
cat("95% Confidence Interval for Intercept (beta0):", CIForIntercept, "\
   n")
cat("95% Confidence Interval for Sigma^2:", CISigma2, "\n")
```

Code for Exercise-4

```
# Exercise - 4
```

```r
X <- c(1, 2, 3, 4, 5, 6)
y <- c(1.60, 4.50, 13.80, 40.20, 125.00, 300.00)
plot(X, y, pch = 16, col = "blue", xlab = "x", ylab = "y", main = "Scatter
    plot with fitted curve")
n <- length(X)

Y <- log(y)
# We note that beta0 is log a and b is beta1
meanX <- sum(X) / length(X)
meanY <- sum(Y) / length(Y)

beta1 <- sum((X - meanX) * (Y - meanY)) / sum((X - meanX)^2)
beta0 <- meanY - beta1 * meanX
a <- exp(beta0)

yPredicted <- beta0 + beta1 * X

# Lack of Fitness Test
SSLF <- sum((Y - yPredicted)^2)
SSE <- sum((Y - meanY)^2)

df1 <- n - 2
df2 <- n - 3
F_statistic <- (SSLF / df1) / (SSE / df2)

alpha <- 0.05
critical_value <- qf(1 - alpha, df1, df2)

cat("F-statistic:", F_statistic, "\n")
cat("Critical value:", critical_value, "\n")

if (F_statistic > critical_value) {
  cat("Reject null hypothesis: Lack of fit is significant.\n")
} else {
  cat("Accept the null hypothesis: Lack of fit is not significant.\n")
}

# Plotting
curve(a * exp(beta1 * x), from = min(X), to = max(X), col = "red", add =
    TRUE)

# Significance Testing for Parameters
# For Beta0:
alpha <- 0.05
SSE <- sum((Y - yPredicted)^2)
StandardErrorBeta0 <- sqrt((SSE / df2) * ((1 / n) + (meanX^2 / sum((X -
    meanX)^2))))
tObserved <- beta0 / StandardErrorBeta0
pValue <- 2 * pt(abs(tObserved), n-2, lower.tail = FALSE)

if (pValue < alpha) {
  cat("Reject null hypothesis: Beta0 is significant.\n")
```

```
} else {
  cat("Accept the null hypothesis: Beta0 is not significant.\n")
}

# For Beta1
alpha <- 0.05
df <- n-2
SSE <- sum((Y - yPredicted)^2)
StandardErrorBeta1 <- sqrt((SSE / df) / sum((X - meanX)^2))
tObserved <- beta1 / StandardErrorBeta1
pValue <- 2 * pt(abs(tObserved), n-2, lower.tail = FALSE)

if (pValue < alpha) {
  cat("Reject null hypothesis: Beta1 is significant.\n")
} else {
  cat("Accept the null hypothesis: Beta1 is not significant.\n")
}
```

Code for Exercise-5

```
# Exercise - 5
X <- c(2, 3, 4, 5, 6)
y <- c(144.0, 172.80, 207.40, 248.50, 298.50)
plot(X, y, pch = 16, col = "blue", xlab = "X", ylab = "Y", main = "Scatter
    plot with fitted curve")

Y <- log(y)
n <- length(X)


# We note that beta0 is log a and b is exp(beta1)
meanX <- sum(X) / length(X)
meanY <- sum(Y) / length(Y)

beta1 <- sum((X - meanX) * (Y - meanY)) / sum((X - meanX)^2)
beta0 <- meanY - beta1 * meanX
a <- exp(beta0)
b <- exp(beta1)

yPredicted <- beta0 + beta1 * X

# Lack of Fitness Test
SSLF <- sum((Y - yPredicted)^2)
SSE <- sum((Y - meanY)^2)

df1 <- n - 2
df2 <- n - 3
F_statistic <- (SSLF / df1) / (SSE / df2)

alpha <- 0.05
critical_value <- qf(1 - alpha, df1, df2)
```

```
cat("F-statistic:", F_statistic, "\n")
cat("Critical value:", critical_value, "\n")

if (F_statistic > critical_value) {
  cat("Reject null hypothesis: Lack of fit is significant.\n")
} else {
  cat("Accept the null hypothesis: Lack of fit is not significant.\n")
}

# Significance Testing for Parameters
# For Beta0:
alpha <- 0.05
SSE <- sum((Y - yPredicted)^2)
StandardErrorBeta0 <- sqrt((SSE / df1) * ((1 / n) + (meanX^2 / sum((X -
    meanX)^2))))
tObserved <- beta0 / StandardErrorBeta0
pValue <- 2 * pt(abs(tObserved), n-2, lower.tail = FALSE)

if (pValue < alpha) {
  cat("Reject null hypothesis: Beta0 is significant.\n")
} else {
  cat("Accept the null hypothesis: Beta0 is not significant.\n")
}

# For Beta1
alpha <- 0.05
SSE <- sum((Y - yPredicted)^2)
StandardErrorBeta1 <- sqrt((SSE / df1) / sum((X - meanX)^2))
tObserved <- beta1 / StandardErrorBeta1
pValue <- 2 * pt(abs(tObserved), n-2, lower.tail = FALSE)

if (pValue < alpha) {
  cat("Reject null hypothesis: Beta1 is significant.\n")
} else {
  cat("Accept the null hypothesis: Beta1 is not significant.\n")
}
```

Polynomial Regression

Polynomial regression is a type of regression analysis in which the relationship between the independent variable x and the dependent variable y is modeled as an nth degree polynomial. It is used when the relationship between the variables is nonlinear, and a straight line cannot adequately capture the relationship.

Algorithm 6 Polynomial Regression Algorithm

Input: Data points $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ and degree of polynomial d
Output: Coefficients $a_0, a_1, ..., a_d$ of the polynomial regression model $y = a_0 + a_1 x + a_2 x^2 + ... + a_d x^d$
Initialize matrix X with dimensions $n \times (d + 1)$
for $i \leftarrow 1$ to n **do**
    for $j \leftarrow 0$ to d **do**
        $X[i, j] \leftarrow x_i^j$
    end for
end for
Compute the transpose of X: X^T
Compute the product of X^T and X: $X^T X$
Compute the inverse of $X^T X$: $(X^T X)^{-1}$
Compute the product of $(X^T X)^{-1}$ and X^T: $(X^T X)^{-1} X^T$
Compute the product of $(X^T X)^{-1} X^T$ and y: $(X^T X)^{-1} X^T y$
return Coefficients $a_0, a_1, ..., a_d$

Algorithm for the Computation of Polynomial Regression Coefficients

Analysis of the Algorithm

The algorithm for computing the coefficients of polynomial regression follows the least squares method. It involves constructing a design matrix X where each row corresponds to a data point and each column corresponds to a power of the independent variable x. Then, the coefficients are computed by performing matrix operations involving the transpose and inverse of the design matrix.

Algorithm for the Computing Confidence Intervals for Polynomial Regression Parameters

Algorithm 7 Confidence Intervals for Polynomial Regression Parameters

Input: Data points $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, polynomial regression coefficients $a_0, a_1, ..., a_d$, significance level α
Output: Confidence intervals for the polynomial regression coefficients $a_0, a_1, ..., a_d$
Compute the residuals: $e_i = y_i - (a_0 + a_1 x_i + a_2 x_i^2 + ... + a_d x_i^d)$
Compute the mean squared error: $MSE = \frac{1}{n-d-1} \sum_{i=1}^{n} e_i^2$
Compute the standard error of the coefficients: $SE(a_j) = \sqrt{\frac{MSE}{\sum_{i=1}^{n} (x_i - \bar{x})^{2(j+1)}}}$, where $\bar{x}$ is the mean of the x values
Compute the t-statistic for the given significance level: $t_{\alpha/2, n-d-1}$
Compute the confidence interval for each coefficient: $(a_j - t_{\alpha/2, n-d-1} \times SE(a_j), a_j + t_{\alpha/2, n-d-1} \times SE(a_j))$
return Confidence intervals for the polynomial regression coefficients $a_0, a_1, ..., a_d$

Analysis of the Algorithm

The algorithm for computing confidence intervals for parameters of a polynomial regression model is based on the standard error of the coefficients and the t-statistic. It involves computing the residuals, mean squared error, standard error of the coefficients, and the t-statistic for the given significance level. The confidence intervals are then calculated based on the standard error and t-statistic.

Algorithm: Lack of Fitness Test for Polynomial Regression

Algorithm 8 Lack of Fitness Test

Input: Residuals e_i from the polynomial regression model, degrees of freedom df_1 and df_2
Output: Result of the lack of fitness test
Compute the sum of squared residuals: $SSR = \sum_{i=1}^{n} e_i^2$
Compute the mean squared error: $MSE = \frac{SSR}{df_2}$
Compute the lack of fitness statistic: $F = \frac{MSE}{\text{Mean Squared Error of the Regression}}$
Compute the critical value from the F-distribution: F_{α,df_1,df_2}
if $F > F_{\alpha,df_1,df_2}$ **then**
    return Reject the null hypothesis: Lack of fit is significant
else
    return Accept the null hypothesis: Lack of fit is not significant
end if

Analysis of Lack of Fitness Test

The lack of fitness test in polynomial regression is used to determine whether the polynomial regression model adequately fits the data. It involves computing the lack of fitness statistic, which compares the mean squared error obtained from the model to the mean squared error of the regression. The test then compares this statistic to a critical value from the F-distribution at a given significance level. If the lack of fitness statistic exceeds the critical value, the null hypothesis is rejected, indicating that the lack of fit is significant. Otherwise, if the lack of fitness statistic does not exceed the critical value, the null hypothesis is accepted, indicating that the lack of fit is not significant.

Algorithm for Significance Testing for Polynomial Regression

Algorithm 9 Significance Testing

```
Input: Coefficients a_0, a_1, ..., a_d from the polynomial regression model, standard errors SE(a_0), SE(a_1), ..., SE(a_d), degrees of freedom df
Output: Results of significance testing for the polynomial regression coefficients
for j ← 0 to d do
    Compute the t-statistic: t_j = a_j / SE(a_j)
    Compute the critical value from the t-distribution: t_{α/2,df}
    if |t_j| > t_{α/2,df} then
        return Reject the null hypothesis: a_j is significant
    else
        return Accept the null hypothesis: a_j is not significant
    end if
end for
```

Analysis of Significance Testing

The significance testing for polynomial regression coefficients is used to determine whether each coefficient in the polynomial regression model is statistically significant. It involves computing the t-statistic for each coefficient by dividing the coefficient by its standard error. The test then compares each t-statistic to a critical value from the t-distribution at a given significance level. If the absolute value of the t-statistic exceeds the critical value, the null hypothesis is rejected, indicating that the coefficient is significant. Otherwise, if the absolute value of the t-statistic does not exceed the critical value, the null hypothesis is accepted, indicating that the coefficient is not significant.

Scatter Plot for the Given Dataset

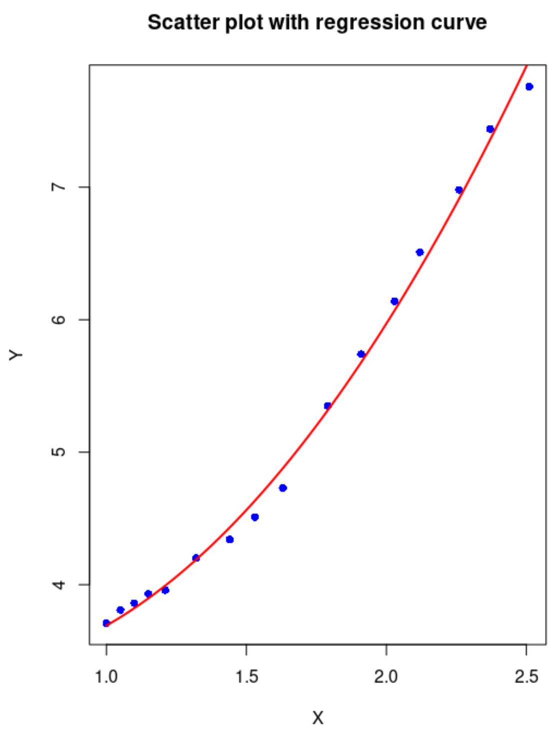

Figure 4: Scatter Plot for Y vs X

Implementation of the above algorithms using R

The following code demonstrates the implementation of computation of parameters of the model, computation of the respective confidence and prediction interval, significance testing and confidence interval testing.

Code for Exercise-3

```
# Exercise - 3
X <- c(1.00, 1.05, 1.10, 1.15, 1.21, 1.32, 1.44, 1.53, 1.63, 1.79, 1.91,
    2.03, 2.12, 2.26, 2.37, 2.51)
y <- c(3.71, 3.81, 3.86, 3.93, 3.96, 4.20, 4.34, 4.51, 4.73, 5.35, 5.74,
    6.14, 6.51, 6.98, 7.44, 7.76)

# a) Fitting a second degree polynomial regression model
XSquared <- X^2
designMatrix <- cbind(rep(1, length(X)), X, XSquared)

productMatrix <- (t(designMatrix)) %*% designMatrix
```

```
productY <- (t(designMatrix)) %*% y
coefficients <- solve(productMatrix, productY)
yPredicted <- coefficients[1] + coefficients[2] * X + coefficients[3] * X^2
cat("Coefficients:\n")
cat("beta_0: ", coefficients[1], "\nbeta_1: ", coefficients[2], "\nbeta_2: "
    , coefficients[3], "\n")


# Plotting of the curve
plot(X, y, pch=16, col="blue", xlab="X", ylab="Y", main="Scatter plot with
    regression curve")
curve(coefficients[1] * 1 + coefficients[2] * x + coefficients[3] * x^2, add
     = TRUE, col = "red", lwd = 2)

residuals <- y - (designMatrix %*% coefficients)
df <- length(y) - ncol(designMatrix)
standardErr <- sqrt(diag(solve(productMatrix)) * sum(residuals^2)/df)

alpha <- 0.05
tObserved <- coefficients / standardErr
t <- qt(1 - alpha/2, df)
lowerBound <- coefficients - t * standardErr
upperBound <- coefficients + t * standardErr


cat("\n95% Confidence Intervals:\n")
for (i in 1:length(coefficients)) {
  cat("beta_", i-1, ": [", lowerBound[i], ", ", upperBound[i], "]\n")
}


# c) Test for significance of coefficients
# Test for significance of each coefficient (using if-else)
for (i in 1:length(coefficients)) {
  if (abs(tObserved[i]) > t) {
    cat("beta_", i-1, "is Significant (p < 0.05)\n")
  } else {
    cat("beta_", i-1, "is Not Significant (p >= 0.05)\n")
  }
}


# d) Computation of R squared
SSRes <- sum((y - yPredicted)^2)

meanY <- mean(y)
SSTotal <- sum((y - meanY)^2)
R2 <- 1 - (SSRes / SSTotal)

cat("Coefficient of determination (R^2):", R2, "\n")
```