PRITHVI YADAV
2021UCA1875

Loading the packages

[1]:
```
# Load the required packages
if (!requireNamespace("caret", quietly = TRUE))
  install.packages("caret")
if (!requireNamespace("ggplot2", quietly = TRUE))
  install.packages("ggplot2")
if (!requireNamespace("ROCR", quietly = TRUE))
  install.packages("ROCR")
if (!requireNamespace("mlbench", quietly = TRUE))
  install.packages("mlbench")

library(caret)
library(ggplot2)
library(ROCR)
library(mlbench)
```

```
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

also installing the dependencies ‘listenv’, ‘parallelly’, ‘future’, ‘globals’,
‘shape’, ‘future.apply’, ‘numDeriv’, ‘progressr’, ‘SQUAREM’, ‘diagram’, ‘lava’,
‘prodlim’, ‘proxy’, ‘iterators’, ‘Rcpp’, ‘clock’, ‘gower’, ‘hardhat’, ‘ipred’,
‘timeDate’, ‘e1071’, ‘foreach’, ‘ModelMetrics’, ‘plyr’, ‘pROC’, ‘recipes’,
‘reshape2’


Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

also installing the dependencies ‘bitops’, ‘gtools’, ‘caTools’, ‘gplots’


Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

Loading required package: ggplot2
```

```
Loading required package: lattice
```

1 Basic Data Structures

R is a programming language and environment for statistical computing and graphics, there are several data structures that are commonly used for organizing and storing data. Some of the primary data structures in R include:

1.0.1 1. Vectors

Vectors are one-dimensional data structures. The only key thing here is all the elements of a vector must be of the identical data type

[2]:
```
# Vectors(ordered collection of same data type)
X = c(1, 3, 5, 7, 8)
```

[3]:
```
X
```

1. 1 2. 3 3. 5 4. 7 5. 8

1.0.2 2. Matrices

Matrices are two-dimensional, homogeneous data structures.It is a rectangular arrangement of numbers in rows and columns. To create a matrix in R you need to use the function called matrix

[4]:
```
A = matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9),
    nrow = 3, ncol = 3, # No of rows and columns
    byrow = TRUE # how to arrange the matrix
)
```

[5]:
```
A
```

A matrix: 3×3 of type dbl

1	2	3
4	5	6
7	8	9

1.0.3 3. Arrays

Arrays are the R data objects which store the data in more than two dimensions. Arrays are n-dimensional data structures. To create an array in R you need to use the function called array()

[6]:
```
# Creating two rectangular matrices each with two rows and two column
A = array(c(1, 2, 3, 4, 5, 6, 7, 8), dim = c(2, 2, 2))
```

[7]:
```
A
```

1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8

1.0.4 4. Lists

A list is a generic object consisting of an ordered collection of objects. Lists are heterogeneous data structures

[8]:
```
empId = c(1, 2, 3, 4)
empName = c("Debi", "Sandeep", "Subham", "Shiba")
numberOfEmp = 4

# Combining all 3 different data types into a list
empList = list(empId, empName, numberOfEmp)
```

[9]:
```
empList
```

1. (a) 1 (b) 2 (c) 3 (d) 4
2. (a) 'Debi' (b) 'Sandeep' (c) 'Subham' (d) 'Shiba'
3. 4

1.0.5 5. Data Frames

These are lists of vectors of equal lengths. Data framesare famous in R because we are comfortable in seeing the data within the tabular form.

Data frames have the following constarins :
* Data Frames must have column names and every row should have a unique name * Each column must have the identical number of items.

- Different columns may have different data types.

[10]:
```
# A character vector
Name = c("Amiya", "Raj", "Asish")

# A character vector
Language = c("R", "Python", "Java")

# A numeric vector
Age = c(22, 25, 45)

# To create dataframe use data.frame command
df = data.frame(Name, Language, Age)
```

[11]:
```
df
```

A data.frame: 3 × 3

Name <chr>	Language <chr>	Age <dbl>
Amiya	R	22
Raj	Python	25
Asish	Java	45

1.0.6 6. Factors

Factors are the data objects which are used to categorize the data and store it as levels. They can store both strings and integers. They are useful to categorize unique values in columns like "TRUE" or "FALSE", or "MALE" or "FEMALE", etc.

[12]:
```
# Creating factor using factor()
fac = factor(c("Male", "Female", "Male", "Male", "Female", "Male", "Female"))
```

[13]:
```
fac
```

1. Male 2. Female 3. Male 4. Male 5. Female 6. Male 7. Female

Levels: 1. 'Female' 2. 'Male'

2 Linear Regression

Linear Regression is a foundational statistical technique used for predicting a continuous outcome based on input features. It assumes a linear relationship between the predictor variables and the target variable. The model estimates coefficients that represent the impact of each feature on the target variable. Linear Regression is simple, interpretable, and provides insights into the magnitude and direction of feature effects. However, it assumes linearity and is sensitive to outliers. Despite its simplicity, Linear Regression remains a valuable tool for tasks where understanding the linear relationship between variables is crucial.

[14]:
```
# Load the CSV file into a data frame
temp_data <- read.csv("temp forecast.csv")

# Display the first few rows
head(temp_data)
```

A data.frame: 6 × 25

	station <int>	Date <chr>	Present_Tmax <dbl>	Present_Tmin <dbl>	LDAPS_RHmin <dbl>	LDAPS_R <dbl>
1	1	2013-06-30	28.7	21.4	58.25569	91.11636
2	2	2013-06-30	31.9	21.6	52.26340	90.60472
3	3	2013-06-30	31.6	23.3	48.69048	83.97359
4	4	2013-06-30	32.0	23.4	58.23979	96.48369
5	5	2013-06-30	31.4	21.9	56.17410	90.15513
6	6	2013-06-30	31.9	23.5	52.43713	85.30725

[15]:
```
# Define a vector of columns to drop
columns_to_drop <- c('station', 'Date', 'LDAPS_PPT2', 'LDAPS_PPT3',␣
↪'LDAPS_PPT4', 'lat', 'lon', 'LDAPS_CC2', 'LDAPS_CC3', 'LDAPS_CC4',␣
↪'LDAPS_PPT1', 'Next_Tmin')

# Drop the specified columns
temp_data <- temp_data[, !(names(temp_data) %in% columns_to_drop)]

# Display the modified data
```

```
head(temp_data)
```

A data.frame: 6 × 13

	Present_Tmax <dbl>	Present_Tmin <dbl>	LDAPS_RHmin <dbl>	LDAPS_RHmax <dbl>	LDAPS_Tma <dbl>
1	28.7	21.4	58.25569	91.11636	28.07410
2	31.9	21.6	52.26340	90.60472	29.85069
3	31.6	23.3	48.69048	83.97359	30.09129
4	32.0	23.4	58.23979	96.48369	29.70463
5	31.4	21.9	56.17410	90.15513	29.11393
6	31.9	23.5	52.43713	85.30725	29.21934

[16]:
```
# Remove rows with missing values
temp_data <- na.omit(temp_data)

# Display the modified data
head(temp_data)
```

A data.frame: 6 × 13

	Present_Tmax <dbl>	Present_Tmin <dbl>	LDAPS_RHmin <dbl>	LDAPS_RHmax <dbl>	LDAPS_Tma <dbl>
1	28.7	21.4	58.25569	91.11636	28.07410
2	31.9	21.6	52.26340	90.60472	29.85069
3	31.6	23.3	48.69048	83.97359	30.09129
4	32.0	23.4	58.23979	96.48369	29.70463
5	31.4	21.9	56.17410	90.15513	29.11393
6	31.9	23.5	52.43713	85.30725	29.21934

[17]:
```
# Create predictor variables (X) by excluding the "Next_Tmax" column
X <- subset(temp_data, select = -c(Next_Tmax))

# Create the target variable (y) with only the "Next_Tmax" column
y <- temp_data$Next_Tmax
```

[18]:
```
# Set the seed for reproducibility
set.seed(1)

# Split the data into training and testing sets
split_index <- createDataPartition(y, p = 0.7, list = FALSE)
X_train <- X[split_index, ]
X_test <- X[-split_index, ]
y_train <- y[split_index]
y_test <- y[-split_index]
```

[19]:
```
# Fit a linear regression model
lm_model <- lm(y_train ~ ., data = cbind(y_train, X_train))

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

```
Call:
lm(formula = y_train ~ ., data = cbind(y_train, X_train))

Residuals:
    Min      1Q  Median      3Q     Max
-7.1956 -0.8757  0.0251  0.9593  5.8031

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)       3.090e+00  5.441e-01   5.680 1.42e-08 ***
Present_Tmax      1.194e-01  1.155e-02  10.335  < 2e-16 ***
Present_Tmin      3.989e-02  1.555e-02   2.565   0.0103 *
LDAPS_RHmin       4.551e-03  3.221e-03   1.413   0.1578
LDAPS_RHmax       4.354e-03  4.232e-03   1.029   0.3037
LDAPS_Tmax_lapse  6.553e-01  1.874e-02  34.961  < 2e-16 ***
LDAPS_Tmin_lapse  1.215e-01  2.390e-02   5.085 3.80e-07 ***
LDAPS_WS         -1.348e-01  1.071e-02 -12.589  < 2e-16 ***
LDAPS_LH          9.689e-03  7.447e-04  13.011  < 2e-16 ***
LDAPS_CC1        -1.588e+00  1.157e-01 -13.725  < 2e-16 ***
DEM              -5.233e-03  6.618e-04  -7.907 3.17e-15 ***
Slope             2.266e-01  2.596e-02   8.727  < 2e-16 ***
Solar.radiation   1.394e-04  5.424e-05   2.570   0.0102 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.55 on 5302 degrees of freedom
Multiple R-squared:  0.7531,        Adjusted R-squared:  0.7526
F-statistic:  1348 on 12 and 5302 DF,  p-value: < 2.2e-16
```

[20]:
```r
# Make predictions on the test set
predictions <- predict(lm_model, newdata = X_test)
```

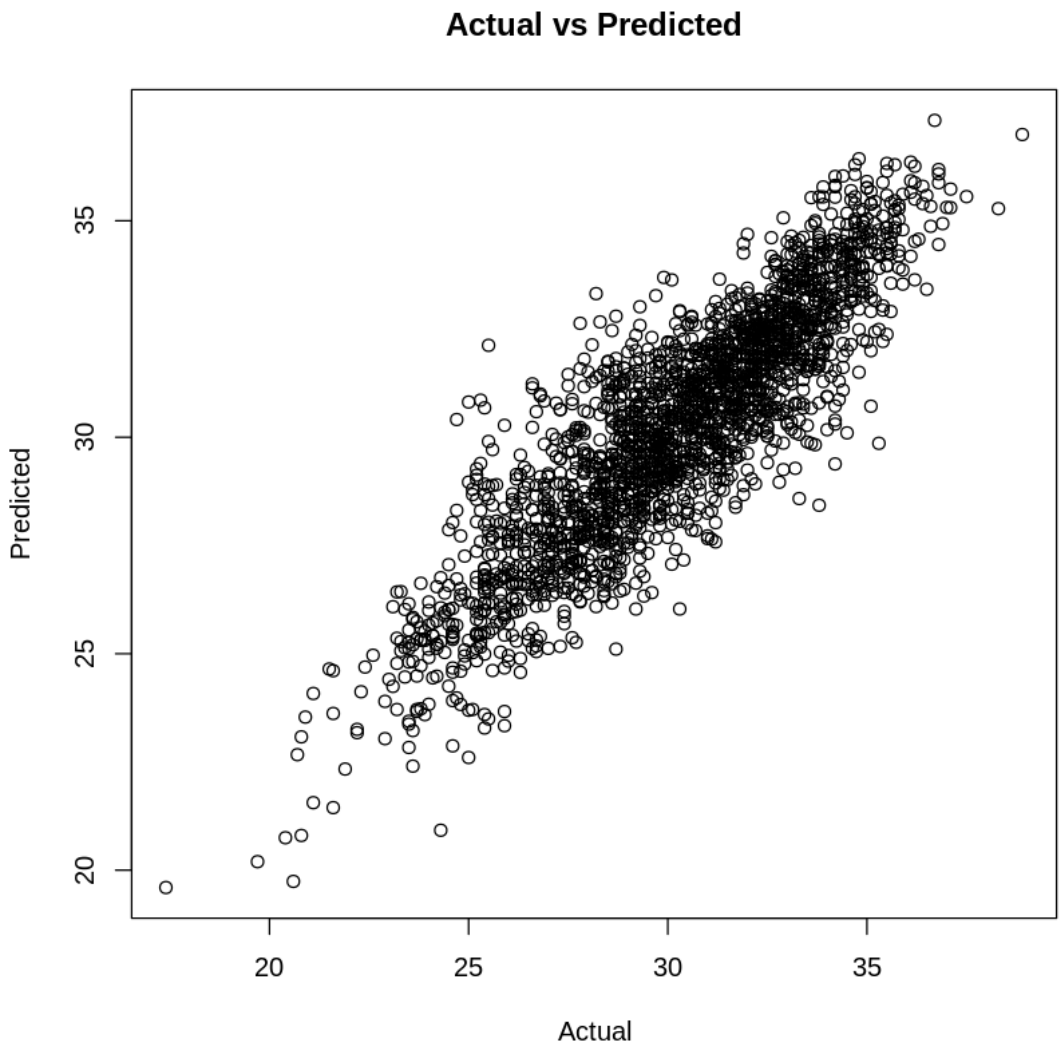
[21]:
```r
# Extract coefficients (slope and intercept)
coefficients <- coef(lm_model)

# Display intercept (c)
intercept <- coefficients["(Intercept)"]
cat("Intercept (c):", intercept, "\n\n")

# Display slopes
slopes <- coefficients[2:length(coefficients)]
cat("Slopes (m) for each predictor variable:\n")
print(slopes)
```

```
Intercept (c): 3.090267

Slopes (m) for each predictor variable:
```

```
    Present_Tmax     Present_Tmin      LDAPS_RHmin      LDAPS_RHmax
    0.1193894227     0.0398851360     0.0045505175     0.0043537088
LDAPS_Tmax_lapse LDAPS_Tmin_lapse         LDAPS_WS         LDAPS_LH
    0.6552638776     0.1215097273    -0.1348340865     0.0096889155
       LDAPS_CC1              DEM            Slope  Solar.radiation
   -1.5880596739    -0.0052331379     0.2265574400     0.0001393925
```

[22]:
```
y_test <- as.numeric(y_test)
predictions <- as.numeric(predictions)
```

[23]:
```
# Create a scatter plot
plot(y_test, predictions, xlab = "Actual", ylab = "Predicted", main = "Actual␣
 ↪vs Predicted")
```



[24]:
```r
# Calculate the R-squared (R2) score
r2_score <- R2(y_test, predictions)

# Display the R2 score
cat("R-squared (R2) Score:", r2_score, "\n")
```

```
R-squared (R2) Score: 0.7742952
```

[25]:
```r
# Calculate the Mean Squared Error (MSE)
mse <- mean((y_test - predictions)^2)

# Display the MSE
cat("Mean Squared Error (MSE):", mse, "\n")
```

```
Mean Squared Error (MSE): 2.192424
```

3 Logistic Regression

Logistic Regression is a widely-used statistical method for binary classification tasks. It models the probability of an event occurring based on input features, employing a logistic function to constrain output between 0 and 1. Logistic Regression is interpretable, computationally efficient, and well-suited for understanding the impact of individual features on the outcome. Regularization techniques, such as L1 and L2 regularization, help prevent overfitting. Its simplicity and effectiveness make Logistic Regression a fundamental tool in predictive modeling, particularly when transparency and ease of interpretation are paramount.

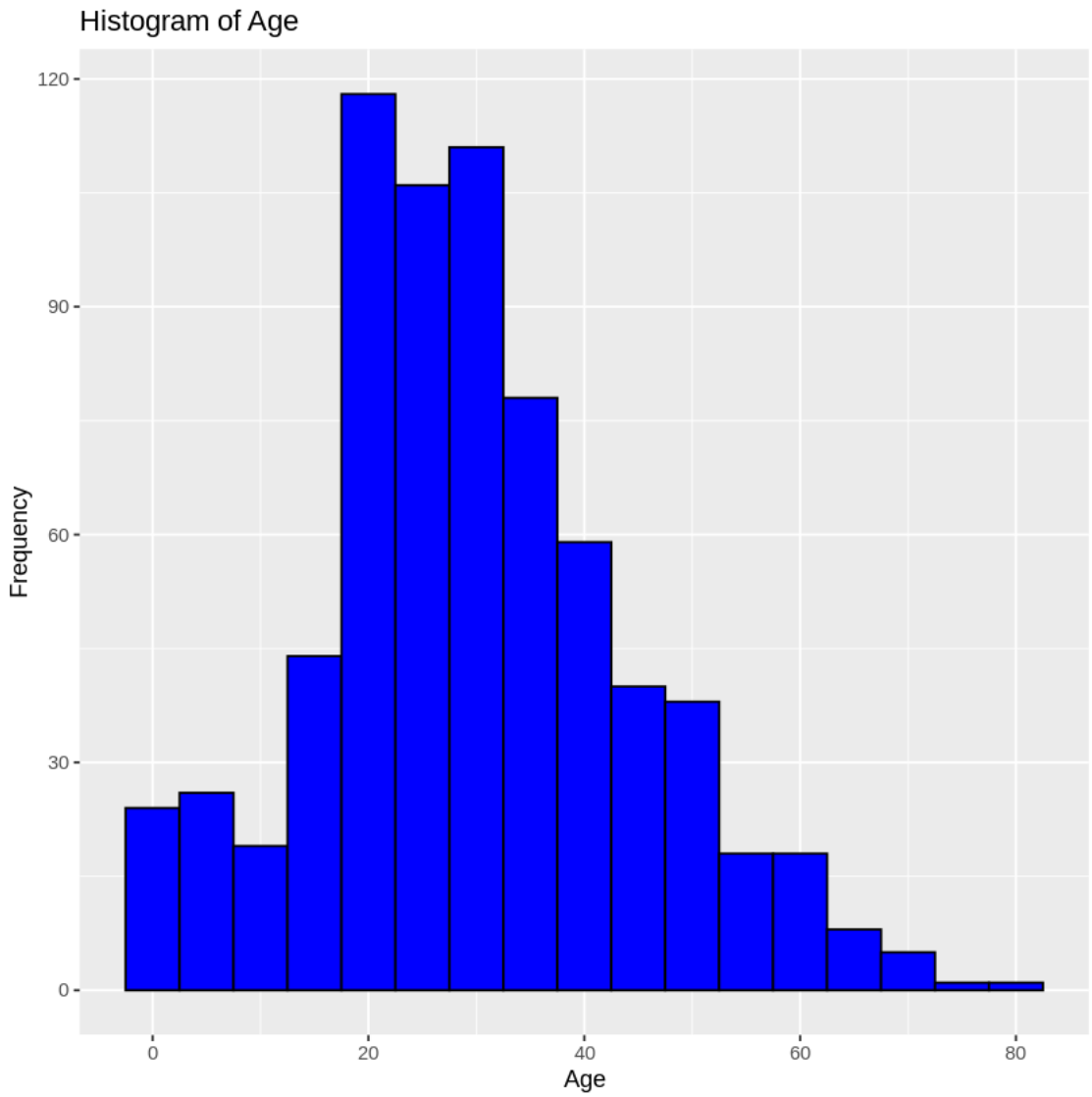
[26]:
```r
# Load the CSV file into a data frame
titanic_data <- read.csv("titanic.csv")

# Display the first few rows
head(titanic_data)
```

A data.frame: 6 × 12

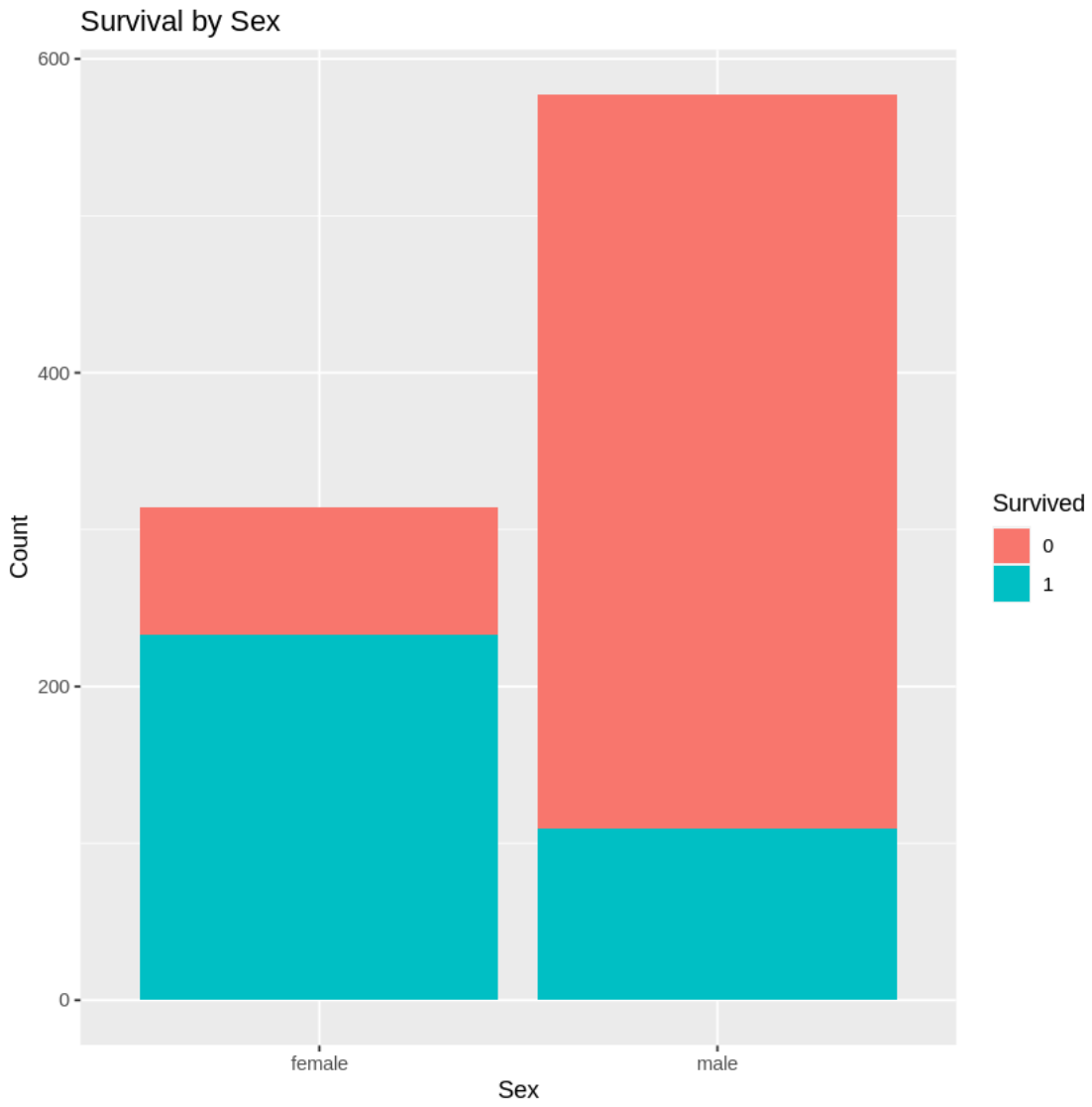
	PassengerId <int>	Survived <int>	Pclass <int>	Name <chr>
1	1	0	3	Braund, Mr. Owen Harris
2	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thaye
3	3	1	3	Heikkinen, Miss. Laina
4	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)
5	5	0	3	Allen, Mr. William Henry
6	6	0	3	Moran, Mr. James

[27]:
```r
# Visualize a histogram of age
ggplot(titanic_data, aes(x = Age)) +
  geom_histogram(binwidth = 5, fill = "blue", color = "black") +
  labs(title = "Histogram of Age", x = "Age", y = "Frequency")
```

```
Warning message:
"Removed 177 rows containing non-finite values (`stat_bin()`)."
```

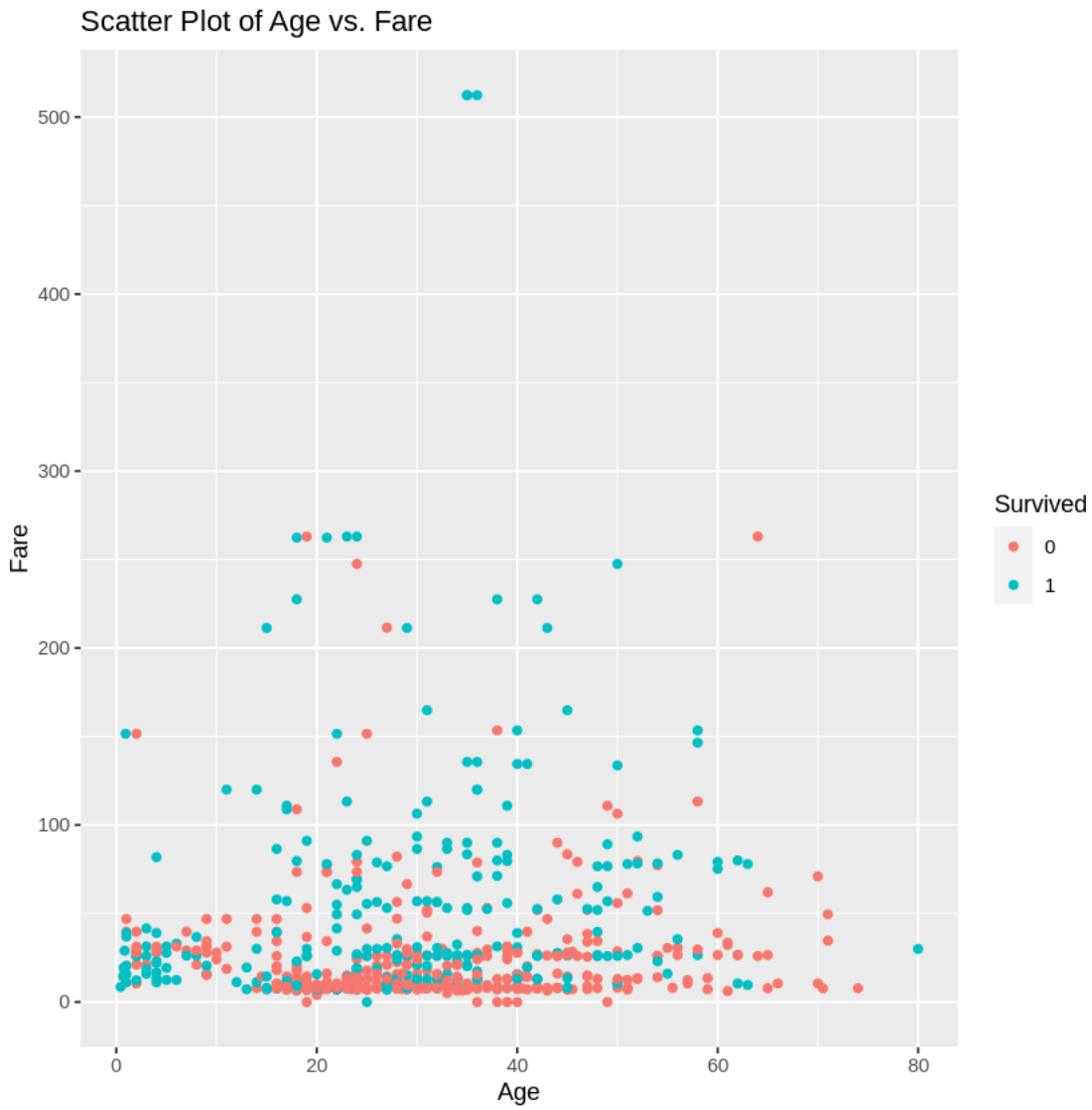


[28]:
```r
# Visualize a bar plot of survival by sex
ggplot(titanic_data, aes(x = Sex, fill = factor(Survived))) +
  geom_bar(position = "stack") +
  labs(title = "Survival by Sex", x = "Sex", y = "Count", fill = "Survived")
```



[29]:
```
# Visualize a scatter plot of age vs. fare
ggplot(titanic_data, aes(x = Age, y = Fare, color = factor(Survived))) +
  geom_point() +
  labs(title = "Scatter Plot of Age vs. Fare", x = "Age", y = "Fare", color =␣
 ↪"Survived")
```

```
Warning message:
"Removed 177 rows containing missing values (`geom_point()`)."
```



[30]:
```
# Drop "Name", "Cabin", "Ticket", "PassengerId", "Fare" and "Age" columns
titanic_data <- subset(titanic_data, select = -c(Name, Cabin, Ticket,␣
 ↪PassengerId, Fare, Age))

# Display the modified data
head(titanic_data)
```

A data.frame: 6 × 6

	Survived <int>	Pclass <int>	Sex <chr>	SibSp <int>	Parch <int>	Embarked <chr>
1	0	3	male	1	0	S
2	1	1	female	1	0	C
3	1	3	female	0	0	S
4	1	1	female	1	0	S
5	0	3	male	0	0	S
6	0	3	male	0	0	Q

[31]:
```
# Remove rows with missing values
titanic_data <- na.omit(titanic_data)

# Display the modified data
head(titanic_data)
```

A data.frame: 6 × 6

	Survived <int>	Pclass <int>	Sex <chr>	SibSp <int>	Parch <int>	Embarked <chr>
1	0	3	male	1	0	S
2	1	1	female	1	0	C
3	1	3	female	0	0	S
4	1	1	female	1	0	S
5	0	3	male	0	0	S
6	0	3	male	0	0	Q

[32]:
```
# One-hot encode "Sex" column
titanic_data$Sex <- as.integer(titanic_data$Sex == "male")

# Rename the "Sex" column to "Male"
colnames(titanic_data)[colnames(titanic_data) == "Sex"] <- "Male"

# Display the modified data
head(titanic_data)
```

A data.frame: 6 × 6

	Survived <int>	Pclass <int>	Male <int>	SibSp <int>	Parch <int>	Embarked <chr>
1	0	3	1	1	0	S
2	1	1	0	1	0	C
3	1	3	0	0	0	S
4	1	1	0	1	0	S
5	0	3	1	0	0	S
6	0	3	1	0	0	Q

[33]:
```
# One-hot encode "Embarked" column
embark <- model.matrix(~ Embarked - 1, data = titanic_data)

# Assign column names to the one-hot encoded matrix
colnames(embark) <- gsub("Embarked", "", colnames(embark))
```

```
# Display the one-hot encoded data
head(embark)
```

A matrix: 6 × 4 of type dbl

	C	Q	S
1	0	0	1
2	0	1	0
3	0	0	1
4	0	0	1
5	0	0	1
6	0	1	0

[34]:
```
# Convert "Pclass" to a factor
titanic_data$Pclass <- as.factor(titanic_data$Pclass)

# One-hot encode "Pclass" column
Pcl <- model.matrix(~ Pclass - 1, data = titanic_data)

# Assign column names to the one-hot encoded matrix
colnames(Pcl) <- gsub("Pclass", "", colnames(Pcl))

# Display the one-hot encoded data
head(Pcl)
```

A matrix: 6 × 3 of type dbl

	1	2	3
1	0	0	1
2	1	0	0
3	0	0	1
4	1	0	0
5	0	0	1
6	0	0	1

[35]:
```
# Drop the original "Embarked" and "Pclass" columns
titanic_data <- subset(titanic_data, select = -c(Embarked, Pclass))

# Join the "Q", "S" and "2", "3" columns from the one-hot encoded matrices to␣
↪titanic_data
titanic_data <- cbind(titanic_data, embark[, c("Q", "S")], Pcl[, c(2, 3)])

# Display the modified data
head(titanic_data)
```

A data.frame: 6 × 8

	Survived <int>	Male <int>	SibSp <int>	Parch <int>	Q <dbl>	S <dbl>	2 <dbl>	3 <dbl>
1	0	1	1	0	0	1	0	1
2	1	0	1	0	0	0	0	0
3	1	0	0	0	0	1	0	1
4	1	0	1	0	0	1	0	0
5	0	1	0	0	0	1	0	1
6	0	1	0	0	1	0	0	1

[36]:
```
# Create predictor variables (X) by excluding the "Survived" column
X <- subset(titanic_data, select = -c(Survived))

# Create the target variable (y) with only the "Survived" column
y <- titanic_data$Survived
```

[37]:
```
# Set the seed for reproducibility
set.seed(1)

# Split the data into training and testing sets
split_index <- createDataPartition(y, p = 0.7, list = FALSE)
X_train <- X[split_index, ]
X_test <- X[-split_index, ]
y_train <- y[split_index]
y_test <- y[-split_index]
```

[38]:
```
# Fit a logistic regression model
model <- glm(y_train ~ ., family = "binomial", data = cbind(y_train, X_train))

# Display the summary of the model
summary(model)
```

```
Call:
glm(formula = y_train ~ ., family = "binomial", data = cbind(y_train,
    X_train))

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   2.7336     0.3323   8.227  < 2e-16 ***
Male         -2.7072     0.2326 -11.638  < 2e-16 ***
SibSp        -0.1444     0.1121  -1.288   0.1976
Parch        -0.1704     0.1274  -1.337   0.1811
Q            -0.2953     0.4543  -0.650   0.5157
S            -0.5674     0.2666  -2.128   0.0333 *
`2`          -0.6514     0.2993  -2.177   0.0295 *
`3`          -1.6998     0.2687  -6.327  2.5e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 834.27  on 623  degrees of freedom
Residual deviance: 581.76  on 616  degrees of freedom
AIC: 597.76

Number of Fisher Scoring iterations: 4
```

[39]:
```r
# Make predictions on the test set
predictions <- predict(model, newdata = cbind(1, X_test), type = "response")
```

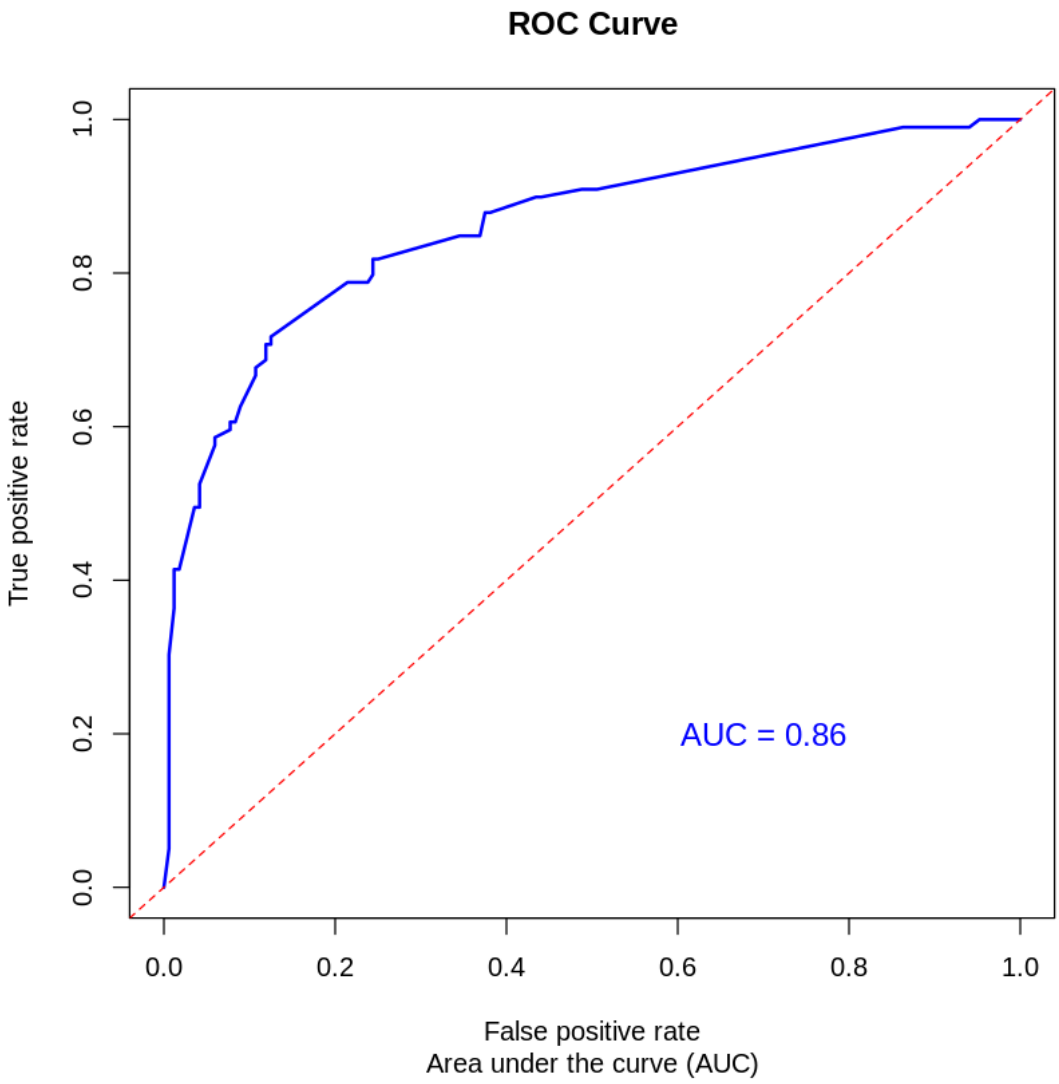
[40]:
```r
# Convert predicted probabilities to class predictions (0 or 1)
predicted_classes <- ifelse(predictions > 0.5, 1, 0)

# Convert y_test to a factor with the same levels as predicted_classes
y_test <- factor(y_test, levels = levels(factor(predicted_classes)))

# Convert both vectors to factors with the same levels
predicted_classes <- factor(predicted_classes, levels = levels(factor(y_test)))
```

[41]:
```r
# Create a confusion matrix
conf_matrix <- confusionMatrix(predicted_classes, y_test)

# Display classification report
print(conf_matrix)
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 150  33
         1  18  66

               Accuracy : 0.809
                 95% CI : (0.7566, 0.8543)
    No Information Rate : 0.6292
    P-Value [Acc > NIR] : 1.334e-10

                  Kappa : 0.5775

 Mcnemar's Test P-Value : 0.04995

            Sensitivity : 0.8929
            Specificity : 0.6667
         Pos Pred Value : 0.8197
         Neg Pred Value : 0.7857
```

```
            Prevalence : 0.6292
        Detection Rate : 0.5618
  Detection Prevalence : 0.6854
     Balanced Accuracy : 0.7798

      'Positive' Class : 0
```

[42]:
```r
# Create a ROC curve
roc_curve <- prediction(predictions, y_test)
roc_perf <- performance(roc_curve, "tpr", "fpr")

# Plot ROC curve
plot(roc_perf, col = "blue", lwd = 2, main = "ROC Curve", sub = "Area under the␣
 ↪curve (AUC)")
abline(a = 0, b = 1, lty = 2, col = "red")

# Display AUC in the plot
auc_value <- round(as.numeric(performance(roc_curve, "auc")@y.values), 2)
text(0.7, 0.2, paste("AUC =", auc_value), col = "blue", cex = 1.2)

# Display the plot
```



4 Feature Selection

Feature selection involves the identification and retention of the most relevant features in a dataset. This process enhances model efficiency, reduces overfitting, and improves interpretability. Techniques include filter methods that evaluate individual feature relevance, wrapper methods using model performance, and embedded methods integrating feature selection into the model training. Considerations such as domain knowledge and model-specific behaviors guide the selection process. Efficient feature selection can significantly enhance model accuracy and streamline computational complexity.

[43]:
```
# Load the CSV file into a data frame
breast_cancer <- read.csv("breast cancer.csv")
```

```
# Display the first few rows
head(breast_cancer)
```

A data.frame: 6 × 33

	id <int>	diagnosis <chr>	radius_mean <dbl>	texture_mean <dbl>	perimeter_mean <dbl>	area_mean <dbl>
1	842302	M	17.99	10.38	122.80	1001.0
2	842517	M	20.57	17.77	132.90	1326.0
3	84300903	M	19.69	21.25	130.00	1203.0
4	84348301	M	11.42	20.38	77.58	386.1
5	84358402	M	20.29	14.34	135.10	1297.0
6	843786	M	12.45	15.70	82.57	477.1

[44]:
```
# Convert "diagnosis" to a factor
breast_cancer$diagnosis <- as.factor(breast_cancer$diagnosis)

# One-hot encode "diagnosis" column
diagnosis <- as.integer(breast_cancer$diagnosis == "M")
```

[45]:
```
# Drop "Name", "Cabin", "Ticket", "PassengerId", "Fare" and "Age" columns
breast_cancer <- subset(breast_cancer, select = -c(id, diagnosis, X))

# Display the modified data
head(breast_cancer)
```

A data.frame: 6 × 30

	radius_mean <dbl>	texture_mean <dbl>	perimeter_mean <dbl>	area_mean <dbl>	smoothness_mean <dbl>	c
1	17.99	10.38	122.80	1001.0	0.11840	0
2	20.57	17.77	132.90	1326.0	0.08474	0
3	19.69	21.25	130.00	1203.0	0.10960	0
4	11.42	20.38	77.58	386.1	0.14250	0
5	20.29	14.34	135.10	1297.0	0.10030	0
6	12.45	15.70	82.57	477.1	0.12780	0

[46]:
```
# Print information about the dataset
str(breast_cancer)
```

```
'data.frame':   569 obs. of  30 variables:
 $ radius_mean            : num  18 20.6 19.7 11.4 20.3 …
 $ texture_mean           : num  10.4 17.8 21.2 20.4 14.3 …
 $ perimeter_mean         : num  122.8 132.9 130 77.6 135.1 …
 $ area_mean              : num  1001 1326 1203 386 1297 …
 $ smoothness_mean        : num  0.1184 0.0847 0.1096 0.1425 0.1003 …
 $ compactness_mean       : num  0.2776 0.0786 0.1599 0.2839 0.1328 …
 $ concavity_mean         : num  0.3001 0.0869 0.1974 0.2414 0.198 …
 $ concave.points_mean    : num  0.1471 0.0702 0.1279 0.1052 0.1043 …
 $ symmetry_mean          : num  0.242 0.181 0.207 0.26 0.181 …
 $ fractal_dimension_mean : num  0.0787 0.0567 0.06 0.0974 0.0588 …
 $ radius_se              : num  1.095 0.543 0.746 0.496 0.757 …
```

```
$ texture_se              : num  0.905 0.734 0.787 1.156 0.781 …
$ perimeter_se            : num  8.59 3.4 4.58 3.44 5.44 …
$ area_se                 : num  153.4 74.1 94 27.2 94.4 …
$ smoothness_se           : num  0.0064 0.00522 0.00615 0.00911 0.01149 …
$ compactness_se          : num  0.049 0.0131 0.0401 0.0746 0.0246 …
$ concavity_se            : num  0.0537 0.0186 0.0383 0.0566 0.0569 …
$ concave.points_se       : num  0.0159 0.0134 0.0206 0.0187 0.0188 …
$ symmetry_se             : num  0.03 0.0139 0.0225 0.0596 0.0176 …
$ fractal_dimension_se    : num  0.00619 0.00353 0.00457 0.00921 0.00511 …
$ radius_worst            : num  25.4 25 23.6 14.9 22.5 …
$ texture_worst           : num  17.3 23.4 25.5 26.5 16.7 …
$ perimeter_worst         : num  184.6 158.8 152.5 98.9 152.2 …
$ area_worst              : num  2019 1956 1709 568 1575 …
$ smoothness_worst        : num  0.162 0.124 0.144 0.21 0.137 …
$ compactness_worst       : num  0.666 0.187 0.424 0.866 0.205 …
$ concavity_worst         : num  0.712 0.242 0.45 0.687 0.4 …
$ concave.points_worst    : num  0.265 0.186 0.243 0.258 0.163 …
$ symmetry_worst          : num  0.46 0.275 0.361 0.664 0.236 …
$ fractal_dimension_worst: num  0.1189 0.089 0.0876 0.173 0.0768 …
```

[47]:
```
# Standardizing the data frame
scaled_df <- as.data.frame(scale(breast_cancer))

# Display the first few rows of the standardized data frame
head(scaled_df)
```

A data.frame: 6 × 30

	radius_mean <dbl>	texture_mean <dbl>	perimeter_mean <dbl>	area_mean <dbl>	smoothness_mean <dbl>	c
1	1.0960995	-2.0715123	1.2688173	0.9835095	1.5670875	3
2	1.8282120	-0.3533215	1.6844726	1.9070303	-0.8262354	-
3	1.5784992	0.4557859	1.5651260	1.5575132	0.9413821	1
4	-0.7682333	0.2535091	-0.5921661	-0.7637917	3.2806668	3
5	1.7487579	-1.1508038	1.7750113	1.8246238	0.2801253	0
6	-0.4759559	-0.8346009	-0.3868077	-0.5052059	2.2354545	1

[48]:
```
# Applying PCA on the standardized data frame
pca_result <- prcomp(scaled_df, center = TRUE, scale. = TRUE)

# Extract the first two principal components
pca_data <- as.data.frame(pca_result$x[, 1:2])

# Display the first few rows of the PCA data
head(pca_data)
```

A data.frame: 6 × 2

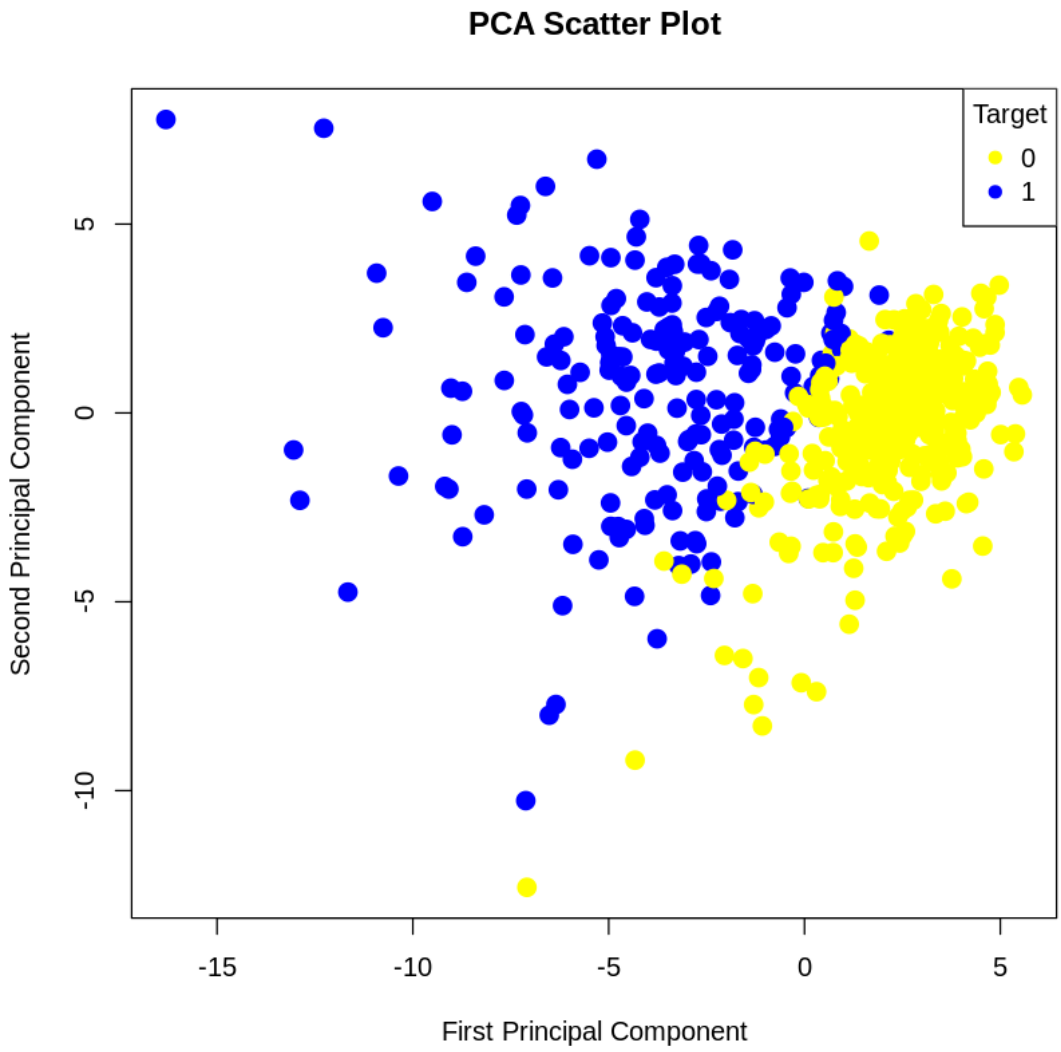
	PC1 <dbl>	PC2 <dbl>
1	-9.184755	-1.946870
2	-2.385703	3.764859
3	-5.728855	1.074229
4	-7.116691	-10.266556
5	-3.931842	1.946359
6	-2.378155	-3.946456

[49]:
```
# Check the shape of 'scaled_df'
dim(scaled_df)

# Check the shape of 'pca_data'
dim(pca_data)
```

1. 569 2. 30

1. 569 2. 2

[50]:
```
# Create a scatter plot of the PCA data
plot(pca_data[, 1], pca_data[, 2], col = ifelse(diagnosis == 0, "yellow",␣
 ↪"blue"),
     main = "PCA Scatter Plot", cex = 1.5, pch = 19,
     xlab = "First Principal Component", ylab = "Second Principal Component")

# Add a colorbar legend
legend("topright", legend = levels(factor(diagnosis)),
    col = c("yellow", "blue"), pch = 19, title = "Target")
```



[51]:
```r
# Create predictor variables (X)
X <- pca_data

# Create the target variable (y)
y <- diagnosis
```

[52]:
```r
# Set the seed for reproducibility
set.seed(1)

# Split the data into training and testing sets
split_index <- createDataPartition(y, p = 0.7, list = FALSE)
X_train <- X[split_index, ]
X_test <- X[-split_index, ]
```

```
y_train <- y[split_index]
y_test <- y[-split_index]
```

[53]:
```
# Fit a logistic regression model
model <- glm(y_train ~ ., family = "binomial", data = cbind(y_train, X_train))

# Display the summary of the model
summary(model)
```

```
Warning message:
"glm.fit: fitted probabilities numerically 0 or 1 occurred"


Call:
glm(formula = y_train ~ ., family = "binomial", data = cbind(y_train,
    X_train))

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -0.5046     0.2601  -1.940   0.0524 .
PC1          -1.8892     0.2569  -7.354 1.93e-13 ***
PC2           0.9833     0.1733   5.674 1.40e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 521.84  on 398  degrees of freedom
Residual deviance: 101.90  on 396  degrees of freedom
AIC: 107.9

Number of Fisher Scoring iterations: 8
```

[54]:
```
# Make predictions on the test set
predictions <- predict(model, newdata = cbind(1, X_test), type = "response")
```

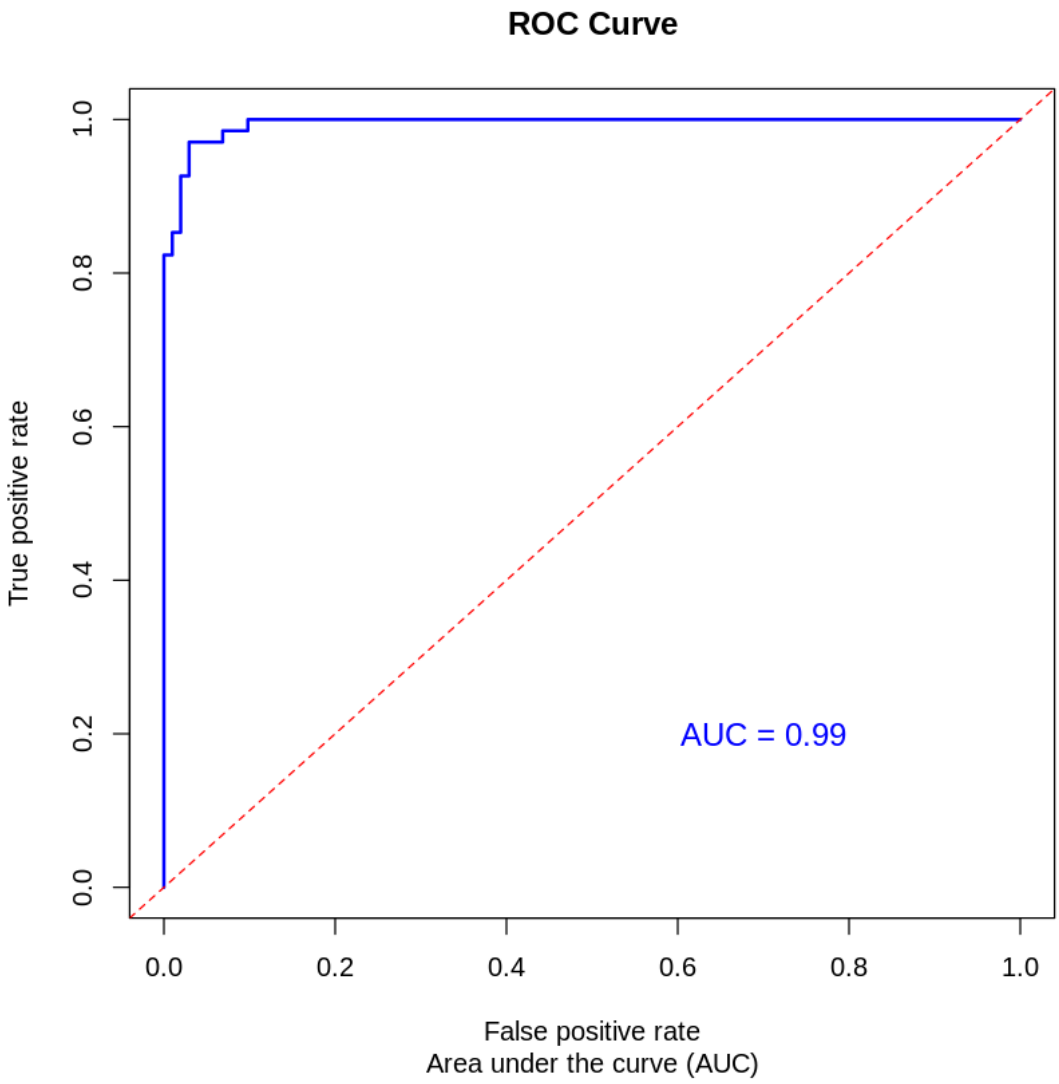
[55]:
```
# Convert predicted probabilities to class predictions (0 or 1)
predicted_classes <- ifelse(predictions > 0.5, 1, 0)

# Convert y_test to a factor with the same levels as predicted_classes
y_test <- factor(y_test, levels = levels(factor(predicted_classes)))

# Convert both vectors to factors with the same levels
predicted_classes <- factor(predicted_classes, levels = levels(factor(y_test)))
```

[56]:
```
# Create a confusion matrix
conf_matrix <- confusionMatrix(predicted_classes, y_test)
```

```
# Display classification report
print(conf_matrix)
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 100   5
         1   2  63

               Accuracy : 0.9588
                 95% CI : (0.917, 0.9833)
    No Information Rate : 0.6
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9136

 Mcnemar's Test P-Value : 0.4497

            Sensitivity : 0.9804
            Specificity : 0.9265
         Pos Pred Value : 0.9524
         Neg Pred Value : 0.9692
             Prevalence : 0.6000
         Detection Rate : 0.5882
   Detection Prevalence : 0.6176
      Balanced Accuracy : 0.9534

       'Positive' Class : 0
```

[57]:
```
# Create a ROC curve
roc_curve <- prediction(predictions, y_test)
roc_perf <- performance(roc_curve, "tpr", "fpr")

# Plot ROC curve
plot(roc_perf, col = "blue", lwd = 2, main = "ROC Curve", sub = "Area under the␣
 ↪curve (AUC)")
abline(a = 0, b = 1, lty = 2, col = "red")

# Display AUC in the plot
auc_value <- round(as.numeric(performance(roc_curve, "auc")@y.values), 2)
text(0.7, 0.2, paste("AUC =", auc_value), col = "blue", cex = 1.2)

# Display the plot
```


ROC Curve
True positive rate
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.99
False positive rate
Area under the curve (AUC)