Demonstrate following pre-processing operations using R

a)Deleting missing values
b)Replacing missing values
c)Imputing missing values
d)Work with categorial variables
e)Work with outliers

Use Dataset: titanic_train.csv

a) Deleting missing values:

To delete missing values in R, we can use the na.omit() function. Here is an example:

Code:

```
# Read the dataset
titanic <- read.csv("titanic_train.csv")

# Remove rows with missing values
titanic_clean <- na.omit(titanic)

# Print the number of rows before and after removing missing values
cat("Number of rows before removing missing values:", nrow(titanic), "\n")
cat("Number of rows after removing missing values:", nrow(titanic_clean),
"\n")
```

b) Replacing missing values:

To replace missing values in R, we can use the replace() function. Here is an example:

Code:

```
# Read the dataset
titanic <- read.csv("titanic_train.csv")

# Replace missing values with the mean value of the column
titanic$Age <- replace(titanic$Age, is.na(titanic$Age), mean(titanic$Age, na.rm
= TRUE))

# Print the first few rows of the Age column to check if missing values have
been replaced
head(titanic$Age)
```

c) Imputing missing values:

To impute missing values in R, we can use various packages such as mice, missForest, or Amelia. Here is an example using the mice package:

Code:

```
# Install and load the mice package
install.packages("mice")
library(mice)

# Read the dataset
titanic <- read.csv("titanic_train.csv")

# Impute missing values using mice
mice_mod <- mice(titanic)

# Extract the completed dataset
titanic_imputed <- complete(mice_mod)

# Print the number of missing values before and after imputation
cat("Number of missing values before imputation:", sum(is.na(titanic)), "\n")
cat("Number of missing values after imputation:", sum(is.na(titanic_imputed)),
"\n")
```

d) Working with categorical variables:

To work with categorical variables in R, we can use the factor() function to convert them to factors. Here is an example:

Code:

```
# Read the dataset
titanic <- read.csv("titanic_train.csv")

# Convert the Sex column to a factor
titanic$Sex <- factor(titanic$Sex)

# Print the levels of the Sex column
levels(titanic$Sex)
```

e) Working with outliers:

To work with outliers in R, we can use various methods such as boxplots, histograms, or the outliers() function from the car package. Here is an example using a boxplot:

Code:

```
# Read the dataset
titanic <- read.csv("titanic_train.csv")

# Create a boxplot of the Age column
boxplot(titanic$Age, main = "Age Boxplot", ylab = "Age")

# Identify and print the outliers using the boxplot.stats() function
outliers <- boxplot.stats(titanic$Age)$out
cat("Outliers:", outliers, "\n")
```

Demonstrate Simple Linear Regression model using R

a)Define Problem Statement
b)Define Null Hypothesis
c)Perform Pre-processing operations on dataset
d)Prepare Model
e)Use Model for prediction
f)Evaluate Model

Use Dataset: Use any suitable dataset

Sure, let's walk through an example of simple linear regression using R.

a) Define Problem Statement:
Suppose we want to analyze the relationship between the number of hours studied and the corresponding exam scores for a class of students.

b) Define Null Hypothesis:
Our null hypothesis is that there is no significant linear relationship between the number of hours studied and exam scores.

c) Perform Pre-processing operations on dataset:
We'll use a sample dataset to demonstrate this example. Let's assume we have a CSV file named "exam_data.csv" with two columns: "hours_studied" and "exam_score".

Before proceeding, we need to import the dataset into R and check for any missing values or outliers. We'll also plot the data to get a visual representation of the relationship between the variables.

Code:

```
# Import dataset
data <- read.csv("exam_data.csv")
```

```
# Check for missing values
sum(is.na(data))

# Check for outliers using boxplot
boxplot(data$hours_studied, data$exam_score)

# Plot data
plot(data$hours_studied, data$exam_score, xlab = "Hours Studied", ylab =
"Exam Score", main = "Relationship between Hours Studied and Exam Score")
```

d) Prepare Model:

Now that we have pre-processed our dataset, we can create a simple linear regression model. We'll use the lm() function to create the model, where "exam_score" is our dependent variable and "hours_studied" is our independent variable.

Code:

```
# Create linear regression model
model <- lm(exam_score ~ hours_studied, data = data)

# View model summary
summary(model)
```

e) Use Model for Prediction:

We can now use our model to make predictions. Let's say we want to predict the exam score for a student who has studied for 7 hours. We'll use the predict() function to do this.

Code:

```
# Predict exam score for 7 hours of studying
new_data <- data.frame(hours_studied = 7)
predicted_score <- predict(model, new_data)
predicted_score
```

f) Evaluate Model:

Lastly, we'll evaluate our model by calculating the R-squared value, which tells us how well our model fits the data. We can also plot our model to visualize the line of best fit.

Code:

```
# Calculate R-squared value
r_squared <- summary(model)$r.squared
r_squared
```

# Plot line of best fit
abline(model, col = "red")
This concludes our example of simple linear regression using R.

Demonstrate Multiple Linear Regression using R/Python [40]

a) Define Problem Statement
b) Define Null Hypothesis
c) Perform Pre-processing operations on dataset
d) Prepare Model
e) Use Model for prediction
f) Evaluate Model

a) Problem Statement:
We have a dataset containing information about different startups such as their R&D Spend, Administration Spend, Marketing Spend, State, and Profit. We want to analyze the dataset and build a multiple linear regression model to predict the profit of a startup based on its R&D Spend, Administration Spend, Marketing Spend, and State.

b) Null Hypothesis:
The null hypothesis for multiple linear regression is that there is no significant relationship between the independent variables (R&D Spend, Administration Spend, Marketing Spend, and State) and the dependent variable (Profit).

c) Pre-processing operations on dataset:
First, we need to import the dataset and check for any missing values or outliers. We can do this using the following code:

{r}
Copy code
# Import the dataset
dataset <- read.csv("Startups.csv")

# Check for missing values
sum(is.na(dataset))

# Check for outliers
boxplot(dataset)
In this case, we do not have any missing values in the dataset and the boxplot shows that there are no outliers.

Next, we need to convert the categorical variable "State" into a numerical variable using one-hot encoding. We can do this using the following code:

{r}
Copy code
```
# Convert categorical variable into numerical using one-hot encoding
dataset <- cbind(dataset, model.matrix(~ State - 1, data = dataset))
dataset <- dataset[, -2]
```
d) Prepare Model:
We can now build a multiple linear regression model using the lm() function in R. We will use R&D Spend, Administration Spend, Marketing Spend, and the one-hot encoded State variables as the independent variables, and Profit as the dependent variable. We can do this using the following code:

{r}
Copy code
```
# Build multiple linear regression model
model <- lm(Profit ~ R.D.Spend + Administration + Marketing.Spend +
California + Florida, data = dataset)

# Print summary of the model
summary(model)
```
e) Use Model for prediction:
We can use the predict() function in R to make predictions using the multiple linear regression model. We can do this using the following code:

{r}
Copy code
```
# Make predictions
new_data <- data.frame(R.D.Spend = 165349.20, Administration = 136897.80,
Marketing.Spend = 471784.10, California = 0, Florida = 1)
predict(model, new_data)
```
This will output the predicted profit for a startup with R&D Spend of 165349.20, Administration Spend of 136897.80, Marketing Spend of 471784.10, and located in Florida.

f) Evaluate Model:
To evaluate the model, we can use the R-squared value, which represents the proportion of variance in the dependent variable (Profit) that is explained by the independent variables (R&D Spend, Administration Spend, Marketing Spend, and State). A higher R-squared value indicates a better fit of the model. We can calculate the R-squared value using the summary() function on the model. In this case, the R-squared value is 0.951, which indicates that the model fits the data well.

{r}
Copy code
Calculate R-squared value
summary(model)$r.squared

Perform following tasks

a) Create any R markdown document implementing any machine learning algorithm of your choice.
b) Upload it in your RStudio account

Step 1: Open RStudio and create a new R Markdown document.

Click on "File" > "New File" > "R Markdown..."
Choose a document type (e.g., "HTML" or "PDF")
Give a title and author name
Choose an output format (e.g., "HTML" or "PDF")

Step 2: Load the dataset

You can load a dataset using the read.csv() function in R.
For example, if you have a CSV file named "data.csv" in your working directory, you can load it using the following code:
{r}
Copy code
data <- read.csv("data.csv")

Step 3: Preprocess the data

Perform any necessary preprocessing steps on the data.
For example, you can remove missing values, scale the data, or encode categorical variables.

Step 4: Implement the machine learning algorithm

Choose a machine learning algorithm of your choice (e.g., linear regression, decision trees, random forests, support vector machines, or neural networks).
Implement the algorithm using the appropriate package and function in R.
For example, you can use the lm() function to implement linear regression:
{r}
Copy code
model <- lm(y ~ x, data = data)

Step 5: Evaluate the model

Evaluate the performance of the model using appropriate metrics (e.g., mean squared error, accuracy, or confusion matrix).
For example, you can use the summary() function to get a summary of the linear regression model and calculate the mean squared error:
{r}
Copy code
summary(model)
mse <- mean((predict(model, data) - data$y)^2)

Step 6: Write the report

Write a report describing your analysis, including the preprocessing steps, model implementation, and evaluation metrics.
You can use R code chunks in the R Markdown document to include the code and the output in the report.

Step 7: Knit the document

Knit the R Markdown document to produce the final report in the chosen output format.
Click on "Knit" > "Knit to [output format]"
Once you have created and tested your R Markdown document locally, you can upload it to any online platform, such as GitHub or RPubs, to share it with others.

Demonstrate Logistic Regression using R

a) Define Problem Statement
b) Define Null Hypothesis
c) Is it classification or prediction problem. Explain.
d) Perform Pre-processing operations on dataset
e) Prepare Model
f) Use Model for prediction
g) Evaluate Model

a) Problem Statement:
We have a dataset containing information about customers of a bank, including their age, salary, credit score, and whether or not they have exited the bank (i.e. closed their account). Our goal is to build a logistic regression model to predict whether or not a customer will exit the bank based on the given features.

b) Null Hypothesis:

Null hypothesis is that there is no significant relationship between the independent variables (age, salary, credit score) and the dependent variable (exit status of the customer).

c) It is a classification problem because the goal is to predict a binary outcome (whether the customer will exit the bank or not) based on given features.

d) Pre-processing operations on dataset:
We will load the dataset into R and check for missing values, outliers, and unnecessary variables. If any missing values are found, we will remove them or impute them as appropriate. We will also check for outliers and remove them if necessary. We will drop the unnecessary variables if any.

e) Prepare Model:
We will split the dataset into training and testing sets. We will then fit a logistic regression model on the training data using the glm() function in R, specifying a binomial family to account for the binary response variable. We will select the best set of variables using different feature selection techniques.

f) Use Model for prediction:
We will use the model to make predictions on the test data and evaluate the performance of the model using different performance metrics like accuracy, confusion matrix, ROC curve.

g) Evaluate Model:
We will evaluate the model by comparing the predicted values with the actual values using various evaluation metrics. We will also visualize the results using an ROC curve to see how well the model is able to distinguish between the positive and negative classes.

Here's the R code to demonstrate Logistic Regression:

{r}
Copy code
```
# Load the required libraries
library(tidyverse)
library(caret)

# Load the dataset
bank_data <- read.csv("bank.csv")

# Check for missing values
```

```
sum(is.na(bank_data))

# Remove the unnecessary variables
bank_data <- bank_data %>%
   select(-c(RowNumber, CustomerId, Surname))

# Split the dataset into training and testing sets
set.seed(1234)
training_indices <- createDataPartition(bank_data$Exited, p = 0.8, list = FALSE)
training_data <- bank_data[training_indices, ]
testing_data <- bank_data[-training_indices, ]

# Fit a logistic regression model
logit_model <- glm(Exited ~ ., data = training_data, family = binomial)

# Use the model to make predictions on the test data
predicted_values <- predict(logit_model, testing_data, type = "response")

# Evaluate the performance of the model
confusionMatrix(table(predicted_values > 0.5, testing_data$Exited))

# Plot the ROC curve
library(pROC)
roc(testing_data$Exited, predicted_values)
```

Perform following Hypothesis testing methods using R

a) One sample t-test
b) Two sampled t-test
c) Paired sampled t-test
d) ANOVA (F-TEST)

Sure, here are examples of each hypothesis testing method using R:

a) One sample t-test:
Suppose we have a sample of 20 students, and we want to know if their average score on a test is significantly different from the population mean of 75. We can perform a one-sample t-test to test the following null hypothesis:

H0: The population mean is 75.
Ha: The population mean is not 75.

We can use the following R code:

{r}

Copy code
```
# Generate random data
set.seed(123)
sample <- rnorm(20, mean = 72, sd = 5)

# One sample t-test
t.test(sample, mu = 75)
```
b) Two sampled t-test:
Suppose we have two samples of test scores from two different classes, and we want to know if the means of the two classes are significantly different. We can perform a two-sample t-test to test the following null hypothesis:

H0: The means of the two classes are equal.
Ha: The means of the two classes are not equal.

We can use the following R code:

{r}
Copy code
```
# Generate random data
set.seed(123)
class1 <- rnorm(20, mean = 70, sd = 5)
class2 <- rnorm(20, mean = 75, sd = 5)

# Two sample t-test
t.test(class1, class2)
```
c) Paired sampled t-test:
Suppose we have a sample of students who took a test before and after a training program, and we want to know if there was a significant improvement in their scores. We can perform a paired-sample t-test to test the following null hypothesis:

H0: The mean difference between the two scores is zero.
Ha: The mean difference between the two scores is not zero.

We can use the following R code:

{r}
Copy code
```
# Generate random data
set.seed(123)
before <- rnorm(20, mean = 70, sd = 5)
after <- before + rnorm(20, mean = 5, sd = 2)
```

# Paired sample t-test
t.test(before, after, paired = TRUE)
d) ANOVA (F-TEST):
Suppose we have three or more samples of test scores from different classes, and we want to know if there is a significant difference in the means of the classes. We can perform an ANOVA (F-test) to test the following null hypothesis:

H0: The means of all classes are equal.
Ha: At least one of the means is different from the others.

We can use the following R code:

{r}
Copy code
# Generate random data
set.seed(123)
class1 <- rnorm(20, mean = 70, sd = 5)
class2 <- rnorm(20, mean = 75, sd = 5)
class3 <- rnorm(20, mean = 80, sd = 5)

# ANOVA (F-test)
anova(lm(c(class1, class2, class3) ~ rep(1:3, each = 20)))

Implement Decision Tree Algorithm (Classifier) using R

a) Define Problem
b) Implement Decision Tree Algorithm on suitable dataset.
c) How to evaluate the above algorithm?

a) Define Problem:

We want to build a decision tree algorithm to predict whether a customer will churn or not based on their demographic and behavioral characteristics.

b) Implement Decision Tree Algorithm on suitable dataset:

We will use the "Telco Customer Churn" dataset from Kaggle. The dataset contains information about customers who left within the last month, including demographics, services used, and payment methods. The dataset can be downloaded from the following link:
https://www.kaggle.com/blastchar/telco-customer-churn.

We will first load the necessary libraries and the dataset.

{r}
Copy code
```
library(rpart)
library(rpart.plot)

churn_data <- read.csv("telco-customer-churn.csv", stringsAsFactors = FALSE)
```
Next, we will perform some pre-processing on the data by removing unnecessary columns and converting categorical variables to factors.

{r}
Copy code
```
churn_data$SeniorCitizen <- factor(churn_data$SeniorCitizen, levels = c(0, 1), labels = c("No", "Yes"))
churn_data$Churn <- factor(churn_data$Churn, levels = c("No", "Yes"))
churn_data$TotalCharges <- as.numeric(churn_data$TotalCharges)

churn_data <- churn_data[, -c(1, 16, 18)]
```
We will then split the data into training and testing datasets.

{r}
Copy code
```
set.seed(123)
train_index <- sample(1:nrow(churn_data), 0.7 * nrow(churn_data))
train_data <- churn_data[train_index, ]
test_data <- churn_data[-train_index, ]
```
We will then build a decision tree model using the rpart() function.

{r}
Copy code
```
tree_model <- rpart(Churn ~ ., data = train_data, method = "class")
```
We can visualize the decision tree using the rpart.plot() function.

{r}
Copy code
```
rpart.plot(tree_model)
```
c) How to evaluate the above algorithm?

We can evaluate the decision tree model using the confusion matrix, accuracy, precision, recall, and F1 score.

{r}
Copy code

```
predicted_churn <- predict(tree_model, test_data, type = "class")
confusion_matrix <- table(test_data$Churn, predicted_churn)
confusion_matrix

accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
precision <- confusion_matrix[2, 2] / sum(confusion_matrix[, 2])
recall <- confusion_matrix[2, 2] / sum(confusion_matrix[2, ])
f1_score <- 2 * precision * recall / (precision + recall)

accuracy
precision
recall
f1_score
```

The confusion matrix shows the number of true positives, true negatives, false positives, and false negatives. The accuracy, precision, recall, and F1 score provide an overall measure of the performance of the model.

Implement PCA using R

a) What is Dimension reduction?
b) What are different methods for dimension reduction?
c) Why Dimension reduction is important?
d) Implement PCA Algorithm on suitable dataset.
e) How to evaluate the above algorithm?

a) Dimension reduction is a process of reducing the number of variables or dimensions in a dataset while retaining as much information as possible. It involves transforming high-dimensional data into a lower-dimensional space.

b) Different methods for dimension reduction include:

Principal Component Analysis (PCA)
Linear Discriminant Analysis (LDA)
t-distributed Stochastic Neighbor Embedding (t-SNE)
Non-negative Matrix Factorization (NMF)
Independent Component Analysis (ICA)
Random Projections

c) Dimension reduction is important for several reasons:

It helps in visualizing high-dimensional data by reducing it to 2D or 3D, making it easier to understand and analyze.
It improves the performance of machine learning models by reducing the noise and redundancy in the data.

It helps in identifying the most important features or variables in the data, which can be used for further analysis or modeling.
It reduces the computational complexity and storage requirements of the data.
d) Here is an implementation of PCA algorithm on the iris dataset:

r
Copy code
```
# Load the iris dataset
data(iris)

# Split the data into features and labels
x <- iris[, 1:4]
y <- iris[, 5]

# Scale the features
x <- scale(x)

# Perform PCA
pca <- prcomp(x)

# Print the summary of PCA
summary(pca)

# Plot the cumulative proportion of variance explained by each principal component
plot(cumsum(pca$sdev^2/sum(pca$sdev^2)), xlab = "Number of components", ylab = "Cumulative proportion of variance")

# Plot the principal components
biplot(pca)
```
e) The above algorithm can be evaluated by examining the variance explained by each principal component and selecting the appropriate number of components to retain. The quality of the projection can also be assessed by examining how well the samples and variables align with each principal component. Additionally, the projected data can be used for further analysis or modeling, such as clustering or classification.

Perform following task using MongoDB

a) Create suitable database in MongoDB.
b) Create suitable collection in database.
c) Insert 3 documents in above collection.
d) Perform CRUD operation on documents inserted in collection.
e) What is use(s) of MongoDB database?

a)Create suitable database in MongoDB:
To create a database in MongoDB, we can use the use command. For example, to create a database named mydatabase, we can use the following command:

perl
Copy code
use mydatabase
b)Create suitable collection in database:
To create a collection in MongoDB, we can use the db.createCollection() method. For example, to create a collection named mycollection in the mydatabase database, we can use the following command:

perl
Copy code
use mydatabase
db.createCollection("mycollection")
c)Insert 3 documents in above collection:
To insert documents in a collection, we can use the db.collection.insert() method. For example, to insert three documents in the mycollection collection, we can use the following commands:

yaml
Copy code
use mydatabase
db.mycollection.insert([
    { name: "John", age: 25, city: "New York" },
    { name: "Sarah", age: 30, city: "London" },
    { name: "David", age: 35, city: "Paris" }
])
d)Perform CRUD operation on documents inserted in collection:
To perform CRUD operations (Create, Read, Update, and Delete) on documents in a collection, we can use the following methods:

Create: db.collection.insertOne() or db.collection.insertMany()
Read: db.collection.find(), db.collection.findOne(), or db.collection.aggregate()
Update: db.collection.updateOne(), db.collection.updateMany(), or db.collection.replaceOne()

Delete: db.collection.deleteOne(), db.collection.deleteMany(), or db.collection.drop()
For example, to find all documents in the mycollection collection, we can use the following command:

lua
Copy code
use mydatabase
db.mycollection.find()
e)What is use(s) of MongoDB database?
MongoDB is a NoSQL database that is used for storing and retrieving large volumes of unstructured data in a flexible and scalable manner. Some of the main uses of MongoDB database are:

Real-time analytics: MongoDB can be used for real-time analytics of large volumes of data, such as clickstream data or social media feeds.
Content management: MongoDB can be used as a content management system for storing and retrieving unstructured data, such as blog posts or multimedia files.
Internet of Things (IoT): MongoDB can be used as a backend database for IoT applications, which generate large volumes of unstructured data from sensors and devices.
Mobile applications: MongoDB can be used as a backend database for mobile applications, which require fast and flexible data storage and retrieval.

Implement K-means clustering using R

a) What is clustering?
b) Write steps of K-means clustering algorithm.
c) How to determine best value of k?
d) Implement K-means clustering on suitable dataset.
e) How to evaluate the above algorithm?

a) Clustering is a technique used in machine learning to group similar data points or objects together based on their characteristics or features.

b) The steps of the K-means clustering algorithm are as follows:

Initialize k number of centroids randomly
Assign each data point to the nearest centroid
Compute the mean of each cluster to get a new centroid for each cluster
Repeat steps 2 and 3 until the centroids no longer move
c) One common method to determine the best value of k is the elbow method. In this method, we plot the sum of squared distances between each data point and its assigned centroid for different values of k. The elbow point,

which looks like an elbow on the graph, is the optimal value of k where adding more clusters does not significantly reduce the sum of squared distances.

d) Here is an example of implementing K-means clustering on a dataset using R:

```
r
Copy code
# Load dataset
data <- iris[,1:4]

# Determine optimal number of clusters using elbow method
wss <- numeric(10)
for (i in 1:10) {
  kmeans <- kmeans(data, centers = i, nstart = 10)
  wss[i] <- kmeans$tot.withinss
}
plot(1:10, wss, type = "b", xlab = "Number of Clusters", ylab = "Within Cluster Sum of Squares")

# Apply K-means clustering
k <- 3 # set number of clusters
kmeans <- kmeans(data, centers = k, nstart = 10)

# Plot clusters
library(cluster)
clusplot(data, kmeans$cluster, color = TRUE, shade = TRUE, labels = 2, lines = 0)
```

e) To evaluate the K-means clustering algorithm, we can use metrics such as silhouette score, Calinski-Harabasz index, or Davies-Bouldin index. These metrics measure the quality of clustering based on how well-separated the clusters are and how tightly-packed the data points are within each cluster.

Implement Time-series forecasting using R

a) What is time-series data? Give example.
b) Define the problem.
c) Implement Time-series forecasting on suitable dataset.
d) How to evaluate the above algorithm?

a) What is time-series data? Give example.

Time-series data is a type of data that is collected over time at regular intervals, with each observation being recorded in chronological order.

Examples of time-series data include stock prices, weather patterns, sales data, and website traffic.

b) Define the problem.

The problem in time-series forecasting is to predict the future values of a variable based on its past values. This can be useful in many different applications, such as predicting future sales, forecasting future stock prices, or estimating future weather patterns.

c) Implement Time-series forecasting on suitable dataset.

To demonstrate time-series forecasting using R, we can use the "AirPassengers" dataset which is built-in in R. This dataset contains the monthly number of airline passengers from January 1949 to December 1960.

First, we will load the dataset and convert it to a time-series object:

r
Copy code
```
# Load the AirPassengers dataset
data(AirPassengers)

# Convert the dataset to a time-series object
tsdata <- ts(AirPassengers, start = c(1949, 1), end = c(1960, 12), frequency = 12)
```
Next, we will split the dataset into a training set and a testing set:

r
Copy code
```
# Split the data into a training set and a testing set
train <- window(tsdata, end = c(1959, 12))
test <- window(tsdata, start = c(1960, 1))
```
Then, we will use the auto.arima() function from the "forecast" package to automatically fit an ARIMA model to the training data:

r
Copy code
```
# Fit an ARIMA model to the training data
library(forecast)
model <- auto.arima(train)
```
Finally, we will use the forecast() function to generate predictions for the testing set:

r
Copy code
# Generate forecasts for the testing set
forecast <- forecast(model, h = length(test))
d) How to evaluate the above algorithm?

To evaluate the accuracy of our time-series forecast, we can calculate the Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) between the predicted values and the actual values in the testing set.

r
Copy code
# Calculate the Mean Absolute Error (MAE)
MAE <- mean(abs(forecast$mean - test))

# Calculate the Root Mean Squared Error (RMSE)
RMSE <- sqrt(mean((forecast$mean - test)^2))
Lower values of MAE and RMSE indicate better accuracy of our time-series forecast