harinris_project2

Harin Rishabh

2024-04-29

```
# Load necessary libraries
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.3.2
```

```
library(readr)
library(forecast)
```

```
## Warning: package 'forecast' was built under R version 4.3.2
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library(zoo)
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

Plotting timeseries as is

```
# Load the data
oil_data <- read_csv("/Users/rishabh/Development/EAS509_SDM2/harinris_project2/oil.cs
v")
```

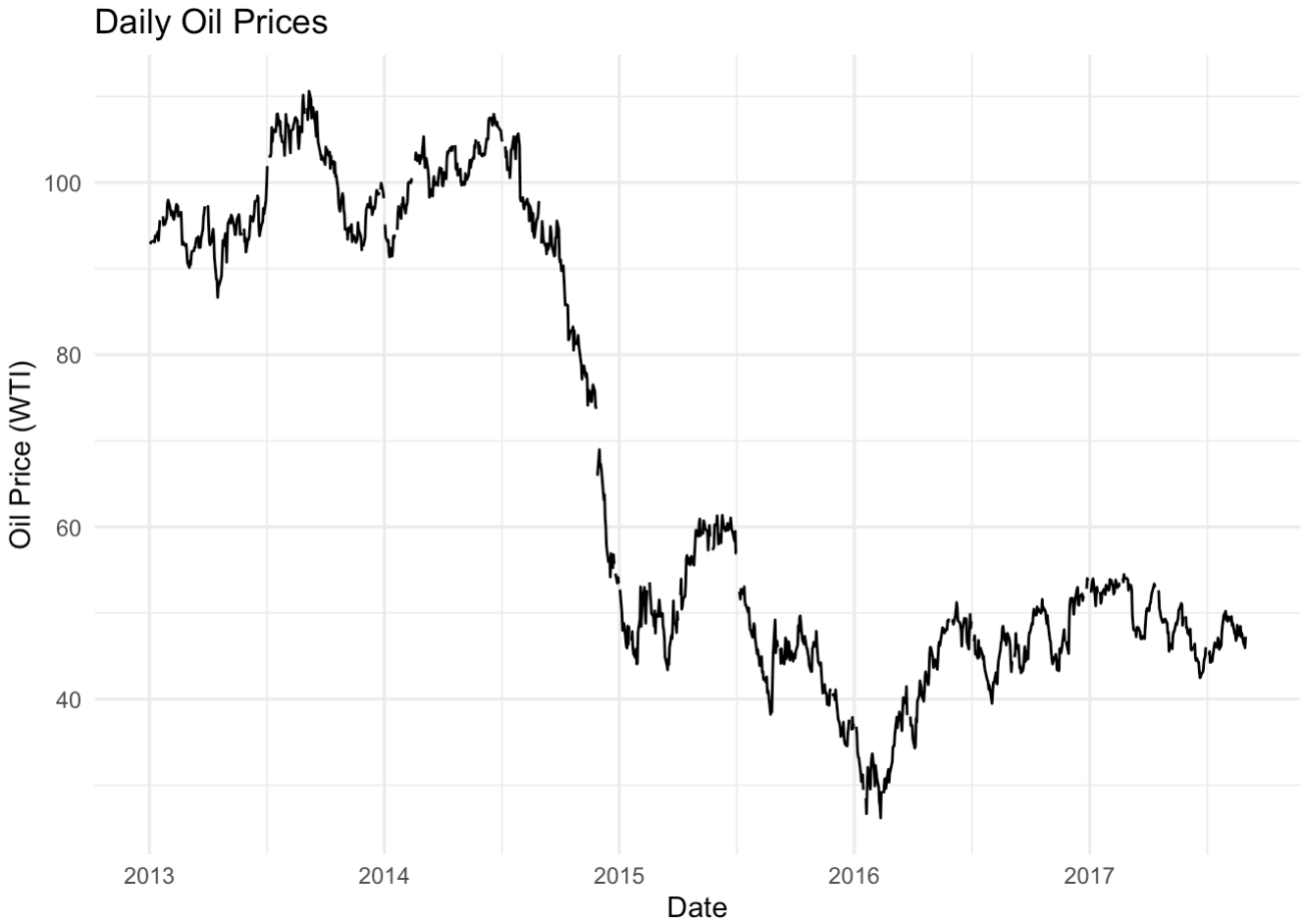
```
## Rows: 1218 Columns: 2
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## dbl  (1): dcoilwtico
## date (1): date
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
spec(oil_data)
```

```
## cols(
##   date = col_date(format = ""),
##   dcoilwtico = col_double()
## )
```

```
# Convert the date to Date type for plotting
oil_data$date <- as.Date(oil_data$date)
# Suppressing the column type information message
oil_data <- read_csv("/Users/rishabh/Development/EAS509_SDM2/harinris_project2/oil.csv", show_col_types = FALSE, col_types = cols(
  date = col_date(),
  dcoilwtico = col_double()
))

# Plot the time series
ggplot(oil_data, aes(x=date, y=dcoilwtico)) +
  geom_line() +
  labs(title="Daily Oil Prices", x="Date", y="Oil Price (WTI)") +
  theme_minimal()
```

```
## Warning: Removed 1 row containing missing values or values outside the scale range
## (`geom_line()`).
```



This warning indicates that there are rows with NA values for dcoilwtico, which are being removed when plotting the line graph. Missing values in time series data are common, especially on dates when no trading occurs or data wasn't recorded.

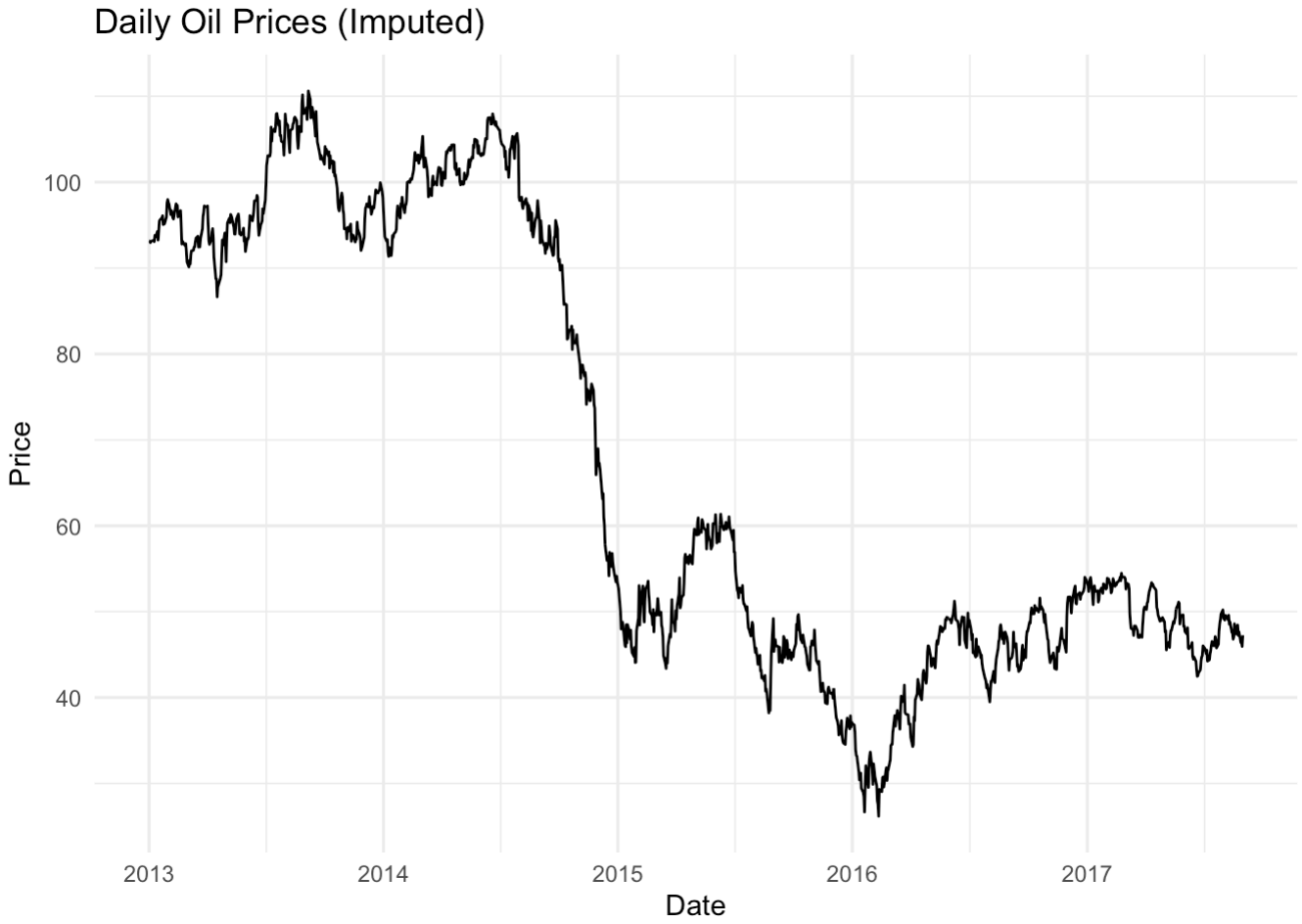
So we perform some steps to fill in the missing values

Filling in missing values using linear interpolation

```
# Ensure the zoo package is installed
if (!require(zoo)) install.packages("zoo", dependencies=TRUE)

# Load the zoo package
library(zoo)

# Check for missing values in the oil prices
missing_count <- sum(is.na(oil_data$dcoilwtico))
print(paste("Number of missing values in oil prices:", missing_count))
```

```
## [1] "Number of missing values in oil prices: 43"
```

```
# If there are missing values, impute them
if (missing_count > 0) {
  # Impute missing values using linear interpolation
  oil_data$dcoilwtico <- na.approx(oil_data$dcoilwtico, na.rm = FALSE)

  # In case there are NAs remaining at the beginning or end, use na.fill
  oil_data$dcoilwtico <- na.fill(oil_data$dcoilwtico, fill = "extend")
}

# Re-plot the time series with imputed data
ggplot(oil_data, aes(x = date, y = dcoilwtico)) +
  geom_line() +
  labs(title = "Daily Oil Prices (Imputed)", x = "Date", y = "Price") +
  theme_minimal()
```



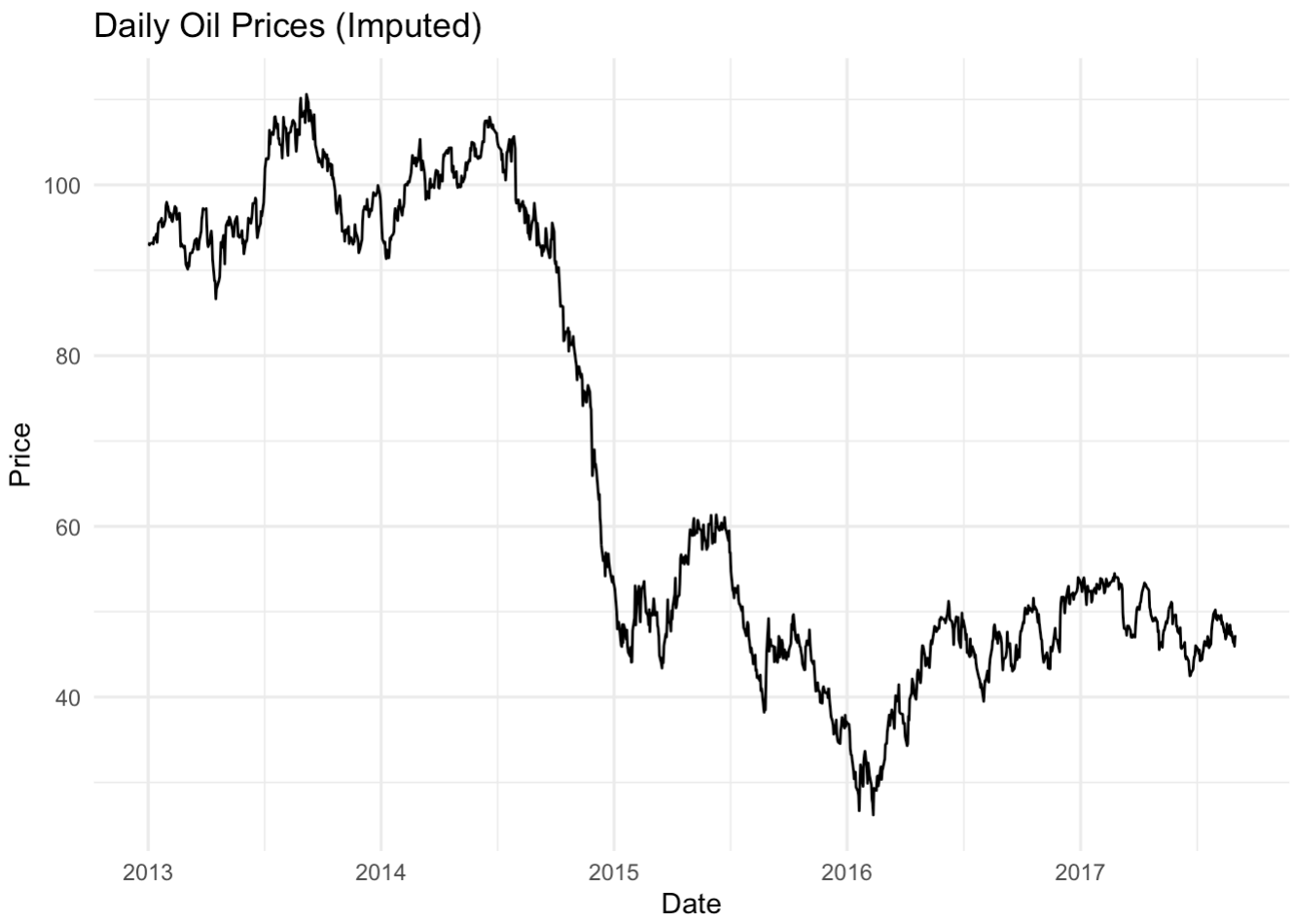
Checking for missing values after imputing

```
# Check for missing values in the oil prices
missing_count <- sum(is.na(oil_data$dcoilwtico))
print(paste("Number of missing values in oil prices:", missing_count))
```

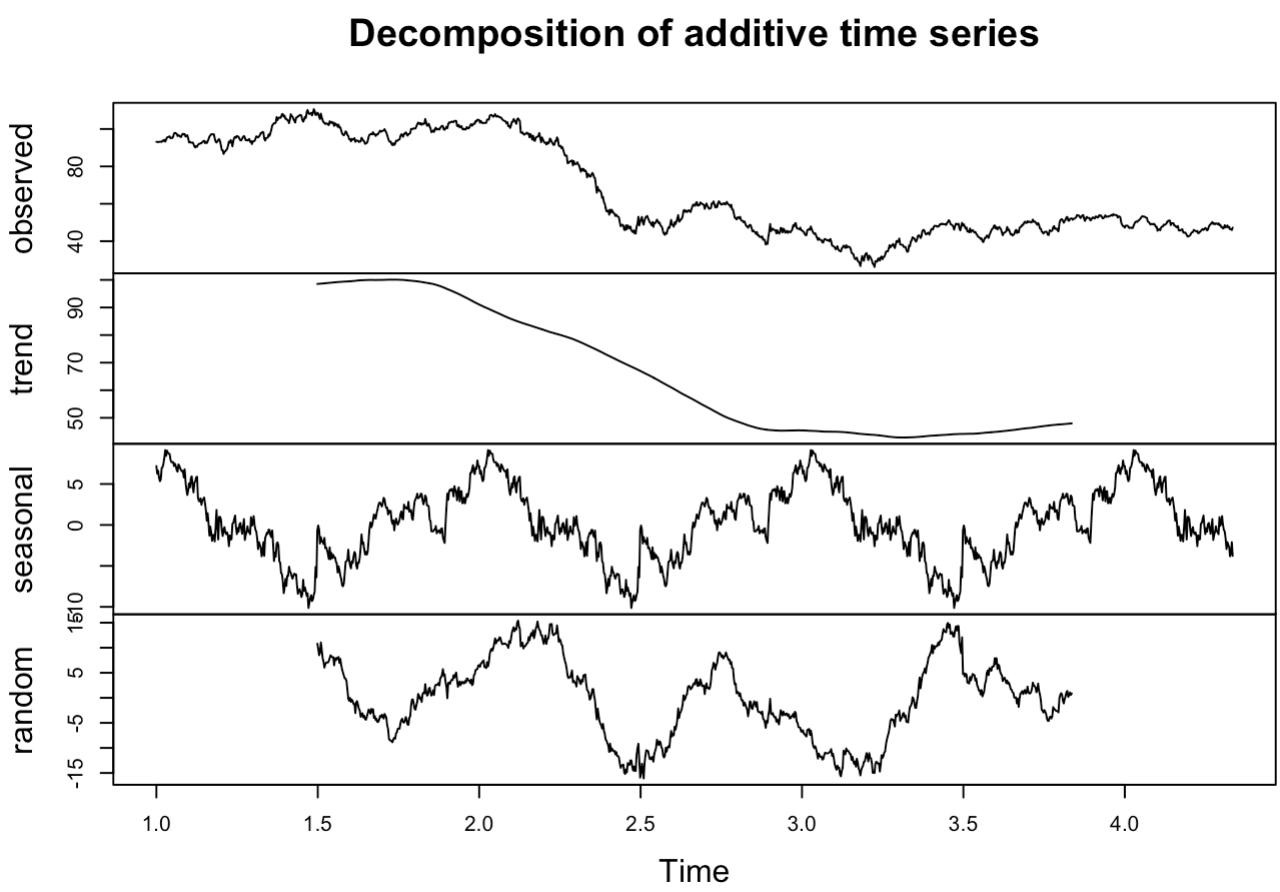
```
## [1] "Number of missing values in oil prices: 0"
```

Plotting trend and seasonality components to inspect visually

```
library(zoo)
library(ggplot2)
# Convert the date to Date type for plotting
oil_data$date <- as.Date(oil_data$date)

# Impute missing values using linear interpolation
oil_data$dcoilwtico <- na.approx(oil_data$dcoilwtico, na.rm = FALSE)
oil_data$dcoilwtico <- na.fill(oil_data$dcoilwtico, fill = "extend")
# Convert to a time series object
oil_ts <- ts(oil_data$dcoilwtico, frequency = 365)
# Plot the time series with imputed data
ggplot(oil_data, aes(x = date, y = dcoilwtico)) +
  geom_line() +
  labs(title = "Daily Oil Prices (Imputed)", x = "Date", y = "Price") +
  theme_minimal()
```




```
# To visually inspect for trend and seasonality
oil_data_ts <- ts(oil_data$dcoilwtico, frequency = 365)  # Assuming daily data with y
early seasonality
decomposed <- decompose(oil_data_ts)
plot(decomposed)
```



Trend: From about the middle of 2014 until the beginning of 2016, there was a noticeable declining trend. After that, the price levelled out and even began to slightly recover. This evolution is very clearly shown by the trend component, which is essential for comprehending the long-term movement in pricing.

Seasonality: There seems to be some volatility in the seasonal component, suggesting that there is a regular variation throughout each year. Still, in relation to the general trend and data noise, the amplitude of these seasonal oscillations is negligible. This shows that although seasonal influences may exist, they do not appear to be the main characteristic of this time series.

Random Component: The time series' "irregular" or "random" component, which represents the noise left over after trend and seasonality are taken into consideration, exhibits a great deal of unpredictability. This suggests that there are numerous variables that affect daily oil prices in addition to the trend and seasonal elements.

In summary, a noteworthy pattern has been noted, particularly a sudden drop that was followed by stability. While present, seasonal impacts are not predominant.The random component indicates that the data is highly volatile. This volatility may be caused by changes in market sentiment, economic data releases, or geopolitical events that are not seasonal or trend-based.

Q5)

ETS Models:

A type of statistical models called ETS models is employed in time series forecasting. A time series is broken down into three parts by them: error, trend, and seasonality. The following is what each part stands for:

Error (E): The data's random variation that cannot be linked to a pattern or seasonality is captured by this component. It shows the remaining variation when seasonality and trend are taken out. Trend (T): The long-term movement or direction of the time series is represented by this component. It can be multiplicative,

changing the pace of growth over time, or additive, changing the rate of change over time at a fixed rate. Seasonality (S): This component captures the periodic fluctuations or patterns in the data that occur at fixed intervals, such as daily, weekly, or yearly. Seasonality can also be additive or multiplicative. ETS models are specified using three letters, each indicating the type of model for the Error, Trend, and Seasonality components. The possible values are:

A: Additive M: Multiplicative N: None (absence of the component) For example:

ETS(A,A,A) indicates additive error, additive trend, and additive seasonality. ETS(M,A,N) indicates multiplicative error, additive trend, and no seasonality. To fit an ETS model, you typically estimate the smoothing parameters (α, β, γ) and initial values for the level (l), trend (b), and seasonal components (s). These parameters control the degree of smoothing applied to each component.

Holt-Winters Models:

Extensive smoothing is extended to manage seasonality in Holt-Winters models, commonly referred to as triple exponential smoothing models. Three smoothing equations are included in them: one for the seasonal component (γ), one for the trend (β), and one for the level (α). This is how they function: Level Equation: Using the observed value and the trend estimate from the previous time step, it updates the level estimate. Level (lt) = (1 - α) * + α * (Observation - Seasonal Component) (Trend + Level) Trend Equation: The difference between the current level and the prior level is used to update the trend estimate. Trend (bt) = (1 - β) * Trend + β * (Level - Previous Level) Seasonal Equation: It updates the seasonal component estimate based on the observed value and the level and trend estimates. Seasonal Component (st) = γ * (Observation - Level) + (1 - γ) * Seasonal Component Holt-Winters models can be either additive or multiplicative, depending on whether the seasonal component is added to or multiplied by the level and trend. They are particularly useful for time series data with trend and seasonality.

To fit a Holt-Winters model, you estimate the smoothing parameters (α, β, γ) and initial values for the level, trend, and seasonal components. These parameters control the degree of smoothing applied to each component.

##Q6)

Based on the accuracy metrics, the ETS model seems to perform the best on the training set, with the lowest RMSE and MAE. However, it's important to note that the test set performance of all models is considerably worse, indicating potential overfitting or inadequacy of the models to generalize the unseen data.

Regarding the presence of trend and seasonality in the data, the analysis suggests the existence of a noticeable declining trend from mid-2014 to early 2016, followed by a leveling out and slight recovery. Seasonal influences are present but not predominant, indicating some regular variation throughout each year. The random component suggests high volatility in the data, possibly influenced by various factors like market sentiment, economic data releases, or geopolitical events.

To suggest a suitable model for the data, considering the information provided, the ETS model may be preferred due to its better performance on the training set and its capability to capture trend and seasonality components effectively. However, further validation and refinement may be necessary to improve the model's performance on unseen data.

##7

```
if (!require(forecast)) install.packages("forecast")
library(forecast)

# Load and prepare the data
oil_data <- read.csv("/Users/rishabh/Development/EAS509_SDM2/harinris_project2/oil.cs
v")
oil_data$date <- as.Date(oil_data$date)

# Ensure no missing values
oil_data$dcoilwtico <- na.approx(oil_data$dcoilwtico, na.rm = FALSE)
oil_data$dcoilwtico <- na.fill(oil_data$dcoilwtico, fill = "extend")

# Convert to a time series object
oil_ts <- ts(oil_data$dcoilwtico, frequency = 365)

# Split the data
train_length <- length(oil_ts) - 30
train_set <- oil_ts[1:train_length]
test_set <- oil_ts[(train_length + 1):length(oil_ts)]

# Fit an ETS model
ets_model <- ets(train_set)
forecast_ets <- forecast(ets_model, h=30)
accuracy_ets <- accuracy(forecast_ets, test_set)

# Fit an ARIMA model
arima_model <- auto.arima(train_set)
forecast_arima <- forecast(arima_model, h=30)
accuracy_arima <- accuracy(forecast_arima, test_set)

# Fit a Holt-Winters model, trying additive first
hw_model <- HoltWinters(oil_ts)
forecast_hw <- forecast(hw_model, h=30)
accuracy_hw <- accuracy(forecast_hw, test_set)

# Print the model summaries and accuracy results
print("ETS Model Summary:")
```

```
## [1] "ETS Model Summary:"
```

```
print(summary(ets_model))
```

```
## ETS(A,N,N)
##
## Call:
##  ets(y = train_set)
##
##   Smoothing parameters:
##     alpha = 0.9705
##
##   Initial states:
##     l = 93.1462
##
##   sigma:  1.1848
##
##      AIC     AICc      BIC
## 8818.065 8818.085 8833.305
##
## Training set error measures:
##                       ME     RMSE       MAE        MPE     MAPE      MASE
## Training set -0.04024871 1.183846 0.9001033 -0.0846279 1.550671 0.9996001
##                       ACF1
## Training set -0.0007726529
```

```
print("ETS Model Accuracy:")
```

```
## [1] "ETS Model Accuracy:"
```

```
print(accuracy_ets)
```

```
##                       ME     RMSE       MAE        MPE     MAPE      MASE
## Training set -0.04024871 1.183846 0.9001033 -0.0846279 1.550671 0.9996001
## Test set      1.33204403 1.812766 1.5248071  2.7069434 3.125584 1.6933583
##                       ACF1
## Training set -0.0007726529
## Test set                NA
```

```
print("\nARIMA Model Summary:")
```

```
## [1] "\nARIMA Model Summary:"
```

```
print(summary(arima_model))
```

```
## Series: train_set
## ARIMA(0,1,0)
##
## sigma^2 = 1.404:  log likelihood = -1885.61
## AIC=3773.22   AICc=3773.22   BIC=3778.3
##
## Training set error measures:
##                       ME    RMSE       MAE        MPE     MAPE      MASE
## Training set -0.03898726 1.18435 0.8997838 -0.0821148 1.549438 0.9992453
##                     ACF1
## Training set -0.0299414
```

```
print("ARIMA Model Accuracy:")
```

```
## [1] "ARIMA Model Accuracy:"
```

```
print(accuracy_arima)
```

```
##                       ME    RMSE       MAE        MPE     MAPE      MASE
## Training set -0.03898726 1.18435 0.8997838 -0.0821148 1.549438 0.9992453
## Test set      1.34233333 1.82034 1.5316667  2.7283613 3.139572 1.7009761
##                     ACF1
## Training set -0.0299414
## Test set             NA
```

```
print("\nHolt-Winters Model Summary:")
```

```
## [1] "\nHolt-Winters Model Summary:"
```

```
print(summary(hw_model))
```

```
##              Length Class  Mode
## fitted       3412   mts    numeric
## x            1218   ts     numeric
## alpha           1   -none- numeric
## beta            1   -none- numeric
## gamma           1   -none- numeric
## coefficients  367   -none- numeric
## seasonal        1   -none- character
## SSE             1   -none- numeric
## call            2   -none- call
```

```
print("Holt-Winters Model Accuracy:")
```

```
## [1] "Holt-Winters Model Accuracy:"
```

```
print(accuracy_hw)
```

```
##                       ME      RMSE      MAE        MPE      MAPE      MASE
## Training set 0.03425942  1.805662 1.083057  0.1042011  2.370827  1.209359
## Test set     9.35742372 11.588266 9.687990 19.5921746 20.311321 10.817766
##                   ACF1
## Training set 0.1492315
## Test set            NA
```

##8)

```
# Load necessary libraries
library(forecast)

# Assuming you have the training and test sets named train_set and test_set respectiv
ely

# Fit ETS model
ets_model <- ets(train_set)
ets_train_rmse <- sqrt(mean(ets_model$residuals^2))
ets_test_rmse <- sqrt(mean((test_set - forecast(ets_model, h = length(test_set))$mea
n)^2))

# Fit ARIMA model
arima_model <- auto.arima(train_set)
arima_train_rmse <- sqrt(mean(arima_model$residuals^2))
arima_test_rmse <- sqrt(mean((test_set - forecast(arima_model, h = length(test_set))
$mean)^2))

# Calculate RMSE for Holt-Winters Model

hw_train_rmse <- sqrt(mean((train_set - hw_model$fitted)^2))
```

```
## Warning in `-.default`(train_set, hw_model$fitted): longer object length is not
## a multiple of shorter object length
```

```
hw_test_rmse <- sqrt(mean((test_set - forecast(hw_model, h = length(test_set))$mean)^
2))

# Print RMSE values
cat("\nETS Model RMSE (Training):", ets_train_rmse, "\n")
```

```
##
## ETS Model RMSE (Training): 1.183846
```

```
cat("ETS Model RMSE (Test):", ets_test_rmse, "\n\n")
```

```
## ETS Model RMSE (Test): 1.812766
```

```
cat("ARIMA Model RMSE (Training):", arima_train_rmse, "\n")
```

```
## ARIMA Model RMSE (Training): 1.18435
```

```
cat("ARIMA Model RMSE (Test):", arima_test_rmse, "\n\n")
```

```
## ARIMA Model RMSE (Test): 1.82034
```

```
cat("Holt-Winters Model RMSE (Training):", hw_train_rmse, "\n")
```

```
## Holt-Winters Model RMSE (Training): 55.08048
```

```
cat("Holt-Winters Model RMSE (Test):", hw_test_rmse, "\n")
```

```
## Holt-Winters Model RMSE (Test): 11.58827
```

Trying Seasonal Trend Decomposition using LOESS (STL) model

```
stl_model <- stl(oil_ts, s.window = "periodic")
trend <- stl_model$time.series[, "trend"]
seasonal <- stl_model$time.series[, "seasonal"]
residual <- stl_model$time.series[, "remainder"]
forecast_stl <- forecast(ts(trend, frequency=365))
accuracy_stl <- accuracy(forecast_stl, test_set)
print("Seasonal Trend Decomposition using LOESS (STL) Model Accuracy:")
```

```
## [1] "Seasonal Trend Decomposition using LOESS (STL) Model Accuracy:"
```

```
print(accuracy_stl)
```

```
##                        ME       RMSE         MAE          MPE       MAPE
## Training set -9.789238e-06 0.03116524 0.007928489  0.001282652 0.01358372
## Test set     -2.410167e+00 2.72711006 2.410167136 -5.084919222 5.08491922
##                    MASE         ACF1
## Training set  0.1455304 8.200693e-05
## Test set     44.2395107           NA
```

```
stl_train_rmse <- sqrt(mean((trend[1:length(train_set)] - train_set)^2))
stl_test_rmse <- sqrt(mean((test_set - forecast(stl_model, h = length(test_set))$mea
n)^2))
cat("Seasonal Trend Decomposition using LOESS Model RMSE (Training):", stl_train_rms
e, "\n")
```

```
## Seasonal Trend Decomposition using LOESS Model RMSE (Training): 7.568825
```

```
cat("Seasonal Trend Decomposition using LOESS Model RMSE (Test):", stl_test_rmse,
"\n")
```

```
## Seasonal Trend Decomposition using LOESS Model RMSE (Test): 2.36131
```

Results

The ETS and ARIMA models perform similarly on the training set based on the RMSE values, with the ETS model having a little lower RMSE. Nonetheless, with a smaller RMSE, the ETS model beats the ARIMA model on the test set.

In contrast to the ETS and ARIMA models, the Holt-Winters model has a significantly larger RMSE on both the training and test sets. This suggests that this dataset may not be as well-suited for the Holt-Winters model.

On trying Seasonal Trend Decomposition using LOESS Model, we actually get decent accuracy as it decomposes the seasonal, trend and residual components from the timeseries. This model gives the lower RMSE than the Holt-Winters model.

Because the STL model has the lowest RMSE on both the training and test sets, it seems to be the best option for predicting this specific time series data based on the RMSE metric.