Package Script

Julian Bara-Mason

default.R Script

```
convert_to_time_series <- function(data, s="monthly") {

  # Check if data is numeric and not empty
  if (!is.numeric(data) || length(data) == 0) {
    stop("Data should be numeric and non-empty.")
  }

  # Check for NA or Inf values in data
  if (any(is.na(data)) || any(is.infinite(data))) {
    stop("Data cannot contain NA or Inf values.")
  }

  # If the data is already a time series, simplify and return it
  if (is.ts(data)) {
    return(ts(as.numeric(data), start = start(data), frequency = frequency(data)))
  }

  # Flatten in case of matrix or dataframe with one column
  if (is.matrix(data) || is.data.frame(data)) {
    if (ncol(data) == 1) {
      data <- as.numeric(data[, 1])
    } else {
      stop("Multivariate data provided. This function expects a univariate series.")
    }
  }

  # Check if s is a character or numeric and if it belongs to the supported values
  allowed_s <- c("weekly", "monthly", "quarterly", "yearly", 52, 12, 4, 1)
  if (!s %in% allowed_s) {
    stop("Invalid value for 's'. Allowed values: 'weekly', 'monthly', 'quarterly', 'yearly' or 52, 12, 4, 1.")
  }

  # Convert s to a string if it's numeric
  if (is.numeric(s)) {
    s <- switch(as.character(s),
                "1" = "yearly",
                "4" = "quarterly",
                "12" = "monthly",
                "52" = "weekly",
                stop("Invalid numeric frequency specified."))
  }

  s <- tolower(s)

  frequency <- switch(s,
                      weekly = 52,
                      monthly = 12,
                      quarterly = 4,
                      yearly = 1,
                      stop("Invalid frequency specified."))

  if (length(data) < 2 * frequency) {
    stop("Insufficient amount of data. Need sufficient data for at least 2 complete chosen seasons")
  }

  data <- ts(data, frequency = frequency)

  # Simplify the data to remove any potential leftover attributes
  data <- ts(as.numeric(data), start = start(data), frequency = frequency)

  return(data)
}

identify_trend <- function(data, s="monthly") {

  if ((!is.character(s) && !is.numeric(s)) ||
      !(s %in% c("weekly", "monthly", "quarterly", "yearly", 52, 12, 4, 1))) {
    stop("Invalid value for 's'. Allowed values: 'weekly', 'monthly', 'quarterly', 'yearly' or 52, 12, 4, 1.")
  }
```

```
  data_ts <- convert_to_time_series(data, s)
  index <- as.numeric(time(data_ts))

  cubic_model <- lm(data_ts ~ poly(index, 3))

  p_values <- summary(cubic_model)$coefficients[,4]

  check_significance <- function(p_value) {
    if (p_value < 0.001) {
      return("a")
    } else if (p_value < 0.01) {
      return("b")
    } else if (p_value < 0.05) {
      return("c")
    } else if (p_value < 0.1) {
      return("d")
    } else {
      return("e")
    }
  }

  significance_levels <- sapply(p_values[-1], check_significance)  # Excluding the intercept

  if (all(significance_levels == "e")) {
    trend <- "unknown"
  } else if (all(significance_levels == significance_levels[1])) {
    trend <- "exponential"
  } else if (significance_levels[1] == significance_levels[2]) {
    trend <- "quadratic"
  } else {
    trend <- "linear"
  }

  return(list("trend" = trend, "data_ts" = data_ts))
}
```

Linear.R Script

```r
linear <- function(data, s = frequency(data), seasons_to_check = 1:s) {
  require(forecast)

  # Decompose the time series using STL
  ts_data <- ts(data, frequency = s)
  decomposed_data <- stl(ts_data, s.window="periodic")

  # Extract the trend, seasonal, and random components
  trend <- as.numeric(decomposed_data$time.series[, "trend"])
  seasonal <- as.numeric(decomposed_data$time.series[, "seasonal"])
  remainder <- as.numeric(decomposed_data$time.series[, "remainder"])

  # De-trended series (seasonal + noise)
  detrended <- seasonal + remainder

  # Initialize Ui, Vi, and di
  Ui <- numeric(length(seasons_to_check))
  Vi <- numeric(length(seasons_to_check))
  di <- numeric(length(seasons_to_check))

  # Calculate Ui and Vi for each specified seasonal period
  for (index in 1:length(seasons_to_check)) {
    i <- seasons_to_check[index]
    S_j <- detrended[seq(i, length(data), by = s)]
    j <- 1:length(S_j)

    # Only proceed if S_j has a sufficient size (at least two non-NA values)
    if (sum(!is.na(S_j)) > 1) {
      b <- 0

      if (s != 1) {  # Prevent division by zero when s == 1
        Ui[index] <- (b^2 * (s * (s + 1) / 12)) + (2 * b / (s - 1)) * sum(j * S_j, na.rm = TRUE) + (1 / (s - 1))
* sum(S_j^2, na.rm = TRUE)
      } else {
        Ui[index] <- NA
      }

      # Vi is the random component for that seasonal period
      Vi[index] <- remainder[i]

      # Calculate di
      di[index] <- Ui[index] - Vi[index]
    } else {
      Ui[index] <- NA
      Vi[index] <- NA
      di[index] <- NA
    }
  }

  return(list(Ui = Ui, Vi = Vi, di = di))
}
```

Quadratic.R Script

```r
quadratic <- function(data, s) {
  require(forecast)

  # Decompose the time series using STL
  decomposed_data <- stl(data, s.window="periodic")

  # Extract the trend, seasonality, and noise components
  trend <- as.numeric(decomposed_data$time.series[, "trend"])
  seasonal <- as.numeric(decomposed_data$time.series[, "seasonal"])
  remainder <- as.numeric(decomposed_data$time.series[, "remainder"])

  # De-trended series (seasonal + noise)
  detrended <- seasonal + remainder

  # Calculate c and b
  time_index <- 1:length(data)
  model <- lm(detrended ~ poly(time_index, 2))
  b <- summary(model)$coefficients[2]
  c <- summary(model)$coefficients[3]


  # Calculate Ui and Vi for each year
  n <- length(data) / s
  Ui <- numeric(n)
  Vi <- numeric(n)

  for (i in 1:n) {
    Sj <- detrended[(s*(i-1) + 1):(s*i)]
    C1 <- sum((1:s) * Sj)
    C2 <- sum((1:s)^2 * Sj)

    # Compute Ui
    Ui[i] <- (s*(s+1)/180)*((2*s-1)*(8*s-11)*c^2 - 30*(s-1)*b*c + 15*b^2) +
      (1/(s-1))*(sum(Sj^2) + 2*(b-2*c*s)*C1 + 2*c*C2) +
      ((s^2*(s+1))/3)*(b*c - c^2*(s-1) + (4*c*s*C1)/(s-1))*i +
      ((s^3*(s+1)*c^2)/3)*i^2

    # Compute Vi (without seasonality, so Sj = 0)
    Sj <- rep(0, s) # reset Sj for Vi
    C1 <- sum((1:s) * Sj)
    C2 <- sum((1:s)^2 * Sj)

    Vi[i] <- (s*(s+1)/180)*((2*s-1)*(8*s-11)*c^2 - 30*(s-1)*b*c + 15*b^2) +
      ((s^2*(s+1))/3)*(b*c - c^2*(s-1))*i +
      ((s^3*(s+1)*c^2)/3)*i^2
  }

  # Calculate Di
  Di <- Ui - Vi

  return(list(Ui = Ui, Vi = Vi, Di = Di))
}
```

Exponential.R Script

```r
exponential <- function(data, s = frequency(data)) {
  require(forecast)
  require(minpack.lm)

  # Decompose the time series using STL
  decomposed_data <- decompose(data)

  # Extract the seasonality and noise components to get de-trended data
  trend <- decomposed_data$trend
  seasonal <- decomposed_data$seasonal
  remainder <- decomposed_data$random

  # De-trended series (seasonal + noise)
  detrended <- seasonal + remainder

  # Use nlsLM for nonlinear regression
  time_index <- 1:length(data)
  start.list <- list(b = 0.5, c = 0.05)
  model <- nlsLM(detrended ~ b * exp(c * time_index), start = start.list)
  a <- coef(model)["a"]
  b <- coef(model)["b"]
  c <- coef(model)["c"]

  # Calculate Ui and Vi for each year
  n <- length(data) / s
  Ui <- numeric(n)
  Vi <- numeric(n)

  for (i in 1:n) {
    Sj <- detrended[(s*(i-1) + 1):(s*i)]

    # Compute Ui
    term1 <- b^2 * exp(2 * c * ((i - 1) * s + 1))
    term2 <- (1 - exp(2 * c * s)) / (1 - exp(2 * c))
    term3 <- (1 / s) * (1 - exp(c * s)) / (1 - exp(c))
    term4 <- sum(Sj^2)
    term5 <- 2 * b * exp(c * (i - 1) * s) * sum(exp(c * seq_len(s)) * Sj)

    Ui[i] <- term1 * (term2 - term3) + term4 + term5

    # Compute Vi (without seasonal effect)
    Sj <- rep(0, s) # reset Sj for Vi calculation
    termVi <- term1 * (term2 - term3)
    Vi[i] <- termVi
    #Vi[i] <- remainder[i]
  }

  # Calculate Di
  Di <- Ui - Vi

  return(list(Ui = Ui, Vi = Vi, Di = Di))
}
```

seasonalityTest.R Script

```r
#' Run Seasonality Test
#'
#' This function serves as the main entry point for running seasonality tests on a given time series data.
#' It internally calls `seasonality_test` or `interactive_seasonality_test` based on the parameters passed.
#'
#' @param data The input data for the seasonality test.
#' @param trend The trend type for the seasonality test (default = NULL). Supported options: linear, quadratic, e
xponential. If NULL, `interactive_seasonality_test` will be called.
#' @param s The seasonality parameter, applicable for the quadratic and exponential trends (default = 12).
#' @param confidence_level The desired confidence level for the statistical tests (default = 0.05).
#' @param summary_data Flag indicating whether to include the results summary (default = TRUE).
#' @return A list containing the result message and the summary statistics for each statistical test.
#' @examples
#' # Example 1: Linear Trend with default Confidence Level and Summary Data
#' data <- c(10, 15, 20, 15, 10)
#' result <- seasonality_test(data, trend = "linear", summary_data = TRUE)
#' print(result)
#'
#' # Example 2: Linear Trend without trend parameter
#' data <- c(10, 15, 20, 15, 10)
#' result <- seasonality_test(data)
```

```r
#' print(result)
#'
#' # Example 2: Quadratic Trend with Different Confidence Level and No Summary Data
#' data <- c(10, 15, 20, 15, 10)
#' result <- seasonality_test(data, trend = "quadratic", s = 2, confidence_level = 0.01)
#' print(result)
#'
#' # Example 3: Exponential Trend with Default Confidence Level and Summary Data
#' data <- c(10, 15, 20, 15, 10)
#' result <- seasonality_test(data, trend = "exponential", s = 0.5, summary_data = TRUE)
#' print(result)
#'
#' @export
run_seasonality_test <- function(data, trend=NULL, s=12, confidence_level = 0.05, seasons_to_check = 1:s, summary
_data = TRUE) {
  if (is.null(trend)) {
    # If trend is NULL, run interactive version of seasonality test
    interactive_seasonality_test(data, s, confidence_level, summary_data)
  } else {
    # If trend is specified, run regular version of seasonality test
    result <- seasonality_test(data, trend, s, confidence_level, summary_data)
    print_result <- result
    print_result$data_ts <- NULL
    print_result$trend_not_specified <- NULL
    print(print_result)
  }
}

seasonality_test <- function(data, trend=NULL, s=12, confidence_level = 0.05, seasons_to_check = 1:s, summary_dat
a = TRUE) {

  #install.packages("crayon")
  require(crayon)
  require(DescTools)

  # Ensure that the data is converted to a time series object
  data <- convert_to_time_series(data, s)

  # Check if confidence_level is between 0 and 1
  if (!is.numeric(confidence_level) || confidence_level <= 0 || confidence_level >= 1) {
    stop("Invalid confidence_level. It should be a number between 0 and 1.")
  }

  # If the trend was not specified, call the identify_trend function
  trend_not_specified <- is.null(trend)
  if (trend_not_specified) {
    source("/Users/jay/Documents/Documents - Jay's Macbook Pro (13281)/MSc Data Science & Analytics - UoL/Dissert
ation/disso 2/R/default.R")
    result <- identify_trend(data, s)

    if (is.null(result$trend)) {
      stop("No clear trend detected in the data.")
    }

    data <- result$data_ts
    trend <- result$trend
  }

  # Check if s belongs to the supported values if not NULL
  if (!is.null(s) &&
      (!is.character(s) && !is.numeric(s)) ||
      !(s %in% c("weekly", "monthly", "quarterly", "yearly", 52, 12, 4, 1))) {
    stop("Invalid value for 's'. Allowed values: 'weekly', 'monthly', 'quarterly', 'yearly' or 52, 12, 4, 1.")
  }


  # Check if trend belongs to the supported values if not NULL
  if (!is.null(trend) && !trend %in% c("linear", "quadratic", "exponential")) {
    stop("Invalid/Unknown trend. Supported options: linear, quadratic, and exponential")
  }

  has_warnings <- FALSE

  if (trend == "linear") {
    # Call the linear function from linear.R
    source("/Users/jay/Documents/Documents - Jay's Macbook Pro (13281)/MSc Data Science & Analytics - UoL/Dissert
ation/disso 2/R/linear.R")
    linear_result <- linear(data)
    Ui <- linear_result$Ui
```

```
    Vi <- linear_result$Vi
  } else if (trend == "quadratic") {
    # Call the quadratic function from quadratic.R
    source("/Users/jay/Documents/Documents - Jay's Macbook Pro (13281)/MSc Data Science & Analytics - UoL/Dissert
ation/disso 2/R/quadratic.R")
    quadratic_result <- quadratic(data, s)
    Ui <- quadratic_result$Ui
    Vi <- quadratic_result$Vi
  } else if (trend == "exponential") {
    # Call the exponential function from exponential.R
    source("/Users/jay/Documents/Documents - Jay's Macbook Pro (13281)/MSc Data Science & Analytics - UoL/Dissert
ation/disso 2/R/exponential.R")
    exponential_result <- exponential(data, s)
    Ui <- exponential_result$Ui
    Vi <- exponential_result$Vi
  }  else {
    stop("Invalid trending curve. Supported options: linear, quadratic, exponential")
  }

  is_seasonal <- FALSE

  # Perform the statistical tests if there are enough observations
  # T Test
  t_test_result <- t.test(Ui, Vi, conf.level = (1-confidence_level))
  p_value_t <- t_test_result$p.value

  # SIGN Test
  sign_test_result <- SignTest(Ui, Vi, conf.level = (1-confidence_level))
  p_value_sign <- sign_test_result$p.value

  #Wilcoxon SR test
  wilcox_test_result <- wilcox.test(Ui, Vi, conf.level = (1-confidence_level), paired = TRUE)
  p_value_wilcox <- wilcox_test_result$p.value

  #is_seasonal <- p_value_t < confidence_level || p_value_sign < confidence_level || p_value_wilcox < confidence_
level
  is_seasonal <- all(p_value_t < confidence_level, p_value_sign < confidence_level, p_value_wilcox < confidence_l
evel)

  # Calculate relevant statistical summary for each test
  t_summary <- summary(t_test_result)
  sign_summary <- summary(sign_test_result)
  wilcox_summary <- summary(wilcox_test_result)

  # Prepare the results summary
  summary_data <- data.frame(
    Test = c("Student t-Distribution", "Sign Test", "Wilcoxon Signed-Ranks Test"),
    p_value = c(p_value_t, p_value_sign, p_value_wilcox),
    stringsAsFactors = FALSE
  )

  # Prepare the result message
  if(is_seasonal){
    result_message <- paste("There appears to be statistically significant seasonality in the data at the", 100-(
confidence_level * 100), "% confidence level.")
  } else {
    result_message <- paste("There does not appear to be statistically significant seasonality in the data at the
", 100-(confidence_level * 100), "% confidence level.")
  }

  # Bundle everything into a list to return
  result <- list(
    message = result_message,
    summary_data = summary_data,
    data_ts = if (trend_not_specified) result$data_ts else data,
    trend = trend,
    trend_not_specified = trend_not_specified,
    has_warnings = has_warnings
  )
  return(result)
}

interactive_seasonality_test <- function(data, s=12, confidence_level = 0.05, seasons_to_check = 1:s, summary_dat
a = TRUE) {
  # First, run the seasonality test without specifying a trend
  result <- seasonality_test(data, trend=NULL, s=s, confidence_level=confidence_level, summary_data=summary_data)
  result_to_print <- result
  result_to_print$data_ts <- NULL
  result_to_print$trend <- NULL
```

```
  result_to_print$trend_not_specified <- NULL
  print(result_to_print)

  # Then, check if a trend was not specified in the result
  if (result$trend_not_specified) {
    # Print a warning and the identified trend
    cat(red("WARNING - UNRELIABLE RESULT: You did not specify a trend type. For greater accuracy, please specify
a trend.\n"))
    cat(blue(paste("Identified trend: ", result$trend, "\n")))

    # Decompose the data and plot the trend component
    decomposed <- decompose(result$data_ts)
    plot(decomposed$trend, main = "Trend Component of Time Series Data", ylab = "Trend")

    # Ask the user if they want to rerun the test with a specified trend
    valid_responses <- c("yes", "Yes", "YES", "No", "NO", "no")
    response <- tolower(readline(prompt = "Would you like to visually inspect the trend and identify the most sui
table trend? (yes/no): "))

    while (!response %in% valid_responses) {
      cat(red("Invalid response. Please enter 'yes' or 'no'.\n"))
      response <- tolower(readline(prompt = "Would you like to visually inspect the trend and identify the most s
uitable trend? (yes/no): "))
    }

    if (tolower(response) == "yes") {
      # Ask the user for the trend to test
      trend <- readline(prompt = "Inspect the plot of the trend and select the most suitable trend (linear, quadr
atic, exponential): ")

      # Rerun the test with the specified trend
      result <- seasonality_test(data, trend, window, s, confidence_level)

      # Print the result excluding data_ts
      print(result$message)
      print(result$summary_data)
    }
  } else {
    print(result$message)
    print(result$summary_data)
  }
}
```

data.R Script

```
require(quantmod)

ticker_sp <- "^GSPC"
ticker_nq <- "^IXIC"
ticker_dj <- "^DJI"

# Set the start and end dates for the data
start_date <- as.Date("1980-01-01")
end_date <- as.Date("2022-12-31")

# Retrieve the S&P 500 index data using quantmod
getSymbols(ticker_sp, from = start_date, to = end_date)
getSymbols(ticker_nq, from = start_date, to = end_date)
getSymbols(ticker_dj, from = start_date, to = end_date)

# Access the S&P 500 index data using the ticker symbol as an object
sp500 <- GSPC$GSPC.Close
nq <- IXIC$IXIC.Close
dj <- DJI$DJI.Close

# Calculate the start and end years
start_year <- as.numeric(format(start_date, "%Y"))
end_year <- as.numeric(format(end_date, "%Y"))

# Convert the stock prices into a time series object with daily frequency
sp500__ts <- ts(sp500, start = c(start_year, 1), end = c(end_year, 12), frequency = 12)
nq__ts <- ts(nq, start = c(start_year, 1), end = c(end_year, 12), frequency = 12)
dj__ts <- ts(dj, start = c(start_year, 1), end = c(end_year, 12), frequency = 12)

length(sp500__ts)

# Plot S&P 500 Time Series
```

```r
plot(sp500__ts, main="Time Series Plot of S&P 500", xlab="Date", ylab="Closing Price", col="black", lwd=2)

# Plot NASDAQ Time Series
plot(nq__ts, main="Time Series Plot of NASDAQ", xlab="Date", ylab="Closing Price", col="black", lwd=2)

# Plot DOW Jones Time Series
plot(dj__ts, main="Time Series Plot of DOW Jones", xlab="Date", ylab="Closing Price", col="black", lwd=2)



############### SYNTHETIC DATA
generate_timeseries <- function(trend, seasonality = TRUE, noise = TRUE) {
  n <- 600  # 10 years of monthly data

  noise_component <- ifelse(noise, rnorm(n, 0, 3), rep(0, n))

  if (trend == "linear") {
    if (seasonality) {
      # Linear with seasonality
      ts_data <- ts(seq(1, n) + 10*sin(seq(1, n)*2*pi/12) + noise_component, frequency = 12)
    } else {
      # Linear without seasonality
      ts_data <- ts(seq(1, n) + noise_component, frequency = 12)
    }
  } else if (trend == "quadratic") {
    noise_component <- ifelse(noise, rnorm(n, 0, 50), rep(0, n))
    if (seasonality) {
      # Quadratic with seasonality
      ts_data <- ts(seq(1, n)^2 + 500*sin(seq(1, n)*2*pi/12) + noise_component, frequency = 12)
    } else {
      # Quadratic without seasonality
      ts_data <- ts(seq(1, n)^2 + noise_component, frequency = 12)
    }
  } else if (trend == "exponential") {
    noise_component <- ifelse(noise, rnorm(n, 0, 50), rep(0, n))
    if (seasonality) {
      # Exponential with seasonality
      ts_data <- ts(2^seq(1, n) + 100*sin(seq(1, n)*2*pi/12) + noise_component, frequency = 12)
    } else {
      # Exponential without seasonality
      ts_data <- ts(2^seq(1, n) + noise_component, frequency = 12)
    }
  } else {
    stop("Invalid trend specified.")
  }

  return(ts_data)
}

# Linear Data Generation
linear_seasonal <- generate_timeseries("linear") # With Seasonality
linear_non_seasonal <- generate_timeseries("linear", seasonality = FALSE, noise = TRUE) # No Seasonality

plot(linear_seasonal, main="Time Series Plot of Linear Seasonal", xlab="Year", ylab="Value", col="black", lwd=2)
plot(linear_non_seasonal, main="Time Series Plot of Linear Non-Seasonal", xlab="Year", ylab="Value", col="black",
lwd=2)

# Quadratic Data Generation
quadratic_seasonal <- generate_timeseries("quadratic") # With Seasonality
quadratic_non_seasonal <- generate_timeseries("quadratic", seasonality = FALSE, noise = TRUE) # No Seasonality

plot(quadratic_seasonal, main="Time Series Plot of Quadratic Seasonal", xlab="Year", ylab="Value", col="blue", lw
d=2)
plot(quadratic_non_seasonal, main="Time Series Plot of Quadratic Non-Seasonal", xlab="Year", ylab="Value", col="b
lue", lwd=2)


# Exponential Data Generation
exponential_seasonal <- generate_timeseries("exponential") # With Seasonality
exponential_non_seasonal <- generate_timeseries("exponential", seasonality = FALSE, noise = TRUE) # No Seasonalit
y

plot(exponential_seasonal, main="Time Series Plot of Exponential Seasonal", xlab="Year", ylab="Value", col="red",
lwd=2)
plot(exponential_non_seasonal, main="Time Series Plot of Exponential Non-Seasonal", xlab="Year", ylab="Value", co
l="red", lwd=2)


######## 4-7
```

```r
# 4. Generic Dataset
generic_data <- rnorm(1200, mean = 50, sd = 10)
length(generic_data)


# 5. Matrix-format Dataset
mat_data <- matrix(rnorm(120 * 2, mean = 50, sd = 10), ncol = 2)
colnames(mat_data) <- c("Series1", "Series2")

mat_data_uni <- matrix(rnorm(120 * 2, mean = 50, sd = 10), ncol = 1)
colnames(mat_data_uni) <- c("Series1")
nrow(mat_data)
mat_data


# 6. Data with Anomalies
anomalous_data <- rnorm(120, mean = 50, sd = 10)
anomalous_data[sample(1:120, 5)] <- NA # Introduce some NA values
anomalous_data[sample(1:120, 5)] <- Inf # Introduce some Inf values

# 7. Attributed Time Series
attributed_ts <- ts(rnorm(120, mean = 50, sd = 10), frequency = 12)
attributes(attributed_ts)$src <- "Synthetic Generator"
attributes(attributed_ts)$updated <- Sys.Date()
attributes(attributed_ts)$index <- sample(1:1000, 120)

# 8. Non-Numeric Data
non_numeric <- factor(c("Red", "Blue", "Green", "Red", "Blue"))

# 9. Weather Data
### Weather Data

# Read the CSV file
data <- read.csv("/Users/jay/Downloads/monthly_csv.csv")

# Display the first few rows to inspect the data
head(data)

# Filter the data to use only one source, say "GCAG"
gcag_data <- subset(data, Source == "GCAG")

# Remove the "Source" column as it's now redundant
gcag_data$Source <- NULL

# Convert the "Date" column to Date type
gcag_data$Date <- as.Date(gcag_data$Date, format="%Y-%m-%d")

# Filter data for the years 2000 to 2022
gcag_data <- subset(gcag_data, as.numeric(format(gcag_data$Date, "%Y")) >= 2000 & as.numeric(format(gcag_data$Dat
e, "%Y")) <= 2022)

# Sort data by date
gcag_data <- gcag_data[order(gcag_data$Date), ]

# Extract only the Year and Month from Date for creating a time series
gcag_data$Year <- as.numeric(format(gcag_data$Date, "%Y"))
gcag_data$Month <- as.numeric(format(gcag_data$Date, "%m"))

# Convert data to a time series object
gcag_ts <- ts(gcag_data$Mean, start=c(min(gcag_data$Year), min(gcag_data$Month)), frequency=12)
```

test.R Script

```
run_seasonality_test(gcag_ts)
run_seasonality_test(sp500__ts, s=12)

run_seasonality_test(sp500__ts, trend="exponential", s=12)
run_seasonality_test(nq__ts, trend="exponential", s=12)
run_seasonality_test(dj__ts, trend="exponential", s=12)

run_seasonality_test(linear_seasonal, trend="linear", seasons_to_check=1:3, s=12)
run_seasonality_test(linear_non_seasonal, trend="linear", s=12)

run_seasonality_test(quadratic_seasonal, trend="quadratic", s=12)
run_seasonality_test(quadratic_non_seasonal, trend="quadratic", s=12)


run_seasonality_test(exponential_seasonal, trend="quadratic", s=12)
run_seasonality_test(exponential_non_seasonal, trend="quadratic", s=12)
```