**Data Preparation:**

# Load libraries

library(dplyr)

library(ggplot2)

library(forecast)

library(tseries)

library(keras)

library(tensorflow)

# Load the dataset

temp\_data <- read.csv("US\_City\_Temp\_Data.csv")

# Convert the `time` column to Date format

temp\_data$time <- as.Date(temp\_data$time, format = "%Y-%m-%d")

# Select data for Albuquerque

city\_data <- temp\_data %>% select(time, albuquerque) %>% na.omit()

# Convert to time series

city\_ts <- ts(city\_data$albuquerque, start = c(1948, 1), frequency = 12)

Explanation:

* **Loading Data:**
  + The dataset US\_City\_Temp\_Data.csv is loaded into the temp\_data data frame.
* **Formatting the time column:**
  + The time column is converted to the Date format using as.Date with the specified format %Y-%m-%d.
* **Subsetting for Albuquerque:**
  + Data specific to the Albuquerque column is selected using dplyr::select, and rows with missing values are removed using na.omit.
* **Conversion to Time Series:**
  + The Albuquerque temperature data is converted into a monthly time series object (ts) starting from January 1948 with a frequency of 12 to represent monthly data.

**Exploratory Data Analysis (EDA):**

# Explore the data

str(temp\_data)

summary(temp\_data)

# Plot the time series

plot(city\_ts, main = "Monthly Temperature for Albuquerque", ylab = "Temperature (°C)", xlab = "Time")

# Visualize Temperature Over Time with ggplot

ggplot(city\_data, aes(x = time, y = albuquerque)) +

geom\_line() +

labs(title = "Temperature Over Time - Albuquerque", x = "Date", y = "Temperature (°C)")

# Seasonality and Trend Analysis

decomposed <- decompose(city\_ts)

plot(decomposed)

# Stationarity Check with ADF Test

adf\_test\_result <- adf.test(city\_ts)

print(adf\_test\_result)

* **Structure and Summary:**
  + str() and summary() functions are used to inspect the dataset's structure and generate summary statistics.
* **Visualization:**
  + A base R plot of the time series is created to observe trends.
  + A ggplot is used for a more detailed visualization, plotting time on the x-axis and albuquerque on the y-axis, showing the temperature over time with a line chart.
* **Seasonality and Trend Analysis:**
  + A decomposition analysis using decompose is performed to separate the time series into seasonal, trend, and residual components.
* **Stationarity Check:**
  + The Augmented Dickey-Fuller (ADF) test is applied to check if the time series is stationary. Stationary data is essential for ARIMA modeling.

**ARIMA Model Implementation:**

# Stationarize the Data

city\_ts\_diff <- diff(city\_ts, differences = 1)

adf\_test\_diff\_result <- adf.test(city\_ts\_diff)

print(adf\_test\_diff\_result)

# Fit the ARIMA Model

arima\_model <- auto.arima(city\_ts)

summary(arima\_model)

# Forecast with ARIMA

arima\_forecast <- forecast(arima\_model, h = 12)

plot(arima\_forecast)

* **Stationarizing the Data:**
  + Differencing (diff) is applied to make the data stationary, and the ADF test is repeated on the differenced data.
* **Model Fitting:**
  + An ARIMA model is fit using auto.arima, which selects the best parameters for the model automatically.
* **Forecasting:**
  + The fitted ARIMA model is used to forecast the next 12 months, and the forecasted values are plotted.

**LSTM Model Implementation:**

# Scale the Data for LSTM

scaled\_data <- scale(city\_data$albuquerque)

# Reshape Data for LSTM

X\_train <- array(scaled\_data, dim = c(length(scaled\_data), 1, 1))

Y\_train <- scaled\_data

# Define the LSTM Model

model <- keras\_model\_sequential() %>%

layer\_lstm(units = 50, input\_shape = c(1, 1), return\_sequences = TRUE) %>%

layer\_lstm(units = 50, return\_sequences = FALSE) %>%

layer\_dense(units = 1)

# Compile and Train the Model

model %>% compile(

optimizer = 'adam',

loss = 'mean\_squared\_error'

)

history <- model %>% fit(

X\_train, Y\_train,

epochs = 50, batch\_size = 32,

validation\_split = 0.2

)

# Make Predictions with LSTM

lstm\_predictions <- model %>% predict(X\_train)

* **Data Scaling:**
  + The temperature data is scaled to improve LSTM model performance since neural networks often perform better with scaled data.
* **Reshaping Data:**
  + The scaled data is reshaped into a 3D array with dimensions (samples, timesteps, features) suitable for LSTM input.
* **Model Definition:**
  + A sequential LSTM model is defined:
    - First LSTM layer has 50 units and returns sequences to the next LSTM layer.
    - Second LSTM layer also has 50 units but does not return sequences.
    - A dense layer with one unit outputs the prediction.
* **Model Compilation and Training:**
  + The model is compiled with adam optimizer and mean\_squared\_error loss function.
  + The model is trained for 50 epochs with a batch size of 32, using 20% of the data for validation.
* **Prediction:**
  + The trained LSTM model generates predictions on the input data.

**Model Evaluation:**

# ARIMA Model Evaluation

arima\_accuracy <- accuracy(arima\_forecast)

print(arima\_accuracy)

# LSTM Model Evaluation

unscaled\_predictions <- lstm\_predictions \* attr(scaled\_data, "scaled:scale") + attr(scaled\_data, "scaled:center")

rmse\_lstm <- sqrt(mean((city\_data$albuquerque - unscaled\_predictions)^2))

print(paste("LSTM RMSE:", rmse\_lstm))

# Hybrid Model Evaluation

hybrid\_predictions <- (arima\_forecast$mean + unscaled\_predictions) / 2

rmse\_hybrid <- sqrt(mean((city\_data$albuquerque - hybrid\_predictions)^2))

print(paste("Hybrid RMSE:", rmse\_hybrid))

* **ARIMA Evaluation:**
  + The forecasted values from ARIMA are evaluated using metrics like RMSE and MAE through the accuracy function.
* **LSTM Evaluation:**
  + The LSTM predictions are unscaled to their original temperature values.
  + The RMSE is calculated by comparing the actual and predicted values.
* **Hybrid Model Evaluation:**
  + Hybrid predictions are calculated as the average of ARIMA and LSTM predictions.
  + The RMSE for the hybrid model is computed to evaluate its performance.

**Visualization of Results:**

# Plot Actual vs Predicted Temperatures

plot(city\_ts, main = "Temperature Forecast Comparison", ylab = "Temperature (°C)", xlab = "Time")

lines(arima\_forecast$mean, col = "blue", lty = 2)

lines(unscaled\_predictions, col = "red", lty = 2)

lines(hybrid\_predictions, col = "green", lty = 2)

# Add Legend

legend("topright", legend = c("Actual", "ARIMA", "LSTM", "Hybrid"),

col = c("black", "blue", "red", "green"), lty = 1:2)

* **Forecast Comparison Plot:**
  + Actual temperature values are plotted alongside predictions from the ARIMA, LSTM, and hybrid models.
  + Different line colors and types are used to distinguish the models.
* **Legend:**
  + A legend is added to the plot to label "Actual," "ARIMA," "LSTM," and "Hybrid" predictions.

**Code Organization:**

* **Library Loading:**
  + Relevant libraries for data manipulation (dplyr), visualization (ggplot2), time series analysis (forecast, tseries), and deep learning (keras, tensorflow) are loaded.
* **Modular Structure:**
  + Each major step is encapsulated in logically grouped sections (e.g., data preparation, EDA, model fitting).
* **Readability:**
  + Proper use of comments ensures clarity on each code segment's purpose.
* **Sequential Workflow:**
  + The workflow progresses systematically from data preparation to model evaluation and visualization.

Updated Code:

# Load Required Libraries

library(dplyr)

library(ggplot2)

library(forecast)

library(tseries)

library(keras)

library(tensorflow)

library(Metrics)

# Load the Dataset

temp\_data <- read.csv("US\_City\_Temp\_Data.csv")

temp\_data$time <- as.Date(temp\_data$time, format = "%Y-%m-%d")

# Data Exploration

str(temp\_data)

summary(temp\_data)

# Select Data for a Specific City (e.g., Albuquerque)

city\_data <- temp\_data %>% select(time, albuquerque) %>% na.omit()

# Convert Data to Time Series

city\_ts <- ts(city\_data$albuquerque, start = c(1948, 1), frequency = 12)

# Plot the Time Series

plot(city\_ts, main = "Monthly Temperature for Albuquerque", ylab = "Temperature (°C)", xlab = "Time")

# Visualize Temperature Over Time

ggplot(city\_data, aes(x = time, y = albuquerque)) +

geom\_line() +

labs(title = "Temperature Over Time - Albuquerque", x = "Date", y = "Temperature (°C)")

# Decompose the Time Series

decomposed <- decompose(city\_ts)

plot(decomposed)

# Stationarity Check

adf.test(city\_ts)

# Stationarize the Data

city\_ts\_diff <- diff(city\_ts, differences = 1)

adf.test(city\_ts\_diff)

# ARIMA Model Implementation

arima\_model <- auto.arima(city\_ts)

summary(arima\_model)

# Forecast with ARIMA

arima\_forecast <- forecast(arima\_model, h = 12)

plot(arima\_forecast)

# LSTM Model Implementation

# Scale the Data

scaled\_data <- scale(city\_data$albuquerque)

# Reshape Data for LSTM

X\_train <- array(scaled\_data, dim = c(length(scaled\_data), 1, 1))

Y\_train <- scaled\_data

# Define the LSTM Model

lstm\_model <- keras\_model\_sequential() %>%

layer\_lstm(units = 50, input\_shape = c(1, 1), return\_sequences = TRUE) %>%

layer\_lstm(units = 50, return\_sequences = FALSE) %>%

layer\_dense(units = 1)

# Compile and Train the LSTM Model

lstm\_model %>% compile(

optimizer = 'adam',

loss = 'mean\_squared\_error'

)

history <- lstm\_model %>% fit(

X\_train, Y\_train,

epochs = 50, batch\_size = 32,

validation\_split = 0.2

)

# Make Predictions with LSTM

lstm\_predictions <- lstm\_model %>% predict(X\_train)

# Hybrid ARIMA-LSTM Model

hybrid\_predictions <- (arima\_forecast$mean + lstm\_predictions) / 2

# Evaluate Models

# Convert LSTM Predictions Back to Original Scale

unscaled\_predictions <- lstm\_predictions \* attr(scaled\_data, "scaled:scale") + attr(scaled\_data, "scaled:center")

# Calculate Metrics for ARIMA

arima\_accuracy <- accuracy(arima\_forecast)

print(arima\_accuracy)

# Calculate Metrics for LSTM

rmse\_lstm <- rmse(city\_data$albuquerque, unscaled\_predictions)

print(paste("LSTM RMSE:", rmse\_lstm))

# Calculate Metrics for Hybrid Model

rmse\_hybrid <- rmse(city\_data$albuquerque, hybrid\_predictions)

print(paste("Hybrid RMSE:", rmse\_hybrid))

# Advanced Evaluation Metrics

evaluate\_forecast <- function(actual, forecast) {

list(

RMSE = rmse(actual, forecast),

MAE = mae(actual, forecast),

MAPE = mean(abs((actual - forecast) / actual)) \* 100,

SMAPE = 2 \* mean(abs(forecast - actual) / (abs(forecast) + abs(actual))) \* 100

)

}

evaluation\_metrics <- evaluate\_forecast(city\_data$albuquerque, hybrid\_predictions)

print(evaluation\_metrics)

# Time Series Cross-Validation

ts\_cv <- function(data, model\_func, h) {

errors <- c()

for (i in seq(1, length(data) - h, by = h)) {

train <- data[1:i]

test <- data[(i + 1):(i + h)]

model <- model\_func(train)

forecast <- forecast(model, h = h)

errors <- c(errors, rmse(test, forecast$mean))

}

mean(errors)

}

# Bidirectional LSTM Implementation

bidirectional\_lstm\_model <- keras\_model\_sequential() %>%

layer\_bidirectional(

layer\_lstm(units = 50, return\_sequences = TRUE),

input\_shape = c(1, 1)

) %>%

layer\_lstm(units = 50, return\_sequences = FALSE) %>%

layer\_dense(units = 1)

# Advanced Visualization

plot\_residuals <- function(model\_residuals) {

tsdisplay(model\_residuals, main = "Model Residuals")

}

plot\_residuals(residuals(arima\_model))

plot\_forecast\_uncertainty <- function(forecast) {

ggplot() +

geom\_ribbon(aes(ymin = forecast$lower[, 2], ymax = forecast$upper[, 2], x = seq\_along(forecast$mean)), alpha = 0.2) +

geom\_line(aes(y = forecast$mean, x = seq\_along(forecast$mean)), color = "blue") +

labs(title = "Forecast with Uncertainty", x = "Time", y = "Temperature")

}

plot\_forecast\_uncertainty(arima\_forecast)

# Diebold-Mariano Test

diebold\_mariano\_test <- function(forecast1, forecast2, actual) {

dm.test(forecast1 - actual, forecast2 - actual)

}

diebold\_mariano\_test(arima\_forecast$mean, unscaled\_predictions, city\_data$albuquerque)