EXPERIMENT 12

Write a R program to Apply Predictive analytics for Weather forecasting.

AIM: To Apply Predictive analytics for Weather forecasting.

Description : Predictive analytics uses historical data and statistical modeling to forecast future outcomes, determining the likelihood of specific events or trends

Weather forecasting : It involves a sequence of steps

Step 1 : Data Collection : Data collects from Ground stations like Temperature, humidity, wind speed, pressure, rainfall, etc.

Step 2 : Data Preprocessing & Quality Control : Raw data is often noisy or incomplete, so: Missing values are estimated or removed.
Outliers are detected and handled.

Step 3 : Feature Engineering : To improve model performance:
Create derived variables (e.g., wind chill, heat index).
Convert date/time into cyclical features.
Convert categorical data (e.g., weather types) into numeric encodings.

Step 4 : . Model Building : Use Statistical or Machine Learning Models:
Use historical data to learn patterns.
Common methods: Linear regression for temperature
SVM / Decision Trees for classification (e.g., rain prediction)
Time series models like ARIMA, LSTM

Step 5 : . Model Evaluation :
Models are tested using Training/testing split or cross-validation
Metrics like: RMSE / MAE for temperature
Accuracy, Precision, Recall for rain or storm predictions

Step 6 : . Forecasting : Forecasts are generated for:
Short-term (1–3 days): Highly accurate
Medium-term (4–7 days): Good reliability
Long-term (>7 days): Increasing uncertainty

Forecasts may include: Temperature, Rainfall likelihood, Wind speed and direction, Storm alerts

Program:

```
install.packages("lubridate")

install.packages("e1071")
```

```
library(lubridate)

library(e1071)


# Load & preprocess

weather_data <- read.csv("D:/R programming/weather_data.csv", stringsAsFactors =
FALSE)

weather_data$date <- dmy(weather_data$date)

weather_data$day_of_year <- yday(weather_data$date)

weather_data$month <- month(weather_data$date)

weather_data$weekday <- wday(weather_data$date)

weather_data$rain_label <- as.factor(ifelse(weather_data$rain > 0, "Yes", "No"))


# ◈ Initial dataset plot

dev.new()

plot(weather_data, main = "Weather Dataset", col = "green")


# Split data

set.seed(123)

idx <- sample(1:nrow(weather_data), size = 0.8 * nrow(weather_data))

train <- weather_data[idx, ]; test <- weather_data[-idx, ]


# Train models

lm_temp <- lm(temperature ~ humidity + pressure + day_of_year + month + weekday, data =
train)

train$predicted_temp <- predict(lm_temp, newdata = train)

svm_rain <- svm(rain_label ~ humidity + predicted_temp + pressure + day_of_year + month
+ weekday,

              data = train, type = "C-classification", kernel = "radial")
```

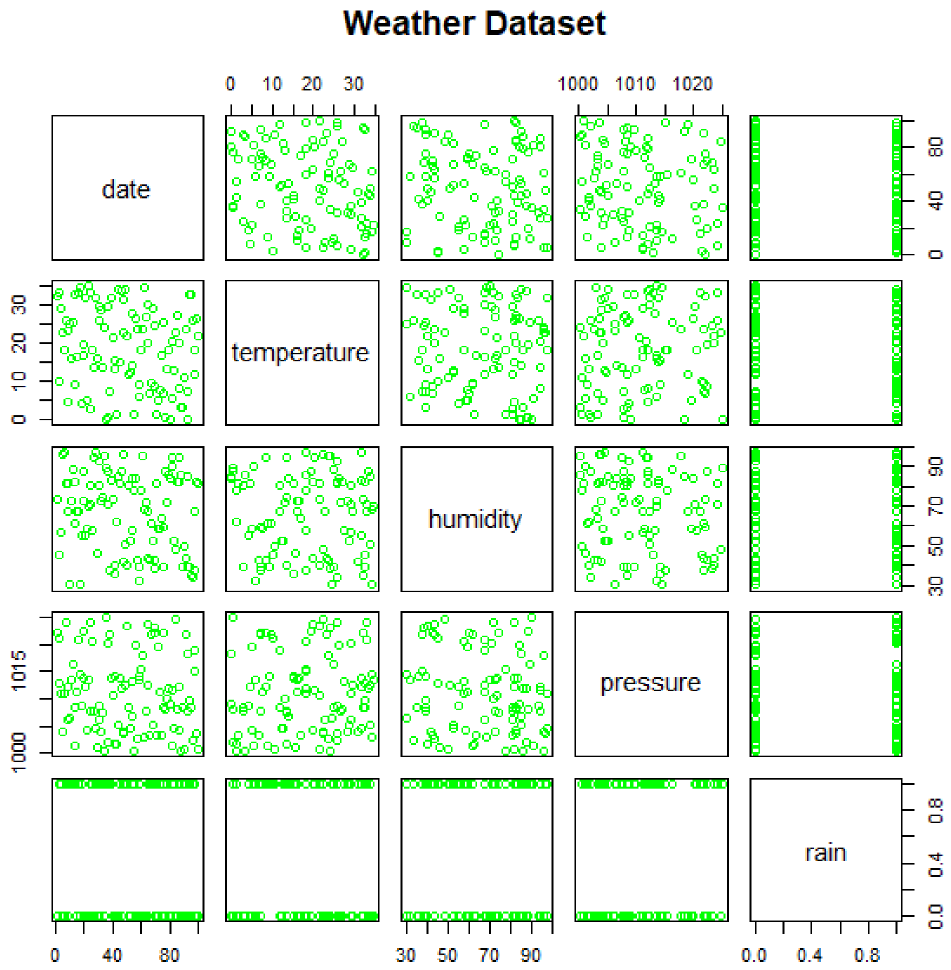
```r
# Evaluate
pred_temp <- predict(lm_temp, newdata = test)
cat("RMSE:", round(sqrt(mean((test$temperature - pred_temp)^2)), 2), "\n")
summary(lm_temp)

# 🔷 Plot actual vs predicted
dev.new()
plot(test$temperature, pred_temp, col = "blue", pch = 16,
     main = "Actual vs Predicted Temperature",
     xlab = "Actual", ylab = "Predicted")
abline(0, 1, col = "red", lwd = 2)

# Forecast next 7 days
future_dates <- seq(max(weather_data$date) + 1, by = "day", length.out = 7)
future <- data.frame(
  date = future_dates,
  day_of_year = yday(future_dates),
  month = month(future_dates),
  weekday = wday(future_dates),
  humidity = mean(train$humidity),
  pressure = mean(train$pressure)
)
future$predicted_temp <- predict(lm_temp, newdata = future)
future$predicted_temperature <- round(future$predicted_temp, 2)
future$rain_prediction <- ifelse(as.character(predict(svm_rain, newdata = future)) == "Yes",
"RAIN=YES", "RAIN=NO")

# 🔷 Show forecast
cat("\nNext 7 Days Forecast:\n")
```

```r
print(future[, c("date", "predicted_temperature", "rain_prediction")])


# 🔷 Forecast plot
dev.new()
plot(future$date, future$predicted_temperature, type = "o", col = "red", lwd = 4, pch = 5,
     main = "7-Day Forecast: Temperature & Rain",
     xlab = "Date", ylab = "Temperature (°C)")
text(future$date, future$predicted_temperature + 0.4, labels = future$predicted_temperature,
col = "red")
rain_colors <- ifelse(future$rain_prediction == "RAIN=YES", "blue", "magenta")
text(future$date, future$predicted_temperature + 1, labels = future$rain_prediction, col =
rain_colors, font = 2)
legend("topright", legend = c("Temperature", "RAIN=YES", "RAIN=NO"),
       col = c("red", "blue", "magenta"), pch = 16, bty = "n")


# 🔷 Raw temperature with rain points
dev.new()
plot(weather_data$date, weather_data$temperature, type = "o", col = "magenta", lwd = 2,
pch = 16,
     main = "Temperature Over Time", xlab = "Date", ylab = "Temperature (°C)")
points(weather_data$date[weather_data$rain > 0],
       weather_data$temperature[weather_data$rain > 0],
       col = "blue", pch = 17, cex = 1.2)
legend("topright", legend = c("Temperature", "Rainy Days"),
       col = c("magenta", "blue"), pch = c(16, 17), bty = "n")
```

OUTPUT:

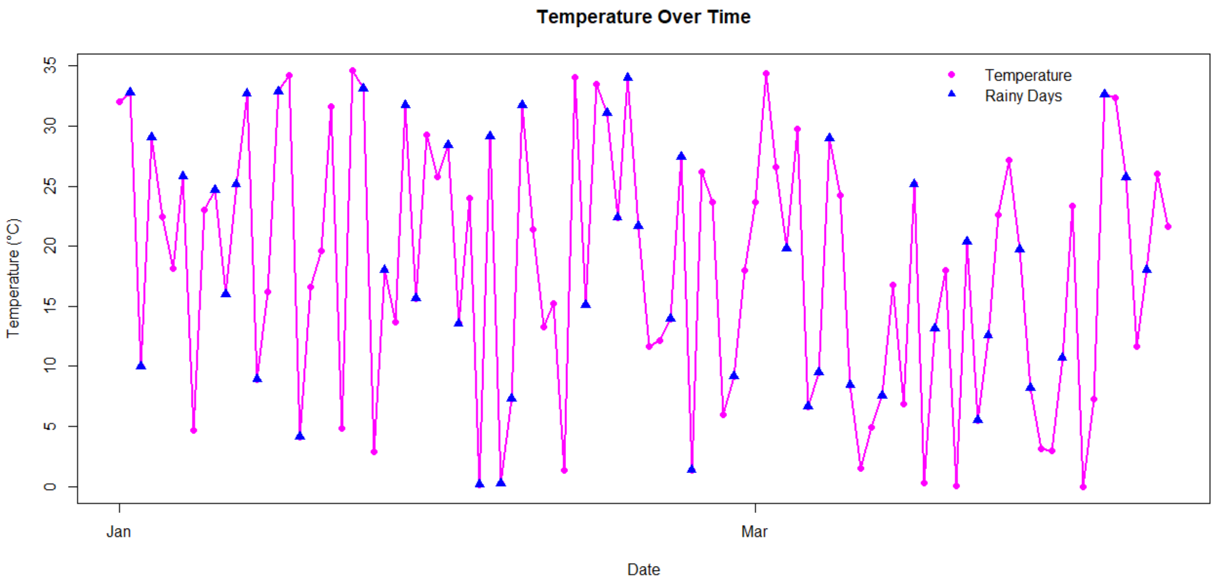
```
> head(weather_data )

        date temperature humidity pressure rain
1 2024-01-01    32.01821 73.83717 1022.128    0
```

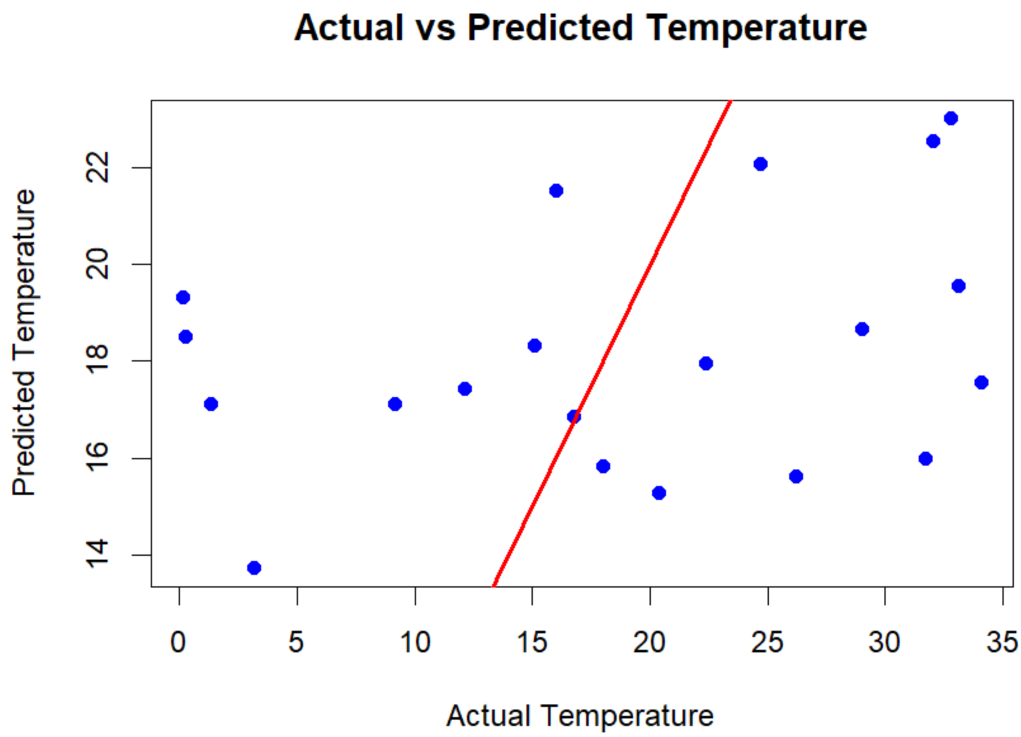
```
2 2024-01-02     32.79764 45.20104 1012.928    1
3 2024-01-03     10.01488 45.15971 1021.298    1
4 2024-01-04     29.06567 57.22615 1011.070    1
5 2024-01-05     22.46109 95.97190 1003.947    0
6 2024-01-06     18.16836 97.38256 1011.058    0
```




```
Root Mean Squared Error (RMSE): 10.86

> #Summary of Models
> summary(model_temp)

Call:
lm(formula = temperature ~ humidity + pressure + day_
of_year +
    month + weekday, data = train_data)

Residuals:
     Min       1Q   Median       3Q      Max
```

```
-16.4739  -9.5256   0.7116   8.9975  16.6716

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -35.73927  174.87503  -0.204    0.839
humidity     -0.02731    0.06225  -0.439    0.662
pressure      0.05533    0.17191   0.322    0.748
day_of_year  -0.17027    0.13703  -1.243    0.218
month         3.19049    3.97729   0.802    0.425
weekday       0.36879    0.58653   0.629    0.531

Residual standard error: 10.41 on 74 degrees of freed
om
Multiple R-squared:  0.05175,	Adjusted R-squared:
 -0.01232
F-statistic: 0.8077 on 5 and 74 DF,  p-value: 0.5479



> summary(model_rain_svm)

Call:
svm(formula = rain_label ~ humidity + predicted_temp
+
    pressure + day_of_year + month + weekday,
    data = train_data, type = "C-classification",
    kernel = "radial")

Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  72

 ( 38 34 )
Number of Classes:  2

Levels:
 No Yes
```






```
Next 7 Days Forecast:

        date predicted_temperature rain_prediction
1 2024-04-10                 15.47          RAIN=NO
```

```
2 2024-04-11                    15.67          RAIN=NO
3 2024-04-12                    15.87          RAIN=NO
4 2024-04-13                    16.07          RAIN=NO
5 2024-04-14                    13.68          RAIN=NO
6 2024-04-15                    13.88          RAIN=NO
7 2024-04-16                    14.08          RAIN=NO
```

Experiment 7 **Date :**

Experiment 1 : Write a R program to Implement ARIMA on Time Series data

Aim: Write a R program program to Implement ARIMA on Time Series data

Description :

ARIMA (Autoregressive Integrated Moving Average) is a statistical model used for time series analysis and forecasting, predicting future values by combining past observations (AR), differencing to achieve stationarity (I), and past errors to refine predictions (MA).

ARIMA models explain a given time series based on its own past values (lags) and lagged forecast errors.
Components:
Autoregressive (AR): This part of the model uses past values of the time series to predict future values.
Integrated (I): This component addresses non-stationarity by differencing the time series data, making it stationary (i.e., having a constant mean and variance over time).

Moving Average (MA): This part incorporates past forecast errors to improve the accuracy of future predictions.

Notation:
A non-seasonal ARIMA model is often represented as ARIMA(p, d, q), where:
p is the order of the autoregressive (AR) part.
d is the order of integration (the number of times the data needs to be differenced).
q is the order of the moving average (MA) part.

To build an ARIMA model:

Data Preparation: Collect and prepare the time series data.

Stationarity Check: Ensure the data is stationary or make it stationary through differencing.

Model Identification: Determine the appropriate values for p, d, and q using techniques like autocorrelation function (ACF) and partial autocorrelation function (PACF) plots.

Parameter Estimation: Estimate the model parameters using techniques like maximum likelihood estimation.

Model Evaluation: Evaluate the model's performance using metrics like root mean squared error (RMSE) or mean absolute error (MAE).

Steps involved in ARIMA Model :

1. Load and Prepare the Time Series Data

For demonstration, we use the built-in AirPassengers dataset.

2. Check for Stationarity

ARIMA requires a stationary series, meaning that statistical properties like mean and variance should be constant over time.

If p-value > 0.05, the data is non-stationary, and we apply differencing.

If p-value ≤ 0.05, the data is stationary.

3. Apply Differencing (If Necessary)

If the time series is non-stationary, differencing is required.

4. Identify ARIMA Parameters (p, d, q)

Determine ARIMA parameters manually using ACF (AutoCorrelation Function) and PACF (Partial AutoCorrelation Function) plots.

Applications:

ARIMA models are widely used for various time series forecasting tasks, including:
Predicting stock prices.
Forecasting sales and demand.
Analyzing financial data.
Understanding and predicting trends in various datasets

Program:

```
# Install and load required packages

if (!require(forecast)) install.packages("forecast", dependencies = TRUE)
```

```
if (!require(tseries)) install.packages("tseries", dependencies = TRUE)
library(forecast)
library(tseries)

# Load time series data
ts_data <- AirPassengers

# 📌 Plot original time series
dev.new()
plot(ts_data, main = "AirPassengers Time Series", ylab = "Passengers", col = "blue")

# 📌 ADF test for stationarity
adf_result <- adf.test(ts_data)
print(adf_result)

# 📌 ACF & PACF plots
dev.new(); acf(ts_data, main = "ACF Plot")
dev.new(); pacf(ts_data, main = "PACF Plot")

# 📌 Apply differencing if needed
ts_diff <- if (adf_result$p.value > 0.05) {
  print("Differencing applied.")
  diff(ts_data)
} else ts_data

# 📌 Re-check ADF test
print(adf.test(ts_diff, na.action = na.omit))
```

📌 Fit best ARIMA model

best_arima <- auto.arima(ts_data)

summary(best_arima)

📌 Forecast next 12 months

fc <- forecast(best_arima, h = 12)

📌 Plot forecast

dev.new(); plot(fc, main = "ARIMA Forecast", col = "blue")

📌 Forecast values

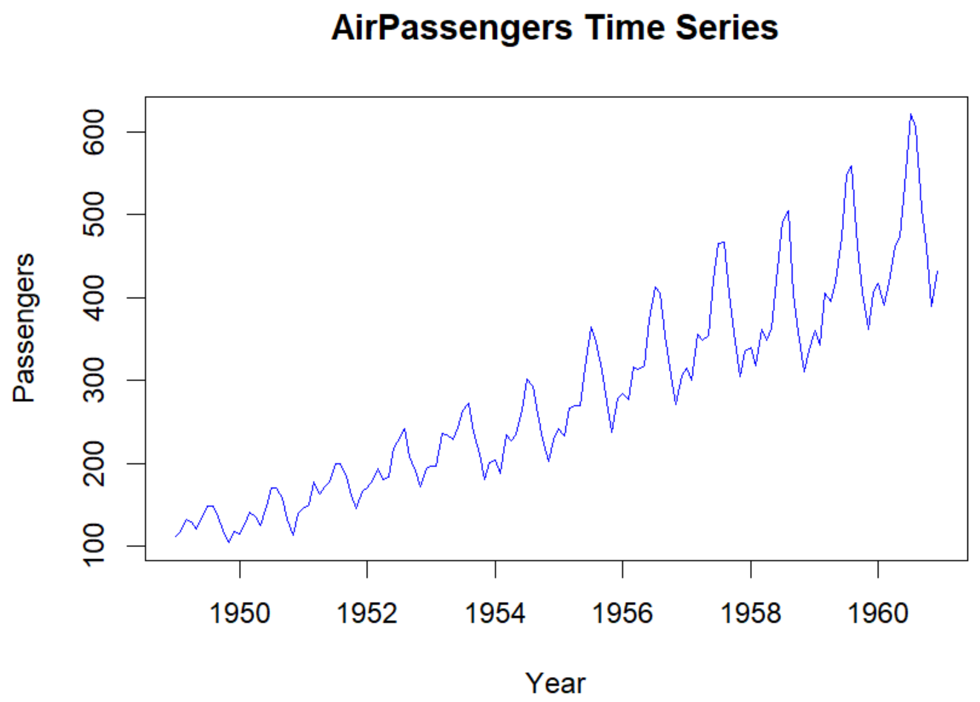
print(fc)

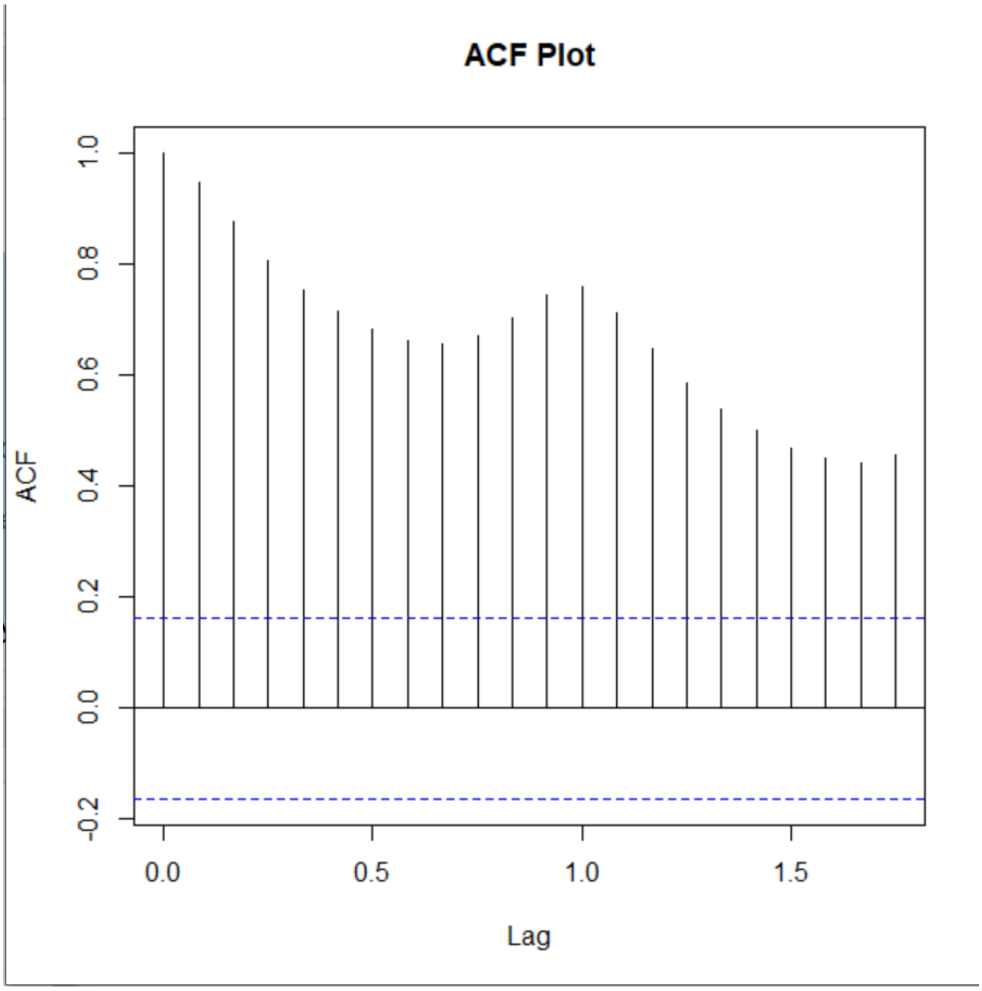
📌 Residual diagnostics

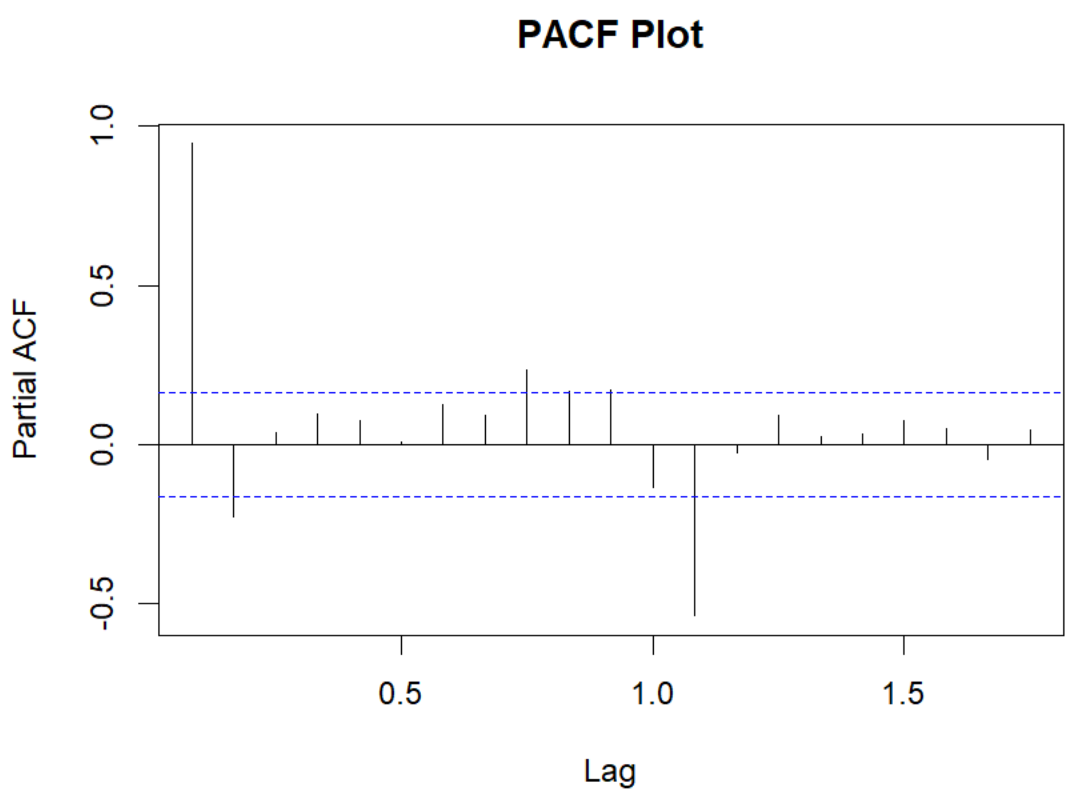
checkresiduals(best_arima)

Output :

```
Augmented Dickey-Fuller Test

data:  ts_data
Dickey-Fuller = -7.3186, Lag order = 5, p-value =
0.01
alternative hypothesis: stationary
```


AirPassengers Time Series
Passengers
100
200
300
400
500
600
1950
1952
1954
1956
1958
1960
Year


ACF Plot
ACF
-0.2
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.5
1.0
1.5
Lag



PACF Plot
Partial ACF
1.0
0.5
0.0
-0.5
0.5
1.0
1.5
Lag