```r
# Simple Linear Regression

cars

plot=scatter.smooth(x=cars$speed, y=cars$dist, main="Dist Vs Speed")


par(mfrow=c(1,2))  # divide graph area in 2 columns

boxplot(cars$speed,main="Speed",sub=paste("Outlier rows: ",
boxplot.stats(cars$speed)$out))

boxplot(cars$dist,main="Distance",sub=paste("Outlier rows: ",
boxplot.stats(cars$dist)$out))


cor(cars$speed,cars$dist)


linear_Model=lm(dist ~ speed, data=cars)

linear_Model


modelSummary=summary(linear_Model)

modelSummary



# Using manual code

modelCoeffs=modelSummary$coefficients

modelCoeffs


n=nrow(cars)

n

alpha=0.05

beta1_estimate=modelCoeffs["speed","Estimate"]

beta1_estimate

std_error=modelCoeffs["speed", "Std. Error"]
```

```
std_error

# Classical Approach
t_cal=beta1_estimate/std_error
t_cal

t_table=qt((alpha/2),n-2,lower.tail = F)
t_table

if(t_cal>t_table) print("reject the null hypothesis that the co-efficient of the predictor is zero") else print("accept the null hypothesis")

# p-value approach

p_value = 2*(1-pt(abs(t_cal),df=nrow(cars)-2))
p_value

if(p_value<0.05)print("reject the null hypothesis that the co-efficient of the predictor is zero") else print("accept the null hypothesis")



f=summary(linear_Model)$fstatistic
f

f_table=qf(.95, df1=f[2], df2=f[3])
f_table

if(f[1]>f_table)print("reject the null hypothesis that the co-efficient of the predictor is zero") else print("accept the null hypothesis")
```

```
summary(linear_Model)$r.squared
summary(linear_Model)$adj.r.squared

confint(linear_Model)

# RESIDUAL ANALYSIS & FORECAST ACCURACY FOR A GIVEN DATASET

# Install the required packages
install.packages("UsingR")
library(UsingR)
install.packages("lmtest")
library(lmtest)
df=diamond
df

# Fit Simple Linear Regression
plot(df$carat,df$price)
model=lm(df$price~df$carat,data=df)
summary(model)

# Extract Model Components
obs_val=df$price
obs_val
fitted_val=fitted(model)
fitted_val
residuals=resid(model)
residuals
```

```
# Residual Analysis
# Residuals vs Fitted Plot
plot(fitted_val,residuals,
    xlab = "Fitted Values",
    ylab = "Residuals",
    main = "Residuals vs Fitted Values")
abline(h = 0, lty = 2, col = "red")
# Random scatter around zero → model is appropriate
# Funnel shape → heteroscedasticity
# Curved pattern → non linearity


# Breusch-Pagan Test(Homoscedasticity)
bp_test=bptest(model)
bp_test
# Decision
# p > 0.05 → homoscedasticity satisfied

# Normal Q-Q Plot
qqnorm(residuals)
qqline(residuals, col = "red")

# Shapiro-Wilk Normality Test
shapiro_test=shapiro.test(residuals)
shapiro_test
# Decision rule
# p>0.05 → residuals are approximately normal
```

```
# Durbin-Watson Test(Independence of Residuals)

dw_test=dwtest(model)

dw_test

# Conclusion

# The Durbin-Watson statistic will always have a value ranging between 0 and 4.

# A value of 2.0 indicates there is no autocorrelation detected in the sample.

# Values from 0 to less than 2 point to positive autocorrelation.

# Values from 2 to 4 mean negative autocorrelation.


# Cook’s Distance (Influential Observations)

# Influential points can distort regression coefficients

# They may Inflate or deflate slope estimates

# Affect hypothesis tests

# Reduce forecast accuracy


plot(cooks.distance(model),

    type = "h",

    main = "Cook's Distance")

abline(h = 4/length(residuals), lty = 2, col = "red")

# In a Cook’s Distance plot:

# Each vertical line represents one observation

# A horizontal reference line is drawn at 4 / n

# Points above the line deserve further investigation


# Decision

# Di < 1          Observation is not influential

# Di ≥ 1          Observation is highly influential
```

```r
# Di > 4/n Potentially influential (commonly used rule)

# Forecasting (Prediction)
# Provide new X values
new_df=data.frame(X=c(0.10,0.12,0.15))
new_df
predicted_future=with(new_df,{coef(model)[1] + coef(model)[2] * X})
predicted_future

# Forecast Accuracy Measures
# Mean Absolute Error (MAE)
# The MAE indicates that, on average, the model's predictions deviate
# from the actual values by MAE units
# A smaller MAE suggests better predictive accuracy
# Lower MAE → better model
MAE=mean(abs(obs_val - fitted_val))
MAE

# Mean Squared Error (MSE)
# The MSE penalizes large forecast errors more heavily,
# indicating the presence or absence of extreme prediction errors
# Sensitive to large errors
MSE=mean((obs_val - fitted_val)^2)
MSE

# Root Mean Squared Error (RMSE)
# The RMSE represents the standard deviation of the forecast errors.
# A lower RMSE implies higher forecast reliability
```

```
RMSE=sqrt(MSE)
RMSE

# Mean Absolute Percentage Error (MAPE)
# The MAPE value of X% indicates that the forecasts deviate from the
# actual values by X% on average
# < 10% - Highly accurate
# 10–20% - Good
# 20–50% - Reasonable
# > 50% - Poor
MAPE=mean(abs((obs_val - fitted_val) / obs_val)) * 100
MAPE

# R-squared
R_squared=summary(model)$r.squared
R_squared

# Print Accuracy Metrics
cat("\nForecast Accuracy Measures:\n")
cat("MAE  =", MAE, "\n")
cat("MSE  =", MSE, "\n")
cat("RMSE =", RMSE, "\n")
cat("MAPE =", MAPE, "%\n")
cat("R^2  =", R_squared, "\n")

# Train-Test Split Accuracy
df1=as.data.frame(df)
df1
```

```
set.seed(123)
train_index=sample(1:nrow(df1),0.8*nrow(df1))
train_index

train_data=df1[train_index, ]
train_data
test_data=df1[-train_index, ]
test_data

model_train=lm(train_data$price~train_data$carat,data=train_data)
model_train
test_pred=with(test_data,{coef(model_train)[1]+coef(model_train)[2]*test_data$carat})
test_pred

MAE_test=mean(abs(test_data$price - test_pred))
MAE_test
RMSE_test=sqrt(mean((test_data$price - test_pred)^2))
RMSE_test
MAPE_test=mean(abs((test_data$price - test_pred) / test_data$price)) * 100
MAPE_test

cat("\nTest Set Forecast Accuracy:\n")
cat("MAE  =", MAE_test, "\n")
cat("RMSE =", RMSE_test, "\n")
cat("MAPE =", MAPE_test, "%\n")
```