Practical 7:- Write a program to Perform a t-test (one-sample, independent) on sample data and interpret the results.

```
from scipy import stats
import numpy as np

# One-sample t-test
# Sample data: Let's say we have test scores of a group and we want to test if the mean score is significantly different from a hypothetical population mean.
sample_data_one = [85, 89, 78, 92, 88, 91, 84, 90]
population_mean = 80  # Hypothetical population mean

# Perform the one-sample t-test
t_stat_one, p_value_one = stats.ttest_1samp(sample_data_one, population_mean)
print("One-sample t-test:")
print(f"T-statistic: {t_stat_one:.2f}, P-value: {p_value_one:.3f}")

# Interpret results for one-sample t-test
alpha = 0.05  # Significance level
if p_value_one < alpha:
    print("Result: Reject the null hypothesis. The sample mean is significantly different from the population mean.")
else:
    print("Result: Fail to reject the null hypothesis. The sample mean is not significantly different from the population mean.")

# Independent (two-sample) t-test
# Sample data for two groups (e.g., scores of students from two different classes)
class_a_scores = [85, 88, 78, 92, 88, 91, 84, 90]
class_b_scores = [82, 86, 80, 85, 87, 79, 83, 84]

# Perform the independent (two-sample) t-test
t_stat_ind, p_value_ind = stats.ttest_ind(class_a_scores, class_b_scores)
print("\nIndependent (two-sample) t-test:")
print(f"T-statistic: {t_stat_ind:.2f}, P-value: {p_value_ind:.3f}")

# Interpret results for independent (two-sample) t-test
if p_value_ind < alpha:
    print("Result: Reject the null hypothesis. There is a significant difference between the two groups.")
else:
    print("Result: Fail to reject the null hypothesis. There is no significant difference between the two groups.")
```

Output:-

```
One-sample t-test:
T-statistic: 5.16, P-value: 0.002
Result: Reject the null hypothesis. The sample mean is significantly different from the population mean.

Independent (two-sample) t-test:
T-statistic: 1.53, P-value: 0.142
Result: Fail to reject the null hypothesis. There is no significant difference between the two groups.
```

Practical 8:- Write a program to Calculate and interpret the Pearson and Spearman correlation coefficients.

```
import numpy as np
import scipy.stats as stats
# Sample data
x = [10, 20, 30, 40, 50]  # Variable 1
y = [15, 25, 35, 45, 55]  # Variable 2

# Calculate Pearson Correlation Coefficient
pearson_corr, pearson_p_value = stats.pearsonr(x, y)
print("Pearson Correlation Coefficient:", pearson_corr)
print("P-value for Pearson Correlation:", pearson_p_value)

# Interpretation of Pearson
if pearson_p_value < 0.05:
    print("The Pearson correlation is statistically significant.")
else:
    print("The Pearson correlation is not statistically significant.")
if pearson_corr > 0:
    print("Positive linear relationship.")
elif pearson_corr < 0:
    print("Negative linear relationship.")
else:
    print("No linear relationship.")

# Calculate Spearman Correlation Coefficient
spearman_corr, spearman_p_value = stats.spearmanr(x, y)
print("\nSpearman Correlation Coefficient:", spearman_corr)
print("P-value for Spearman Correlation:", spearman_p_value)

# Interpretation of Spearman
if spearman_p_value < 0.05:
    print("The Spearman correlation is statistically significant.")
else:
    print("The Spearman correlation is not statistically significant.")
if spearman_corr > 0:
    print("Positive monotonic relationship.")
elif spearman_corr < 0:
    print("Negative monotonic relationship.")
else:
    print("No monotonic relationship.")
```

Output:-

```
Pearson Correlation Coefficient: 1.0
P-value for Pearson Correlation: 0.0
The Pearson correlation is statistically significant.
Positive linear relationship.

Spearman Correlation Coefficient: 1.0
P-value for Spearman Correlation: 0.0
The Spearman correlation is statistically significant.
Positive monotonic relationship.
```

Practical 9:- Write a program to implement Conduct ANOVA (One-Way and Two-Way) test.

```
import numpy as np
import pandas as pd
from scipy.stats import f_oneway
import statsmodels.api as sm
from statsmodels.formula.api import ols

# ==============================
# One-Way ANOVA
# ==============================

# Sample data for one-way ANOVA
group1 = [23, 45, 67, 34, 45]  # Group 1 data
group2 = [25, 46, 69, 36, 48]  # Group 2 data
group3 = [22, 43, 65, 33, 47]  # Group 3 data

# Conduct one-way ANOVA
f_stat, p_value = f_oneway(group1, group2, group3)

print("One-Way ANOVA Results:")
print(f"F-statistic: {f_stat}")
print(f"P-value: {p_value}")

# Interpretation of One-Way ANOVA results
if p_value < 0.05:
   print("At least one group is significantly different.")
else:
   print("No significant differences between groups.")

# ==============================
# Two-Way ANOVA
# ==============================

# Sample data for two-way ANOVA
data = {
   'Factor_A': ['Low', 'Low', 'Low', 'High', 'High', 'High', 'High', 'Low', 'High', 'Low'],
   'Factor_B': ['A', 'A', 'B', 'A', 'B', 'B', 'A', 'B', 'A', 'B'],
   'Output': [33, 45, 42, 38, 47, 41, 49, 44, 48, 37]
}

# Convert to pandas DataFrame
df = pd.DataFrame(data)

# Perform two-way ANOVA
```

```
model = ols('Output ~ C(Factor_A) + C(Factor_B) + C(Factor_A):C(Factor_B)', data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

print("\nTwo-Way ANOVA Results:")
print(anova_table)

# Interpretation of Two-Way ANOVA results
print("\nInterpretation of Two-Way ANOVA:")
if anova_table["PR(>F)"]["C(Factor_A)"] < 0.05:
   print("- Factor A has a significant effect.")
else:
   print("- Factor A does not have a significant effect.")

if anova_table["PR(>F)"]["C(Factor_B)"] < 0.05:
   print("- Factor B has a significant effect.")
else:
   print("- Factor B does not have a significant effect.")

if anova_table["PR(>F)"]["C(Factor_A):C(Factor_B)"] < 0.05:
   print("- Interaction between Factor A and Factor B is significant.")
else:
   print("- Interaction between Factor A and Factor B is not significant.")
```

Output:-

```
One-Way ANOVA Results:
F-statistic: 0.141
P-value: 0.868
No significant differences between groups.


Two-Way ANOVA Results:
                   sum_sq   df         F    PR(>F)
C(Factor_A)        36.9000  1.0  10.542857  0.0201
C(Factor_B)        12.1000  1.0   3.457143  0.1087
C(Factor_A):C(Factor_B) 0.1000  1.0   0.028571  0.8723
Residual           35.0000  6.0       NaN     NaN

Interpretation of Two-Way ANOVA:
- Factor A has a significant effect.
- Factor B does not have a significant effect.
- Interaction between Factor A and Factor B is not significant.
```

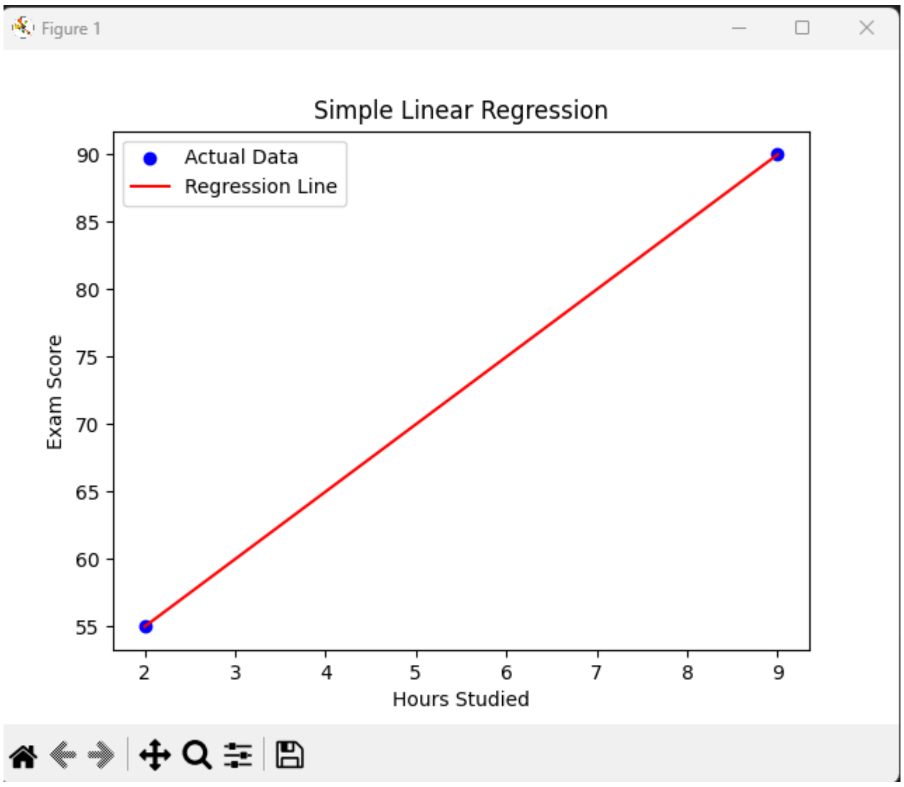
Practical 10:- Write a program to implement simple linear regression, Multiple Linear Regression on given dataset.

```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# ============================
# Example Dataset
# ============================
# Creating a sample dataset
data = {
    'Hours_Studied': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Tutor_Support': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
    'Exam_Score': [50, 55, 60, 65, 70, 75, 80, 85, 90, 95]
}

# Convert to DataFrame
df = pd.DataFrame(data)

# ============================
# Split Dataset into Features and Target
# ============================
# Features and target variable
X_simple = df[['Hours_Studied']]  # For Simple Linear Regression
X_multiple = df[['Hours_Studied', 'Tutor_Support']]  # For Multiple Linear Regression
y = df['Exam_Score']

# Train-Test Split
X_train_simple, X_test_simple, y_train, y_test = train_test_split(X_simple, y, test_size=0.2,
random_state=42)
X_train_multiple, X_test_multiple, _, _ = train_test_split(X_multiple, y, test_size=0.2,
random_state=42)

# ============================
# Simple Linear Regression
# ============================
print("=== Simple Linear Regression ===")
# Initialize and train the model
simple_model = LinearRegression()
simple_model.fit(X_train_simple, y_train)
```

```
# Predict using the simple model
y_pred_simple = simple_model.predict(X_test_simple)

# Model evaluation
print(f"Coefficients: {simple_model.coef_}")
print(f"Intercept: {simple_model.intercept_}")
print(f"Mean Squared Error (MSE): {mean_squared_error(y_test, y_pred_simple):.2f}")
print(f"R-squared (R2): {r2_score(y_test, y_pred_simple):.2f}")

# Plot the results
plt.scatter(X_test_simple, y_test, color='blue', label='Actual Data')
plt.plot(X_test_simple, y_pred_simple, color='red', label='Regression Line')
plt.xlabel("Hours Studied")
plt.ylabel("Exam Score")
plt.title("Simple Linear Regression")
plt.legend()
plt.show()

# ============================
# Multiple Linear Regression
# ============================
print("\n=== Multiple Linear Regression ===")
# Initialize and train the model
multiple_model = LinearRegression()
multiple_model.fit(X_train_multiple, y_train)

# Predict using the multiple model
y_pred_multiple = multiple_model.predict(X_test_multiple)

# Model evaluation
print(f"Coefficients: {multiple_model.coef_}")
print(f"Intercept: {multiple_model.intercept_}")
print(f"Mean Squared Error (MSE): {mean_squared_error(y_test, y_pred_multiple):.2f}")
print(f"R-squared (R2): {r2_score(y_test, y_pred_multiple):.2f}")
```

Output:-

```
=== Simple Linear Regression ===
Coefficients: [5.]
Intercept: 45.0
Mean Squared Error (MSE): 0.00
R-squared (R2): 1.00

=== Multiple Linear Regression ===
Coefficients: [5. 0.]
Intercept: 45.0
Mean Squared Error (MSE): 0.00
R-squared (R2): 1.00
```



Practical 11:- Write a program to implement simple logistic regression on given dataset.

```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_curve, auc

# ============================
# Example Dataset
# ============================
# Creating a sample dataset
data = {
    'Study_Hours': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Pass_Exam': [0, 0, 0, 0, 1, 1, 1, 1, 1, 1]  # Binary output: 0 -> Fail, 1 -> Pass
}

# Convert to DataFrame
df = pd.DataFrame(data)

# ============================
# Split Dataset into Features and Target
# ============================
# Features and target variable
X = df[['Study_Hours']]  # Independent variable
y = df['Pass_Exam']      # Dependent variable (binary classification)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# ============================
# Logistic Regression Model
# ============================
# Initialize and train the model
logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)

# Predict on test data
y_pred = logistic_model.predict(X_test)
y_prob = logistic_model.predict_proba(X_test)[:, 1]  # Probabilities for class 1
# ============================
# Model Evaluation
# ============================
print("=== Logistic Regression Evaluation ===")
# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print(f"Confusion Matrix:\n{cm}")

# Classification Report
cr = classification_report(y_test, y_pred)
print(f"Classification Report:\n{cr}")

# ============================
# Visualization
# ============================
# Scatter plot of data
plt.scatter(X, y, color='blue', label='Actual Data')

# Line plot of predicted probabilities
x_vals = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)
y_vals = logistic_model.predict_proba(x_vals)[:, 1]
plt.plot(x_vals, y_vals, color='red', label='Logistic Curve')

plt.xlabel("Study Hours")
plt.ylabel("Probability of Passing")
plt.title("Logistic Regression Curve")
plt.legend()
plt.show()

# ============================
# ROC Curve and AUC
# ============================
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

# Plot ROC Curve
plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', lw=2)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

Output:-

```
=== Logistic Regression Evaluation ===
Accuracy: 1.00
Confusion Matrix:
[[1 0]
 [0 1]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         1

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2
```

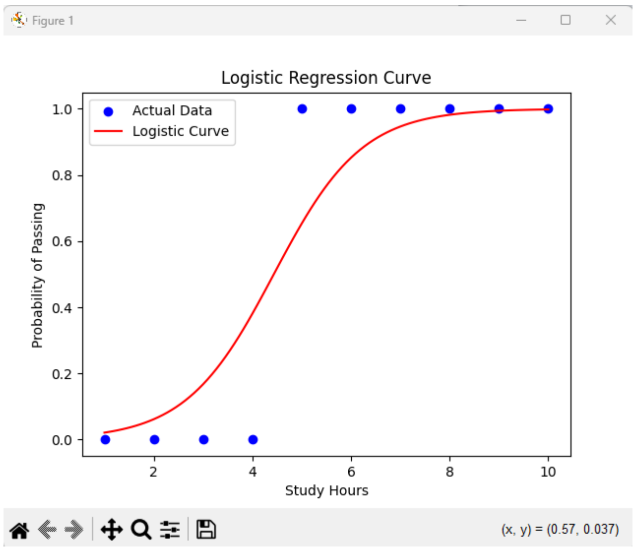


Figure 1
Logistic Regression Curve
Actual Data
Logistic Curve
Probability of Passing
Study Hours
(x, y) = (0.57, 0.037)

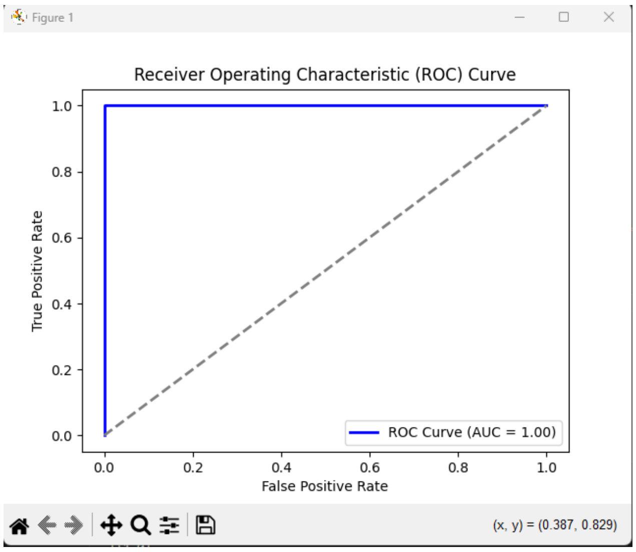


Figure 1
Receiver Operating Characteristic (ROC) Curve
ROC Curve (AUC = 1.00)
True Positive Rate
False Positive Rate
(x, y) = (0.387, 0.829)