Institute of Distance and Open Learning

Vidya Nagari, Kalina, Santacruz East – 400098

CERTIFICATE

This is to certify that Mr/Ms. ____ of **Master in Computer Application** (MCA) **Semester I** of **FYMCA** with Examination Seat No. ____________________ & Application No.____________________________ has completed the specified term work in the subject of ____ at PCP center of **Vidyavardhini's College Of Engineering & Technology**, Vasai Road satisfactorily within this institute as laid down by University of Mumbai during the academic year **2025 to 2026**.

Subject in-charge	External Examination	Coordinator - MCA

University of Mumbai

Institute of Distance and Open Learning (IDOL)

PCP CENTER: DTSS, MALAD

INDEX

Subject:

Practical No. 1

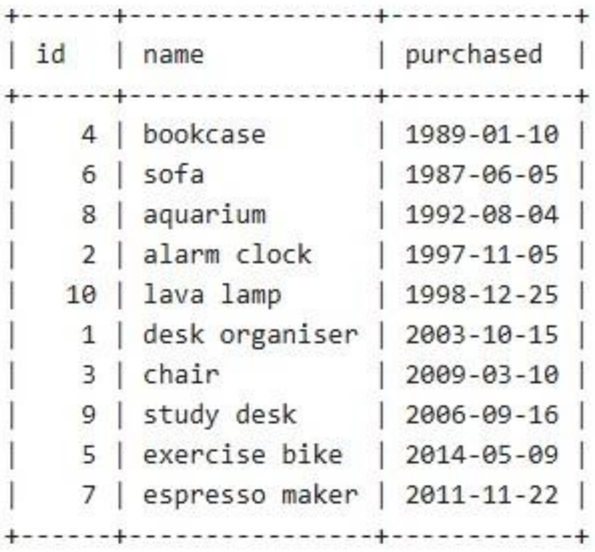
Aim: Implementation of Data partitioning through Range

Objective: A table that is partitioned by range is partitioned in such a way that each partition contains rows for which the partitioning expression value lies within a given range. Ranges should be contiguous but not overlapping, and are defined using the **VALUES LESS THAN** operator. Below example demonstrate the partitioning in range into mysql database.

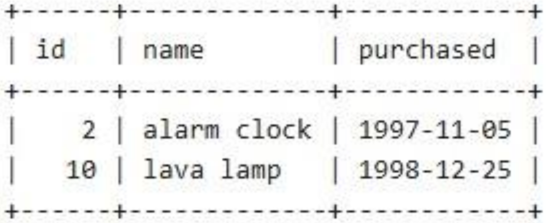
CODE:

```
CREATE TABLE tr (

   id INT,     name
VARCHAR(50),

   purchased DATE

)

PARTITION BY RANGE(YEAR(purchased))

(

   PARTITION p0 VALUES LESS THAN (1990),

   PARTITION p1 VALUES LESS THAN (1995),

   PARTITION p2 VALUES LESS THAN (2000),
   PARTITION p3 VALUES LESS THAN (2005),

   PARTITION p4 VALUES LESS THAN (2010),

   PARTITION p5 VALUES LESS THAN (2015),

   PARTITION p_future VALUES LESS THAN MAXVALUE
);


INSERT INTO tr VALUES

(1, 'desk organiser', '2003-10-15'),
(2, 'alarm clock', '1997-11-05'),

(3, 'chair', '2009-03-10'),

(4, 'bookcase', '1989-01-10'),

(5, 'exercise bike', '2014-05-09'),
(6, 'sofa', '1987-06-05'),
(7, 'espresso maker', '2011-11-22'),
(8, 'aquarium', '1992-08-04'),
(9, 'study desk', '2006-09-16'),
(10, 'lava lamp', '1998-12-25');

SELECT * FROM tr;

SELECT * FROM tr PARTITION (p2);

SELECT * FROM tr PARTITION (p5);

SELECT * FROM tr PARTITION (p4);

SELECT
   PARTITION_NAME,
   TABLE_ROWS
FROM
   INFORMATION_SCHEMA.PARTITIONS
WHERE
   TABLE_NAME = 'tr';
```

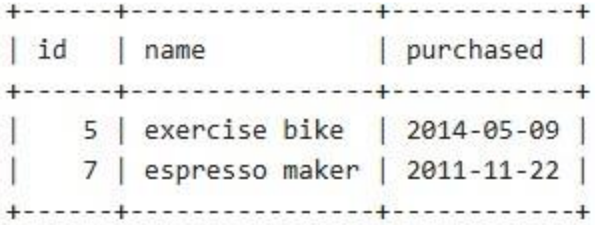
OUTPUT:



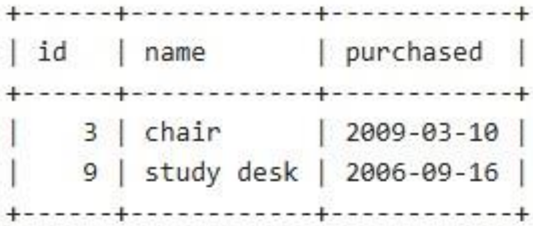
```
+------+----------------+------------+
| id   | name           | purchased  |
+------+----------------+------------+
|    4 | bookcase       | 1989-01-10 |
|    6 | sofa           | 1987-06-05 |
|    8 | aquarium       | 1992-08-04 |
|    2 | alarm clock    | 1997-11-05 |
|   10 | lava lamp      | 1998-12-25 |
|    1 | desk organiser | 2003-10-15 |
|    3 | chair          | 2009-03-10 |
|    9 | study desk     | 2006-09-16 |
|    5 | exercise bike  | 2014-05-09 |
|    7 | espresso maker | 2011-11-22 |
+------+----------------+------------+
```



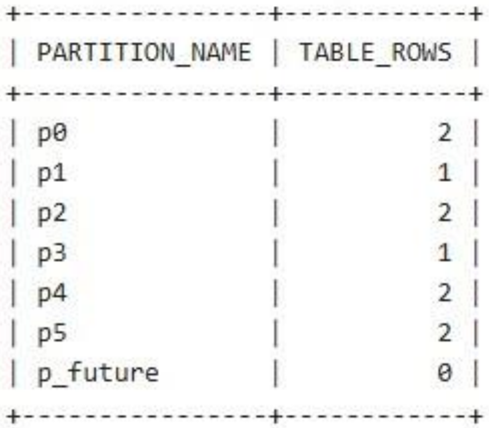
```
+------+-------------+------------+
| id   | name        | purchased  |
+------+-------------+------------+
|    2 | alarm clock | 1997-11-05 |
|   10 | lava lamp   | 1998-12-25 |
+------+-------------+------------+
```



```
+------+----------------+------------+
| id   | name           | purchased  |
+------+----------------+------------+
|    5 | exercise bike  | 2014-05-09 |
|    7 | espresso maker | 2011-11-22 |
+------+----------------+------------+
```



```
+------+------------+------------+
| id   | name       | purchased  |
+------+------------+------------+
|    3 | chair      | 2009-03-10 |
|    9 | study desk | 2006-09-16 |
+------+------------+------------+
```



```
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |          2 |
| p1             |          1 |
| p2             |          2 |
| p3             |          1 |
| p4             |          2 |
| p5             |          2 |
| p_future       |          0 |
+----------------+------------+
```

Practical No. 2

Aim: Implementation of Analytical queries like **Roll_UP**, **CUBE**, **First**, **Last**

Concept:

The last decade has seen a tremendous increase in the use of query, reporting, and on-line analytical processing (**OLAP**) tools, often in conjunction with data warehouses and data marts. Enterprises accustomed to easy and fast computer consumption expect these tools to provide the maximum possible decision-making value from their data resources.

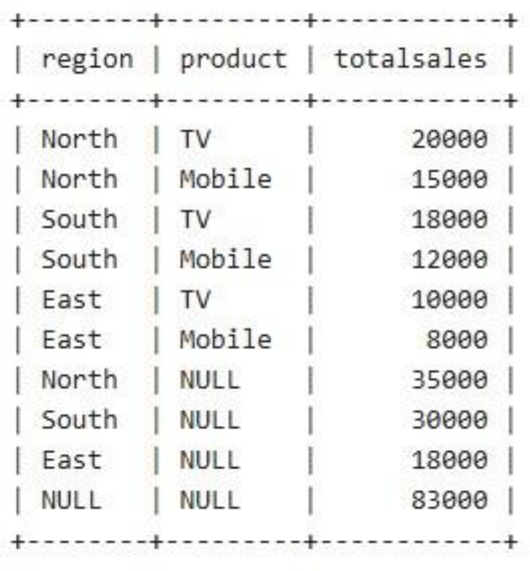
ROLLUP and CUBE are complex extensions to the SELECT statement's GROUP BY clause. ROLLUP creates subtotals at any level of aggregation needed, from the most detailed up to a grand total. CUBE is an extension similar to ROLLUP, enabling a single statement to calculate all possible combinations of subtotals.

PROGRAM:

```
CREATE TABLE sales (
   region VARCHAR(20),
   product VARCHAR(20),
   salesamount INT );



INSERT INTO sales (region, product, salesamount) VALUES ('North', 'TV',
20000);
INSERT INTO sales (region, product, salesamount) VALUES ('North',
'Mobile', 15000);
INSERT INTO sales (region, product, salesamount) VALUES ('South', 'TV',
18000);
INSERT INTO sales (region, product, salesamount) VALUES ('South',
'Mobile', 12000);
INSERT INTO sales (region, product, salesamount) VALUES ('East', 'TV',
10000);
INSERT INTO sales (region, product, salesamount) VALUES ('East', 'Mobile',
8000);
```

```
-- Detail region + product totals
SELECT
    region,
    product,     SUM(salesamount) AS totalsales

FROM sales
GROUP BY region, product

UNION ALL

-- Region subtotal
SELECT     region,

    NULL AS product,
    SUM(salesamount) AS totalsales
FROM sales
GROUP BY region
UNION ALL


-- Grand total (all regions)
SELECT
    NULL AS region,
    NULL AS product,
    SUM(salesamount) AS totalsales
FROM sales;
```

OUTPUT:



```
+--------+---------+------------+
| region | product | totalsales |
+--------+---------+------------+
| North  | TV      |      20000 |
| North  | Mobile  |      15000 |
| South  | TV      |      18000 |
| South  | Mobile  |      12000 |
| East   | TV      |      10000 |
| East   | Mobile  |       8000 |
| North  | NULL    |      35000 |
| South  | NULL    |      30000 |
| East   | NULL    |      18000 |
| NULL   | NULL    |      83000 |
+--------+---------+------------+
```

Practical No. 3

Aim: Implementation of Abstract Data Type & Reference

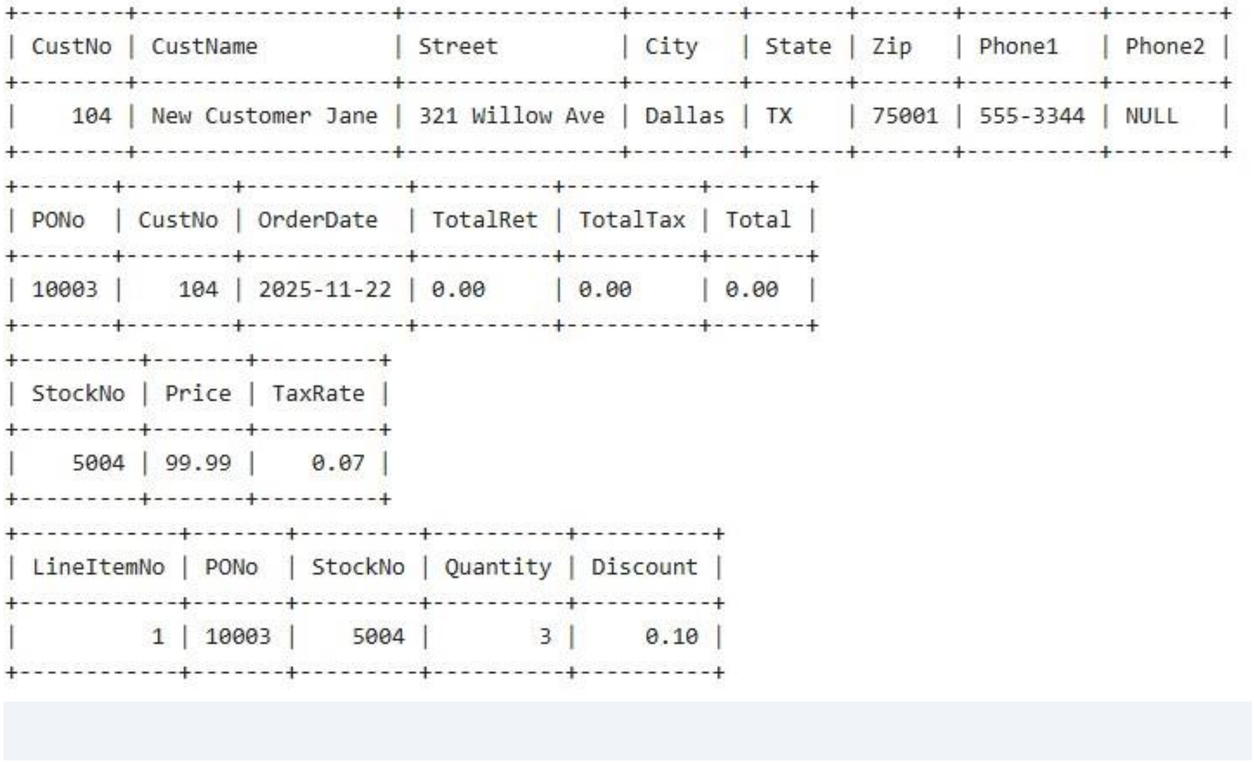
Objective:

The relational approach normalizes everything into tables. The table names are Customer_reltab, PurchaseOrder_reltab, and Stock_reltab. Each part of an address becomes a column in the Customer_reltab table. Structuring telephone numbers as columns sets an arbitrary limit on the number of telephone numbers a customer can have. The relational approach separates line items from their purchase orders and puts each into its own table, named Purchase Order_reltab and LineItems_reltab. The relational approach results in the tables describe in the following sections.

PROGRAM:

```
CREATE TABLE Customer_reltab (

   CustNo INT NOT NULL,

   CustName VARCHAR(200) NOT NULL,

   Street VARCHAR(200) NOT NULL,

   City VARCHAR(200) NOT NULL,

   State VARCHAR(20) NOT NULL,

   Zip VARCHAR(20),

   Phone1 VARCHAR(20),

   Phone2 VARCHAR(20),

   PRIMARY KEY (CustNo)

);

CREATE TABLE PurchaseOrder_reltab (

   PONo INT,
   CustNo INT,
   OrderDate DATE,
   TotalRet VARCHAR(200),
   TotalTax VARCHAR(200),
   Total VARCHAR(200),
   PRIMARY KEY (PONo),
   FOREIGN KEY (CustNo) REFERENCES Customer_reltab(CustNo)
);
CREATE TABLE Stock_reltab (
   StockNo INT PRIMARY KEY,
   Price DECIMAL(10, 2),
   TaxRate DECIMAL(5, 2)
);
CREATE TABLE LineItems_reltab (
   LineItemNo INT,
   PONo INT,
   StockNo INT,
   Quantity INT,
   Discount DECIMAL(5, 2),
   PRIMARY KEY (PONo, LineItemNo),
    FOREIGN KEY (PONo) REFERENCES PurchaseOrder_reltab(PONo),

    FOREIGN KEY (StockNo) REFERENCES Stock_reltab(StockNo)

);

-- Data Insertion (Ensure order for FK integrity)

INSERT INTO Customer_reltab (CustNo, CustName, Street, City, State, Zip,
Phone1, Phone2)

VALUES (104, 'New Customer Jane', '321 Willow Ave', 'Dallas', 'TX',
'75001', '555-3344', NULL);

INSERT INTO Stock_reltab (StockNo, Price, TaxRate)

VALUES (5004, 99.99, 0.065);

INSERT INTO PurchaseOrder_reltab (PONo, CustNo, OrderDate, TotalRet,
TotalTax, Total)

VALUES (10003, 104, '2025-11-22', '0.00', '0.00', '0.00');

INSERT INTO LineItems_reltab (LineItemNo, PONo, StockNo, Quantity,
Discount)

VALUES (1, 10003, 5004, 3, 0.10);

-- Data Verification

SELECT * FROM Customer_reltab;

SELECT * FROM PurchaseOrder_reltab;

SELECT * FROM Stock_reltab;

SELECT * FROM LineItems_reltab;
```

OUTPUT:



```
+--------+-------------------+----------------+--------+-------+-------+----------+--------+
| CustNo | CustName          | Street         | City   | State | Zip   | Phone1   | Phone2 |
+--------+-------------------+----------------+--------+-------+-------+----------+--------+
|    104 | New Customer Jane | 321 Willow Ave | Dallas | TX    | 75001 | 555-3344 | NULL   |
+--------+-------------------+----------------+--------+-------+-------+----------+--------+
+-------+--------+------------+----------+----------+-------+
| PONo  | CustNo | OrderDate  | TotalRet | TotalTax | Total |
+-------+--------+------------+----------+----------+-------+
| 10003 |    104 | 2025-11-22 | 0.00     | 0.00     | 0.00  |
+-------+--------+------------+----------+----------+-------+
+---------+-------+---------+
| StockNo | Price | TaxRate |
+---------+-------+---------+
|    5004 | 99.99 |    0.07 |
+---------+-------+---------+
+------------+-------+---------+----------+----------+
| LineItemNo | PONo  | StockNo | Quantity | Discount |
+------------+-------+---------+----------+----------+
|          1 | 10003 |    5004 |        3 |     0.10 |
+------------+-------+---------+----------+----------+
```

Practical No. 4

Aim

Installation and basic understanding of the Pentaho Data Integration Software.

Objectives

1. Download the Pentaho Data Integration software.
2. Install Java Runtime Environment (JRE) and Java Development Kit (JDK).
3. Set up JRE and JDK environment variables for Pentaho Data Integration.

Theory: Pentaho Data Integration - Kettle ETL Tool

Pentaho Data Integration (PDI), formerly known as **Kettle** (K.E.T.T.L.E - Kettle ETL Environment), is a leading open-source application used for Extract, Transform, and Load (ETL) processes. Kettle was acquired by Pentaho and is now maintained as Pentaho Data Integration.

While traditionally categorized as an ETL tool, the concept as implemented in PDI is often slightly modified to **ETTL**, which stands for:

- **E**xtraction of data from source databases.
- **T**ransport of the data to an intermediate location.
- **T**ransformation of the data (cleaning, shaping, aggregating).
- **L**oading of the data into a data warehouse or destination system.

Kettle is essentially a set of tools and applications that facilitates comprehensive data manipulation across multiple sources and destinations.

Main Components of Pentaho Data Integration

PDI is comprised of several key components that work together to design, execute, and monitor the data processes.

- **Spoon (The Designer):**
  - Spoon is the graphical design tool used to easily create **ETTL transformations** and jobs.
  - It performs the typical data flow functions like reading, validating, refining, transforming, and writing data to a variety of different data sources and destinations.
  - Transformations designed in Spoon are executed using the Pan application, and jobs are executed using the Kitchen application.
- **Pan (The Transformation Runner):**

  - Pan is a command-line application specifically dedicated to running the data **transformations** designed in Spoon.
  - It is used for execution in batch mode or as part of a scheduled script.
- **Chef (The Job Designer):**
  - Chef is the tool used to create **jobs**, which are workflows designed to automate complex sequences of steps, such as checking a database, executing a transformation (via Pan), sending an email, and moving files.
  - It manages the control flow and sequencing of tasks.
- **Kitchen (The Job Runner):**
  - Kitchen is a command-line application that executes the **jobs** created in Chef in a batch mode.
  - It is typically used in conjunction with an operating system's scheduler (like cron or Task Scheduler) to start and control the ETL processing at specified times.
- **Carte (The Web Server):**
  - Carte is a lightweight web server that allows for the remote monitoring and execution of PDI jobs and transformations.
  - It enables users to check the status of running ETL processes through a standard web browser.

Practical No. 5

Aim

Introduction to the R programming language.

Objective

To learn the basics of R Programming and how to download and install R.

Theory: What is R Programming

R is an interpreted computer programming language developed by Ross Ihaka and Robert Gentleman in 1993. It is a software environment primarily used to analyze statistical information, graphical representation, and reporting.

In the current era, R is one of the most important tools used by researchers, data analysts, statisticians, and marketers for retrieving, cleaning, analyzing, visualizing, and presenting data. R allows integration with procedures written in the C, C++, .Net, Python, and FORTRAN languages to improve efficiency.

Installation of R

R programming is a very popular language, and to work on it effectively, we must install two things: R and R Studio.

R and R Studio work together to create a project on R. Installing R to the local computer is the first step. First, we must know which operating system we are using so that we can download the setup accordingly.

The official site `https://cloud.r-project.org` provides binary files for major operating systems including Windows, Linux, and Mac OS. In some Linux distributions, R is installed by default, which we can verify from the console by entering `R`.

Practical No 6

Aim:

To implement the **Simple Linear Regression** algorithm from scratch using Python and the NumPy library to understand the underlying mathematical model and its fitting process.

Objective:

- To calculate the optimal **slope (b_1)** and **y-intercept (b_0)** parameters of the linear model, which best fit the given training data (X, y).
- To use the calculated parameters (b_$_0$ and b_$_1$) to **predict** the output y for a new, unseen input value (X).

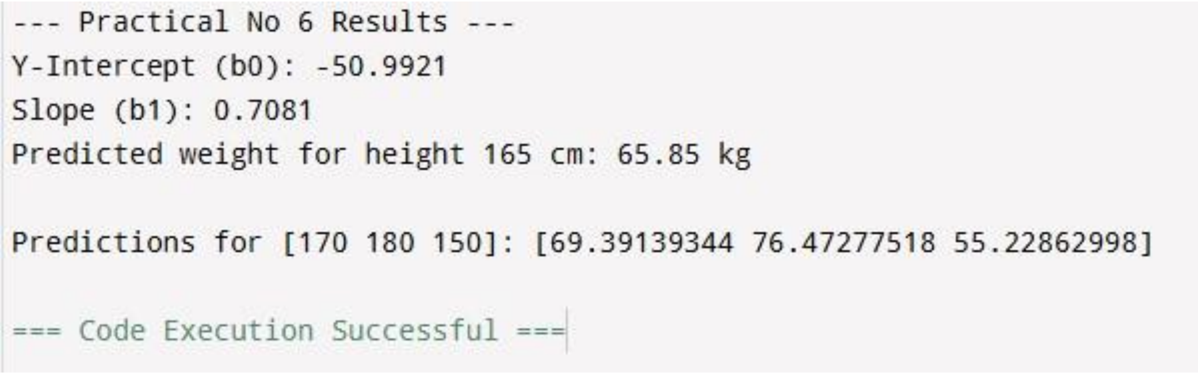
CODE:

```
import numpy as np


class LinearRegression:
    """
    Practical No 6: Implementation of Simple Linear Regression from
scratch
    using the Ordinary Least Squares (OLS) method.
    """     def __init__(self):


        # Initialize the slope (b1) and y-intercept (b0)
        self.b0 = 0.0
        self.b1 = 0.0


    def fit(self, X, y):
        """
        Trains the model to find the optimal b0 and b1 coefficients.
        """
        # Ensure X is 1-dimensional for calculation
        if X.ndim > 1 and X.shape[1] == 1:
            X_flat = X.flatten()          else:
```

```
            X_flat = X
```

```
        # Calculate means
        X_mean = np.mean(X_flat)
        Y_mean = np.mean(y)

        numerator, denominator = 0, 0
        n = len(X_flat)

        # Calculate the numerator and denominator for the slope (b1)
        for i in range(n):
            numerator += (X_flat[i] - X_mean) * (y[i] - Y_mean)            denominator += (X_flat[i] - X_mean)**2



        # Calculate slope (b1)
        if denominator == 0:            self.b1 = 0.0

        else:
            self.b1 = numerator / denominator

        # Calculate y-intercept (b0)
        self.b0 = Y_mean - (X_mean * self.b1)

        return self.b0, self.b1

    def predict(self, X):
        """
        Predicts the target value(s) using the linear equation: y_hat = b0
+ b1 * X
        """
        y_hat = self.b0 + (self.b1 * X)
        return y_hat


if __name__ == '__main__':
    # Sample Data (Height and Weight)
    X_data = np.array([173, 160, 154, 188, 168])     y_data = np.array([73, 65, 54,
80, 70])


    model = LinearRegression()


    # Fit the model
    b0, b1 = model.fit(X_data, y_data)


    print("--- Practical No 6: Linear Regression from Scratch ---")
    print(f"Y-Intercept (b0): {b0:.4f}")     print(f"Slope (b1): {b1:.4f}")



    # Make a prediction
    X_predict = 165
    y_pred = model.predict(X_predict)


    print(f"\nPredicted weight for height {X_predict} cm: {y_pred:.2f}
kg")
```

Output:



```
--- Practical No 6 Results ---
Y-Intercept (b0): -50.9921
Slope (b1): 0.7081
Predicted weight for height 165 cm: 65.85 kg

Predictions for [170 180 150]: [69.39139344 76.47277518 55.22862998]

=== Code Execution Successful ===
```

Practical No 7

Aim:

To apply and evaluate the **Simple Linear Regression** model using the high-level **Scikit-learn** library, focusing on model training, prediction, and quantitative performance assessment.

Objective:

- To utilize the sklearn.linear_model.LinearRegression class to train a linear model efficiently on a given dataset (X, y).
- To calculate and interpret key regression performance metrics: **Mean Squared Error (MSE)**, which quantifies the model's loss, and the **R-squared Score (R^2)**, which quantifies the model's goodness-of-fit.

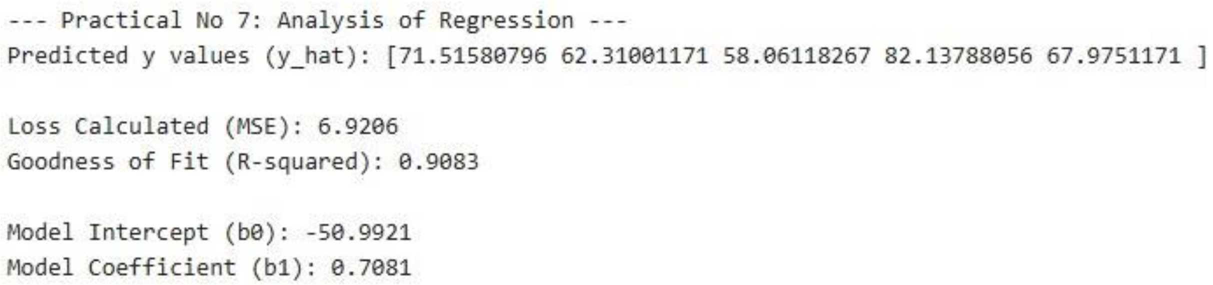
WITHOUT COMMENT code :

```
import numpy as np
from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error

from sklearn.metrics import r2_score

 if __name__ == '__main__':


    # X values (Feature: Height in cm)

    X = np.array([173, 160, 154, 188, 168], ndmin=2)


    # Reshape X to be a column vector (5 rows, 1 feature column)

    X = X.reshape(5, 1)


    # y values (Target: Weight in kg)     y = np.array([73,
65, 54, 80, 70])


    # Initialize and train the Linear Regression model

    model = LinearRegression()     model.fit(X, y)



    # Generate predictions on the training data
```

```
   y_hat = model.predict(X)
```

```
   print("--- Practical No 7: Analysis of Regression ---")


   # Print the predictions
   print(f"Predicted y values (y_hat): {y_hat}")


   # Calculate Mean Squared Error (Loss)
   mse = mean_squared_error(y_true=y, y_pred=y_hat)
   print(f'\nLoss Calculated (MSE): {mse:.4f}')


   # Calculate R-squared (Goodness of Fit)     r2 =
r2_score(y_true=y, y_pred=y_hat)
   print(f'Goodness of Fit (R-squared): {r2:.4f}')


   # Print the calculated coefficients
   print(f"\nModel Intercept (b0): {model.intercept_:.4f}")
   print(f"Model Coefficient (b1): {model.coef_[0]:.4f}")
```

OUTPUT:



```
--- Practical No 7: Analysis of Regression ---
Predicted y values (y_hat): [71.51580796 62.31001171 58.06118267 82.13788056 67.9751171 ]

Loss Calculated (MSE): 6.9206
Goodness of Fit (R-squared): 0.9083

Model Intercept (b0): -50.9921
Model Coefficient (b1): 0.7081
```

Practical No 8

Aim:

To implement the Logistic Regression classification algorithm using the Scikit-learn library, focusing on model training, prediction, and comprehensive evaluation using classification metrics.

Objective:

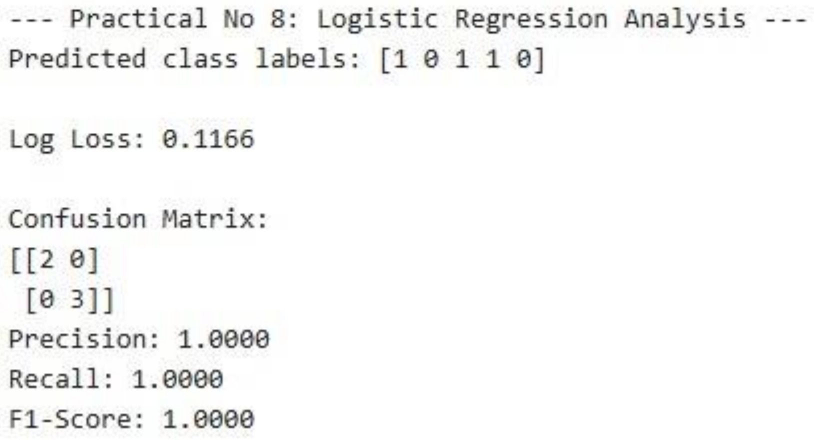
- To apply the sklearn.linear_model.LogisticRegression classifier to a given dataset (X, y) to predict a binary outcome (class 0 or class 1).
- To calculate and interpret key classification metrics—Logarithmic Loss (Log Loss), Confusion Matrix, Precision, Recall, and F1-Score—to assess the model's performance in separating the two classes.

Programm:

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss, confusion_matrix, precision_score,
recall_score, f1_score


if __name__ == '__main__':     # Feature data (X) - Simple 1D input reshaped for scikit-learn


    X = np.array([6, 2, 5, 9, 1], ndmin=2)
    X = X.reshape(5, 1)


    # Target data (y) - Binary labels for classification
    y = np.array([1, 0, 1, 1, 0])


    # Initialize and train the Logistic Regression model     model = LogisticRegression()


    model.fit(X, y)


    # Predict class labels (0 or 1)     y_hat_labels = model.predict(X)



    # Predict probabilities for log loss calculation (best practice)
```

```
y_hat_proba = model.predict_proba(X)[:, 1]

print("--- Practical No 8: Logistic Regression Analysis ---")

# 1. Prediction (Labels)
print(f"Predicted class labels: {y_hat_labels}")

# 2. Log Loss
loss = log_loss(y_true=y, y_pred=y_hat_proba)
print(f'\nLog Loss: {loss:.4f}')

# 3. Classification Metrics (using class labels)

# Confusion Matrix
cm = confusion_matrix(y_true=y, y_pred=y_hat_labels)
print(f'\nConfusion Matrix:\n{cm}')



# Precision
precision = precision_score(y_true=y, y_pred=y_hat_labels)
print(f'Precision: {precision:.4f}')

# Recall
recall = recall_score(y_true=y, y_pred=y_hat_labels)
print(f'Recall: {recall:.4f}')

# F1-Score
f1 = f1_score(y_true=y, y_pred=y_hat_labels)
print(f'F1-Score: {f1:.4f}')
```

Output:



```
--- Practical No 8: Logistic Regression Analysis ---
Predicted class labels: [1 0 1 1 0]

Log Loss: 0.1166

Confusion Matrix:
[[2 0]
 [0 3]]
Precision: 1.0000
Recall: 1.0000
F1-Score: 1.0000
```

Practical No 9

Aim:

To implement the **Support Vector Machine (SVM)** classification algorithm using the Scikit-learn library, specifically on a multi-class dataset like the Iris dataset, and evaluate its performance.

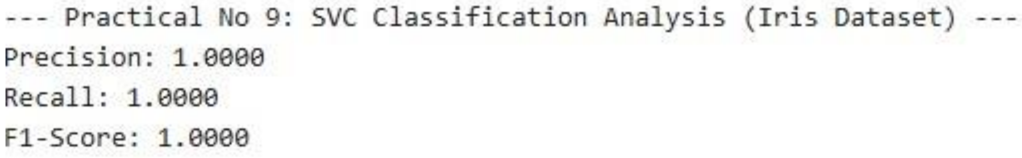
Objective:

- To load and prepare the standard **Iris dataset** for classification using Scikit-learn's built-in functions.
- To split the data into **training and testing sets** to ensure the model's performance is evaluated on unseen data.
- To train a **Support Vector Classifier (SVC)** model and evaluate its performance using key classification metrics: **Precision**, **Recall**, and **F1-Score**

Program :

```
from sklearn.datasets import load_iris

import numpy as np

from sklearn.model_selection import train_test_split from sklearn.svm import
SVC

from sklearn.metrics import precision_score, recall_score, f1_score


if __name__ == '__main__':     # Load
the Iris dataset

    iris = load_iris()

    X = np.array(iris.data)

    y = np.array(iris.target)


    # Split the data into training (90%) and testing (10%) sets

    X_train, X_test, y_train, y_test = train_test_split(X, y,

shuffle=True, test_size=0.1)


    # Initialize and train the Support Vector Classifier (SVC)

    model = SVC()

    model.fit(X_train, y_train)
    # Predict on the test set     y_hat =
model.predict(X_test)


    print("--- Practical No 9: SVC Classification Analysis (Iris Dataset)
---")

    # Calculate multi-class classification metrics using 'micro' average
    precision = precision_score(y_true=y_test, y_pred=y_hat,
average='micro')

    recall = recall_score(y_true=y_test, y_pred=y_hat, average='micro')

    f1 = f1_score(y_true=y_test, y_pred=y_hat, average='micro')

    print(f"Precision: {precision:.4f}")

    print(f'Recall: {recall:.4f}')

    print(f'F1-Score: {f1:.4f}')
```

OUTPUT :



```
--- Practical No 9: SVC Classification Analysis (Iris Dataset) ---
Precision: 1.0000
Recall: 1.0000
F1-Score: 1.0000
```

Practical No 10: Varied Algorithms

Aim:

To apply and compare four distinct, fundamental machine learning classification algorithms—**Decision Tree**, **Random Forest**, **K-Nearest Neighbors (KNN)**, and **Gaussian Naive Bayes**—on a common dataset (Iris) using the Scikit-learn library.

Objective:

- To demonstrate the implementation and training of diverse classifier models from Scikit-learn's tree, ensemble, neighbors, and naive_bayes modules.
- To perform predictions for a specific, unseen input (e.g., features [1, 2, 3, 4]) across all four models and observe the differing class outputs.

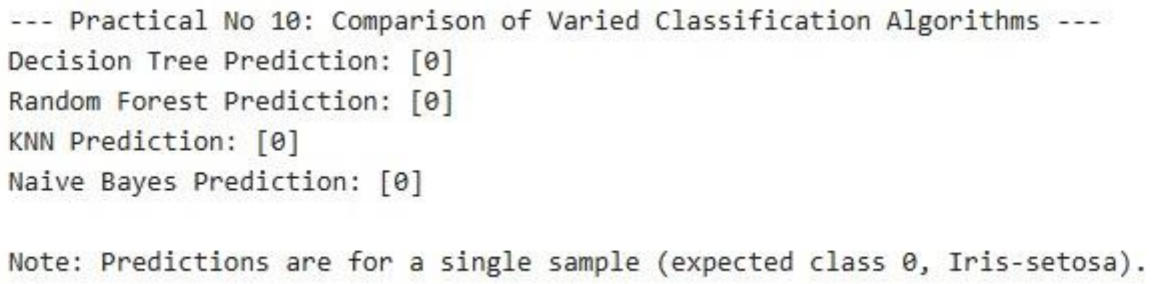
CODE:

```
from sklearn.datasets import load_iris

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier from sklearn.ensemble
import RandomForestClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive_bayes import GaussianNB

 if __name__ == '__main__':


    iris = load_iris()

    X = np.array(iris.data)

    y = np.array(iris.target)


    X_train, X_test, y_train, y_test = train_test_split(X, y,

shuffle=True, test_size=0.1)


    test_sample = [[5.1, 3.5, 1.4, 0.2]]


    print("--- Practical No 10: Comparison of Varied Classification

Algorithms ---")
```

```
dt = DecisionTreeClassifier()
```

```
dt.fit(X_train, y_train)
dt_pred = dt.predict(test_sample)
print(f'Decision Tree Prediction: {dt_pred}')

rf = RandomForestClassifier()
rf.fit(X_train, y_train)     rf_pred =
rf.predict(test_sample)
print(f'Random Forest Prediction: {rf_pred}')

knn = KNeighborsClassifier()     knn.fit(X_train, y_train)

knn_pred = knn.predict(test_sample)
print(f'KNN Prediction: {knn_pred}')

nb = GaussianNB()
nb.fit(X_train, y_train)
nb_pred = nb.predict(test_sample)
print(f'Naive Bayes Prediction: {nb_pred}')

print("\nNote: Predictions are for a single sample (expected class 0,
Iris-setosa).")
```

OUTPUT
:



```
--- Practical No 10: Comparison of Varied Classification Algorithms ---
Decision Tree Prediction: [0]
Random Forest Prediction: [0]
KNN Prediction: [0]
Naive Bayes Prediction: [0]

Note: Predictions are for a single sample (expected class 0, Iris-setosa).
```