DSBDA
Lab Assignment No: 1

AIM:
To perform basic Data Wrangling (Part I) using Python.

OBJECTIVES:

- To learn how data science concepts are used to solve real-world problems.
- To understand what data wrangling is.
- To apply basic data wrangling techniques in Python.

PROBLEM STATEMENT:

Data Wrangling – Part I

Using Python and any open-source dataset (for example, data.csv), do the following:

1. Import all the necessary Python libraries.
2. Find an open-source dataset from the internet (for example, from https://www.kaggle.com).
    - Clearly describe the dataset.
    - Mention the source and give the URL of the website.
3. Load the dataset into a pandas DataFrame.
4. **Data Preprocessing:**
    - Check for missing values using pandas.isnull().
    - Use describe() to view basic statistics about the data.
    - Explain the variables: what each column represents, types of variables, etc.
    - Check the size (dimensions) of the DataFrame.
5. **Data Formatting and Normalization:**
    - Check the data types of all columns (e.g., string, numeric, integer, categorical, boolean).
    - If any column has the wrong data type, convert it to the correct type.
6. Convert categorical (text) variables into numerical variables in Python.

Along with the code and outputs, explain every step you perform. Also clearly explain how you imported, read, or downloaded (scraped) the dataset.

OUTCOME:

CO1: Use data science principles to analyze real-world problems.

SOFTWARE & HARDWARE REQUIREMENTS:

- **Software:** Python 3.8.1 / Jupyter Notebook
- **Hardware:** Windows 7 or any open-source Linux operating system

---

THEORY:

Python is one of the most popular programming languages today because it is simple, powerful, and flexible. It is a high-level, open-source language that can be used for many general-purpose programming tasks.

Python has many open-source libraries. One of the most important for data work is **pandas**.
Pandas is:

- Fast
- Powerful
- Flexible

It is mainly used for:

- Data analysis
- Working with tabular data (dataframes/datasets)
- Reading and writing data in different file formats such as:
    - CSV (comma-separated values)
    - TXT
    - XLS / XLSX (Microsoft Excel)

In this experiment, you will learn some basic features of pandas in Python and how to use them in practice.

Prerequisite: You should know basic Python programming.

INSTALLATION:

If you are new to pandas, install it first.

1. Open **Command Prompt** (run as administrator).
2. Make sure your computer is connected to the internet.
3. Type:

```
pip install pandas
```

1. Press Enter and wait for the installation to complete.

DATASET: “Iris.csv”

The **Iris dataset** is considered the “Hello World” of Data Science. If you are beginning in Data Science and Machine Learning, you will often start with this dataset for simple ML algorithms.

The Iris dataset contains 5 columns:

- Sepal Length
- Sepal Width
- Petal Length
- Petal Width
- Species (type of flower)

Iris is a type of flowering plant. Researchers measured different parts of many iris flowers and stored this information digitally.

BASIC OPERATIONS WITH THE DATASET

Assume we loaded the dataset into a DataFrame called data.

1. Display the top rows of the dataset

The head() function shows the first few rows.
By default, head() shows 5 rows, but you can pass a number to see more:

```
data.head(10)      # shows the first 10 rows
```

2. Display random rows from the dataset

The sample() function returns random rows from the DataFrame:

```
data.sample(10)    # shows 10 random rows
```

3. Display the column names

The columns attribute gives a list of all column names:

```
data.columns       # prints all column names
```

4. Display the shape (size) of the dataset

The shape attribute shows:

- Number of rows (entries)
- Number of columns (features)

```
data.shape         # e.g. (150, 5)
```

5. Slicing rows

Slicing means selecting a portion of the rows, for example from the 10th to the 20th row:

```
print(data[10:21])       # prints rows from index 10 to 20
# You can also store this slice in a new variable:
sliced_data = data[10:21]
print(sliced_data)
```

6. Display only specific columns

Sometimes you only want a few columns from the dataset. You can select them like this:

```
specific_data = data[["Id", "Species"]]
# general form:
# data[["column1", "column2", "column3"]]
print(specific_data.head(10))   # show first 10 rows of these columns
```

CONCLUSION:

Thus, we have carried out and demonstrated basic data wrangling operations using Python and pandas.

Lab Assignment No: 2

AIM:
To do Data Wrangling – Part II using Python on any open-source dataset.

OBJECTIVES:

- To learn and use basic data science concepts on real-world problems.
- To clearly understand and implement key techniques used in data science and big data analytics.

PROBLEM STATEMENT:
Data Wrangling – II: Do the following tasks in Python on any open-source dataset (for example, data.csv):

1. Check all variables for missing values and inconsistent data.
   If you find missing or inconsistent values, handle them using any suitable method (e.g., filling, removing, etc.).

2. Check all numerical variables for outliers.
   If you find outliers, handle them using a suitable technique (e.g., removing, capping, transformation, etc.).

3. Apply a data transformation to at least one variable.
   The goal of the transformation should be one of these:

   - Change the scale so that the variable is easier to understand.
   - Convert a non-linear relationship into a more linear one.
   - Reduce skewness and make the distribution closer to a normal distribution.
     Clearly write down the reason and explain your approach.

OUTCOME:
CO1: Use data science principles to analyze real-world problems.

SOFTWARE & HARDWARE REQUIREMENTS:

- Jupyter / IPython
- Python 3.8.1

THEORY

Identification and Handling of Null (Missing) Values

Missing data appears when some information is not recorded for one or more entries. This is very common in real-life datasets. In pandas, missing data is usually shown as NA (Not Available) values.

Examples:

- In a survey, some users may not share their income.
- Some users may leave their address blank.
  These situations create missing values in the dataset.

In pandas, missing values are mainly represented by:

1. None
   - A special Python object used to show missing data in normal Python code.
2. NaN (Not a Number)
   - A special floating-point value used in numerical computations (IEEE standard).

Pandas treats None and NaN in almost the same way for missing or null values.
To work with them, pandas provides several useful functions to detect, delete, or replace null values:

- isnull()
- notnull()
- dropna()
- fillna()
- replace()

Checking for Missing Values using isnull() and notnull()

1. Using isnull()

isnull() checks where values are missing. It returns a DataFrame of True/False values:

- True → value is NaN / missing
- False → value is present

Algorithm using isnull():

Step 1: Import pandas and numpy.

```
import pandas as pd
import numpy as np
```

Step 2: Load the dataset into a DataFrame object df.

```
df = pd.read_csv("/content/StudentsPerformanceTest1.csv")
```

Step 3: Display the DataFrame.

```
df
```

Step 4: Use isnull() to check for missing values in the whole dataset.

```
df.isnull()
```

Step 5: Create a Boolean series for NaN values in a specific column
(for example, the column "math score") and then display only those rows.

```
series = pd.isnull(df["math score"])
df[series]
```

2. Using notnull()

notnull() does the opposite of isnull(). It returns True where data is present and False where data is missing.

Algorithm using notnull():

Step 1: Import pandas and numpy.

```
import pandas as pd
import numpy as np
```

Step 2: Load the dataset into DataFrame df.

```
df = pd.read_csv("/content/StudentsPerformanceTest1.csv")
```

Step 3: Display the DataFrame.

```
df
```

Encoding Categorical Data (Label Encoding / One-Hot Encoding)

Categorical columns (like gender, city, etc.) often need to be converted into numeric format.

Example using Label Encoding:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

```
df['gender'] = le.fit_transform(df['gender'])
newdf = df
newdf
```

Filling Missing Values: dropna(), fillna(), replace()

To handle null values in a DataFrame, we can use:

- fillna() → to fill missing values with a specified value or method
- replace() → to replace specific values (e.g., "Na", "na") with NaN
- dropna() → to remove rows/columns with missing values

Example: Treating certain text values as missing

```
missing_values = ["Na", "na"]
df = pd.read_csv("StudentsPerformanceTest1.csv",
na_values=missing_values)
df
```

Filling all missing values with a single value (e.g., 0):

Algorithm:

Step 1: Import libraries.

```
import pandas as pd
import numpy as np
```

Step 2: Load the dataset.

```
df = pd.read_csv("/content/StudentsPerformanceTest1.csv")
```

Step 3: Display the DataFrame.

```
df
```

Step 4: Fill missing values using fillna().

```
ndf = df
ndf = ndf.fillna(0)
ndf
```

Deleting Missing Values using dropna()

dropna() removes rows or columns that contain missing values.

You can:

1. Drop rows with at least one null value.
2. Drop rows only if all values are null.
3. Drop columns with at least one null value.
4. Use the same logic directly when reading a CSV.

Algorithm:

Step 1: Import pandas and numpy.

```
import pandas as pd
import numpy as np
```

Step 2: Load the dataset.

```
df = pd.read_csv("/content/StudentsPerformanceTest1.csv")
```

Step 3: Display the DataFrame.

```
df
```

Step 4: Drop rows with at least one missing value.

```
ndf = df.dropna()
ndf
```

Detecting Outliers using a Scatter Plot

A scatter plot is useful when:

- You have two numerical variables.
- You want to see the relationship between them.
- You want to visually detect possible outliers.

In this example, we use:

- placement score
- placement offer count

Algorithm:

Step 1: Import the required libraries.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Step 2: Load the dataset into df.

```
df = pd.read_csv("/content/demo.csv")
```

Step 3: Display the DataFrame.

```
df
```

Step 4: Create a scatter plot for placement score vs placement offer count.

```
fig, ax = plt.subplots(figsize=(18, 10))
ax.scatter(df['placement score'], df['placement offer count'])
plt.show()
```

(Optional) Add labels to the axes:

```
ax.set_xlabel('(Proportion non-retail business acres)/(town)')
ax.set_ylabel('(Full-value property-tax rate)/($10,000)')
```

CONCLUSION

In this experiment, we used Python libraries (mainly pandas and matplotlib) to:

- Identify and handle missing values using functions like isnull(), notnull(), fillna(), replace(), and dropna().
- Detect outliers visually using scatter plots.
- Apply data transformation techniques to create new variables and reduce skewness in the data.

These steps are essential parts of data wrangling, helping to clean and prepare data for further analysis and modeling.

Lab Assignment No: 3

AIM:
To use Basic Statistics (measures of central tendency and variability) on any open-source dataset.

OBJECTIVES

- To learn how to apply data science concepts to real-world problems.
- To gain a clear understanding of, and hands-on practice with, important techniques used in data science and big data analytics.

PROBLEM STATEMENT

Use an open-source dataset (for example, data.csv) and perform the following tasks related to Basic Statistics – Measures of Central Tendency and Variance:

1. **Summary statistics grouped by a categorical variable**
    - Choose:
        - At least one **numeric variable** (e.g., age, income, marks, etc.), and
        - One **categorical (qualitative) variable** (e.g., gender, age group, city, department, etc.).
    - Calculate and display the following summary statistics for the numeric variable, grouped by the categories:
        - Mean
        - Median
        - Minimum
        - Maximum
        - Standard deviation
    - Example:
      If the categorical variable is **age group** and the numeric variable is **income**, then show the summary statistics of income for each age group (e.g., 18–25, 26–35, etc.).
    - Also, create a Python list that stores a numeric value for each category (for example, average income for each age group).
2. **Statistics on Iris dataset**
    - Use the iris.csv dataset.
    - Write a Python program that prints basic statistical details (such as percentiles, mean, standard deviation, etc.) for each species:
        - Iris-setosa

        - Iris-versicolor
        - Iris-virginica
    - Show the Python code and its output.
    - Clearly explain every step you perform in the code (what each command does and why you are using it).

OUTCOMES

- **CO1:** Use data science principles to analyze real-world problems.
- **CO2:** Represent and summarize data using statistical methods.

SOFTWARE & HARDWARE REQUIREMENTS

- Jupyter / IPython Notebook
- Python 3.8.1

THEORY

1. Measures of Center (Central Tendency)

These measures tell us what a “typical” or “center” value of a dataset looks like.

a) Mean (Average)

- The **mean** is what we commonly call the **average**.
- To find it:
    - Add all the values.
    - Divide by the number of values.
- **Population mean** (for the whole population) is written as **μ (mu)**.
- **Sample mean** (for a sample from the population) is written as **x̄ (x-bar)**.

Formulas:

- Population mean:
  $\mu = (\Sigma x_i) / N$

- Sample mean:
  $\bar{x} = (\Sigma x_i) / n$

where:

- x_i = each value in the data
- N = number of values in the population
- n = number of values in the sample

The mean is a good estimate of the center, but it is **affected by extreme values** (outliers). If the data is very skewed (has a long tail), the mean might not represent the center well.

---

b) Median

- The **median** is the **middle value** when all the data values are sorted from smallest to largest.

- It splits the data into two equal parts:

  - 50% of the values are below the median.
  - 50% are above the median.
- The median is **not strongly affected by outliers**, so it can be a better measure of center when the data is very skewed.

How to find the median:

- If the number of values (n) is **odd**:
  - Sort the data.
  - The median is the value exactly in the middle.
- If n is **even**:
  - Sort the data.
  - Take the average of the two middle values.

---

c) Mode

- The **mode** is the **value that appears most often** in the dataset.
- It is especially useful for **categorical (qualitative) data** because you cannot calculate a mean or median for categories like “red,” “blue,” “male,” “female,” etc.
- For quantitative data:
  - Sometimes all values are different, so there may be **no meaningful mode**.
  - There can also be more than one mode (bimodal, multimodal).

Relationship Between Mean, Median, and Distribution Shape

Understanding how the mean and median compare can help us understand the shape of the distribution:

- **Right-skewed (positively skewed)**:
  - There are a few **very large values** pulling the tail to the right.
  - The **mean is usually greater than the median** (mean > median), because it is pulled up by the large values.
- **Symmetric distribution**:
  - The data is balanced on both sides.
  - The **mean, median, and mode are close to each other**.
- **Left-skewed (negatively skewed)**:
  - There are a few **very small values** pulling the tail to the left.
  - The **mean is usually less than the median** (mean < median), because it is pulled down by the small values.

---

2. Measures of Dispersion (Spread)

Measures of center tell us about the middle.
Measures of dispersion tell us how spread out the values are.

- If values are **close to each other**, the variation is **small**.
- If values are **far apart**, the variation is **large**.

a) Range

- The **range** is the simplest measure of spread.
- Formula:
  Range = Maximum value − Minimum value
- It only uses the largest and smallest values, and ignores everything in between.

---

b) Variance

- **Variance** measures how far each value is from the mean, on average.

- Steps:

1. Find the mean.
2. Subtract the mean from each value (to get the deviation).
3. Square each deviation (to make them all positive).
4. Add up all squared deviations.
5. Divide by:
    - **N** for population variance, or
    - **(n − 1)** for sample variance.

- Population variance is written as **σ^2 (sigma squared)**.

- Sample variance is written as **s^2**.

Formulas:

- Population variance:
  $\sigma^2 = \Sigma (x_i - \mu)^2 / N$

- Sample variance:
  $s^2 = \Sigma (x_i - \bar{x})^2 / (n - 1)$

The sample variance (s^2) is an **unbiased estimator** of the population variance (σ^2).

---

c) Standard Deviation

- The **standard deviation** is the **square root of the variance**.

- It is very commonly used because it is in the **same units** as the original data (while variance is in squared units).

- Population standard deviation: **σ**

- Sample standard deviation: **s**

Formulas:

- Population standard deviation:
  $\sigma = \sqrt{\sigma^2}$

- Sample standard deviation:
  $s = \sqrt{s^2}$

Standard deviation tells us how much the values typically differ from the mean. A **larger** standard deviation means the data is **more spread out**.

CONCLUSION

In this assignment, we use Python and its libraries to explore basic statistical concepts such as measures of central tendency (mean, median, mode) and measures of dispersion (range, variance, standard deviation). By applying these to real datasets (like an open-source CSV file and the Iris dataset), we learn how to summarize and understand data using fundamental tools of data science.

Lab Assignment No: 4

Problem Description

Build a **Linear Regression model** using **Python or R** to predict house prices with the **Boston Housing Dataset**
(https://www.kaggle.com/c/boston-housing).

- The dataset has **506 rows (houses)** and **14 features** (e.g., crime rate, rooms, etc.).
- Goal: **Predict the house price** (target variable) using the given features (input variables).

Objectives

- Understand how **data science principles** are used to solve **real-world problems**.
- Learn how to perform **data analysis using Linear Regression** in Python on any open-source dataset.

Outcomes

- **CO1**: Use data science concepts to analyse real-life data problems.
- **CO2**: Use statistics to represent and analyze data.

Software / Hardware Requirements

- **Software**: Jupyter Notebook / IPython
- **Python Version**: 3.8.1 (or similar)

Theory

1. Linear Regression (Univariate and Multivariate)

Linear Regression is a **supervised learning** algorithm used to predict a numeric (continuous) value based on one or more input features.

- It is used to find the **relationship** between:
    - **Input (independent) variables** – usually denoted by X
    - **Output (dependent/target) variable** – usually denoted by Y
- We assume there is a **linear relationship** between X and Y.

A simple linear regression model is written as:

$$Y = mX + b + e$$

Where:

- Y = predicted value (target)
- X = input (feature)
- m = slope of the line
- b = intercept (value of Y when $X = 0$)
- e = error term (difference between actual and predicted value)

The goal is to find the **best line** through the data that minimizes the prediction error.

Example of Linear Relationship

- Suppose we study the relationship between **height (cm)** and **weight (kg)**.
- Let:
    - x = height (independent variable / predictor)
    - y = weight (dependent variable / response)
- If the points roughly form a straight line, we can use **simple linear regression** to model this.

Simple vs Multivariate Regression

- **Simple Linear Regression**:
    - One input variable (one predictor).
    - Example: Predict salary based on years of experience.
- **Multivariate (Multiple) Linear Regression**:
    - Two or more input variables.
    - Example: Predict house price based on rooms, area, location, etc.

Sometimes we transform features into **polynomial features**, e.g.:

Original simple model:

$$Y = a + bX$$

Polynomial model:

$$Y = a + bX + cX^2$$

If the polynomial degree is too high, the model may **overfit** (it starts fitting noise instead of the real pattern).

---

2. Least Squares Method for Linear Regression

Linear Regression tries to find the **best fit line** for the data.
The standard method used is called the **Least Squares Method**.

For simple linear regression:

$$y = \beta_0 + \beta_1 x + \varepsilon$$

Where:

- y = dependent variable
- x = independent variable
- β_0 = intercept
- β_1 = slope
- ε = error term (noise)

We want to find β_0 and β_1 such that the **sum of squared errors** is minimum:

$$\text{Minimize } \Sigma (y_i - \hat{y}_i)^2$$

where:

- y_i = actual value
- $\hat{y}_i$ = predicted value from the line

Using formulas, we can estimate:

$$\beta_1 = \Sigma(x_i - \bar{x})(y_i - \bar{y}) / \Sigma(x_i - \bar{x})^2$$

$$\beta_0 = \bar{y} - \beta_1\bar{x}$$

where:

- $\bar{x}$ = mean of x values
- $\bar{y}$ = mean of y values

Once the model is found, we can **predict y** for any x value **within the range of observed data**. Predicting far outside this range is called **extrapolation**, and it is often unreliable.

---

3. Measuring the Performance of Linear Regression

To check how good our model is, we use some **evaluation metrics**.

a) Mean Squared Error (MSE)

MSE measures the average of the squared differences between actual and predicted values.

$$MSE = (1/n)\ \Sigma\ (y_i - \hat{y}_i)^2$$

- Lower MSE → better model.
- MSE = 0 means perfect predictions (no error).

b) Root Mean Squared Error (RMSE)

RMSE is just the **square root of MSE**:

$$RMSE = \sqrt{[(1/n)\ \Sigma\ (y_i - \hat{y}_i)^2]}$$

- RMSE has the **same units** as the target variable.
- Lower RMSE → better performance.

c) R-Squared (R^2)

R^2 tells us how much of the variation in the target variable is explained by the model.

$$R^2 = SSR / SST$$

Where:

- **SST** = Total Sum of Squares (total variation in y)
- **SSR** = Sum of Squares due to Regression (explained by model)
- **SSE** = Sum of Squared Errors (unexplained part)

R^2 ranges from **0 to 1**:

- R^2 close to 1 → model explains most of the variance → **good fit**
- R^2 close to 0 → model explains little of the variance → **poor fit**

MSE and R^2 both are useful; R^2 is easier to interpret because it is always between 0 and 1.

---

4. Example of Simple Linear Regression (Student Scores)

We have data of 5 students:

Student	Score in X (X_i)	Score in XII (Y_i)
1	95	85
2	85	95
3	80	70
4	70	65
5	60	70

From this data (using formulas shown earlier), we get:

- Mean of X, $\bar{x} = 78$
- Mean of Y, $\bar{y} = 77$

Also calculated:

- $\Sigma(x_i - \bar{x})^2 = 730$
- $\Sigma(x_i - \bar{x})(y_i - \bar{y}) = 470$

So:

$$\beta_1 = 470 / 730 \approx 0.644$$
$$\beta_0 = \bar{y} - \beta_1\bar{x} = 77 - 0.644 \times 78 \approx 26.768$$

So the regression equation is:

$$\hat{y} = 26.768 + 0.644x$$

Interpretation

1. **Slope (0.644)**:
   For every **1 mark increase in X standard**, the **XII standard score** is expected to **increase by about 0.644** marks (on average).

2. **Intercept (26.768)**:
   If a student's score in X standard were 0, the model predicts a XII score of about **26.768** marks.
   (In practice, this is just a mathematical result; it may not be meaningful if X cannot actually be 0.)

3. **Prediction Example**
   If a student's score in X standard is 65, the predicted XII score is:

   $\hat{y} = 26.768 + 0.644 \times 65 \approx 68.38$

---

5. Training Dataset, Testing Dataset, and Generalization

Machine Learning has two main phases:

1. **Training Phase**
2. **Testing Phase**

(a) Training Phase

- **Input**: Training dataset (features + known labels/targets).
- We pass this data to the algorithm.
- **Output**: A trained **model**.
- **Training Error (E_in)**: Error when the model is tested on the **same data** it was trained on.

(b) Testing Phase

- **Input**: Testing dataset (features, but usually treated as unknown labels).
- Data is **not used during training**.
- The model makes predictions on test data.
- **Testing Error (E_out)**: Error on this new, unseen data.
- Usually **E_out > E_in**.

(c) Generalization

- **Generalization** means how well the model performs on **new, unseen data**.
- We want to **minimize generalization error**.
- That is why we **split** our original dataset into:

    - **Training set** – to build the model.
    - **Test set** – to evaluate how well it will work on new data.

Algorithm – Simple Example (Synthetic Dataset)

We will implement linear regression using **NumPy** on the student score data.

Step 1: Import required libraries

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt
```

Step 2: Create arrays for X and Y

```
x = np.array([95, 85, 80, 70, 60])

y = np.array([85, 95, 70, 65, 70])
```

Step 3: Fit a linear regression model using polyfit

```
model = np.polyfit(x, y, 1)  # degree 1 for linear
```

Step 4: Check the model coefficients

```
model

# Output example:

# array([0.64383562, 26.78082192])
```

This means:

- slope ≈ 0.6438
- intercept ≈ 26.7808

Step 5: Predict Y for a single X value

```
predict = np.poly1d(model)

predict(65)
```

```
# Output:

# 68.63 (approximately)
```

Step 6: Predict Y for all X values

```
y_pred = predict(x)

y_pred

# Output:

# array([81.50684932, 87.94520548, 71.84931507, 68.63013699,
71.84931507])
```

Step 7: Compute R-Squared

NumPy does not provide R^2 directly, so we use scikit-learn:

```
from sklearn.metrics import r2_score

r2_score(y, y_pred)

# Output example:

# 0.4803218090889323
```

Step 8: Plot the regression line and points

```
y_line = model[1] + model[0] * x

plt.plot(x, y_line, c='r')        # regression line

plt.scatter(x, y_pred)           # predicted points

plt.scatter(x, y, c='r')         # actual points

plt.show()
```

Algorithm – Boston Housing Dataset

Step 1: Import libraries

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
```

Step 2: Load the Boston Housing dataset

```
from sklearn.datasets import load_boston

boston = load_boston()
```

Step 3: Create a DataFrame

```
data = pd.DataFrame(boston.data)
```

Step 4: Add feature names as column names

```
data.columns = boston.feature_names

data.head()
```

Step 5: Add the target (price) column

```
data['PRICE'] = boston.target
```

Step 6: Check for missing values

```
data.isnull().sum()
```

Step 7: Split features and target

```
X = data.drop(['PRICE'], axis=1)

y = data['PRICE']
```

Step 8: Split into training and testing sets

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.2, random_state=0

)
```

Step 9: Train Linear Regression model

```
from sklearn.linear_model import LinearRegression

lm = LinearRegression()

model = lm.fit(X_train, y_train)
```

Step 10: Make predictions

```
y_train_pred = lm.predict(X_train)

y_test_pred  = lm.predict(X_test)
```

Step 11: View predictions (optional)

```
df_train = pd.DataFrame({'Actual': y_train, 'Predicted': y_train_pred})

df_test  = pd.DataFrame({'Actual': y_test,  'Predicted': y_test_pred})
```

Step 12: Calculate MSE for train and test sets

```
from sklearn.metrics import mean_squared_error, r2_score

# Train MSE

mse_train = mean_squared_error(y_train, y_train_pred)

print(mse_train)

# Test MSE

mse_test = mean_squared_error(y_test, y_test_pred)

print(mse_test)

# Sample outputs (as in original):

# Train MSE ≈ 33.44897999767638

# Test MSE  ≈ 19.32647020358573
```

Step 13: Plot True vs Predicted values

```
plt.scatter(y_train, y_train_pred, c='blue', marker='o', label='Training
data')

plt.scatter(y_test,  y_test_pred,  c='lightgreen', marker='s',
label='Test data')
```

```
plt.xlabel('True values')

plt.ylabel('Predicted values')

plt.title('True value vs Predicted value')

plt.legend(loc='upper left')

plt.plot()

plt.show()
```

Conclusion

We have:

- Applied **Linear Regression** to the **Boston Housing Dataset**.
- Used various features to **predict house prices**.
- Evaluated the model using metrics like **MSE** and visual comparison of **true vs predicted values**.

Assignment Questions

1. For the student score example below, compute:
    - SST (Total Sum of Squares)
    - SSE (Sum of Squared Errors)
    - SSR (Sum of Squares due to Regression)
    - MSE (Mean Squared Error)
    - RMSE (Root Mean Squared Error)
    - R^2 (R-Squared)

2.

Student	Score in X (X_i)	Score in XII (Y_i)
1	95	85

2	85	95
3	80	70
4	70	60
5	65	70

3. Based on your calculated values, comment on whether the model is a **good fit** or not.
4. Write Python code to calculate **R-Squared** for the Boston Housing dataset (using the linear regression model you created in the practical).

Conclusion

We have explored the **Linear Regression model**, learned how it works mathematically (least squares), how to measure its performance (MSE, RMSE, R^2), and how to apply it in Python to real data (Boston Housing) to **predict house prices**.

Lab Assignment No: 5

Title: Logistic Regression

Goal

To use Python or R to build a classification model using Logistic Regression on the Social_Network_Ads.csv dataset.

Objectives

- To learn how to use data science to solve real-life problems.
- To learn how to analyze data using Logistic Regression on open-source data.

The Problem

You need to analyze the Social_Network_Ads.csv dataset. Your task is to build a Logistic Regression model to classify the data (predict an outcome). Afterward, you must evaluate how good your model is by creating a **Confusion Matrix** and calculating **Accuracy, Error Rate, Precision,** and **Recall**.

Outcomes

By the end of this assignment, you will be able to:

1. Apply data science methods to real problems.
2. Use statistics to represent data.
3. Build and test data analytics algorithms.

Requirements: Jupyter Notebook or IPython (version 3.8.1 or higher).

THEORY: What is this about?

1. What is Logistic Regression?

In Data Science, about 70% of problems are **Classification** problems. This means we are trying to sort things into categories (like "Will buy" vs. "Won't buy," or "Spam" vs. "Not Spam").

Logistic Regression is the most common method for solving these **Binary Classification** problems (where there are only two choices).

- It predicts the **probability** of an event happening.

- It is different from Linear Regression because it is used when the target is categorical (Yes/No), not a specific number.

2. Linear Regression vs. Logistic Regression

- **Linear Regression:** Predicts a continuous number (e.g., predicting the exact price of a house or a stock). It uses a straight line to fit data.
- **Logistic Regression:** Predicts a category (e.g., predicting if a patient has cancer: Yes or No). It results in a probability between 0 and 1.

3. The Sigmoid Function

Logistic Regression uses math called the **Sigmoid Function** (or Logit function).

- It creates an **"S" shaped curve**.
- It takes any number and squashes it into a value between **0 and 1**.
- **The Rule:** If the value is above 0.5, we classify it as **1 (Yes)**. If it is below 0.5, we classify it as **0 (No)**.

4. Types of Logistic Regression

1. **Binary:** Only two outcomes (e.g., Pass/Fail).
2. **Multinomial:** Three or more categories without order (e.g., Cat, Dog, Sheep).
3. **Ordinal:** Three or more categories with an order (e.g., Rating a movie 1 to 5 stars).

5. Evaluating the Model: The Confusion Matrix

To see how well our model worked, we use a **Confusion Matrix**. This is a table that compares the **Predicted** answers with the **Actual** answers.

The 4 Key Terms:

- **True Positive (TP):** We predicted YES, and it was actually YES. (Correct)
- **True Negative (TN):** We predicted NO, and it was actually NO. (Correct)
- **False Positive (FP):** We predicted YES, but it was actually NO. (False Alarm)
- **False Negative (FN):** We predicted NO, but it was actually YES. (Missed it)

Formulas to measure success:

- **Accuracy:** How often is the classifier correct overall?
  - *Formula:* (Correct Guesses) / (Total Guesses) or (TP + TN) / Total
  - *Ideal value:* 1.0 (100%)
- **Error Rate:** How often is the classifier wrong?
  - *Formula:* (Wrong Guesses) / (Total Guesses) or 1 - Accuracy
- **Precision:** When it predicts YES, how often is it right?

    - *Formula:* TP / (TP + FP)
- **Recall (Sensitivity):** Out of all the actual YES cases, how many did it find?
    - *Formula:* TP / (TP + FN)

ALGORITHM: Step-by-Step Guide

Step 1: Setup
Import the necessary Python libraries (Pandas for data, Numpy for math, Matplotlib for graphs).

Step 2: Load Data
Import the Social_Network_Ads dataset into your program.

Step 3: Organize
Put the data into a Data Frame so it is easy to read.

Step 4: Pre-processing (Cleaning the Data)

- Change text data into numbers if needed.
- Check for empty values (Nulls).
- Separate the data into **Features (X)** (the input info) and the **Target (Y)** (what we want to predict).
- **Split the data:** Keep some data to train the machine (Training Set) and set aside the rest to test it later (Testing Set).
- **Scale:** Adjust the numbers so they are all on a similar scale (e.g., don't compare age 30 to salary 50,000 directly; scale them down).

Step 5: Train the Model
Use the LogisticRegression class from Scikit-Learn.

```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()

logreg.fit(x_train, y_train) # The machine learns here
```

Step 6: Predict
Ask the model to predict the answers for the Test Set.

```
y_pred = logreg.predict(x_test)
```

Step 7 & 8: Evaluate
Compare your predictions (y_pred) with the actual answers (y_test) to calculate accuracy, precision, and recall.

```
from sklearn.metrics import confusion_matrix, accuracy_score

cm = confusion_matrix(y_test, y_pred)

print(accuracy_score(y_test, y_pred))
```

LAB ASSIGNMENT QUESTIONS

1. Calculate Metrics
Imagine you have a binary classification task. Based on your results, calculate the values for:

- True Positives (TP), True Negatives (TN), False Positives (FP), False Negatives (FN).
- Calculate the Accuracy, Error Rate, Precision, and Recall using the formulas above.

2. Analyze the Model
Look at the numbers you calculated in Question 1. Is this model a "Best Fit"? Explain why or why not based on the accuracy and error rate.

3. Coding Task
Write the Python code specifically for **Step 4 (Pre-processing)**.

- Include the code to split the data (Train/Test split).
- Include the code for Feature Scaling.
- **Explain:** Write a comment explaining exactly what every line of that code is doing.

Conclusion:
By completing this assignment, you have successfully explored how the Logistic Regression model works, how to implement it in Python, and how to verify if it is accurate.

Lab Assignment No: 6

Data Analytics III

AIM:
Use Python to implement the **Naïve Bayes** classification algorithm on the Iris flower dataset.

OBJECTIVES:

- To learn how data science is used to solve real-world classification problems.
- To learn how to analyze open-source data using Python tools.

PROBLEM STATEMENT:

1. Write a program to classify Iris flowers using the **Naïve Bayes** algorithm.
2. Evaluate the performance of your model by calculating the **Confusion Matrix** (which helps identify True Positives, False Positives, etc.) and checking the model's **Accuracy**.

OUTCOMES:

- **CO1:** Apply data science logic to analyze problems.
- **CO2:** Arrange data using statistical methods.
- **CO3:** Build and test data analytics algorithms.

REQUIREMENTS:

- **Software:** Jupyter Notebook or IPython (version 3.8.1 or higher).

THEORY: What is Naïve Bayes?

The "Naïve" Part:
This algorithm is called "Naïve" because it makes a very simple assumption: it pretends that every feature of data is completely separate and unrelated to the others.

- *Example:* If you are identifying a fruit, the algorithm looks at color (yellow), shape (curved), and taste (sweet). Even though these things usually go together to make a Banana, Naïve Bayes treats them as three separate clues that independently suggest "Banana."

The "Bayes" Part:
It relies on **Bayes' Theorem**, a mathematical formula used to calculate probability.

- **Formula:** $P(A|B)=P(B|A)\times P(A)P(B)P(A|B)=P(B)P(B|A)\times P(A)$
- In simple terms: It calculates the probability of an event happening (Event A) based on prior knowledge of conditions related to the event (Event B).

How it works in this assignment:
We will look at the measurements of a flower (length and width). Based on these numbers, the algorithm calculates the probability of the flower belonging to a specific species (like *Setosa* or *Virginica*). The species with the highest probability is chosen as the answer.

PROBLEM ANALYSIS: The Dataset

We are using the **Iris Flower Dataset**.

- **The Goal:** Predict the species of a flower.
- **The Classes (Answers):** There are 3 types of flowers: *Setosa, Versicolor, and Virginica*.
- **The Features (Clues):** We have 4 measurements for every flower:
    1. Sepal Length
    2. Sepal Width
    3. Petal Length
    4. Petal Width

STEP-BY-STEP IMPLEMENTATION

Step 1: Import the necessary tools

First, we need to load the Python libraries that will help us do the math and handle the data tables.

- **Numpy:** For math.
- **Pandas:** For handling the data table.
- **Matplotlib:** For graphing (if needed).

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd
```

Step 2: Load the Data

We download the Iris data and separate it into two parts:

- **X (Features):** The 4 measurement columns.
- **Y (Target):** The 'Species' column (the answer key).

```
# Reading the file from a web link

dataset =
pd.read_csv('https://raw.githubusercontent.com/mk-gurucharan/Classificat
ion/master/IrisDataset.csv')

# Selecting columns [all rows, first 4 columns] for X

X = dataset.iloc[:,:4].values

# Selecting the 'species' column for Y

y = dataset['species'].values

# Display the first 5 rows to check data

dataset.head(5)
```

Step 3: Split the Data (Train vs. Test)

We cannot test the model on the same data it studied, or it will just memorize the answers. We split the data:

- **80% Training:** Used to teach the model.
- **20% Testing:** Used to see if the model actually works.

```
from sklearn.model_selection import train_test_split

# test_size=0.2 means 20% of data is for testing

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2)
```

Step 4: Feature Scaling

Some measurements might be large (e.g., 5.1) and others small (e.g., 0.2). Feature scaling adjusts these numbers so they are all on a similar scale. This helps the computer calculate much faster.

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
```

```
# Fit and transform the training data

X_train = sc.fit_transform(X_train)

# Only transform the test data (do not fit again)

X_test = sc.transform(X_test)
```

Step 5: Train the Model

We use the **Gaussian Naïve Bayes** model (GaussianNB). This is the standard version used for continuous numbers like our flower measurements. We "fit" the model using our Training data.

```
from sklearn.naive_bayes import GaussianNB

classifier = GaussianNB()

# The model 'studies' the data here

classifier.fit(X_train, y_train)
```

Step 6: Make Predictions

Now that the model is trained, we give it the Test questions (X_test) and ask it to predict the answers (y_pred).

```
y_pred = classifier.predict(X_test)

print(y_pred)a
```

Step 7: Check Accuracy (Confusion Matrix)

We compare the model's guesses (y_pred) against the actual answers (y_test).

- **Confusion Matrix:** A table showing how many it got right vs. wrong for each flower type.
- **Accuracy:** The percentage of total correct answers.

```
from sklearn.metrics import confusion_matrix, accuracy_score

cm = confusion_matrix(y_test, y_pred)

print("Accuracy : ", accuracy_score(y_test, y_pred))

print(cm)

# Sample Output Explanation:

# [[14,  0,  0],  <- Class 1: 14 correct, 0 wrong
#  [ 0,  7,  0],  <- Class 2: 7 correct, 0 wrong

#  [ 0,  1,  8]]  <- Class 3: 8 correct, 1 wrong (it thought it was
Class 2)
```

Result Analysis: In the example above, out of 30 flowers, 29 were correct. The accuracy is 96.67%.

Step 8: Side-by-Side Comparison

To see exactly where the mistake happened, we can put the Real Values and Predicted Values next to each other in a table.

```
df = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})

print(df)
```

CONCLUSION

We successfully implemented the Naïve Bayes algorithm. By splitting our data and training the model, we achieved high accuracy (approx 96%) in classifying the Iris flower species based on their petal and sepal measurements.

Lab Assignment No: 7

AIM:
Text Analytics – Take a sample text document and perform basic text preprocessing.

OBJECTIVES

- To learn the basic ideas of data science used for solving real-world problems.
- To enable students to perform data analysis using logistic regression in Python on any open-source dataset.

PROBLEM STATEMENT

Text Analytics:

1. Take a sample text document.
2. Apply the following preprocessing steps:
   - Tokenization
   - POS (Part-of-Speech) Tagging
   - Stop word removal
   - Stemming
   - Lemmatization
3. Represent the document using:
   - Term Frequency (TF)
   - Inverse Document Frequency (IDF)

OUTCOMES

- **CO3:** Implement and test data analytics algorithms.
- **CO4:** Perform text preprocessing on text data.

SOFTWARE & HARDWARE REQUIREMENTS

- Software: Jupyter / IPython, Python 3.8.1
- Usual computer hardware that supports Python and Jupyter.

THEORY

Introduction

NLTK (Natural Language Toolkit) is a popular Python library used for working with human language data (text). Using NLTK, we can do tasks like:

- Sentiment analysis
- Tokenization
- POS tagging
- Stemming and lemmatization
- Topic segmentation
- Named Entity Recognition, etc.

It is free, open source, easy to use, well documented, and has a large user community.
NLTK helps us clean, analyze, and understand text so that computers can work with it.

Text analytics is widely used today. Some examples:

- Twitter: Analyze tweets to find trending topics and public reactions.
- Amazon: Understand user reviews and product feedback.
- BookMyShow: Analyze people's opinions about movies.
- YouTube: Understand viewers' comments on videos.

Key topics:

- Text Analytics and Natural Language Processing (NLP)
- Differences between Text Analytics, NLP, and Text Mining
- Text processing operations using NLTK:
    - Tokenization
    - Stopword removal
    - Lexicon normalization (Stemming and Lemmatization)
    - POS Tagging
- Sentiment Analysis
- Text Classification
- Doing Sentiment Analysis using Text Classification

TEXT ANALYSIS OPERATIONS USING NLTK

Installing and Importing NLTK

```
!pip install nltk
```

Then in Python:

```
import nltk
```

NLTK provides many functions for text processing, such as tokenizing text, tagging words with parts of speech, stemming, and more.

TOKENIZATION

Tokenization is usually the first step in text processing.
It means splitting a large text (like a paragraph) into smaller units (tokens):

- Sentence tokenization: split text into sentences
- Word tokenization: split text into words

A *token* is one piece of text, for example a word or a sentence.

Sentence Tokenization

```
from nltk.tokenize import sent_tokenize

text = """Hello Mr. Smith, how are you doing today? The weather is
great, and city is awesome.

The sky is pinkish-blue. You shouldn't eat cardboard"""

tokenized_text = sent_tokenize(text)

print(tokenized_text)
```

Output:

['Hello Mr. Smith, how are you doing today?', 'The weather is great, and city is awesome.', 'The sky is pinkish-blue.', "You shouldn't eat cardboard"]

The paragraph is split into individual sentences.

Word Tokenization

```
from nltk.tokenize import word_tokenize

tokenized_word = word_tokenize(text)

print(tokenized_word)
```

Output:

['Hello', 'Mr.', 'Smith', ',', 'how', 'are', 'you', 'doing', 'today', '?', 'The', 'weather', 'is', 'great', ',', 'and', 'city', 'is', 'awesome', '.', 'The', 'sky', 'is', 'pinkish-blue', '.', 'You', 'should', "n't", 'eat', 'cardboard']

Now the text is split into individual words and punctuation marks.

FREQUENCY DISTRIBUTION

We can count how often each word appears using FreqDist.

```
from nltk.probability import FreqDist

fdist = FreqDist(tokenized_word)

print(fdist)
```

Example output:

<FreqDist with 25 samples and 30 outcomes>

To see the most frequent words:

```
fdist.most_common(2)
```

Output:

```
[('is', 3), (',', 2)]
```

To plot the frequency distribution:

```
import matplotlib.pyplot as plt

fdist.plot(30, cumulative=False)

plt.show()
```

STOPWORDS

Stopwords are very common words that usually do not add much meaning to the text for analysis, such as: "is, am, are, this, a, an, the," etc.

In NLTK, we can load a list of English stopwords and remove them from our tokens.

```
from nltk.corpus import stopwords

stop_words = set(stopwords.words("english"))

print(stop_words)
```

Removing Stopwords

```
filtered_sent = []

for w in tokenized_word:

    if w not in stop_words:

        filtered_sent.append(w)

print("Tokenized Sentence:", tokenized_word[:10])

print("Filtered Sentence:", filtered_sent[:10])
```

Example (for part of a sentence):

Tokenized Sentence: ['Hello', 'Mr.', 'Smith', ',', 'how', 'are', 'you', 'doing', 'today', '?']

Filtered Sentence: ['Hello', 'Mr.', 'Smith', ',', 'today', '?']

Words like "how, are, you, doing" are removed as stopwords.

LEXICON NORMALIZATION

Different forms of a word can appear in text, for example:

- connection, connected, connecting

These all relate to the same basic word “connect”.
Lexicon normalization tries to reduce such word forms to a common root/base form.
Two common methods are:

- Stemming
- Lemmatization

STEMMING

Stemming is a simple method to cut off word endings (suffixes) to get a root form, often not a real word but a stem.

Example:
“connection”, “connected”, “connecting” → “connect”

```
from nltk.stem import PorterStemmer

from nltk.tokenize import word_tokenize

ps = PorterStemmer()

stemmed_words = []

for w in filtered_sent:

    stemmed_words.append(ps.stem(w))

print("Filtered Sentence:", filtered_sent)

print("Stemmed Sentence:", stemmed_words)
```

Example output:

Filtered Sentence: ['Hello', 'Mr.', 'Smith', ',', 'today', '?']

Stemmed Sentence: ['hello', 'mr.', 'smith', ',', 'today', '?']

(Here, no big change because these words are already simple.)

LEMMATIZATION

Lemmatization reduces words to their dictionary base form (lemma), which is a correct word in the language.
It uses vocabulary and grammatical information.

It is usually more accurate than stemming.
For example, the lemma of “better” is “good” (this cannot be found by just cutting suffixes).

```
from nltk.stem.wordnet import WordNetLemmatizer

from nltk.stem.porter import PorterStemmer

lem = WordNetLemmatizer()

stem = PorterStemmer()

word = "flying"

print("Lemmatized Word:", lem.lemmatize(word, "v"))

print("Stemmed Word:", stem.stem(word))
```

Output:

Lemmatized Word: fly

Stemmed Word: fli

Lemmatization gives the full word “fly”, while stemming produces “fli”.

POS (PART-OF-SPEECH) TAGGING

POS tagging marks each word with its grammatical type:

- Noun (NN), Proper Noun (NNP)
- Verb (VB, VBD, VBN, etc.)
- Adjective (JJ)
- Adverb (RB), etc.

It uses the context of the sentence to decide the correct tag.

```
sent = "Albert Einstein was born in Ulm, Germany in 1879."

tokens = nltk.word_tokenize(sent)

print(tokens)
```

Output:

['Albert', 'Einstein', 'was', 'born', 'in', 'Ulm', ',', 'Germany', 'in', '1879', '.']

Now apply POS tagging:

```
nltk.pos_tag(tokens)
```

Example output:

```
[('Albert', 'NNP'),

 ('Einstein', 'NNP'),

 ('was', 'VBD'),

 ('born', 'VBN'),

 ('in', 'IN'),

 ('Ulm', 'NNP'),

 (',', ','),

 ('Germany', 'NNP'),

 ('in', 'IN'),

 ('1879', 'CD'),

 ('.', '.')]
```

This can be written as:

```
Albert/NNP Einstein/NNP was/VBD born/VBN in/IN Ulm/NNP ,/, Germany/NNP
in/IN 1879/CD ./.
```

CONCLUSION

In this assignment, we learned the basic steps of text analytics using and NLTK:

- Tokenizing text into sentences and words
- Removing stopwords
- Normalizing words using stemming and lemmatization
- Tagging words with their parts of speech

These preprocessing steps are essential before performing advanced tasks like text classification, sentiment analysis, and calculating TF–IDF for document representation.

Lab Assignment No: 8

AIM:
Data Visualization – I (Using the built-in Titanic dataset)

OBJECTIVES

- To learn basic ideas of data science for solving real-world problems.
- To perform data analysis using logistic regression in Python on an open-source dataset.

PROBLEM STATEMENT

Use the inbuilt dataset **"titanic"** from Seaborn.
This dataset has 891 rows and contains information about passengers who travelled on the Titanic.

Using the **Seaborn** library, explore the dataset and try to find any patterns in the data.

Write Python code to check how the ticket price (column: **fare**) is distributed by plotting a **histogram**.

OUTCOMES

- **CO3:** Implement and evaluate data analytics algorithms.
- **CO4:** Implement data visualization techniques.
- **CO5:** Implement data visualization techniques.

SOFTWARE & HARDWARE REQUIREMENTS

- Jupyter Notebook / IPython
- Python 3.8.1

THEORY

Seaborn is a very useful Python library for data visualization.
It is built on top of **Matplotlib** and provides more advanced and attractive plotting options.

With Seaborn, you can create many types of plots such as:

- Distribution plots
- Categorical plots
- Regression plots
- Matrix plots
- Grid plots

In this experiment, we will focus on **distribution** and **categorical** plots.
(Regression, matrix and grid plots can be studied later.)

Installing Seaborn

You can install Seaborn in two common ways:

1. Using **pip** (if you use standard Python installation):

```
pip install seaborn
```

2. Using **conda** (if you use Anaconda):

```
conda install seaborn
```

The Dataset

We will use the **Titanic** dataset that comes built-in with Seaborn.
You do not need to download it separately. Just call load_dataset() with the dataset name.

Example:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

dataset = sns.load_dataset('titanic')

dataset.head()
```

Distributional Plots

Distribution plots show how data values are spread or distributed.

1. distplot

distplot() (in older Seaborn versions) draws a histogram for a single column and can also show a smooth curve (KDE – Kernel Density Estimate).

To see how ticket prices (fare) are distributed:

```
sns.distplot(dataset['fare'])
```

This will show:

- Bars (histogram) for ticket prices
- A smooth line (KDE) on top of the bars

From this, you can observe that most ticket prices are between 0 and 50 dollars.

If you want to hide the density (KDE) curve and only show the histogram:

```
sns.distplot(dataset['fare'], kde=False)
```

You can also change the number of **bins** (bars) in the histogram to control how detailed it looks:

```
sns.distplot(dataset['fare'], kde=False, bins=10)
```

Here, the histogram is divided into 10 bins.
From such a plot, you may notice that more than 700 passengers had ticket prices between 0 and 50.

---

2. jointplot

jointplot() shows the relationship between **two** columns.
It combines:

- A distribution plot for the x-axis variable (top)
- A distribution plot for the y-axis variable (right side)
- A central plot (usually a scatter plot) showing how both variables relate

Example: relationship between **age** and **fare**:

```
sns.jointplot(x='age', y='fare', data=dataset)
```

This creates:

- Histogram of age at the top
- Histogram of fare on the right
- Scatter plot of age vs fare in the center

From this, we generally do not see a strong correlation between age and fare.

You can change the central plot type using the kind parameter.
For example, to plot a **hexbin** (hexagonal) plot instead of scatter:

```
sns.jointplot(x='age', y='fare', data=dataset, kind='hex')
```

In a hex plot:

- The plane is divided into hexagons.
- Darker hexagons mean more data points in that area.

From such a plot, you may see that:

- Many passengers were between 20 and 30 years old.
- Many of them paid between 10 and 50 for their ticket.

---

3. pairplot

pairplot() automatically creates joint plots for all pairs of **numeric** and **boolean** columns in the dataset.

You just pass the whole dataset:

```
sns.pairplot(dataset)
```

This gives a grid of plots, showing distributions and relationships between many variables at once.

CONCLUSION

In this assignment, we learned how to:

- Load the Titanic dataset using Seaborn.
- Visualize the distribution of the ticket fare using histograms.

- Use Seaborn plots like distplot, jointplot, and pairplot to explore data visually.

Thus, we have carried out basic **Data Visualization – I** using the Titanic dataset.

Lab Assignment No: 9

AIM:
Data Visualization – II

1. Use the built-in dataset **"titanic"** (same as in the previous assignment).

OBJECTIVES

- To understand the basic ideas of data science for solving real-world problems.
- To enable students to perform data analysis (for example, using logistic regression) in Python on any open-source dataset.

PROBLEM STATEMENT

Data Visualization – II

1. Use the built-in **"titanic"** dataset.
   - Draw a **box plot** to show how **age** is distributed for each **gender**, and also show whether the passengers **survived or not**.
   - Use the columns: **"gender"** and **"age"**, and survival information.
   - Write your observations and conclusions based on this plot.

OUTCOMES

- **CO3:** Implement and evaluate data analytics algorithms.
- **CO4:** Implement data visualization techniques.
- **CO5:** Implement data visualization techniques.

SOFTWARE & HARDWARE REQUIREMENTS

- Jupyter / IPython
- Python 3.8.1

THEORY

1. Categorical Plots

Categorical plots are used when we work with **categorical (discrete)** data such as gender, class, or yes/no values.
They usually show:

- A categorical column vs. another categorical column, or
- A categorical column vs. a numeric column.

Below are some common categorical plots.

---

2. Bar Plot

A **bar plot** shows the **average** (or some other summary) of a numeric variable for each category.

In Seaborn, sns.barplot() is used.

- The first argument: categorical column (x-axis)
- The second argument: numeric column (y-axis)
- The third argument: dataset

Example: to find the mean age of male and female passengers:

sns.barplot(x='gender', y='age', data=dataset)

This will show that:

- The **average age of male** passengers is a little less than 40.
- The **average age of female** passengers is around 33.

You can also use bar plots to show other summary values (like **standard deviation**, **median**, etc.).
To do this, pass a function to the estimator parameter.

Example: standard deviation of age for each gender:

```
import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns
```

```
sns.barplot(x='gender', y='age', data=dataset, estimator=np.std)
```

Here, np.std (from NumPy) is used to calculate the standard deviation of age for male and female passengers.

---

3. Count Plot

A **count plot** is similar to a bar plot, but instead of showing a summary (like mean), it shows the **count of rows** in each category.

Example: to count how many male and female passengers there are:

```
sns.countplot(x='gender', data=dataset)
```

This will display bars representing the number of male and female passengers.

---

4. Box Plot

A **box plot** shows the **distribution** of a numeric variable (like age) for each category (like gender). It displays:

- Median
- Quartiles
- Minimum and maximum (within a range)
- **Outliers** as separate points

Basic box plot of age by gender:

```
sns.boxplot(x='gender', y='age', data=dataset)
```

Any points that lie far outside the main range of values are plotted as dots and are called **outliers**.

You can also add more information to the box plot using the hue parameter. For example, to see age distribution by gender **and** survival status:

```
sns.boxplot(x='gender', y='age', data=dataset, hue="survived")
```

From this plot, you can observe things like:

- Among **male** passengers, **younger males** seem to have survived more often than older ones.
- For **female** passengers, the spread (variation) in age is larger for those who **did not survive** compared to those who **survived**.

5. Violin Plot (Optional Extension)

A **violin plot** combines a box plot with a **density plot**, giving more detail about the distribution of the data.

Instead of drawing two separate plots for “survived” and “not survived”, we can:

- Use one violin plot,
- Split it into two halves: one half for survived, one half for not survived.

Example:

```
sns.violinplot(x='gender', y='age', data=dataset, hue='survived',
split=True)
```

This gives a clear comparison of age distribution for:

- Survived vs. not survived
- For both male and female passengers

CONCLUSION

In this assignment, we used the **Titanic dataset** to practice **Data Visualization – II** by creating and interpreting various categorical plots (bar plots, count plots, box plots, and violin plots).

Lab Assignment No: 10

AIM:
Data Visualization – III
Load the Iris flower dataset (or any other dataset) into a DataFrame and analyze it.

OBJECTIVES

- To understand basic data science concepts for solving real-world problems.
- To perform data analysis using logistic regression in Python on any open-source dataset.

PROBLEM STATEMENT

Data Visualization – III

Download the Iris flower dataset (or any similar dataset) and load it into a DataFrame
(example: https://archive.ics.uci.edu/ml/datasets/Iris).

Then:

1. Find how many features (columns) the dataset has and identify their types (for example, numeric, categorical/nominal).
2. Draw a histogram for each feature to show how the values are distributed.
3. Draw a boxplot for each feature. Compare the distributions and find any outliers.

OUTCOMES

- **CO3:** Apply and evaluate data analytics algorithms.
- **CO5:** Use different data visualization methods.

SOFTWARE & HARDWARE REQUIREMENTS

- Jupyter Notebook / IPython
- Python 3.8.1

THEORY

Data Analysis on the Iris Flower Dataset

Download the Iris dataset (or any similar dataset) and load it into a DataFrame.
(Example link: https://archive.ics.uci.edu/ml/datasets/Iris)

Using Python or R, do the following:

1. Find how many features (columns) are present and what type each feature is (numeric, nominal, etc.).
2. Calculate and display summary statistics for every feature in the dataset, such as:
    - Minimum value
    - Maximum value
    - Mean
    - Range
    - Standard deviation
    - Variance
    - Percentiles
3. **Data Visualization – Histograms:**
   Create a histogram for each feature to show the distribution of its values. Plot every histogram.
4. **Data Visualization – Boxplots:**
   Create a boxplot for each feature. Combine all boxplots into one figure if possible. Compare the distributions and identify any outliers.

SAMPLE CODE (USING PANDAS, MATPLOTLIB, SEABORN)

```
import matplotlib.pyplot as plt

import pandas as pd

path = "iris.csv"

df = pd.read_csv(path, header=None)

headers = ["Sepal-length", "Sepal-width", "Petal-length", "Petal-width",
"Species"]

df.columns = headers

print(df.head())

print(df.tail())
print(df.info())

print(df.shape)

print(df.dtypes)

print(df.describe())

# Histograms for each feature

df.hist()

plt.show()

# Boxplots for each feature

df.boxplot()

plt.show()

# Scatter plots between different pairs of features

plt.scatter(df["Sepal-length"], df["Sepal-width"])

plt.xlabel('Sepal Length')

plt.ylabel('Sepal Width')

plt.show()

plt.scatter(df["Sepal-length"], df["Petal-length"])

plt.xlabel('Sepal Length')

plt.ylabel('Petal Length')

plt.show()

plt.scatter(df["Sepal-length"], df["Petal-width"])

plt.xlabel('Sepal Length')

plt.ylabel('Petal Width')

plt.show()
plt.scatter(df["Sepal-width"], df["Sepal-length"])

plt.xlabel('Sepal Width')

plt.ylabel('Sepal Length')

plt.show()

plt.scatter(df["Sepal-width"], df["Petal-length"])

plt.xlabel('Sepal Width')

plt.ylabel('Petal Length')

plt.show()

plt.scatter(df["Sepal-width"], df["Petal-width"])

plt.xlabel('Sepal Width')

plt.ylabel('Petal Width')

plt.show()

plt.scatter(df["Petal-length"], df["Sepal-length"])

plt.xlabel('Petal Length')

plt.ylabel('Sepal Length')

plt.show()

plt.scatter(df["Petal-length"], df["Sepal-width"])

plt.xlabel('Petal Length')

plt.ylabel('Sepal Width')

plt.show()

plt.scatter(df["Petal-length"], df["Petal-width"])

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

plt.show()
plt.scatter(df["Petal-width"], df["Sepal-length"])

plt.xlabel('Petal Width')

plt.ylabel('Sepal Length')

plt.show()

plt.scatter(df["Petal-width"], df["Sepal-width"])

plt.xlabel('Petal Width')

plt.ylabel('Sepal Width')

plt.show()

plt.scatter(df["Petal-width"], df["Petal-length"])

plt.xlabel('Petal Width')

plt.ylabel('Petal Length')

plt.show()
```

Another example with generic column names:

```
import numpy as np

import pandas as pd

df = pd.read_csv("iris-flower-dataset.csv", header=None)

df.columns = ["col1", "col2", "col3", "col4", "col5"]

df.head()

# Number of columns

column_count = len(list(df))

print(column_count)

df.info()

# Unique classes in target column

print(np.unique(df["col5"]))
```

```
df.describe()
```

Histograms and boxplots using Seaborn and Matplotlib:

```
import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

fig, axes = plt.subplots(2, 2, figsize=(16, 8))

axes[0, 0].set_title("Distribution of First Column")

axes[0, 0].hist(df["col1"])

axes[0, 1].set_title("Distribution of Second Column")

axes[0, 1].hist(df["col2"])

axes[1, 0].set_title("Distribution of Third Column")

axes[1, 0].hist(df["col3"])

axes[1, 1].set_title("Distribution of Fourth Column")

axes[1, 1].hist(df["col4"])

# Data for boxplots

data_to_plot = [df["col1"], df["col2"], df["col3"], df["col4"]]

sns.set_style("whitegrid")

# Create figure for boxplot

fig = plt.figure(1, figsize=(12, 8))

# Add axes

ax = fig.add_subplot(111)

# Create the boxplot

bp = ax.boxplot(data_to_plot)
```

```
plt.show()
```

CONCLUSION

In this experiment, we loaded the Iris dataset into a DataFrame, calculated summary statistics, and created different visualizations (histograms, boxplots, and scatter plots). In this way, we have completed Data Visualization III.

Lab Assignment No: 11

AIM:
 Write a Java program for a simple WordCount application that counts how many times each word appears in a given input text, using the Hadoop MapReduce framework in local (standalone) mode.

OBJECTIVES:

- To understand the basic principles of data science for solving real-world problems.
- To enable students to perform data analysis using logistic regression in Python on any open-source dataset.

PROBLEM STATEMENT:
 Develop a Java program for a simple WordCount application that uses the Hadoop MapReduce framework (running in local/standalone mode) to count the number of occurrences of each word in a given input file.

OUTCOMES:
 CO1: Apply data science principles to analyze real-time / real-world problems.
 CO6: Use modern tools and technologies to process and analyze Big Data.

SOFTWARE & HARDWARE REQUIREMENTS:

- Jupyter / IPython
- Python 3.8.1

THEORY:
 This is an example of a Hadoop MapReduce application working on weather data.
 The program reads text input files, splits each line into weather records for different stations, and then calculates the average values for temperature, dew point, and wind speed.

The final output is a locally sorted list of weather stations, each having a 12-element attribute vector representing the average temperature, dew point, and wind speed for 4 sections of each month.

This assignment will focus on:

- Installing Hadoop
- Writing Java code for WordCount
- Preparing an input file

Steps to install Hadoop

Step 1) Create a directory named words

```
mkdir words
```

Step 2) Download hadoop-core-1.2.1.jar, which is needed to compile and run the MapReduce program. Go to the following link:
http://mvnrepository.com/artifact/org.apache.hadoop/hadoop-core/1.2.1

Step 3) Copy the downloaded JAR file into the words folder.

Step 4) Write the WordCount.java program and place it inside the words directory.

Step 5) Create an input file input1.txt in your home directory and add some sample text.

Step 6) Go to the words directory and compile the Java code. For example:

```
javac -classpath /home/vijay/words/hadoop-core-1.2.1.jar
/home/vijay/words/WordCount.java
```

(or, depending on your Hadoop installation:)

```
javac -classpath \

$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-core-3.2.4.j
ar:\

$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-common-3.2.4
.jar:\

$HADOOP_HOME/share/hadoop/common/hadoop-common-3.2.4.jar \

/home/gurukul/WordCount.java
```

Step 7) Create a JAR file from the compiled classes:

```
jar -cvf words.jar -c words/ .
```

Step 8) Move one level up (to the parent directory) and run the following Hadoop commands:

```
hadoop fs -mkdir /input

hadoop fs -put input1.txt /input

hadoop fs -ls /input

hadoop jar /home/vijay/words/words12.jar WordCount /input/input1.txt
/out321
```

```
hadoop fs -ls /out321

hadoop fs -cat /out321/part-r-00000
```

(You can also check the output using the HDFS web interface:
Browsing HDFS -> Utilities -> Browse the File System -> /)

Java Code for WordCount

```
import java.io.IOException;

import java.util.*;

import org.apache.hadoop.conf.*;

import org.apache.hadoop.fs.*;

import org.apache.hadoop.conf.*;

import org.apache.hadoop.io.*;

import org.apache.hadoop.mapreduce.*;

import org.apache.hadoop.mapreduce.lib.input.*;

import org.apache.hadoop.mapreduce.lib.output.*;

import org.apache.hadoop.util.*;


public class WordCount extends Configured implements Tool

{

    public static void main(String args[]) throws Exception

    {

        int res = ToolRunner.run(new WordCount(), args);

        System.exit(res);
```

```
}

public int run(String[] args) throws Exception
{
    Path inputPath = new Path(args[0]);
    Path outputPath = new Path(args[1]);
    Configuration conf = getConf();
    Job job = new Job(conf, this.getClass().toString());

    job.setJarByClass(WordCount.class);

    FileInputFormat.setInputPaths(job, inputPath);
    FileOutputFormat.setOutputPath(job, outputPath);

    job.setJobName("WordCount");

    job.setMapperClass(Map.class);
    job.setCombinerClass(Reduce.class);
    job.setReducerClass(Reduce.class);

    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);

    job.setOutputKeyClass(Text.class);
```

```
        job.setOutputValueClass(IntWritable.class);


        job.setInputFormatClass(TextInputFormat.class);

        job.setOutputFormatClass(TextOutputFormat.class);


        return job.waitForCompletion(true) ? 0 : 1;

    }


    public static class Map extends Mapper<LongWritable, Text, Text, 
IntWritable>

    {

        private final static IntWritable one = new IntWritable(1);

        private Text word = new Text();


        public void map(LongWritable key, Text value, Mapper.Context 
context)

                throws IOException, InterruptedException

        {

            String line = value.toString();

            StringTokenizer tokenizer = new StringTokenizer(line);


            while (tokenizer.hasMoreTokens())

            {

                word.set(tokenizer.nextToken());

                context.write(word, one);
            }
        }
    }

    public static class Reduce extends Reducer<Text, IntWritable, Text,
IntWritable>
    {
        public void reduce(Text key, Iterable<IntWritable> values,
Context context)
                throws IOException, InterruptedException
        {
            int sum = 0;
            for (IntWritable value : values)
            {
                sum += value.get();
            }
            context.write(key, new IntWritable(sum));
        }
    }
}
```

Sample Input File

Example content of input1.txt:

Pune

Mumbai

Nashik

Pune

Nashik

Kolapur

Additional Installation Notes (Weather Data Project)

Because this is a MapReduce project, you must first install Apache Hadoop.
For detailed instructions, see:

https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

After installing Hadoop, you can download and build the sample weather data project using Git and Maven:

```
git clone https://github.com/vasanth-mahendran/weather-data-hadoop.git

cd weather-data-hadoop

mvn install
```

To run it, first add the sample input file to HDFS:

```
hdfs dfs -put sample_weather.txt
```

Then start the MapReduce job:

```
hadoop jar ./target/weather-1.0.jar sample_weather.txt dataOutput
```

Assignment Questions

1. What is MapReduce? Explain it with a small example.
2. Write the steps to install Hadoop.

CONCLUSION

In this assignment, we learned how to use the Hadoop MapReduce framework on a local (standalone) setup by implementing a simple WordCount program and understanding how to run it with Hadoop.

Lab Assignment No: 12

AIM
To build a distributed application using MapReduce that reads and processes a system log file.

OBJECTIVES

- Understand the basic principles of data science used to solve real-world problems.
- Learn to do data analysis using logistic regression in Python on any open-source dataset.

PROBLEM STATEMENT
Develop a distributed MapReduce program that processes a system's log file.

OUTCOMES

- **CO1:** Use data science principles to analyze real-time / real-world problems.
- **CO6:** Work with modern tools and technologies to handle and analyze Big Data.

SOFTWARE & HARDWARE REQUIREMENTS

- Jupyter / IPython, Python 3.8.1
- Hadoop installed on Ubuntu (single-node setup)

THEORY

This experiment shows how to install and run Hadoop on:

- a standalone (single) machine, and
- a machine that can work as a node in a Hadoop cluster.

We will focus on setting up a **single-node Hadoop cluster** on Ubuntu, using **HDFS (Hadoop Distributed File System)**.

Hadoop is a Java-based framework used to run applications on large clusters of normal, low-cost machines. It has:

- a distributed file system similar to the Google File System (GFS)
- a parallel processing model based on MapReduce

HDFS is:

- a distributed file system
- fault-tolerant (can handle machine failures)
- designed for low-cost hardware
- best suited for applications that read/write very large data sets

Before setup, understand these components:

1. DataNode

- Stores the actual data blocks in HDFS.
- A real HDFS cluster has many DataNodes.
- Data is **replicated** across multiple DataNodes for reliability.

2. NameNode

- Central server that manages HDFS.
- Stores the **directory structure** (file names, locations, permissions).
- Knows *where* each piece of data is stored across the cluster.
- Does **not** store file content itself, only metadata.

3. JobTracker (MapReduce v1)

- Service that distributes MapReduce jobs to nodes in the cluster.
- Tries to assign tasks where the data is already stored (or at least in the same rack) to reduce network traffic.

4. TaskTracker (MapReduce v1)

- Runs on worker nodes.
- Receives tasks (Map, Reduce, Shuffle) from the JobTracker and executes them.

5. Secondary NameNode

- Helper node for the NameNode.
- Periodically creates checkpoints of HDFS metadata.
- Helps in recovery of the NameNode metadata, but it is **not** a backup NameNode.

6. ResourceManager (YARN)

- Manages all cluster resources (CPU, memory, etc.) under YARN.

- Coordinates with NodeManagers and ApplicationMasters for running distributed applications.

7. NodeManager (YARN)

- Runs on each worker machine.
- Manages containers (where tasks run), monitors resource usage (CPU, memory), and reports to ResourceManager.

8. JobHistoryServer

- Provides information about completed MapReduce jobs.
- Handles client requests related to job history (logs, status, etc.).

Prerequisite: Java

Hadoop needs Java to run.

Check Java version:

```
java -version
```

If Java is not installed, install it first.

STEPS: INSTALL HADOOP (DISTRIBUTED / SINGLE-NODE SETUP)

1. **Create a working folder**

   On the desktop, create a folder named logfiles1.
   Inside logfiles1, store these files:

   - access_log_short.csv (input log file)
   - SalesMapper.java
   - SalesCountryReducer.java
   - SalesCountryDriver.java

Step 1: Format the NameNode

Go to the Hadoop home directory and format the NameNode:

```
cd hadoop-2.7.3
bin/hadoop namenode -format
```

Formatting initializes HDFS metadata on this machine.

Step 2: Start Hadoop Daemons / Services

Go to the sbin directory:

```
cd hadoop-2.7.3/sbin
```

Now start the following services:

1. **Start NameNode**

   The NameNode manages the HDFS namespace and tracks all files and blocks.

```
./hadoop-daemon.sh start namenode
```

Start DataNode

The DataNode connects to the NameNode and stores the actual data blocks.

```
./hadoop-daemon.sh start datanode
```

Start ResourceManager

ResourceManager manages cluster resources and coordinates running applications under YARN.

```
./yarn-daemon.sh start resourcemanager
```

Start NodeManager

NodeManager runs on each node and manages containers and resource usage.

```
./yarn-daemon.sh start nodemanager
```

Start JobHistoryServer

This service provides job history information for completed MapReduce jobs.

```
      ./mr-jobhistory-daemon.sh start historyserver
```

Step 3: Check Running Hadoop Services

Use the jps command to verify which Java processes are running:

```
jps
```

You should see NameNode, DataNode, ResourceManager, NodeManager, JobHistoryServer, etc.

Prepare Files and Directories for MapReduce Job

From your home directory:

```
cd
sudo mkdir mapreduce_vijay
sudo chmod 777 -R mapreduce_vijay/
sudo chown -R vijay mapreduce_vijay/
sudo cp /home/vijay/Desktop/logfiles1/* ~/mapreduce_vijay/
cd mapreduce_vijay/
ls
sudo chmod +r *.*
```

Explanation:

- Create mapreduce_vijay folder.
- Give full permissions so you can read/write.
- Change the owner to user vijay.
- Copy all files from logfiles1 into mapreduce_vijay.
- Ensure all files are readable.

Set CLASSPATH for Hadoop and Your Classes

```
export
CLASSPATH="/home/vijay/hadoop-2.7.3/share/hadoop/mapreduce/hadoop-mapred
uce-client-core-2.7.3.jar:/home/vijay/hadoop-2.7.3/share/hadoop/mapreduc
e/hadoop-mapreduce-client-common-2.7.3.jar:/home/vijay/hadoop-2.7.3/shar
e/hadoop/common/hadoop-common-2.7.3.jar:~/mapreduce_vijay/SalesCountry/*
:$HADOOP_HOME/lib/*"
```

This tells javac and java where to find Hadoop libraries and your classes.

Compile the Java MapReduce Program

```
javac -d . SalesMapper.java SalesCountryReducer.java
SalesCountryDriver.java
ls
cd SalesCountry/
ls #(check that .class files are created)
cd ..
```

- -d . creates the proper package directory structure (SalesCountry/) for compiled classes.

Create Manifest File and JAR

```
nano Manifest.txt
```

Add this line to Manifest.txt:

```
Main-Class: SalesCountry.SalesCountryDriver
```

Save and close.

Now create the JAR file:

```
jar -cfm mapreduce_vijay.jar Manifest.txt SalesCountry/*.class
ls
```

mapreduce_vijay.jar is your MapReduce application JAR.

Put Input File into HDFS and Run the Job

```
cd
cd mapreduce_vijay/
sudo mkdir /input200
sudo cp access_log_short.csv /input200
$HADOOP_HOME/bin/hdfs dfs -put /input200 /
$HADOOP_HOME/bin/hadoop jar mapreduce_vijay.jar /input200 /output200
hadoop fs -ls /output200
hadoop fs -cat /out321/part-00000
```

Meaning:

- Create a local directory /input200 and copy access_log_short.csv into it.
- Upload /input200 directory to HDFS root using hdfs dfs -put.
- Run the MapReduce job:

    - Input HDFS path: /input200
    - Output HDFS path: /output200
- List the output directory in HDFS.
- Display the content of the output file part-00000.

Check HDFS Web Interface

- Open Mozilla Firefox (or any browser) and go to:

http://localhost:50070/dfshealth.html

Here you can see the NameNode web UI and HDFS health/status.

JAVA CODE TO PROCESS LOG FILE

This MapReduce program counts how many times each "country" (or item/word in the log) appears.

Mapper Class

```
package SalesCountry;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;

public class SalesMapper extends MapReduceBase implements
Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);

    public void map(LongWritable key, Text value, OutputCollector<Text,
IntWritable> output,
                    Reporter reporter) throws IOException {
        String valueString = value.toString();
        String[] SingleCountryData = valueString.split("-");
        output.collect(new Text(SingleCountryData[0]), one);
    }
}
```

- Reads each line of the log.
- Splits the line by -.

- Takes the first part (SingleCountryData[0]) as the key (e.g., country name).
- Emits (country, 1).

Reducer Class

```
package SalesCountry;

import java.io.IOException;
import java.util.*;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;

public class SalesCountryReducer extends MapReduceBase
        implements Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text t_key, Iterator<IntWritable> values,
                       OutputCollector<Text, IntWritable> output,
Reporter reporter) throws IOException {

        Text key = t_key;
        int frequencyForCountry = 0;

        while (values.hasNext()) {
            IntWritable value = (IntWritable) values.next();
            frequencyForCountry += value.get();
        }

        output.collect(key, new IntWritable(frequencyForCountry));
    }
}
```

- For each country key, sums all the 1s.
- Outputs (country, total_count).

Driver Class

```
package SalesCountry;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
```

```
public class SalesCountryDriver {
    public static void main(String[] args) {
        JobClient my_client = new JobClient();

        // Create a configuration object for the job
        JobConf job_conf = new JobConf(SalesCountryDriver.class);

        // Set a name of the Job
        job_conf.setJobName("SalePerCountry");

        // Specify data type of output key and value
        job_conf.setOutputKeyClass(Text.class);
        job_conf.setOutputValueClass(IntWritable.class);

        // Specify names of Mapper and Reducer Class
        job_conf.setMapperClass(SalesCountry.SalesMapper.class);

job_conf.setReducerClass(SalesCountry.SalesCountryReducer.class);

        // Specify formats of the data type of Input and output
        job_conf.setInputFormat(TextInputFormat.class);
        job_conf.setOutputFormat(TextOutputFormat.class);

        // Set input and output directories using command line arguments
        // args[0] = input directory on HDFS
        // args[1] = output directory to store the result
        FileInputFormat.setInputPaths(job_conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(job_conf, new Path(args[1]));

        my_client.setConf(job_conf);

        try {
            // Run the job
            JobClient.runJob(job_conf);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

- Configures and runs the MapReduce job on Hadoop.
- Uses command line arguments for HDFS input and output paths.

Sample Input File

Example contents of access_log_short.csv (each line will be processed):

```
Pune
Mumbai
Nashik
Pune
Nashik
```

The MapReduce program will count how many times each city appears.

CONCLUSION
We learned how to design and run a distributed application using Hadoop MapReduce to process a system log file, including setting up Hadoop services, compiling and packaging the Java MapReduce code, running the job on HDFS, and viewing the processed output.

WT
Lab Assignment No: 1

Title: *Case Study on Website Design Evaluation*

Objective:

To analyze and evaluate different websites to identify effective and ineffective design practices, and understand key considerations for website development prior to implementation.

Problem Statement:

Before the coding phase of website development, planning and analysis of existing websites is essential. Students are expected to visit a minimum of five websites from different domains, evaluate their design features (good or bad), and identify crucial design issues and usability considerations that influence website quality.

Software and Hardware Requirements

Hardware:
Standard PC/Laptop with internet connectivity

Software:
Modern Web Browser (Chrome/Firefox/Edge), OS (Windows/Linux/macOS)

Theory

1. Website and Web Application Overview

A website is a collection of interlinked web pages accessible via the internet through a web browser, generally providing static and informational content. Websites primarily serve content consumption purposes, such as business information, portfolios, blogs, and organizational communication.

A web application is an interactive system that supports user input, processing, and output through dynamic features. Web applications typically include authentication, database connectivity, and transactional components (e.g., e-commerce platforms, banking applications, learning management systems).

Key differences include:

- Interactivity
- User Input Handling
- Dynamic Content
- Backend Processing
- Database Integration

Modern web development emphasizes usability, performance, responsive design, accessibility, navigation clarity, and visual consistency.

---

2. Steps of Execution

1. Selection of 5 different websites from different domains.
2. Inspection of UI, UX, responsiveness, readability, performance, and navigation.
3. Classification as good or bad design based on evaluation criteria.
4. Recording observations in tabulated format.
5. Identification of design issues and best practices.
6. Preparing concluding remarks based on comparative analysis.

---

3. Evaluation Table

Sr. No.	Website URL	Purpose of Website	Things Liked in the Website	Things Disliked in the Website	Overall Evaluation (Good / Bad)

1

2

3

4

5

Conclusion

From the evaluation of these websites, I learned that good website design depends on clear navigation, neat layout, and easy user interaction. While developing a website, it is important to focus on usability, readability, and performance to make the site user-friendly.

Lab Assignment No: 2

Aim:
To design a static client web page (index.htm) for a sample client website using HTML and all three types of CSS: inline, internal, and external.

Objectives:

1. To understand basic HTML structure and commonly used tags.
2. To apply inline, internal, and external CSS for webpage styling.
3. To design a simple, structured client homepage with text, media, and form elements.

Software Used:
Operating System: Windows / Linux
Language: HTML5, CSS3
Tools: Any Text Editor (VS Code / Notepad), Web Browser (Chrome/Firefox)
Frameworks/Libraries: Not applicable (pure HTML and CSS)

Theory :
HTML (HyperText Markup Language) is a markup language used to structure content on the web using tags and attributes. A basic HTML document includes a <!DOCTYPE>, <html>, <head>, and <body> section.

Common tags: Heading tags (<h1>–<h6>), paragraph (<p>), line break (<br>), formatting (<b>, <i>, <u>), lists (<ul>, <ol>, <li>), tables (<table>, <tr>, <td>, <th>), images (<img>), and links (<a>). Frames (in older HTML) divide the browser window into multiple sections using <frameset> and <frame>.

Forms (<form>) collect user input through controls like <input>, <textarea>, <select>, and <button>. Action and method attributes define where and how form data is submitted.

CSS (Cascading Style Sheets) controls the presentation (layout, colors, fonts, spacing) of HTML elements. The CSS box model describes each element as content plus padding, border, and margin.

Types of CSS:

1. Inline CSS: style attribute within an HTML tag.
2. Internal CSS: <style> block inside the <head> of the HTML page.
3. External CSS: separate .css file linked using <link> in the <head>. External CSS improves reusability and consistency across multiple pages.

Algorithm / Procedure:

1. Identify the client website theme (e.g., restaurant, college, shop) and decide basic sections: header, navigation, main content, gallery, contact form, footer.
2. Create a new folder for the website project and plan the file structure (e.g., index.htm and style.css).
3. Open a text editor and create index.htm with proper HTML5 structure: <!DOCTYPE html>, <html>, <head>, and <body>.
4. In the <head> section, set the page title using <title> and define character encoding using <meta charset="UTF-8">.
5. Add an internal CSS block using <style> in the <head> to define some basic styles (e.g., body font, background, heading colors).
6. Create an external CSS file (e.g., style.css) in the same folder and link it in index.htm using the <link> tag in the <head>.
7. In the <body>, add a header section with a main heading (<h1>) and optionally a logo image (<img>).
8. Design a navigation menu using an unordered list (<ul>, <li>) with links (<a>) to different sections or sample pages.
9. Create content sections:
   a. Use <h2>, <p>, and lists (<ul>/<ol>) for textual information (about, services, menu, etc.).
   b. Insert images using <img> with appropriate attributes (src, alt, width/height).
   c. Use tables (<table>, <tr>, <td>, <th>) for structured data if required (e.g., price list, timings).
10. Demonstrate frames (if required by syllabus) by creating an additional simple frameset example HTML file that divides the window into two or more frames.
11. Add a contact/feedback form using <form> with controls: <input> (text, email, number), <textarea>, <select>, and submit/reset buttons.
12. Apply inline CSS for specific elements (e.g., style attribute for a single heading or paragraph) to demonstrate usage.
13. Add or refine internal and external CSS rules to control layout (margins, padding), colors, fonts, borders, and alignment of the page elements.
14. Save all files and open index.htm in a web browser to test layout, links, forms, and overall design.
15. Validate the page visually and logically; correct any alignment, styling, or content issues as needed.

Q&A

1. Q: What is HTML?
   A: HTML (HyperText Markup Language) is a markup language used to structure and

present content on the World Wide Web using tags.

2. Q: What is the purpose of the <!DOCTYPE html> declaration?
   A: It informs the browser about the HTML version and ensures standards-compliant rendering (HTML5 in this case).

3. Q: Differentiate between block-level and inline elements.
   A: Block-level elements start on a new line and take full width (e.g., <div>, <p>), while inline elements flow within a line and take only necessary width (e.g., <span>, <a>).

4. Q: List the three types of CSS.
   A: Inline CSS, Internal CSS, and External CSS.

5. Q: What is the main advantage of external CSS?
   A: External CSS allows a single stylesheet to control the appearance of multiple web pages, improving maintainability and consistency.

6. Q: Write any two attributes of the <img> tag and their purpose.
   A: src specifies the image file path; alt provides alternative text if the image cannot be displayed.

7. Q: What is the use of the <form> tag?
   A: The <form> tag is used to create an input form for collecting user data and submitting it to a server or processing script.

8. Q: Define a hyperlink and name the tag used for it.
   A: A hyperlink links one document or resource to another; it is created using the <a> (anchor) tag.

9. Q: What is the CSS box model?
   A: The CSS box model describes the layout of HTML elements as boxes consisting of content, padding, border, and margin.

10. Q: What is the role of the <head> section in an HTML document?
    A: The <head> section contains metadata, title, links to CSS/JS files, and internal styles that do not appear directly as page content.

Conclusion:
A static client homepage was successfully designed using HTML tags and all three types of

CSS, demonstrating basic web layout and styling concepts. The assignment helps in understanding how structure and presentation are combined to create a complete webpage.

Lab Assignment No: 3

Aim:
To design an XML document to store employee information of a business organization, validate it using DTD and XML Schema, and display the contents in tabular form using CSS/XSL.

Objectives:

1. To understand the structure and syntax of XML documents for data representation.
2. To validate XML documents using DTD and XML Schema.
3. To format and display XML data using CSS and XSL.

Software Used:

- Operating System: Windows / Linux
- Languages/Technologies: XML, DTD, XML Schema (XSD), CSS, XSL/XSLT
- Tools: Any text/XML editor (e.g., Notepad++, VS Code), Web browser (Chrome/Firefox/Edge)
- Optional Tools: XML validator or IDE with XML support (e.g., Eclipse, IntelliJ)

Theory :
XML (Extensible Markup Language) is a markup language used to store and transport structured data in a platform-independent and self-descriptive format. It uses user-defined tags and a hierarchical tree structure of elements and attributes.

DTD (Document Type Definition) defines the legal structure of an XML document. It specifies allowed elements, attributes, their relationships, and occurrence constraints. DTD can be internal (within the XML file) or external (in a separate .dtd file). It ensures the XML document is valid with respect to a given grammar.

XML Schema (XSD) is an XML-based schema language used to define the structure and data types of XML elements and attributes. It supports strong typing (string, integer, date, etc.), occurrence constraints, default values, and complex type definitions, making it more powerful and expressive than DTD.

CSS (Cascading Style Sheets) can be applied to XML to control the visual presentation of elements (fonts, colors, layouts). It uses selectors based on element names and attributes to style the XML when viewed in a browser that supports XML + CSS.

XSL (Extensible Stylesheet Language) and particularly XSLT (XSL Transformations) are used to transform XML documents into other formats, such as HTML. By defining XSL templates and match rules, XML content can be processed and output in a desired structure, such as an HTML table for tabular display of employee information.

Algorithm / Procedure:

1. Identify the data fields for employees (e.g., employee ID, name, department, designation, salary, contact details).
2. Design the logical structure of the XML document (root element, employee element, child elements, attributes if any).
3. Create a well-formed XML document containing multiple employee records based on the decided structure.
4. Design a DTD that specifies:
   a. The root element and its child elements.
   b. The structure and sequence of elements for an employee.
   c. Any attributes required for specific elements.
5. Link the DTD to the XML document (internal or external) and validate the XML for correctness.
6. Design an XML Schema (XSD) that:
   a. Declares elements and attributes.
   b. Assigns appropriate simple data types (string, integer, decimal, date, etc.).
   c. Defines complex types for employee and root-level structures.
7. Associate the XML document with the XML Schema using the proper namespace and schemaLocation attributes.
8. Validate the XML document against the XML Schema using a browser or XML validation tool.
9. Create a CSS file to style the XML elements (e.g., display employee records with fonts, borders, and basic layout).
10. Link the CSS stylesheet to the XML document and view the styled XML in a browser.
11. Create an XSL (XSLT) stylesheet that:
    a. Defines the output method (e.g., HTML).
    b. Iterates over employee elements.
    c. Generates an HTML table and maps XML elements to table rows and cells.
12. Link the XSL stylesheet to the XML document and open the XML in a browser to see the employees displayed in tabular form.
13. Verify correctness of data display, structure, and validation.

Q&A :

1. Q: What is XML and why is it used?
   A: XML (Extensible Markup Language) is a markup language used to store and

transport structured data in a platform-independent, self-describing format.

2. Q: Differentiate between well-formed and valid XML.
   A: Well-formed XML follows basic XML syntax rules; valid XML is well-formed and also conforms to a DTD or XML Schema.

3. Q: What is a DTD?
   A: DTD (Document Type Definition) defines the structure, elements, and attributes allowed in an XML document.

4. Q: List two limitations of DTD.
   A: It does not support rich data types and is not XML-based, making it less expressive and harder to extend than XML Schema.

5. Q: What is XML Schema (XSD)?
   A: XML Schema is an XML-based language used to define the structure, data types, and constraints of elements and attributes in an XML document.

6. Q: Give two advantages of XML Schema over DTD.
   A: It supports strong data types and namespaces, and it is itself written in XML, making it more powerful and extensible.

7. Q: What is the role of CSS in XML?
   A: CSS controls the visual presentation of XML elements (fonts, colors, layout) when the XML is viewed in a supporting browser.

8. Q: What is XSLT?
   A: XSLT (XSL Transformations) is a language used to transform XML documents into other formats such as HTML, text, or another XML structure.

9. Q: Why is XSLT preferred for tabular display of XML data?
   A: XSLT can transform XML into HTML tables, allowing fine-grained control over layout and enabling browsers to render structured tabular views.

10. Q: What is the purpose of the schemaLocation attribute in XML?
    A: It specifies the location (URL/path) of the XML Schema used to validate the XML document.

Conclusion:
The experiment successfully demonstrated how to design and validate an employee information

XML document using DTD and XML Schema and how to present the data in a readable tabular form using CSS/XSL.

Lab Assignment No: 4

Aim:

To design and implement a JavaScript-based calculator with input validation and interactive prompt/alert messages using HTML and CSS for the user interface.

Objectives:

1. To design a calculator UI using HTML form elements and CSS styling.
2. To implement JavaScript functions for arithmetic operations and input validation.
3. To use `alert()` and `prompt()` dialogs for user interaction and error handling.

Software Used:

- **Operating System:** Windows / Linux
- **Languages:** HTML5, CSS3, JavaScript
- **Tools/Editor:** Visual Studio Code / Notepad++ / Any text editor
- **Browser:** Google Chrome / Mozilla Firefox / Edge
- **Frameworks/Libraries:** Not required (plain JavaScript)

Theory :

- **JavaScript:** A client-side scripting language used to add behavior and interactivity to web pages.
- **DOM (Document Object Model):** Represents HTML elements as objects; JavaScript accesses and manipulates these objects using methods like `getElementById()`.
- **Events:** Actions such as `onclick`, `onchange`, or `onsubmit` that trigger JavaScript functions.

- **Form Validation:** Checking user input before processing, ensuring correct type (e.g., numeric) and non-empty values.
- **Alert & Prompt:**
  - `alert(message)`: Displays a message in a modal dialog to inform the user.
  - `prompt(message, default)`: Asks the user for input and returns the entered string.
- **Calculator Logic:** Uses basic arithmetic operators (`+`, `-`, `*`, `/`) and mathematical operations (e.g., square using `value * value` or `Math.pow()`).

Algorithm / Procedure:

1. **Design UI using HTML:**
   - Create text fields for input numbers and output/result.
   - Add buttons for digits (0–9) and operators (addition, subtraction, multiplication, division, square).
   - Add a clear/reset button.
2. **Apply CSS:**
   - Style the calculator container (width, background, padding).
   - Style text fields and buttons (font, size, color, spacing, hover effects if needed).
3. **Link JavaScript:**
   - Include an internal or external JavaScript file in the HTML document.
4. **Access Elements via DOM:**
   - Use `document.getElementById()` or similar to access input fields, result field, and buttons.
5. **Implement Event Handling:**
   - Attach `onclick` events to each button.
   - On each click, call corresponding JavaScript functions (e.g., `add()`, `subtract()`, `multiply()`, `divide()`, `square()`).
6. **Input Validation:**
   - Check for empty fields before performing calculations.
   - Verify that input values are numeric using `isNaN()` or similar checks.

    - If invalid, use `alert()` to show error messages such as “Enter valid numeric values”.
7. **Use Prompt and Alert:**
    - Use `prompt()` to optionally take input directly from a user dialog (e.g., asking for a number to square).
    - Use `alert()` to display results or invalid input notifications.
8. **Display Output:**
    - Perform the required operation using JavaScript arithmetic.
    - Set the result field value using DOM (e.g., `resultField.value = result;`).
9. **Test the Application:**
    - Test all operations: addition, subtraction, multiplication, division, square.
    - Test with valid, invalid, and boundary values (e.g., divide by zero, empty input).

Q&A :

1. **Q:** What is the main purpose of JavaScript in a web page?
   A: To add interactivity, dynamic behavior, and client-side logic to the web page.

2. **Q:** How do you display a simple message box in JavaScript?
   A: By using the `alert("message");` function.

3. **Q:** What is the use of the `prompt()` function in JavaScript?
   A: It displays a dialog asking the user for input and returns the entered text.

4. **Q:** Name any two HTML elements commonly used in a calculator interface.
   A: `<input>` (for text fields/buttons) and `<button>`.

5. **Q:** Which JavaScript method is used to access an element by its ID?
   A: `document.getElementById("id");`.

6. **Q:** How do you check if a value is *not* a number in JavaScript?
   A: By using the function `isNaN(value)`.

7. **Q:** Why is client-side validation important in web applications?
   A: It prevents invalid data entry, improves user experience, and reduces unnecessary server requests.

8. **Q:** Which operator is used for multiplication and division in JavaScript?
   A: `*` for multiplication and `/` for division.

9. **Q:** How can you find the square of a number in JavaScript?
   A: Using `num * num` or `Math.pow(num, 2)`.

10. **Q:** What is an event in JavaScript with respect to buttons in a calculator?
    A: An event is an action like a button click (`onclick`) that triggers a JavaScript function.

Conclusion:

The JavaScript-based calculator was successfully designed with HTML/CSS UI, arithmetic operations, and proper input validation using `alert()` and `prompt()` for user interaction. The experiment demonstrated effective use of client-side scripting for a functional web application.

Lab Assignment No: 5

Aim:
To implement a Java Servlet that connects to a database, executes an SQL `SELECT` query on the `ebookshop` table, and displays the table contents in an HTML response.

Objectives:

1. To understand the use of Java Servlets for dynamic web content generation.
2. To establish database connectivity using JDBC and execute SQL `SELECT` queries.
3. To display relational data in tabular HTML format through a servlet.

Software Used:

- **Operating System:** Windows / Linux
- **Programming Language:** Java
- **Web Server / Container:** Apache Tomcat (or any compatible servlet container)
- **Database:** MySQL / Oracle (with JDBC driver)
- **Tools / IDE:** JDK, Eclipse / NetBeans / IntelliJ IDEA
- **APIs / Frameworks:** Java Servlet API, JDBC

Theory :
A *Java Servlet* is a server-side Java program that extends the capabilities of a web server by generating dynamic responses to client requests. Servlets run inside a *servlet container* (e.g., Tomcat) that manages their lifecycle (`init()`, `service()`, `destroy()`) and handles HTTP protocol details.

`HttpServlet` provides methods like `doGet()` and `doPost()` to process HTTP requests. The servlet reads the request, performs processing (such as database access), and writes an HTTP response, usually in HTML, using a `PrintWriter`.

Database interaction in servlets is commonly performed using *JDBC (Java Database Connectivity)*. JDBC provides classes and interfaces to connect to a database, send SQL statements, and process results. For a `SELECT` query, `Statement` or `PreparedStatement` objects execute the query and return results as a `ResultSet`, which is iterated to fetch row-wise data.

In this experiment, the servlet connects to a database containing an `ebookshop` table with fields like `book_id`, `book_title`, `book_author`, `book_price`, and `quantity`, executes `SELECT` to retrieve all rows, and formats them in an HTML table for display in the browser.

Algorithm / Procedure:

1. **Database Preparation**
   1.1 Install and configure MySQL/Oracle database.
   1.2 Create a database/schema (e.g., `wtl_db`).
   1.3 Create the table `ebookshop(book_id, book_title, book_author, book_price, quantity)`.
   1.4 Insert a few sample records in `ebookshop` for testing.

2. **Environment Setup**
   2.1 Install JDK and Apache Tomcat.
   2.2 Configure the IDE (Eclipse/NetBeans/IntelliJ) with the server runtime.
   2.3 Add the appropriate JDBC driver JAR (e.g., MySQL Connector/J) to the project library.

3. **Create Web Application**
   3.1 Create a new Dynamic Web Project / Web Application.
   3.2 Ensure the Servlet API is available in the project build path.
   3.3 Set the project's target runtime to Tomcat.

4. **Configure Database Connection Parameters**
   4.1 Identify JDBC URL, username, and password for the database.
   4.2 Optionally, keep them in constants or configuration for reuse.

5. **Servlet Design (Logic Outline, No Code)**
   5.1 Extend `HttpServlet` to create a new servlet class.
   5.2 Override `doGet()` (or `doPost()`) to handle the request.

5.3 In `doGet()`/`doPost()`:
- Set response content type as `text/html`.
- Obtain `PrintWriter` from `HttpServletResponse`.
- Load JDBC driver class.
- Establish `Connection` to the database using JDBC.
- Create `Statement` or `PreparedStatement`.
- Execute `SELECT * FROM ebookshop;`.
- Retrieve result as `ResultSet`.
- Start building an HTML document and table (`<table>`).
- Iterate through `ResultSet` rows and print table rows (`<tr>`) with columns (`<td>`).
- Close HTML tags.
- Close `ResultSet`, `Statement`, and `Connection`.

6. **Servlet Mapping**
   6.1 Map the servlet to a URL pattern using `web.xml` or `@WebServlet` annotation.
   6.2 Ensure the mapping (e.g., `/DisplayBooksServlet`) is unique and correct.

7. **Build and Deploy**
   7.1 Clean and build the project in the IDE.
   7.2 Deploy the web application to Tomcat (automatic via IDE or by copying the WAR file).
   7.3 Start/restart the Tomcat server.

8. **Execution and Testing**
   8.1 Open a web browser.
   8.2 Enter the servlet URL (e.g., `http://localhost:8080/<project-name>/DisplayBooksServlet`).
   8.3 Verify that all records from `ebookshop` are displayed in a properly formatted HTML table.
   8.4 Check for handling of empty table or connection errors, if applicable.

Q&A :

1. **Q:** What is a Java Servlet?
   A: A Java Servlet is a server-side Java component that processes client requests and

generates dynamic web responses, typically over HTTP.

2. **Q:** Which class is commonly extended to create an HTTP-based servlet?
   A: The `HttpServlet` class.

3. **Q:** What are the main lifecycle methods of a servlet?
   A: `init()`, `service()`, and `destroy()`.

4. **Q:** What is a servlet container?
   A: It is a server component (e.g., Tomcat) that manages the lifecycle, configuration, and execution of servlets and handles HTTP communication.

5. **Q:** What is JDBC?
   A: JDBC (Java Database Connectivity) is a Java API that enables Java programs to connect to and interact with relational databases using SQL.

6. **Q:** List the basic steps for database access using JDBC.
   A: Load driver, get `Connection`, create `Statement`/`PreparedStatement`, execute query, process `ResultSet`, close resources.

7. **Q:** What is a `ResultSet`?
   A: `ResultSet` is a JDBC object that holds the data returned by executing an SQL `SELECT` query and allows row-wise access.

8. **Q:** Write the SQL query to retrieve all rows from the `ebookshop` table.
   A: `SELECT * FROM ebookshop;`

9. **Q:** How is a servlet mapped to a URL?
   A: By defining a mapping in `web.xml` or using the `@WebServlet` annotation with a URL pattern.

10. **Q:** Which HTTP method is generally preferred for read-only operations like displaying data, and why?
    A: `GET`, because it is intended for safe, idempotent retrieval of resources without modifying server data.

Conclusion:
By implementing this experiment, I learned how to connect a Java Servlet to a database using JDBC, execute an SQL `SELECT` query on the `ebookshop` table, and display the retrieved records in an HTML response.

Lab Assignment No: 6

Aim:
To implement a JSP program that connects to a database and displays the contents of the `students_info` table using the SQL `SELECT` query.

Objectives:

1. To understand the basic structure and execution model of JSP pages.
2. To perform database connectivity in JSP using JDBC.
3. To retrieve and display database records dynamically using SQL `SELECT`.

Software Used:

- **Operating System:** Windows / Linux
- **Languages:** Java, SQL, HTML
- **Database:** MySQL / Oracle
- **Tools/Servers:** JDK, Apache Tomcat, JDBC Driver, Web Browser
- **IDE (optional):** Eclipse / NetBeans / IntelliJ IDEA

Theory :

- **JSP (JavaServer Pages)** is a server-side technology used to create dynamic web pages by embedding Java code in HTML.
- JSP is translated into a **Servlet** by the container, then compiled and executed on the server.
- **JDBC (Java Database Connectivity)** provides APIs to connect Java applications (including JSP) to relational databases.
- To access a database from JSP, a **JDBC driver** is used to:
    1. Load the driver class.
    2. Establish a `Connection`.
    3. Create a `Statement` or `PreparedStatement`.
    4. Execute SQL queries and store results in a `ResultSet`.

- The `SELECT` query is used to read records from a table such as `students_info (stud_id, stud_name, class, division, city)`.
- The JSP page iterates over the `ResultSet` and prints the data in HTML (e.g., an HTML table).

Algorithm / Procedure:

1. **Create Database and Table**
   1.1 Start the database server (MySQL/Oracle).
   1.2 Create a database/schema (e.g., `college_db`).
   1.3 Create the table `students_info(stud_id, stud_name, class, division, city)`.
   1.4 Insert sample records into `students_info`.

2. **Configure Web Application Environment**
   2.1 Install and configure JDK.
   2.2 Install and configure Apache Tomcat server.
   2.3 Add the appropriate JDBC driver (e.g., MySQL Connector/J) to the web application's `lib` folder or server classpath.

3. **Create Web Application Structure**
   3.1 Create a dynamic web project or web folder structure (`/WEB-INF`, `web.xml`).
   3.2 Ensure Tomcat recognizes the application (deploy in `webapps` or via IDE).

4. **Design JSP Page for Display**
   4.1 Create a JSP file (e.g., `displayStudents.jsp`) in the web application.
   4.2 In the JSP, import necessary Java and JDBC classes using page directives.
   4.3 Specify database connection parameters (URL, username, password, driver).

5. **Establish Database Connection in JSP**
   5.1 Load the JDBC driver class.
   5.2 Open a `Connection` to the database.

6. **Execute SQL SELECT Query**
   6.1 Create a `Statement` or `PreparedStatement` object.
   6.2 Execute `SELECT * FROM students_info;`.

6.3 Store the results in a `ResultSet` object.

7. **Display Data Using JSP and HTML**
   7.1 Create an HTML table structure in JSP.
   7.2 Iterate through the `ResultSet`.
   7.3 For each record, print table rows and columns with `stud_id`, `stud_name`, `class`, `division`, and `city`.

8. **Close Resources**
   8.1 Close `ResultSet`, `Statement`, and `Connection` in a `finally` block or using appropriate clean-up logic.

9. **Deploy and Run the Application**
   9.1 Deploy the web application to Tomcat.
   9.2 Start Tomcat server.
   9.3 Open a browser and access the JSP page (e.g., `http://localhost:8080/<app-name>/displayStudents.jsp`).
   9.4 Verify that all records from `students_info` are displayed correctly.

Q&A :

1. **Q:** What is JSP?
   A: JSP (JavaServer Pages) is a server-side technology that allows embedding Java code into HTML to create dynamic web pages.

2. **Q:** How is a JSP page executed on the server?
   A: The container translates the JSP into a Servlet, compiles it, executes it, and sends the generated HTML response to the client.

3. **Q:** What is the role of JDBC in JSP-based applications?
   A: JDBC provides APIs for JSP/Java code to connect to a database, execute SQL queries, and process results.

4. **Q:** Which SQL statement is used to display table records?
   A: The `SELECT` statement (e.g., `SELECT * FROM students_info;`).

5. **Q:** Name any two implicit objects available in JSP.
   A: `request`, `response`, `out`, `session`, `application` (any two).

6. **Q:** What is the use of the `page` directive in JSP?
   A: It defines page-level settings such as imported packages, error handling, and content type (e.g., `<%@ page import="java.sql.*" %>`).

7. **Q:** Why should database connections be closed in JSP?
   A: To release resources, avoid memory leaks, and prevent exhaustion of database connections.

8. **Q:** Differentiate between Servlet and JSP in one line.
   A: Servlets are Java classes for handling requests, while JSP is a view technology focused on embedding Java into HTML for presentation.

9. **Q:** What is `ResultSet` in JDBC?
   A: `ResultSet` is an object that holds the data returned by executing an SQL `SELECT` query.

10. **Q:** Why is it recommended to separate business logic from JSP?
    A: To improve maintainability, reusability, and adherence to MVC architecture by keeping JSP mainly for presentation.

Conclusion:
By performing this experiment, I learned how to connect a JSP page to a database using JDBC and successfully display the records of the `students_info` table using an SQL `SELECT` query.

Lab Assignment No: 7

Aim:
To build a dynamic web application using PHP and MySQL that performs Create, Read, Update, and Delete (CRUD) operations with proper database connectivity.

Objectives:

1. To design and create relational database tables in MySQL for a web application.
2. To establish a PHP–MySQL connection using appropriate extensions.
3. To implement add, update, delete, and retrieve operations in a PHP-based web interface.

Software Used:

- **Operating System:** *Windows / Linux*
- **Languages:** *PHP, SQL, HTML, CSS*
- **Database:** *MySQL*
- **Web Server / Tools:** *Apache (XAMPP/WAMP/LAMP), phpMyAdmin, Web Browser*
- **Optional Frameworks/Libraries:** *Bootstrap (for UI styling, if required)*

Theory :
A *dynamic web application* generates content at runtime based on user input or stored data. **PHP** is a server-side scripting language that runs on the web server and processes HTTP requests, interacts with databases, and returns HTML output. **MySQL** is a relational database management system (RDBMS) used to store and manage structured data in tables.

The PHP–MySQL interaction follows the client–server model:

- The **client (browser)** sends an HTTP request (GET/POST).
- The **web server** invokes PHP scripts.
- **PHP** connects to **MySQL**, executes SQL queries, and fetches results.
- PHP dynamically embeds data into HTML and sends the response back to the client.

CRUD operations:

- *Create* – INSERT new records into tables.
- *Read* – SELECT and display records.
- *Update* – MODIFY existing records.
- *Delete* – REMOVE records.

Database tables typically include a *primary key* column and other fields describing the entity. Secure PHP–MySQL applications also consider input validation, SQL injection prevention, and proper error handling.

Algorithm / Procedure:

1. **Problem Definition and Design**
   1.1 Identify the entity (e.g., student, product, employee) and required fields.
   1.2 Draw a simple ER diagram and finalize table structure (with primary key).

2. **Database Creation in MySQL**
   2.1 Start MySQL server (via XAMPP/WAMP/LAMP).
   2.2 Create a database for the application.
   2.3 Create required tables with suitable data types and constraints.
   2.4 Verify table structure and insert some sample data (optional).

3. **Configure Web Server and Project Folder**
   3.1 Start Apache web server.
   3.2 Create a project directory under the server's document root (e.g., `htdocs/` or `www/`).
   3.3 Organize folders for PHP scripts, CSS, and other resources.

4. **Create Database Connection Script in PHP**
   4.1 Configure database parameters (hostname, username, password, database name).
   4.2 Use `mysqli` or `PDO` to establish a connection with MySQL.
   4.3 Implement basic error handling for connection failures.
   4.4 Include/reuse this connection script in all other PHP files.

5. **Design HTML Forms and UI Pages**
   5.1 Create an HTML form for adding new records (Create).
   5.2 Create a listing page to display all records in tabular format (Read).
   5.3 Provide Edit and Delete links/buttons for each record.
   5.4 Apply basic CSS for layout and readability.

6. **Implement Create Operation (INSERT)**
    6.1 Capture form data sent via POST.
    6.2 Validate input fields (empty check, type check, etc.).
    6.3 Construct and execute an SQL INSERT query using PHP.
    6.4 Redirect or show a message on successful insertion or error.

7. **Implement Read Operation (SELECT)**
    7.1 Connect to the database and execute an SQL SELECT query.
    7.2 Fetch all rows into a result set.
    7.3 Display records inside an HTML table.
    7.4 Provide navigation (e.g., link to Add New Record form).

8. **Implement Update Operation (UPDATE)**
    8.1 From the listing page, pass the record's primary key (e.g., ID) to an Edit page.
    8.2 Retrieve existing record data and prefill an HTML form.
    8.3 On form submission, validate inputs.
    8.4 Execute an SQL UPDATE query using the primary key.
    8.5 Display success or error message and return to listing page.

9. **Implement Delete Operation (DELETE)**
    9.1 From the listing page, capture the primary key of the record to delete.
    9.2 Optionally display a confirmation prompt (client-side or server-side).
    9.3 Execute an SQL DELETE query on the specified record.
    9.4 Refresh listing and confirm deletion to the user.

10. **Testing and Validation**
    10.1 Test all CRUD operations with valid and invalid inputs.
    10.2 Check for connection errors and SQL errors.
    10.3 Ensure proper navigation between pages and clear user messages.
    10.4 Verify that database changes reflect accurately after each operation.

11. **Documentation**
    11.1 Record database schema, page flow, and main functions.
    11.2 Note any limitations and possible enhancements (e.g., authentication).

Q&A :

1. **Q:** What is a dynamic web application?
   A: A web application where content and responses are generated at runtime based on user input, database data, or server-side logic.

2. **Q:** Why is MySQL commonly used with PHP?
   A: Because MySQL is an open-source, lightweight RDBMS with strong support in PHP through built-in extensions like `mysqli` and `PDO`.

3. **Q:** Define CRUD in the context of databases.
   A: CRUD stands for Create, Read, Update, and Delete—the four basic operations performed on persistent data in a database.

4. **Q:** What is the role of a primary key in a table?
   A: A primary key uniquely identifies each record in a table and prevents duplicate and NULL values in that column.

5. **Q:** Differentiate between GET and POST methods in PHP forms.
   A: GET appends data to the URL and is suitable for non-sensitive data; POST sends data in the request body and is used for sensitive or larger amounts of data.

6. **Q:** What is the purpose of the database connection script in PHP?
   A: To establish a reusable connection between PHP scripts and the MySQL database, enabling execution of SQL queries.

7. **Q:** Why is input validation important in PHP–MySQL applications?
   A: To prevent invalid data, enhance reliability, and protect against security issues like SQL injection and XSS.

8. **Q:** What is SQL injection?
   A: A security attack where malicious SQL code is inserted into input fields to manipulate or access the database unauthorizedly.

9. **Q:** Which SQL commands are used for CRUD in MySQL?
   A: INSERT for Create, SELECT for Read, UPDATE for Update, and DELETE for Delete.

10. **Q:** What is the advantage of separating HTML and PHP logic?
    A: It improves code readability, maintainability, and enables easier design–logic separation for future enhancements.

Conclusion:
The PHP–MySQL dynamic web application was successfully developed to perform all CRUD operations. The experiment demonstrated practical integration of server-side scripting with a relational database for data-driven web development.

Lab Assignment No: 8

Aim:

To design a login page using Struts framework with validations for name, mobile number, email ID, and empty fields, including redisplay of invalid values and a welcome page on successful login.

Objectives:

1. To understand MVC-based web application development using Struts.
2. To implement server-side form validation for name, mobile number, and email.
3. To configure Struts validation and navigation for error and success pages.

Software Used:

- **Operating System:** Windows / Linux
- **Programming Language:** Java, JSP
- **Web Server:** Apache Tomcat
- **Framework:** Apache Struts
- **Tools/IDE:** Eclipse / NetBeans / IntelliJ IDEA
- **Browser:** Any modern web browser

Theory :

Struts is a Java-based MVC (Model-View-Controller) web framework.

- **Model** represents business logic and data.
- **View** is implemented using JSP pages.
- **Controller** is implemented using Struts controller components like `ActionServlet` and `Action` classes.

In Struts, user input is captured in form beans (e.g., `ActionForm` in Struts 1 or properties in Struts 2 `Action` classes). The framework supports validation using a **validation framework** or programmatic validation methods.

- Validation rules (like required fields, email format, numeric checks) are typically configured in XML files or annotations.
- On validation failure, control is returned to the input JSP with error messages.
- On success, control is forwarded to a success or welcome page.

For a login form:

- **Name**: checked for non-empty and valid alphabetic pattern.
- **Mobile number**: checked for non-empty, numeric, and length.
- **Email ID**: checked for non-empty and valid email pattern.
- **Empty fields**: checked using “required” validation.
  Struts automatically redisplays the form with previous values and error messages when validation fails.

Algorithm / Procedure:

1. **Create Struts Web Project**
   - Set up a Java web project.
   - Add Struts framework libraries to the project.
   - Configure web deployment descriptor to use Struts (e.g., map the Struts front controller).
2. **Configure Struts Framework**
   - Create Struts configuration file (e.g., `struts-config.xml` or `struts.xml`).
   - Define form bean or action class for the login form.
   - Map the login action with input JSP and success JSP.
3. **Design Login JSP Page**
   - Create a JSP page containing fields: *name, mobile number, email ID* and a *login* button.
   - Use Struts tags for form and input fields to bind them with the form bean or action properties.

    - Add placeholders/areas to display validation error messages.
4. **Create Form Bean / Action Class**
    - Define properties for name, mobile number, and email ID.
    - Provide getters and setters for these properties.
    - Implement validation logic: either using Struts validation framework configuration or overriding validation methods to check:
        - Correct name pattern.
        - Correct mobile number format and length.
        - Correct email pattern.
        - Non-empty fields.
5. **Configure Validation Rules**
    - Define validation rules in the validation configuration (XML or annotations):
        - Mark all fields as *required*.
        - Add format checks for name, mobile, and email.
    - Associate validation rules with the login form/action.
6. **Define Navigation / Forwards**
    - In the Struts configuration, define:
        - An input page (login JSP) for validation failure.
        - A success page (congratulations and welcome JSP) for valid input.
    - Map these as forwards/results in the configuration.
7. **Create Success (Welcome) JSP Page**
    - Design a JSP page displaying a congratulatory message and a welcome text.
    - Optionally, show the user's name.
8. **Deploy and Run Application**
    - Deploy the application on Apache Tomcat.
    - Access the login URL through a browser.
    - Test the following scenarios: empty fields, invalid name/mobile/email, and all correct entries.
    - Verify that invalid input redisplays the form with messages and valid input goes to the welcome page.

Q&A

1. **Q:** What is Struts?
   A: Struts is a Java-based MVC framework for developing web applications using a model-view-controller architecture.

2. **Q:** Which design pattern does Struts follow?
   A: Struts follows the MVC (Model-View-Controller) design pattern.

3. **Q:** What is the role of the Controller in Struts?
   A: The controller intercepts requests, invokes appropriate actions, manages navigation, and coordinates between model and view.

4. **Q:** Why is validation required in web forms?
   A: Validation ensures that user input is correct, complete, and in the proper format before processing, improving reliability and security.

5. **Q:** How are error messages typically displayed in Struts applications?
   A: Error messages are stored in an error collection and displayed on JSP pages using Struts tag libraries for messages.

6. **Q:** What kind of validation is used to ensure that a field is not left blank?
   A: Required field validation is used to check that a value is provided.

7. **Q:** How is email validation generally implemented in Struts?
   A: Email validation is implemented using a predefined email rule or a regular expression in the validation configuration.

8. **Q:** What checks are needed for mobile number validation?
   A: Mobile number validation usually checks for non-empty value, numeric characters only, and a fixed or allowed length.

9. **Q:** What happens if validation fails in a Struts form?
   A: Control returns to the input JSP, the original values are redisplayed, and corresponding error messages are shown.

10. **Q:** What is the purpose of a success or welcome page in a login application?
    A: It confirms successful validation and login, and usually serves as the starting page for authorized user activities.

Conclusion:

The experiment successfully demonstrated creating a Struts-based login page with server-side validations for name, mobile number, email ID, and empty fields, along with proper error redisplay and a welcome page on successful input.

Lab Assignment No: 9

Aim:

To design a simple web application using **AngularJS** that provides:

- *Registration page* (first name, last name, username, password)
- *Login page* using the registered credentials.

Objectives:

1. To understand the basic architecture and components of **AngularJS** (module, controller, view).
2. To implement *two-way data binding* and *form validation* in AngularJS.
3. To design simple *registration* and *login* functionality using AngularJS.

Software Used:

- **Operating System:** Windows / Linux
- **Languages:** HTML, CSS, JavaScript
- **Framework:** AngularJS (1.x)
- **Web Browser:** Chrome / Firefox / Edge
- **Editor/IDE:** VS Code / Sublime Text / Notepad++

Theory :

AngularJS is a client-side JavaScript framework used to build dynamic single-page applications (SPAs). It follows the *Model–View–Controller (MVC)* or *Model–View–ViewModel (MVVM)* pattern.

Key Concepts:

- **Module:** Logical container for different parts of an app (controllers, services, etc.), created using `angular.module()`.

- **Controller:** JavaScript function that controls data and behavior of the view; attached using `ng-controller`.
- **Data Binding:** *Two-way data binding* automatically synchronizes data between model (JavaScript objects) and view (HTML).
- **Directives:** Special HTML attributes prefixed with `ng-` (e.g., `ng-model`, `ng-repeat`, `ng-show`) used to extend HTML behavior.
- **Expressions:** Written inside `{{ }}`, used to display values, perform simple operations, and bind data to the view.
- **Form Validation:** AngularJS provides built-in validation through directives like `ng-required`, `ng-minlength`, and uses form and input states (`$valid`, `$invalid`, `$pristine`, `$dirty`).

In a *registration and login* application:

- Registration form captures user details and stores them (temporarily in scope/local storage for basic demo).
- Login form checks user-entered username and password against stored data and then displays *success* or *failure* messages.

Algorithm / Procedure:

1. **Problem Definition**
    - Identify required fields for registration: *first name, last name, username, password*.
    - Identify required fields for login: *username, password*.
2. **Create Project Structure**
    - Create a project folder (e.g., `AngularRegistrationLogin`).
    - Create required files: `index.html`, `app.js`, and optional `style.css`.
3. **Include AngularJS Library**
    - Add AngularJS (CDN or local) in `index.html` using a `<script>` tag.
    - Add `ng-app` directive in the root HTML element to initialize the AngularJS application.
4. **Define AngularJS Module**

   - In `app.js`, create an AngularJS module using `angular.module('appName', [])`.

5. **Create Controller**

   - Define a controller (e.g., `MainController`) within the module.
   - Inject `$scope` and create scope variables for:
     - Registration fields: `firstName`, `lastName`, `regUsername`, `regPassword`.
     - Login fields: `loginUsername`, `loginPassword`.
     - Status messages (e.g., `registerMessage`, `loginMessage`).

6. **Implement Registration Logic**

   - In the controller, create a function (e.g., `register()`).
   - Store registration details in scope variables or a user object/array.
   - Set a success message if registration fields pass basic validation.

7. **Implement Login Logic**

   - In the controller, create a function (e.g., `login()`).
   - Compare `loginUsername` and `loginPassword` with stored registration credentials.
   - If they match, set a *login success* message; otherwise set an *error* message.

8. **Design HTML Views**

   - Create a *registration form* with input fields bound via `ng-model` to registration scope variables.
   - Create a *login form* with input fields bound via `ng-model` to login scope variables.
   - Use AngularJS directives for validation:
     - `ng-required="true"` on mandatory fields.
     - Optionally use `ng-minlength`, `ng-pattern` for simple constraints.
   - Display validation error messages using form and field state (`formName.inputName.$invalid`).

9. **Bind Functions to UI**

   - Attach `ng-submit` or `ng-click` to call `register()` on registration submit.
   - Attach `ng-submit` or `ng-click` to call `login()` on login submit.
   - Display success/error messages using Angular expressions.

10. **Testing**

    - Run the page in a browser.
    - Test registration with valid and invalid inputs (check validation behavior).
    - Test login with correct and incorrect credentials.
    - Verify that appropriate messages are displayed and that binding works correctly.

Q&A

1. **Q:** What is AngularJS?
   A: AngularJS is a client-side JavaScript framework for building dynamic, single-page web applications using MVC/MVVM patterns.

2. **Q:** What is an AngularJS module?
   A: A module is a container for different parts of an AngularJS app (controllers, services, directives, etc.) created using `angular.module()`.

3. **Q:** Define a controller in AngularJS.
   A: A controller is a JavaScript function that manages application data and logic for a view, and it is associated with the view using the `ng-controller` directive.

4. **Q:** What is two-way data binding?
   A: Two-way data binding means changes in the view automatically update the model and changes in the model automatically update the view.

5. **Q:** What are AngularJS directives?
   A: Directives are special markers (attributes, elements) such as `ng-model`, `ng-repeat`, `ng-bind` that extend HTML with custom behavior.

6. **Q:** What is the purpose of `ng-model`?
   A: `ng-model` binds the value of HTML controls (input, select, textarea) to application data (scope variables).

7. **Q:** How does AngularJS help in form validation?
   A: AngularJS provides validation directives like `ng-required`, `ng-minlength`, and exposes form/input states (`$valid`, `$invalid`) to control error messages and styles.

8. **Q:** What is `ng-submit` used for?
   A: `ng-submit` specifies the AngularJS function to call when a form is submitted.

9. **Q:** How can you display messages in AngularJS views?
   A: By binding scope variables to the view using expressions like `{{message}}` or directives such as `ng-bind`.

10. **Q:** Why is AngularJS suitable for a registration and login page?
    A: Because it supports two-way data binding, built-in form validation, and clear separation of view and logic, simplifying interactive form-based interfaces.

Conclusion:

The registration and login application using AngularJS was successfully designed. The experiment demonstrated how to use AngularJS modules, controllers, data binding, and form validation to build a simple dynamic web interface.

Lab Assignment No: 10

Aim:
To design and implement a business interface with required business logic for a web application using Enterprise JavaBeans (EJB), e.g., deposit and withdraw amount transactions.

Objectives:

1. To understand the design of EJB-based business interfaces for web applications.
2. To implement core business logic (deposit, withdraw, balance) using session beans.
3. To integrate the EJB component with a web client for transaction processing.

Software Used:

- Operating System: Windows / Linux
- Programming Language: Java
- Application Server: GlassFish / WildFly / JBoss
- IDE/Tools: NetBeans / Eclipse, JDK
- Database: MySQL / PostgreSQL (via JDBC)
- Framework/Technology: Enterprise JavaBeans (Session Beans), Servlet/JSP (as web client)

Theory :
Enterprise JavaBeans (EJB) is a server-side component architecture for modular construction of enterprise applications in Java EE. It separates business logic from presentation and data layers.
A *business interface* in EJB (local or remote) defines the business methods that clients can invoke. A *session bean* implements this interface and encapsulates business rules, transaction management, and security.
For banking-like operations (deposit, withdraw), a stateful or stateless session bean can be used. The bean methods typically retrieve/update account data via DAO/JPA, with the container managing transactions, security, concurrency, and lifecycle. Annotations or deployment descriptors configure the EJB.

Algorithm / Procedure:

1. **Problem Analysis**
   1.1. Identify entities and operations: Account, deposit, withdraw, checkBalance.
   1.2. Define input/output for each operation (account number, amount, status, new balance).

2. **Environment Setup**
 2.1. Install and configure JDK and Java EE–compliant application server (GlassFish/WildFly/JBoss).
 2.2. Configure IDE (NetBeans/Eclipse) with server integration.
 2.3. Set up database schema and tables for accounts with required fields (e.g., account_no, name, balance).

3. **Database Layer Design**
 3.1. Create the database and account table.
 3.2. Configure JDBC connection pool and data source on the application server.
 3.3. Optionally design DAO/JPA entity classes for Account.

4. **EJB Business Interface Design**
 4.1. Define a local/remote business interface (e.g., `AccountService`) with methods:
 - `deposit(accountId, amount)`
 - `withdraw(accountId, amount)`
 - `getBalance(accountId)`
 4.2. Specify method parameters and return types as per requirements.

5. **Session Bean Implementation**
 5.1. Create a session bean class implementing the business interface.
 5.2. Inject data source / entity manager for DB access.
 5.3. Implement `deposit` method:
 - Validate account and amount.
 - Fetch current balance.
 - Add amount and update balance in DB.
 5.4. Implement `withdraw` method:
 - Validate account and amount.
 - Check sufficient balance.
 - Subtract amount and update balance in DB.
 5.5. Implement `getBalance` method:
 - Fetch and return current balance.
 5.6. Configure transaction attributes (e.g., container-managed, required) via annotations/descriptors.

6. **Web Client Layer Design**
 6.1. Design JSP/HTML forms for deposit, withdraw, and balance enquiry (fields: accountId, amount).
 6.2. Implement a Servlet/JSP controller to:

- Read form inputs.
- Look up the EJB via JNDI or injection.
- Invoke the corresponding business methods.
- Forward outcomes (success/failure, balance) to view pages.

7. **Deployment**
   7.1. Package EJB and web module in an EAR/WAR as required.
   7.2. Deploy the application on the configured application server.
   7.3. Verify JNDI bindings and data source configuration.

8. **Testing and Validation**
   8.1. Test deposit with valid account and positive amount.
   8.2. Test withdraw with sufficient and insufficient balance conditions.
   8.3. Validate correctness of updated balances in the database.
   8.4. Check error handling and message display for invalid inputs.

Q&A :

1. **Q:** What is Enterprise JavaBeans (EJB)?
   A: EJB is a server-side component model in Java EE used to build scalable, transactional, and secure enterprise applications by encapsulating business logic in reusable components.

2. **Q:** What is a business interface in EJB?
   A: A business interface is a Java interface (local or remote) that declares the business methods of a session bean which clients can invoke.

3. **Q:** Differentiate between local and remote business interfaces.
   A: A local interface is used for same-JVM access with better performance, while a remote interface supports distributed access across different JVMs or machines.

4. **Q:** Why is EJB suitable for banking operations like deposit and withdraw?
   A: Because EJB provides built-in transaction management, security, concurrency control, and scalability, which are essential for financial operations.

5. **Q:** What is a session bean?
   A: A session bean is an EJB component that implements business logic and interacts with clients on a per-session or per-request basis.

6. **Q:** Name the main types of session beans.
   A: Stateless session beans, stateful session beans, and singleton session beans.

7. **Q:** What is container-managed transaction (CMT) in EJB?
   A: In CMT, the EJB container automatically begins, commits, or rolls back transactions based on configuration, without explicit transaction code in the bean.

8. **Q:** How does an EJB client access a session bean?
   A: The client obtains a reference via dependency injection or JNDI lookup and then calls the methods defined in the bean's business interface.

9. **Q:** What is the role of annotations in EJB?
   A: Annotations are used to declare bean type, business interfaces, transaction attributes, security roles, and other metadata directly in the source code.

10. **Q:** How is business logic separated from presentation in an EJB-based web application?
    A: Business logic resides in EJBs (server-side components), while presentation is handled by Servlets/JSP or other UI technologies that invoke EJB methods.

Conclusion:
The business interface and corresponding EJB session bean for deposit and withdraw operations were successfully designed and implemented, demonstrating separation of business logic and secure, transactional processing in a web application.

Lab Assignment No: 11

Aim:
To design and implement a dynamic web application for a selected business functionality using appropriate web technologies learned in previous assignments.

Objectives:

1. To analyze business requirements and design a multi-tier web application.
2. To implement client-side and server-side components with database connectivity.
3. To deploy, test, and validate a complete dynamic web application.

Software Used:

- **Operating System:** Windows / Linux
- **Languages:** HTML, CSS, JavaScript, PHP / Java (Servlet/JSP), SQL
- **Databases:** MySQL / any relational DBMS
- **Web Server / Container:** Apache / Tomcat / XAMPP / WAMP
- **Tools / IDEs:** VS Code / Eclipse / NetBeans / IntelliJ
- **Frameworks (if used):** AngularJS / Struts / EJB / Bootstrap (optional)

Theory:
A **dynamic web application** generates content at runtime based on user input, requests, and database data. It typically follows a **three-tier architecture**:

- **Presentation layer:** HTML, CSS, JavaScript for user interface.
- **Business logic layer:** Server-side technologies (PHP, Servlet, JSP, EJB, Struts) implementing application rules.
- **Data layer:** Database (e.g., MySQL) for persistent storage using SQL queries (CRUD operations).

MVC (Model–View–Controller) architecture separates concerns:

- **Model:** Data and business rules.

- **View:** User interface pages.
- **Controller:** Request handling and navigation.

Key aspects include **session management**, **form validation**, **input sanitization**, **security**, and **error handling** to ensure reliable and secure web applications.

Algorithm / Procedure:

1. **Problem Definition and Planning**
   1.1 Select a business domain (e.g., online shop, library, student management, banking).
   1.2 Define clear functional requirements (modules, users, operations).
   1.3 Identify non-functional requirements (performance, security, usability).

2. **System Analysis and Design**
   2.1 Draw context diagram and use case diagrams (optional).
   2.2 Design ER diagram and relational schema for the database.
   2.3 Normalize tables and define primary/foreign keys.
   2.4 Design navigation structure and page flow (site map, wireframes).
   2.5 Decide technology stack and architecture (e.g., MVC, 3-tier).

3. **Database Design and Creation**
   3.1 Create database and tables using SQL (CREATE statements).
   3.2 Define constraints (NOT NULL, UNIQUE, FK, CHECK if needed).
   3.3 Insert initial test data for validation and testing.

4. **Client-Side Design**
   4.1 Design HTML pages for forms, lists, dashboards, and reports.
   4.2 Apply CSS for layout, typography, and responsiveness.
   4.3 Implement JavaScript validation (required fields, email, numbers, etc.).
   4.4 Use alerts/prompts/DOM manipulation where appropriate.

5. **Server-Side Implementation**
   5.1 Configure web server/application server (Apache/Tomcat/XAMPP).
   5.2 Implement business logic using PHP / Servlet/JSP / other chosen tech.
   5.3 Establish database connection using appropriate APIs/driver.
   5.4 Implement CRUD operations (Create, Read, Update, Delete).
   5.5 Implement authentication (login/logout) and authorization if required.
   5.6 Manage sessions and cookies for user state.

6. **Integration of Modules**
   6.1 Link client-side forms to server-side scripts/pages.
   6.2 Ensure proper request/response flow and redirections.
   6.3 Integrate validations at both client and server side.
   6.4 Implement error messages and success notifications.

7. **Testing and Validation**
   7.1 Perform unit testing for each module.
   7.2 Conduct integration testing for workflows (login → operation → logout).
   7.3 Test for invalid inputs, boundary values, and security-related inputs.
   7.4 Validate form fields, constraints, and database integrity.

8. **Deployment and Documentation**
   8.1 Deploy application on local server or institutional server.
   8.2 Create user manual (steps for usage and sample inputs/outputs).
   8.3 Prepare technical documentation (architecture, DB schema, tech stack).
   8.4 Take screenshots and record test cases and results.

9. **Review and Improvements**
   9.1 Collect feedback from users/peers.
   9.2 Optimize queries and code where possible.
   9.3 Enhance UI/UX and handle additional edge cases.

Q&A :

1. **Q:** What is a dynamic web application?
   A: A web application where content and responses are generated at runtime based on user input, server logic, and database data.

2. **Q:** Define three-tier architecture.
   A: It is an architecture with presentation layer (UI), business logic layer (application logic), and data layer (database), each separated into distinct tiers.

3. **Q:** What does CRUD stand for?
   A: CRUD stands for Create, Read, Update, and Delete operations on data.

4. **Q:** Why is MVC architecture used in web applications?
   A: MVC separates data (Model), UI (View), and control logic (Controller), improving

modularity, maintainability, and testability.

5. **Q:** What is the role of the server-side script in a dynamic website?
   A: It processes client requests, applies business rules, accesses the database, and generates dynamic responses (HTML/JSON/XML).

6. **Q:** Mention two advantages of using a database in a web application.
   A: It provides persistent data storage and supports efficient querying, integrity constraints, and concurrent access.

7. **Q:** What is session management?
   A: Session management is the mechanism to maintain user-specific state across multiple HTTP requests, typically using session IDs or cookies.

8. **Q:** Why is input validation important in web applications?
   A: It prevents invalid data entry, reduces errors, and protects against attacks such as SQL injection and XSS.

9. **Q:** Differentiate between client-side and server-side validation.
   A: Client-side validation runs in the browser (e.g., JavaScript) for quick feedback; server-side validation runs on the server and is more secure and reliable.

10. **Q:** What is deployment in the context of a web project?
    A: Deployment is the process of configuring and publishing the web application on a server so that end users can access it via a URL.

Conclusion:
The mini project helped in applying complete web development concepts to design, implement, and test a real-time dynamic web application with proper integration of client, server, and database tiers.

LP-II
AI Assignment 1

TITLE: Execution of BFS and DFS Procedures. Utilize an unweighted graph to create a self-referential (recursive) method for traversing every node within a graph or tree-based structure.

GOAL: The purpose of this laboratory exercise is to design and code BFS and DFS search methodologies using a programming language.

LEARNING OBJECTIVES: To achieve the primary goal, the following targets have been set:

1. To *analyze* the **Breadth-First Search (BFS)** mechanism.
2. To *investigate* the **Depth-First Search (DFS)** mechanism.
3. To *translate* algorithmic theory into functional source code.

1. Breadth-First Search:

- Breadth-first search is a prevalent technique used for navigating through tree or graph architectures. This method prioritizes horizontal exploration, which is why it is termed "*breadth-first*."1
- The BFS procedure commences at the origin (root) and explores every neighbor at the current depth before progressing to nodes at the subsequent distance level.
- This strategy serves as a fundamental model for general-purpose graph exploration.
- BFS is typically managed using a **First-In-First-Out (FIFO)** queue structure.

Example:
In a standard tree, BFS moves from the starting node S toward a target K. The algorithm proceeds layer by layer, following a horizontal sequence. The resulting traversal sequence would be:
S → A → B → C → D → G → H → E → F → I → K

- **Time Efficiency:** The temporal complexity of BFS is determined by the total nodes visited until the shallowest solution is reached. If *b* represents the branching factor and *d* represents the depth of the solution:
  $T(b)=1+b^2+b^3+...+b^d=O(b^d)$
- **Memory Usage:** The spatial complexity depends on the size of the frontier (the nodes currently stored), expressed as $O(b^d)$
- **Reliability (Completeness):** BFS is considered complete; provided the target exists at a finite depth, the algorithm is guaranteed to locate it.
- **Effectiveness (Optimality):** BFS provides the optimal path if the cost of moving between nodes is a non-decreasing function of depth.

Algorithm:

1. Select a starting node, visit its neighboring unvisited points, flag them as visited, output the data, and add them to a queue.
2. If no unvisited neighbors remain for the current node, remove the front item from the queue.
3. Continue the first and second steps until the queue becomes empty or the target is identified.

Program:

```
# Adjacency list representation

network = {

    'A' : ['B','C'],

    'B' : ['D', 'E'],

    'C' : ['F'],

    'D' : [],

    'E' : ['F'],

    'F' : []

}


discovered = [] # Tracking list for visited nodes

line_queue = [] # Initialize the FIFO queue


def breadth_first_search(discovered, network, start_node):

    discovered.append(start_node)

    line_queue.append(start_node)


    while line_queue:

        current = line_queue.pop(0)

        print(current, end=" ")
        for child in network[current]:

            if child not in discovered:

                discovered.append(child)

                line_queue.append(child)


# Execution

print("Path generated via Breadth-First Search:")

breadth_first_search(discovered, network, 'A')
```

Output:

```
Path generated via Breadth-First Search:
A B C D E F
```

---

2. Depth-First Search:

- Depth-first search is a recursive-style technique used for navigating tree or graph data frameworks.
- It is named "depth-first" because it starts at the source and dives down a single branch to its maximum depth before pivoting to the next available branch.
- DFS is traditionally implemented using a Last-In-First-Out (LIFO) stack structure or via recursion.
- The logic flow of DFS mirrors BFS but changes the order of node prioritization.

Example:
In a search tree, DFS follows a specific vertical order: Root → Left Child → Right Child.
Starting at S, it explores A, then B, then D and E. Once it hits a dead end at E, it backtracks because no further successors exist. It then moves to C and eventually G, terminating once the goal is reached.

- **Reliability (Completeness):** In a finite state space, DFS is complete because it eventually visits every branch within the bounded tree.
- **Time Efficiency:** The time taken is proportional to the nodes explored. It is calculated as:
  $T(n)=1+n^2+n^3+...+n^m=O(n>m)$
  where *m* represents the maximum depth, which can be significantly larger than the shallowest solution depth *d*.
- **Memory Usage:** DFS only needs to keep track of a single path from the root to the current leaf. Thus, the space complexity is O(bm)O(bm), where *b* is the branching factor.

- **Effectiveness (Optimality):** DFS is generally non-optimal, as it might find a much longer path to the goal before exploring shorter alternatives.

Algorithm:

1. Place a starting vertex from the graph onto a stack.
2. Retrieve the top item from the stack and include it in the "visited" collection.
3. Identify all adjacent nodes of that vertex. Push the neighbors that have not been visited yet onto the stack.
4. Iterate through steps 2 and 3 until the stack is entirely empty.

Program:

```
# Adjacency list using a dictionary

network = {

    'A' : ['B','C'],

    'B' : ['D', 'E'],

    'C' : ['F'],

    'D' : [],

    'E' : ['F'],

    'F' : []

}


seen = set() # Set to track visited nodes



def depth_first_search(seen, network, current_node): # Recursive DFS
function

    if current_node not in seen:

        print(current_node)

        seen.add(current_node)

        for neighbor in network[current_node]:
            depth_first_search(seen, network, neighbor)


# Execution

print("Path generated via Depth-First Search:")

depth_first_search(seen, network, 'A')
```

Output:

```
Path generated via Depth-First Search:
A
B
D
E
F
C
```

Q&A

1. **Q:** What is the goal of this practical?
   A: To execute and understand Breadth-First Search (BFS) and Depth-First Search (DFS) traversal procedures using an unweighted graph.

2. **Q:** What are the learning objectives of this assignment?
   A: To study BFS, analyze DFS, and convert theoretical algorithms into executable code.

3. **Q:** What does BFS stand for and what does it do?
   A: BFS stands for Breadth-First Search and explores graph nodes level-by-level in a horizontal manner.

4. **Q:** Which data structure is commonly used for BFS implementation?
   A: A FIFO Queue data structure is typically used for BFS.

5. **Q:** What does DFS stand for and how does it operate?
   A: DFS stands for Depth-First Search and traverses by exploring one branch deeply before backtracking.

6. **Q:** Which data structure is used for implementing DFS?
   A: A LIFO Stack or recursion mechanism is used for DFS.

7. **Q:** Which traversal guarantees optimality in uniform edge cost situations?
   A: BFS provides optimal paths when edge costs are uniform.

8. **Q:** Is DFS guaranteed to find the shallowest solution first?
   A: No, DFS may locate deeper solutions before shallower ones, making it non-optimal.

9. **Q:** What is the time complexity of BFS in terms of branching factor bbb and depth ddd?
   A: The time complexity is O(bd)O(b^d)O(bd).

10. **Q:** Why is this practical important?
    A: It develops understanding of fundamental search algorithms widely used in AI, graph theory, and problem-solving applications.

CONCLUSION: Through this practical, we have explored the mechanics of BFS and DFS algorithms in depth and successfully implemented them using recursive and iterative programming logic.

AI Assignment 2

TITLE: Using the A* Algorithm for Game Pathfinding

AIM: To learn and apply the logic of the A* algorithm to solve movement problems in games.

OBJECTIVES:

1. To understand how the A* algorithm works.
2. To explore different types of game search challenges.
3. To evaluate how effectively A* finds paths in various game scenarios.

MOTIVATION: In the real world and in video games, we need to find the quickest way to get from point A to point B while avoiding obstacles like walls, water, or mountains. This algorithm helps solve that problem efficiently.

INTRODUCTION

*The A Search Algorithm:**
A* (A-Star) is a popular and powerful tool used for pathfinding. Unlike basic search methods that move blindly, A* is considered "smart" because it uses logic to choose the best direction. It is widely used in web maps and video games to find the shortest route between two points.

Imagine a grid with a starting point, a goal, and several blocks in between. A* finds the best path by calculating a score (**f**) for every potential step. This score is made of two parts:

- **g:** The actual cost (distance) moved from the starting point to the current spot.
- **h (Heuristic):** An educated guess of the distance remaining to reach the goal. We don't know the exact distance yet because of obstacles, so we estimate it.

The Formula: `f = g + h`
The algorithm always chooses the path with the lowest **f** score to ensure it stays on the most efficient track.

ALGORITHM

We maintain two lists: **Open** (spots to be checked) and **Closed** (spots already checked).

1. Place the starting point in the **Open** list.

2. While there are spots in the **Open** list:
   a) Pick the spot with the lowest **f** value and call it "q."
   b) Remove "q" from the **Open** list.
   c) Look at all 8 neighboring spots around "q."
   d) For each neighbor:
      - If the neighbor is the **Goal**, stop; the path is found!
      - Calculate **g, h, and f** for that neighbor.
      - If a better (shorter) path to this spot already exists in the Open or Closed lists, ignore this neighbor.
      - Otherwise, add this neighbor to the **Open** list.
   e) Put "q" into the **Closed** list so we don't check it again.

HEURISTICS (ESTIMATING DISTANCE)

How do we calculate the "guess" (**h**)? We can either be 100% exact (which is slow) or use an approximation (which is fast).

Common Ways to Estimate:

1. Manhattan Distance

- **How:** Add the horizontal and vertical distance.
- **Use when:** You can only move in four directions (Up, Down, Left, Right).
- *Formula:* `h = |current.x − goal.x| + |current.y − goal.y|`

2. Diagonal Distance

- **How:** Calculates the cost when you can move diagonally.
- **Use when:** You can move in eight directions (like a King in Chess).

3. Euclidean Distance

- **How:** A straight-line "as the crow flies" distance using the Pythagorean theorem.
- **Use when:** You can move in any direction at any angle.
- *Formula:* `h = sqrt((current.x − goal.x)² + (current.y − goal.y)²)`

EXECUTION:

To make the code run fast, we usually use specific data structures:

- **Set or Priority Queue:** To quickly find the spot with the lowest "f" score.

- **Hash Table:** To keep track of the "Closed" list efficiently.
  While it works similarly to Dijkstra's algorithm, A* is faster for specific goals because the "heuristic" (the guess) points it in the right direction.

Q&A

Q1. Which search algorithm explores nodes level-by-level?
A: Breadth-First Search (BFS)

Q2. Which search algorithm explores as deep as possible before backtracking?
A: Depth-First Search (DFS)

Q3. Which data structure is commonly used for BFS?
A: FIFO Queue

Q4. Which data structure is commonly used for DFS?
A: LIFO Stack or Recursion

Q5. Which search guarantees the shortest path in unweighted graphs?
A: BFS

Q6. Which search may find longer paths and is generally non-optimal?
A: DFS

Q7. Which search uses more memory due to storing frontier nodes?
A: BFS

Q8. Which search requires less memory because it stores only a single path?
A: DFS

Q9. What is the time complexity of BFS in terms of branching factor and depth?
A: O(b^d)

Q10. Which search is complete for finite graphs?
A: Both BFS and DFS

CONCLUSION: We have explored the theory behind the A* algorithm, understood how it calculates the best path using heuristics, and discussed how to apply it to game development.

AI Assignment 3

Aim:
To study and implement Greedy search algorithms for solving classical optimization problems such as sorting, minimum spanning tree, shortest path, and job scheduling.

Objectives

1. To understand the Greedy strategy and its working principles.
2. To analyze different Greedy-based algorithms used in optimization problems.
3. To apply Greedy techniques to real-world computational problems.

Software Used

- **Operating System:** Windows / Linux
- **Language/Tools:** Python / C / C++ / Java, Any standard IDE

Theory (related to the topic)

A *Greedy algorithm* is a problem-solving approach that builds a solution step by step by selecting the locally optimal choice at each stage, without reconsidering previous decisions. The aim is to reach a globally optimal solution through a sequence of locally optimal choices. Greedy algorithms are efficient and simple to implement, but they are not universally applicable. They work correctly only when the problem satisfies two fundamental properties:

1. **Greedy-choice property:**
   A global optimum can be reached by choosing a local optimum at each step.

2. **Optimal substructure:**
   An optimal solution to the problem contains optimal solutions to its subproblems.

Greedy algorithms are widely used in areas such as network design, routing, scheduling, and resource allocation. Examples include *Selection Sort* (selecting minimum element repeatedly), *Minimum Spanning Tree* algorithms like *Prim's* and *Kruskal's* (choosing minimum cost edges without forming cycles), *Dijkstra's algorithm* (selecting the nearest unvisited node), and *Job Scheduling* (selecting jobs based on maximum profit within deadlines). These algorithms significantly reduce computational complexity compared to exhaustive search methods.

Algorithm

General Greedy Algorithm Framework:

1. Define the objective function and greedy selection criterion.
2. Initialize the solution set as empty.
3. While the solution is not complete:
   a. Select the best available candidate according to greedy rule.
   b. Check feasibility of the selected candidate.
   c. Include the candidate in the solution set.
4. Repeat until termination condition is satisfied.
5. Output the final optimized solution.

Applications of Greedy Algorithm

- **Selection Sort:** Repeatedly selects the smallest element and places it in the correct position.
- **Minimum Spanning Tree:** Repeatedly selects the minimum weight edge without forming a cycle.
- **Single Source Shortest Path:** Selects the nearest unvisited vertex and relaxes its edges.
- **Job Scheduling:** Selects jobs based on highest profit under deadline constraints.

Q&A

Q1. Which property ensures that a Greedy algorithm always makes a correct local choice?
A: Greedy-choice property

Q2. Which problem characteristic is required along with greedy-choice property?
A: Optimal substructure

Q3. Which algorithm is used to find the shortest path from a single source using Greedy approach?
A: Dijkstra’s algorithm

Q4. Which algorithms are commonly used for Minimum Spanning Tree?
A: Prim’s and Kruskal’s algorithms

Q5. Which sorting algorithm is based on Greedy selection of minimum element?
A: Selection Sort

Q6. Which data structure is commonly used to implement Dijkstra’s algorithm efficiently?
A: Priority queue

Q7. Which algorithm avoids cycle formation while selecting edges?
A: Kruskal’s algorithm

Q8. Which approach is used in Job Scheduling to maximize profit?
A: Greedy selection based on profit and deadline

Q9. Which type of problems are best suited for Greedy algorithms?
A: Optimization problems

Conclusion

Thus, Greedy algorithms were successfully studied and applied to classical optimization problems, demonstrating efficient solution construction using locally optimal decisions.

AI Assignment 4

Aim:
To implement a solution for a Constraint Satisfaction Problem (CSP) using Branch and Bound and Backtracking for the N-Queens problem.

Objectives:

- To understand the fundamental concepts of Constraint Satisfaction Problems (CSP).
- To apply Backtracking and Branch and Bound strategies to solve combinatorial problems.
- To analyze the state-space tree for the safe placement of N queens on an N×N chessboard.

Software Used:

- **Operating System:** Windows / Linux
- **Programming Language:** Python / C++ / Java

Theory:

A Constraint Satisfaction Problem (CSP) is a mathematical question defined as a set of objects whose state must satisfy a number of constraints or limitations. The N-Queens problem is a classic CSP where the goal is to place N queens on an N×N chessboard so that no two queens attack each other. This requires that no two queens share the same row, same column, or same diagonal.

Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time.

Branch and Bound is an algorithm design paradigm which is generally used for solving combinatorial optimization problems. It systematically enumerates candidate solutions by means of state space search: the set of candidate solutions is thought of as forming a rooted tree. The algorithm explores branches of this tree, which represent subsets of the solution set. Before enumerating the candidate solutions of a branch, the branch is checked against upper and lower estimated bounds on the optimal solution, and is discarded if it cannot produce a better solution than the best one found so far by the algorithm.

Algorithm:

1. Start in the leftmost column.
2. **Base Case:** If all queens are placed, return `True`.
3. **Recursive Step:** Consider all rows in the current column.
4. For every row in the current column:
   a. **Constraint Check:** Check if the queen can be placed safely in this [row, column] (ensure no other queen is in the same row or diagonals).
   b. **Place:** If safe, mark this [row, column] as part of the solution path.
   c. **Recur:** Recursively check if placing the queen here leads to a solution in the next column.
   d. **Success:** If the recursive call returns `True`, return `True`.
   e. **Backtrack:** If the recursive call returns `False`, unmark this [row, column] (remove the queen) and try the next row.
5. If all rows have been tried and no valid placement is found, return `False`.
6. Stop.

Q&A:

1. **What are the three main components of a CSP?**
   Variables, Domains, and Constraints.

2. **What is the specific constraint in the N-Queens problem?**
   No two queens can share the same row, column, or diagonal.

3. **What is the core principle of Backtracking?**
   It builds candidates recursively and abandons a candidate ("backtracks") if it determines it cannot lead to a valid solution.

4. **How does Branch and Bound differ from pure Backtracking?**
   Branch and Bound uses a bounding function to estimate solution costs and prune the tree more aggressively than simple constraint checking.

5. **What is the worst-case time complexity of the N-Queens problem?**
   The complexity is O(N!), which is factorial time.

6. **What is Pruning in the context of search algorithms?**
   Pruning is the process of ignoring sections of the search tree that are known not to contain the solution.

7. **What is the smallest N for which a solution exists in N-Queens (excluding N=1)?**
   A solution exists for N=4.

8. **Is N-Queens an optimization problem or a decision problem?**
   It is primarily a decision/satisfaction problem, though finding the "best" configuration can be

optimization.

9. **What data structure is typically used to implement Backtracking?**
   A Stack (implicitly via the recursion stack).

10. **Why is BFS not preferred for N-Queens over DFS/Backtracking?**
    BFS requires exponential memory to store all states at a specific level, whereas DFS is memory efficient.

Conclusion:
In this lab, we successfully implemented the N-Queens problem using the Backtracking algorithm and analyzed how constraints allow us to prune the search space to find a valid arrangement effectively.

AI Assignment 5

Aim:
To design and implement an elementary chatbot using pattern matching and rule-based techniques.

Objectives:

- To understand the basic concepts of Natural Language Processing (NLP) and human-computer interaction.
- To implement a response generation system based on keyword matching and predefined rules.

Software Used:

- **OS:** Windows/Linux
- **Language/Tools:** Python 3.x, NLTK or SpaCy libraries

Theory:
A chatbot is an Artificial Intelligence software designed to simulate a conversation with human users. Elementary chatbots are primarily **rule-based** or **pattern-matching** systems. They process user input by identifying specific keywords or patterns and mapping them to predefined responses in a knowledge base. These systems utilize basic NLP tasks such as **tokenization** (splitting text into words) and **normalization** (converting text to lowercase) to match user intent with the correct output.

Algorithm:

1. **Start** the program and display a welcome message to the user.
2. **Define** a dictionary or database containing pairs of user input patterns and corresponding responses.
3. **Accept** text input from the user via the command line or interface.
4. **Pre-process** the input by converting it to lowercase and removing special characters.
5. **Compare** the processed input against the predefined patterns using string matching or regular expressions.
6. **Execute Match:** If a match is found, display the associated response.
7. **Fallback:** If no match is found, display a default response (e.g., "I am sorry, I do not understand").
8. **Loop:** Repeat steps 3 to 7 until the user enters a termination command like "exit" or "bye".
9. **End** the program.

Q&A:
1. What is the role of pattern matching in elementary chatbots?
Ans: Pattern matching identifies specific keywords or phrases in user input to trigger a pre-determined response from the system's database.

2. Mention one limitation of rule-based chatbots.
Ans: They cannot handle complex queries, context, or variations in language that were not explicitly programmed into their ruleset.

3. What is the purpose of a fallback response?
Ans: It ensures the chatbot remains interactive by providing a neutral reply when it fails to recognize user input.

Conclusion:
I have successfully implemented an elementary chatbot that can simulate a basic conversation using rule-based pattern matching.

AI Assignment No: 6

TITLE:
Implement any one of the following Expert System
I. Information management
II. Hospitals and medical facilities
III. Help desks management
IV. Employee performance evaluation
V. Stock market trading
VI. Airline scheduling and cargo schedules

AIM:
To create an Expert System for any one of the above chosen topics.

OBJECTIVES:
Based on the above main aim, the following are the objectives:

1. To study the theory of Expert Systems.
2. To create an Expert System.

Theory

Expert System

An expert system is a computer program that is designed to solve complex problems and to provide decision-making ability like a human expert. It performs this by extracting knowledge from its knowledge base using the reasoning and inference rules according to the user queries.

The expert system is a part of AI, and the first ES was developed in the year 1970, which was the first successful approach of artificial intelligence. It solves the most complex issue as an expert by extracting the knowledge stored in its knowledge base. The system helps in decision making for complex problems using both facts and heuristics like a human expert. It is called so because it contains the expert knowledge of a specific domain and can solve any complex problem of that particular domain. These systems are designed for a specific domain, such as medicine, science, etc.

The performance of an expert system is based on the expert's knowledge stored in its knowledge base. The more knowledge stored in the KB, the more that system improves its performance. One of the common examples of an ES is a suggestion of spelling errors while typing in the Google search box.

Below are some popular examples of Expert Systems:

- **DENDRAL:**
  It was an artificial intelligence project that was made as a chemical analysis expert system. It was used in organic chemistry to detect unknown organic molecules with the help of their mass spectra and knowledge base of chemistry.

- **MYCIN:**
  It was one of the earliest backward chaining expert systems that was designed to find the bacteria causing infections like bacteraemia and meningitis. It was also used for the recommendation of antibiotics and the diagnosis of blood clotting diseases.

- **PXDES:**
  It is an expert system that is used to determine the type and level of lung cancer. To determine the disease, it takes a picture from the upper body, which looks like a shadow. This shadow identifies the type and degree of harm.

- **CaDeT:**
  The CaDet expert system is a diagnostic support system that can detect cancer at early stages.

Characteristics of Expert System

- **High Performance:**
  The expert system provides high performance for solving any type of complex problem of a specific domain with high efficiency and accuracy.

- **Understandable:**
  It responds in a way that can be easily understandable by the user. It can take input in human language and provides the output in the same way.

- **Reliable:**
  It is reliable for generating efficient and accurate outputs.

- **Highly responsive:**
  It provides results for complex queries within a very short period of time.

Components of Expert System

An expert system mainly consists of three components:

- User Interface
- Inference Engine
- Knowledge Base

1. User Interface

With the help of a user interface, the expert system interacts with the user, takes queries as input in a readable format, and passes them to the inference engine. After getting the response from the inference engine, it displays the output to the user. In other words, it is an interface that helps a non-expert user to communicate with the expert system to find a solution.

2. Inference Engine (Rules Engine)

- The inference engine is known as the **brain** of the expert system as it is the main processing unit of the system. It applies inference rules to the knowledge base to derive a conclusion or deduce new information. It helps in deriving an error-free solution of queries asked by the user.
- With the help of an inference engine, the system extracts the knowledge from the knowledge base.

There are two types of inference engine:

- **Deterministic Inference Engine:**
  The conclusions drawn from this type of inference engine are assumed to be true. It is based on facts and rules.

- **Probabilistic Inference Engine:**
  This type of inference engine contains uncertainty in conclusions, and is based on probability.

Inference engine uses the below modes to derive the solutions:

- **Forward Chaining:**
  It starts from the known facts and rules, and applies the inference rules to add their conclusions to the known facts.

- **Backward Chaining:**
  It is a backward reasoning method that starts from the goal and works backward to prove the known facts.

3. Knowledge Base

The knowledge base is a type of storage that stores knowledge acquired from different experts of a particular domain. It is considered as a large storage of knowledge. The larger and richer the knowledge base, the more precise will be the Expert System.

It is similar to a database that contains information and rules of a particular domain or subject. One can also view the knowledge base as collections of objects and their attributes. For example: a **Lion** is an object and its attributes are: it is a mammal, it is not a domestic animal, etc.

Explain your implemented Expert System

Chosen Domain:
IV. Employee performance evaluation

Title of Implemented System:
Expert System for Employee Performance Evaluation

Problem Definition

Organizations need to evaluate employee performance periodically for purposes such as promotion, training, rewards, and feedback. Manual evaluation can be subjective, inconsistent, and time-consuming. An expert system can support HR managers by providing consistent, rule-based evaluations based on measurable criteria.

Inputs and Outputs

- **Input (Facts) about an employee:**
  - Attendance percentage
  - Punctuality (late arrivals per month)
  - Task completion rate (%)
  - Work quality score (e.g., Low / Medium / High)
  - Teamwork score (e.g., Low / Medium / High)
  - Initiative / innovation (Yes / No / High / Medium / Low)
  - Number of missed deadlines
  - Customer / client feedback rating
  - Number of formal warnings or disciplinary actions
- **Output (Conclusions):**
  - Overall performance rating (Outstanding / Good / Average / Poor)
  - Promotion recommendation (Yes / No)
  - Training recommendation (e.g., “Technical training”, “Soft skills training”)
  - Remarks (short explanation)

Knowledge Base (Sample Rules)

The knowledge base stores expert HR rules in the form of IF–THEN production rules. Example rules:

- **Rule 1:**
  IF attendance ≥ 95%
  AND task_completion_rate ≥ 90%
  AND work_quality = High
  AND teamwork = High
  AND warnings = 0
  THEN performance = Outstanding, promotion_recommended = Yes.

- **Rule 2:**
  IF attendance ≥ 85%
  AND task_completion_rate ≥ 80%
  AND work_quality = Medium_or_High
  AND warnings ≤ 1
  THEN performance = Good, promotion_recommended = Consider.

- **Rule 3:**
  IF attendance < 75%
  OR warnings ≥ 3
  THEN performance = Poor, promotion_recommended = No, training = "Discipline & time management".

- **Rule 4:**
  IF work_quality = Low
  AND task_completion_rate < 70%
  THEN performance = Poor, training = "Technical skills".

- **Rule 5:**
  IF teamwork = Low
  AND customer_feedback = Low
  THEN training = "Communication & teamwork skills".

- **Rule 6:**
  IF performance = Outstanding
  AND initiative = High
  THEN remark = "High potential employee".

- **Rule 7:**
  IF performance = Average
  AND missed_deadlines ≥ 3

THEN training = “Planning & deadline management”.

- **Rule 8:**
  IF attendance between 75% and 85%
  AND work_quality = Medium
  AND task_completion_rate between 70% and 80%
  THEN performance = Average.

These rules can be extended as needed.

Inference Engine and Reasoning Mode

- **Type:** Deterministic rule-based inference engine.
- **Reasoning Mode:** Forward chaining (data-driven).

Working:

1. The system asks the HR user to input employee data (attendance, quality score, etc.).
2. These values are stored as **facts** in working memory.
3. The **forward chaining** engine scans all rules and finds those whose IF-conditions match the current facts.
4. When a rule’s conditions are satisfied, it **fires**:
   - Adds new facts (e.g., performance = Good, training = “Technical skills”).
5. Newly added facts may trigger further rules (e.g., rules about promotion based on performance), until no more rules can fire.
6. The final set of facts is presented as the evaluation result:
   - Overall rating, promotion suggestion, and training recommendations.

Simple Algorithm (Step-wise Working)

```
1. Start.
2. Initialize knowledge base with HR rules.
3. Initialize empty working memory.
4. Input employee metrics from user (HR) through the user interface.
5. Add all input data as facts to working memory.
6. Repeat:
    a. For each rule in the knowledge base,
    b. If all IF-conditions of the rule match facts in working memory and the rule is not yet fired,
        ○ Then fire the rule and add its THEN-part conclusions as new facts.
7. Stop when no new rules can fire.
8. Extract final facts related to performance, promotion, and training.
9. Display results and brief explanation to user.
10. End.
```

Q&A

1. **What is an Expert System?**
   An Expert System is an AI program that uses a knowledge base and an inference engine to solve complex problems and make decisions similar to a human expert in a specific domain.

2. **What are the three main components of an Expert System?**
   The three main components are:
   - Knowledge Base
   - Inference Engine
   - User Interface
3. **What is the role of the Knowledge Base in an Expert System?**
   The Knowledge Base stores domain-specific knowledge in the form of facts, rules, and relationships. It represents the expertise of human specialists and is used by the inference engine to derive conclusions.

4. **What does the Inference Engine do?**
   The Inference Engine is the reasoning mechanism. It applies logical inference rules (e.g., forward or backward chaining) to the knowledge base and available facts to derive new facts, explanations, or decisions.

5. **Differentiate between deterministic and probabilistic inference engines.**
   - A **deterministic** inference engine assumes its conclusions are certain and based strictly on rules and facts.
   - A **probabilistic** inference engine deals with uncertainty and associates conclusions with probabilities or confidence levels.
6. **What is Forward Chaining?**
   Forward chaining is a data-driven reasoning method that starts from known facts and applies rules to infer new facts until a goal or conclusion is reached.

7. **What is Backward Chaining?**
   Backward chaining is a goal-driven reasoning method that starts from a hypothesis or goal and works backward, checking which facts and rules are needed to support that goal.

8. **Which reasoning method is used in the Employee Performance Evaluation Expert System and why?**
   The system uses **forward chaining** because it starts from available employee data (facts) and incrementally applies rules to derive performance ratings, promotion recommendations, and training suggestions.

9. **What input parameters are used by the Employee Performance Evaluation Expert System?**
   Input parameters include: attendance percentage, punctuality, task completion rate, work quality score, teamwork score, initiative, number of missed deadlines, customer feedback, and number of warnings.

10. **What are the advantages of using an Expert System for employee performance evaluation?**
    - Provides consistent and objective evaluations.
    - Reduces human bias and subjectivity.
    - Saves time in periodic assessments.
    - Supports HR managers with clear, rule-based recommendations.
    - Can suggest targeted training and development plans.

Conclusion

In this experiment, the concept and theory of Expert Systems were studied, including their characteristics, components, and reasoning modes. An Expert System for **Employee Performance Evaluation** was designed using a rule-based knowledge base and a forward-chaining inference engine. The system accepts employee-related performance data as input and produces an overall rating, promotion recommendations, and training suggestions. This demonstrates how Expert Systems can assist organizations in making consistent, transparent, and efficient decisions in human resource management.

Cloud Computing Assignment No: 1

Practical Title: Case study on Amazon EC2 and understanding Amazon EC2 web services

Objectives:

- To understand what Amazon EC2 is
- To learn how Amazon EC2 web services work

Hardware Requirements:

- Computer with at least Pentium IV or better

Software Requirements:

- Ubuntu 20.04

Theory:

An EC2 instance is simply a virtual machine (virtual server) in Amazon Web Services (AWS).
EC2 stands for **Elastic Compute Cloud**.

It is a cloud service where an AWS user can request and create a server (instance) in the AWS cloud and use it to run applications.

With an **on-demand EC2 instance**, you can:

- Rent a virtual server by the hour
- Run your own applications on it
- Pay only for the time you use it

The cost depends on the **instance type** you choose. Different instance types have different CPU, memory, and pricing, so you can pick one that matches your business or project needs.

You can:

- Select an instance with the CPU and RAM you need
- Use it as long as you want

- Stop or terminate it when you don’t need it anymore

This helps reduce **capital expenses (CAPEX)** because you don’t need to buy physical servers.

Below are the steps to create (launch) an on-demand EC2 instance in AWS.

Steps to Launch an On-Demand EC2 Instance

Step 1: Log in and Open the EC2 Service

1. Log in to your **AWS account**.
2. Click on the **“Services”** menu at the top left.
3. You will see AWS services grouped by categories like **Compute**, **Storage**, **Database**, etc.
4. Under the **Compute** category, click on **EC2**.
5. This opens the **EC2 Dashboard**, where you can see a summary of your EC2 resources (instances, volumes, etc.).

Step 2: Select AWS Region

1. On the top right corner of the EC2 Dashboard, choose the **AWS Region** where you want to create your EC2 instance.
2. For example, you can select **N. Virginia**.
   AWS has multiple regions around the world, and you can pick one based on your needs (like latency or compliance).

Step 3: Start Creating an Instance

1. After you select the region, go back to the **EC2 Dashboard**.
2. Click on the **“Launch Instance”** button under the “Create Instance” section.
3. This will open the **instance creation wizard**.

Choose an AMI (Amazon Machine Image)

Step 1: Select an AMI

1. The first step in the wizard is to choose an **AMI (Amazon Machine Image)**.

    - An AMI is a template for the operating system (OS) of your instance.
    - It includes the OS and sometimes pre-installed software.
2. When you launch an instance from an AMI, the instance will start with that OS.
3. Here, we select the default **Amazon Linux (64-bit)** AMI.

Choose EC2 Instance Type

Step 1: Select Instance Type

1. Next, you need to choose the **instance type** based on your requirements.
2. For this example, select **t2.micro**, which has:
    - 1 vCPU (virtual CPU)
    - 1 GB of memory
3. Click **"Configure Instance Details"** to continue.

Configure Instance Details

Step 1: Number of Instances

1. In this step, you set how many instances you want to launch at once.
2. You can launch up to **20 instances** at a time.
3. For now, we will launch **one** instance.

Step 2: Purchasing Options

1. Under **Purchasing options**, there is an option called **"Request Spot Instances"**.
2. Leave this **unchecked** for now, because we are creating an **on-demand** instance (not a Spot instance).

Step 3: Networking – Choose VPC and Subnet

1. Now you must choose basic networking settings for your instance.

2. You need to decide:

    - In which **VPC (Virtual Private Cloud)** the instance will run
    - In which **subnet** inside that VPC it will be placed
3. It is better to plan your network design in advance:

   - Decide IP ranges for your subnets
   - Decide which resources are public or private
4. For example:
   - A **web server** is usually placed in a **public subnet**
   - A **database server** is usually placed in a **private subnet**
5. Under **Network**, you will see a list of available VPCs:
   - You can select an existing VPC
   - Or create a new VPC if needed
6. Here, we select an **existing VPC** to launch the instance.

Step 4: Choose Subnet

1. A **VPC** contains **subnets**. Each subnet uses a specific IP range and can control access.
2. Under **Subnet**, choose where you want to place your instance:
   - Select an existing subnet, for example a **public subnet**
   - You can also create a new subnet if required
3. If you place the instance in a **public subnet**, AWS can assign it a **public IP** from its pool automatically.

Step 5: Public IP Assignment

1. You can choose whether AWS should:
   - Automatically assign a public IP
   - Or you will assign it manually later
2. You can set this using the **"Auto-assign Public IP"** option.
3. In this example, we plan to give the instance a **static IP** later, called an **Elastic IP (EIP)**.
   So, we **disable** Auto-assign Public IP for now.

Q&A

Q1. What does EC2 stand for in AWS?
A: Elastic Compute Cloud

Q2. What is an EC2 instance?
A: A virtual machine (virtual server) running in the AWS cloud

Q3. What type of billing model is used for on-demand EC2 instances?
A: Pay-as-you-go billing

Q4. What is the purpose of an AMI in EC2?
A: It provides the OS template and software configuration for the instance

Q5. Which instance type is typically used for beginners and small workloads?
A: t2.micro

Q6. Which AWS component allows users to choose where an instance runs geographically?
A: AWS Region

Q7. Which networking component defines the virtual network for EC2 instances?
A: VPC (Virtual Private Cloud)

Q8. Which subnet type is commonly used for public-facing servers?
A: Public subnet

Q9. What static public IP service can be attached to an EC2 instance?
A: Elastic IP (EIP)

Q10. How do on-demand EC2 instances reduce infrastructure cost?
A: By eliminating the need to purchase physical servers and paying only for usage

Conclusion:

In this exercise, we learned how to create an **on-demand EC2 instance** in AWS.

Since it is an on-demand instance:

- You pay only for the time it is running
- You can **start** it when you need it
- You can **stop** or **terminate** it when not needed

This helps you **reduce costs**, because you don't have to keep a server running all the time if you are not using it.

Cloud Computing Assignment No: 2

Practical Title: Installation and configuration of Google App Engine

Objectives:
• To understand the basics of Google App Engine
• To install and set up Google App Engine

Hardware Requirements:
• Computer with at least Pentium IV or higher configuration

Software Requirements:
• Ubuntu 20.04
• Google App Engine (web application platform)

Theory

Introduction

Google App Engine is a platform for hosting **web applications**.
A web application is any application that users access over the internet, usually through a web browser.
Examples include:

- Online shopping sites
- Social networking platforms
- Multiplayer games
- Mobile back-end services
- Survey tools
- Project management and collaboration tools
- Publishing systems

App Engine can also host normal website content such as images and documents, but it is mainly designed for **dynamic, real-time applications**. Web browsers are the most common clients, but App Engine can also be used by mobile apps and other types of clients.

A key feature of Google App Engine is that it is built to handle **many users at the same time**.
When an application can support a large number of users without slowing down, we say that it **scales** well.

On Google App Engine, applications scale **automatically**:

- As more users use your app, App Engine automatically provides more resources (like processing power and storage).

- You do not need to manage the underlying servers yourself.
- Your application code does not need to know about or manage these resources directly.

Google App Engine is a **cloud-based platform** that combines:

- IaaS (Infrastructure as a Service)
- PaaS (Platform as a Service)
- SaaS (Software as a Service)

It supports developing, testing, and delivering applications on demand in a cloud environment that can support millions of users and is highly scalable.

Google provides this platform to developers and companies so they can build **SaaS (Software as a Service)** solutions at a lower cost and with less maintenance.
It is especially useful for:

- Small and medium businesses
- Large enterprises
  that have web-based applications requiring **high scalability** without losing performance.

Well-known companies such as **Best Buy** and **Khan Academy** use Google App Engine to host their applications.

Google App Engine Overview

Google App Engine is a **fully managed PaaS cloud platform**.
It provides built-in services that help you run your apps without worrying about server management.

Key points:

- You download the **Software Development Kit (SDK)** to start development.
- You can choose a programming language (like Python, Java, PHP, etc.) and follow the developer guide for that language.
- After signing up for a **Google Cloud** account, you can quickly start building and deploying your application.

Cloud SQL and Generally Available Features

Google App Engine provides many built-in services, such as **Cloud SQL** (a managed relational database that you can use, for example, from PHP apps).

Some features are marked as **Generally Available (GA)**:

- These features are covered by the App Engine **service-level agreement (SLA)** and **deprecation policy**.
- GA features are considered stable and safe to use in production.
- Changes to these features are backward-compatible (they will not break existing apps).

These features include capabilities for:

- Data storage, retrieval, and search
- Communication
- Process management
- Computation
- Application configuration and management

Examples of services and components:

- Cloud SQL
- Logs
- Datastore
- Dedicated Memcache
- Blobstore
- Memcache
- Search
- Cloud Endpoints
- Modules
- SSL for custom domains
- Remote access
- Multitenancy support

Advantages of Google App Engine

1. Strong Security Infrastructure

Google's global internet infrastructure is known for being highly secure.

- Application code and data are stored on secure Google servers.
- Unauthorized access is very rare.
- Your app can be served to users around the world from many Google data centers.
- Google's security and privacy policies also apply to apps built on Google App Engine.

2. Scalability

Scalability is one of the most important features for any successful app.

- Google uses technologies like **GFS (Google File System)** and **Bigtable** for its own services, and similar technologies are used in App Engine.
- When you use Google App Engine, you mainly need to focus on writing your application code.
- App Engine automatically tests and scales your app as needed.
- Whether your app has a few users or millions, App Engine can automatically scale up or down based on demand.

3. Performance and Reliability

Google is a global leader in terms of performance and reliability. Over many years, it has built very fast and reliable services.

- Google App Engine benefits from this same infrastructure.
- Apps running on App Engine can achieve high performance and uptime similar to other Google products.

4. Cost Savings

With App Engine:

- You do **not** need to buy and maintain your own servers.
- You do **not** need to hire a separate team to manage infrastructure.
- You can instead spend your time and money on improving your business and your application.

5. Platform Independence

- Your data is not strongly locked into App Engine.
- You can move your data and application to another environment if needed.
- This reduces dependency on a single platform and gives you more flexibility.

Q&A
Q1. What type of platform is Google App Engine?
A: A fully managed Platform as a Service (PaaS)

Q2. What is Google App Engine primarily used for?
A: Hosting scalable web applications and services

Q3. What is a key feature of Google App Engine regarding scalability?
A: It scales applications automatically based on user demand

Q4. Which types of clients commonly access applications on Google App Engine?
A: Web browsers and mobile applications

Q5. Which service provides managed relational databases for App Engine apps?
A: Cloud SQL

Q6. What does Generally Available (GA) mean in App Engine services?
A: Features are stable, production-ready, and covered by SLA

Q7. What programming languages can be used with Google App Engine?
A: Languages such as Python, Java, and PHP

Q8. What advantage does App Engine provide regarding server management?
A: Developers do not need to manage or configure physical servers

Q9. Name a major benefit of using App Engine for businesses.
A: Cost savings due to reduced infrastructure and maintenance requirements

Q10. Which major companies use Google App Engine for applications?
A: Organizations like Best Buy and Khan Academy

Conclusion

Thus, we have learned the basics of Google App Engine and understood how it can be installed and configured as a cloud platform for building and deploying scalable web applications.

Cloud Computing Assignment No: 3

Practical Title: Creating an Application in Salesforce.com using the Apex Programming Language

---

Objectives:

• To understand basic Salesforce cloud administration
• To create an application in Salesforce.com using the Apex language

---

Hardware Requirements:

• Computer with at least Pentium IV or higher configuration

Software Requirements:

• Ubuntu 20.04
• Web application: Salesforce.com

Theory

What is Apex?

Apex is a special programming language created by Salesforce.com for its cloud platform.

Officially, Apex is a **strongly typed, object-oriented programming language** that lets developers run flow and transaction control statements on the Force.com platform, together with calls to the Force.com API.

Key points:

- It looks and feels similar to **Java** (syntax is Java-like).
- It behaves like **database stored procedures** (runs on the server close to the data).
- It lets you add **business logic** to many system events, such as:
    - Button clicks
    - Updates to related records
    - Visualforce page actions

- Apex code can run:
    - From **triggers** on objects (e.g., when a record is inserted or updated)
    - From **web service** requests

Apex is available in:

- Developer Edition
- Enterprise Edition
- Unlimited Edition
- Performance Edition

Features of Apex as a Language

1. Integrated

- Apex has built-in support for **DML (Data Manipulation Language)** operations like:
    - INSERT
    - UPDATE
    - DELETE
- It also has built-in **DML exception handling**.
- You can write **SOQL** (Salesforce Object Query Language) and **SOSL** (Salesforce Object Search Language) queries directly inside Apex.
- These queries return **sObject** records (Salesforce objects like Account, Contact, etc.).

2. Java-like Syntax and Easy to Use

- Apex syntax is very similar to Java:
    - Variable declarations
    - Loops (for, while)
    - Conditional statements (if-else)
- This makes it easier to learn if you already know Java or similar languages.

3. Strongly Integrated With Data

- Apex is **data-centric**:
    - It is designed to run many queries and DML operations together.
    - It can handle multiple database operations in a single process or transaction.

4. Strongly Typed

- Apex is a **strongly typed** language:
    - You must declare the data type for each variable.

    - It uses direct references to schema objects (like Account, Contact).
    - If a field or object is removed or has the wrong type, the code will fail at compile time, helping catch errors early.

5. Multitenant Environment

- Salesforce is **multitenant** (many customers share the same infrastructure).
- The Apex runtime engine is designed to:
    - Prevent any one piece of code from using too many shared resources.
    - Enforce **governor limits** (limits on queries, CPU time, etc.).
- If code exceeds these limits, it stops and shows clear error messages.

6. Automatic Upgrades

- Apex is automatically upgraded when Salesforce releases new versions.
- You do not need to manually upgrade your Apex code environment.

7. Easy Testing

- Apex has built-in support for:
    - Writing **unit tests**
    - Running tests and viewing results
    - Measuring **code coverage** (how much of your code is tested)
- This helps improve code quality and reliability.

When Should a Developer Use Apex?

You should use Apex when built-in Salesforce configuration tools (like formulas, validation rules, workflows, and flows) are **not enough** to handle your business requirements.

Typical use cases for Apex:

- **Custom email logic**

    - Creating complex email services (e.g., mass emails with specific rules, advanced email handling).
- **Complex validation across multiple objects**

    - When you must check data across different objects at the same time.
    - When validation rules are too simple for your needs.
- **Advanced business logic**

    - Logic that cannot be done with standard workflows, Process Builder, or Flow.
    - Multi-step or conditional processes that are too complex for point-and-click tools.
- **Operations on many records or objects in one transaction**

    - When you need to update or process multiple related records at once.
    - When you require full control of a transaction (all-or-nothing behavior).
- **Triggers**

    - When something should happen automatically after an event such as:
        - Insert, update, delete, or undelete of records.
    - Example: When a record is updated, automatically update related records.

Working Structure of Apex

Apex runs completely on the **Force.com platform**, and works on demand.

There are two main action flows:

1. When the **developer** writes and saves code
2. When an **end user** does something that runs the code

1. Developer Action

When a developer writes Apex code and clicks **Save**:

- The Salesforce application server:
    - Compiles the Apex code into instructions that the Apex runtime can understand.
    - Stores these compiled instructions as **metadata** in Salesforce.

2. End User Action

When an end user makes an action that invokes Apex (for example, clicking a button or opening a Visualforce page):

- The Salesforce server:
    - Gets the compiled Apex instructions from metadata.
    - Sends them to the Apex runtime interpreter.
    - Executes the code and returns the result to the user.

From the end user's point of view:

- The app behaves like any normal Salesforce page.
- There is no visible delay compared to normal platform operations.

Limitations of Apex

Because Apex is a **proprietary** language built specifically for Salesforce, it does not include all the features of general-purpose languages.

In general:

- It cannot directly control or create custom browser-level UI elements (you use Visualforce or Lightning for that).
- It cannot fully replace or stop some standard Salesforce platform behaviors.
- It does not support low-level operations such as direct access to the operating system, file system, etc., like other languages might.

Understanding Basic Apex Syntax

Typical parts of Apex code are similar to other programming languages:

1. Variable Declaration

- Because Apex is strongly typed, every variable must have a data type.
- Example:
  `List<Account> lstAcc;`
  Here, `lstAcc` is a variable of type "list of Account records."

2. SOQL Query

- **SOQL (Salesforce Object Query Language)** is used to get data from Salesforce.
- Example:
  `lstAcc = [SELECT Id, Name FROM Account];`
  This fetches records from the **Account** object.

3. Loop Statement

- Used to repeat code for each item in a collection or for a set number of times.
- Example:

for (Account acc : lstAcc) {

  // code to process each account

}

This loop runs once for each record in `lstAcc`.

4. Flow Control Statement

- **If** statements are used to decide whether a block of code should run.
- Example:

```
if (!lstAcc.isEmpty()) {
    // run this code only if there are records
      }
```

5. DML Statement

- DML (Data Manipulation Language) is used to change records in the database:
    - INSERT, UPDATE, UPSERT, DELETE
- Example:

```
update lstAcc;
```

  This updates the accounts in `lstAcc` with their new field values.

Apex Code Development Tools

In supported Salesforce editions, you can write and manage Apex using the standard Salesforce developer tools (such as the Developer Console or other supported IDEs and online editors provided by Salesforce).

Q&A

Q1. What programming language is used for developing applications on Salesforce.com?
A: Apex

Q2. What type of programming language is Apex?
A: Strongly typed, object-oriented language

Q3. Which platform runs Apex code?
A: Force.com (Salesforce platform)

Q4. Which Salesforce feature allows automatic execution after record changes?
A: Triggers

Q5. Which query language is used in Apex to fetch records?
A: SOQL (Salesforce Object Query Language)

Q6. Which type of limits does Salesforce enforce for resource usage?
A: Governor limits

Q7. What Salesforce service provides relational database support for Apex applications?
A: Cloud SQL (via Salesforce environment services)

Q8. In which editions is Apex available?
A: Developer, Enterprise, Unlimited, and Performance Editions

Q9. How is Apex similar to Java?
A: It has Java-like syntax including variables, loops, and conditionals

Q10. When should developers use Apex?
A: When built-in Salesforce configuration tools cannot handle complex business logic

Conclusion

Thus, we have created an application in Salesforce.com using the Apex programming language and understood its basic features, structure, and when to use it.

Cloud Computing Assignment No: 4

Practical Title: Design and develop a custom Application (Mini Project) using Salesforce Cloud

---

Objectives:

• To learn Salesforce cloud administration
• To install and configure Salesforce cloud administrative features

---

Hardware Requirements:

• Computer with at least Pentium IV or higher configuration

Software Requirements:

• Ubuntu 20.04
• Web application: salesforce.com

Theory

Introduction

Salesforce.com, Inc. is an American cloud-based software company based in San Francisco, California.

Most of its income comes from its **CRM (Customer Relationship Management)** product, but it also provides other business applications for:

- Customer service
- Marketing automation
- Data analytics
- Application development

Salesforce (the CRM part) gives companies:

- Tools for **case management** (handling customer issues)
- **Task management** (tracking work to be done)
- A system to automatically route and escalate important events

The **Salesforce customer portal** allows customers to:

- Track their own support cases
- Use social features to join discussions about the company (via social plugins)
- Use analytics tools
- Get email alerts
- Use Google search
- View entitlements (what support/services they are allowed) and contracts

---

Lightning Platform (Force.com)

Lightning Platform (earlier called **Force.com**) is a **Platform as a Service (PaaS)**.

It lets developers build custom applications that work together with the main Salesforce system.
Key points:

- These extra (add-on) apps run on Salesforce's own cloud infrastructure.
- Apps can be built using:
    - **Declarative tools** (point-and-click configuration)
    - **Apex** (Salesforce's Java-like programming language)
    - **Lightning** framework
    - **Visualforce** (a framework with XML-based syntax used mainly to create HTML pages)

Salesforce usually releases **three major updates per year**.
Because the platform is provided as a cloud service:

- Every developer or customer environment is automatically updated.
- You always work on the latest platform version.

Community Cloud

Community Cloud lets Salesforce customers create **online portals and communities** such as:

- Customer support portals
- Partner or reseller portals (channel sales)
- Collaboration sites
- Other custom external sites

These communities:

- Are built inside a Salesforce org
- Are tightly integrated with:

    - **Sales Cloud**
    - **Service Cloud**
    - **App Cloud**

Community Cloud can be quickly customized to create many types of web properties (for example, help centers, partner communities, or customer forums).

Salesforce Sales Cloud

Sales Cloud is Salesforce's main **CRM platform** for managing:

- Sales
- Marketing
- Customer support

It works for:

- **B2B (Business-to-Business)**
- **B2C (Business-to-Customer)**

Sales Cloud:

- Is fully customizable
- Brings all customer information together in one unified platform
- Supports:
    - Lead generation
    - Sales processes
    - Customer service
    - Business analytics
    - Access to thousands of extra apps via the **AppExchange**

Sales Cloud is provided as **Software as a Service (SaaS)**:

- You access it using a **web browser**
- A **mobile app** is also available

There is also a **real-time social feed** (like a collaboration stream) where users can:

- Share information
- Ask questions
- Collaborate with colleagues

Salesforce offers different editions of Sales Cloud, charged **per user per month**:

- Group
- Professional
- Enterprise
- Unlimited
- Performance

Support plans include:

- Standard Success Plan
- Premier Success Plan
- Premier+ Success Plan

Creating Custom Apps for Salesforce Classic

Custom apps in Salesforce Classic help users access all required items (tabs, objects, tools) in **one place**.

If you are completely new to building apps, Salesforce recommends using the **Lightning Platform Quick Start** to create a basic app in one step.

If you already have your **objects, tabs, and fields** created, you can manually set up a custom app as follows.

Steps to Create a Custom App in Salesforce Classic

1. From **Setup**, type **Apps** in the **Quick Find** box, then click **Apps**.
2. Click **New** to create a new app.
3. If Salesforce Console is available, choose whether you want to create:
    - A **custom app**, or
    - A **Salesforce Console** app.
4. Enter an **App Name** and a **Description**.
    - The app name can be up to **40 characters**, including spaces.
5. (Optional) Add a **custom logo** to brand your app.
6. Select which **tabs/items** should appear in the app (for example, Accounts, Contacts, custom objects, etc.).
7. (Optional) Set the **Default Landing Tab**:
    - This is the first tab a user sees when they log in using this app.
8. Choose which **profiles** can see this app.
9. Tick the **Default** checkbox next to a profile if you want this app to be the **default app** for that profile.
    - New users with that profile will see this app first when they log in.
10. Click **Save** to finish creating the app.

Difference Between a Custom Application and a Console Application in Salesforce

Custom Application:

- A custom app is a group of:
  - Tabs
  - Objects
  - Other components
- These pieces work together to solve a specific business problem (for example, a "Support App" or "Recruitment App").

Console Application:

- A console app uses a **special Salesforce user interface** called the **console**.
- It shows information in a **single, tabbed workspace**.
- Designed to improve productivity by:
  - Allowing users to work on multiple records from one screen
  - Reducing the need to open many separate windows or tabs

So, in simple terms:

- A **custom app** is about which tabs and objects are grouped together.
- A **console app** is about a **different, more efficient UI** that helps heavy users (like support agents) work faster.

Q&A

Q1. What cloud platform is used to build custom applications in Salesforce?
A: Lightning Platform (Force.com)

Q2. What is Salesforce primarily known for in industry?
A: CRM (Customer Relationship Management)

Q3. What type of service model does Salesforce Sales Cloud follow?
A: Software as a Service (SaaS)

Q4. What tools can be used to build custom apps in Salesforce?
A: Declarative tools, Apex, Visualforce, and Lightning

Q5. What is the purpose of Community Cloud?
A: To create external portals and online communities for customers and partners

Q6. What is a custom application in Salesforce Classic composed of?
A: Tabs, objects, and components grouped for a specific business function

Q7. What is a console application designed for?
A: High productivity workflows using a single tabbed workspace

Q8. How are Salesforce platforms upgraded?
A: Automatically through three releases per year

Q9. Which Salesforce cloud manages sales, marketing, and customer support?
A: Sales Cloud

Q10. What is the main objective of creating a custom app in Salesforce?
A: To tailor workflows and data access to specific business needs

Conclusion

Thus, we have designed and developed a custom application using Salesforce Cloud, understanding its main components (Sales Cloud, Lightning Platform, Community Cloud) and how to configure and create a custom app in Salesforce.

Cloud Computing Assignment No: 5

Practical Title: Set up your own cloud for Software as a Service (SaaS) over the existing LAN using open-source tools and HDFS, with basic encrypted file operations

---

Objectives:

- To build your own cloud environment for SaaS over the existing LAN
- To implement basic file operations such as splitting a file into blocks/segments

---

Hardware Requirements:

- Computer with at least Pentium IV or higher configuration

Software Requirements:

- Ubuntu 20.04
- VMware ESXi (for building a private cloud environment)

Theory

In this practical, we are setting up a **private cloud** using **VMware ESXi** over the existing LAN.
On top of this environment, you can later implement your own **cloud controller** and integrate with **HDFS** to:

- Split files into segments or blocks
- Upload and download files to/from the cloud in **encrypted** form

The theory section below focuses on **installing and configuring VMware ESXi** as the base cloud platform.

Installing VMware ESXi (Host / Node Setup)

ESXi Hardware Requirements

For VMware ESXi 6.7, the host machine must meet the following requirements:

- At least **two CPU cores**
- **64‑bit x86** processor
- **NX/XD bit** enabled in the BIOS (for security hardware support)
- Minimum **4 GB RAM** (at least **8 GB** recommended for running virtual machines in a normal production-like setup)
- For running 64‑bit virtual machines, **hardware virtualization** support must be enabled in BIOS:
    - Intel VT‑x or
    - AMD‑V
- One or more **Gigabit or faster Ethernet** network interfaces (NICs)
- **SCSI disk** or a local, non‑network **RAID LUN** with unpartitioned space for virtual machines

For **SATA disks**:

- They must be connected via a supported SAS controller or supported onboard SATA controller.
- ESXi generally treats SATA disks as **remote**, not local, so by default they are **not** used as a scratch partition.

ESXi Installer – Basic Installation Steps

1. **Boot the ESXi installer** on the host machine.
2. **Accept the License Agreement** when prompted.
3. **Select the Storage Device** where ESXi will be installed (e.g., a local disk or RAID volume).
4. **Select the Keyboard Layout** based on your region.
5. **Set the ESXi Root Password**:
    - This is the administrator password for managing the ESXi host.
6. Start the installation and wait for it to complete.
7. After installation, **reboot** the host.

Once the host restarts, you can perform **basic configuration** using the Direct Console User Interface (CLI-like local menu) and then move to GUI-based management.

Initial Configuration from the Local Interface (CLI-style Menu)

On the ESXi console screen (yellow/grey screen), use the menu to:

1. **Configure Management Network**

    - Set the network adapter used for management.

2. **Set IPv4 Settings**
    - Assign a **static IP address**, subnet mask, and gateway to the ESXi host.
3. **Set DNS Server and Hostname**
    - Configure the **DNS server address**.
    - Set the **hostname** for the ESXi host.
4. **Restart Management Network**
    - Apply the changes by restarting the management network services.

After this, you can access the ESXi host remotely through a web browser or via vCenter.

GUI Access and Cluster Setup (vSphere / vCenter)

Next, use the **vSphere Client** or **vCenter** (if available) to manage ESXi using a graphical interface and to create a private cloud cluster.

1. Access vCenter / vSphere Web Client

- Open a browser and connect to the **vCenter Server** or directly to the ESXi host (if managing standalone).
- Log in with the appropriate credentials (e.g., root for ESXi host, or vCenter admin).

2. Create a Datacenter

- In vCenter, create a new **Datacenter** object.
    - This acts as a logical container for clusters and hosts.

3. Create a Cluster

- Inside the Datacenter, create a **Cluster**.
- Give the cluster a **name** (e.g., “Lab-Cluster”).
- Optionally enable features like:
    - **DRS (Distributed Resource Scheduler)**
    - **HA / Failover** (High Availability)

4. Add ESXi Hosts to the Cluster

For each host:

1. **Add Host**
   - Choose **Add Host** from the cluster or datacenter.
2. **Enter Host IP / Name**
   - Enter the **IP address** or hostname of the ESXi host.
3. **Enter Host Credentials**
   - Provide the **root username and password** for the ESXi host.
4. **Review Host Summary**
   - Check CPU, memory, storage, and network details of the host.
5. **Lockdown Mode (Optional)**
   - Decide whether to enable **Lockdown Mode**:
     - Restricts direct login to the host and forces management through vCenter.
6. **Add Host to Resource Pool / Cluster**
   - Confirm adding the host to the cluster or a specific **resource pool** if configured.
7. **Finish**
   - Complete the wizard. The host will appear under the cluster in vCenter.

Viewing and Managing Resources

- **Host View and Configuration**
  - Check details like:
    - CPU usage
    - Memory usage
    - Local storage
    - Network configuration
- **Cluster View and Configuration**
  - Monitor:
    - Combined resources of all hosts
    - Virtual machines running in the cluster
    - DRS, HA, and failover settings

This ESXi + vSphere setup forms your **private cloud infrastructure**.
On top of this, you can:

- Deploy virtual machines to act as **HDFS nodes**
- Write your **cloud controller code** to:
  - Split files into blocks
  - Encrypt them
  - Upload and download them from the HDFS-based cloud storage over the LAN

Q&A

Q1. Which hypervisor platform is used to build the private cloud in this practical?
A: VMware ESXi

Q2. What service model is implemented over the private cloud LAN?
A: Software as a Service (SaaS)

Q3. Which distributed storage framework is used for file segmentation and cloud operations?
A: HDFS (Hadoop Distributed File System)

Q4. What is the purpose of using HDFS in this setup?
A: To split files into blocks and support upload/download operations

Q5. Which tool provides GUI-based management for ESXi hosts and clusters?
A: vSphere / vCenter

Q6. What BIOS feature must be enabled for running 64-bit virtual machines?
A: Intel VT-x or AMD-V

Q7. What network settings must be configured on the ESXi console for remote access?
A: IP address, subnet mask, gateway, DNS, and hostname

Q8. What is the root account used for in ESXi?
A: Administration and host management

Q9. What is created in vCenter before forming a cluster?
A: A Datacenter object

Q10. What is the benefit of building a private cloud over an existing LAN?
A: It enables scalable SaaS deployment with local control and reduced infrastructure cost

Conclusion

In this practical, we have configured a **vSphere private cloud** using VMware ESXi over the existing LAN. This private cloud environment can now be used as a base to implement SaaS services and custom file operations (splitting files into blocks and handling encrypted upload/download using HDFS and your own cloud controller code).