1. Explain how you would handle missing data in a dataset and provide a code snippet demonstrating this.

Handling missing data in a dataset is an important step in data preprocessing to ensure accurate and reliable analysis. There are several approaches to deal with missing data, and the choice depends on the nature and characteristics of the dataset. Some commonly used methods are:

a) Deletion: If the missing data is minimal and does not significantly affect the analysis, one can simply delete the rows or columns containing missing values. For example, if a column has more than a set threshold of, say, 30% of missing data, then one can consider dropping the whole column. This approach is straightforward but may lead to loss of valuable information.

```
import pandas as pd

# Load the dataset
df = pd.read_csv('dataset.csv')

# Calculate the percentage of missing values in each column
missing_percentages = df.isnull().sum() / len(df) * 100

# Identify columns with missing values exceeding 30%
columns_to_delete = missing_percentages[missing_percentages > 30].index

# Delete the columns with excessive missing values
df_dropped_columns = df.drop(columns_to_delete, axis=1)

# Print the resulting dataset
print(df_dropped_columns.head())
```

b) Imputation: Imputation involves filling in the missing values with estimated values. This can be done using various techniques such as mean imputation, median imputation, mode imputation, or regression imputation. Mean imputation replaces missing values with the mean of the available values in that column, median imputation uses the median (the middle value from the list of sorted values present in the column), and mode imputation uses the mode (value appearing maximum number of times in the said column). Regression imputation involves creating a regression model (using the best fit line) to predict the missing values based on other variables in the dataset.

```
import pandas as pd

# Load the dataset
df = pd.read_csv('dataset.csv')

# Impute missing values with mean
df_mean_imputed = df.fillna(df.mean())

# Impute missing values with median
```

```
df_median_imputed = df.fillna(df.median())

# Impute missing values with mode
df_mode_imputed = df.fillna(df.mode().iloc[0])

# Print the resulting datasets
print(df_mean_imputed.head())
print(df_median_imputed.head())
print(df_mode_imputed.head())
```

c) Using a placeholder: In some cases, it may be appropriate to use a specific value (e.g., "Unknown" or "-1") to represent missing data. This approach is useful when the missing data has a specific meaning or when the missing-ness itself is informative.

```
import pandas as pd

# Load the dataset
df = pd.read_csv('dataset.csv')

# Replace missing values with -1
df_placeholder = df.fillna(-1)

# Replace missing values with "Unknown"
df_placeholder_string = df.fillna("Unknown")

# Print the resulting datasets
print(df_placeholder.head())
print(df_placeholder_string.head())
```

2. Prepare a high-level lesson plan for an introductory session on deep learning.

Every session should have an objective. Since, it is an introductory session to deep learning, it is necessary to start from the absolute basics and then move up to more complex topics. Jargons should be avoided and the session should be made as relatable to the audience as possible, which can be done by using relevant examples. However, since it is just an introductory session, we can set the objective as:

Objective: Introduce students to the fundamentals of deep learning and provide an overview of its applications.

1. Introduction (15 minutes)
   - Define Machine Learning and Artificial Intelligence.

- Define Deep Learning and its relationship to Artificial Intelligence and Machine Learning.
- Explain the importance and impact of Deep Learning in different domains.

2. Neural Networks Basics (25 minutes)
   - Introduce the concept of neurons, their working, and their role in information processing. Relate it to biological neurons to help the audience visualise the process better.
   - Explain activation functions, weights, bias, threshold, learning rate and their significance in artificial neural networks (ANNs).
   - Can explain Linear separability, McCulloch Pitts Neuron (Theory and Architecture) and Hebb Network (Theory and Algorithm); depending on audience and time constraint. (This can help the audience understand the terms and working of the neurons better)
   - Define feedforward and backpropagation algorithms. Simply introducing the concept without going in too much detail to not confuse the audience.
3. Deep Learning Architectures (30 minutes)
   - Introduce popular deep learning architectures such as Convolutional Neural Networks (CNNs) for image processing and Recurrent Neural Networks (RNNs) for sequential data.
   - Simply introduce the components of each architecture. This could be too complex for the audience, so could be avoided.
   - Present real-world examples and applications for each architecture.
4. Training Deep Learning Models (25 minutes)
   - Explain the process of training deep learning models using labelled data. which should include the importance of data preprocessing, normalization, and regularization techniques.
   - Demonstrate the use of gradient descent optimization algorithms.
5. Deep Learning Tools and Frameworks (15 minutes)
   - Introduce popular deep learning frameworks like TensorFlow and PyTorch.
   - Discuss their advantages, features, and community support.
6. Q&A and Conclusion with Future Scope (No set time period. Can be extended to solve all queries)
   - Allow students to ask questions and clarify any doubts.
   - Summarize the key concepts covered in the session. Also discuss the future scope of Deep Learning, what is being done, what could be done and improved upon, etc.
   - Provide additional learning resources.

3. How would you troubleshoot a machine learning model whose performance is not as expected? Discuss your approach briefly.

Troubleshooting an underperforming machine learning model involves a systematic approach to identify the underlying issues and potential solutions. Troubleshooting can be carried out by:

1. Data Inspection:
   - Evaluate the quality and completeness of the training data.
   - Check for missing values, outliers, or class imbalance.
   - Ensure the data is correctly labelled and properly pre-processed.
2. Model Evaluation:

- Analyse the model's performance metrics (accuracy, precision, recall, etc.).
- Compare the metrics against the expected performance and baseline models.
- Identify if the model is overfitting or underfitting.

3. Error Analysis:
   - Examine the specific examples where the model is failing.
   - Look for patterns or common characteristics in the misclassified instances.
   - Determine if there are specific classes or features that pose challenges.
4. Model Improvement Strategies:
   - Collect more data, if possible, to increase the diversity and quantity of training samples.
   - Perform feature engineering to extract more meaningful features.
   - Experiment with different algorithms or model architectures.
   - Adjust hyperparameters (e.g., learning rate, regularization) through cross-validation.
5. Regularization Techniques:
   - Apply regularization methods such as L1 or L2 regularization to prevent overfitting.
   - Consider using dropout, which randomly sets a fraction of inputs to zero during training.
6. Ensemble Methods:
   - Explore ensemble techniques like bagging, boosting, or stacking.
   - Combine multiple models to leverage their collective predictive power.

4. Explain in simple terms what Natural Language Processing (NLP) is and its realworld applications.

Natural Language Processing (NLP) is a subfield of artificial intelligence that focuses on the interaction between computers and human language. It involves the development of algorithms and models to process, analyse, and understand human language in textual or spoken form. NLP has a wide range of real-world applications, including:

a) *Sentiment Analysis:* NLP can be used to determine the sentiment expressed in text, such as classifying movie reviews as positive or negative based on the language used.
b) *Machine Translation:* NLP techniques are employed in machine translation systems to automatically translate text from one language to another, enabling communication across different languages.
c) *Named Entity Recognition:* NLP can extract specific information from text, such as identifying names of people, organizations, locations, or dates, which is useful in various applications like information extraction or entity linking.
d) *Question Answering Systems:* NLP enables the development of question-answering systems that can understand natural language questions and provide relevant answers, like virtual assistants or chatbots.
e) *Text Summarization:* NLP techniques can automatically summarize large amounts of text, extracting the most important information and creating concise summaries.
f) *Speech Recognition:* NLP plays a crucial role in speech recognition systems, converting spoken language into written text, enabling applications like voice assistants and transcription services.
g) *Chatbots and Conversational Agents:* NLP is used to develop chatbots and conversational agents that can understand and respond to natural language inputs, enabling human-like interactions.

These are just a few examples of how NLP is used in various real-world applications, and as the field continues to advance, it offers new possibilities for language understanding and processing.

5. Write a SQL query to retrieve specific information from a relational database. The schema will be provided.

To write a SQL query to retrieve specific information from a relational database, one can follow the following step-by-step process to construct the query:

1. *Understand the Database Schema:*
   - Review the schema provided to understand the tables, their relationships, and the attributes/columns within each table.
   - Identify the relevant tables and their connections based on the specific information one wants to retrieve.

2. *Choose the Appropriate SQL Keywords:*
   - SELECT: Used to specify the columns or attributes to retrieve from the tables.
   - FROM: Specifies the table(s) from which one wants to retrieve the data from.
   - WHERE: Filters the rows based on specific conditions.
   - JOIN: Combines related tables based on their defined relationships.
   - GROUP BY: Groups the data based on specific attributes.
   - HAVING: Filters the groups based on conditions after the GROUP BY clause.
   - ORDER BY: Sorts the result set based on specified columns.
   - LIMIT or TOP: Restricts the number of rows returned by the query (syntax may vary depending on the database system).

3. *Construct the Query:*
   - Start with the SELECT keyword and specify the columns one wants to retrieve.
   - Use the FROM keyword followed by the table(s) involved in the query.
   - Apply any necessary JOIN clauses to connect related tables based on the relationships defined in the schema.
   - Add the WHERE clause to filter the rows based on specific conditions.
   - Include the GROUP BY clause if one wants to group the data.
   - Use the HAVING clause to filter the groups based on conditions.
   - Apply the ORDER BY clause to sort the result set if needed.
   - Finally, one can use the LIMIT or TOP clause to restrict the number of rows returned.

4. *Test and Execute the Query:*
   - Use a SQL client or an interface (e.g., MySQL Workbench, pgAdmin, or SQLiteStudio) to connect to the database.
   - Paste the constructed query into the query editor.
   - Execute the query and review the results to ensure the retrieved information matches one's expectations.

The actual syntax and specific keywords may vary slightly depending on the database management system one is using (e.g., MySQL, PostgreSQL, SQLite, etc.). It is a good

practice to refer to the documentation or resources specific to the database system one is working with to ensure accurate syntax and usage.

Example:

Let us assume we have a simple database with two tables: "Customers" and "Orders." The "Customers" table contains customer information, and the "Orders" table contains order details. The tables have the following structure:

Customers Table:

- customer_id (int) ○ customer_name (varchar) ○ email (varchar)
- age (int)

Orders Table:

- order_id (int) ○ customer_id (int) ○ order_date (date) ○ total_amount (decimal)

Now, let us say we want to retrieve the names and email addresses of all customers who have a total order amount greater than $100. We can write an SQL query for that:

```
SELECT customer_name, email
FROM Customers
WHERE customer_id IN (
  SELECT customer_id
  FROM Orders
  WHERE total_amount > 100
);
```

In this example query:

- We start with the SELECT keyword to specify the columns we want to retrieve (customer_name and email).
- We use the FROM keyword to indicate the "Customers" table from which we want to retrieve the data.
- The WHERE clause is used to filter the rows based on specific conditions. In this case, we use a subquery to retrieve the customer IDs who have placed an order with a total amount greater than $100.
- The subquery selects customer IDs from the "Orders" table, filtering on the total_amount condition.
- Finally, the main query selects the customer_name and email columns from the "Customers" table, using the WHERE clause to restrict the result to the customer IDs obtained from the subquery.

Executing this query will return the names and email addresses of customers who meet the specified criteria.