Question 1: What is Pandas, and why is it commonly used in data cleaning tasks?

Answer: Pandas is an open-source data manipulation and analysis library for Python. It provides data structures, such as Series and DataFrame, for efficiently handling and manipulating structured data. Pandas is commonly used in data cleaning tasks because it offers powerful and flexible tools for tasks such as handling missing data, filtering, grouping, merging, reshaping, and more. The DataFrame, in particular, is well-suited for representing and working with tabular data, making it an essential tool in the data cleaning and preparation process.

Question 2: Given a DataFrame with missing values, how would you check for missing values in each column and count the total number of missing values?

Answer: You can use the **isnull()** method to check for missing values in each column and the **sum()** method to count the total number of missing values in each column.

Assuming 'df' is your DataFrame

missing_values_per_column = df.isnull().sum()

total_missing_values = df.isnull().sum().sum()

print("Missing values per column:")

print(missing_values_per_column)

print("\nTotal number of missing values:", total_missing_values)

Question 3: How can you remove duplicates from a DataFrame while retaining the first occurrence of each unique row?

Answer: You can use the **drop_duplicates()** method to remove duplicates from a DataFrame. By default, this method keeps the first occurrence of each unique row.

df_no__duplicates = df.drop_duplicates()

Question 4: If you have a DataFrame with a column containing string values, how can you convert all the values in that column to lowercase?

Answer: You can use the **str.lower()** method to convert all the string values in a column to lowercase.

df['column_name'] = df['column_name'].str.lower()

Question 5: How do you replace missing values in a DataFrame with a specific value, like 0, for a particular column?

Answer: You can use the **fillna()** method to replace missing values in a DataFrame with a specific value.

Assuming 'df' is your DataFrame and 'column_name' is the column with missing values

df['column_name'].fillna(0, inplace=True)

Question 6: If you have a DataFrame with a datetime column, how can you extract the year, month, and day into separate columns?

Answer: You can use the **dt** accessor with the **year**, **month**, and **day** attributes to extract the corresponding components from a datetime column.

Assuming 'df' is your DataFrame and 'datetime_column' is the datetime column

df['year'] = df['datetime_column'].dt.year

df['month'] = df['datetime_column'].dt.month

df['day'] = df['datetime_column'].dt.day

Question 7: How can you filter rows in a DataFrame where a specific column's values meet a certain condition (e.g., all rows where 'age' is greater than 30)?

Answer: You can use boolean indexing to filter rows based on a specific condition.

Assuming 'df' is your DataFrame and 'age' is the column to filter

filtered_df = df[df['age'] > 30]

Question 8: What is the purpose of the .apply() function in Pandas, and how would you use it to create a new column based on values from existing columns?

Answer: The **.apply()** function in Pandas is used to apply a function along the axis of a DataFrame. It is often used to perform element-wise operations or transformations on DataFrame columns. You can use it to create a new column based on values from existing columns.

df['double_age'] = df['age'].apply(lambda x: x * 2)

Question 9: Suppose you want to merge two DataFrames, 'df1' and 'df2,' on a common column 'key.' How would you perform this merge operation in Pandas?

Answer: You can use the **merge()** function to merge two DataFrames on a common column.

 # Assuming 'df1' and 'df2' are your DataFrames

merged_df = pd.merge(df1, df2, on='key')

Question 10: You have a DataFrame with a column containing messy text data. How can you clean and standardize the text data (e.g., remove punctuation and convert to lowercase) in that column?

Answer: You can use the **str.replace()** method along with regular expressions to remove punctuation, and the **str.lower()** method to convert text to lowercase.

 # Assuming 'df' is your DataFrame and 'text_column' is the column with messy text data

df['text_column'] = df['text_column'].str.replace('[^\w\s]', '').str.lower()