Hbase Handler Class

```python
## Author: Goh Boon Xiang
import happybase
import uuid
from pyspark.sql import SparkSession, Row
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, ArrayType, DoubleType
from pyspark.ml.linalg import Vectors, VectorUDT

class HBaseHandler:
    """
    A class to handle operations with HBase.
    """
    def __init__(self, host='localhost', port=9090):
        """
        Initializes the HBase connection.

        Parameters:
        - host: str, HBase server host.
        - port: int, HBase server port.
        """
        self.host = host
        self.port = port
        self.connection = happybase.Connection(self.host, port=self.port)
        self.connection.open()

    def list_tables(self):
        """
        Lists all the tables in HBase.
        """
        print("Available tables:", self.connection.tables())
```

```python
    def delete_table(self, table_name):
        """
        Deletes a specified table from HBase.

        Parameters:
        - table_name: str, the name of the table to delete.
        """
        if table_name.encode('utf-8') in self.connection.tables():
            try:
                self.connection.disable_table(table_name)
                print(f"Table '{table_name}' disabled successfully.")
            except happybase.hbase.ttypes.IOError as e:
                if 'TableNotDisabledException' in str(e):
                    print(f"Table '{table_name}' is already disabled.")
                else:
                    raise e

            self.connection.delete_table(table_name)
            print(f"Table '{table_name}' deleted successfully.")
        else:
            print(f"Table '{table_name}' does not exist.")

        self.list_tables()
```

```python
    def create_table(self, table_name, families):
        """
        Creates a table in HBase.

        Parameters:
        - table_name: str, the name of the table to create.
        - families: dict, the column families to be created.
        """
        self.connection.create_table(table_name, families)
        print(f"Table '{table_name}' created successfully.")
        self.list_tables()

    def generate_row_key(self, row):
        """
        Generates a unique row key for storing in HBase.

        Parameters:
        - row: Row object containing the data.

        Returns:
        - str, the generated row key.
        """
        unique_id = uuid.uuid4()
        key_hash = hash((row['Sentiment'], row['Review']))
        return f"{unique_id}_{key_hash}"
```

```
    def save_to_hbase(self, df, table_name):
        """
        Saves data from a DataFrame to HBase.

        Parameters:
        - df: DataFrame, the data to be stored.
        - table_name: str, the name of the HBase table.
        """
        table = self.connection.table(table_name)
        total = 0

        for row in df.collect():
            row_key = self.generate_row_key(row)
            data_to_store = {
                b'cf1:Review': row['Review'].encode() if row['Review'] else b'',
                b'cf2:Sentiment': str(row['Sentiment']).encode() if row['Sentiment'] is not None else b'',
                b'cf3:SkuInfo_index': str(row['SkuInfo_index']).encode() if row['SkuInfo_index'] is not None else b'',
                b'cf4:tokens': ','.join(row['tokens']).encode() if row['tokens'] else b'',
                b'cf5:number_of_tokens': str(row['number_of_tokens']).encode() if row['number_of_tokens'] is not None else b'',

            }
            table.put(row_key.encode(), data_to_store)
            total += 1

        print(f'Data successfully stored in HBase with {total} records')
```

```
    def retrieve_from_hbase(self, table_name):
        """
        Retrieves data from HBase and returns it as a DataFrame.

        Parameters:
        - table_name: str, the name of the table to retrieve data from.

        Returns:
        - DataFrame, the retrieved data.
        """
        table = self.connection.table(table_name)
        rows = table.scan()

        data = []
        for key, row in rows:

            data.append(Row(

                Review=row[b'cf1:Review'].decode('utf-8') if b'cf1:Review' in row else None,
                Sentiment=int(row[b'cf2:Sentiment'].decode('utf-8')) if b'cf2:Sentiment' in row else None,
                SkuInfo_index=float(row[b'cf3:SkuInfo_index'].decode('utf-8')) if b'cf3:SkuInfo_index' in row else None,
                tokens=row[b'cf4:tokens'].decode('utf-8').split(',') if b'cf4:tokens' in row else None,
                number_of_tokens=int(row[b'cf5:number_of_tokens'].decode('utf-8')) if b'cf5:number_of_tokens' in row else None,
            ))

        schema = StructType([
            StructField("Review", StringType(), True),
            StructField("Sentiment", IntegerType(), True),
            StructField("SkuInfo_index", DoubleType(), True),
            StructField("tokens", ArrayType(StringType()), True),
            StructField("number_of_tokens", IntegerType(), True)

        ])

        spark = SparkSession.builder.appName("HBase Retrieval").getOrCreate()
        df = spark.createDataFrame(data, schema)

        return df

    def close(self):
        """
        Closes the HBase connection.
        """
        if self.connection is not None:
            self.connection.close()
```