```python
# Import necessary PySpark functions
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, to_date, lit, expr, countDistinct, when, date_add, date_sub, min as pyspark_min, max as pyspark_ma
from pyspark.sql.types import StructType, StructField, StringType, DateType, IntegerType
import os
import pandas as pd # For creating snapshot dates easily

# --- 0. Mount Google Drive (if using Google Colab) ---
try:
    from google.colab import drive
    drive.mount('/content/drive')
    print("Google Drive mounted successfully.")
    google_drive_base_path = '/content/drive/MyDrive/'
except ImportError:
    print("Not running in Google Colab or google.colab.drive module not found. Assuming local file system.")
    google_drive_base_path = ""

# Initialize SparkSession
spark = SparkSession.builder.appName("InactivityPatternAnalysis") \
    .config("spark.sql.legacy.timeParserPolicy", "LEGACY") \
    .getOrCreate()

# Define paths to data files
input_base_dir_drive = os.path.join(google_drive_base_path, 'Tables/')
login_data_dir_drive = os.path.join(google_drive_base_path, 'LOG_NEW/')

client_details_filename = "client_details.txt" # For ActivationDate if needed, though not strictly for this analysis if we just focus on
trade_data_filename = "trade_data.txt"
login_data_path_pattern = os.path.join(login_data_dir_drive, "LOGIN_*.txt")

# Paths (client_details might not be used heavily here but good to have)
client_details_path = os.path.join(input_base_dir_drive, client_details_filename)
trade_data_path = os.path.join(input_base_dir_drive, trade_data_filename)

print(f"Trade data path: {trade_data_path}")
print(f"Login data pattern: {login_data_path_pattern}")
```

```
Mounted at /content/drive
Google Drive mounted successfully.
Trade data path: /content/drive/MyDrive/Tables/trade_data.txt
Login data pattern: /content/drive/MyDrive/LOG_NEW/LOGIN_*.txt
```

```python
# --- Load Trade Data ---
# Header: CLIENTCODE,TRADE_DATE,TOTAL_GROSS_BROKERAGE_DAY
# Delimiter: comma (,)
# Date Format: dd/MM/yyyy
try:
    trades_df_raw = spark.read.format("csv") \
        .option("header", "true") \
        .option("delimiter", ",") \
        .load(trade_data_path)

    trades_df = trades_df_raw.select(
        col("CLIENTCODE").alias("ClientCode"),
        to_date(col("TRADE_DATE"), "dd/MM/yyyy").alias("ActivityDate")
    ).filter(col("ActivityDate").isNotNull()) \
     .distinct() # Distinct ClientCode, ActivityDate pairs for trades

    trades_df.persist() # Persist for multiple uses
    print("Trade data loaded and processed (distinct ClientCode, ActivityDate):")
    trades_df.show(5, truncate=False)
    print(f"Total distinct trade day records: {trades_df.count()}")

except Exception as e:
    print(f"Error loading trade_data.txt: {e}")
    # spark.stop()
    # exit()
```

```
Trade data loaded and processed (distinct ClientCode, ActivityDate):
+----------+------------+
|ClientCode|ActivityDate|
+----------+------------+
|PA3459    |2020-08-04  |
|RP7880    |2020-08-07  |
|PP7043    |2020-08-07  |
|MG12407   |2020-08-07  |
|N105280   |2020-08-07  |
+----------+------------+
only showing top 5 rows

Total distinct trade day records: 17254800
```

```
# --- Load Login Data ---
# Format: ClientCode,DD/MM/YYYY (no header)
try:
    login_schema = StructType([
        StructField("ClientCode_raw", StringType(), True),
        StructField("LoginDate_str", StringType(), True)
    ])

    logins_df_raw = spark.read.format("csv") \
        .schema(login_schema) \
        .option("delimiter", ",") \
        .load(login_data_path_pattern)

    logins_df = logins_df_raw.select(
        col("ClientCode_raw").alias("ClientCode"),
        to_date(col("LoginDate_str"), "dd/MM/yyyy").alias("ActivityDate")
    ).filter(col("ActivityDate").isNotNull()) \
     .distinct() # Distinct ClientCode, ActivityDate pairs for logins

    logins_df.persist() # Persist for multiple uses
    print("Login data loaded and processed (distinct ClientCode, ActivityDate):")
    logins_df.show(5, truncate=False)
    print(f"Total distinct login day records: {logins_df.count()}")

except Exception as e:
    print(f"Error loading login data: {e}")
    # spark.stop()
    # exit()
```

```
Login data loaded and processed (distinct ClientCode, ActivityDate):
+----------+------------+
|ClientCode|ActivityDate|
+----------+------------+
|GA5091    |2023-07-03  |
|SS24660   |2023-07-03  |
|HM006     |2023-07-03  |
|RB5800    |2023-07-03  |
|TG1522    |2023-07-03  |
+----------+------------+
only showing top 5 rows

Total distinct login day records: 39125229
```

```
# --- Determine Overall Data Date Range and Generate Snapshot Dates ---

# To determine a reasonable snapshot range, find min/max dates from activity data
if 'trades_df' in locals() and 'logins_df' in locals():
    min_max_trade_dates = trades_df.agg(
        pyspark_min("ActivityDate").alias("MinTradeDate"),
        pyspark_max("ActivityDate").alias("MaxTradeDate")
    ).first()

    min_max_login_dates = logins_df.agg(
        pyspark_min("ActivityDate").alias("MinLoginDate"),
        pyspark_max("ActivityDate").alias("MaxLoginDate")
    ).first()

    overall_min_date = None
    overall_max_date = None

    if min_max_trade_dates and min_max_trade_dates["MinTradeDate"]:
        overall_min_date = min_max_trade_dates["MinTradeDate"]
    if min_max_login_dates and min_max_login_dates["MinLoginDate"]:
        if overall_min_date is None or min_max_login_dates["MinLoginDate"] < overall_min_date:
            overall_min_date = min_max_login_dates["MinLoginDate"]

    if min_max_trade_dates and min_max_trade_dates["MaxTradeDate"]:
        overall_max_date = min_max_trade_dates["MaxTradeDate"]
    if min_max_login_dates and min_max_login_dates["MaxLoginDate"]:
        if overall_max_date is None or min_max_login_dates["MaxLoginDate"] > overall_max_date:
            overall_max_date = min_max_login_dates["MaxLoginDate"]

    print(f"Overall Min Activity Date: {overall_min_date}")
    print(f"Overall Max Activity Date: {overall_max_date}")

    # Define snapshot period
    if overall_max_date:
        snapshot_start_date = pd.to_datetime("2021-01-01")
        max_prediction_window = 365
        snapshot_end_date = pd.to_datetime(overall_max_date) - pd.Timedelta(days=max_prediction_window)

        if snapshot_end_date < snapshot_start_date:
```

```
            print(f"Warning: Snapshot end date ({snapshot_end_date}) is before start date ({snapshot_start_date}). Adjusting or aborting
            # Handle this case appropriately if it occurs in different data
            snapshots_df = None
        else:
            print(f"Snapshot Start Date: {snapshot_start_date.strftime('%Y-%m-%d')}")
            print(f"Snapshot End Date (calculated): {snapshot_end_date.strftime('%Y-%m-%d')}")

            # Generate monthly snapshot dates (end of month)
            # Using 'ME' for month-end as 'M' is deprecated
            snapshot_dates_pd = pd.date_range(start=snapshot_start_date, end=snapshot_end_date, freq='ME')
            snapshot_dates_list = [(d.strftime('%Y-%m-%d'),) for d in snapshot_dates_pd]

            if snapshot_dates_list:
                snapshots_df = spark.createDataFrame(snapshot_dates_list, ["SnapshotDate_str"])
                snapshots_df = snapshots_df.withColumn("SnapshotDate", to_date(col("SnapshotDate_str"), "yyyy-MM-dd")) \
                                           .select("SnapshotDate")
                if snapshots_df.count() > 0: # Check if snapshots_df is not empty
                    snapshots_df.persist()
                    print(f"\nGenerated {snapshots_df.count()} snapshot dates:")
                    snapshots_df.orderBy("SnapshotDate").show(5)
                    snapshots_df.orderBy(col("SnapshotDate").desc()).show(5)
                else:
                    print("No snapshot dates generated (empty list). Check date ranges and logic.")
                    snapshots_df = None
            else:
                print("No snapshot dates generated (empty list). Check date ranges.")
                snapshots_df = None
    else:
        print("Could not determine overall_max_date. Cannot generate snapshots.")
        snapshots_df = None
else:
    print("Skipping snapshot generation as trades_df or logins_df is missing.")
    snapshots_df = None
```

```
Overall Min Activity Date: 2020-08-03
Overall Max Activity Date: 2024-04-30
Snapshot Start Date: 2021-01-01
Snapshot End Date (calculated): 2023-05-01

Generated 28 snapshot dates:
+------------+
|SnapshotDate|
+------------+
|  2021-01-31|
|  2021-02-28|
|  2021-03-31|
|  2021-04-30|
|  2021-05-31|
+------------+
only showing top 5 rows

+------------+
|SnapshotDate|
+------------+
|  2023-04-30|
|  2023-03-31|
|  2023-02-28|
|  2023-01-31|
|  2022-12-31|
+------------+
only showing top 5 rows
```

```
# --- Phase 2: Generate Client-Snapshot Base & Calculate Forward Activity ---

if 'trades_df' in locals() and 'logins_df' in locals() and snapshots_df is not None and snapshots_df.count() > 0:
    # Get all unique clients from trades and logins
    all_clients_trades_df = trades_df.select("ClientCode").distinct()
    all_clients_logins_df = logins_df.select("ClientCode").distinct()

    client_universe_df = all_clients_trades_df.unionByName(all_clients_logins_df).distinct()
    client_universe_df.persist()

    print(f"Total unique clients in universe: {client_universe_df.count()}")

    # Cross join client universe with snapshot dates to create the base ABT structure
    # Each client will have a row for each snapshot date
    client_snapshot_base_df = client_universe_df.crossJoin(snapshots_df)
    client_snapshot_base_df.persist()

    print(f"Total client-snapshot records: {client_snapshot_base_df.count()}")
    client_snapshot_base_df.show(5, truncate=False)
else:
    print("Skipping client-snapshot base generation due to missing DataFrames (trades, logins, or snapshots).")
```

```
Total unique clients in universe: 358755
Total client-snapshot records: 10045140
+----------+------------+
|ClientCode|SnapshotDate|
+----------+------------+
|KS11754   |2021-01-31  |
|KS11754   |2021-02-28  |
|KS11754   |2021-03-31  |
|KS11754   |2021-04-30  |
|KS11754   |2021-05-31  |
+----------+------------+
only showing top 5 rows
```

```
if 'client_snapshot_base_df' in locals() and client_snapshot_base_df.is_cached:

    n_day_windows = [60, 90, 270, 365]

    # Alias dataframes for join clarity BEFORE starting the loop
    cs_df_aliased = client_snapshot_base_df.alias("cs") # Alias the base client_snapshot
    t_df_aliased = trades_df.alias("trades")
    l_df_aliased = logins_df.alias("logins")

    # Initialize the DataFrame to which we'll add feature columns
    activity_features_df = client_snapshot_base_df # Start with the original unaliased one for the final result

    for n in n_day_windows:
        print(f"\nCalculating forward activity for {n}-day window...")

        # --- Forward Trade Days ---
        # Use aliased cs_df_aliased for groupBy to avoid ambiguity with the cs_df in the join
        forward_trades_count_df = cs_df_aliased.join(
            t_df_aliased,
            (col("cs.ClientCode") == col("trades.ClientCode")) & \
            (col("trades.ActivityDate") > col("cs.SnapshotDate")) & \
            (col("trades.ActivityDate") <= date_add(col("cs.SnapshotDate"), n)),
            "left"
        ).groupBy(col("cs.ClientCode"), col("cs.SnapshotDate")) \
         .agg(countDistinct(col("trades.ActivityDate")).alias(f"Trade_Days_In_FWD_{n}D"))

        # --- Forward Login Days ---
        forward_logins_count_df = cs_df_aliased.join(
            l_df_aliased,
            (col("cs.ClientCode") == col("logins.ClientCode")) & \
            (col("logins.ActivityDate") > col("cs.SnapshotDate")) & \
            (col("logins.ActivityDate") <= date_add(col("cs.SnapshotDate"), n)),
            "left"
        ).groupBy(col("cs.ClientCode"), col("cs.SnapshotDate")) \
         .agg(countDistinct(col("logins.ActivityDate")).alias(f"Login_Days_In_FWD_{n}D"))

        # Join these counts back to the main features DataFrame
        # When joining back, ensure keys are unambiguous.
        # activity_features_df has 'ClientCode' and 'SnapshotDate'
        # forward_trades_count_df has 'ClientCode' (from cs.ClientCode) and 'SnapshotDate' (from cs.SnapshotDate)

        activity_features_df = activity_features_df.join(
            forward_trades_count_df,
            # Specify join condition explicitly if column names are identical and from different sources
            (activity_features_df.ClientCode == forward_trades_count_df.ClientCode) & \
            (activity_features_df.SnapshotDate == forward_trades_count_df.SnapshotDate),
            "left"
```

```
        ).drop(forward_trades_count_df.ClientCode).drop(forward_trades_count_df.SnapshotDate) # Drop redundant key columns from right DF

        activity_features_df = activity_features_df.join(
            forward_logins_count_df,
            (activity_features_df.ClientCode == forward_logins_count_df.ClientCode) & \
            (activity_features_df.SnapshotDate == forward_logins_count_df.SnapshotDate),
            "left"
        ).drop(forward_logins_count_df.ClientCode).drop(forward_logins_count_df.SnapshotDate) # Drop redundant key columns

        # Fill NA for counts with 0
        activity_features_df = activity_features_df.fillna(0, subset=[f"Trade_Days_In_FWD_{n}D", f"Login_Days_In_FWD_{n}D"])

    activity_features_df.persist()
    print("\nClient-snapshot data with forward activity counts:")
    # Ensure columns are what we expect before showing
    expected_cols = ["ClientCode", "SnapshotDate"] + \
                    [f"Trade_Days_In_FWD_{n}D" for n in n_day_windows] + \
                    [f"Login_Days_In_FWD_{n}D" for n in n_day_windows]
    activity_features_df.select(expected_cols).show(10, truncate=False)

    print(f"Total records in activity_features_df: {activity_features_df.count()}")
    print(f"Columns in activity_features_df: {activity_features_df.columns}")


    # Unpersist intermediate DFs
    if 'client_universe_df' in locals() and client_universe_df.is_cached:
        client_universe_df.unpersist()
    if 'client_snapshot_base_df' in locals() and client_snapshot_base_df.is_cached:
        client_snapshot_base_df.unpersist()
    if 'trades_df' in locals() and trades_df.is_cached:
        trades_df.unpersist()
    if 'logins_df' in locals() and logins_df.is_cached:
        logins_df.unpersist()
else:
    print("Skipping forward activity calculation as client_snapshot_base_df is missing or not cached.")
```

```
Calculating forward activity for 60-day window...

Calculating forward activity for 90-day window...

Calculating forward activity for 270-day window...

Calculating forward activity for 365-day window...

Client-snapshot data with forward activity counts:
+----------+------------+--------------------+--------------------+---------------------+---------------------+----------------
|ClientCode|SnapshotDate|Trade_Days_In_FWD_60D|Trade_Days_In_FWD_90D|Trade_Days_In_FWD_270D|Trade_Days_In_FWD_365D|Login_Days_In_FWI
+----------+------------+--------------------+--------------------+---------------------+---------------------+----------------
|100319    |2021-09-30  |0                   |0                   |0                    |0                    |0
|100705    |2021-11-30  |0                   |0                   |0                    |0                    |0
|103219    |2021-04-30  |0                   |0                   |0                    |0                    |0
|104408    |2022-01-31  |0                   |0                   |0                    |0                    |0
|106286    |2021-07-31  |0                   |0                   |0                    |0                    |36
|106293    |2021-02-28  |0                   |0                   |0                    |0                    |0
|106297    |2022-06-30  |0                   |0                   |0                    |0                    |0
|108713    |2021-01-31  |0                   |0                   |0                    |0                    |8
|109001    |2021-01-31  |0                   |0                   |0                    |0                    |0
|111632    |2022-01-31  |0                   |0                   |0                    |0                    |0
+----------+------------+--------------------+--------------------+---------------------+---------------------+----------------
only showing top 10 rows

Total records in activity_features_df: 10045140
Columns in activity_features_df: ['ClientCode', 'SnapshotDate', 'Trade_Days_In_FWD_60D', 'Login_Days_In_FWD_60D', 'Trade_Days_In_FWI
```

```
# --- Phase 3: Categorize Inactivity and Analyze ---

if 'activity_features_df' in locals() and activity_features_df.is_cached: # Ensure it exists and was persisted

    n_day_windows = [60, 90, 270, 365]
    categorized_df = activity_features_df # Start with the df containing forward counts

    for n in n_day_windows:
        trade_fwd_col = f"Trade_Days_In_FWD_{n}D"
        login_fwd_col = f"Login_Days_In_FWD_{n}D"
        category_col = f"Inactivity_Category_{n}D"

        categorized_df = categorized_df.withColumn(
            category_col,
            when((col(trade_fwd_col) == 0) & (col(login_fwd_col) > 0), "Stopped_Trading_Only")
            .when((col(trade_fwd_col) > 0) & (col(login_fwd_col) == 0), "Stopped_Logging_In_Only")
            .when((col(trade_fwd_col) == 0) & (col(login_fwd_col) == 0), "Stopped_Both")
            .when((col(trade_fwd_col) > 0) & (col(login_fwd_col) > 0), "Remained_Active_Both")
```

```
            .otherwise("Error_Categorizing") # Should not happen if counts are always >= 0
        )

    categorized_df.persist()
    print("\nClient-snapshot data with inactivity categories:")

    # Select a subset of columns for display to keep it readable
    display_cols = ["ClientCode", "SnapshotDate"] + \
                   [f"Trade_Days_In_FWD_{n}D" for n in [60,365]] + \
                   [f"Login_Days_In_FWD_{n}D" for n in [60,365]] + \
                   [f"Inactivity_Category_{n}D" for n in n_day_windows]

    categorized_df.select(display_cols).show(15, truncate=False)

    # Unpersist the previous df if it's different
    if activity_features_df is not categorized_df and activity_features_df.is_cached:
         activity_features_df.unpersist()
else:
    print("Skipping inactivity categorization as activity_features_df is missing or not cached.")
```

```
Client-snapshot data with inactivity categories:
+----------+------------+--------------------+---------------------+--------------------+---------------------+----------------
|ClientCode|SnapshotDate|Trade_Days_In_FWD_60D|Trade_Days_In_FWD_365D|Login_Days_In_FWD_60D|Login_Days_In_FWD_365D|Inactivity_Catego
+----------+------------+--------------------+---------------------+--------------------+---------------------+----------------
|100319    |2021-09-30  |0                   |0                    |0                   |0                    |Stopped_Both
|100705    |2021-11-30  |0                   |0                    |0                   |1                    |Stopped_Both
|103219    |2021-04-30  |0                   |0                    |0                   |0                    |Stopped_Both
|104408    |2022-01-31  |0                   |0                    |0                   |188                  |Stopped_Both
|106286    |2021-07-31  |0                   |0                    |36                  |231                  |Stopped_Trading_(
|106293    |2021-02-28  |0                   |0                    |0                   |0                    |Stopped_Both
|106297    |2022-06-30  |0                   |0                    |0                   |110                  |Stopped_Both
|108713    |2021-01-31  |0                   |0                    |8                   |187                  |Stopped_Trading_(
|109001    |2021-01-31  |0                   |0                    |0                   |0                    |Stopped_Both
|111632    |2022-01-31  |0                   |0                    |0                   |0                    |Stopped_Both
|112602    |2022-03-31  |0                   |0                    |39                  |250                  |Stopped_Trading_(
|113304    |2021-03-31  |0                   |0                    |2                   |28                   |Stopped_Trading_(
|114212    |2023-04-30  |0                   |0                    |0                   |0                    |Stopped_Both
|118505    |2022-09-30  |0                   |0                    |35                  |223                  |Stopped_Trading_(
|118535    |2021-08-31  |0                   |0                    |23                  |23                   |Stopped_Trading_(
+----------+------------+--------------------+---------------------+--------------------+---------------------+----------------
only showing top 15 rows
```

```
import pyspark.sql.functions as F # Import F for convenience if using many functions

if 'categorized_df' in locals() and categorized_df.is_cached:

    n_day_windows_analysis = [60, 90, 270, 365] # Can be a subset if needed for quicker analysis
    overall_summary_list = []

    print("\n--- Proportions of Inactivity Categories ---")
    for n in n_day_windows_analysis:
        category_col = f"Inactivity_Category_{n}D"

        print(f"\n--- Analysis for {n}-Day Window ---")

        # Count occurrences of each category for the current N-day window
        category_counts_df = categorized_df.groupBy(category_col).count()

        # Calculate total snapshots for percentage calculation
        # This assumes categorized_df contains all snapshots.
        # If we filtered it (e.g. for clients with prior activity), this total might need adjustment.
        # For now, using the count of the categorized_df.
        total_snapshots_for_n = categorized_df.select("SnapshotDate", "ClientCode").distinct().count() # Should be same as categorized

        print(f"Total unique Client-Snapshot pairs considered for {n}D: {total_snapshots_for_n}")
        category_counts_df = category_counts_df.withColumn(
            "Percentage",
            (F.col("count") / F.lit(total_snapshots_for_n)) * 100
        )

        category_counts_df.show(truncate=False)

        # Store for overall summary (optional)
        # For a more structured summary, you might pivot this or collect results
        # For example, collecting to a list of dictionaries:
        for row in category_counts_df.collect():
            overall_summary_list.append({
                "N_Day_Window": n,
                "Category": row[category_col],
                "Count": row["count"],
                "Percentage": row["Percentage"]
```

```
            })

        # Display overall summary if created
        if overall_summary_list:
            overall_summary_spark_df = spark.createDataFrame(pd.DataFrame(overall_summary_list))
            print("\n--- Overall Summary of Inactivity Categories ---")
            overall_summary_spark_df.orderBy("N_Day_Window", "Category").show(truncate=False)

        if categorized_df.is_cached:
            categorized_df.unpersist()
    else:
        print("Skipping aggregation as categorized_df is missing or not cached.")

    # Stop Spark Session
    spark.stop()
```

```
--- Proportions of Inactivity Categories ---

--- Analysis for 60-Day Window ---
Total unique Client-Snapshot pairs considered for 60D: 10045140
+-----------------------+-------+------------------+
|Inactivity_Category_60D|count  |Percentage        |
+-----------------------+-------+------------------+
|Stopped_Both           |6828806|67.98119289527075 |
|Stopped_Trading_Only   |1052142|10.474139733244137|
|Stopped_Logging_In_Only|375079 |3.7339350173317647|
|Remained_Active_Both   |1789113|17.81073235415335 |
+-----------------------+-------+------------------+


--- Analysis for 90-Day Window ---
Total unique Client-Snapshot pairs considered for 90D: 10045140
+-----------------------+-------+------------------+
|Inactivity_Category_90D|count  |Percentage        |
+-----------------------+-------+------------------+
|Stopped_Both           |6417198|63.88360938722606 |
|Stopped_Trading_Only   |1131993|11.269061456584975|
|Stopped_Logging_In_Only|421094 |4.192017234204799 |
|Remained_Active_Both   |2074855|20.65531192198416 |
+-----------------------+-------+------------------+


--- Analysis for 270-Day Window ---
Total unique Client-Snapshot pairs considered for 270D: 10045140
+------------------------+-------+------------------+
|Inactivity_Category_270D|count  |Percentage        |
+------------------------+-------+------------------+
|Stopped_Both            |4841854|48.20096086266592 |
|Stopped_Trading_Only    |1426598|14.201872746422648|
|Stopped_Logging_In_Only |576775 |5.7418313731814585|
|Remained_Active_Both    |3199913|31.85533501772997 |
+------------------------+-------+------------------+


--- Analysis for 365-Day Window ---
Total unique Client-Snapshot pairs considered for 365D: 10045140
+------------------------+-------+------------------+
|Inactivity_Category_365D|count  |Percentage        |
+------------------------+-------+------------------+
|Stopped_Both            |4285328|42.660709557059434|
|Stopped_Trading_Only    |1538507|15.31593387449055 |
|Stopped_Logging_In_Only |606320 |6.03595370497574  |
|Remained_Active_Both    |3614985|35.98740286347428 |
+------------------------+-------+------------------+


--- Overall Summary of Inactivity Categories ---
+------------+-----------------------+-------+------------------+
|N_Day_Window|Category               |Count  |Percentage        |
+------------+-----------------------+-------+------------------+
|60          |Remained_Active_Both   |1789113|17.81073235415335 |
|60          |Stopped_Both           |6828806|67.98119289527075 |
```