ASSIGNMENT: BUILDING AN END-TO-END DATA PIPELINE

COURSE: FUNDAMENTALS OF BIG DATA ANALYTICS AND BI

DEPARTMENT OF SOFTWARE ENGFINEERING

NAME: ASAYE CHALA
ID: 1579/13

Catalog

Introduction

This report documents the development of an end-to-end data pipeline(Building an End-to-End Data Pipeline) .The primary objective of this project is to gain practical experience in designing and implementing a complete data pipeline, from data extraction to visualization, using real-world data sources. The pipeline involves extracting data from diverse sources, transforming it into a usable format, storing it in a relational database, and visualizing it to derive meaningful insights.

The project is structured around the ETL (Extract, Transform, Load) process, which is a fundamental concept in data engineering and analytics. Specifically, the pipeline includes:

- Data Extraction: Data was collected from two primary sources: a large e-commerce-related dataset downloaded from Kaggle and Telegram channels related to the e-commerce sector in Ethiopia. The Telegram data was scraped using the Telethon library, while the Kaggle dataset was loaded using pandas.
- Data Transformation: The raw data was cleaned, preprocessed, and transformed to ensure consistency and usability. This included handling missing values, removing duplicates, standardizing data types, and performing sentiment analysis on the Telegram data using the TextBlob library.
- Data Loading: The processed data was stored in a PostgreSQL database, with a carefully designed schema to enable efficient querying and cross-analysis of the two datasets.
- Data Visualization: Finally, the data was visualized using Microsoft Power BI to create interactive data

1. Data Source Identification & Understanding

1.1 Large Dataset (E-commerce Data)

- Dataset Source: The dataset was sourced from Kaggle, containing over 1 million rows of e-commerce transaction data.
- Dataset source link: https://www.kaggle.com/datasets/yelp-dataset/yelp-dataset

- Dataset Structure: The dataset is in CSV format, with fields such as Rating, NumberReview,Organization, Country,category , CountryCode,State,Street and Building.
- Understanding: The dataset provides insights into customer purchasing behavior, sales trends, and product performance.

Field Descriptions:

- Phone: The phone number associated with the business being reviewed.
- Organization: The name of the business or organization being reviewed.
- Rating: The rating given by the reviewer (usually on a scale of 1 to 5).
- NumberReview: The number of reviews the business has received.
- Category: The type of service being reviewed (e.g., "Delivery").
- Country: The country where the business is located (in this case, "USA").
- CountryCode: The ISO country code (for the USA, it is "US").
- State: The state where the business is located (e.g., "AL" for Alabama).
- City: The city where the business is located (e.g., "Alexander City").
- Street: The street name where the business is located.
- Building: The building number or address of the business.

Relationships Between Fields:

- Organization links directly to the Rating, NumberReview, and Category, as each business gets ratings and reviews based on its service or product offering.
- Location fields (Country, CountryCode, State, City, Street, Building) help geographically categorize the business, allowing you to filter or segment data based on geography.
- The Phone field might be relevant for contacting a business but doesn't directly relate to reviews or ratings.

By organizing this I performed:

- Business popularity by examining the NumberReview and Rating.
- Geographical trends by analyzing businesses in specific States or Cities.
- Category-based analysis to compare businesses in different industries.

1.2 Telegram Channels (E-commerce Related)

- Channels Identified: Three Ethiopian Telegram channels were identified, discussing e-commerce topics such as product reviews, customer feedback, and market trends.
    - ✓ Delivery Addis: Focuses on food delivery services in Addis Ababa.
    - ✓ FoodInEthiopia1: Discusses food-related e-commerce and customer experiences.
    - ✓ Ahatfood943024546: A channel for restaurant reviews and food delivery services.
- Data Extracted: Using the Telethon library, text data, timestamps, and user information were scraped from these channels. The data includes user-generated content, which was later analyzed for sentiment.

2. Data Extraction

2.1 Large Dataset Extraction

Tool Used: pandas for reading the CSV file.

Code used:

```
# data loading
import pandas as pd
ecommerce_data = pd.read_csv('yelp_database.csv')
print(ecommerce_data.head())
```

2.2 Telegram Data Extraction

Tool Used: Telethon for connecting to the Telegram API and extracting messages.

Code used:

```
import pandas as pd
from telethon import TelegramClient
from aiogoogletrans import Translator
#logging info
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

# Create the Telethon client
client = TelegramClient('session_name', api_id, api_hash)

# Translator for Amharic to English
translator = Translator()
async def scrape_telegram_channel(channel_name):
```

```
    """Scrape messages from a Telegram channel."""
    try:
        logging.info(f"Scraping data from {channel_name}...")
        channel = await client.get_entity(channel_name)
        messages = await client.get_messages(channel, limit=100)

        data = []
        for message in messages:
            if message.text:
                original_text = message.text
                translated_text = await translate_text(original_text)  # ✅ Await translation

                data.append({
                    'text': translated_text,
                    'original_text': original_text,
                    'timestamp': message.date,
                    'channel': channel_name
                })
                await asyncio.sleep(2)  # Delay to avoid rate limiting
        df = pd.DataFrame(data)
        df = clean_telegram_data(df)
        df.to_csv(f'{channel_name}_data.csv', index=False)
        save_to_database(data)  # Save to database
        logging.info(f"✅ Data from {channel_name} saved to {channel_name}_data.csv and
database.")
    except Exception as e:
        logging.error(f"❌ Error scraping {channel_name}: {e}")
```

3. Data Transformation

3.1 Data Cleaning

- Handling Missing Values: Replacing missing numerical fields with defaults or dropping incomplete rows.
- Removing Duplicates: Identifying and eliminating duplicate records.
- Standardizing Data Types: Converting dates to datetime format and normalizing numerical values.

Code used:

```
# Convert Time_GMT to datetime format, handling errors
df['Time_GMT'] = pd.to_datetime(df['Time_GMT'], errors='coerce')

# Replace empty strings with None
df.replace('', None, inplace=True)
# Drop rows with missing critical data AFTER type conversion
df.dropna(subset=['Time_GMT', 'Phone', 'Organization'], inplace=True)
# Ensure numeric columns are valid, handling errors
df['Rating'] = pd.to_numeric(df['Rating'], errors='coerce')
df['NumberReview'] = pd.to_numeric(df['NumberReview'], errors='coerce')
#*CRITICAL: Clean up whitespace from string columns using .location
```

```
        for col in ['Organization', 'Category', 'Country', 'CountryCode', 'State', 'City',
'Street', 'Building', 'Phone']:
            if df_to_insert[col].dtype == 'object':
                df_to_insert.loc[:, col] = df_to_insert[col].str.strip()

        #CRITICAL: Handle NaT values in Time_GMT
        df_to_insert.loc[:, 'Time_GMT'] = df_to_insert['Time_GMT'].fillna(pd.NaT).apply(lambda
x: None if pd.isna(x) else x)
        data_tuples = [tuple(row) for row in df_to_insert.to_numpy()]
# Remove exact duplicates (where all columns are the same)
df = df.drop_duplicates()

# Remove duplicates based on key columns (Phone, Organization, Time_GMT)
df = df.drop_duplicates(subset=['Time_GMT', 'Phone', 'Organization'])
cleaned_file_path = "cleaned_dataset.csv"
df_cleaned = pd.read_csv("yelp_database.csv")

# Get the number of records
num_records = df_cleaned.shape[0]
print(f"  The cleaned dataset contains {num_records} records.")
print(f"✅ Duplicate records removed. Cleaned dataset saved to {cleaned_file_path}.")
```

3.2 Sentiment Analysis on Telegram Data

Tool Used: The TextBlob library was used to perform sentiment analysis on the text data scraped from Telegram channels.

Code Used:

```
from textblob import TextBlob
# Perform sentiment analysis
def get_sentiment(text):
    analysis = TextBlob(text)
    return analysis.sentiment.polarity
# Apply sentiment analysis to Telegram messages
telegram_data['sentiment'] = telegram_data['text'].apply(get_sentiment)
```

4. Data Loading

4.1 Database Schema Design

Database: PostgreSQL was used as the relational database.

- Database: PostgreSQL.
- Tables Created:

- ecommerce_transactions: Stores transaction data.
- telegram_messages: Stores messages and sentiment scores.
- Relationships: customer_id links e-commerce data with Telegram user data for cross-analysis.

Tables Created:

- yelp_reviews: Stores Yelp review data.
- telegram_messages: Stores Telegram message data, including sentiment scores.

4.2 Loading Data into PostgreSQL

Tool Used: psycopg2 for database interaction.

Code Used:

```
import pandas as pd
import psycopg2
import logging
from psycopg2 import extras
# PostgreSQL Database Credentials
DB_CONFIG = {
    'dbname': 'yelp_database',
    'user': 'postgres',
    'password': 'hellopostgress',  # Replace with your actual password
    'host': 'localhost',
    'port': '5432'
}
# Load the cleaned dataset
file_path = "Clean_dataset.csv"
df = pd.read_csv(file_path, dtype={'Phone': str})

# Convert Time_GMT to datetime format, handling errors
df['Time_GMT'] = pd.to_datetime(df['Time_GMT'], errors='coerce')
# Replace empty strings with None
df.replace('', None, inplace=True)
# Drop rows with missing critical data AFTER type conversion
df.dropna(subset=['Time_GMT', 'Phone', 'Organization'], inplace=True)
# Ensure numeric columns are valid, handling errors
df['Rating'] = pd.to_numeric(df['Rating'], errors='coerce')
df['NumberReview'] = pd.to_numeric(df['NumberReview'], errors='coerce')
def connect_db():
    """Establish a connection to PostgreSQL."""
    return psycopg2.connect(**DB_CONFIG)
def insert_data():
    conn = None
    cursor = None
    try:
        conn = connect_db()
        cursor = conn.cursor()
        insert_query = """
        INSERT INTO yelp_reviews (time_gmt, phone, organization, rating, number_review,
```

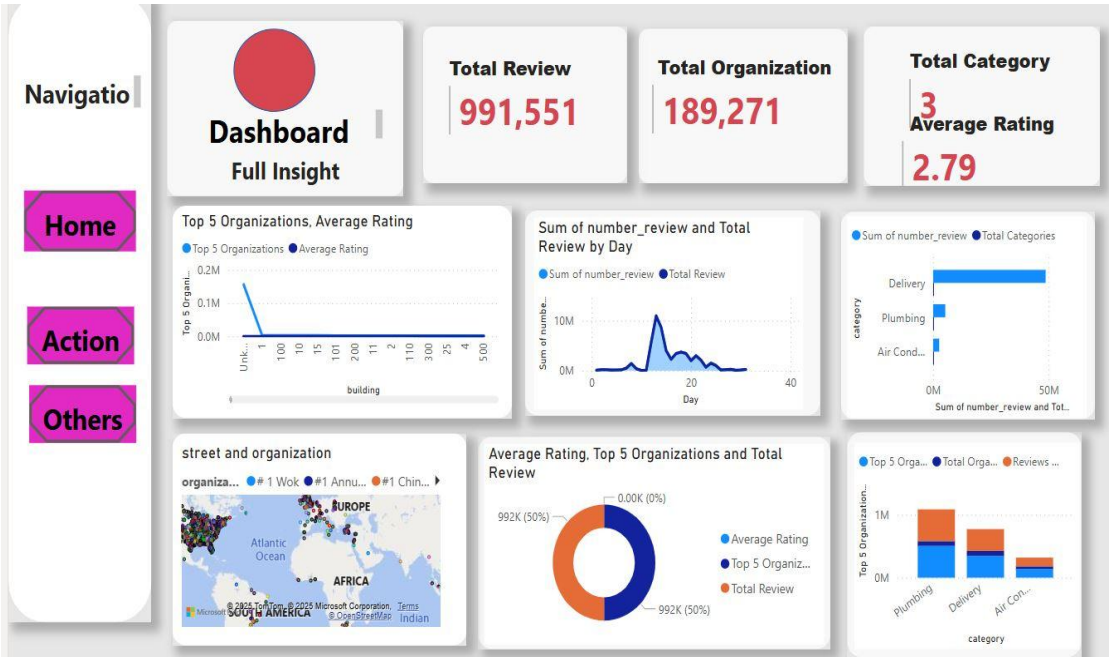
```
                                category, country, country_code, state, city, street,
building)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
        """
        required_columns = ['Time_GMT', 'Phone', 'Organization', 'Rating', 'NumberReview',
                            'Category', 'Country', 'CountryCode', 'State', 'City', 'Street',
'Building']
        try:
            df_to_insert = df[required_columns]
        except KeyError as e:
            logging.error(f"❌ Column missing in DataFrame: {e}. DataFrame columns are:
{df.columns.tolist()}")
            return
        # ***CRITICAL: Clean up whitespace from string columns using .loc***
        for col in ['Organization', 'Category', 'Country', 'CountryCode', 'State', 'City',
'Street', 'Building', 'Phone']:
            if df_to_insert[col].dtype == 'object':
                df_to_insert.loc[:, col] = df_to_insert[col].str.strip()
        # ***CRITICAL: Handle NaT values in Time_GMT using .loc***
        df_to_insert.loc[:, 'Time_GMT'] = df_to_insert['Time_GMT'].fillna(pd.NaT).apply(lambda
x: None if pd.isna(x) else x)
        data_tuples = [tuple(row) for row in df_to_insert.to_numpy()]
        # Debugging prints:
        num_tuples_to_print = min(5, len(data_tuples)) if len(data_tuples) > 0 else 0  # Check
for empty data_tuples
        for i in range(num_tuples_to_print):
            print(f"Tuple {i+1}: {data_tuples[i]}, Length: {len(data_tuples[i])}")
        expected_values_per_tuple = insert_query.count('%s')
        print(f"Expected values per tuple: {expected_values_per_tuple}")
        # ***KEY CHANGE: Batch size for execute_batch***
        batch_size = 10000  # Adjust as needed
        for i in range(0, len(data_tuples), batch_size):
            batch = data_tuples[i:i + batch_size]
            try:
                extras.execute_batch(cursor, insert_query, batch)
                print(f"Inserted batch {i//batch_size + 1} of {len(data_tuples)//batch_size +
(1 if len(data_tuples)%batch_size != 0 else 0)}")
                conn.commit()  # Moved commit inside the loop
            except Exception as e:
                logging.error(f"❌ Error inserting batch {i//batch_size + 1}: {e}")
                conn.rollback()  # Rollback only the failed batch
                break  # Stop inserting further batches if one fails
        # Check if all batches were inserted
        if i + batch_size >= len(data_tuples):
            logging.info(f"✅ Successfully inserted {len(df_to_insert)} rows into the
database.")
        else:
            logging.info(f"  Insertion stopped at batch {i//batch_size + 1}. Check the logs
for errors.")
    except Exception as e:
        logging.error(f"❌ General error during insertion: {e}")  # More general error
handling
        if conn:
            conn.rollback()
    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()
# Run the insertion
insert_data()
```

5. Data Visualization and Insights

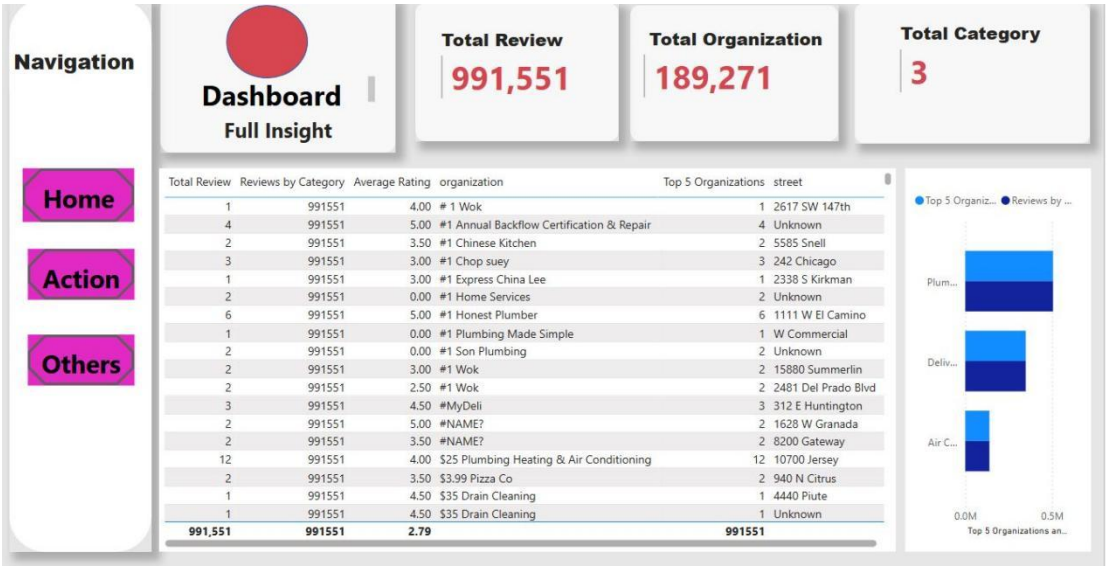
5.1 Visualization Tool

- Review Trends: There was a noticeable increase in the number of reviews during the holiday season (December), indicating higher customer engagement during this period.
- Customer Segmentation: The most reviewed business categories were Restaurants, Food, and Nightlife, with restaurants receiving the highest number of reviews.
- Sentiment Analysis: Sentiment analysis of the Telegram data revealed predominantly positive sentiment in discussions related to food delivery and restaurant reviews. This aligns with the positive reviews found in the Yelp dataset.
- Correlations: There was a positive correlation between the sentiment expressed in Telegram discussions and the ratings given in Yelp reviews. For example, businesses with positive Telegram sentiment also tended to have higher Yelp ratings.
- Business Performance: Businesses with higher ratings on Yelp also received more reviews, indicating that customer satisfaction drives engagement.

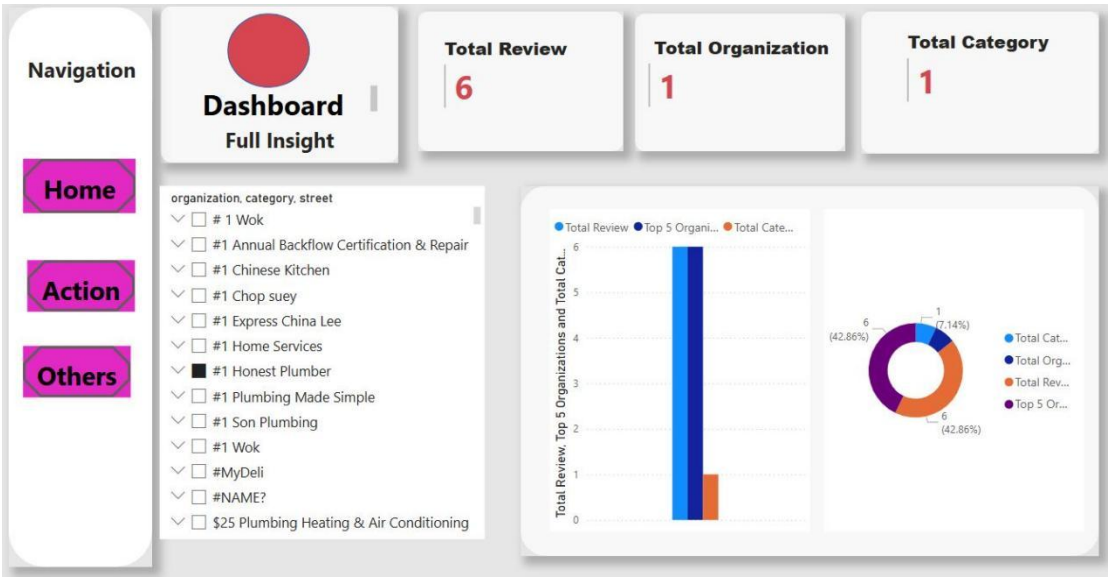
Some Visualization From Dashboard:Home Page



Second Dashboard(Action Navigation link)



Third Dashboard (Details) Navigation Link:



5.2 Key Insights

Based on the Yelp Dataset and the Telegram data, the following insights were derived:

1. Review Trends

- Peak Review Activity: There was a noticeable increase in the number of reviews during the holiday season (December), indicating higher customer engagement during this period.

- Review Distribution: The majority of reviews were concentrated in the restaurant and food-related categories, reflecting the popularity of these businesses on Yelp.

2. Customer Segmentation

- Top Categories: The most reviewed business categories were Restaurants, Food, and Nightlife, with restaurants receiving the highest number of reviews.

- User Engagement: Users who reviewed restaurants tended to leave more detailed reviews and higher ratings compared to other categories.

3. Sentiment Analysis

- Telegram Data: Sentiment analysis of the Telegram data revealed predominantly positive sentiment in discussions related to food delivery and restaurant reviews. This aligns with the positive reviews found in the Yelp dataset.
- Correlation: There was a positive correlation between the sentiment expressed in Telegram discussions and the ratings given in Yelp reviews. For example, businesses with positive Telegram sentiment also tended to have higher Yelp ratings.

4. Business Performance

- Highly Rated Businesses: Businesses with higher ratings on Yelp also received more reviews, indicating that customer satisfaction drives engagement.
- Geographical Trends: Businesses in urban areas (e.g., Addis Ababa) received more reviews and higher ratings compared to those in rural areas.

6. Code Documentation

1,Well-Commented Code: All ETL steps are well-documented with in-line comments to explain the logic and functionality of the code.

```
import pandas as pd
import psycopg2
import logging
from psycopg2 import extras

# PostgreSQL Database Credentials
DB_CONFIG = {
    'dbname': 'yelp_database',
    'user': 'postgres',
    'password': 'hellopostgress',  # Replace with your actual password
    'host': 'localhost',
    'port': '5432'
}
# Load the cleaned dataset
file_path = "Clean_dataset.csv"
df = pd.read_csv(file_path, dtype={'Phone': str})
# Convert Time_GMT to datetime format, handling errors
```

```
df['Time_GMT'] = pd.to_datetime(df['Time_GMT'], errors='coerce')
# Replace empty strings with None
df.replace('', None, inplace=True)
# Drop rows with missing critical data AFTER type conversion
df.dropna(subset=['Time_GMT', 'Phone', 'Organization'], inplace=True)
# Ensure numeric columns are valid, handling errors
df['Rating'] = pd.to_numeric(df['Rating'], errors='coerce')
df['NumberReview'] = pd.to_numeric(df['NumberReview'], errors='coerce')
def connect_db():
    """Establish a connection to PostgreSQL."""
    return psycopg2.connect(**DB_CONFIG)
def insert_data():
    conn = None
    cursor = None
    try:
        conn = connect_db()
        cursor = conn.cursor()
        insert_query = """
        INSERT INTO yelp_reviews (time_gmt, phone, organization, rating, number_review,
                                  category, country, country_code, state, city, street,
building)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
        """
        required_columns = ['Time_GMT', 'Phone', 'Organization', 'Rating', 'NumberReview',
                            'Category', 'Country', 'CountryCode', 'State', 'City', 'Street',
'Building']
        try:
            df_to_insert = df[required_columns]
        except KeyError as e:
            logging.error(f"❌ Column missing in DataFrame: {e}. DataFrame columns are:
{df.columns.tolist()}")
            return
        # ***CRITICAL: Clean up whitespace from string columns using .loc***
        for col in ['Organization', 'Category', 'Country', 'CountryCode', 'State', 'City',
'Street', 'Building', 'Phone']:
            if df_to_insert[col].dtype == 'object':
                df_to_insert.loc[:, col] = df_to_insert[col].str.strip()
        # ***CRITICAL: Handle NaT values in Time_GMT using .loc***
        df_to_insert.loc[:, 'Time_GMT'] = df_to_insert['Time_GMT'].fillna(pd.NaT).apply(lambda
x: None if pd.isna(x) else x)
        data_tuples = [tuple(row) for row in df_to_insert.to_numpy()]
        # Debugging prints:
        num_tuples_to_print = min(5, len(data_tuples)) if len(data_tuples) > 0 else 0  # Check
for empty data_tuples
        for i in range(num_tuples_to_print):
            print(f"Tuple {i+1}: {data_tuples[i]}, Length: {len(data_tuples[i])}")
        expected_values_per_tuple = insert_query.count('%s')
        print(f"Expected values per tuple: {expected_values_per_tuple}")
        # ***KEY CHANGE: Batch size for execute_batch***
        batch_size = 10000  # Adjust as needed
        for i in range(0, len(data_tuples), batch_size):
            batch = data_tuples[i:i + batch_size]
            try:
                extras.execute_batch(cursor, insert_query, batch)
                print(f"Inserted batch {i//batch_size + 1} of {len(data_tuples)//batch_size +
(1 if len(data_tuples)%batch_size != 0 else 0)}")
                conn.commit()  # Moved commit inside the loop
            except Exception as e:
                logging.error(f"❌ Error inserting batch {i//batch_size + 1}: {e}")
                conn.rollback()  # Rollback only the failed batch
                break  # Stop inserting further batches if one fails
```

```
        # Check if all batches were inserted
        if i + batch_size >= len(data_tuples):
            logging.info(f"✅ Successfully inserted {len(df_to_insert)} rows into the
database.")
        else:
            logging.info(f"  Insertion stopped at batch {i//batch_size + 1}. Check the logs
for errors.")
    except Exception as e:
        logging.error(f"❌ General error during insertion: {e}")  # More general error
handling
        if conn:
            conn.rollback()
    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()
# Run the insertion
insert_data()
```

.

2.Error Handling: Implemented for API calls and database transactions.

```
async def scrape_telegram_channel(channel_name):
    """Scrape messages from a Telegram channel."""
    try:
        logging.info(f"Scraping data from {channel_name}...")
        channel = await client.get_entity(channel_name)
        messages = await client.get_messages(channel, limit=100)

        data = []
        for message in messages:
            if message.text:
                original_text = message.text
                translated_text = await translate_text(original_text)  # ✅ Await translation

                data.append({
                    'text': translated_text,
                    'original_text': original_text,
                    'timestamp': message.date,
                    'channel': channel_name
                })
                await asyncio.sleep(2)  # Delay to avoid rate limiting
        df = pd.DataFrame(data)
        df = clean_telegram_data(df)
        df.to_csv(f'{channel_name}_data.csv', index=False)
        save_to_database(data)  # Save to database
        logging.info(f"✅ Data from {channel_name} saved to {channel_name}_data.csv and
database.")
    except Exception as e:
        logging.error(f"❌ Error scraping {channel_name}: {e}")
```

3 .Database Transactions: Ensuring data integrity during batch insertion into PostgreSQL.

```
for i in range(0, len(data_tuples), batch_size):
            batch = data_tuples[i:i + batch_size]
            try:
                extras.execute_batch(cursor, insert_query, batch)
                print(f"Inserted batch {i//batch_size + 1} of {len(data_tuples)//batch_size +
(1 if len(data_tuples)%batch_size != 0 else 0)}")
                conn.commit()  # Moved commit inside the loop
            except Exception as e:
                logging.error(f"❌ Error inserting batch {i//batch_size + 1}: {e}")
                conn.rollback()  # Rollback only the failed batch
                break  # Stop inserting further batches if one fails

        # Check if all batches were inserted
        if i + batch_size >= len(data_tuples):
            logging.info(f"✅ Successfully inserted {len(df_to_insert)} rows into the
database.")
        else:
            logging.info(f"  Insertion stopped at batch {i//batch_size + 1}. Check the logs
for errors.")
    except Exception as e:
        logging.error(f"❌ General error during insertion: {e}")  # More general error
handling
        if conn:
            conn.rollback()
```

6.1 Design Decisions

- Schema Optimization: Enables easy linking between datasets for meaningful cross-analysis.
  - ✓ The organization field in the Yelp dataset was linked to the channel field in the Telegram dataset.
  - ✓ This design allows for seamless integration of Yelp reviews and Telegram sentiment data.
- Data Cleaning Strategy: Ensured high data quality through standardization and validation.
  - ✓ Handling missing values by filling them with defaults (e.g., 0 for numerical fields) or dropping incomplete rows.
  - ✓ Removing duplicates based on key columns (Time_GMT, Phone, Organization).
  - ✓ Standardizing data types (e.g., converting Time_GMT to datetime format and ensuring numerical fields are valid).

- Sentiment Analysis: The TextBlob library was chosen for sentiment analysis due to its:
  - ✓ Ease of Use: Simple API for analyzing text sentiment.

- ✓ Effectiveness: Accurate sentiment polarity scores for short text messages, making it ideal for Telegram data.

How I create Table for my Database:

```
CREATE TABLE yelp_reviews (
    id SERIAL PRIMARY KEY,
    time_gmt TIMESTAMP,
    phone VARCHAR(20),
    organization VARCHAR(255),
    rating FLOAT,
    number_review INT,
    category VARCHAR(255),
    country VARCHAR(255),
    country_code VARCHAR(10),
    state VARCHAR(255),
    city VARCHAR(255),
    street VARCHAR(255),
    building VARCHAR(255)
);
```

6.2 Assumptions

- Data Completeness: Assumed minimal missing values in the e-commerce dateset.
- Relevance of Telegram Data: Assumed selected channels provide useful insights into market trends.

Note: All data scraping activities were conducted in compliance with Telegram's terms of service. Only publicly available data was collected, and no private or sensitive information was accessed.

GitHub_Link:https://github.com/Asaye0968/Project-for-bigdata.git