# Blue Illustration Revolutionary Technology Poster

Catalog

**[Introduction 3](#_Toc26004)**

**[1. Data Source Identification & Understanding 3](#_Toc1487)**

[1.1 Large Dataset (E-commerce Data) 3](#_Toc27931)

[ Dataset Source: The dataset was sourced from Kaggle, containing over 1 million rows of e-commerce transaction data. 3](#_Toc15604)

[ Dataset Structure: The dataset is in CSV format, with fields such as order\_id, customer\_id, product\_id, order\_date, sales\_amount, and product\_category. 3](#_Toc25377)

[ Understanding: The dataset provides insights into customer purchasing behavior, sales trends, and product performance. 3](#_Toc31389)

[1.2 Telegram Channels (E-commerce Related) 3](#_Toc29835)

**[2. Data Extraction 3](#_Toc27214)**

[2.1 Large Dataset Extraction 3](#_Toc12370)

[2.2 Telegram Data Extraction 4](#_Toc12944)

**[3. Data Transformation 4](#_Toc4201)**

[3.1 Data Cleaning 4](#_Toc25811)

[3.2 Sentiment Analysis on Telegram Data 5](#_Toc2723)

**[4. Data Loading 5](#_Toc10556)**

[4.1 Database Schema Design 6](#_Toc29122)

[4.2 Loading Data into PostgreSQL 6](#_Toc28143)

**[5. Data Visualization and Insights 8](#_Toc28704)**

[5.1 Visualization Tool 8](#_Toc23954)

[Some Visualization from Telegram data: 10](#_Toc28823)

[5.2 Key Insights 12](#_Toc16168)

**[6. Code Documentation 12](#_Toc12166)**

**[6.1 Design Decisions 12](#_Toc12653)**

**[6.2 Assumptions 13](#_Toc15656)**

# 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

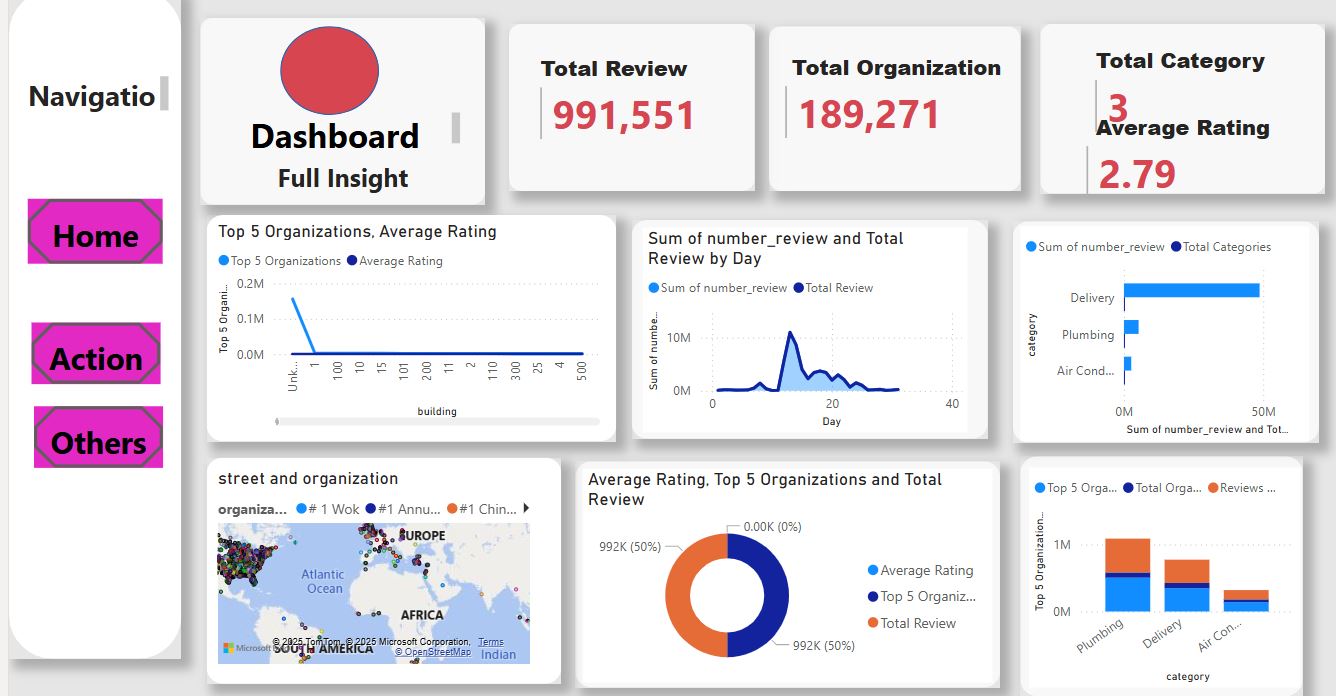
insert\_data()

# 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**

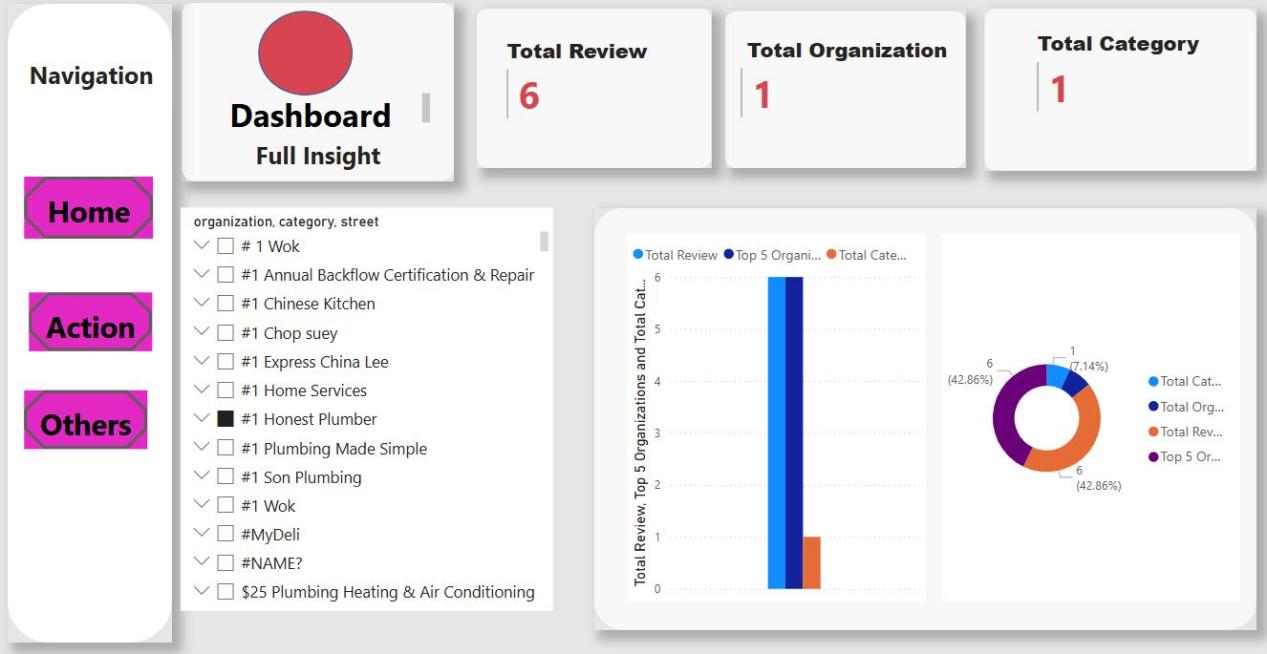
****

## **Second Dashboard(Action Navigation link)**

## **dashboard2**

## 

**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 dataset.
* 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.