Project: Customer Review Sentimental Analysis

Members:

- Abrar Sobhi Basyouni Khodeir
- Ahmed Fathy
- Heba Khali
- Mahmoud Mohamed Ibrahim Ghazi
- Moataz Majid Ahmed
- Nada Hany
- Riham Salama

Project overview:

This project focuses on using **Azure Synapse Analytics** to process consumer review data from Amazon, performing sentiment analysis, and structuring the results in a data warehouse.

1. Data Acquisition

- Sourced data from Kaggle's Amazon Consumer Reviews Dataset.
- Uploaded raw data to **Azure Data Lake Gen2** for storage and access.

2. Data Pipeline: Cleaning and Transformation

- Built a pipeline in **Azure Synapse** and **Azure Functions** for data cleaning and transformation. Key cleaning steps include:
    - Lowercasing text, removing whitespaces, HTML tags, digits, punctuation, and stop words.
    - Collapsing multiple spaces and normalizing text.
    - Stemming words to their base form.

3. Sentiment Analysis: Machine Learning Model

- Applied **TF-IDF** for feature extraction.
- Split the dataset into **training** and **testing** subsets.
- Trained a sentiment analysis model using machine learning algorithms.
- Deployed and tested the model to predict sentiment (positive or negative) for all reviews.
- Stored the trained model in **Azure Data Lake** for reuse.

4. Data Modeling: Fact and Dimension Tables

- Applied a data modeling approach to structure the processed review data into **Fact and Dimension Tables**.
    - **Fact Tables** store detailed transactional data (e.g., sentiment scores).
    - **Dimension Tables** contain supporting data such as product details and review metadata.

5. Loading and Querying Data

- Loaded the processed and modeled data into a **serverless SQL warehouse** in **Azure Synapse**.
- Converted the Parquet files stored in **Azure Data Lake** into **external tables** in Synapse SQL, enabling efficient querying and reporting for business insights.

6. Key Project Deliverables

- **Automated data pipeline** for transforming and analyzing customer reviews at scale.
- **Sentiment analysis model** applied to the full dataset, offering sentiment predictions.
- **Optimized data structure** through Fact and Dimension tables for fast, flexible querying.

Dataset overview

The date set is a list of over 3,000 consumer reviews for Amazon products like the Kindle, Fire TV Stick, and more provided by Datafiniti's Product Database. The dataset includes basic product information, rating, review text, and more for each product.

Schema Design Overview

- **Goal**: The database is designed to support sentiment analysis on product review data. It enables efficient querying for both review metrics and product details, supporting various analyses such as sentiment trends, product feedback, and reviewer behavior.

Tables:

1. **Fact_Reviews**: Stores detailed review data, including ratings, timestamps, and user actions.
2. **Dim_Product**: Contains product-specific information such as name, brand, and category.
3. **Dim_User**: Captures user details (e.g., username, location).
4. **Dim_Time**: Stores the breakdown of review dates for time-series analysis.

Design Rationale:

- **Star Schema**: The star schema is selected to optimize read-heavy analytical queries.
- **Fact Table**: Captures measures (ratings, review helpfulness) and links to dimensions.
- **Dimension Tables**: Provide context (product, user, time) for analysis, avoiding redundancy.

Database Schema:

1. **Fact Table: Fact_Reviews**

Column	Type	Description
user_id	NVARCHAR(50)	Unique identifier for the fact user.
product_id	NVARCHAR(50)	Unique identifier for the fact product.
date_id	NVARCHAR(50)	Unique identifier for the fact date entity.
reviews.doRecommend	BIT	Indicates if the user recommends the product (Yes/No).
reviews.numHelpful	FLOAT	Number of users who found the review helpful.
reviews.rating	FLOAT	Numeric rating given by the reviewer (e.g., 1-5).
reviews.sourceURLs	NVARCHAR(MAX)	URL where the review was originally posted.

2. Dimension Table: Dim_Product

Column	Type	Description
product_id	NVARCHAR(50)	Unique identifier for each product.
`name`	NVARCHAR(255)	Name of the product.
`asins`	NVARCHAR(50)	Amazon Standard Identification Number (ASIN) for the product.
`brand`	NVARCHAR(255)	Brand name of the product.
`categories`	NVARCHAR(MAX)	Product categories, often in hierarchical format.
`keys`	NVARCHAR(MAX)	Keywords or tags associated with the product.
`manufacturer`	NVARCHAR(255)	Manufacturer of the product.

3. Dimension Table: Dim_User

Column	Type	Description
user_id	NVARCHAR(50)	Unique identifier for the user.
`reviews.username`	NVARCHAR(255)	Username of the person who submitted the review.
`reviews.userCity`	NVARCHAR(255)	City of the reviewer (if available).
`reviews.userProvince`	NVARCHAR(255)	Province of the reviewer (if available).

3. Dimension Table: Dim_Time

Column	Type	Description
date_id	NVARCHAR(50)	Unique identifier for the time record.
`date`	DATETIME2	Date of the event (review).
`year`	INT	Year extracted from the `date` field.
`month`	INT	Month extracted from the `date` field.
`day`	INT	Day extracted from the `date` field.

Date preprocessing

Python module used for text preprocessing in natural language processing (NLP). our module includes functions to clean text, remove stopwords and stem words for improved machine learning and natural language understanding.

Overview

The text processing module aims to clean and prepare textual data for modeling. The steps include:

1. **Text Cleaning**: Converting to lowercase, removing extra spaces, HTML tags, digits, and punctuation.
2. **Stopword Removal**: Removing frequently occurring words that do not contribute much meaning.
3. **Stemming**: Reducing words to their base/root form by removing suffixes.

Functions

1. `clean_text(text: str) -> str`

Cleans the input text by performing the following operations:

- Converts the text to lowercase.
- Removes leading and trailing whitespaces.
- Removes HTML tags (e.g., `<br>`, `<div>`).
- Replaces digits with spaces.
- Replaces punctuation with spaces.
- Collapses multiple spaces and tabs into a single space.

Parameters:

- `text`: A string containing the text to be cleaned.

Returns:

- A string of cleaned text.

2. `remove_stopwords(text: str) -> str`

It removes common stopwords from the text. Stopwords are words like "the," "is," and "in," which often do not add much meaning in text analysis.

Parameters:

- `text`: A string of cleaned text.

Returns:

- A string with stopwords removed.

Note: The function uses the NLTK stopwords list, which can be modified as needed.

---

3. `stemm_text(text: str) -> str`

Applies stemming to the input text. Stemming reduces words to their base form (e.g., "jumping" -> "jump"). It’s a rule-based method that removes suffixes.

Parameters:

- `text`: A string of cleaned text.

Returns:

- A string with each word reduced to its root form using stemming.

---

4. `get_wordnet_pos(tag: str) -> str`

A helper function that maps Part-of-Speech (POS) tags to the format expected by the WordNet lemmatizer. Based on its POS tag, this function identifies if a word is a noun, verb, adjective, or adverb.

Parameters:

- `tag`: A string representing the POS tag of a word.

Returns:

- If applicable, a WordNet POS tag (NOUN, VERB, ADJ, or ADV) defaults to NOUN otherwise.

Sentimental analysis

TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF is applied to convert textual data into numerical features that reflect the importance of words in a document. The process involves creating a vectorizer with limited features and transforming the text into a numerical matrix using this vectorization.

Steps:

- Use **TfidfVectorizer** to select the top 700 words based on their scores:

  ```
  vectorizer = TfidfVectorizer(max_features=700)
  vectorizer.fit(data['text'])
  features = vectorizer.transform(data['text'])
  ```
- Convert this feature matrix into a DataFrame:

  ```
  tf_idf = pd.DataFrame(features.toarray(),
  columns=vectorizer.get_feature_names())
  ```

3. Splitting the Dataset for Training and Testing

The dataset is split into training and testing subsets for model evaluation using **train_test_split**:

```
X_train, X_test, y_train, y_test = train_test_split(tf_idf,
data['sentiment_score'], test_size=0.2, random_state=42)
```

4. Storing and Accessing the Model in Azure Data Lake Gen2

After training, the model is serialized and stored in **Azure Data Lake Gen2** as a **Pickle** file for later use:

```
import joblib
joblib.dump(model, '/path/to/data-lake/model.pkl')
```

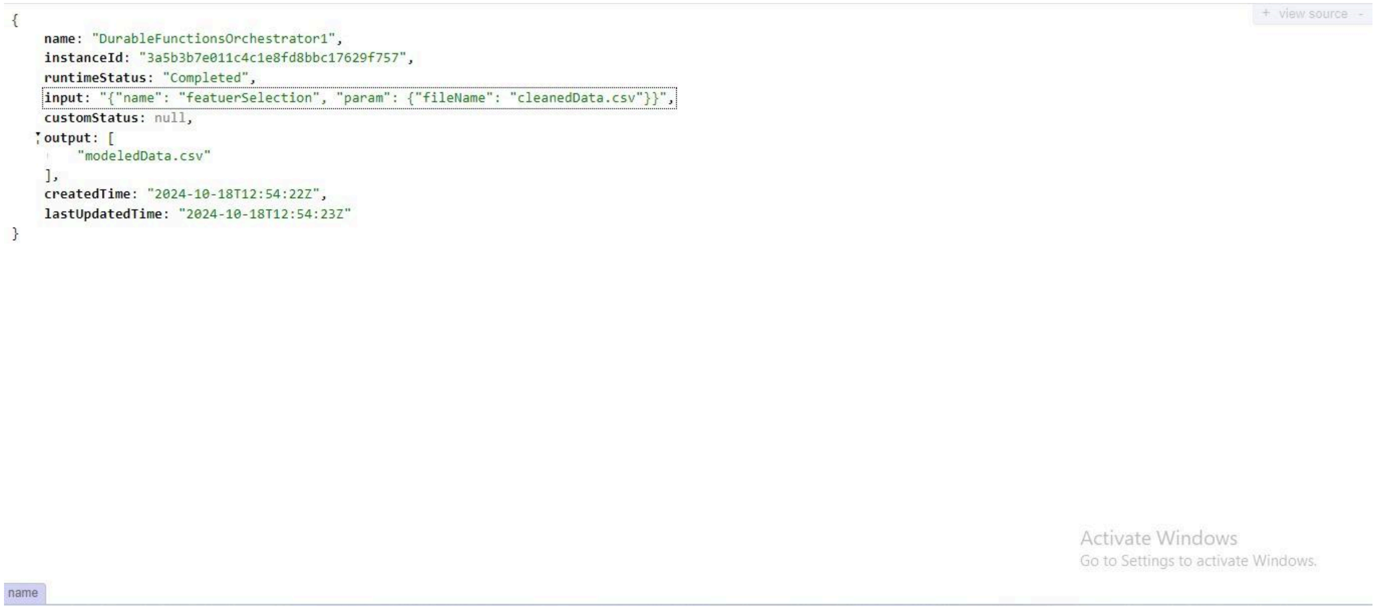
The model can then be accessed in an **Azure Function** for real-time prediction:

```
model = joblib.load('/path/to/data-lake/model.pkl')
```

5. Azure Function for Sentiment Analysis

An **Azure Function** loads the saved model, applies it to new review data, and outputs sentiment predictions:

```
def predict_sentiment(review_text):
    review_vect = vectorizer.transform([review_text])
    return model.predict(review_vect)
```



```
{
    name: "DurableFunctionsOrchestrator1",
    instanceId: "3a5b3b7e011c4c1e8fd8bbc17629f757",
    runtimeStatus: "Completed",
    input: "{"name": "featuerSelection", "param": {"fileName": "cleanedData.csv"}}",
    customStatus: null,
    output: [
        "modeledData.csv"
    ],
    createdTime: "2024-10-18T12:54:22Z",
    lastUpdatedTime: "2024-10-18T12:54:23Z"
}
```

6. Fact and Dimension Table Creation using Parquet

After prediction, the data is processed to generate **Fact and Dimension tables** in **Parquet** format, optimizing them for analytical querying. The fact table includes detailed information about reviews, while the dimensions contain information like product details, review metadata, and users. This is achieved by creating separate datasets for facts and dimensions, and storing them in **Azure Data Lake** or **Synapse SQL Serverless**.

Example:

```
reviews_fact.to_parquet('/path/to/data-lake/fact_reviews.parquet')
product_dim.to_parquet('/path/to/data-lake/dim_product.parquet')
```

Integration into Azure Synapse Pipelines

Using the **Azure Synapse Pipeline**, you can automate the extraction, transformation, and loading (ETL) process. This pipeline integrates the sentiment predictions with existing data to create structured tables for reporting and analysis.

Converting Parquet Files to External Tables in Azure Synapse SQL

1. Process Overview

Once the data is processed and stored in **Parquet** format within **Azure Data Lake Gen2**, the next step involves creating **external tables** in **Azure Synapse SQL**. This allows the structured data to be queried without moving it from the Data Lake, leveraging **Synapse SQL Serverless** for efficient data analysis.

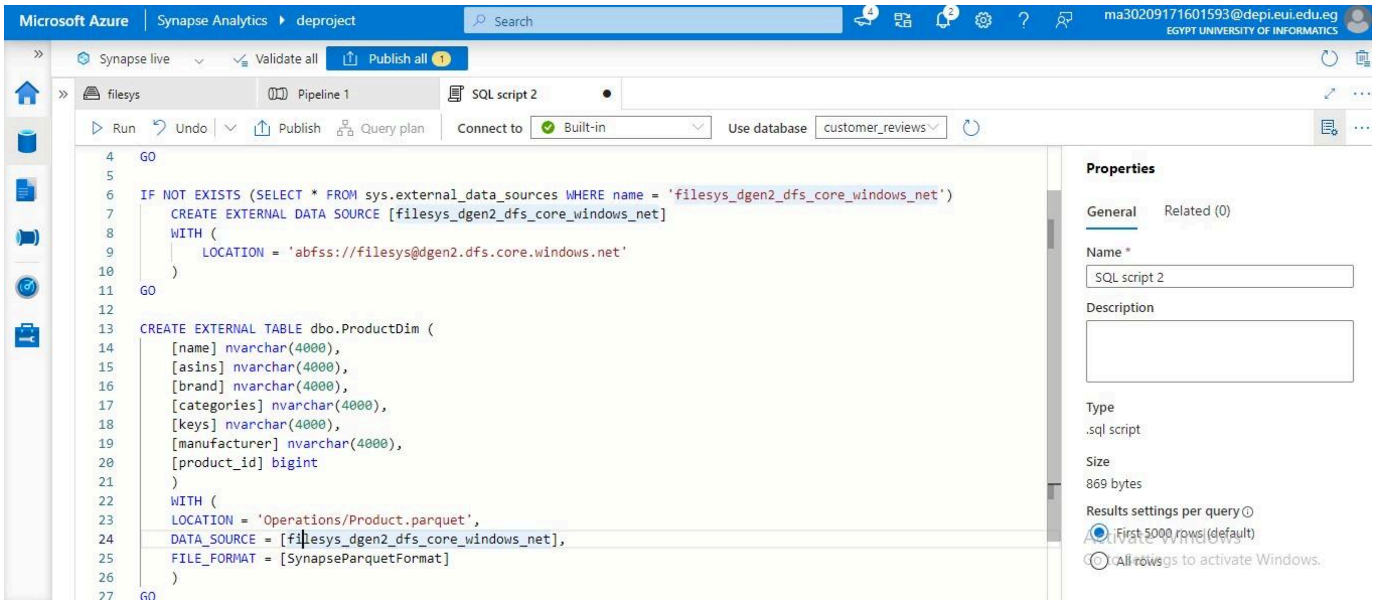
---

2. Prerequisites

- Ensure that your **Azure Synapse workspace** is connected to **Azure Data Lake Storage Gen2** where the Parquet files reside.
- Necessary permissions are granted for accessing Data Lake storage from Synapse.

---

3. Steps for Converting Parquet to External Tables



Connect to: Built-in
Use database: customer_reviews

```sql
GO

IF NOT EXISTS (SELECT * FROM sys.external_data_sources WHERE name = 'filesys_dgen2_dfs_core_windows_net')
    CREATE EXTERNAL DATA SOURCE [filesys_dgen2_dfs_core_windows_net]
    WITH (
        LOCATION = 'abfss://filesys@dgen2.dfs.core.windows.net'
    )
GO

CREATE EXTERNAL TABLE dbo.ProductDim (
    [name] nvarchar(4000),
    [asins] nvarchar(4000),
    [brand] nvarchar(4000),
    [categories] nvarchar(4000),
    [keys] nvarchar(4000),
    [manufacturer] nvarchar(4000),
    [product_id] bigint
    )
    WITH (
    LOCATION = 'Operations/Product.parquet',
    DATA_SOURCE = [filesys_dgen2_dfs_core_windows_net],
    FILE_FORMAT = [SynapseParquetFormat]
    )
GO
```

Properties

General | Related (0)

Name *: SQL script 2

Description:

Type: .sql script

Size: 869 bytes

Results settings per query: [✓ First 5000 rows (default)] All rows

Step 1: Create a Data Source

define an **external data source** that points to your Data Lake.

```sql
CREATE EXTERNAL DATA SOURCE [CustomerReviews]
WITH (
      LOCATION =
'https://<storageaccountname>.dfs.core.windows.net/fact_reviews.parquet'
);
```

Step 2: Create an External File Format
This defines how the data is structured in the Parquet files.

```
CREATE EXTERNAL FILE FORMAT parquet_file_format
WITH (
    FORMAT_TYPE = PARQUET
);
```

Step 3: Create an External Table
This is the final step where you map the external Parquet file to an **external table** in Synapse SQL.

```
CREATE EXTERNAL TABLE dbo.FactReviews
(
    review_id INT,
    product_id INT,
    sentiment_score FLOAT,
    review_text NVARCHAR(4000),
    review_date DATE
)
WITH (
    LOCATION = '/path/to/parquet/fact_reviews.parquet',
    DATA_SOURCE = [YourDataSourceName],
    FILE_FORMAT = parquet_file_format
);
```

5. Benefits of External Tables

- **Cost-efficient**: Data is queried directly from the Data Lake without requiring storage in Synapse, saving costs.
- **Scalability**: External tables scale well, allowing analysis on large datasets.

Extra: For check sentimental analysis results:

```
def transformer_check(n=1000):
  """ Return Dataframe of reviews sentimental prediction and actual sentimental"""
  res_list = []
  for _ in range(n):
    res_dict = {}
    i = random.choice(range(len(data)))
    text_ = data['text'].iloc[i]
    sen_act = data['sentiment'].iloc[i]
    sen_pred = sentiment_analyzer(text_)[0]['label']
    res_dict['review'] = text_
    res_dict['sen_act'] = sen_act
    res_dict['sen_pred'] = sen_pred
    res_dict['match'] = sen_act == sen_pred

    res_list.append(res_dict)

  return pd.DataFrame(res_list)
```