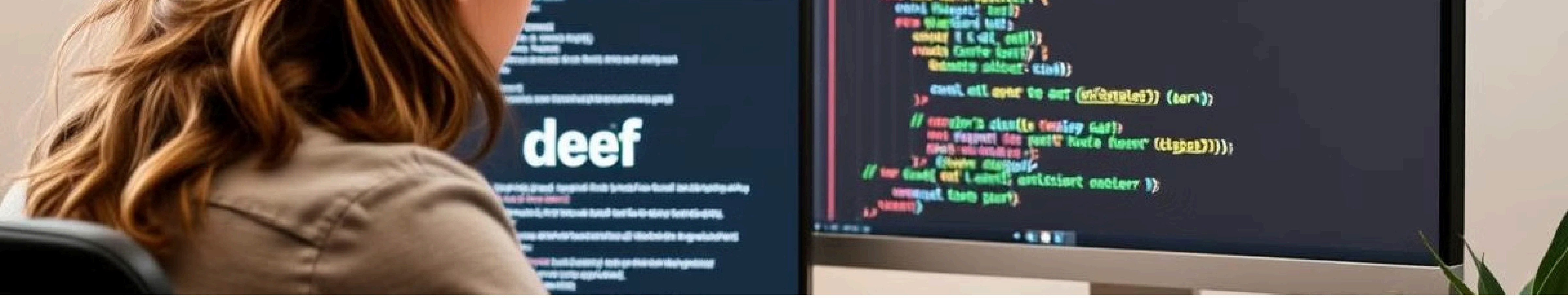
Data Analysis Assignment: A Comprehensive Guide

This assignment outlines a step-by-step approach to a data analysis project. The assignment will use a real-world dataset, and we'll demonstrate the entire data processing pipeline, from data acquisition to insights and visualization.

by Akash saxena



Step 1: Data Acquisition

The first step in this data analysis project is to acquire the necessary data. We'll use Python's 'requests' library to make an API request to fetch the data. This API will be assumed to provide data for 2017, but the Python script can be modified to remove or update the date filter. This allows for greater flexibility in data acquisition.

Here's a basic Python snippet that illustrates the data acquisition using an API:

```
# Step 1: Make an API request for 2017 data; this code can be further updated to remove/modify the date filter
url = "https://public.opendatasoft.com/api/explore/v2.1/catalog/datasets/openaq/records?
where=measurements_lastupdated%20%3E%3D%20date%272017%27%20and%20measurements_lastupdated%
20%3C%20date%272018%27&limit=100"
response = requests.get(url)

# Step 2: Check if the request was successful
if response.status_code == 200:
    data = response.json()
else:
    print(f"Failed to fetch data. Status code: {response.status_code}")

# Step 3: Convert the JSON data to a DataFrame
df = pd.json_normalize(data['results'])

#Display the DataFrame
df.reset_index(drop=True, inplace=True)
df
```

Step 2: Exploratory Data Analysis (EDA)

Once we have the data, we'll perform exploratory data analysis (EDA) to gain insights into the data's characteristics and patterns. This step involves understanding the data's structure, identifying missing values, and detecting outliers. We'll use the 'ydata-profiling' Python library to automatically generate a comprehensive data profile report.

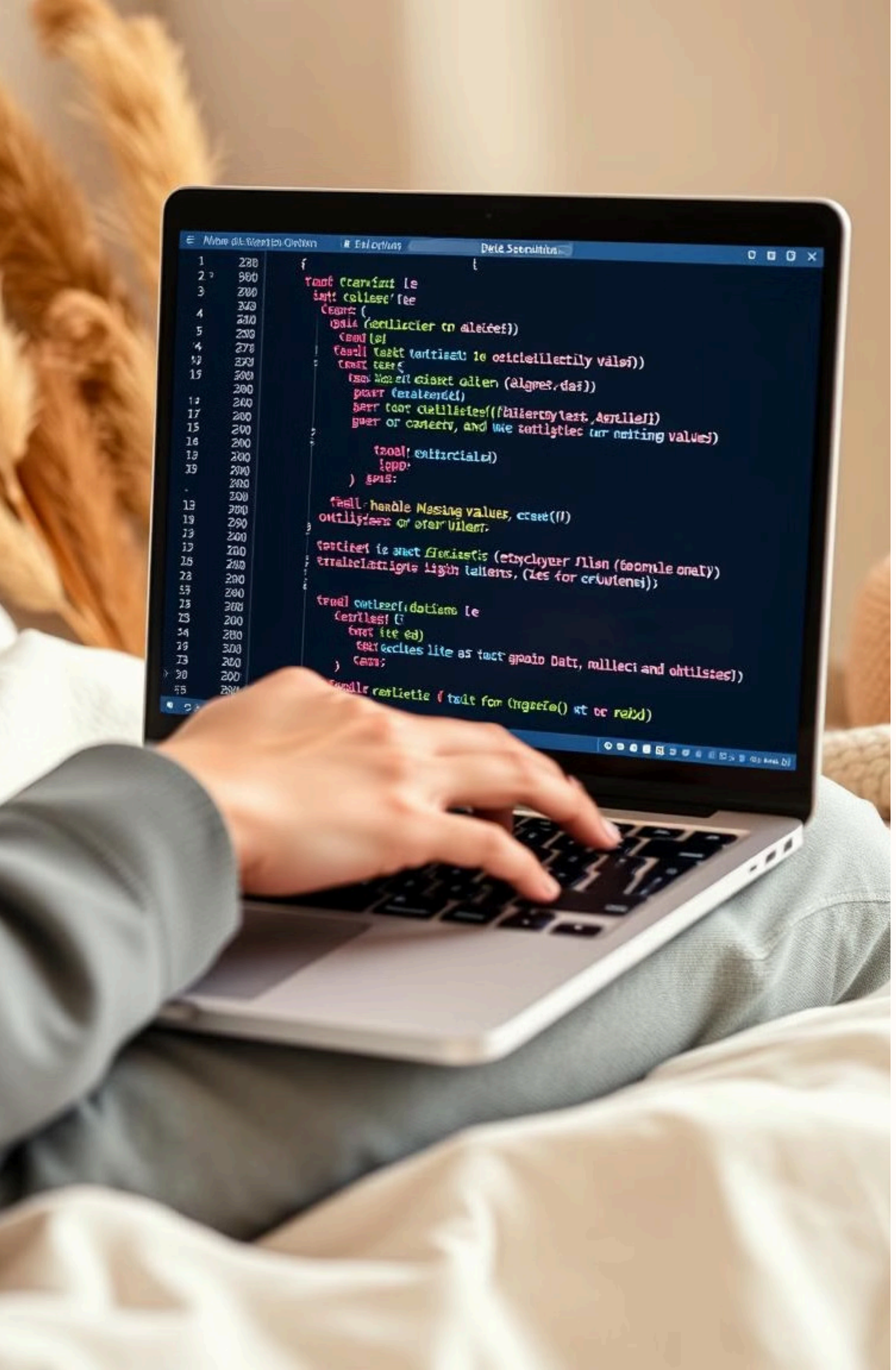
Here's an example of how to generate a data profile report with the 'ydata-profiling' library:

```
# Generate the profiling report
from ydata_profiling import ProfileReport
profile = ProfileReport(df, title="YData Profiling Report",
explorative=True)

# Save the report as an HTML file
profile.to_file("data_profile_report.html")

# Or display the report in a Jupyter Notebook (if applicable)
profile.to_notebook_iframe()
```

This will create an interactive HTML report with summaries of data types, missing values, descriptive statistics, histograms, correlations, and more.



Step 3: Data Cleaning & Preparation

We'll address missing data and outliers in this step to ensure the data's quality. Handling missing data is crucial for building reliable models. For this assignment, we'll use interpolation to fill in missing data. Interpolation assumes a relationship between data points and estimates values based on surrounding data. We can use various interpolation techniques, such as linear interpolation or spline interpolation, depending on the data's characteristics.

Outliers are extreme values that deviate significantly from the rest of the data. Outlier detection is essential as outliers can distort statistical calculations and lead to inaccurate results. We'll use the Interquartile Range (IQR) method to identify outliers and filter them. This method involves calculating the IQR, which is the difference between the 75th and 25th percentiles of the data. Values outside a specified range, typically 1.5 times the IQR below the first quartile or above the third quartile, are considered outliers.

Step 4: Data Separation

Before moving on to data storage and analysis, it's crucial to organize the data for efficient processing and analysis. This step involves separating the data columns of the dataframe based on their data types. We'll classify the columns into three categories: integer, categorical, and ordinal. Integer columns represent numerical data with whole numbers, while categorical columns represent non-numerical data that can take on a limited number of values. Ordinal columns, on the other hand, represent non-numerical data with an inherent order or ranking. This separation allows us to apply appropriate data handling techniques and models to each category.

1. **Integer Columns**

   These columns typically represent quantitative data, such as age, income, or sales figures.

2. **Categorical Columns**

   These columns contain non-numerical data with distinct categories, such as gender, country, or product type.

3. **Ordinal Columns**

   These columns contain non-numerical data with a predefined order, such as customer satisfaction ratings (low, medium, high), or educational levels (high school, bachelor's, master's).

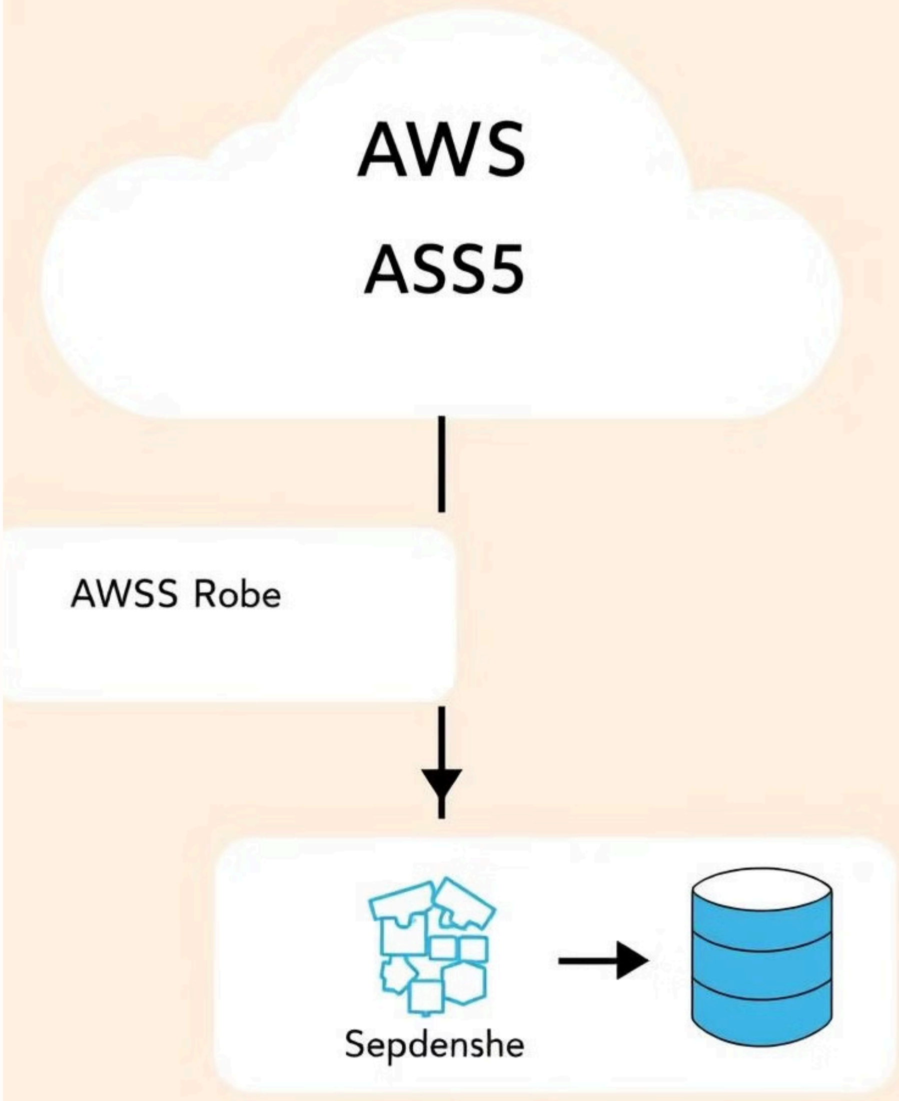
Step 5: Snowflake Database Integration

In this step, we'll connect to a Snowflake public database to store and analyze the data. We'll use a secure connection and appropriate credentials to access the database. We'll create an external stage in Snowflake to interact with an AWS S3 bucket. This external stage allows us to seamlessly move data between Snowflake and the S3 bucket for efficient data management.

Here's an example of creating an external stage in Snowflake:

```
CREATE STAGE my_stage
  URL = 's3://my-s3-bucket'
  CREDENTIALS = (AWS_KEY_ID='YOUR_AWS_ACCESS_KEY_ID',
AWS_SECRET_KEY='YOUR_AWS_SECRET_ACCESS_KEY');
```

This stage will be used to load data into Snowflake from the specified S3 bucket.



Made with Gamma

Step 6: Data Storage and Upload

After cleaning and preparing the data, we'll save the final dataframe to a CSV file using the 'csv' library in Python. This CSV file will be used to load data into Snowflake. We'll use the 'boto3' library to upload the CSV file to the S3 bucket, which we'll connect to our external stage in Snowflake. This approach ensures a consistent and efficient data pipeline for uploading data to Snowflake.

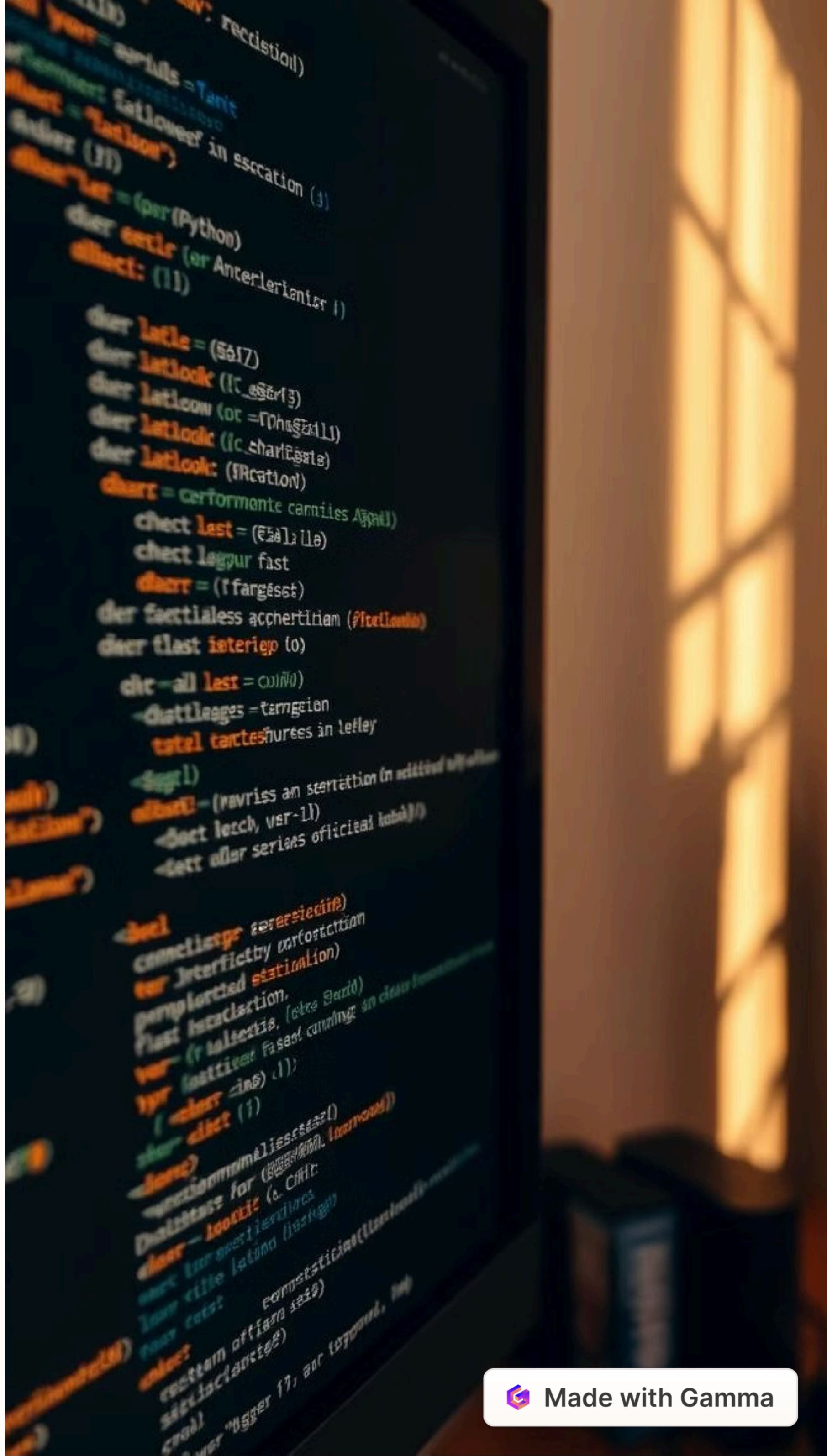
Here's an example of saving a dataframe to a CSV file and uploading it to S3 using 'boto3':

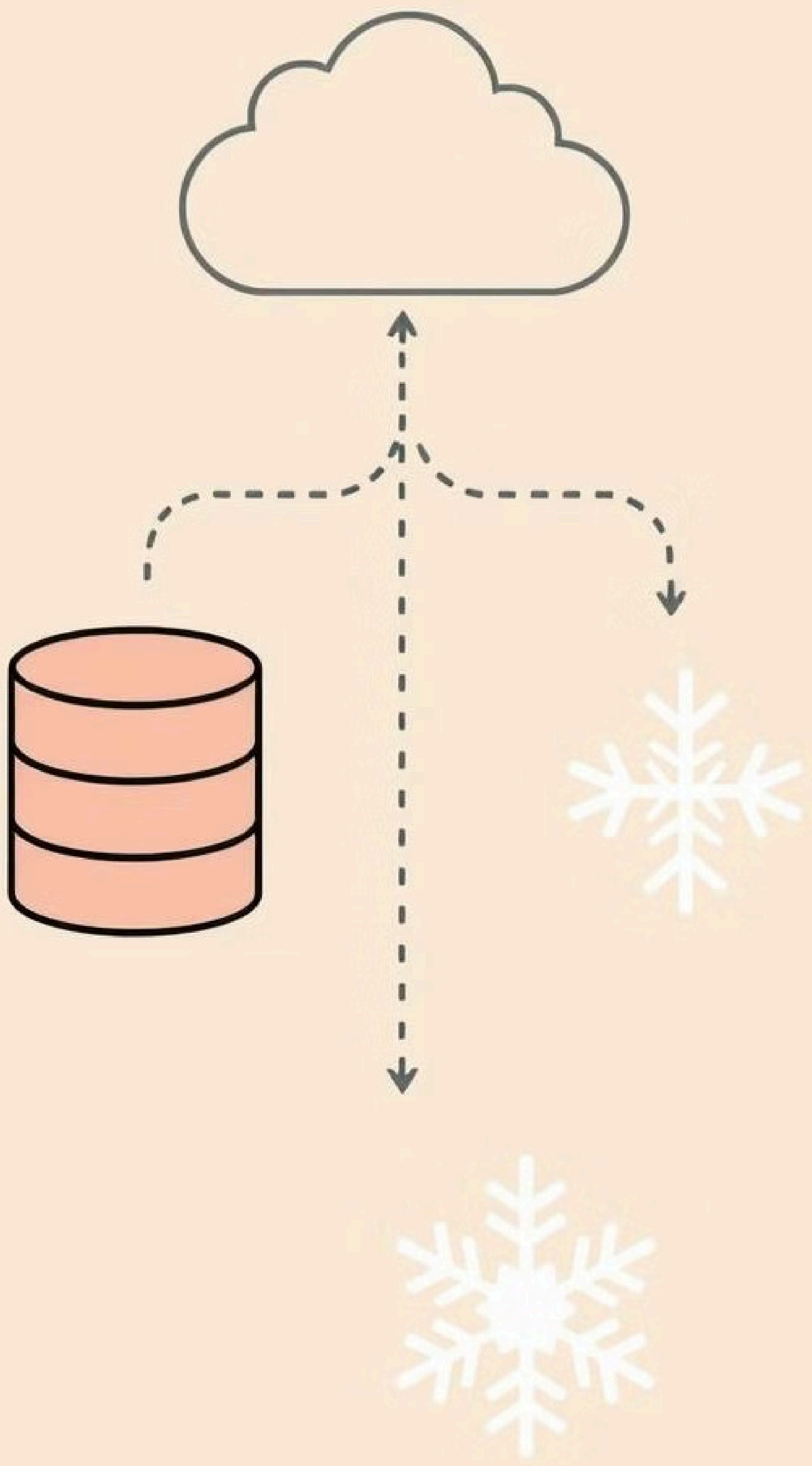
```
import boto3
import csv

# Save dataframe to CSV file
dataframe.to_csv('data.csv', index=False)

# Create S3 client
s3 = boto3.client('s3')

# Upload CSV file to S3 bucket
s3.upload_file('data.csv', 'my-s3-bucket', 'data.csv')
```





Step 7: Snowflake Data Loading and Transformation

We'll create a staging table in Snowflake to hold the data loaded from the CSV file. This staging table ensures that the data is in a consistent format before it is loaded into the final data tables. The staging table will contain only the valid data from the CSV file. We'll then create a Snowflake pipe to automatically copy data from the S3 bucket into the staging table. The pipe will use the external stage we created earlier and will be configured with 'auto_ingest=true' for automatic data ingestion.

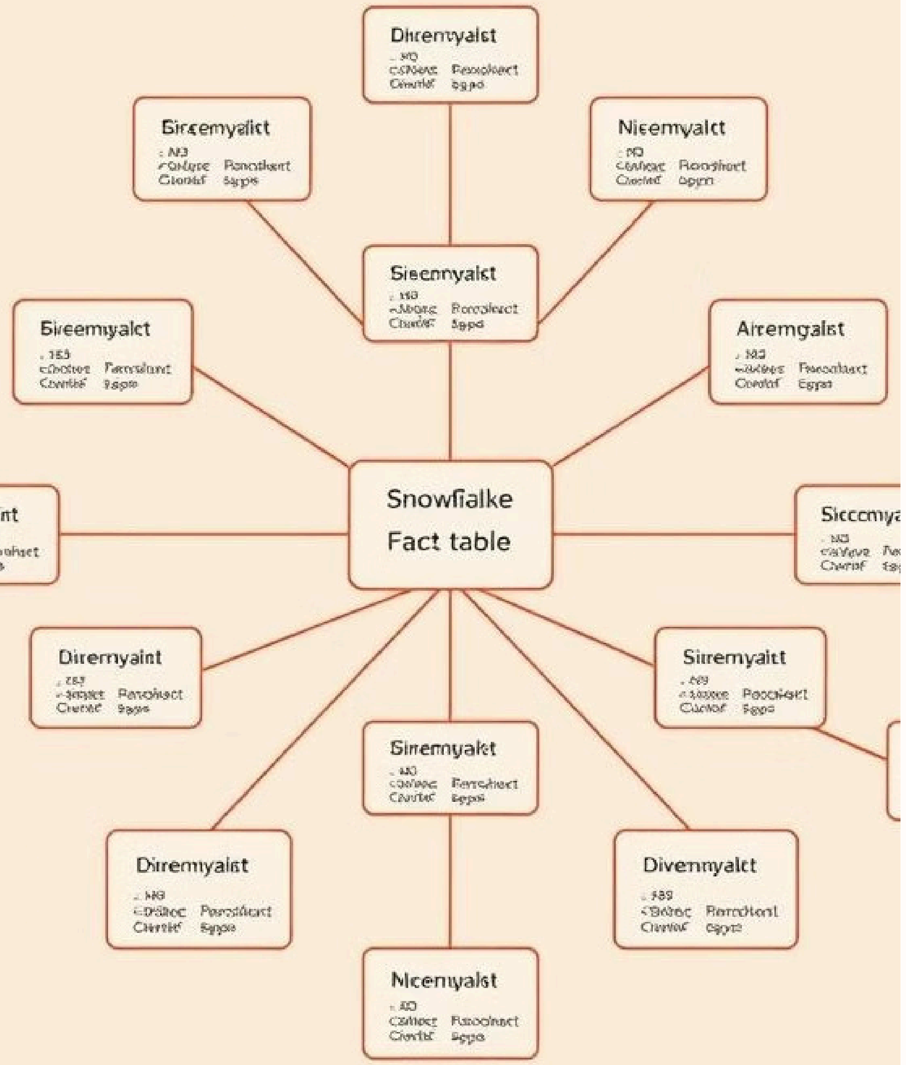
Here's an example of creating a staging table and a pipe in Snowflake:

```
CREATE TABLE staging_table (
  -- Define columns
);

CREATE PIPE my_pipe
  AS
  COPY INTO staging_table
  FROM @my_stage/data.csv
  FILE_FORMAT = (TYPE = CSV);

ALTER PIPE my_pipe SET AUTO_INGEST = TRUE;
```

This will create a pipe named 'my_pipe' that will automatically load data from the 'data.csv' file in the external stage 'my_stage' into the 'staging_table' whenever the file is updated in the S3 bucket.



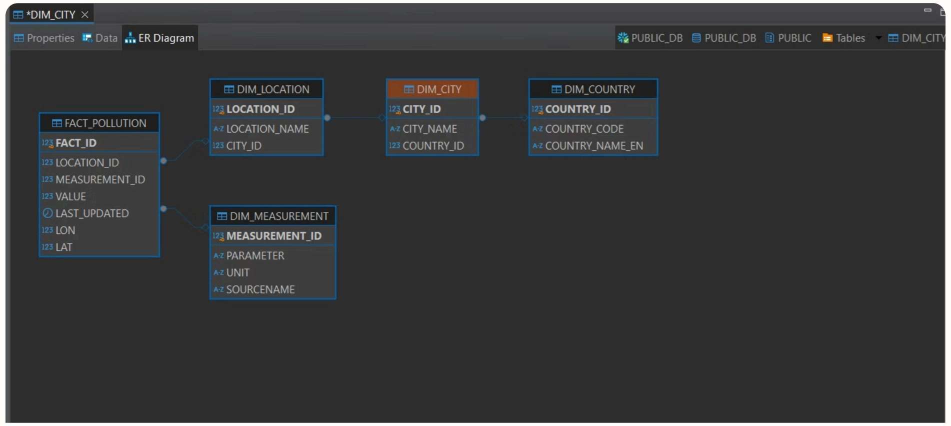
Step 8: Snowflake Schema Design and ER Diagram

To effectively analyze and retrieve valuable insights from our pollution data, we'll implement a Snowflake schema that leverages a star schema structure. This design centers around a pollution fact table, which stores the core pollution data. This fact table is then linked to several dimensional tables, providing contextual information about the pollution measurements.

The dimensional tables we'll create include a location dimension table, a city dimension table, and a country dimension table. These tables are interconnected, with the location dimension table linking to the city dimension table, and the city dimension table linking to the country dimension table. This hierarchical structure allows us to easily filter and aggregate data based on location details.

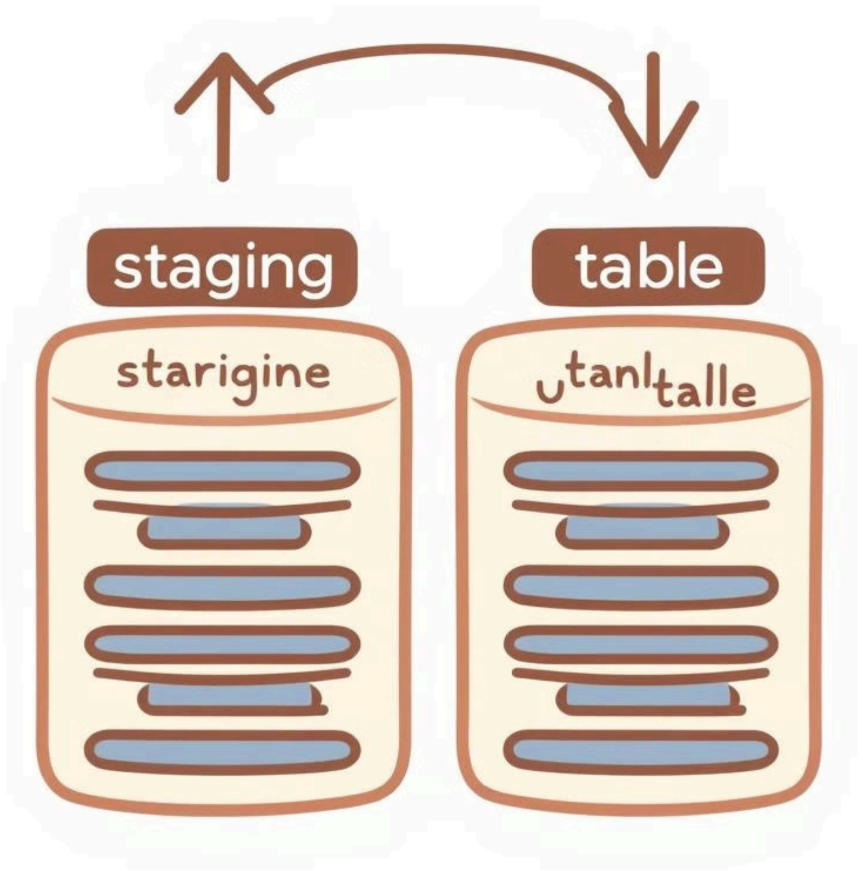
Furthermore, the pollution fact table is also linked to a measurement parameters dimension table. This table contains information about the different measurement parameters used for pollution monitoring, such as particulate matter, ozone, and sulfur dioxide. This link enables us to analyze pollution trends for specific parameters.

The ER diagram below provides a visual representation of this schema design, clearly showcasing the relationships between the pollution fact table and the dimensional tables. This diagram serves as a blueprint for building our Snowflake database, ensuring a structured and efficient data storage and retrieval process.



ER Diagram

This diagram shows the relationships between the tables in our star schema model. This model uses a central fact table to store the core data, with surrounding dimensional tables providing additional context.



Step 9: Data Loading into Final Tables

Once the data is loaded into the staging table, we'll load it into the final fact and dimensional tables. This step involves selecting the relevant data from the staging table based on the star schema design. Data transformations, if necessary, will be performed during this loading process. We'll use SQL statements to load data into the final tables.

Here's an example of loading data into a final table from the staging table in Snowflake:

```
INSERT INTO final_table
SELECT *
FROM staging_table;
```

Step 10: Report Generation and Visualization

The final step involves creating a report for the Business Analyst, which will include insights and visualizations based on the data analysis. We'll use SQL queries to extract data from the final tables and then use Python libraries like Seaborn and Matplotlib to generate various visualizations, such as bar charts, line graphs, and scatter plots. This will provide the Business Analyst with valuable insights into the data.

Here's an example of how to create a visualization using Seaborn and Matplotlib:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Load data from Snowflake
data = snowflake_conn.cursor().execute('SELECT * FROM
final_table').fetchall()

# Create a bar chart
sns.barplot(x='category', y='value', data=data)
plt.title('Category vs Value')
plt.show()
```

This code snippet demonstrates a simple bar chart to illustrate a relationship between categories and values in the data. You can explore other visualizations using Seaborn and Matplotlib to present the data in a clear and insightful way.