Part 1 SQL Challenge (Converting parquet to csv) to use it easily in DBeaver

```
import pyarrow.parquet as pq
trips = pq.read_table('yellow_tripdata_2024-01.parquet')
trips = trips.to_pandas()
# Export to CSV
trips.to_csv('yellow_tripdata_2024-01.csv', index=False)
```

Part 2: Data Pipeline Design (Sudo code to outline my approach)

```
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.models import Variable
from datetime import datetime, timedelta
import pandas as pd
import pyarrow.parquet as pq
import requests
from sqlalchemy import create_engine
import logging

# PostgreSQL connection details
POSTGRES_URI =
'postgresql+psycopg2://username:password@localhost/dbname'

# Retrieve user-defined variables from Airflow's Variables feature
YEAR = Variable.get("taxi_data_year", default_var="2024")
MONTH = Variable.get("taxi_data_month", default_var="01")
MIN_FARE_AMOUNT = float(Variable.get("min_fare_amount",
default_var=10))  # Default filter for fares > $10
NOTIFICATION_EMAIL = Variable.get("notification_email",
default_var="mkharoof@gmail.com")

# Data file URL and name based on user inputs
DATA_URL =
f"https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_{YEA
R}-{MONTH}.parquet"
FILE_NAME = f"yellow_tripdata_{YEAR}-{MONTH}.parquet"

# Define helper functions

def download_data():
    """Download data file from the specified URL and save locally."""
```

```python
    try:
        response = requests.get(DATA_URL, stream=True)
        if response.status_code == 200:
            with open(FILE_NAME, 'wb') as f:
                f.write(response.content)
            logging.info("File downloaded successfully.")
        else:
            raise Exception(f"Failed to download data, status code:
{response.status_code}")
    except Exception as e:
        logging.error("Error during download:", exc_info=e)
        raise

def transform_data():
    """Transform data: filter trips with user-defined minimum fare,
rename columns, and apply data quality checks."""
    try:
        # Load the .parquet file
        trips = pq.read_table(FILE_NAME).to_pandas()

        # Filter for trips with fare amounts over the user-defined
minimum fare amount
        trips = trips[trips['fare_amount'] > MIN_FARE_AMOUNT]

        # Rename columns to align with target schema
        trips.rename(columns={
            'tpep_pickup_datetime': 'pickup_time',
            'tpep_dropoff_datetime': 'dropoff_time',
            'PULocationID': 'pickup_location',
            'DOLocationID': 'dropoff_location',
            'fare_amount': 'fare_amount'
        }, inplace=True)

        # Data Quality Checks
        trips.dropna(subset=['pickup_time', 'dropoff_time',
'fare_amount'], inplace=True)
        trips = trips[trips['pickup_time'] < trips['dropoff_time']]
        trips = trips[trips['fare_amount'] > 0]

        # Additional quality check - log the count of rows that pass
and fail checks
        logging.info(f"Rows after filtering: {len(trips)}")

        # Save transformed data locally for loading
        trips.to_csv('transformed_trips.csv', index=False)
        logging.info("Transformation complete and saved to CSV.")
    except Exception as e:
        logging.error("Error during transformation:", exc_info=e)
        raise
```

```
def load_data():
    """Load the transformed data into PostgreSQL in batches."""
    try:
        engine = create_engine(POSTGRES_URI)
        chunksize = 10000  # Adjust this based on system's memory
capacity

        # Read and load data in chunks
        for chunk in pd.read_csv('transformed_trips.csv',
chunksize=chunksize):
            chunk.to_sql('taxi_trips', engine, if_exists='append',
index=False)
            logging.info(f"Loaded chunk of {len(chunk)} rows to the
database.")

        engine.dispose()
        logging.info("Data loading complete.")
    except Exception as e:
        logging.error("Error during data loading:", exc_info=e)
        raise

# Airflow DAG definition
default_args = {
    'owner': 'airflow',
    'start_date': datetime(2023, 1, 1),
    'retries': 3,
    'retry_delay': timedelta(minutes=5),
    'email_on_failure': True,
    'email': NOTIFICATION_EMAIL  # Email notification based on user
input
}

with DAG(
    'nyc_taxi_pipeline',
    default_args=default_args,
    description='A pipeline for NYC Taxi data ETL process with user-
defined inputs',
    schedule='@monthly',
    catchup=False,
) as dag:

    download_task = PythonOperator(
        task_id='download_data',
        python_callable=download_data
    )

    transform_task = PythonOperator(
        task_id='transform_data',
        python_callable=transform_data
    )
    load_task = PythonOperator(
        task_id='load_data',
        python_callable=load_data
    )

    download_task >> transform_task >> load_task
```

Part 3: Data Processing with Python (My code approach)

```
import os
import requests
import pandas as pd
import pyarrow.parquet as pq
from sqlalchemy import create_engine
import datetime

# Configuration
YEAR = "2024"       # Set the year
MONTH = "01"        # Set the month
FILE_NAME = f"yellow_tripdata_{YEAR}-{MONTH}.parquet"
DATA_URL = f"https://d37ci6vzurychx.cloudfront.net/trip-
data/{FILE_NAME}"

# PostgreSQL connection details
DB_NAME = "postgres"          # Database name
DB_USER = "postgres"          # Username
DB_PASSWORD = "ROOT"          # Password
DB_HOST = "localhost"         # Server host
DB_PORT = "5432"              # Port
TABLE_NAME = "nyc_taxi_trips"

# 1. Download the specified month's dataset if it is not already
available locally
def download_data():
    if not os.path.exists(FILE_NAME):
        print(f"Downloading {FILE_NAME}...")
        response = requests.get(DATA_URL, stream=True)
        if response.status_code == 200:
            with open(FILE_NAME, 'wb') as f:
                f.write(response.content)
            print("Download complete.")
        else:
            raise Exception(f"Failed to download data:
{response.status_code}")
    else:
        print(f"{FILE_NAME} already exists locally.")

# 2. Connect to PostgreSQL database and load the data
def load_data_to_db():
    print("Loading data into PostgreSQL...")
    # Establish a database connection
    engine = create_engine(f"postgresql+psycopg2://{DB_USER}:
{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/{DB_NAME}")

    # Read the data from the parquet file
    trips = pq.read_table(FILE_NAME).to_pandas()

    # Write the DataFrame to PostgreSQL
    trips.to_sql(TABLE_NAME, engine, if_exists='replace', index=False)
    print("Data loaded into PostgreSQL.")

# 3. Calculate the average fare per day of the week for the specified
month
def calculate_average_fare():
    print("Calculating average fare per day of the week...")
    # Establish a database connection
    engine = create_engine(f"postgresql+psycopg2://{DB_USER}:
{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/{DB_NAME}")

    # Query to calculate the average fare per day of the week
    query = f"""
        SELECT
            EXTRACT(DOW FROM tpep_pickup_datetime) AS day_of_week,
            AVG(fare_amount) AS avg_fare
        FROM {TABLE_NAME}
        WHERE EXTRACT(YEAR FROM tpep_pickup_datetime) = {YEAR}
          AND EXTRACT(MONTH FROM tpep_pickup_datetime) = {MONTH}
        GROUP BY day_of_week
        ORDER BY day_of_week;
    """
    avg_fare_per_day = pd.read_sql_query(query, engine)

    # Map day of the week to weekday names
    avg_fare_per_day['day_of_week'] =
avg_fare_per_day['day_of_week'].map({
        0: 'Sunday', 1: 'Monday', 2: 'Tuesday', 3: 'Wednesday',
        4: 'Thursday', 5: 'Friday', 6: 'Saturday'
    })

    # Close the engine connection
    engine.dispose()
    print("Average fare calculation complete.")
    return avg_fare_per_day

# 4. Generate a summary report and save it as a CSV file
```

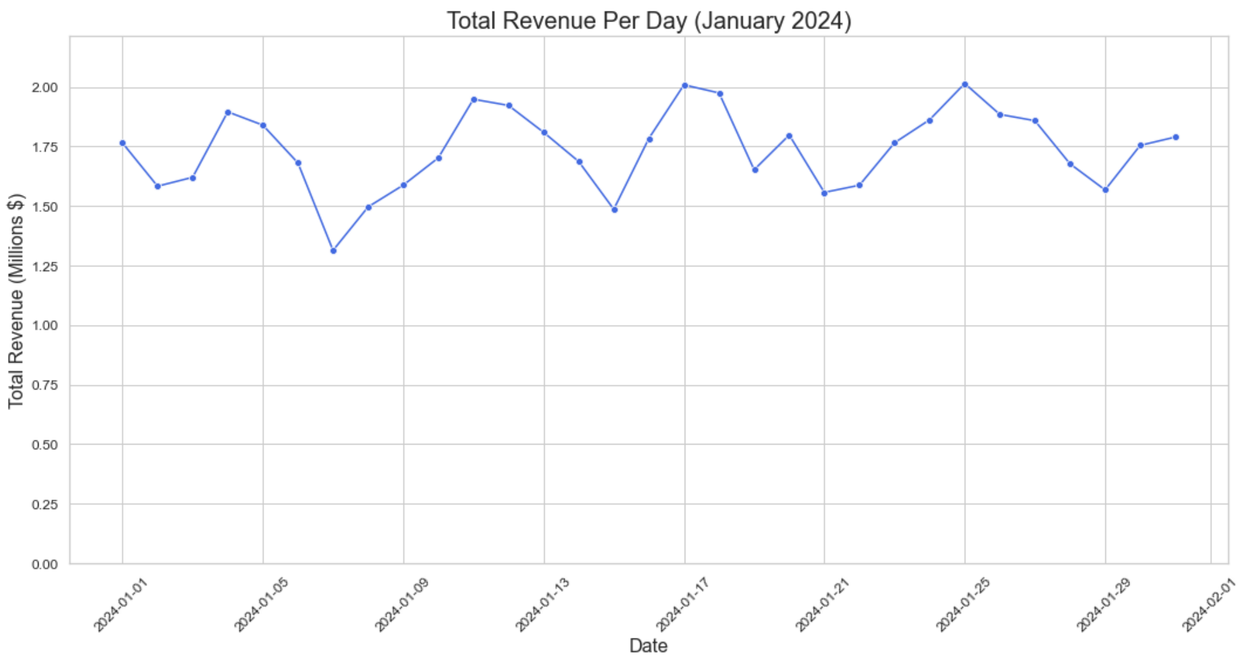
```
def save_to_csv(dataframe):
    output_file = "average_fare_per_day.csv"
    dataframe.to_csv(output_file, index=False)
    print(f"Report saved as {output_file}.")

# Run the entire workflow
if __name__ == "__main__":
    try:
        # Step 1: Download data if not already present
        download_data()

        # Step 2: Load data into PostgreSQL
        load_data_to_db()

        # Step 3: Calculate the average fare per day of the week
        avg_fare_per_day = calculate_average_fare()

        # Step 4: Save the results to a CSV file
        save_to_csv(avg_fare_per_day)

    except Exception as e:
        print(f"Error: {e}")
```

```
yellow_tripdata_2024-01.parquet already exists locally.
Loading data into PostgreSQL...
Data loaded into PostgreSQL.
Calculating average fare per day of the week...
Average fare calculation complete.
Report saved as average_fare_per_day.csv.
```

Part 4: Data Visualization for Revenue By Day in January 2024 in Millions (Dollars)

```
import pyarrow.parquet as pq
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the data from the .parquet file
data = pq.read_table('yellow_tripdata_2024-01.parquet').to_pandas()

# Convert the pickup datetime column and filter for dates in January
2024 only
data['tpep_pickup_datetime'] =
pd.to_datetime(data['tpep_pickup_datetime'])
data = data[(data['tpep_pickup_datetime'].dt.year == 2024) &
(data['tpep_pickup_datetime'].dt.month == 1)]
```

```
# Set the Seaborn style for better visuals
sns.set_theme(style="whitegrid")

# Prepare data for the visualization
# Group by date and calculate total revenue in millions
data['date'] = data['tpep_pickup_datetime'].dt.date
revenue_per_day = data.groupby('date')['fare_amount'].sum() /
1_000_000  # Convert to millions

# Create a single large plot for Total Revenue Per Day
plt.figure(figsize=(15, 8))  # Increased figure size for better
visibility
sns.lineplot(x=revenue_per_day.index, y=revenue_per_day.values,
marker='o', color="royalblue")
plt.title('Total Revenue Per Day (January 2024)', fontsize=20)
plt.xlabel('Date', fontsize=16)
plt.ylabel('Total Revenue (Millions $)', fontsize=16)  # Updated label
plt.xticks(rotation=45, fontsize=12)
plt.yticks(fontsize=12)
plt.ylim(0, revenue_per_day.max() * 1.1)  # Adjust y-axis limit for
padding

# Save and show the figure
plt.tight_layout()
plt.savefig('data_visualization.png')
plt.show()
```


Total Revenue Per Day (January 2024)
Total Revenue (Millions $)
Date