

✓ Overview of the Architecture and Approach

1. **Data Extraction:** The pipeline fetches commits from a GitHub repository using `GitHub's REST API`. We focus on commits from the past 6 months (180 days).
 2. **Transformation:** After retrieving the data, the commits are parsed and structured to be inserted into a database. Basic processing and cleaning occur during this step to ensure that missing values or errors are handled correctly.
 3. **Loading:** The pipeline stores commits in an SQLite database named `github_commits.db`.
 4. **Analysis:** After data is loaded into the database, the pipeline performs several analyses: Top 5 committers, Longest commit streak and Generates Commit heatmap.
 5. **Visualization:** A 2D heatmap and a 3D scatter plot heatmap is created, providing view of commit counts across days and hours.
-

✓ GENERAL CODE BLOCKS TO SETUP PIPELINE

Install dependencies:

```
!pip install requests
```

```
🔗 Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (2.32.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests) (2024.8.30)
```

Import necessary libraries:

```
import requests
import sqlite3
import os
from datetime import datetime, timedelta
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
```

Declare variables to be used in the next functions:

```
REPO_OWNER = "apache"
REPO_NAME = "airflow"
GITHUB_API_URL = "https://api.github.com"

headers = {
    "Authorization": "Bearer ghp_awRbNinerG2BTZ3M6vdc2Z1BI7534d378LcC"
}
since_date = (datetime.now() - timedelta(days=180)).isoformat()

url = f"{GITHUB_API_URL}/repos/{REPO_OWNER}/{REPO_NAME}/commits"
params = {
    "since": since_date,
    "per_page": 100
}

db_name="github_commits.db"
conn = sqlite3.connect(db_name)
cursor = conn.cursor()
```

✓ GITHUB REST API FUNCTION

- The GitHub API is accessed using the `requests` library.
- Commits are fetched in pages using the `per_page` and `since` parameters.
- We loop through paginated results and accumulate commits until all pages are retrieved

```
def fetch_commits():
    all_commits = []
    current_url = url
    while current_url:
        response = requests.get(current_url, headers=headers, params=params)
        response.raise_for_status()
        commits = response.json()
        all_commits.extend(commits)

        # Handle pagination
        current_url = response.links.get("next", {}).get("url")
    return all_commits
```

✓ DATABASE SET UP FUNCTIONS

SQLite is used because it's lightweight, easy to set up, and requires no server. It's perfect for small projects, offering fast queries, portability, and reliable data integrity with ACID transactions. This makes it ideal for storing and managing commit data efficiently.

✓ Function - `save_to_database(commits)`

- Creates a table in the SQLite database if it doesn't exist.
- Saves each commit with details like sha, author_name, commit_date, and message into the database.

```
def save_to_database(commits):
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS commits (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            sha TEXT UNIQUE NOT NULL,
            author_name TEXT,
            committer_name TEXT,
            commit_date TEXT,
            message TEXT
        )
    ''')

    for commit in commits:
        sha = commit.get("sha")
        author_name = commit.get("commit", {}).get("author", {}).get("name")
        committer_name = commit.get("commit", {}).get("committer", {}).get("name")
        commit_date = commit.get("commit", {}).get("committer", {}).get("date")
        message = commit.get("commit", {}).get("message")
        try:
            cursor.execute('''
                INSERT OR IGNORE INTO commits (sha, author_name, committer_name, commit_date, message)
                VALUES (?, ?, ?, ?, ?)
            ''', (sha, author_name, committer_name, commit_date, message))
        except sqlite3.IntegrityError as e:
            print(f"Error inserting commit {sha}: {e}")
    print(f"Saved {len(commits)} commits to {db_name}")
```

✓ Function - `delete_database(db_name)`

- Deletes the database file to start fresh, useful for cleanup or testing purposes.

```
def delete_database():
    db_name="github_commits.db"
```

```

if os.path.exists(db_name):
    try:
        os.remove(db_name)
        print(f"Database file '{db_name}' deleted successfully.")
    except OSError as e:
        print(f"Error deleting database file '{db_name}': {e}")
else:
    print(f"Database file '{db_name}' does not exist.")

```

✓ TASKS

✓ TASK 1: Get top 5 committers based on number of commits

Function - `get_top_committers()`

- Queries the database to get the top 5 committers (authors with the most commits).
- Returns the committer's name and commit count.

```

def get_top_committers():

    # SQL query to count commits by committer
    query = '''
        SELECT author_name, COUNT(*) as commit_count
        FROM commits
        WHERE author_name IS NOT NULL
        GROUP BY author_name
        ORDER BY commit_count DESC
        LIMIT ?
    '''

    try:
        cursor.execute(query, (5,))
        top_committers = cursor.fetchall()
    except sqlite3.Error as e:
        print(f"Error querying database: {e}")
        top_committers = []
    return top_committers

```

✓ TASK 2: Top committer with the longest commit day streak

Function - `get_longest_commit_streak_query()`

- This query finds the author with the longest streak of consecutive commits.

```

def get_longest_commit_streak_query():
    query = '''
        WITH Commit_Gaps AS (
            SELECT
                author_name,
                DATE(commit_date) AS commit_day,
                LAG(DATE(commit_date)) OVER (PARTITION BY author_name ORDER BY commit_date) AS previous_commit_day
            FROM commits
            WHERE author_name IS NOT NULL
        ),
        Streaks AS (
            SELECT
                author_name,
                commit_day,
                ROW_NUMBER() OVER (PARTITION BY author_name ORDER BY commit_day) -
                ROW_NUMBER() OVER (PARTITION BY author_name ORDER BY commit_day) AS streak_group
            FROM Commit_Gaps
            WHERE previous_commit_day IS NULL
                OR DATE(commit_day) != DATE(previous_commit_day, '+1 day')
        )
        SELECT
            author_name,
            MIN(commit_day) AS streak_start_day,

```

```

        MAX(commit_day) AS streak_end_day,
        COUNT(*) AS streak_length
    FROM Streaks
    GROUP BY author_name, streak_group
    ORDER BY streak_length DESC
    LIMIT 1;
'''
return query

```

Function - get_author_with_longest_streak()

- Executes the query to retrieve the author with the longest streak of consecutive commits.

```

def get_author_with_longest_streak():
    try:
        cursor.execute(get_longest_commit_streak_query())
        result = cursor.fetchone()
        return result

    except sqlite3.Error as e:
        print(f"Error executing query: {e}")
        return None

```

✓ TASK 3: Produce heatmap of no. of commit by days and 3 hours block

Function - get_commit_counts_by_user()

- This function retrieves the number of commits made by each user, grouped by day of the week and 3-hour time block.
- Uses SQLite strftime functions to group the commits by day and time.

```

def get_commit_counts_by_user():
    query = '''
        SELECT
            author_name,
            strftime('%w', commit_date) AS day_of_week,
            strftime('%H', commit_date) / 3 * 3 AS commit_hour_block,
            COUNT(*) AS commit_count
        FROM commits
        WHERE author_name IS NOT NULL
        GROUP BY author_name, day_of_week, commit_hour_block
        ORDER BY author_name, day_of_week, commit_hour_block;
    '''

    try:
        df = pd.read_sql_query(query, conn)
        return df
    except sqlite3.Error as e:
        print(f"Error executing query: {e}")
        return None

```

Function - generate_2d_heatmap(df)

- This function generates a 2D heatmap using matplotlib.
- It creates a matrix where rows represent days of the week and columns represent 3-hour time blocks.
- The values in the matrix are the commit counts.

```

def generate_2d_heatmap(df):
    # Pivot the DataFrame to create a structure suitable for 2D plotting
    pivot_df = df.pivot_table(
        index='day_of_week',
        columns='commit_hour_block',
        values='commit_count',
        aggfunc='sum',
        fill_value=0
    )

    # Convert day_of_week to actual day names for better readability

```

```

pivot_df.index = pivot_df.index.map({
    '0': 'Sun', '1': 'Mon', '2': 'Tue', '3': 'Wed',
    '4': 'Thu', '5': 'Fri', '6': 'Sat'
})

# Define the 3-hour blocks for the columns
pivot_df.columns = [f"{i}-{i+2}" for i in range(0, 24, 3)]

# Create the 2D heatmap plot using imshow
plt.figure(figsize=(6.3, 4.3))
plt.imshow(pivot_df.values, cmap='Blues', aspect='auto', origin='lower')
plt.colorbar(label='Commit Count')
plt.xticks(np.arange(len(pivot_df.columns)), pivot_df.columns, rotation=45)
plt.yticks(np.arange(len(pivot_df.index)), pivot_df.index)
plt.title('Commit Heatmap by Day of the Week and 3-Hour Block')
plt.xlabel('3-Hour Block')
plt.ylabel('Day of the Week')
plt.tight_layout()
plt.show()

```

Function - generate_3d_heatmap_scatter(df)

- This function creates a 3D scatter plot using matplotlib's Axes3D.
- Similar to the 2D heatmap, but represented in 3D with the Z-axis representing the commit count.

```

def generate_3d_heatmap_scatter(df):
    # Pivot the DataFrame to create a structure suitable for 3D plotting
    pivot_df = df.pivot_table(
        index='day_of_week',
        columns='commit_hour_block',
        values='commit_count',
        aggfunc='sum',
        fill_value=0
    )

    pivot_df.index = pivot_df.index.map({
        '0': 'Sun', '1': 'Mon', '2': 'Tue', '3': 'Wed',
        '4': 'Thu', '5': 'Fri', '6': 'Sat'
    })

    pivot_df.columns = [f"{i}-{i+2}" for i in range(0, 24, 3)]

    # Get the values to plot in the 3D scatter plot
    x_vals = np.array([i for i in range(len(pivot_df.columns)) for j in range(len(pivot_df.index))])
    y_vals = np.array([j for i in range(len(pivot_df.columns)) for j in range(len(pivot_df.index))])
    z_vals = pivot_df.values.flatten()

    fig = plt.figure(figsize=(10, 6))
    ax = fig.add_subplot(111, projection='3d')

    # Normalize the color map based on commit counts
    color_map = cm.ScalarMappable(cmap=cm.Greens_r, norm=plt.Normalize(vmin=np.min(z_vals), vmax=np.max(z_vals)))

    # Create the 3D scatter plot
    img = ax.scatter(x_vals, y_vals, z_vals, c=z_vals, cmap=cm.Greens_r, marker='o', s=100)

    plt.colorbar(color_map, ax=ax, label='Commit Count')
    ax.set_xlabel('3-Hour Block', labelpad=20)
    ax.set_ylabel('Day of the Week')
    ax.set_zlabel('Commit Count')
    ax.set_xticks(np.arange(len(pivot_df.columns)))
    ax.set_xticklabels(pivot_df.columns, rotation=45, ha="right", fontsize=10)
    ax.set_yticks(np.arange(len(pivot_df.index)))
    ax.set_yticklabels(pivot_df.index, fontsize=10)
    ax.set_title('3D Commit Heatmap by Day of the Week and 3-Hour Block')
    plt.show()

```

✓ MAIN EXECUTION

The pipeline begins by fetching commits from GitHub, then storing them in an SQLite database.

```
if __name__ == "__main__":
    try:
        print("Fetching commits from GitHub...")
        commits = fetch_commits()
        print(f"Fetched {len(commits)} commits.")

        print("Saving commits to SQLite database...")
        save_to_database(commits)
        print("Data saved successfully.")
    except requests.exceptions.RequestException as e:
        print(f"Error fetching data from GitHub API: {e}")
    except sqlite3.Error as e:
        print(f"Error working with SQLite database: {e}")
```

➔ Fetching commits from GitHub...
Fetched 2379 commits.
Saving commits to SQLite database...
Saved 2379 commits to github_commits.db
Data saved successfully.

✓ Result (Task 1): Find top 5 committers by number of commits.

```
top_committers = get_top_committers()
print("Top 5 Committers:")
for idx, (name, count) in enumerate(top_committers, start=1):
    print(f"{idx}. {name}: {count} commits")
```

➔ Top 5 Committers:
1. Jarek Potiuk: 284 commits
2. Kaxil Naik: 143 commits
3. Jens Scheffler: 140 commits
4. Daniel Standish: 85 commits
5. Wei Lee: 80 commits

✓ Result (Task 2): Determine the author with the longest streak of commits by day.

```
author_name, streak_start_day, streak_end_day, streak_length = get_author_with_longest_streak()

if author_name:
    print(f"Author: {author_name}")
    print(f"Streak Start Day: {streak_start_day}")
    print(f"Streak End Day: {streak_end_day}")
    print(f"Longest Streak Length: {streak_length} days")
else:
    print("No commits found or an error occurred.")
```

➔ Author: Jarek Potiuk
Streak Start Day: 2024-06-11
Streak End Day: 2024-12-03
Longest Streak Length: 197 days

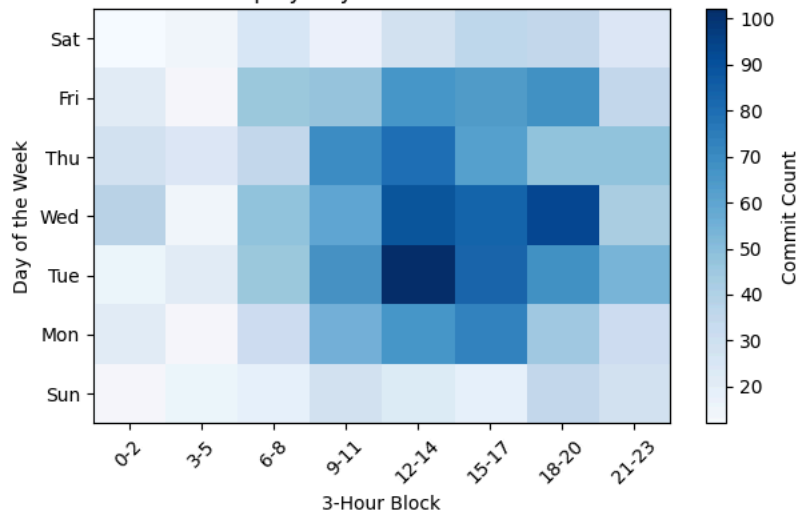
✓ Result (Task 3): Analyze commit patterns based on day and time using heatmaps.

```
df_commit_counts = get_commit_counts_by_user()

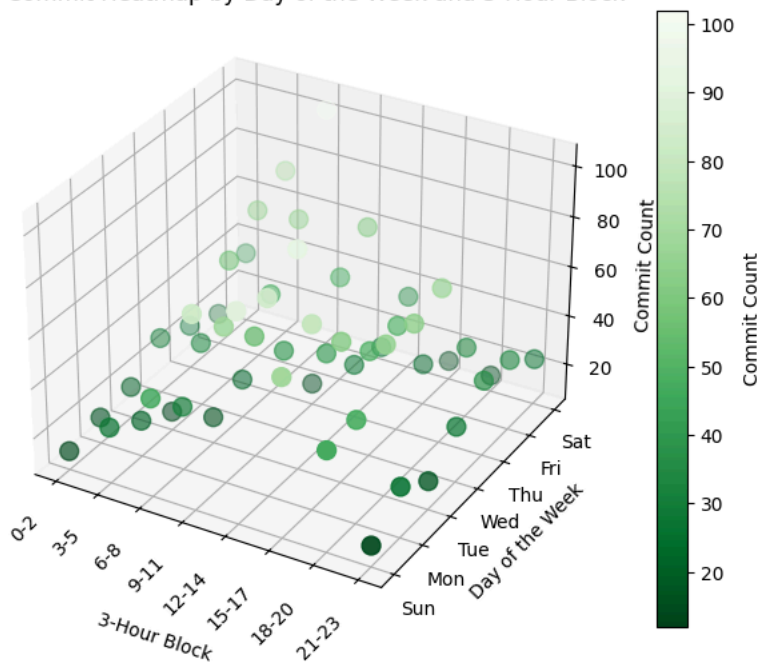
if df_commit_counts is not None:
    generate_2d_heatmap(df_commit_counts)
    print()
    generate_3d_heatmap_scatter(df_commit_counts)
```



Commit Heatmap by Day of the Week and 3-Hour Block



3D Commit Heatmap by Day of the Week and 3-Hour Block



End Execution

Finally, close the connection before end of execution.

```
conn.close()
delete_database()
print("Connection closed.")
print("Exit Program...")
```



```
Database file 'github_commits.db' deleted successfully.
Connection closed.
Exit Program...
```

