1. **Code Documentation for `generate_data.py`**

```
"""
generate_data.py

This script generates large synthetic datasets for performance testing.
The dataset includes columns with random integers and categories.

Usage:
    python generate_data.py --size <size_in_mb>
    Example: python generate_data.py --size 50

Arguments:
    --size: Size of the dataset in megabytes (MB).

Output:
    CSV file containing the generated dataset, saved in the `data/` folder.
"""

import pandas as pd
import numpy as np
import argparse
import os

def generate_large_dataset(num_rows: int) -> pd.DataFrame:
    """
    Generates a dataset with random integers and categories.

    Args:
        num_rows (int): The number of rows in the dataset.

    Returns:
        pd.DataFrame: Generated dataset with 'id', 'value', 'category', and 'description' columns.
    """
    data = {
        'id': np.arange(1, num_rows + 1),
        'value': np.random.randint(1, 200, size=num_rows),  # Random values between 1 and 200
        'category': np.random.choice(['A', 'B', 'C', 'D'], size=num_rows),  # Random categories
        'description': np.random.choice(['Lorem', 'Ipsum', 'Dolor', 'Sit', 'Amet'], size=num_rows)  #
Random text
    }
    return pd.DataFrame(data)

def save_dataset(dataset: pd.DataFrame, size_in_mb: int):
    """
    Saves the dataset to a CSV file.

    Args:
        dataset (pd.DataFrame): The dataset to be saved.
        size_in_mb (int): The approximate size of the dataset in MB, used for filename.
    """
    output_path = f"data/dataset_{size_in_mb}MB.csv"
    os.makedirs('data', exist_ok=True)  # Ensure 'data' folder exists
    dataset.to_csv(output_path, index=False)
    print(f"Dataset saved to {output_path}")

if __name__ == "__main__":
    # Argument parsing for size input
    parser = argparse.ArgumentParser(description='Generate a dataset of specified size.')
    parser.add_argument('--size', type=int, required=True, help='Size of dataset in megabytes
(MB)')

    args = parser.parse_args()

    # Estimate rows based on dataset size
    avg_row_size_bytes = 8 + 4 + 1 + 5  # Estimate row size in bytes
    num_rows = (args.size * 1e6) // avg_row_size_bytes

    # Generate and save dataset
    dataset = generate_large_dataset(int(num_rows))
    save_dataset(dataset, args.size)
```

2. **Code Documentation for** `process_data.py`

```
"""
process_data.py

This script processes a large dataset, calculating summary statistics such as the mean and standard deviation.
It supports both local file processing and cloud (Google Cloud Storage) file processing.

Usage:
    python process_data.py --local
    python process_data.py --cloud

Arguments:
    --local: Processes the dataset stored locally in the `data/` folder.
    --cloud: Processes the dataset stored in a Google Cloud Storage bucket.

Requirements:
    - Google Cloud SDK and credentials set up (for cloud processing).
"""

import pandas as pd
import timeit
import argparse

def process_data(file_path: str) -> pd.DataFrame:
    """
    Reads the dataset and calculates summary statistics.

    Args:
        file_path (str): Path to the dataset CSV file.

    Returns:
        pd.DataFrame: Summary statistics (mean, std, etc.) of the dataset.
    """
    print(f"Processing file: {file_path}")
    data = pd.read_csv(file_path)
    summary = data.describe()  # Calculates mean, std, min, max, etc.
    print("Summary statistics:\n", summary)
    return summary

def process_local_data():
    """
    Processes the dataset stored locally.
    """
    file_path = "data/dataset_50MB.csv"  # Local file path
    return process_data(file_path)

def process_cloud_data():
    """
    Processes the dataset stored in a Google Cloud Storage bucket.
    """
    from google.cloud import storage

    bucket_name = 'your-bucket-name'  # Replace with your bucket name
    file_name = 'dataset_50MB.csv'

    # Download the file from Google Cloud Storage
    storage_client = storage.Client()
    bucket = storage_client.bucket(bucket_name)
    blob = bucket.blob(file_name)
    file_path = f"data/{file_name}"
    blob.download_to_filename(file_path)

    print(f"Downloaded {file_name} from Google Cloud Storage.")
    return process_data(file_path)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Process dataset and calculate statistics.')
    parser.add_argument('--local', action='store_true', help='Process data stored locally.')
    parser.add_argument('--cloud', action='store_true', help='Process data from Google Cloud Storage.')

    args = parser.parse_args()

    if args.local:
        print("Processing local data...")
        process_local_data()
    elif args.cloud:
        print("Processing cloud data...")
        process_cloud_data()
    else:
        print("Please specify --local or --cloud to process data.")
```

3. **Code Documentation for `generate_bar_graph.py`**

```
"""
generate_bar_graph.py

This script generates bar graphs comparing the execution time and resource usage between the local and cloud setups.

Usage:
    python generate_bar_graph.py

Output:
    - A bar graph saved in the `barGraph/` folder comparing execution times for cloud and local setups.
"""

import matplotlib.pyplot as plt

def generate_bar_graph(local_time: float, cloud_time: float):
    """
    Generates a bar graph comparing local and cloud execution times.

    Args:
        local_time (float): Execution time for local processing.
        cloud_time (float): Execution time for cloud processing.

    Output:
        Saves a bar graph to the 'barGraph/' folder.
    """
    labels = ['Local', 'Cloud']
    times = [local_time, cloud_time]

    plt.bar(labels, times, color=['blue', 'green'])
    plt.ylabel('Execution Time (seconds)')
    plt.title('Performance Comparison: Cloud vs Local')

    plt.savefig('barGraph/performance_comparison.png')
    plt.show()
    print("Bar graph saved to 'barGraph/performance_comparison.png'")

if __name__ == "__main__":
    local_execution_time = 5.34  # Example time in seconds
    cloud_execution_time = 3.89  # Example time in seconds
    generate_bar_graph(local_execution_time, cloud_execution_time)
```