### Project Name: Providing a Complete Analytical Solution for an E-commerce Website

### Project Description:

We undertook a project for a startup e-commerce company that lacked a comprehensive system to manage and track their increasing data. Previously, the company stored their data in various formats such as Excel, CSV, flat files, and other in-house tools. As their data grew, they sought to implement a centralized system to effectively manage and analyze their data. Our objective was to deliver an end-to-end analytical solution, including the creation of a centralized (enterprise) data warehouse and dashboards for performance tracking.

### Data Analyst Role in the Project:

As a data analyst, my role involved multiple key responsibilities throughout the project:

1. \*\*Requirement Gathering and Documentation:\*\*

- Met with the client to understand their requirements and objectives.

- Prepared technical documentation detailing the amount of data, data sources, and formats used by the client.

2. \*\*Building the Enterprise Data Warehouse:\*\*

- Utilized SQL Server Integration Services (SSIS) to construct the ETL (Extract, Transform, Load) pipeline.

- Extracted data from multiple sources, including Excel, CSV, and flat files.

- Transformed the data to address anomalies and inconsistencies, as much of the data was manually entered.

- Loaded the cleaned and transformed data into a SQL Server database.

3. \*\*Data Preprocessing and Analysis:\*\*

- Conducted data preprocessing and analysis using Python.

- Addressed data cleaning tasks such as handling null values and fixing anomalies.

- Used libraries like Pandas, NumPy, Matplotlib, and Plotly for data analysis and visualization.

4. \*\*Dashboard Creation:\*\*

- Built interactive dashboards using Tableau and PowerBI.

- Created multiple dashboards to showcase various data aspects, including sales data, orders data, finance data, customer data, and product data.

### Example Scenario:

If the client wanted to analyze the sales of a particular product within a specific timeframe, the process would involve:

- Pulling the relevant data from the SQL Server database.

- Performing data cleaning and validation to check for null values or any anomalies.

- Preparing a report using Python with visualization libraries like Matplotlib or Plotly.

- Alternatively, building an interactive dashboard in Tableau or PowerBI to display the sales data.

### Tools and Technologies Used:

- \*\*ETL Pipeline:\*\* SSIS (SQL Server Integration Services)

- \*\*Database:\*\* SQL Server (alternatively, Oracle can be used)

- \*\*Data Preprocessing and Analysis:\*\* Python, Pandas, NumPy, Matplotlib, Plotly

- \*\*Dashboard Creation:\*\* Tableau, PowerBI

### Project Outcome:

The project successfully provided the startup e-commerce company with a robust and scalable analytical solution. The centralized data warehouse enabled efficient data management, while the interactive dashboards facilitated real-time performance tracking and informed decision-making. The end-to-end solution significantly improved the client's ability to manage and analyze their data, driving better business insights and strategies.

### Directory Structure and Files

1. \*\*Create the repository structure:\*\*

```plaintext

ecommerce-analytics-solution/

│

├── data/

│ ├── raw/

│ │ ├── sales\_data.csv

│ │ ├── orders\_data.csv

│ │ └── customer\_data.csv

│ └── processed/

│ ├── sales\_data\_cleaned.csv

│ ├── orders\_data\_cleaned.csv

│ └── customer\_data\_cleaned.csv

│

├── etl/

│ ├── extract.py

│ ├── transform.py

│ └── load.py

│

├── analysis/

│ ├── sales\_analysis.ipynb

│ ├── orders\_analysis.ipynb

│ └── customer\_analysis.ipynb

│

├── dashboards/

│ ├── sales\_dashboard.twb

│ ├── orders\_dashboard.twb

│ └── customer\_dashboard.twb

│

├── docs/

│ ├── requirements.md

│ ├── design.md

│ └── user\_guide.md

│

├── scripts/

│ ├── generate\_sample\_data.py

│ ├── start\_etl.sh

│ └── setup\_environment.sh

│

├── tests/

│ ├── test\_extract.py

│ ├── test\_transform.py

│ └── test\_load.py

│

├── .gitignore

├── README.md

└── requirements.txt

```

2. \*\*README.md\*\*

```markdown

# E-commerce Analytics Solution

This repository contains the code and documentation for providing a complete analytical solution for an e-commerce website. The project includes building an enterprise data warehouse, performing data analysis, and creating dashboards to track various metrics.

## Project Description

We worked with a startup e-commerce company to build a centralized system for managing and tracking their data. Previously, the data was stored in multiple formats such as Excel, CSV, and flat files. We provided an end-to-end solution from building an ETL pipeline to creating interactive dashboards.

## Data Analyst Role

As a Data Analyst, my responsibilities included:

- Understanding client requirements

- Preparing technical documentation

- Building an enterprise data warehouse

- Creating an ETL pipeline using SSIS

- Data preprocessing and analysis using Python

- Building dashboards in Tableau/PowerBI

## Data Sources

- Sales data

- Orders data

- Finance data

- Customer data

- Products data

## Tools Used

- SSIS for ETL

- SQL Server for data storage

- Python for data preprocessing and analysis

- Tableau/PowerBI for dashboard creation

## Directory Structure

- `data/`: Contains raw and processed data files.

- `etl/`: Contains scripts for extracting, transforming, and loading data.

- `analysis/`: Contains Jupyter notebooks for data analysis.

- `dashboards/`: Contains Tableau/PowerBI dashboard files.

- `docs/`: Contains project documentation files.

- `scripts/`: Contains utility scripts for data generation and environment setup.

- `tests/`: Contains unit tests for ETL processes.

- `README.md`: Project documentation.

- `requirements.txt`: List of Python dependencies.

## Setup

1. Clone the repository:

```bash

git clone https://github.com/yourusername/ecommerce-analytics-solution.git

```

2. Install the required Python packages:

```bash

pip install -r requirements.txt

```

3. Run the environment setup script:

```bash

./scripts/setup\_environment.sh

```

4. Generate sample data (optional):

```bash

python scripts/generate\_sample\_data.py

```

5. Update the ETL scripts in the `etl/` directory to connect to your data sources.

6. Run the ETL scripts to load data into the SQL Server:

```bash

./scripts/start\_etl.sh

```

7. Use the Jupyter notebooks in the `analysis/` directory to perform data analysis.

8. Open the Tableau/PowerBI files in the `dashboards/` directory to view the interactive dashboards.

## Usage

- Run the ETL scripts to extract, transform, and load data into the SQL Server.

- Use the Jupyter notebooks for data analysis and to generate reports.

- Open the Tableau/PowerBI dashboard files to interact with the visualizations.

## License

This project is licensed under the MIT License.

```

3. \*\*data/raw/sales\_data.csv\*\*

```csv

order\_id,product\_id,customer\_id,quantity,price,date

1,101,1001,2,20,2023-01-01

2,102,1002,1,10,2023-01-02

3,103,1003,5,50,2023-01-03

4,104,1004,3,30,2023-01-04

5,105,1005,4,40,2023-01-05

```

4. \*\*data/raw/orders\_data.csv\*\*

```csv

order\_id,customer\_id,order\_date,status

1,1001,2023-01-01,shipped

2,1002,2023-01-02,pending

3,1003,2023-01-03,delivered

4,1004,2023-01-04,cancelled

5,1005,2023-01-05,shipped

```

5. \*\*data/raw/customer\_data.csv\*\*

```csv

customer\_id,name,email,phone

1001,John Doe,john.doe@example.com,1234567890

1002,Jane Smith,jane.smith@example.com,0987654321

1003,Emily Davis,emily.davis@example.com,1122334455

1004,Michael Brown,michael.brown@example.com,2233445566

1005,Jessica Wilson,jessica.wilson@example.com,3344556677

```

6. \*\*etl/extract.py\*\*

```python

import pandas as pd

def extract\_data():

sales\_data = pd.read\_csv('../data/raw/sales\_data.csv')

orders\_data = pd.read\_csv('../data/raw/orders\_data.csv')

customer\_data = pd.read\_csv('../data/raw/customer\_data.csv')

return sales\_data, orders\_data, customer\_data

if \_\_name\_\_ == "\_\_main\_\_":

sales\_data, orders\_data, customer\_data = extract\_data()

print(sales\_data.head())

print(orders\_data.head())

print(customer\_data.head())

```

7. \*\*etl/transform.py\*\*

```python

import pandas as pd

def transform\_data(sales\_data, orders\_data, customer\_data):

# Example transformation: Remove rows with missing values

sales\_data\_cleaned = sales\_data.dropna()

orders\_data\_cleaned = orders\_data.dropna()

customer\_data\_cleaned = customer\_data.dropna()

# Save cleaned data

sales\_data\_cleaned.to\_csv('../data/processed/sales\_data\_cleaned.csv', index=False)

orders\_data\_cleaned.to\_csv('../data/processed/orders\_data\_cleaned.csv', index=False)

customer\_data\_cleaned.to\_csv('../data/processed/customer\_data\_cleaned.csv', index=False)

if \_\_name\_\_ == "\_\_main\_\_":

sales\_data = pd.read\_csv('../data/raw/sales\_data.csv')

orders\_data = pd.read\_csv('../data/raw/orders\_data.csv')

customer\_data = pd.read\_csv('../data/raw/customer\_data.csv')

transform\_data(sales\_data, orders\_data, customer\_data)

```

8. \*\*etl/load.py\*\*

```python

import pyodbc

import pandas as pd

def load\_data():

conn = pyodbc.connect('DRIVER={SQL Server};SERVER=your\_server\_name;DATABASE=your\_database\_name;UID=your\_username;PWD=your\_password')

cursor = conn.cursor()

# Load sales data

sales\_data = pd.read\_csv('../data/processed/sales\_data\_cleaned.csv')

for index, row in sales\_data.iterrows():

cursor.execute("INSERT INTO Sales (order\_id, product\_id, customer\_id, quantity, price, date) VALUES (?, ?, ?, ?, ?, ?)",

row['order\_id'], row['product\_id'], row['customer\_id'], row['quantity'], row['price'], row['date'])

# Load orders data

orders\_data = pd.read\_csv('../data/processed/orders\_data\_cleaned.csv')

for index, row in orders\_data.iterrows():

cursor.execute("INSERT INTO Orders (order\_id, customer\_id, order\_date, status) VALUES (?, ?, ?, ?)",

row['order\_id'], row['customer\_id'], row['order\_date'], row['status'])

# Load customer data

customer\_data = pd.read\_csv('../data/processed/customer\_data\_cleaned.csv')

for index, row in customer\_data.iterrows():

cursor.execute("INSERT INTO Customers (customer\_id, name, email, phone) VALUES (?, ?, ?, ?)",

row['customer\_id'], row['name'], row['email'], row['phone'])

conn.commit()

conn.close()

if \_\_name\_\_ == "\_\_main\_\_":

load\_data()

```

9. \*\*analysis/sales\_analysis.ipynb\*\*

```python

import pandas as pd

import matplotlib.pyplot as plt

# Load cleaned sales data

sales\_data = pd.read\_csv('../data/processed/sales\_data\_cleaned.csv')

# Perform analysis

sales\_summary = sales\_data.groupby('product\_id').agg({'quantity': 'sum', 'price': 'sum'}).reset\_index()

# Plot sales summary

plt.figure(figsize=(10, 5))

plt.bar(sales\_summary['product\_id'], sales\_summary['price'])

plt.xlabel('Product ID')

plt.ylabel('Total Sales')

plt.title('Total Sales by Product')

plt.show()

```

10. \*\*analysis/orders\_analysis.ipynb\*\*

```python

import pandas as pd

import matplotlib.pyplot as plt

# Load cleaned orders data

orders\_data = pd.read\_csv('../data/processed/orders\_data\_cleaned.csv')

# Perform analysis

order\_status\_summary = orders\_data['status'].value\_counts().reset\_index()

order\_status\_summary.columns = ['status', 'count']

# Plot order status summary

plt.figure(figsize=(10, 5))

plt.bar(order\_status\_summary['status'], order\_status\_summary['count'])

plt.xlabel('Order Status')

plt.ylabel('Count')

plt.title('Order Status Distribution')

plt.show()

```

11. \*\*analysis/customer\_analysis.ipynb\*\*

```python

import pandas as pd

import matplotlib.pyplot as plt

# Load cleaned customer data

customer\_data = pd.read\_csv('../data/processed/customer\_data\_cleaned.csv')

# Perform analysis

customer\_summary = customer\_data['customer\_id'].value\_counts().reset\_index()

customer\_summary.columns = ['customer\_id', 'count']

# Plot customer summary

plt.figure(figsize=(10, 5))

plt.bar(customer\_summary['customer\_id'], customer\_summary['count'])

plt.xlabel('Customer ID')

plt.ylabel('Count')

plt.title('Customer Distribution')

plt.show()

```

12. \*\*scripts/generate\_sample\_data.py\*\*

```python

import pandas as pd

import numpy as np

# Generate sample sales data

sales\_data = pd.DataFrame({

'order\_id': range(1, 101),

'product\_id': np.random.randint(100, 200, size=100),

'customer\_id': np.random.randint(1000, 1100, size=100),

'quantity': np.random.randint(1, 10, size=100),

'price': np.random.randint(10, 100, size=100),

'date': pd.date\_range(start='2023-01-01', periods=100, freq='D')

})

# Generate sample orders data

orders\_data = pd.DataFrame({

'order\_id': range(1, 101),

'customer\_id': np.random.randint(1000, 1100, size=100),

'order\_date': pd.date\_range(start='2023-01-01', periods=100, freq='D'),

'status': np.random.choice(['shipped', 'pending', 'delivered', 'cancelled'], size=100)

})

# Generate sample customer data

customer\_data = pd.DataFrame({

'customer\_id': range(1000, 1100),

'name': [f'Customer {i}' for i in range(1000, 1100)],

'email': [f'customer{i}@example.com' for i in range(1000, 1100)],

'phone': [f'123-456-789{i % 10}' for i in range(1000, 1100)]

})

# Save to CSV

sales\_data.to\_csv('../data/raw/sales\_data.csv', index=False)

orders\_data.to\_csv('../data/raw/orders\_data.csv', index=False)

customer\_data.to\_csv('../data/raw/customer\_data.csv', index=False)

print("Sample data generated and saved.")

```

13. \*\*scripts/start\_etl.sh\*\*

```bash

#!/bin/bash

echo "Starting ETL process..."

python ../etl/extract.py

python ../etl/transform.py

python ../etl/load.py

echo "ETL process completed."

```

14. \*\*scripts/setup\_environment.sh\*\*

```bash

#!/bin/bash

echo "Setting up the environment..."

# Install Python dependencies

pip install -r ../requirements.txt

# Additional setup commands can go here

echo "Environment setup completed."

```

15. \*\*tests/test\_extract.py\*\*

```python

import unittest

import pandas as pd

from etl.extract import extract\_data

class TestExtract(unittest.TestCase):

def test\_extract\_data(self):

sales\_data, orders\_data, customer\_data = extract\_data()

self.assertIsInstance(sales\_data, pd.DataFrame)

self.assertIsInstance(orders\_data, pd.DataFrame)

self.assertIsInstance(customer\_data, pd.DataFrame)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

```

16. \*\*tests/test\_transform.py\*\*

```python

import unittest

import pandas as pd

from etl.transform import transform\_data

class TestTransform(unittest.TestCase):

def test\_transform\_data(self):

sales\_data = pd.read\_csv('../data/raw/sales\_data.csv')

orders\_data = pd.read\_csv('../data/raw/orders\_data.csv')

customer\_data = pd.read\_csv('../data/raw/customer\_data.csv')

transform\_data(sales\_data, orders\_data, customer\_data)

sales\_data\_cleaned = pd.read\_csv('../data/processed/sales\_data\_cleaned.csv')

orders\_data\_cleaned = pd.read\_csv('../data/processed/orders\_data\_cleaned.csv')

customer\_data\_cleaned = pd.read\_csv('../data/processed/customer\_data\_cleaned.csv')

self.assertFalse(sales\_data\_cleaned.empty)

self.assertFalse(orders\_data\_cleaned.empty)

self.assertFalse(customer\_data\_cleaned.empty)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

```

17. \*\*tests/test\_load.py\*\*

```python

import unittest

import pyodbc

from etl.load import load\_data

class TestLoad(unittest.TestCase):

def test\_load\_data(self):

# Connect to the database

conn = pyodbc.connect('DRIVER={SQL Server};SERVER=your\_server\_name;DATABASE=your\_database\_name;UID=your\_username;PWD=your\_password')

cursor = conn.cursor()

# Load data

load\_data()

# Check if data is loaded into the database

cursor.execute("SELECT COUNT(\*) FROM Sales")

sales\_count = cursor.fetchone()[0]

cursor.execute("SELECT COUNT(\*) FROM Orders")

orders\_count = cursor.fetchone()[0]

cursor.execute("SELECT COUNT(\*) FROM Customers")

customers\_count = cursor.fetchone()[0]

self.assertGreater(sales\_count, 0)

self.assertGreater(orders\_count, 0)

self.assertGreater(customers\_count, 0)

conn.close()

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

```

18. \*\*.gitignore\*\*

```

# Byte-compiled / optimized / DLL files

\_\_pycache\_\_/

\*.py[cod]

\*$py.class

# C extensions

\*.so

# Packages

\*.egg

\*.egg-info/

dist/

build/

eggs/

parts/

bin/

var/

sdist/

develop-eggs/

.installed.cfg

lib/

lib64/

\_\_pypackages\_\_/

# Installer logs

pip-log.txt

pip-delete-this-directory.txt

# Unit test / coverage reports

htmlcov/

.tox/

.nox/

.coverage

coverage.\*

.cache

nosetests.xml

coverage.xml

\*.cover

.hypothesis/

.pytest\_cache/

# Jupyter Notebook

.ipynb\_checkpoints

# Environments

.env

.venv

env/

venv/

ENV/

env.bak/

venv.bak/

# PyCharm

.idea/

# VSCode

.vscode/

```

19. \*\*docs/requirements.md\*\*

```markdown

# Project Requirements

This document outlines the requirements for the E-commerce Analytics Solution project.

## Functional Requirements

1. \*\*Data Extraction\*\*

- Extract data from multiple sources (CSV, Excel, databases).

- Handle different data formats and ensure data consistency.

2. \*\*Data Transformation\*\*

- Clean and preprocess data.

- Handle missing values, duplicates, and data anomalies.

- Perform necessary data transformations.

3. \*\*Data Loading\*\*

- Load data into a centralized SQL Server database.

- Ensure data integrity during the loading process.

4. \*\*Data Analysis\*\*

- Perform data analysis using Python.

- Generate reports and visualizations.

5. \*\*Dashboards\*\*

- Create interactive dashboards using Tableau/PowerBI.

- Provide insights into sales, orders, and customer data.

## Non-Functional Requirements

1. \*\*Performance\*\*

- Ensure the ETL process is efficient and can handle large volumes of data.

- Optimize database queries for fast data retrieval.

2. \*\*Scalability\*\*

- Design the solution to accommodate future growth in data volume and complexity.

3. \*\*Security\*\*

- Implement data security measures to protect sensitive information.

- Ensure secure access to the database and dashboards.

4. \*\*Usability\*\*

- Provide user-friendly interfaces for data analysis and dashboard viewing.

- Ensure documentation is clear and comprehensive.

## Tools and Technologies

- SSIS for ETL

- SQL Server for data storage

- Python for data preprocessing and analysis

- Tableau/PowerBI for dashboard creation

```

20. \*\*docs/design.md\*\*

```markdown

# Design Document

This document outlines the design of the E-commerce Analytics Solution project.

## Architecture

The project follows a typical ETL (Extract, Transform, Load) architecture, where data is extracted from various sources, transformed to ensure consistency and quality, and then loaded into a centralized database for analysis and reporting.

## Data Flow

1. \*\*Data Extraction\*\*

- Extract data from CSV files (sales\_data.csv, orders\_data.csv, customer\_data.csv).

- Use Python scripts for extraction.

2. \*\*Data Transformation\*\*

- Clean and preprocess the extracted data.

- Handle missing values, duplicates, and anomalies.

- Use Python scripts for transformation.

3. \*\*Data Loading\*\*

- Load the cleaned data into a SQL Server database.

- Use Python scripts with pyodbc for loading.

4. \*\*Data Analysis\*\*

- Perform data analysis using Jupyter notebooks and Python libraries

(pandas, matplotlib).

- Generate insights and visualizations.

5. \*\*Dashboards\*\*

- Create interactive dashboards using Tableau/PowerBI.

- Provide visual insights into sales, orders, and customer data.

## Database Schema

### Sales Table

| Column Name | Data Type | Description |

|-------------|-----------|-------------|

| order\_id | INT | Unique identifier for each order |

| product\_id | INT | Unique identifier for each product |

| customer\_id | INT | Unique identifier for each customer |

| quantity | INT | Quantity of the product ordered |

| price | FLOAT | Price of the product |

| date | DATE | Date of the order |

### Orders Table

| Column Name | Data Type | Description |

|-------------|-----------|-------------|

| order\_id | INT | Unique identifier for each order |

| customer\_id | INT | Unique identifier for each customer |

| order\_date | DATE | Date of the order |

| status | VARCHAR | Status of the order (shipped, pending, delivered, cancelled) |

### Customers Table

| Column Name | Data Type | Description |

|-------------|-----------|-------------|

| customer\_id | INT | Unique identifier for each customer |

| name | VARCHAR | Name of the customer |

| email | VARCHAR | Email of the customer |

| phone | VARCHAR | Phone number of the customer |

## ETL Process

1. \*\*Extract\*\*

- Use `extract.py` to read data from CSV files and load them into pandas DataFrames.

2. \*\*Transform\*\*

- Use `transform.py` to clean and preprocess the data.

- Handle missing values, duplicates, and anomalies.

- Save the cleaned data into processed CSV files.

3. \*\*Load\*\*

- Use `load.py` to load the cleaned data into the SQL Server database.

- Insert data into Sales, Orders, and Customers tables.

## Dashboards

1. \*\*Sales Dashboard\*\*

- Visualize total sales, sales by product, and sales trends over time.

2. \*\*Orders Dashboard\*\*

- Visualize order status distribution, orders by date, and customer order history.

3. \*\*Customer Dashboard\*\*

- Visualize customer distribution, top customers, and customer order patterns.

## Scripts

1. \*\*generate\_sample\_data.py\*\*

- Generates sample data for testing and development.

2. \*\*start\_etl.sh\*\*

- Starts the ETL process by running the extract, transform, and load scripts.

3. \*\*setup\_environment.sh\*\*

- Sets up the development environment by installing required dependencies.

## Testing

1. \*\*test\_extract.py\*\*

- Unit tests for the data extraction process.

2. \*\*test\_transform.py\*\*

- Unit tests for the data transformation process.

3. \*\*test\_load.py\*\*

- Unit tests for the data loading process.

```

21. \*\*docs/user\_guide.md\*\*

```markdown

# User Guide

This document provides instructions on how to use the E-commerce Analytics Solution project.

## Prerequisites

- Python 3.x installed on your system

- SQL Server installed and running

- Tableau/PowerBI installed for dashboard viewing

## Setup

1. Clone the repository:

```bash

git clone https://github.com/yourusername/ecommerce-analytics-solution.git

```

2. Navigate to the repository directory:

```bash

cd ecommerce-analytics-solution

```

3. Install the required Python packages:

```bash

pip install -r requirements.txt

```

4. Run the environment setup script:

```bash

./scripts/setup\_environment.sh

```

5. Generate sample data (optional):

```bash

python scripts/generate\_sample\_data.py

```

## Running the ETL Process

1. Update the ETL scripts in the `etl/` directory to connect to your data sources and database.

2. Run the ETL process:

```bash

./scripts/start\_etl.sh

```

This will extract data from the raw data files, transform and clean the data, and load it into the SQL Server database.

## Data Analysis

1. Open the Jupyter notebooks in the `analysis/` directory:

```bash

jupyter notebook analysis/sales\_analysis.ipynb

jupyter notebook analysis/orders\_analysis.ipynb

jupyter notebook analysis/customer\_analysis.ipynb

```

2. Perform data analysis and generate reports and visualizations.

## Dashboards

1. Open the Tableau/PowerBI files in the `dashboards/` directory:

- sales\_dashboard.twb

- orders\_dashboard.twb

- customer\_dashboard.twb

2. Interact with the dashboards to explore insights into sales, orders, and customer data.

## Testing

1. Run the unit tests in the `tests/` directory:

```bash

python -m unittest discover -s tests

```

2. Verify that all tests pass successfully.

## Contributing

1. Fork the repository.

2. Create a new branch:

```bash

git checkout -b feature/your-feature-name

```

3. Make your changes and commit them:

```bash

git commit -m "Add some feature"

```

4. Push to the branch:

```bash

git push origin feature/your-feature-name

```

5. Open a pull request.

## License

This project is licensed under the MIT License.

```