Data Engineering Project Report

Project Overview:

This project demonstrates the ability to establish a connection between a Python application and a SQL Server database using the pyodbc library. It showcases:

- SQL Server database setup with a user data table.

- Insertion of synthetic user data using SQL scripts.

- Retrieval and display of data using Python.

- Robust error handling and connection management.

The use case simulates a common data engineering task: connecting ETL scripts to databases, validating connectivity, and performing sample data extraction.

Step 1: Setting Up the Python Environment

To connect Python to a SQL Server, we typically use the **pyodbc** library.

Installing the required package:

```
pip install pyodbc
```

Step 2: Creating the SQL Database and Table

The SQL Server database used is 'tws', and the table name is 'users'. Data was inserted into this table using the SQL script (Employee.sql). The users table includes the following columns:

- id: Primary ID (may be null for some rows)

- name: Name of the user
- email: Email address
- city: User's city
- year: Some relevant year (used as date)
- new_id: Alternate unique ID

The following script populates the users table with over 100 rows of sample data.

See file: Employees.sql

Required information for the connection:

- **IP Address of SQL Server: `xxx.xxx.xxx.xxx`**
- **Database name: `DB`**
- **Username: `Username`**
- **Password: `Password123`**

Connection String Format for SQL Server using pyodbc:

```
import pyodbc

# Define connection details
server = 'xxx.xxx.xxx.xxx'
database = 'CompanyDB'
username = 'YourUsername'
password = 'YourPassword123'

# Define connection string
conn_str = (
    f'DRIVER={{ODBC Driver 17 for SQL Server}};'
    f'SERVER={server};'
    f'DATABASE={database};'
    f'UID={username};'
    f'PWD={password}'
)

# Attempt connection
try:
    conn = pyodbc.connect(conn_str)
    print("Connection successful!")
except Exception as e:
    print(" Connection failed:", e)
```

Step 3: Python Script to Connect & Fetch Data

A Python script was written to:

- Connect to the database using the pyodbc driver
- Run a query to retrieve the top 5 rows from the users table
- Print the results to validate the connection and data retrieval

See file: doc-script.py

Code Overview:
server = '192.168.0.20'
database = 'CompanyDB'
username = 'Vaish22'
password = 'V@aish22'

The output confirms successful connection to SQL Server and displays the first 5 rows from the table.

```
import pyodbc

# Step 1: Define credentials and server details
server = 'xxx.xxx.xxx.xxx'
database = 'DB'
```

```python
username = 'Username'
password = 'Password123'

# Step 2: Define connection string
conn_str = (
    f'DRIVER={{ODBC Driver 17 for SQL Server}};'
    f'SERVER={server};'
    f'DATABASE={database};'
    f'UID={username};'
    f'PWD={password}'
)

try:
    # Step 3: Establish the connection
    conn = pyodbc.connect(conn_str)
    print(" Connected to SQL Server.")

    # Step 4: Create a cursor and run the query
    cursor = conn.cursor()
    query = "SELECT TOP 5 * FROM users"
    cursor.execute(query)

    # Step 5: Fetch and print rows
    rows = cursor.fetchall()
    print("\n First 5 rows from users table:")
    for row in rows:
        print(row)

    # Step 6: Clean up
    cursor.close()
    conn.close()

except Exception as e:
    print(" Error occurred:", e)
```

Step 4: <u>Error Handling and Cleanup</u>

Robust error handling is implemented using try-except-finally:

- try: Attempts to connect and query
- except: Catches connection or execution errors
- finally: Ensures the connection is closed gracefully

```
except pyodbc.Error as err:
        print(" Database connection or query failed:")
        print(err)
```

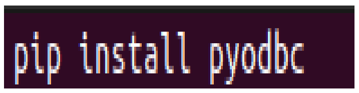
This ensures the script does not crash and provides meaningful feedback.

What we should ensure:

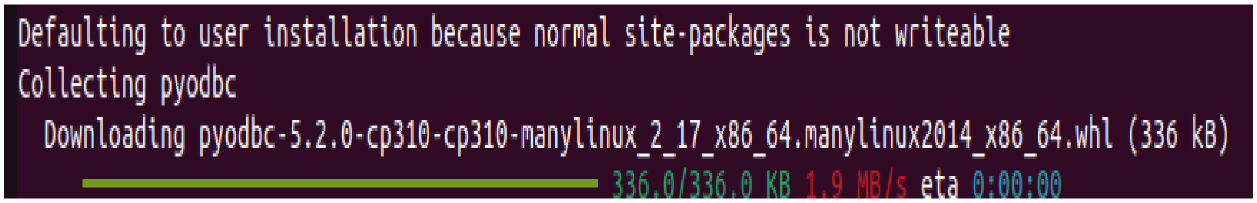
- Correct **IP address**
- SQL Server is reachable from your system
- Correct **ODBC driver** (e.g., "ODBC Driver 17 for SQL Server")
- Your credentials are valid

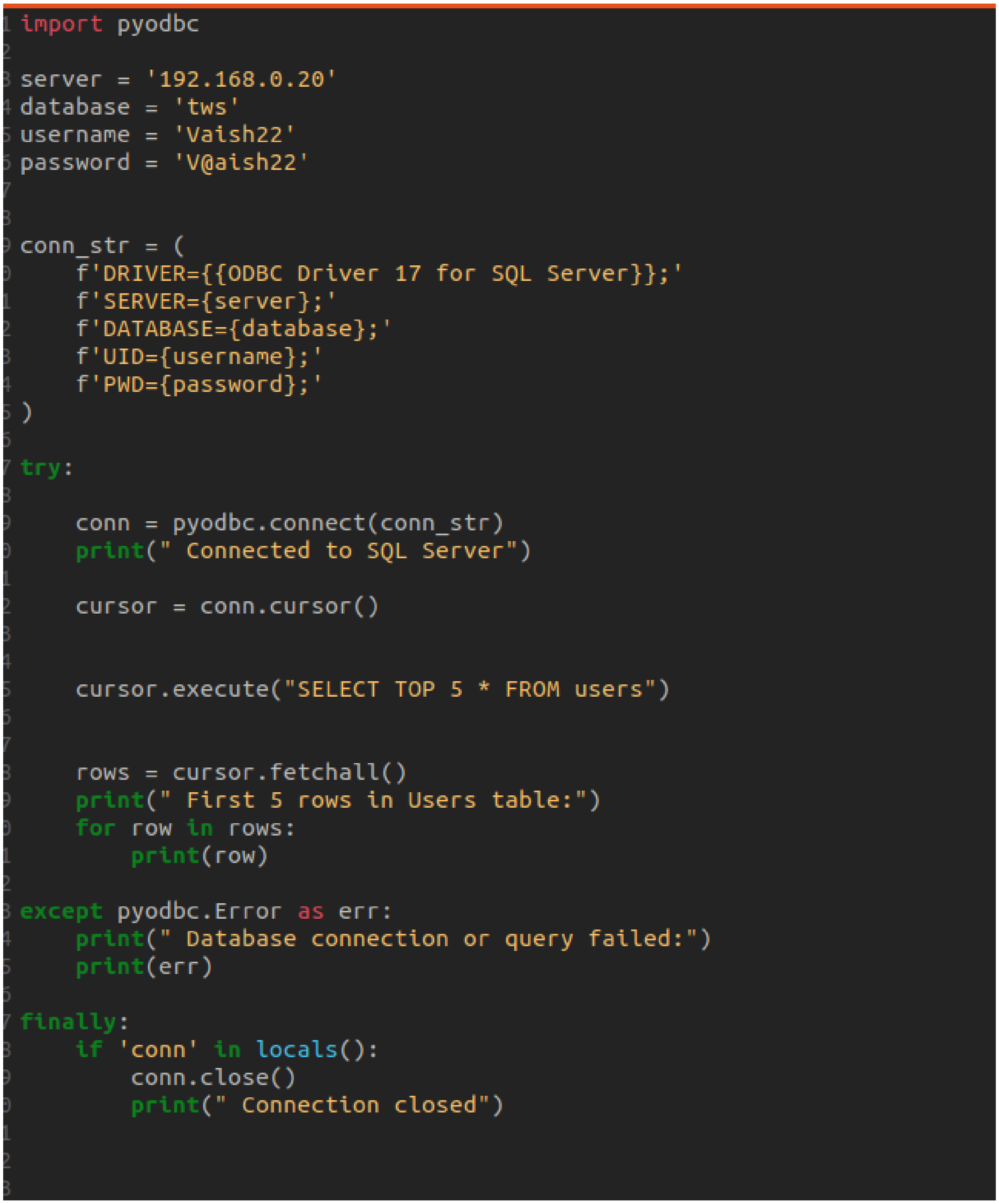
Step 5: <u>Output Screenshot (Verification)</u>

- `Installing Required Package:-`



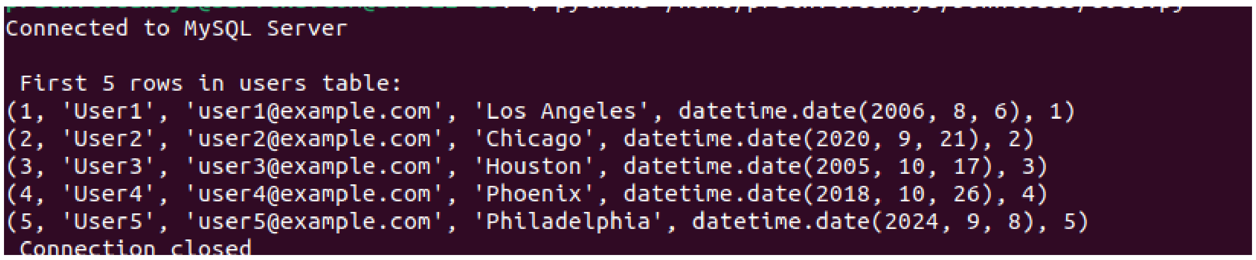
```
pip install pyodbc
```


```
Defaulting to user installation because normal site-packages is not writeable
Collecting pyodbc
  Downloading pyodbc-5.2.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (336 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 336.0/336.0 KB 1.9 MB/s eta 0:00:00
```



```python
import pyodbc

server = '192.168.0.20'
database = 'tws'
username = 'Vaish22'
password = 'V@aish22'


conn_str = (
    f'DRIVER={{ODBC Driver 17 for SQL Server}};'
    f'SERVER={server};'
    f'DATABASE={database};'
    f'UID={username};'
    f'PWD={password};'
)

try:

    conn = pyodbc.connect(conn_str)
    print(" Connected to SQL Server")

    cursor = conn.cursor()


    cursor.execute("SELECT TOP 5 * FROM users")


    rows = cursor.fetchall()
    print(" First 5 rows in Users table:")
    for row in rows:
        print(row)

except pyodbc.Error as err:
    print(" Database connection or query failed:")
    print(err)

finally:
    if 'conn' in locals():
        conn.close()
        print(" Connection closed")
```

<u>**OUTPUT:**</u>



```
Connected to MySQL Server

 First 5 rows in users table:
(1, 'User1', 'user1@example.com', 'Los Angeles', datetime.date(2006, 8, 6), 1)
(2, 'User2', 'user2@example.com', 'Chicago', datetime.date(2020, 9, 21), 2)
(3, 'User3', 'user3@example.com', 'Houston', datetime.date(2005, 10, 17), 3)
(4, 'User4', 'user4@example.com', 'Phoenix', datetime.date(2018, 10, 26), 4)
(5, 'User5', 'user5@example.com', 'Philadelphia', datetime.date(2024, 9, 8), 5)
 Connection closed
```

Step 6: Additional Notes

- The year field in the SQL data is stored as a date string. In a production setting, this might be used for time-based filtering.
- The use of null IDs in some rows simulates real-world dirty data scenarios.

Step 7: Tools & Technologies Used

COMPONENT	DESCRIPTION
SQL Server	Backend database engine
Python 3.X	Programming language
pyodbc	Python-SQL Server connectivity
SQL insert script	Bulk data population
VS Code / IDE	Python development environment

Conclusion

This mini project demonstrates the key steps a Data Engineer often follows when integrating Python with a SQL Server database:

- Ensuring environment setup
- Writing clean, reusable scripts
- Handling exceptions
- Populating realistic data for testing
- Verifying end-to-end connectivity and data access