KGuite
BIDD330
04/21/2024
Module 03 Writeup - Intro to Jupyter Notebook

Overview
This week, we focused on establishing and manipulating databases directly from Jupyter Notebook. We managed databases hosted on a class-linked server, involving data imports, procedural queries, and database exports.

Development Environment Setup
To streamline our data analysis and database management tasks, we utilized Jupyter Notebook for its robust, interactive computing capabilities. Below are the steps and configurations implemented:

Imports:
We started by importing necessary Python libraries to facilitate database connectivity, data manipulation, and visualization:

- pyodbc: For connecting to the SQL Server.
- pandas: Essential for data manipulation and querying in a tabular format.
- SQLAlchemy: Utilized for database connection pooling and providing a higher abstraction for database manipulations.
- matplotlib.pyplot: For plotting graphs directly from the data.
- warnings: To suppress deprecation warnings that could clutter the notebook's output.

Database Connections:
Two methods were used to connect to our SQL Server, ensuring robust access and query capabilities:

- Direct Connection using pyodbc: This traditional method provides detailed control over database operations.
- Enhanced Connection using SQLAlchemy: Offers a more flexible and powerful approach to manage SQL operations and transactions.

Imports

[1]:
```
import pyodbc
import pandas as pd
from sqlalchemy import create_engine, text
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

Connections

Note: Connect via BigIP and SQL Server first

[11]:
```
# Connect using pyodbc
conn = pyodbc.connect('DRIVER={SQL Server};SERVER=uwc-studentsql.continuum.uw.edu\\uwcbiddsql;DATABASE=Crimson_Katharine;[redacted]
```

[13]:
```
# Connect using SQLAlchemy
engine = create_engine('mssql+pyodbc:[redacted]w@uwc-studentsql.continuum.uw.edu\\uwcbiddsql/Crimson_Katharine?driver=SQL+Server')

# Example to test the connection and execute a query
with engine.connect() as connection:
    # Wrap the SQL command in text() for explicit execution
    result = connection.execute(text("SELECT TOP 10 * FROM [dbo].[FactUnemployment]"))
    for row in result:
        print(row)  # This will print the rows from the FactUnemployment table
```

```
(1, 'Alabama', '2020-01-04', 4578, '2019-12-28', 18523, 1923741, 0.96)
(2, 'Alabama', '2020-01-11', 3629, '2020-01-04', 21143, 1923741, 1.1)
(3, 'Alabama', '2020-01-18', 2483, '2020-01-11', 17402, 1923741, 0.9)
(4, 'Alabama', '2020-01-25', 2129, '2020-01-18', 18390, 1923741, 0.96)
(5, 'Alabama', '2020-02-01', 2170, '2020-01-25', 17284, 1923741, 0.9)
(6, 'Alabama', '2020-02-08', 2176, '2020-02-01', 16745, 1923741, 0.87)
(7, 'Alabama', '2020-02-15', 1981, '2020-02-08', 16571, 1923741, 0.86)
(8, 'Alabama', '2020-02-22', 1735, '2020-02-15', 16059, 1923741, 0.83)
(9, 'Alabama', '2020-02-29', 1575, '2020-02-22', 14721, 1923741, 0.77)
(10, 'Alabama', '2020-03-07', 1663, '2020-02-29', 13657, 1923741, 0.71)
```

Query Execution and Dataframe Creation:

Utilizing both connection methods, I executed queries to fetch data from `FactUnemployment` and `FactCovid` tables. This was done using both raw SQL execution via pyodbc and pandas integration with SQLAlchemy.

SQL Queries

SQL Query - FactUnemployment

[20]:
```
cursor = conn.cursor()
cursor.execute("SELECT TOP 10 * FROM [dbo].[FactUnemployment]")
for i in cursor:
    print(i)
```

```
(1, 'Alabama', '2020-01-04', 4578, '2019-12-28', 18523, 1923741, 0.96, 'USA')
(2, 'Alabama', '2020-01-11', 3629, '2020-01-04', 21143, 1923741, 1.1, 'USA')
(3, 'Alabama', '2020-01-18', 2483, '2020-01-11', 17402, 1923741, 0.9, 'USA')
(4, 'Alabama', '2020-01-25', 2129, '2020-01-18', 18390, 1923741, 0.96, 'USA')
(5, 'Alabama', '2020-02-01', 2170, '2020-01-25', 17284, 1923741, 0.9, 'USA')
(6, 'Alabama', '2020-02-08', 2176, '2020-02-01', 16745, 1923741, 0.87, 'USA')
(7, 'Alabama', '2020-02-15', 1981, '2020-02-08', 16571, 1923741, 0.86, 'USA')
(8, 'Alabama', '2020-02-22', 1735, '2020-02-15', 16059, 1923741, 0.83, 'USA')
(9, 'Alabama', '2020-02-29', 1575, '2020-02-22', 14721, 1923741, 0.77, 'USA')
(10, 'Alabama', '2020-03-07', 1663, '2020-02-29', 13657, 1923741, 0.71, 'USA')
```

SQL Query - FactCovid

[21]:
```
cursor = conn.cursor()
cursor.execute("SELECT TOP 10 * FROM [dbo].[FactCovid]")
for i in cursor:
    print(i)
```

```
(1, 77765279, '2020-10-22', 512, 10, 25, 0, 0, 0, '42.17028', '-76.3063', 'US', 'USA', 'United States', 'New York', 'US-NY', 'Tioga County')
(2, 78096938, '2020-10-23', 549, 37, 25, 0, 0, 0, '42.17028', '-76.3063', 'US', 'USA', 'United States', 'New York', 'US-NY', 'Tioga County')
(3, 78440992, '2020-10-24', 572, 23, 25, 0, 0, 0, '42.17028', '-76.3063', 'US', 'USA', 'United States', 'New York', 'US-NY', 'Tioga County')
(4, 78767346, '2020-10-25', 589, 17, 26, 1, 0, 0, '42.17028', '-76.3063', 'US', 'USA', 'United States', 'New York', 'US-NY', 'Tioga County')
(5, 79068611, '2020-10-26', 610, 21, 26, 0, 0, 0, '42.17028', '-76.3063', 'US', 'USA', 'United States', 'New York', 'US-NY', 'Tioga County')
(6, 79419515, '2020-10-27', 629, 19, 27, 1, 0, 0, '42.17028', '-76.3063', 'US', 'USA', 'United States', 'New York', 'US-NY', 'Tioga County')
(7, 79674733, '2020-10-28', 653, 24, 27, 0, 0, 0, '42.17028', '-76.3063', 'US', 'USA', 'United States', 'New York', 'US-NY', 'Tioga County')
(8, 80116653, '2020-10-29', 672, 19, 30, 3, 0, 0, '42.17028', '-76.3063', 'US', 'USA', 'United States', 'New York', 'US-NY', 'Tioga County')
(9, 80467166, '2020-10-30', 689, 17, 32, 2, 0, 0, '42.17028', '-76.3063', 'US', 'USA', 'United States', 'New York', 'US-NY', 'Tioga County')
(10, 80802061, '2020-10-31', 699, 10, 33, 1, 0, 0, '42.17028', '-76.3063', 'US', 'USA', 'United States', 'New York', 'US-NY', 'Tioga County')
```

Next, I ran a Jupyter Magic command- this was required for the exercise.

Jupyter Magic Command Demo

[22]:
```
!pip install pretty
```

```
Requirement already satisfied: pretty in c:\users\test\appdata\local\programs\python\python312\lib\site-packages (0.1)
```

Setting PANDAS dataframes

Create PANDAS dataframe - FactUnemployment

[24]:
```
dataframeUnemployment = pd.read_sql("SELECT U.[State], SUM(U.[Initial Claims]) [ClaimsYTD] FROM [dbo].[FactUnemployment] U GROUP BY U.[State];", conn)
print(dataframeUnemployment)
```

```
                   State  ClaimsYTD
0            Puerto Rico    1398560
1               Illinois   10768238
2               Oklahoma    2968662
3              Wisconsin    4581194
4                  Maine     751146
5           Pennsylvania    9953896
6          Massachusetts    6038466
7                   Ohio   10027234
8                Arizona    3378458
9                 Oregon    3261200
10               Wyoming     322152
11             Louisiana    3400916
12                Hawaii    1458728
13         New Hampshire     842658
14              Virginia    4275742
15              New York   16341314
16          South Dakota     231972
17   Run Date: 3/22/2024          0
18              Maryland    3529852
19            California   41148682
20              Missouri    3687392
21             Minnesota    4101638
22  District of Columbia     717422
```

At the end of this query, I encountered this error:

```
46                  Utah    1154140
47               Alabama    3196876
48             Tennessee    3411958
49          Rhode Island    1348646
50              Arkansas    1527088
51                Nevada    2707192
52                Kansas    1839014
53              Kentucky    4133742
54            New Mexico    1165336
C:\Users\test\AppData\Local\Temp\ipykernel_10868\1269065339.py:1: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or databas
e string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.
  dataframeUnemployment = pd.read_sql("SELECT U.[State], SUM(U.[Initial Claims]) [ClaimsYTD] FROM [dbo].[FactUnemployment] U GROUP BY U.[State];", conn)
```

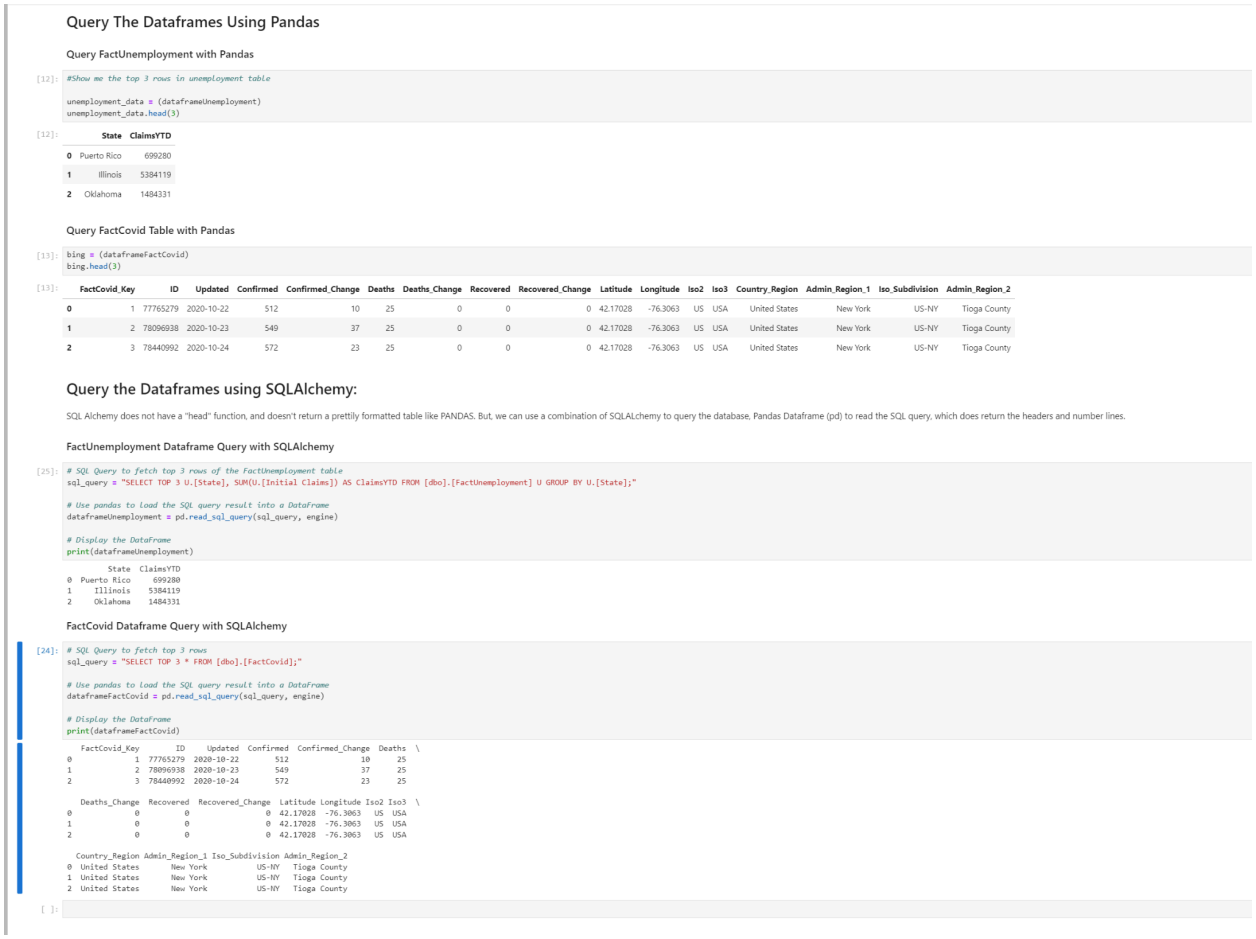
So then I recreated the dataframe using SQLAlchemy:

The code block below uses the SQLAlchemy engine to execute a query and load the result into a DataFrame.

[19]:
```
# Using SQL Alchemy engine to execute a SQL query and load the result into a DataFrame
sql_query = "SELECT U.[State], SUM(U.[Initial Claims]) AS ClaimsYTD FROM [dbo].[FactUnemployment] U GROUP BY U.[State];"
dataframeUnemployment = pd.read_sql(sql_query, engine)
print(dataframeUnemployment)
```

```
                   State  ClaimsYTD
0            Puerto Rico     699280
1               Illinois    5384119
2               Oklahoma    1484331
3              Wisconsin    2290597
4                  Maine     375573
5           Pennsylvania    4976948
6          Massachusetts    3019233
7                   Ohio    5013617
8                Arizona    1689229
9                 Oregon    1630600
10               Wyoming     161076
11             Louisiana    1700458
12                Hawaii     729364
13         New Hampshire     421329
14              Virginia    2137871
15              New York    8170657
16          South Dakota     115986
17   Run Date: 3/22/2024          0
18              Maryland    1764926
19            California   20574341
20              Missouri    1843696
21             Minnesota    2050819
```

Data Visualization:
With the data successfully queried and loaded into pandas DataFrames, I utilized `matplotlib` to create visual representations (not shown in the code snippet but mentioned for completeness).



Query The Dataframes Using Pandas

Query FactUnemployment with Pandas

```
[12]: #Show me the top 3 rows in unemployment table

unemployment_data = (dataframeUnemployment)
unemployment_data.head(3)
```

[12]:

	State	ClaimsYTD
0	Puerto Rico	699280
1	Illinois	5384119
2	Oklahoma	1484331

Query FactCovid Table with Pandas

```
[13]: bing = (dataframeFactCovid)
bing.head(3)
```

[13]:

	FactCovid_Key	ID	Updated	Confirmed	Confirmed_Change	Deaths	Deaths_Change	Recovered	Recovered_Change	Latitude	Longitude	Iso2	Iso3	Country_Region	Admin_Region_1	Iso_Subdivision	Admin_Region_2
0	1	77765279	2020-10-22	512	10	25	0	0	0	42.17028	-76.3063	US	USA	United States	New York	US-NY	Tioga County
1	2	78096938	2020-10-23	549	37	25	0	0	0	42.17028	-76.3063	US	USA	United States	New York	US-NY	Tioga County
2	3	78440992	2020-10-24	572	23	25	0	0	0	42.17028	-76.3063	US	USA	United States	New York	US-NY	Tioga County

Query the Dataframes using SQLAlchemy:

SQL Alchemy does not have a "head" function, and doesn't return a prettily formatted table like PANDAS. But, we can use a combination of SQLALchemy to query the database, Pandas Dataframe (pd) to read the SQL query, which does return the headers and number lines.

FactUnemployment Dataframe Query with SQLAlchemy

```
[25]: # SQL Query to fetch top 3 rows of the FactUnemployment table
sql_query = "SELECT TOP 3 U.[State], SUM(U.[Initial Claims]) AS ClaimsYTD FROM [dbo].[FactUnemployment] U GROUP BY U.[State];"

# Use pandas to load the SQL query result into a DataFrame
dataframeUnemployment = pd.read_sql_query(sql_query, engine)

# Display the DataFrame
print(dataframeUnemployment)
```

```
        State  ClaimsYTD
0  Puerto Rico     699280
1     Illinois    5384119
2     Oklahoma    1484331
```

FactCovid Dataframe Query with SQLAlchemy

```
[24]: # SQL Query to fetch top 3 rows
sql_query = "SELECT TOP 3 * FROM [dbo].[FactCovid];"

# Use pandas to load the SQL query result into a DataFrame
dataframeFactCovid = pd.read_sql_query(sql_query, engine)

# Display the DataFrame
print(dataframeFactCovid)
```

```
   FactCovid_Key        ID     Updated  Confirmed  Confirmed_Change  Deaths  \
0              1  77765279  2020-10-22        512                10      25
1              2  78096938  2020-10-23        549                37      25
2              3  78440992  2020-10-24        572                23      25

   Deaths_Change  Recovered  Recovered_Change  Latitude Longitude Iso2 Iso3  \
0              0          0                 0  42.17028  -76.3063   US  USA
1              0          0                 0  42.17028  -76.3063   US  USA
2              0          0                 0  42.17028  -76.3063   US  USA

  Country_Region Admin_Region_1 Iso_Subdivision Admin_Region_2
0  United States       New York           US-NY   Tioga County
1  United States       New York           US-NY   Tioga County
2  United States       New York           US-NY   Tioga County
```

```
[ ]:
```

Learning and Utility:

- Jupyter Magic Commands: I explored and applied Jupyter-specific commands (`!pip install package_name`) demonstrating the notebook's capability to handle package management internally.
- Practical Application: The integration of SQLAlchemy and pandas significantly streamlined the process of querying, which was evident when handling data retrieval and manipulation tasks. This setup not only facilitated a more profound understanding of these tools but also enhanced the workflow efficiency.

Reflection:
This module enriched my understanding and proficiency with pandas and SQLAlchemy. It highlighted the practical applications and efficiencies these tools bring to data handling and analysis within a Jupyter Notebook environment.