**ETL/ELT (Snowflake and Python) Assignment**

This project is likely to take longer than the time we have to complete the project. I expect that each person devote about 10 hours on the assignment and complete as much as possible of part 2 given this time constraint.

**Part 1 – Introduction to Snowflake (individual, but nothing due)**

1. Sign-up for a [free trial account](https://signup.snowflake.com/) (select Enterprise and AWS US, West Oregon). You should receive an email with a link to activate your account. The email also contains a URL that has your account identifier (in bold), e.g., https://**brkvxvv-wnb73917**.snowflakecomputing.com/console/login. You need the account identifier in the next step. Next activate your account and log in to Snowflake (using your browser). Remember your username and password. After you have logged into Snowflake you do not need to do anything inside Snowflake from your browser.
2. Open up a Python editor and run the following code in Python (run each step individually):

pip install --upgrade snowflake-connector-python

*if you get an error message saying that pyarrow is the wrong version, then run the following command: pip install pyarrow==10.0.1*

import snowflake.connector

conn = snowflake.connector.connect(

user='Your\_UserName',

password='Your\_Password!',

account='Your\_Snowflake\_Account'

)

*Step 1 describes how to get the account information needed above (e.g., brkvxvv-wnb73917 in this example). You can also find your account identifier by accessing Snowflake using your browser, clicking on Admin on the left-hand side of your screen, and then selecting Accounts. If you move your mouse over your account and then click on the exclamation mark, you get the full account string, e.g.,* https://***brkvxvv-wnb73917***.snowflakecomputing.com. *You can alternatively get this information from the URL of your Snowflake web interface, e.g., https://app.snowflake.com/****brkvxvv/wnb73917****/worksheets. The string in the URL may not be sufficient to identify the account if you used a different region than US West or a different cloud provider than AWS (i.e., if you selected Google Cloud Platform or Azure in the set up). For example, if the region is different, then you might be able to identify the account using the following format: xy12345.us-east-1. If Google or Azure was selected as the cloud provider, then the account might follow the following formats: xy12345.us-east-1.gcp or xy12345.west-us-2.azure. If this does not work then try to identify your account identified using the approach described in step 1 above.*

cs = conn.cursor()

*conn is the object that connects you to your Snowflake account.*

*cs is a database cursor object for execute and fetch operations. You will use this object to execute SQL commands with the following format: cs.execute("YOUR SQL COMMAND")*

1. Create Snowflake Objects

First create a virtual warehouse (virtual warehouses contain the compute resources that are required to perform queries and DML operations with Snowflake):

cs.execute("CREATE WAREHOUSE IF NOT EXISTS my\_first\_warehouse")

In addition to creating a warehouse, this command also sets that warehouse as the active warehouse for your session. You can change to a different warehouse (do not run this code at this point) using: cs.execute("USE WAREHOUSE different\_warehouse\_mg")

Next create a database (databases contain your schemas, which contain your database objects). This is done in a similar manner to creating a warehouse:

cs.execute("CREATE DATABASE IF NOT EXISTS testdb")

Creating a database also sets that database as the active one for the current session (you can change this using USE DATABASE…).

Next up are schemas. Schemas are grouping of database objects, including tables, data within them, and views. Schemas are found within databases:

cs.execute("CREATE SCHEMA IF NOT EXISTS testschema")

Next create a table and insert data (this is regular DDL and DML):

cs.execute(

"CREATE OR REPLACE TABLE "

"test\_table(col1 integer, col2 string)")

cs.execute(

"INSERT INTO test\_table(col1, col2) "

"VALUES (123, 'test string1'), (456, 'test string2')")

Just like in Postgres, you can also create a table from a SELECT statement. You can also create views and materialized views (which are more like tables created by select statements). You can also load data from file using PUT and COPY INTO commands (the code below is just an example, i.e., you cannot run it the way it is currently written - to run this you need some data to put into the stage, you also need to have created the stage, and the structure of the data you move from the stage to test\_table has match in ), e.g., cs.execute("PUT file:///tmp/data/file\* @test\_table\_stage") and then cs.execute("COPY INTO test\_table FROM @test\_table\_stage").

The PUT command places the file in a staging area and the COPY INTO command is copying that data from the stage into a specified table (see <https://docs.snowflake.com/en/sql-reference/sql/copy-into-table.html> for more options). You can also use the COPY INTO command to copy data from an external stage.

1. Finally, run a SELECT query and view the result:

cs.execute('SELECT \* FROM test\_table')

print(cs.fetchmany(2))

1. To see the objects you create, go to Snowflake and click Data (on the left-hand side of the screen) 🡪 Databases 🡪 TESTDB 🡪 TESTSCHEMA 🡪 Tables 🡪 TEST\_TABLE. Then look for Data Preview on the right-hand side of your screen. To see the Virtual Warehouse you created, click Admin 🡪 Warehouses
2. You will also likely need to work with [FILE FORMAT](https://docs.snowflake.com/en/sql-reference/sql/create-file-format.html) objects when importing data. In addition to browsing Snowflake’s documentation as needed when working on part 2, the following sections of the documentation are recommended:

[Understanding Snowflake’s architecture (e.g., database vs. virtual warehouses)](https://docs.snowflake.com/en/user-guide/intro-key-concepts.html#snowflake-architecture)

[Loading Data into Snowflake](https://docs.snowflake.com/en/user-guide-data-load.html)

Additional tutorials that may be useful:

[Zero to Snowflake: Creating Your First Database](https://interworks.com/blog/chastie/2019/10/18/zero-to-snowflake-creating-your-first-database/)

[Zero to Snowflake: ETL vs. ELT](https://interworks.com/blog/chastie/2019/11/12/zero-to-snowflake-etl-or-elt/)

[Zero to Snowflake: Staging Explained](https://interworks.com/blog/chastie/2019/12/05/zero-to-snowflake-staging-explained/)

[Zero to Snowflake: Importing Data without Code via the User Interface](https://interworks.com/blog/chastie/2019/12/12/zero-to-snowflake-importing-data-without-code-via-the-user-interface/)

[Zero to Snowflake: Importing Data with Code via SnowSQL](https://interworks.com/blog/chastie/2019/12/20/zero-to-snowflake-importing-data-with-code-via-snowsql/)

[Zero to Snowflake: Python and Snowflake](https://interworks.com/blog/chastie/2020/01/02/zero-to-snowflake-python-and-snowflake/)

[Zero to Snowflake: Role-Based Security Access](https://interworks.com/blog/cmurray/2020/01/10/zero-to-snowflake-role-based-security-access/)

[Zero to Snowflake: Structured Data and Snowflake](https://interworks.com/blog/chastie/2020/01/15/zero-to-snowflake-structured-data-and-snowflake/)

[Zero to Snowflake: An Introduction to Semi-Structured JSON Data Formats](https://interworks.com/blog/chastie/2020/01/21/zero-to-snowflake-an-introduction-to-semi-structured-json-data-formats/)

[Zero to Snowflake: Loading and Querying Semi-Structured JSON Data](https://interworks.com/blog/chastie/2020/01/28/zero-to-snowflake-loading-and-querying-semi-structured-json-data/)

[Zero to Snowflake: Defining Virtual Warehouses](https://interworks.com/blog/2020/02/12/zero-to-snowflake-defining-virtual-warehouses/)

[Zero to Snowflake: Simple SQL Stored Procedures](https://interworks.com/blog/2020/02/18/zero-to-snowflake-simple-sql-stored-procedures/)

[Zero to Snowflake: Multi-Threaded Bulk Loading with Python](https://interworks.com/blog/2020/03/04/zero-to-snowflake-multi-threaded-bulk-loading-with-python/)

[Zero to Snowflake: Tips for Query Building in Snowflake](https://interworks.com/blog/2020/03/11/zero-to-snowflake-tips-for-query-building-in-snowflake/)

**Part 2 – Python and Snowflake Case (group)**

This part of the assignment is a group project. Submit one deliverable per group using the assignment drop box on the course website by the due date listed in the syllabus.

You will use Python (you can use any Python editor) and Snowflake to extract, load, and transform data. You should have all your code written in Python, but as much of the work as possible should be done by Snowflake (i.e., as much of your code as possible should be cs.execute… commands). For example, instead of using Pandas to transform data, do so instead using a SQL statement written in Python that is sent to Snowflake. Note that your deliverable should not require any manual work using a Snowflake GUI.

You will be working with purchase order, supplier, invoice, and weather data that should be loaded into Snowflake. You should then create queries that calculate differences between invoice and purchase order amounts and see if the weather has anything to do with these differences (there is nothing interesting to find and I am not expecting a report about this). The data are stored in the following formats/sources:

* csv (comma delimited) – 41 files with monthly purchase order data (at the line item level)
* XML – one file with supplier invoice
* postgres – one table with supplier information (to prepare the data for the assignment, first download the supplier\_case.psql code from Canvas and run it in the Postgres extension in VS Code or pgweb)
* Snowflake Marketplace – Weather & Environment data from Cybersyn.
* txt (tab delimited) – to work with the weather data you will use weather station latitudes and longitude information in the Cybersyn and merge the station information with a mapping table that allows you to determine a representative weather station for each zip code in the supplier data. The mapping table is a tab delimited txt file that contains zip codes along with latitude and longitude information.

Specific Tasks to Perform (note that these will be graded not only based on the completion of each task, but also the quality of the code, especially in terms of performance assuming the code might need to handle more data (and at a higher frequency). Note that while ChatGTP is often able to come up with solutions (or at least code that can be fixed fairly easily) that get the work done (the solutions are effective), these solutions are not always the most efficient solutions. Try to find snowflake documentation to determine if there might be a better way to write your code.

Before completing the extract, load, and transform tasks below, examine the data and the other requirements. During extract and load, exclude columns that do not appear to be relevant (and that you do not think would be relevant for other problems, e.g., columns with all Null values or columns with only a single value across all rows). Also convert datatypes as needed, e.g., Snowflake is more efficient when dates are stored as dates rather than varchar. You can combine multiple tasks below into single processing steps (or split a single task into multiple processing steps).

1. Extract and load the 41 comma delimited purchases data files and form a single table of purchases data;
   1. Preferably follow these [guidelines](https://docs.snowflake.com/en/user-guide/data-load-considerations-stage.html) when staging the files (this staging approach does not make sense for our data as the files are small, but it is good practice if you have more data and if the data is loaded over time)
   2. Use Python to automate the PUT process, e.g., use glob to iterate through and PUT all purchases files automatically
   3. COPY INTO is generally preferred over INSERT INTO (this applies to the entire project);
   4. To the extent possible, perform transformations such as selecting columns and setting data types during the COPY INTO process
2. Create a calculated field that shows purchase order totals, i.e., for each order, sum the line item amounts (defined as ReceivedOuters \* ExpectedUnitPricePerOuter), and name this field POAmount
3. Extract and load the supplier invoice XML data
   1. shred the data into a table (preferably in the COPY INTO process) where each row corresponds to a single invoice
4. Join the purchases data from step 2 and the supplier invoices data from step 3 (only include matching rows); assuming that step 2 was completed correctly, you can assume the following relationships among the four tables (the other two tables are discussed below):

Daily Weather

Supplier

Invoice

Purchase Order Header

1, 1 0, 1 0, M 1, 1 0, M 0, M

1. Using the joined data from step 4, create a calculated field that shows the difference between AmountExcludingTax and POAmount, name this field invoiced\_vs\_quoted, and save the result as a materialized view named purchase\_orders\_and\_invoices
   1. If your version of Snowflake does not support materialized views then create a table instead using the join (this applies to all requirements about materialized views)
2. Extract the supplier\_case data from postgres (if you did not earlier set up the supplier\_case table in postgres then make sure to download the supplier\_case.psql code from Canvas and run it manually in the Postgres extension in VS Code or pgweb; this is not part of your deliverable, it is simply a step to create data to work with). Do not import the data into Python, instead use Python to move the data from postgres to your local drive and then directly into a Snowflake stage:
   1. You need to use psycopg2 or a similar Python library to connect to the postgres database within Python, issue a command to postgres to have postgres save the supplier\_case data to file, and then use cs.execute to move the file to an internal Snowflake stage and eventually into a table.
   2. Consider creating a Python function that can take a csv file path as input and then generate field definitions (field names and datatypes based on the header and data types in the file) that can then be used in CREATE TABLE statement.
3. Connect manually (using the Snowflake web browser interface) to NOAA data using Marketplace (Data Products 🡪 Marketplace). From inside Snowflake Marketplace (from the home screen click Data Products) search for NOAA and click Weather & Environment from Snowflake Public Data Products. Then click Get and then Get (keep all the defaults). The name of the datasets that you will be using are: WEATHER\_\_ENVIRONMENT.CYBERSYN.NOAA\_WEATHER\_METRICS\_TIMESERIES and WEATHER\_\_ENVIRONMENT.CYBERSYN.NOAA\_WEATHER\_STATION\_INDEX (NOAA\_WEATHER\_METRICS\_ATTRIBUTES additionally contains data definitions that might be helpful):
   1. The goal is to next extract weather data for each unique zip code in the supplier\_case table (suppliers can have the same zip code but you only need to extract weather data for each zip code once). However, while the weather station data contain zip codes, we will pretend that this table does not have this information and instead use latitude and longitude information to determine which weather station to use for each zip code. The approach used in <https://medium.com/data-science/noaa-weather-data-in-snowflake-free-20e90ee916ed> can be helpful (note that this is based on a different data set, but the idea of using latitude and longitude is the same) when finding weather stations closest to each zip code (only use one weather station per zip code). For this to work you need to find a data file with zip code – geo location mappings, e.g., from the US census (the data zip folder on Canvas contains a ZCTA file with this information; in this file GEOID is the five digit ZIP Code, INTPTLAT is Latitude, and INTPTLONG is Longitude);
   2. Create a materialized view named supplier\_zip\_code\_weather that contains the unique zip codes (PostalPostalCode) from the supplier data, date, and daily high temperatures, i.e., the view should have three columns (zip code, date, and high temperature) and one row per day and unique supplier zip code
4. Join purchase\_orders\_and\_invoices, supplier\_case, and supplier\_zip\_code\_weather based on zip codes and the transaction date. Only include transactions that have matching temperature readings

**Deliverable**

Please submit one deliverable per group for part 2 of the case (part 1 does not have a deliverable). The deliverable should include all your Python code (including SQL code embedded inside Python) and any SQL code that you have created inside Snowflake (but such code should preferably be inside Python and passed into Snowflake from Python). Also include screenshots of any manual work that you have done inside Snowflake using the GUI, e.g., if you used the GUI to create databases, schemas, etc. (this should preferably be done inside Python and passed into Snowflake from Python), SQL code that you created inside Snowflake, etc. Create enough comments so that we can understand your code, e.g., indicate which task in the assignment the code. If you have done any other work towards the assignment that is not captured in your Python code or screenshots, and you are not sure how to submit then please reach out to us. If all your code is created in Python (I do not think it is possible to set up the Marketplace subscription in Python, but apart from this all work is possible in Python) and the Python code is well-commented, then your submission is just your python file.