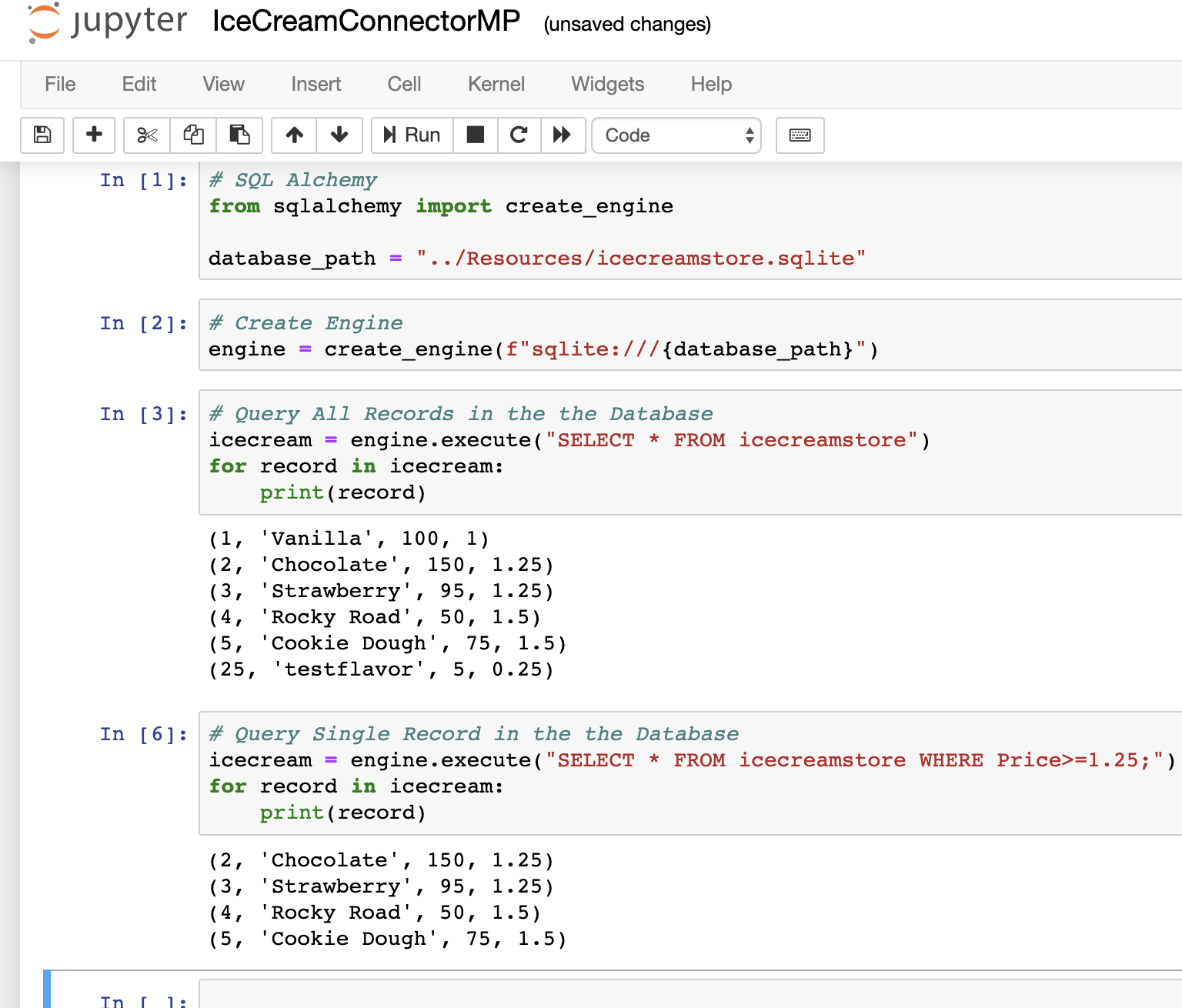
In the first activity students were asked to get in groups and describe an ORM. ORM stands for Object-Relational Mapping and is beneficial for interacting with a database and not having to write with 2 languages of codes, such as 2 different kinds of SQL. SQLAlchemy allows Python to talk to a database. The main drawback is the initial setup can be complex.

In order to operate the code I had to run 2 codes in my terminal:

* conda install -c anaconda sqlalchemy
* conda install -c anaconda sqlite

The first activity IceCreamConnector was pretty straight forward. We were asked to first use the create\_engine module in sqlalchemy to get records in a table. The cool thing is we can write SQL code inside a Python code to pull information we want. One of the students in the solution overview spoke about using data.keys() to display the header role before showing the data in case we wanted to include specific rows.



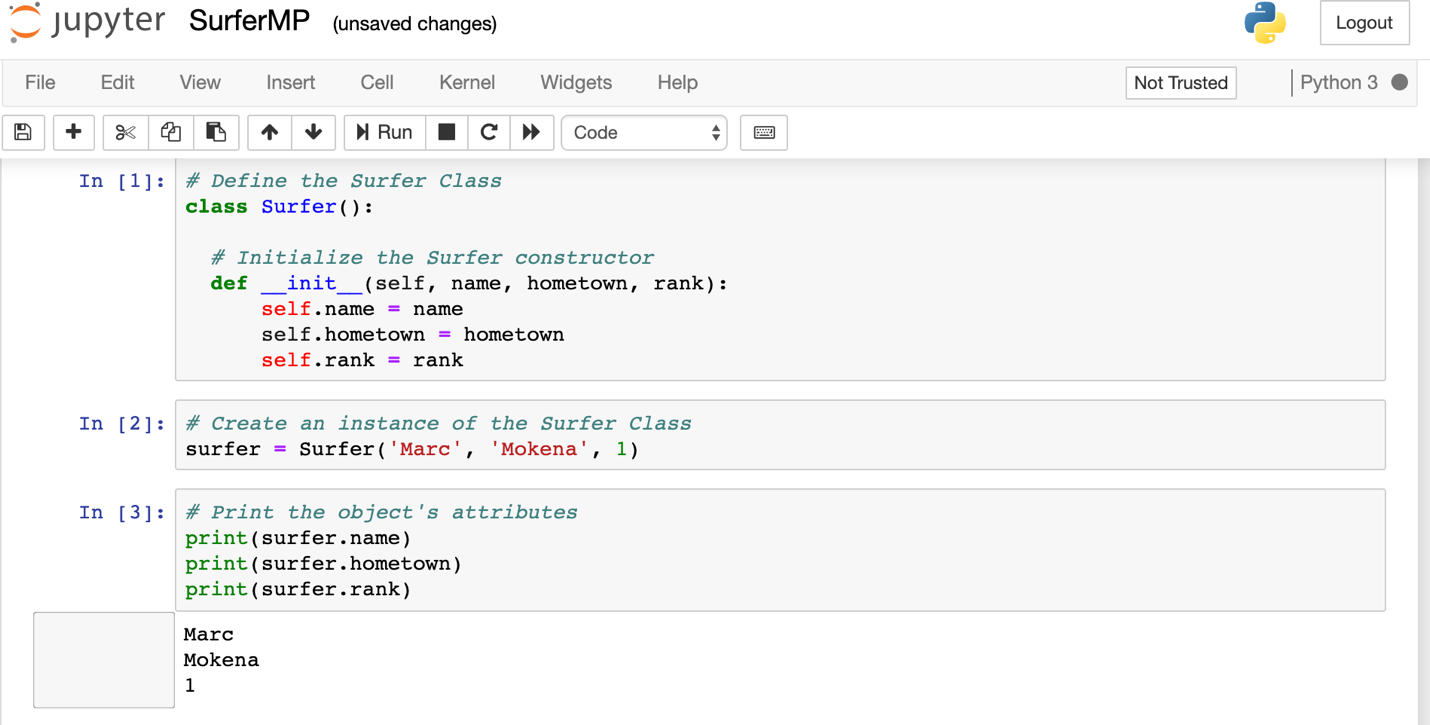
The next student exercise Read\_All\_The­\_SQLs is incredibly helpful because you can pull a bunch of SQL tables to then manipulate the data back in Python using Pandas. This example is helpful because manipulating the data in Pandas is more familiar to us right now and we’re able to merge tables like we already know how. I restarted the kernel to display more code instead of just showing the .head() results. I also had to go back and review the merging in order to get the documentation correct. When going over the answers for this activity, Daisy recommended a helpful bit of troubleshooting code to see if our merge is correct:

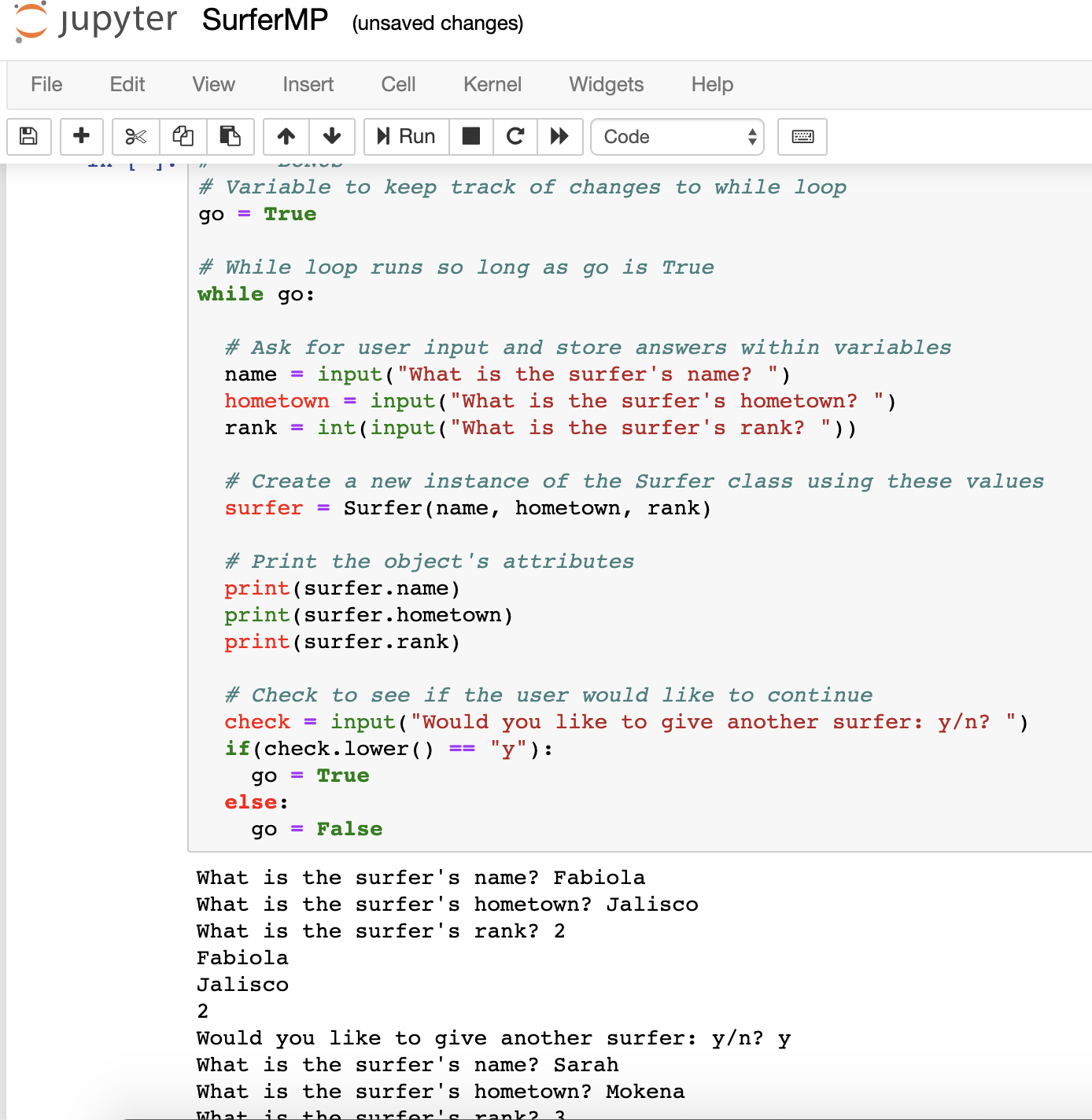
[c for c in census\_data["CityState"].unique() if c in zip\_data["CityState"].unique()]

This will show the entries in a duplicated column that are the same in both databases. In this particular example only 1 City/State was in both databases.

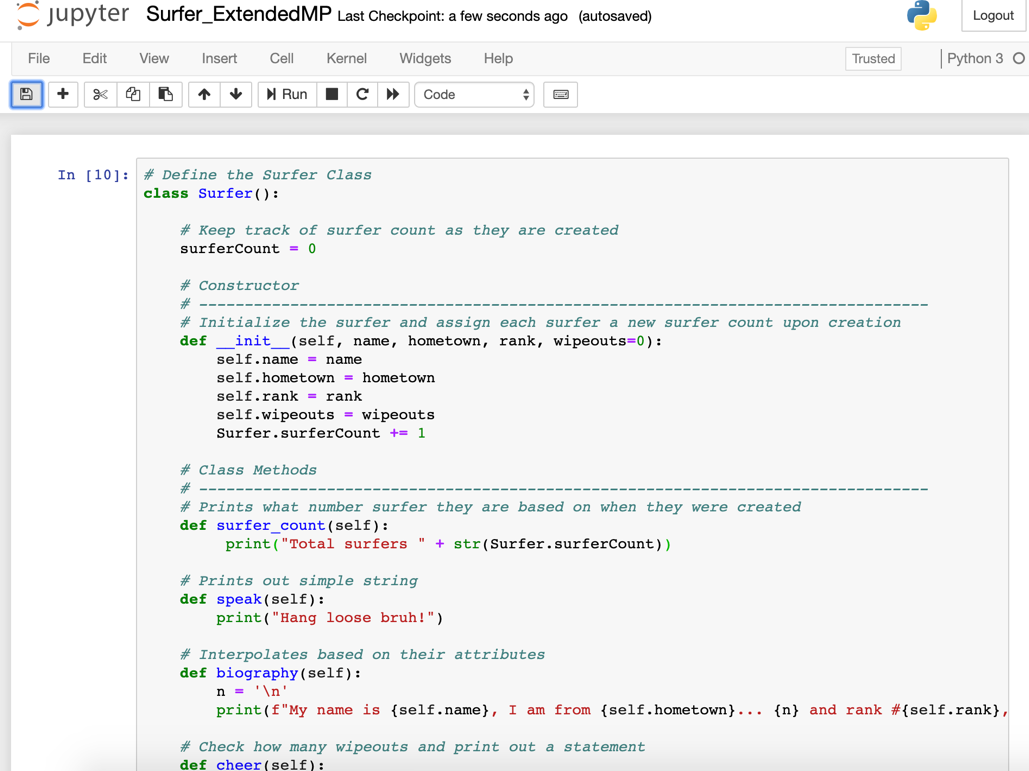


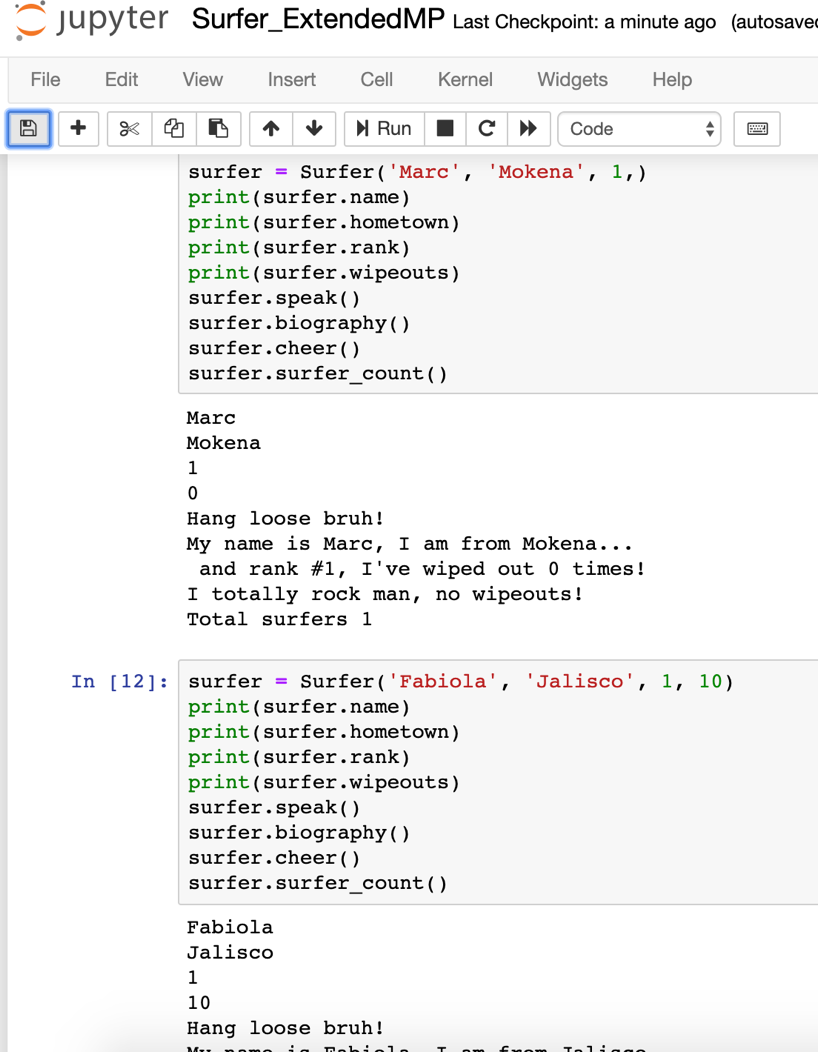
The next student activity, Surfer, allowed us to create a specific type of class, populate the data according to the given structure and call the data. In the bonus, we used the input() function to create questions and used ‘while’ to loop more data input. I needed a little help reviewing the bonus since the while variable = true didn’t come to me right away. The input() part was easy enough.





The Surfer Extended activity gets a little more complex by building in a class with multiple components that we will use to get multiple print statements. I had to dig into this activity a little more and got stuck because I wanted to manipulate my print statement to use a linebreak and I didn’t know how to incorporate this with an f-string print statement. I was able to figure it out by assigning ‘\n’ to a variable to insert into the statement.





The final activity, Stu\_Surfer\_SQL, builds on every component we covered so far in class and allows us to put our specific instances in a database that is provided in the helper code. Then, we need to use SQLAlchemy to pull data that already exists in a database and use the session.add and session.commit functions to update the database. This activity was a lot easier because we built onto this skill in the next class, which I was present.

