Built-in read/write functions

In [208…
```
# create our text file to read
with open('textfile.txt', mode='w') as f:
    f.writelines(['This is a text file.\n', 'Now you can read it!'])
```

In [209…
```
# open a file and read the contents
file = open(file='textfile.txt', mode='r')
text = file.readlines()
text  # remember if our last line is a variable, jupyter notebook will print
```

Out[209… `['This is a text file.\n', 'Now you can read it!']`

In [210…
```
# this is the file object
file
```

Out[210… `<_io.TextIOWrapper name='textfile.txt' mode='r' encoding='UTF-8'>`

In [211…
```
# the file is 'open' until we close it.
file.close()
```

In [212…
```
# the 'with' context will automatically close the file once we are out of th
with open(file='textfile.txt', mode='r') as f:
    text = f.readlines()

text
```

Out[212… `['This is a text file.\n', 'Now you can read it!']`

In [213…
```
# the read function reads the entire file at once
with open(file='textfile.txt', mode='r') as f:
    text = f.read()

print(text)
```

```
This is a text file.
Now you can read it!
```

In [214…
```
# remember we can take a subset of strings like this
text[:10]
```

Out[214… `'This is a '`

In [215…
```
# writing to a file
# open the file in a text editor to see the results, or read it with Python
with open(file='writetest.txt', mode='w') as f:
    f.write('testing writing out')
```

In [216…
```
# writing a list of text to a file
# open the file in a text editor or read it through Python to see the result
text_lines = ['This is text for testing writing.', 'Now you can write to a f
```

```
with open(file='writetest2.txt', mode='w') as f:
    f.writelines(text_lines)
```

JSON

In [217…
```
import json
data_dictionary = {'books': 12, 'articles': 100, 'subjects': ['math', 'progr
```

In [218…
```
json_string = json.dumps(data_dictionary)
json_string
```

Out[218…
```
'{"books": 12, "articles": 100, "subjects": ["math", "programming", "data s
cience"]}'
```

In [219…
```
data_dict = json.loads(json_string)
data_dict
```

Out[219…
```
{'books': 12,
 'articles': 100,
 'subjects': ['math', 'programming', 'data science']}
```

In [220…
```
with open('reading.json', 'w') as f:
    json.dump(data_dictionary, f)
```

In [221…
```
with open('reading.json') as f:
    loaded_data = json.load(f)
```

In [222…
```
loaded_data
```

Out[222…
```
{'books': 12,
 'articles': 100,
 'subjects': ['math', 'programming', 'data science']}
```

Credentials in a .py file

In [223…
```
import credentials as creds
print(f'username: {creds.username}\npassword: {creds.password}')
```

```
username: datasci
password: iscool
```

The pickle library

pickle can be used for saving and loading raw Python objects.

In [224…
```
import pickle as pk

data_dictionary = {'books': 12, 'articles': 100, 'subjects': ['math', 'progr
```

```
with open('readings.pk', 'wb') as f:
    pk.dump(data_dictionary, f)
```

In [225…
```
with open('readings.pk', 'rb') as f:
    data = pk.load(f)

print(data)
```

```
{'books': 12, 'articles': 100, 'subjects': ['math', 'programming', 'data scie
nce']}
```

The joblib library

We can also save and load data with joblib. First be sure to install it with `conda install -c conda-forge joblib -y` if you don't already have it installed. Joblib has extra features beyond pickle, such as compression, automatic opening and closing of files, and features to make saving/loading specific data (numpy arrays) faster. Pickle is often faster than joblib, except in special situations (which is why we didn't cover it in the book). This extra section is for your extra knowledge.

In [226…
```
import joblib

joblib.dump(value=data_dictionary, filename='readings.job', compress=True)
```

Out[226…
```
['readings.job']
```

In [227…
```
data = joblib.load(filename='readings.job')
data
```

Out[227…
```
{'books': 12,
 'articles': 100,
 'subjects': ['math', 'programming', 'data science']}
```

We can time how long something takes with the magic command `%%timeit` in Jupyter Notebooks. Note that pickle is faster for saving and reading this dictionary.

In [228…
```
%%timeit
joblib.dump(value=data_dictionary, filename='readings.job', compress=True)
```

```
148 µs ± 10 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

In [229…
```
%%timeit
with open('readings.pk', 'wb') as f:
    pk.dump(data_dictionary, f)
```

```
51.7 µs ± 5.97 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

In [230…
```
%%timeit
data = joblib.load(filename='readings.job')
```

```
57.8 µs ± 4 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

In [231…
```
%%timeit
with open('readings.pk', 'rb') as f:
    data = pk.load(f)
```

```
10.3 µs ± 859 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

We can see that for normal dictionaries, pickle is much faster for saving and loading than joblib.

sqlite3

When we install Python, we also install SQLite3. We can use it from within Python, but also from the command line. If we open a terminal and type `sqlite3`, it takes us to the SQLite shell. From the shell, we can run any SQLite command. For example, if we are in the directory with the chinook.db file from this GitHub repo (within the same folder that contains this notebook), we can type `.open chinook.db` to load the database. Then we can see the tables within the database with `.tables`. We can also connect to the database through Python and run commands as show below.

In [232…
```
import sqlite3
connection = sqlite3.connect('chinook.db')
cursor = connection.cursor()
```

In [233…
```
# list out tables -- .table does not work from Python sqlite3
cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
cursor.fetchall()
```

Out[233…
```
[('albums',),
 ('sqlite_sequence',),
 ('artists',),
 ('customers',),
 ('employees',),
 ('genres',),
 ('invoices',),
 ('invoice_items',),
 ('media_types',),
 ('playlists',),
 ('playlist_track',),
 ('tracks',),
 ('sqlite_stat1',)]
```

In [234…
```
# get table information (column names, types, settings)
cursor.execute('PRAGMA table_info(artists);')
cursor.fetchall()
```

Out[234…
```
[(0, 'ArtistId', 'INTEGER', 1, None, 1),
 (1, 'Name', 'NVARCHAR(120)', 0, None, 0)]
```

In [235…
```
# SELECT the first 5 rows of artists
cursor.execute('SELECT * FROM artists LIMIT 5;')
cursor.fetchall()
```

Out[235…
```
[(1, 'AC/DC'),
 (2, 'Accept'),
 (3, 'Aerosmith'),
 (4, 'Alanis Morissette'),
 (5, 'Alice In Chains')]
```

In [236…
```
# especially for longer queries, it helps to format them like this, with eac
query = """
SELECT *
FROM artists
LIMIT 5;
"""
cursor.execute(query)
cursor.fetchall()
```

Out[236…
```
[(1, 'AC/DC'),
 (2, 'Accept'),
 (3, 'Aerosmith'),
 (4, 'Alanis Morissette'),
 (5, 'Alice In Chains')]
```

In [237…
```
# get table information (column names, types, settings)
cursor.execute('PRAGMA table_info(invoices);')
cursor.fetchall()
```

Out[237…
```
[(0, 'InvoiceId', 'INTEGER', 1, None, 1),
 (1, 'CustomerId', 'INTEGER', 1, None, 0),
 (2, 'InvoiceDate', 'DATETIME', 1, None, 0),
 (3, 'BillingAddress', 'NVARCHAR(70)', 0, None, 0),
 (4, 'BillingCity', 'NVARCHAR(40)', 0, None, 0),
 (5, 'BillingState', 'NVARCHAR(40)', 0, None, 0),
 (6, 'BillingCountry', 'NVARCHAR(40)', 0, None, 0),
 (7, 'BillingPostalCode', 'NVARCHAR(10)', 0, None, 0),
 (8, 'Total', 'NUMERIC(10,2)', 1, None, 0)]
```

In [238…
```
# save table column names in a list
cursor.execute('PRAGMA table_info(invoices);')
results = cursor.fetchall()
column_names = [r[1] for r in results]
```

In [239…
```
column_names
```

Out[239…
```
['InvoiceId',
 'CustomerId',
 'InvoiceDate',
 'BillingAddress',
 'BillingCity',
 'BillingState',
 'BillingCountry',
 'BillingPostalCode',
 'Total']
```

In [240…
```
cursor.execute('SELECT * FROM invoices LIMIT 5;')
cursor.fetchall()
```

Out[240…
```
[(1,
  2,
  '2009-01-01 00:00:00',
  'Theodor-Heuss-Straße 34',
  'Stuttgart',
  None,
  'Germany',
  '70174',
  1.98),
 (2,
  4,
  '2009-01-02 00:00:00',
  'Ullevålsveien 14',
  'Oslo',
  None,
  'Norway',
  '0171',
  3.96),
 (3,
  8,
  '2009-01-03 00:00:00',
  'Grétrystraat 63',
  'Brussels',
  None,
  'Belgium',
  '1000',
  5.94),
 (4,
  14,
  '2009-01-06 00:00:00',
  '8210 111 ST NW',
  'Edmonton',
  'AB',
  'Canada',
  'T6G 2C7',
  8.91),
 (5,
  23,
  '2009-01-11 00:00:00',
  '69 Salem Street',
  'Boston',
  'MA',
  'USA',
  '2113',
  13.86)]
```

In [241…
```
# ORDER BY
cursor.execute('SELECT Total, InvoiceDate from invoices ORDER BY Total DESC
cursor.fetchall()
```

Out[241…
```
[(25.86, '2013-11-13 00:00:00'),
 (23.86, '2012-08-05 00:00:00'),
 (21.86, '2010-02-18 00:00:00'),
 (21.86, '2011-04-28 00:00:00'),
 (18.86, '2010-01-18 00:00:00')]
```

In [242…
```
# WHERE statement
cursor.execute('SELECT Total, BillingCountry from invoices WHERE BillingCoun
cursor.fetchall()
```

Out[242…
```
[(8.91, 'Canada'),
 (8.91, 'Canada'),
 (0.99, 'Canada'),
 (1.98, 'Canada'),
 (13.86, 'Canada')]
```

In [243…
```
# WHERE using an inserted argument
cursor.execute('SELECT Total, BillingCountry from invoices WHERE BillingCoun
cursor.fetchall()
```

Out[243…
```
[(8.91, 'Canada'),
 (8.91, 'Canada'),
 (0.99, 'Canada'),
 (1.98, 'Canada'),
 (13.86, 'Canada')]
```

In [244…
```
# LIKE command
cursor.execute('SELECT Total, BillingCountry from invoices WHERE BillingCoun
cursor.fetchall()
```

Out[244…
```
[(8.91, 'Canada'),
 (8.91, 'Canada'),
 (0.99, 'Canada'),
 (1.98, 'Canada'),
 (13.86, 'Canada')]
```

In [245…
```
# GROUP BY statement
cursor.execute('SELECT SUM(Total), BillingCountry from invoices GROUP BY Bil
cursor.fetchall()
```

Out[245…
```
[(523.0600000000003, 'USA'),
 (303.9599999999999, 'Canada'),
 (195.09999999999994, 'France'),
 (190.09999999999997, 'Brazil'),
 (156.48, 'Germany')]
```

In [246…
```
# examine column names for invoice_items table
cursor.execute('PRAGMA table_info(invoice_items);')
cursor.fetchall()
```

Out[246…
```
[(0, 'InvoiceLineId', 'INTEGER', 1, None, 1),
 (1, 'InvoiceId', 'INTEGER', 1, None, 0),
 (2, 'TrackId', 'INTEGER', 1, None, 0),
 (3, 'UnitPrice', 'NUMERIC(10,2)', 1, None, 0),
 (4, 'Quantity', 'INTEGER', 1, None, 0)]
```

In [247…
```
# examine a sample of the data
cursor.execute('SELECT * FROM invoice_items LIMIT 5;')
cursor.fetchall()
```

Out[247…
```
[(1, 1, 2, 0.99, 1),
 (2, 1, 4, 0.99, 1),
 (3, 2, 6, 0.99, 1),
 (4, 2, 8, 0.99, 1),
 (5, 2, 10, 0.99, 1)]
```

In [248…
```
# aliases can be used to rename columns and tables
# according to some SQL style guides, it's not best practice to alias a tabl
cursor.execute('SELECT i.TrackID as tid, i.UnitPrice as up FROM invoice_item
cursor.fetchall()
```

Out[248…
```
[(2, 0.99), (4, 0.99), (6, 0.99), (8, 0.99), (10, 0.99)]
```

In [249…
```
# DISTINCT
cursor.execute('SELECT DISTINCT UnitPrice FROM invoice_items;')
cursor.fetchall()
```

Out[249…
```
[(0.99,), (1.99,)]
```

In [250…
```
# JOIN
# get tracks that were purchased and combine with the country
query = """
SELECT invoices.BillingCountry, invoice_items.TrackId
FROM invoices
JOIN invoice_items
ON invoices.InvoiceId = invoice_items.InvoiceId
LIMIT 5;
"""
cursor.execute(query)
cursor.fetchall()
```

Out[250…
```
[('Germany', 2), ('Germany', 4), ('Norway', 6), ('Norway', 8), ('Norway', 1
0)]
```

In [251…
```
# get number of purchased tracks for each track by country, sorted by the to
query = """
SELECT invoice_items.TrackId, COUNT(invoice_items.TrackId), invoices.Billing
FROM invoices
JOIN invoice_items
ON invoices.InvoiceId = invoice_items.InvoiceId
GROUP BY invoices.BillingCountry
ORDER BY COUNT(invoice_items.TrackId) DESC
LIMIT 5;
"""
cursor.execute(query)
cursor.fetchall()
```

Out[251…
```
[(99, 494, 'USA'),
 (42, 304, 'Canada'),
 (234, 190, 'France'),
 (738, 190, 'Brazil'),
 (2, 152, 'Germany')]
```

In [252…
```
# multiple JOINs
query = """
```

```
SELECT tracks.Name, COUNT(invoice_items.TrackId), invoices.BillingCountry
FROM invoices
JOIN invoice_items
ON invoices.InvoiceId = invoice_items.InvoiceId
JOIN tracks
ON tracks.TrackId = invoice_items.TrackId
GROUP BY invoices.BillingCountry
ORDER BY COUNT(invoice_items.TrackId) DESC
LIMIT 5;
"""
cursor.execute(query)
cursor.fetchall()
```

Out[252…
```
[('Your Time Has Come', 494, 'USA'),
 ('Right Through You', 304, 'Canada'),
 ('Morena De Angola', 190, 'France'),
 ('Admirável Gado Novo', 190, 'Brazil'),
 ('Balls to the Wall', 152, 'Germany')]
```

In [253…
```
# this same command as above can also be done with a subquery like this, but
query = """
SELECT tracks.Name, invoice_merged.track_count, invoice_merged.BillingCountr
FROM
    (SELECT ii.TrackId, COUNT(ii.TrackId) as track_count, i.BillingCountry
    FROM invoices as i
    JOIN invoice_items as ii
    ON i.InvoiceId = ii.InvoiceId
    GROUP BY BillingCountry) as invoice_merged
JOIN tracks
ON tracks.TrackId = invoice_merged.TrackId
ORDER BY track_count DESC
LIMIT 5;
"""
cursor.execute(query)
cursor.fetchall()
```

Out[253…
```
[('Your Time Has Come', 494, 'USA'),
 ('Right Through You', 304, 'Canada'),
 ('Admirável Gado Novo', 190, 'Brazil'),
 ('Morena De Angola', 190, 'France'),
 ('Balls to the Wall', 152, 'Germany')]
```

In [254…
```
# be sure to close the connection when done
connection.close()
```

Storing data in a sqlite3 database

In [255…
```
# hypothetical book sales data
book_data = [('12-1-2020', 'Practical Data Science With Python', 19.99, 1),
       ('12-15-2020', 'Python Machine Learning', 27.99, 1),
       ('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1)]
```

In [256…
```
# CREATE and INSERT
connection = sqlite3.connect('book_sales.db')
```

```
cursor = connection.cursor()
```

In [257…

```
# Create table
cursor.execute('''CREATE TABLE IF NOT EXISTS book_sales
             (date text, book_title text, price real, quantity real)''')
```

Out[257…

```
<sqlite3.Cursor at 0x7fceb20c5740>
```

In [258…

```
# the table is now there
cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
cursor.fetchall()
```

Out[258…

```
[('book_sales',)]
```

In [259…

```
# Insert a row of data
cursor.execute("INSERT INTO book_sales VALUES (?, ?, ?, ?)", book_data[0])
```

Out[259…

```
<sqlite3.Cursor at 0x7fceb20c5740>
```

In [260…

```
cursor.execute('SELECT * FROM book_sales ;')
cursor.fetchall()
```

Out[260…

```
[('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0),
 ('12-15-2020', 'Python Machine Learning', 27.99, 1.0),
 ('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0),
 (None, 'machine learining', 10.99, None),
 ('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0),
 ('12-15-2020', 'Python Machine Learning', 27.99, 1.0),
 ('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0),
 ('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0),
 ('12-15-2020', 'Python Machine Learning', 27.99, 1.0),
 ('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0),
 ('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0),
 ('12-15-2020', 'Python Machine Learning', 27.99, 1.0),
 ('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0),
 ('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0)]
```

In [261…

```
# Save the changes with .commit()
# Without this line, the inserted data will not be saved in the database aft
connection.commit()
```

In [262…

```
# insert several records at a time
cursor.executemany('INSERT INTO book_sales VALUES (?, ?, ?, ?)', book_data[1
# don't forget to save the changes
connection.commit()
```

In [263…

```
cursor.execute('SELECT * FROM book_sales;')
cursor.fetchall()
```

Out[263…
```
[('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0),
 ('12-15-2020', 'Python Machine Learning', 27.99, 1.0),
 ('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0),
 (None, 'machine learining', 10.99, None),
 ('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0),
 ('12-15-2020', 'Python Machine Learning', 27.99, 1.0),
 ('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0),
 ('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0),
 ('12-15-2020', 'Python Machine Learning', 27.99, 1.0),
 ('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0),
 ('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0),
 ('12-15-2020', 'Python Machine Learning', 27.99, 1.0),
 ('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0),
 ('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0),
 ('12-15-2020', 'Python Machine Learning', 27.99, 1.0),
 ('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0)]
```

In [264…
```
connection.close()
```

SQLAlchemy

In [265…
```
from sqlalchemy import create_engine
engine = create_engine('sqlite:///book_sales.db')
connection = engine.connect()
```

In [266…
```
from sqlalchemy import text
result = connection.execute(text("select * from book_sales"))
result
```

Out[266…
```
<sqlalchemy.engine.cursor.CursorResult at 0x7fceb1e257f0>
```

In [267…
```
list(result)
```

Out[267…
```
[('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0),
 ('12-15-2020', 'Python Machine Learning', 27.99, 1.0),
 ('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0),
 (None, 'machine learining', 10.99, None),
 ('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0),
 ('12-15-2020', 'Python Machine Learning', 27.99, 1.0),
 ('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0),
 ('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0),
 ('12-15-2020', 'Python Machine Learning', 27.99, 1.0),
 ('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0),
 ('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0),
 ('12-15-2020', 'Python Machine Learning', 27.99, 1.0),
 ('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0),
 ('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0),
 ('12-15-2020', 'Python Machine Learning', 27.99, 1.0),
 ('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0)]
```

In [268…
```
for row in result:
    print(row['date'])
```

In [269…
```
result = connection.execute(text("select * from book_sales"))
for row in result:
    print(row)
```

```
('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0)
('12-15-2020', 'Python Machine Learning', 27.99, 1.0)
('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0)
(None, 'machine learining', 10.99, None)
('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0)
('12-15-2020', 'Python Machine Learning', 27.99, 1.0)
('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0)
('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0)
('12-15-2020', 'Python Machine Learning', 27.99, 1.0)
('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0)
('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0)
('12-15-2020', 'Python Machine Learning', 27.99, 1.0)
('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0)
('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0)
('12-15-2020', 'Python Machine Learning', 27.99, 1.0)
('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0)
```

In [270…
```
# be sure to close the connection when finished
connection.close()
```

In [271…
```
# we can also use the with clause to automatically close the connection
with engine.connect() as connection:
    result = connection.execute(text("select * from book_sales"))
    for row in result:
        print(row)
```

```
('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0)
('12-15-2020', 'Python Machine Learning', 27.99, 1.0)
('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0)
(None, 'machine learining', 10.99, None)
('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0)
('12-15-2020', 'Python Machine Learning', 27.99, 1.0)
('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0)
('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0)
('12-15-2020', 'Python Machine Learning', 27.99, 1.0)
('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0)
('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0)
('12-15-2020', 'Python Machine Learning', 27.99, 1.0)
('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0)
('12-1-2020', 'Practical Data Science With Python', 19.99, 1.0)
('12-15-2020', 'Python Machine Learning', 27.99, 1.0)
('12-17-2020', 'Machine Learning For Algorithmic Trading', 34.99, 1.0)
```

In [272…
```
connection.closed
```

Out[272… True

In [274…
```
# the connection is closed from the 'with' statement, so we can't use it
# notice in the middle and at the bottom of the error, it says 'This Connect
try:
    result = connection.execute(text("select * from book_sales"))
```

```
except Exception as e:
    print(e)
```

```
This Connection is closed
```

In [275…
```
from sqlalchemy import MetaData, Table

metadata = MetaData(engine)
book_sales = Table('book_sales', metadata, autoload=True)
conn = engine.connect()
```

```
---------------------------------------------------------------------------
ArgumentError                             Traceback (most recent call last)
Cell In[275], line 3
      1 from sqlalchemy import MetaData, Table
----> 3 metadata = MetaData(engine)
      4 book_sales = Table('book_sales', metadata, autoload=True)
      5 conn = engine.connect()

File ~/CS-620/.venv/lib64/python3.13/site-packages/sqlalchemy/sql/schema.py:5
537, in MetaData.__init__(self, schema, quote_schema, naming_convention, inf
o)
   5422 """Create a new MetaData object.
   5423
   5424 :param schema:
   (...)   5534
   5535 """
   5536 if schema is not None and not isinstance(schema, str):
-> 5537     raise exc.ArgumentError(
   5538         "expected schema argument to be a string, "
   5539         f"got {type(schema)}."
   5540     )
   5541 self.tables = util.FacadeDict()
   5542 self.schema = quoted_name.construct(schema, quote_schema)

ArgumentError: expected schema argument to be a string, got <class 'sqlalchem
y.engine.base.Engine'>.
```

In [276…
```
res = conn.execute(book_sales.select())
for r in res:
    print(r)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[276], line 1
----> 1 res = conn.execute(book_sales.select())
      2 for r in res:
      3     print(r)

NameError: name 'conn' is not defined
```

In [277…
```
ins = book_sales.insert().values(book_title='machine learining', price='10.9
conn.execute(ins)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[277], line 1
----> 1 ins = book_sales.insert().values(book_title='machine learining', pric
e='10.99')
      2 conn.execute(ins)

NameError: name 'book_sales' is not defined
```

In [278…

```
res = conn.execute(book_sales.select())
for r in res:
    print(r)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[278], line 1
----> 1 res = conn.execute(book_sales.select())
      2 for r in res:
      3     print(r)

NameError: name 'conn' is not defined
```