

Access a Database with Python - Iris Dataset

The Iris dataset is a popular dataset especially in the Machine Learning community, it is a set of features of 50 Iris flowers and their classification into 3 species. It is often used to introduce classification Machine Learning algorithms.

First let's download the dataset in `SQLite` format from Kaggle:

<https://www.kaggle.com/uciml/iris/> (<https://www.kaggle.com/uciml/iris/>)

Download `database.sqlite` and save it in the `data/iris` folder.



From [Wikimedia](#)

([https://commons.wikimedia.org/wiki/File:Iris_germanica_\(Purple_bearded_Iris\),_Wakehurst_Place,_UK_-_Diliff.jpg#/media/File:Iris_germanica_\(Purple_bearded_Iris\),_Wakehurst_Place,_UK_-_Diliff.jpg](https://commons.wikimedia.org/wiki/File:Iris_germanica_(Purple_bearded_Iris),_Wakehurst_Place,_UK_-_Diliff.jpg#/media/File:Iris_germanica_(Purple_bearded_Iris),_Wakehurst_Place,_UK_-_Diliff.jpg)), by [Diliff](#) ([//commons.wikimedia.org/wiki/User:Diliff](https://commons.wikimedia.org/wiki/User:Diliff)) - Own work, CC BY-SA 3.0 (<http://creativecommons.org/licenses/by-sa/3.0>), [Link](#) (<https://commons.wikimedia.org/w/index.php?curid=33037509>)

First let's check that the sqlite database is available and display an error message if the file is not available (`assert` checks if the expression is `True`, otherwise throws `AssertionError` with the error message string provided):

In [1]:

```
import os
#os.getcwd()
#data_iris_folder_content = os.listdir("iris-species/iris")
```

In [2]:

```
#error_message = "Error: sqlite file not available, check instructions above to download"
#assert "database.sqlite" in data_iris_folder_content, error_message
```

Access the Database with the `sqlite3` Package

We can use the `sqlite3` package from the Python standard library to connect to the `sqlite` database:

In [3]:

```
import sqlite3
```

In [4]:

```
conn = sqlite3.connect('iris-species/database.sqlite')
```

In [5]:

```
cursor = conn.cursor()
```

In [6]:

```
type(cursor)
```

Out[6]:

```
sqlite3.Cursor
```

A `sqlite3.Cursor` object is our interface to the database, mostly through the `execute` method that allows to run any `SQL` query on our database.

First of all we can get a list of all the tables saved into the database, this is done by reading the column `name` from the `sqlite_master` metadata table with:

```
SELECT name FROM sqlite_master
```

The output of the `execute` method is an iterator that can be used in a `for` loop to print the value of each row.

In [7]:

```
for row in cursor.execute('SELECT name FROM sqlite_master'):
    print(row)
```

```
('Iris',)
```

a shortcut to directly execute the query and gather the results is the `fetchall` method:

In [8]:

```
cursor.execute("SELECT name FROM sqlite_master").fetchall()
#cursor.fetchall() and list(cursor) are essentially the same. The different option is
#and instead just loop over the bare cursor object:
```

Out[8]:

```
[('Iris',)]
```

Notice: this way of finding the available tables in a database is specific to `sqlite`, other databases like `MySQL` or `PostgreSQL` have different syntax.

Then we can execute standard `SQL` query on the database, `SQL` is a language designed to interact with data stored in a relational database. It has a standard specification, therefore the commands below work on any database.

If you need to connect to another database, you would use another package instead of `sqlite3`, for example:

- [MySQL Connector \(<https://dev.mysql.com/doc/connector-python/en/>\)](https://dev.mysql.com/doc/connector-python/en/) for MySQL

- [Psycopg \(<http://initd.org/psycopg/docs/install.html>\)](http://initd.org/psycopg/docs/install.html) for PostgreSQL
- [pymssql \(<http://pymssql.org/en/stable/>\)](http://pymssql.org/en/stable/) for Microsoft MS SQL

then you would connect to the database using specific host, port and authentication credentials but then you could execute the same exact SQL statements.

Let's take a look for example at the first 3 rows in the Iris table:

In [9]:

```
sample_data = cursor.execute("SELECT * FROM Iris LIMIT 20").fetchall()
```

In [10]:

```
print(type(sample_data))
sample_data
```

```
<class 'list'>
```

Out[10]:

```
[(1, 5.1, 3.5, 1.4, 0.2, 'Iris-setosa'),
 (2, 4.9, 3, 1.4, 0.2, 'Iris-setosa'),
 (3, 4.7, 3.2, 1.3, 0.2, 'Iris-setosa'),
 (4, 4.6, 3.1, 1.5, 0.2, 'Iris-setosa'),
 (5, 5, 3.6, 1.4, 0.2, 'Iris-setosa'),
 (6, 5.4, 3.9, 1.7, 0.4, 'Iris-setosa'),
 (7, 4.6, 3.4, 1.4, 0.3, 'Iris-setosa'),
 (8, 5, 3.4, 1.5, 0.2, 'Iris-setosa'),
 (9, 4.4, 2.9, 1.4, 0.2, 'Iris-setosa'),
 (10, 4.9, 3.1, 1.5, 0.1, 'Iris-setosa'),
 (11, 5.4, 3.7, 1.5, 0.2, 'Iris-setosa'),
 (12, 4.8, 3.4, 1.6, 0.2, 'Iris-setosa'),
 (13, 4.8, 3, 1.4, 0.1, 'Iris-setosa'),
 (14, 4.3, 3, 1.1, 0.1, 'Iris-setosa'),
 (15, 5.8, 4, 1.2, 0.2, 'Iris-setosa'),
 (16, 5.7, 4.4, 1.5, 0.4, 'Iris-setosa'),
 (17, 5.4, 3.9, 1.3, 0.4, 'Iris-setosa'),
 (18, 5.1, 3.5, 1.4, 0.3, 'Iris-setosa'),
 (19, 5.7, 3.8, 1.7, 0.3, 'Iris-setosa'),
 (20, 5.1, 3.8, 1.5, 0.3, 'Iris-setosa')]
```

In [11]:

```
[row[0] for row in cursor.description]
```

Out[11]:

```
['Id',
 'SepalLengthCm',
 'SepalWidthCm',
 'PetalLengthCm',
 'PetalWidthCm',
 'Species']
```

It is evident that the interface provided by `sqlite3` is low-level, for data exploration purposes we would like to directly import data into a more user friendly library like `pandas`.

Import data from a database to pandas

In [12]:

```
import pandas as pd
```

In [13]:

```
conn = sqlite3.connect('iris-species/database.sqlite')
#iris_data = pd.read_csv('./iris-species/iris.csv')
```

In [14]:

```
iris_data = pd.read_sql_query('SELECT * FROM Iris', conn)
```

In [15]:

```
iris_data.head()
```

Out[15]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [16]:

```
iris_data.dtypes
```

Out[16]:

```
Id          int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species        object
dtype: object
```

pandas.read_sql_query takes a SQL query and a connection object and imports the data into a DataFrame , also keeping the same data types of the database columns. pandas provides a lot of the same functionality of SQL with a more user-friendly interface.

However, sqlite3 is extremely useful for downselecting data **before** importing them in pandas .

For example you might have 1 TB of data in a table stored in a database on a server machine. You are interested in working on a subset of the data based on some criterion, unfortunately it would be impossible to first load data into pandas and then filter them, therefore we should tell the database to perform the filtering and just load into pandas the downsized dataset.

In [17]:

```
iris_setosa_data = pd.read_sql_query("SELECT * FROM Iris WHERE Species == 'Iris-seto"
```

In [18]:

```
iris_setosa_data
print(iris_setosa_data.shape)
print(iris_data.shape)
```

```
(50, 6)
(150, 6)
```

In [19]:

```
iris_versicolor_data = pd.read_sql_query("SELECT * FROM Iris WHERE Species == 'Iris-
```

In [20]:

```
iris_versicolor_data
print(iris_versicolor_data.shape)
```

```
(50, 6)
```

In []: