Notebook_Validation

January 23, 2019

SQL command demo

Jupyter magic functions allow you to replace a boilerplate code snippets with more concise one. Magic functions are pre-defined functions("magics") in Jupyter kernel that executes supplied commands. There are two kinds of magics line-oriented and cell-oriented prefaced with % and %% respectively. To enable the magic we need an ipython-sql library.

```
In [4]: %load_ext sql
        %config SqlMagic.autocommit=False # for engines that do not support autommit
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
In [5]: %%sql sqlite://
            CREATE TABLE writer (first_name, last_name, year_of_death);
            INSERT INTO writer VALUES ('William', 'Shakespeare', 1616);
            INSERT INTO writer VALUES ('Bertold', 'Brecht', 1956);
```

```
Done.
1 rows affected.
1 rows affected.
```

```
Out[5]: []
```

```
In [6]: %sql     select * from writer
```

```
 * sqlite://
Done.
```

```
Out[6]: [('William', 'Shakespeare', 1616), ('Bertold', 'Brecht', 1956)]
```

```
In [7]: result = %sql select * from writer
        df = result.DataFrame()
        df.loc[df['first_name']=='William', ['first_name', 'last_name']]
```

```
 * sqlite://
Done.
```

```
Out[7]:   first_name    last_name
        0    William  Shakespeare
```

```
In [8]: %sql PERSIST df;

        %sql select * from df;
```

```
 * sqlite://
 * sqlite://
Done.
```

```
Out[8]: [(0, 'William', 'Shakespeare', 1616), (1, 'Bertold', 'Brecht', 1956)]
```

```
In [14]: import pandas as pd
         from sqlalchemy import create_engine

         df = pd.read_csv('../datasets/test.csv', encoding = "ISO-8859-1")
         engine = create_engine('sqlite:///tmp.db')
         df.to_sql(con=engine, name='test', if_exists='replace')
```

```
In [15]: %%sql sqlite:///tmp.db

         select * from test
```

```
Done.
```

```
Out[15]: [(0, 30, '1/11/2008', 1, 12, 'Nairobi - Amboseli', 67029, 67429, 0.0, 0, 0, 0, 0, 1),
          (1, 31, '1/14/2008', 1, 12, 'Tsavo - Nairobi', 67429, 67848, 0.0, 0, 0, 0, 0, 1),
          (2, 32, '1/14/2008', 1, 12, 'CRS - Westlands - CRS', 67848, 67850, 0.0, 0, 0, 0, 0,
          (3, 33, '1/17/2008', 1, 12, 'Nairobi - Machakos - Within Town', 67850, 68072, 0.0, 0
          (4, 34, '1/19/2008', 1, 12, 'Machakos - Nairobi', 68072, 68186, 0.0, 0, 0, 0, 0, 1),
          (5, 36, '1/22/2008', 1, 12, 'Nairobi - Nakuru', 68186, 68418, 110.22000120000001, 850
          (6, 37, '1/23/2008', 1, 12, 'Kakuru - Within Town', 68418, 68501, 0.0, 0, 0, 0, 0, 1)
          (7, 38, '1/24/2008', 1, 12, 'Nakuru - Mogotio - Nakuru', 68501, 68737, 0.0, 0, 0, 0,
          (8, 39, '1/26/2008', 1, 12, 'Nakuru - IDP Camps within  Nakuru', 68737, 69389, 0.0, 0
          (9, 74, '1/6/2008', 1, 35, 'Nairobi - nairobi', 49180, 49219, 34.38000107, 3015, 0, 0
```

Note: to store a large volume dataset, recommend to use other relational database, such as mysql, postgres, mssql, etc., rather than sqlite

Test mysql connection

```
In [20]: import mysql.connector
         from mysql.connector import Error

         connection = mysql.connector.connect(host='localhost', database='mysql', user='root',
         cursor = connection.cursor()
         cursor.execute("select * from mysql.user")
         record = cursor.fetchall()
         print(record)
```

```
[('localhost', 'root', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', '
```

```
In [ ]: %%sql

        mysql+mysqlconnector://root:123@localhost/mysql

        %%sql

        select * from user;
```

Linux Commands Demo

Any command that works at the command-line can be used in IPython by prefixing it with the ! character. For example,

```
In [1]: !ls
```

```
DataProfiler.ipynb     derby.log         input01.txt
Spark_DataFrame.ipynb  fibonacci.ipynb         ipython-sql-demo.ipynb
Untitled.ipynb               fibonacci.py         metastore_db
checkPointFolder       gradesdict.json         people.csv
decorator_intro.ipynb  gradesdict.pkl
```

```
In [2]: !pwd
```

```
/mnt/c/github/Learning-Notes/jupyter_notes
```

```
In [13]: directory = !pwd
         print(directory)
         !ls {directory}/.vscode
```

```
['/mnt/c/github/Learning-Notes/jupyter_notes']
ls: cannot access '[/mnt/c/github/Learning-Notes/jupyter_notes]/.vscode': No such file or direc
```

```
In [12]: !ll
```

```
/bin/sh: 1: ll: not found
```

if you play with IPython's shell commands for a while, you might notice that you cannot use !cd to navigate the filesystem:

```
In [15]: !pwd
         !cd metastore_db
         !pwd
```

```
/mnt/c/github/Learning-Notes/jupyter_notes
/mnt/c/github/Learning-Notes/jupyter_notes
```

The reason is that shell commands in the notebook are executed in a temporary subshell. If you'd like to change the working directory in a more enduring way, you can use the %cd magic command:

```
In [3]: !pwd
        %cd ..
        !pwd

/mnt/c/github/Learning-Notes/jupyter_notes
/mnt/c/github/Learning-Notes
/mnt/c/github/Learning-Notes


In [4]: %pwd

Out[4]: '/mnt/c/github/Learning-Notes'
```

This is known as an automagic function, and this behavior can be toggled with the %automagic magic function.

Besides %cd, other available shell-like magic functions are %cat, %cp, %env, %ls, %man, %mkdir, %more, %mv, %pwd, %rm, and %rmdir, any of which can be used without the % sign if automagic is on. This makes it so that you can almost treat the IPython prompt as if it's a normal shell:

```
In [5]: %ls
        %mkdir tmp
        %ls
        %rm -r tmp
        %ls

README.md*  jupyter_notes/  python_codes/  visualization/
README.md*  jupyter_notes/  python_codes/  tmp/  visualization/
README.md*  jupyter_notes/  python_codes/  visualization/
```