Problem set 1 - DDM17 - solutions

Trying out git

This I am assuming was straightforward - the lecture notes should have enough information for this,

SQL tasks

SQL-1. Create the Stars and Observations tables

In the DDM2017 git repository there is a file Database/make_tables_python.py that shows how to create these tables using Python.

SQL-2. Querying using Python

Should be trivial

SQL-2. Completing the lecture

- Where is the FITS image stored for star S5?

```
In [1]: import sqlite3 as lite
In [2]: con = lite.connect("DDM17-python.db")
In [3]: rows = con.execute("""Select s.StarID, o.WhereStored
   ...:    From Stars as s Join Observations as o on s.FieldID = o.ID
   ...:    Where s.Star = 'S5' """)
In [4]: for row in rows:
    print "Star {0} is stored at {1}".format(row[0], row[1])
   ...:
Star 3 is stored at /disks/yaeps-1/StF-045.fits
```

- Give me a list of all stars observed with the same FieldID.

There are many ways to answer this. You can do the work in Python or in SQL. Let me focus on the SQL solution - so I will just show the SQL, you can either type this into sqlite3 or do this in Python. I will also show this in three steps - each step better than the previous. The solution I will go for here is a join with the stars table itself:

First go:

```
select s1.fieldID, s1.Star, s2.Star
from
  Stars as s1
  JOIN Stars as s2
  ON s1.fieldID = s2.fieldID
```

The result of this is

```
# FieldID Star Star
    1      S1   S1
    1      S2   S1
    1      S1   S2
    1      S2   S2
    3      S5   S5
    2      S7   S7
```

which is not very satisfying because there is a lot of repetition and symmetric solutions. We should at least get rid of matches to the star itself.

```
select s1.fieldID, s1.Star, s2.Star
from
  Stars as s1
  JOIN Stars as s2
  ON s1.fieldID = s2.fieldID
Where s1.Star != S2.Star
```

which gives

```
# FieldID Star Star
    1       S2   S1
    1       S1   S2
```

That is better, but still there is a duplication here. The key now is to realise that we have to impose an order instead of a not equal statement. This gives

```
select s1.fieldID, s1.Star, s2.Star
from
  Stars as s1
  JOIN Stars as s2
  ON s1.fieldID = s2.fieldID
Where s1.star > s2.star
```

which then gives:

```
# FieldID Star Star
    1       S2   S1
```

(you can of course swap the order if you wish!)

This is perhaps a type of problem where you would handle it in Python - but if your database is very large that is not a good option.

SQL-2. Creating simple tables.

Several of you sent links to your GitHub accounts which showed that you had been able to solve the problem. The approaches taken are quite different - which is as expected and also perfectly fine. Several used a standard Python approach and that is also what is available now in the GitHub repository - see `Problemsets/Solution-ProblemSet1/make_simple_tables.py`

For the querying there is a second file named query-simple-tables.py that does that. But the code using Pandas is sufficiently simple that we can look at a couple here:

```
con = lite.connect('SimpleTables-default.db')
a)
t = pd.read_sql_query("Select Name, Ra, Decl From MagTable Where B > 16", con)
print(t)
b)
query = """
SELECT m.B, m.R, p.Teff, p.FeH
FROM MagTable as m OUTER LEFT JOIN PhysTable as p
ON m.Name = p.Name
"""
t = pd.read_sql_query(query, con)

c) - almost the same as b but with an extra WHERE FeH > 0
d)
# First get one using a query
col_t = pd.read_sql_query("SELECT Name, B-R as BR FROM MagTable", con)

# Then save it
col_t.to_sql("BRTable", con, if_exists="replace")
```

Running SQL queries on the SDSS databases - CasJobs

This should by straightforward.

Linear regression

See the `Exploring Regression for problem set - solution.ipynb` in the `Problemsets/Solution-ProblemSet1`directory.

But note that for some reason the Lasso routine in sklearn stopped working for me so the solution is not complete for the Lasso.

Finding stars in SDSS

Log into CasJobs. Here you need to build the results by using the UNION operator:

```
SELECT * into mydb.UnionOfStars2 from
 (
(SELECT TOP 100 objID, psfMag_r, psfMag_g-psfMag_r as gr
FROM Star as s
WHERE psfMag_r >= 10 AND psfMag_r < 11)

UNION

 (SELECT TOP 100 objID, psfMag_r, psfMag_g-psfMag_r as gr
FROM Star as s
WHERE psfMag_r >= 11 AND psfMag_r < 12)

 UNION

 (SELECT TOP 100 objID, psfMag_r, psfMag_g-psfMag_r as gr
FROM Star as s
WHERE psfMag_r >= 12 AND psfMag_r < 13)

   UNION

 (SELECT TOP 100 objID, psfMag_r, psfMag_g-psfMag_r as gr
FROM Star as s
WHERE psfMag_r >= 13 AND psfMag_r < 14)

   UNION

     (SELECT TOP 100 objID, psfMag_r, psfMag_g-psfMag_r as gr
FROM Star as s
WHERE psfMag_r >= 14 AND psfMag_r < 15)

   UNION

 (SELECT TOP 100 objID, psfMag_r, psfMag_g-psfMag_r as gr
FROM Star as s
WHERE psfMag_r >= 15 AND psfMag_r < 16)
) as stars
```

Problem - the fraction of red galaxies with redshift.

This is most easily achieved by dividing up the query in two: A first that counts the red galaxies grouped by redshift bins, and a second one which counts all galaxies and then outside you combine these two with a JOIN on the redshift.

```
SELECT red.redshift, cast(red.num AS float)/cast(b.num AS float) as fraction
FROM  (
  Select floor(0.5+z*50.)/50 as redshift, count(*) as num
   From SpecPhoto
  Where
   modelMag_u-modelMag_g > 2
  AND
   z between 0.001 and 0.5
  GROUP BY floor(0.5+z*50.)/50
) as red JOIN
(
Select floor(0.5+z*50.)/50 as redshift, count(*) as num
From SpecPhoto
Where
   z between 0.001 and 0.5
GROUP BY floor(0.5+z*50.)/50
) AS b ON b.redshift = red.redshift

ORDER BY red.redshift
```

The rest of the problem set is not easily given as a solution so I won't spend time on that here.