Working with the Social Sciences Computing Cluster and the School of Professional Studies Database Servers: Overview Tutorial for Students in Predictive Analytics

Students in Predictive Analytics have access to Linux systems at Northwestern University. The Social Sciences Computing Cluster (SSCC) provides full user access to a wide range of analytics software, as well as a path to School of Professional Studies database servers.

Courses in Predictive Analytics utilize R, Python, and/or SAS as analytics software. R, an object-oriented, open-source language for programming with data, is available worldwide and runs on PC/Windows, Mac/OSX, and Linux/Unix computers. We often begin by using R on our personal laptop or desktop computers. For large or computer-intensive jobs, however we can use R on the SSCC.

Python is an object-oriented, open-source language. It is a general-purpose programming language especially strong in data and text preparation, as well as a wide range of applications relevant to predictive analytics and data science. Like R, it runs on PC/Windows, Mac/OSX, and Linux/Unix computers. We often begin by using Python on our personal laptop or desktop computers. For large or computer-intensive jobs, however, we can use Python on the SSCC.

SAS Studio is a commercial product from SAS Inc. It runs on Linux systems and is accessed through web browsers. It is available through the SSCC.

The SSCC is a number of Linux computers in Evanston, Illinois and serves as a research facility for graduate students in the social sciences and business. The School of Professional Studies (SPS) is one of many programs at Northwestern University supporting the SSCC, so SPS graduate students have access to the extensive software of the SSCC. General information about the SSCC is available at

http://www.it.northwestern.edu/research/services/sscc.html

Graduate students in Predictive Analytics are given accounts on the SSCC. If you have not received an account, you can apply for an SSCC account by starting at http://sscc.northwestern.edu and then following the account application link. There is one on the left margin of the SSCC home Web page and another at the bottom of the page. Going thru that link makes database entries that lead to the account creation. The process also puts the applicant thru the usage agreement process. Students should identify themselves as graduate students in SPS.

SSCC accounts are not associated with individual courses or instructors. They are for your use only and remain available as long as you maintain a valid Northwestern NetID. Do not share your SSCC account by giving your NetID password to others. Your SSCC account is tied to your Northwestern University network identity.

Communication between the student and SSCC management is through Northwestern University e-mail.

SSCC account holders have the ability to store files on their user accounts. SPS Predictive Analytics has its own log-in host: dornick, so use dornick for your work. Files can be uploaded and downloaded using software tools employing secure file transfer protocol (SFTP). You may use sftp directly or use FileZilla. FileZilla is available for PC/Windows and Mac/OSX computers, is a free public-domain tool for file transfer. Additional information about file transfers is available at

http://www.it.northwestern.edu/research/sscc/filetransfer.html

Using the SSCC means using Linux because the statistical software programs reside on the SSCC, not on the students' personal computers. Essential Linux commands include those for directory and file management, such as ls, mkdir, cd, and cp. Remember that Linux is case-sensitive.

SSCC management provides a page with recommended Linux reference books at

http://www.it.northwestern.edu/research/sscc/booklist.html

Course syllabi in Predictive Analytics provide additional software references.

Efficient use of the SSCC for statistical programming sometimes requires the use of a Linux-based editor. SSCC management recommends the vim editor for general Linux text editing tasks. For learning the vim editor, you can type the Linux command **vimtutor** Note that you will have less need to learn the vim editor if most of your work is within systems that have embedded editors, such as RStudio, Enthought Canopy, or SAS Studio.

For connecting up and using the SSCC, students can refer to instructions at

http://www.it.northwestern.edu/research/sscc/connect.html

Connecting to the SSCC from a PC/Windows system requires the use of X-Win32 software and establishing a virtual private network (VPN) connection. Connecting to the SSCC from a Mac/OSX computer the use of terminal or X11 software and establishing a virtual private network (VPN) connection.

The VPN connection must be established before logging into the SSCC and doing any work on the dornick host of the SSCC and on the SPS database servers.

Project 1: PostgreSQL

The SPS PostgreSQL database server is shared by students and faculty in the Predictive Analytics program. It is a repository for databases and data sets used across all courses in the program and, as such, it is a protected resource. Students are granted read-only access to selected database objects on the database server.

We work with the dornick SSCC host computer, and we use psql to work with the SPS PostgreSQL server. In the commands below, replace "**netid**" with your Northwestern University NetID. When requested by the systems for security, enter the password associated your NetID. One way to connect to the SPS PostgreSQL server is through the SSCC.

Before connecting to the SSCC, locate yourself in a working directory on your personal computer. It may be convenient to create a directory for the assignment or project and locate yourself within that named directory on your personal computer. We will use the name **project_1** in this example.

You begin by connecting to the SSCC via VPN and logging onto the dornick host. Here is a command that works for that:

```
\ssh -YC netid@dornick.it.northwestern.edu
```

You will be asked to enter your password. The screen cursor will show your dornick home page. Check the directory by typing **pwd**

It may be convenient to create a directory for your work on the dornick SSCC host and then move to that directory. For example, you could type

```
mkdir project_1
cd project_1
```

You can check your location again by typing **pwd** When you exchange files between the SSCC and SPS PostgeSQL server, those files will be found in the directory you have set up for this purpose.

You can connect to the SPS PostgreSQL from the SSCC by using the IP address (or DNS name) for the PostgreSQL server and your NetID. We will use the IP address in this example. The DNS name is **spspostgresql.it.northwestern.edu**.

```
psql -h 129.105.208.226 -U netid -d postgres
```

You should see the general PostgreSQL prompt at this point: **postgres=#**

To see the available databases on the PostgreSQL server, type **\l**

To work with the xyz database, for example, type **\c xyz**

You should now be seeing a PostgreSQL prompt that looks like **xyz=#**

Working within the xyz database, to see the number of records in the items table under the pilot schema, type **SELECT COUNT(acctno) FROM pilot.item;**

When using psql, note that commands that begin with a backslash are **psql** commands and do not require an ending semicolon. But commands that do not begin with a backslash are standard SQL commands and do require an ending semicolon. For the structure of the items database under the **pilot** schema, type

```
\d pilot.item
```

For a listing of the first five records of the xyz database, use an SQL select statement:

```
SELECT * FROM pilot.item limit 5;
```

When working within a user session in PostgreSQL, it is convenient to keep the original tables of a database intact, and work instead with your own views of the database tables or portions of tables. To avoid issues with concurrent users, each user should set up his/her own unique view name. Suppose you use your NetId as part of the view name, selecting just the records of the database that correspond to the month of January. Note that January is in the xyz database as the column/field trandate, coded as 01 or the number 1:

```
CREATE TEMP VIEW netidwork as SELECT * FROM pilot.item
WHERE extract(month FROM trandate) = 1;
```

Let's check the creation of the temporary view and the selection of January items only. To see how many records are in the view, we type

```
SELECT count(acctno) FROM netidwork;
```

To see the first five records of this view, ensuring that all have the correct month coding (date column/field 2009-01-XX, where XX is the day of the month), we type:

```
SELECT * FROM netidwork limit 5;
```

To bring the entire set of January records from the SPS PostgreSQL server to your current working directory on the SSCC dornick computer, you can use an SQL copy command. Specify a comma-delimited text file format and give the file a name. Here we will use the view name as the file name:

```
\copy (SELECT * FROM netidwork) TO 'netidwork.csv'
WITH DELIMITER ','  NULL AS '\null' CSV HEADER
```

When you see the **xyz=#** prompt again, the file transfer will have been complete. Then you can quit your psql connection to the SPS PostgreSQL server by typing **\q**

After quitting psql on the SPS PostgreSQL server, should be back working on the dornick SSCC host computer, where you can continue to work with the new file you have created in the directory where you are located. Or you can use sftp or FileZilla to transport the file down to your personal computer and work with it there. But before you quit dornick and try the sftp or FileZilla, take a look at the file you have downloaded to dornick. First check where you are located with **pwd**

Then check to see that the file exists in the working directory project_1 by typing **ls -la** which will show the file, its size, and file permissions.

Then look at the contents of the file with **more netidwork** which will print the beginning of the file to the console. You can end the printing by typing **q**

Quit the ssh connection to dornick by typing **logout**

Secure file transfer protocol (sftp) is used to transfer files between computers. If you are working on Windows, use FileZilla to execute these transfers.

If you are working on a personal Mac OSX or Linux computer, you can use sftp directly, as shown on the indented text below:

> Suppose you want to use sftp for file transfers to your Mac OSX personal computer. Then you could type
>
> ```
> sftp netid@dornick.it.northwestern.edu
> ```
>
> After you enter your password to log onto sftp, you should see the prompt **sftp>**
>
> To bring the file from dornick down to your personal computer, you can enter the following commands, checking your location, finding your way to the correct working directory, checking the files in that directory, and copying the desired file down to your personal computer:
>
> ```
> pwd
> cd project_1
> ls –la
> get netidwork.csv
> ```
>
> This should bring the file down to your personal computer, adding it to the directory/folder location on your personal computer. Quit your sftp connection to dornick by typing **quit**

When you are finished with your work on the Linux systems, both the SSCC and the SPS database server, you can close your VPN connection.

Back working on your personal computer, you can do what you want with the newly created comma-delimited text file **netidwork.csv** If you began your work from a directory/folder named **project_1** , then that is where you should find the file.

Note that assignments in some classes may require a log of what you have done on the Linux system (the SSCC in our example). Some personal computer applications do not permit a copy-and-paste operation from the command or terminal window. Regardless, you can use the Linux system itself to produce a log of your work, sometimes referred to as a session log. To start collecting the information from the session, you can use the **script** application by typing

```
script project_1.log
```

Then when you are finished collecting information from the Linux session, type **exit** to exit the script application. The session log should be in file **project_1.log** in the directory you have been working within on the Linux system. While located in the working directory on dornick, you can look at this session log by typing

```
more project_1.log
```

When using the **more** process, you see a portion of the file and can hit the return key to see additional lines of the file. Or you can type **control C** to end the **more** process. The **more** process limits the amount of text displayed in the terminal window. A **cat** process, on the other hand, would show the entire file.

As a final step, you would use **sftp** (via FileZilla on Windows or directly on Mac OSX) to transfer the session log file **project_1.log** to your personal computer. This is a plain text file that may be viewed in any text editor.

If you are using a personal computer under Mac OSX or Linux, you can create a session log by executing a **script** process from your personal computer. Being positioned within the project or assignment working directory means that the session log will go into that directory along with data sets, programs, and output generated as you work on the assignment. Then the entire working directory can be copied to a compressed zip archive for delivery with the paper for your assignment or report for the project.

Project 2: Joining Tables/Views with PostgreSQL

Suppose you have another project to do: **`project_2`** . And suppose you set up directories as you have previously, working directories on both your personal computer and the dornick host. Suppose the assignment asks you to join selected columns from two tables in a database called xyz. You check to see that you have a VPN connection established between your personal computer and Northwestern University. Then on dornick, you begin as you did before, but now you are working on a new assignment:

```
ssh -YC netid@dornick.it.northwestern.edu
mkdir project_2
cd project_2
psql -h 129.105.208.226 -U netid -d postgres
\c xyz
```

As a student or faculty member in Predictive Analytics, you cannot create a new database or update existing databases. But for each class you will be able to read data from selected database objects. That is, you can execute SELECT queries against those database objects and create views of those database objects.

A view is a stored query. By creating a view, you create a new database object that acts very much like a table. A view does not itself contain data. It is merely a window into data which are stored on actual tables. You cannot update or delete data from a view, but you can select data from a view, much as you would select data from a table. The SQL syntax is similar to what you would use for a table. Here is an example of a view that joins selected information from two tables in the xyz database. It includes all customer records, adding mail information where it exists.

```
CREATE TEMP VIEW netidleftview AS
SELECT acctno AS customerid, zip FROM pilot.customer;
CREATE TEMP VIEW netidrightview AS SELECT * FROM pilot.mail;
CREATE TEMP VIEW netidjoin AS SELECT * FROM netidleftview
LEFT OUTER JOIN netidrightview ON customerid = acctno;
\d netidjoin
```

The first 100 records of this view are then copied to a text file on dornick:

```
\copy (SELECT * FROM netidjoin LIMIT 100) TO
'netidjoin.csv' WITH DELIMITER ','  NULL AS '\null' CSV
HEADER
```

Quit **`psql`**, logout of dornick, and disconnect from the VPN. Then work with files as required for your assignment or project, transferring those files to your personal computer with FileZilla (on Windows or Mac OSX) or directly with **`sftp`** (on Mac OSX or Linux).

Project 3: Using a MongoDB Document Collection

JavaScript Object Notation (JSON) is the most popular data interchange format on the web, and MongoDB is well suited for working with JSON. An SPS MongoDB database server is shared by students and faculty in the Predictive Analytics program. It is a repository for databases and data sets (document collections) used across all courses in the program and, as such, it is a protected resource. Students are granted read-only access to selected databases on the MongoDB server.

Suppose you want to work with the dornick SSCC host computer and use the mongo shell to connect with the SPS MongoDB server. In the commands below, replace "**netid**" with your Northwestern University NetID. When requested by the systems, enter the password associated your NetID.

Before connecting to the SSCC, locate yourself in a working directory on your personal computer. It may be convenient to create a directory for the assignment or project and locate yourself within that named directory on your personal computer. We will use the name **project_4** in this example.

You begin by connecting to the SSCC via VPN and logging onto the dornick host. Here is the command: **ssh -YC netid@dornick.it.northwestern.edu**

It may be convenient to create a directory for your work on the dornick SSCC host and then move to that directory. For example, you could type

```
mkdir project_3
cd project_3
script project_3.log
```

You can check your location again by typing **pwd** When you exchange files between the SSCC and SPS MongoDB server, those files will be found in the SSCC directory you have set up for this purpose.

You can connect to the SPS MongoDB from the SSCC by using a **mongo** shell with arguments for the IP address of the MongoDB server and your NetID. The MongoDB database shell **mongo** employs a detailed login format. **Ensure that this command to execute the mongo shell is entered in its entirety with no line feeds:**

```
mongo enron --host 129.105.208.225 -u netid -p
  --authenticationMechanism PLAIN
  --authenticationDatabase '$external'
```

You specify the **enron** database both in your login and in a **use** command within the mongo shell. You have access to collections under the **enron** database. These commands show how to begin working with the **messages** collection under the **enron** database:

```
use enron
show collections
db.messages.count()
db.messages.stats()
db.messages.findOne()
```

The **findOne** provides a pretty printing of one document or e-mail message from the **messages** collection:

```
{
      "_id" : ObjectId("4f16fc97d1e2d32371003e27"),
      "body" : "the scrimmage is still up in the air...\n\n\nwebb
said that they didnt want to scrimmage...\n\nthe aggies  are
scrimmaging each other... (the aggie teams practiced on
\nSunday)\n\nwhen I called the aggie captains to see if we could
use their field.... they \nsaid that it was tooo smalll for us to
use...\n\n\nsounds like bullshit to me... but what can we
do....\n\n\nanyway... we will have to do another practice Wed.
night....    and I dont' \nknow where we can practice.... any
suggestions...\n\n\nalso,  we still need one  more person...",
      "headers" : {
            "Date" : "Tue, 14 Nov 2000 08:22:00 -0800 (PST)",
            "From" : "michael.simmons@enron.com",
            "Message-ID" :
"<6884142.1075854677416.JavaMail.evans@thyme>",
            "Subject" : "Re: Plays and other information",
            "To" : "eric.bass@enron.com",
            "X-From" : "Michael Simmons",
            "X-To" : "Eric Bass",
            "X-bcc" : "",
            "X-cc" : ""
      },
      "mailbox" : "bass-e",
      "subFolder" : "notes_inbox"
}
```

From this document, you can see the names of document fields in a nested JSON structure, with fields under **headers** one level removed from the top level. We see various formats for the sender (**From** and **X-From**) and receiver (**X-To**, **X-cc**, and **X-bcc**) fields. The actual e-mail message is under **body**. The values of **mailbox** and **subFolder** show the location of the message within the original e-mail system.

The Enron e-mail archive contains hundreds of thousands of e-mail documents. Suppose you want to find documents relating to selected terms of interest. Searching for text strings within the **messages** collection is fast because database indices have been created for this purpose. You can begin accessing documents within the **messages** collection by using shell commands to count the frequency of selected text strings:

```
db.messages.find({$text:{$search:'Silverpeak'}}).count()

db.messages.find({$text:{$search:'\"death star\"'}}).count()

db.messages.find({$text:{$search:'mark-to-market'}}).count()

db.messages.find(
{$text:{$search:'\"california rolling blackout\"'}}).count()
```

Notice that text string searches are not case-sensitive. Suppose you were to look at the two documents obtained from the last search:

```
db.messages.find(
    {$text:{$search:'\"california rolling blackout\"'}})
```

You would see long messages with distinct mail system identifiers. The two messages came from the mailbox of Jeff Dasovich, though they were located in distinct subfolders of his company e-mail. You can see that they are the same message because they have the same **`Date`** and **`From`** values.

Here is a partial listing of the first message:

```
{ "_id" : ObjectId("4f16fcc8d1e2d32371011a50"), "body" :
"Please see the following . . .

"Date" : "Tue, 12 Jun 2001 03:25:00 -0700 (PDT)", "From" :
"miyung.buster@enron.com", "Message-ID" :
"<12572024.1075849449217.JavaMail.evans@thyme>"

. . .

"mailbox" : "dasovich-j", "subFolder" : "notes_inbox" . . .}
```

Here is a partial listing of the second message:

```
{ "_id" : ObjectId("4f16fcbbd1e2d3237100eda8"), "body" :
"Please see the following . . .

 "Date" : "Tue, 12 Jun 2001 03:25:00 -0700 (PDT)", "From" :
"miyung.buster@enron.com", "Message-ID" :
"<29357302.1075849258369.JavaMail.evans@thyme>"

"mailbox" : "dasovich-j", "subFolder" : "all_documents" . . .}
```

The Enron the **`messages`** collection consists of 145 executive mailboxes, a small portion of the Enron employee population, and there is extensive duplication within the database. When finished, **`exit`** the MongoDB shell and **`exit`** the script process to get your log file. You can look at the contents of your script file on the Linux server by typing

```
more project_3.log
```

Project 4: Working with Text in PostgreSQL

JavaScript Object Notation (JSON), the most popular data interchange format on the web, has been supported on PostgreSQL since version 9.2. Althought PostgreSQL is not thought of as a document database, it can do many of the things a NoSQL document database can do. PostgreSQL NoSQL may be thought of as "not only SQL." For archival text stores that do not require updating of text data in place, PostgesSQL and its JSON data type serves the purpose quite well.

For this example, you use the SPS PostgreSQL to access another repository for the Enron e-mail archive. Students in Predictive Analytics are granted read-only access to this database.

Before connecting to the SSCC, locate yourself in a working directory on your personal computer. It may be convenient to create a directory for this assignment or project and locate yourself within that named directory on your personal computer. We will use the name **`project_4`** in this example.

If you are working on Mac OSX or a personal computer with Linux, then you may want to start a process to log your work on the personal computer side:

```
script project_4.log
```

Suppose you continue working with the dornick SSCC host computer and use the **`psql`** shell to connect with the SPS PostgreSQL server. In the commands below, replace "**`netid`**" with your Northwestern University NetID. And when requested by the server, enter the password associated your NetID.

You begin by connecting to the SSCC via VPN and logging onto the dornick host. Here is a command that works for that:

```
ssh -YC netid@dornick.it.northwestern.edu
```

It may be convenient to create a directory for your work on the dornick SSCC host, move to that directory, and start a process to log your work on dornick:

```
mkdir project_4
cd project_4
script project_4.log
```

You can check your location again by typing **`pwd`** And you can check on all the files, subdirectories, and permissions by typing **`ls -la`**

When you exchange files between the SSCC and SPS PostgreSQL server, those files will be found in the SSCC directory you have set up for this purpose.

You can connect to the SPS PostgreSQL from the SSCC by using the **psql** shell with arguments for the IP address for the PostgreSQL server and your NetID:

```
psql -h 129.105.208.226 -U netid -d postgres
```

To see the available databases under PostgreSQL type **\l**

To work with the **enron_mail** database on the PostgreSQL server, type

```
\c enron_mail
```

These commands show how to begin working with the tables within the **enron_mail** database. There is a **messages** table of JSON records and an associated relational table of text columns called **messages_table**.

The **enron_mail** database **messages** table on the PostgreSQL server has the same Enron data as the **enron** database **messages** collection on the MongoDB server. These are JSON data, and we can use PostgreSQL JSON query operators to look at these data:

```
\d prep.messages
SELECT COUNT(*) FROM prep.messages;
SELECT * FROM prep.messages LIMIT 1;
```

At the top level of JSON keys, there are these data from one e-mail record:

```
SELECT json_object_keys(json_data)
    FROM prep.messages LIMIT 5;
SELECT json_data->>'mailbox' AS "mailbox"
    FROM prep.messages LIMIT 1;
SELECT json_data->>'subFolder' AS "subfolder"
    FROM prep.messages LIMIT 1;
SELECT json_data->>'body' AS "body"
    FROM prep.messages LIMIT 1;
SELECT json_data->>'_id' AS "mongodbid"
   FROM prep.messages LIMIT 1;
```

Drilling down into the header key, there is another level of JSON values:

```
SELECT json_extract_path(json_data, 'headers')
    FROM prep.messages LIMIT 1;
SELECT json_extract_path_text(json_data, 'headers',
    'Date') AS "datetext" FROM prep.messages LIMIT 1;
SELECT json_extract_path_text(json_data, 'headers',
    'Message-ID') AS "messageid"
    FROM prep.messages LIMIT 1;
SELECT json_extract_path_text(json_data, 'headers',
    'Subject') AS "subject" FROM prep.messages LIMIT 1;
```

```
SELECT json_extract_path_text(json_data, 'headers',
    'From') AS "fromaddress" FROM prep.messages LIMIT 1;
SELECT json_extract_path_text(json_data, 'headers',
    'X-From') AS "fromname" FROM prep.messages LIMIT 1;
SELECT json_extract_path_text(json_data, 'headers',
    'To') AS "toaddress" FROM prep.messages LIMIT 1;
SELECT json_extract_path_text(json_data, 'headers',
    'X-To') AS "toname" FROM prep.messages LIMIT 1;
SELECT json_extract_path_text(json_data, 'headers',
    'X-bcc') AS "bccaddress" FROM prep.messages LIMIT 1;
SELECT json_extract_path_text(json_data, 'headers',
    'X-cc') AS "ccaddress" FROM prep.messages LIMIT 1;
SELECT * FROM prep.messages
    WHERE json_data->>'mailbox' = 'phanis-s' LIMIT 1;
```

The **enron_mail** database also has a relational table **messages_table**. With the **messages_table**, it is possible to execute standard SQL queries on the Enron e-mail archive:

```
\d prep.messages_table
SELECT * FROM prep.messages_table
    WHERE mailbox = 'phanis-s' LIMIT 1;
SELECT COUNT(*) FROM prep.messages_table;
SELECT mailbox, COUNT(mailbox)
    FROM prep.messages_table GROUP BY mailbox;
SELECT mailbox, COUNT(mailbox)
    FROM prep.messages_table GROUP BY mailbox
    HAVING COUNT(mailbox) < 100;
SELECT mailbox FROM prep.messages_table LIMIT 1;
SELECT subfolder FROM prep.messages_table LIMIT 1;
SELECT subject FROM prep.messages_table LIMIT 1;
SELECT fromaddress FROM prep.messages_table LIMIT 1;
SELECT fromname FROM prep.messages_table LIMIT 1;
SELECT toaddress FROM prep.messages_table LIMIT 1;
SELECT toname FROM prep.messages_table LIMIT 1;
SELECT ccaddress FROM prep.messages_table LIMIT 1;
SELECT bccaddress FROM prep.messages_table LIMIT 1;
```

To trace all the messages to and from a particular Enron executive, you can use his or her e-mail address, selecting within all the fields that contain e-mail addresses: **fromaddress**, **toaddress**, **ccaddress**, and **bccaddress**. Remember to use single-quotation marks around the text in the **WHERE** clause. For example:

```
SELECT DISTINCT ON (datetext) subject
    FROM prep.messages_table
    WHERE ((fromaddress LIKE '%stephanie.panus%')
    OR (toaddress LIKE '%stephanie.panus%')
    OR (ccaddress LIKE '%stephanie.panus%')
    OR (bccaddress LIKE '%stephanie.panus%')) LIMIT 10;
```

Topics from Enron corporate announcements may be examined:

```
SELECT DISTINCT ON (datetext) subject
    FROM prep.messages_table
    WHERE ((fromaddress LIKE '%enron.announcement%')
    AND (subject ILIKE '%BANKRUPTCY%'))
    LIMIT 10;
```

You might query for a list of the top senders of e-mail messages:

```
SELECT messages_table.fromaddress, messages_table.toaddress,
   COUNT(messages_table.fromaddress)
   FROM prep.messages_table
   GROUP BYmessages_table.toaddress, messages_table.fromaddress
   ORDER BY count(*) DESC LIMIT 100;
```

Extensive query results can be routed to an external file for further text preparation and analysis. Suppose you wanted to gather e-mail addresses as node identifiers for subsequent social network analysis:

```
CREATE TEMP VIEW netidprelim AS
SELECT fromaddress, toaddress, ccaddress, bccaddress,
    datetext FROM prep.messages_table
 WHERE ((fromaddress LIKE '%stephanie.panus%')
 OR (toaddress LIKE '%stephanie.panus%')
 OR (ccaddress LIKE '%stephanie.panus%')
    OR (bccaddress LIKE '%stephanie.panus%'));
\d netidprelim
SELECT COUNT(*) FROM netidprelim;

CREATE TEMP VIEW netiddata AS
SELECT DISTINCT ON (datetext)
    fromaddress, toaddress, ccaddress, bccaddress, datetext
    FROM netidprelim;
\d netiddata
SELECT COUNT(*) FROM netiddata;
```

The records of this view are then copied to a text file on dornick:

```
    \copy (SELECT * FROM netiddata) TO 'netiddata.csv' WITH
    DELIMITER ','  NULL AS '\null' CSV HEADER
```

Quit **`psql`** and check data and log files you have created on dornick. Log out of dornick. You can pull down (**`get`**) database-generated files to your personal computer with FileZilla (on Windows or Mac OSX) or directly with **`sftp`** (on Mac OSX or Linux). Finally, disconnect from the VPN.

Of course, database access and file downloads are just the beginning of the project. Much additional work is needed to prepare text data for analysis. This work is best accomplished with a general-purpose computer language like Python.

Project 5: Accessing MongoDB Databases Directly from Python

You can access a MongoDB database with a Python driver. Begin by installing the **pymongo** package into your Python environment, and then import that package into your namespace of your Python program.

Make sure you have established a VPN connection to the Northwestern University systems prior to using **pymongo** to access data on the SPS MongoDB server. The Python program can ask for you to input your NetID and password. The following code shows how to use **pymongo** to access data from the Enron e-mail archive on the SPS MongoDB database server. It also shows how to work with these data, obtaining information about the messages in the document collection.

```
# Python Access to MongoDB Databases: Jump-Start Script

# prepare for Python version 3x features and functions
from __future__ import division, print_function

# ensure that the pymongo package has been installed
# by using the package manager under Enthought Canopy

# load package into the namespace for this program
from pymongo import MongoClient  # work with MongoDB
import pandas as pd  # DataFrame object work
from datetime import datetime  # text/date manipulation

print('First connect to the Northwestern VPN\n')

# prompt for user's NetID and password
my_netid = raw_input('Enter your NetID: ')
my_password = raw_input('Enter your password: ')

try:
    client = MongoClient("129.105.208.225")
    client.enron.authenticate(my_netid, my_password,\
        source='$external', mechanism='PLAIN')
    print('\nConnected to MongoDB enron database\n')
    success = True
except:
    print('\nUnable to connect to the enron database')

# if connection is successful, work with the database

print('\nCollections in the enron database:')
cols = client.enron.collection_names()
for col in cols:
    print(col)

# work with documents in the messages collection
workdocs = client.enron.messages

# inquire about the documents in messages collection
print('\nNumber of documents: ', workdocs.count())
print('\nOne such document:\n', workdocs.find_one())
one_dict = workdocs.find_one() # create dictionary
```

```
print('\nType of object workdocs: ', type(one_dict))

# how many documents contain 'Silverpeak' in text field
print('How many documents contain the string <Silverpeak>?')
print(workdocs.find({'$text':{'$search':'Silverpeak'}}).count())

# store documents in a list of dictionary objects
selectdocs =\
    list(workdocs.find({'$text':{'$search':'Silverpeak'}}))
print('\nCreated Python object with Silverpeak documents')
print('\nType of object selectdocs', type(selectdocs))
print('\nNumber of items in selectdocs: ', len(selectdocs))

# flatten the nested dictionaries in selectdocs
# and remove _id field
list_of_emails_dict_data = []
for message in selectdocs:
    tmp_message_flattened_parent_dict = message
    tmp_message_flattened_child_dict = message['headers']
    del tmp_message_flattened_parent_dict['headers']
    del tmp_message_flattened_parent_dict['_id']
    tmp_message_flattened_parent_dict.\
        update(tmp_message_flattened_child_dict)
    list_of_emails_dict_data.\
        append(tmp_message_flattened_parent_dict.copy())

print('\nType of object list_of_emails_dict_data',\
    type(list_of_emails_dict_data))
print('\nNumber of items in list_of_emails_dict_data: ',\
    len(list_of_emails_dict_data))

# we can use Python pandas to explore and analyze these data
# create pandas DataFrame object to begin analysis
enron_email_df = pd.DataFrame(list_of_emails_dict_data)
print('\nType of object enron_email_df', type(enron_email_df))

# set missing data fields
enron_email_df.fillna("", inplace=True)

# user-defined function to create simple date object (no time)
def convert_date_string (date_string):
    try:
        return(datetime.strptime(str(date_string)[:16].\
            lstrip().rstrip(), '%a, %d %b %Y'))
    except:
        return(None)

# apply function to convert string Date to date object
enron_email_df['Date'] = \
    enron_email_df['Date'].apply(lambda d: convert_date_string(d))

# date of Enron bankruptcy
BANKRUPTCY = datetime.strptime(str('Sun, 2 Dec 2001'), '%a, %d %b
%Y')

print('\nExamine enron_email_df: ', type(enron_email_df))
print('\nNumber of observations:', len(enron_email_df))
print('\nBeginning observations:')
```

```
print(enron_email_df.head())
# calculate the length of To, From, Subject, and body fields in bytes
# storing the results in a dictionary object message_data
# which is the basis for a list of dictionaries of field sizes
data = []  # initialize list
for i in enron_email_df.index:
    message_data = {}
    message_data['days_before_bankruptcy'] = (BANKRUPTCY - \
        enron_email_df.ix[i]['Date']).days
    message_data['From_bytes'] = len(enron_email_df.ix[i]['From'])
    message_data['To_bytes'] = len(enron_email_df.ix[i]['To'])
    message_data['Subject_bytes'] =
len(enron_email_df.ix[i]['Subject'])
    message_data['body_bytes'] = len(enron_email_df.ix[i]['body'])
    data.append(message_data)
# create DataFrame object for subsequent analysis
enron_email_field_data_df = pd.DataFrame(data)

print('\nSummary statistics for enron_email_field_data_df: ')
print(enron_email_field_data_df.describe())

print('\nRemember to disconnect from the Northwestern VPN')
```

There are clear advantages to using a general-purpose programming language for database access. You connect directly to the database. You eliminate the need for data file exports and imports—no intermediate comma-delimited text or JSON files to manipulate, no **`sftp`** file transfers required. For your database access and queries, you use the same language you use for other work in data science.

With **`pymongo`** access to MongoDB, data are directly converted into Python data objects, and you can manipulate those objects (dictionaries and lists, for example) using all the tools available in Python. The work of data science is easier when you can use one language for database access and queries, data preparation, statistical analysis, and modeling. Python is a good choice in this regard.