Report on Movie Recommender System

**Introducing SAS Software**

Millions of people uses SAS software all over the world—in over 134 countries, at over 60,000 sites. SAS (pronounced sass) is both a company and software. When people say SAS, they sometimes mean the software running on their computers and sometimes mean the company. People often ask what SAS stands for. Originally the letters S-A-S stood for Statistical Analysis

System. SAS products have become so diverse that several years back SAS officially dropped the name Statistical Analysis System, now outgrown, and became simply SAS.

**SAS products**

The roots of SAS software reach back to the 1970s when it started out as a software package for statistical analysis, but SAS didn’t stop there. By the mid-1980s SAS had already branched out into graphics, online data entry, and compilers for the C programming language. In the 1990s the SAS family tree grew to include tools for visualizing data, administering data warehouses, and building interfaces to the World Wide Web. In the new century, SAS has

continued to grow with products designed for cleansing messy data, discovering, and developing drugs, and detecting money laundering. Just as AT&T is now more than telephones and telegraphs, SAS is more than statistics. While SAS has a diverse family of products, most of these products are integrated; that is, they can be put together like building blocks to construct a seamless system. For example, you might use SAS/ACCESS software to read data stored in an external database such as Oracle, analyze it using SAS/ETS software (business modeling and forecasting), use ODS Graphics to produce sophisticated plots, and then forward the results in an e-mail message to your colleagues, all in a single computer program. To find out more about products available from SAS, see the Web site [www.sas.com](http://www.sas.com)

**Operating environments**

SAS software runs in a wide range of operating environments. You can take a program written on a personal computer and run it on a mainframe after changing only the file-handling statements specific to each operating environment. And because SAS programs are as portable as possible, SAS programmers are as portable as possible too. If you know SAS in one operating environment, you can switch to another operating environment without having to relearn SAS.

**The SAS Language**

Many software applications are either menu driven, or command driven (enter a command—see the result). SAS is neither. With SAS, you use statements to write a series of instructions called a SAS program. The program communicates what you want to do and is written using the SAS language. There are some menu-driven front ends to SAS, for example SAS Enterprise Guide, which make SAS appear like a point-and-click program. However, these front ends still use the SAS language to write programs for you. You will have much more flexibility using SAS if you learn to write your own programs using the SAS language. Maybe learning a new language is the last thing you want to do, but be assured that although there are parallels between SAS and languages you know (be they English or JAVA), SAS is much easier to learn.

**SAS programs**

A SAS program is a sequence of statements executed in order. A statement gives information or instructions to SAS and must be appropriately placed in the program. An everyday analogy to a SAS program is a trip to the bank. You enter your bank, stand in line, and when you finally reach the teller’s window, you say what you want to do. The statements you give

can be written down in the form of a program:

I would like to make a withdrawal.

My account number is 0937.

I would like $200.

Give me five 20s and two 50s.

Note that you first say what you want to do, then give all the information the teller needs to carry out your request. The order of the subsequent statements may not be important, but you must start with the general statement of what you want to do. You would not, for example, go up to a bank teller and say, “Give me five 20s and two 50s.” This is not only bad form, but would probably make the teller’s heart skip a beat or two. You must also make sure that all the subsequent statements belong with the first. You would not say, “I want the largest box you have” when making a withdrawal from your checking account. That statement belongs with “I would like to open a safe deposit box.” A SAS program is an ordered set of SAS statements like the ordered set of instructions you use when you go to the bank.

**SAS statements**

As with any language, there are a few rules to follow when writing SAS programs. Fortunately for us, the rules for writing SAS programs are much fewer and simpler than those for English.

The most important rule is

**Every SAS statement ends with a semicolon.**

This sounds simple enough. But while children generally outgrow the habit of forgetting the period at the end of a sentence, SAS programmers never seem to outgrow forgetting the semicolon at the end of a SAS statement. Even the most experienced SAS programmer will at least occasionally forget the semicolon. You will be two steps ahead if you remember this simple rule.

**Layout of SAS programs**

There really aren’t any rules about how to format your SAS program. While it is helpful to have a neat looking program with each statement on a line by itself and indentions to show the various parts of the program, it isn’t necessary.

* SAS statements can be in upper- or lowercase.
* Statements can continue on the next line (as long as you don’t split words in two).
* Statements can be on the same line as other
* Statements can start in any column

**Comments**

To make your programs more understandable, you can insert comments into your programs. It doesn’t matter what you put in your comments—SAS doesn’t look at it. You could put your favorite cookie recipe in there if you want. However, comments are usually used to annotate the program, making it easier for someone to read your program and understand what you have done and why.

There are two styles of comments you can use: one starts with an asterisk (\*) and ends with a semicolon (;). The other style starts with a slash asterisk (/\*) and ends with an asterisk slash (\*/). The following SAS program shows the use of both of these style comments:

\* Read animals' weights from file;

DATA animals;

INFILE 'c:\MyRawData\Zoo.dat';

INPUT Lions Tigers;

PROC PRINT DATA = animals; /\* Print the results \*/

RUN;

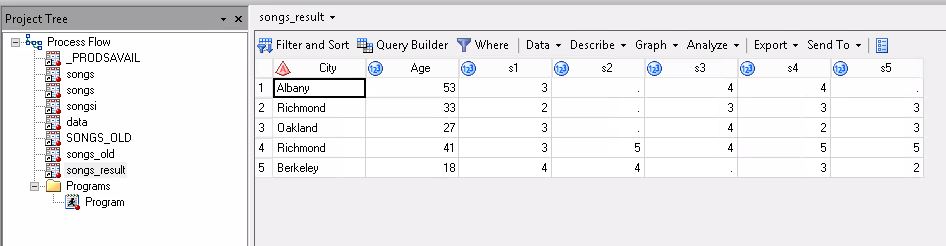
Since some operating environments interpret a slash asterisk (/\*) in the first column as the end of a job, be careful when using this style of comment not to place it in the first column.

**SAS Data Sets**

Before you run an analysis, before you write a report, before you do anything with your data, SAS must be able to read your data. Before SAS can analyze your data, the data must be in a special form called a SAS data set. (See section 2.1 for exceptions.) Getting your data into a SAS data set is usually quite simple as SAS is very flexible and can read almost any data. Once your data have been read into a SAS data set, SAS keeps track of what is where and in what form. All you must do is specify the name and location of the data set you want, and SAS figures out what is in it.

**Variables and observations**

Data, of course, are the primary constituent of any data set. In traditional SAS terminology, the data consist of variables and observations. Adopting the terminology of relational databases, SAS data sets are also called tables, observations are also called rows, and variables are also called columns. Below you see a rectangular table containing a small data set. Each line represents one observation, City, Age, S1, S2, S3, S4 and S5 are variables.



**Data types**

Raw data come in many different forms, but SAS simplifies this. In SAS there are just two data types: numeric and character. Numeric fields are, well, numbers. They can be added and subtracted, can have any number of decimal places, and can be positive or negative. In addition to numerals, numeric fields can contain plus signs (+), minus signs (-), decimal points (.), or E for

scientific notation. Character data are everything else. They may contain numerals, letters, or

special characters (such as $ or !) and can be up to 32,767 characters long.

If a variable contains letters or special characters, it must be a character variable. However, if it

contains only numbers, then it may be numeric or character. You should base your decision on

how you will use the variable. (If disk space is a problem, you may also choose to base your

decision on storage size. See section 11.14.) Sometimes data that consist solely of numerals make more sense as character data than as numeric. ZIP codes, for example, are made up of numerals, but it just doesn’t make sense to add, subtract, multiply, or divide ZIP codes. Such numbers make more sense as character data.

**Missing data**

Sometimes despite your best efforts, your data may be incomplete. The value of a variable may be missing for some observations. In those cases, blanks represent missing character data, and missing numeric data are represented by a single period (.). In the preceding data set, the value of S2 for observation 1 is missing, and its place is marked by a period.

**Rules for names of variables and SAS data set members**

You make up names for the variables in your data and for the data sets themselves. It is helpful to make up names that identify what the data represent, especially for variables. While the variable names A, B, and C might seem like perfectly fine, easy-to-type names when you write your program, the names Sex, Height, and Weight will probably be more helpful when you go back to look at the program six months later.

Follow these simple rules when making up names for variables and data set members:

* Names must be 32 characters or fewer in length.
* Names must start with a letter or an underscore ( \_ ).
* Names can contain only letters, numerals, or underscores ( \_ ). No %$!\*&#@, please.1
* Names can contain upper- and lowercase letters.

This last point is an important one. SAS is insensitive to case so you can use uppercase, lowercase, or mixed case—whichever looks best to you. SAS doesn’t care. The data set name heightweight is the same as HEIGHTWEIGHT or HeightWeight. Likewise, the variable name BirthDate is the same as BIRTHDATE and birThDaTe. However, there is one difference for variable names. SAS remembers the case of the first occurrence of each variable name and uses that case when printing results.

**DATA and PROC Steps**

SAS programs are constructed from two basic building blocks: DATA steps and PROC steps. A typical program starts with a DATA step to create a SAS data set and then passes the data to a PROC step for processing. Here is a simple program that converts miles to kilometers in a DATA step and prints the results with a PROC step:

Data Distance;

Miles = 34.56;

Kilometer = miles\*1.69;

Proc Print data= distance;

Run;

DATA and PROC steps are made up of statements. A step may have as few as one or as many as

hundreds of statements. Most statements work in only one type of step—in DATA steps but not

PROC steps, or vice versa. A common mistake made by beginners is to try to use a statement in the wrong kind of step. You’re not likely to make this mistake if you remember that DATA steps read and modify data while PROC steps analyze data, perform utility functions, or print reports.

DATA steps start with the DATA statement, which starts, not surprisingly, with the word DATA.

This keyword is followed by a name that you make up for a SAS data set.

The DATA step above produces a SAS data set named DISTANCE. In addition to reading data from external, raw data files, DATA steps can include DO loops, IF-THEN/ELSE logic, and a large assortment of numeric and character functions. DATA steps can also combine data sets in just about any way you want, including concatenation and match-merge.

Procedures, on the other hand, start with a PROC statement in which the keyword PROC is

followed by the name of the procedure (PRINT, SORT, or MEANS, for example). Most SAS

procedures have only a handful of possible statements. Like following a recipe, you use basically the same statements or ingredients each time. SAS procedures do everything from simple sorting and printing to analysis of variance and 3D graphics.

A step ends when SAS encounters a new step (marked by a DATA or PROC statement); a RUN,

QUIT, STOP, or ABORT statement; or, if you are running in batch mode, the end of the program.

RUN statements tell SAS to run all the preceding lines of the step and are among those rare, global statements that are not part of a DATA or PROC step. In the program above, SAS knows that the DATA step has ended when it reaches the PROC statement. The PROC step ends with a RUN statement, which coincides with the end of the program.

While a typical program starts with a DATA step to input or modify data and then passes the data to a PROC step, that is certainly not the only pattern for mixing DATA and PROC steps. Just as you can stack building blocks in any order, you can arrange DATA and PROC steps in any order. A program could even contain only DATA steps or only PROC steps.

Project title: Movie Recommendation Engine

Project Idea:

By collecting the data by asking different people to rate all the movies they watched, we would like to group all the movies based on the ratings and suggest the movies that the users may like.

For calculating the similarity between the movies here, we are considering the "Correlation between the movies".

Programming language:

We would like to do this project using SAS.

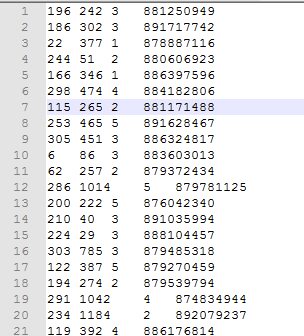
**Code:**

\*\* Assigning library;

Libname Test '\\filer.uncc.edu\home\vkanugul\Test';

\*\* Loading ratings dataset ;

filename source "\\filer.uncc.edu\home\vkanugul\Test\u.data" ;



data Test.Movies\_Ratings ;

attrib

movie\_id informat=best8.

user\_id informat=best8.

rating informat=best8. ;

infile

source dlm='09'x dsd missover ;

input

movie\_id

user\_id

rating ;

run ;

Here we are creating a data set **Movies\_ratings** in Test library by Reading the Data from the source created above.

This program uses the DLM=’09’X option. In ASCII, 09 is the hexadecimal equivalent of a tab character, and the notation ‘09’X means a hexadecimal 09.

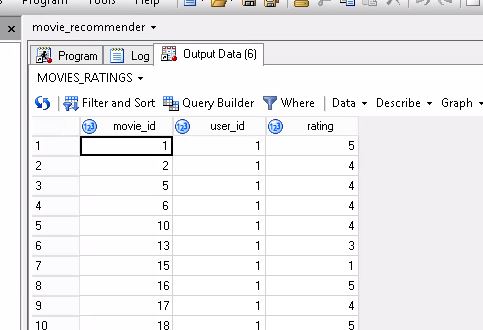
The DSD (Delimiter-Sensitive Data) option for the INFILE statement does three things for you. First, it ignores delimiters in data values enclosed in quotation marks. Second, it does not read quotation marks as part of the data value. Third, it treats two delimiters in a row as a missing value. The DSD option assumes that the delimiter is a comma. If your delimiter is not a comma, then you can use the DLM= option with the DSD option to specify the delimiter.

proc sort data=TEST.movies\_ratings;

by user\_id movie\_id;

run;

Sorting the data set movie\_ratings by user\_id and Movie\_id in ascending order.



filename source clear ;

\*\* Creating movies\_ratings2 dataset which consists of movie\_id and user\_id with ratings greater than 3;

data Test.movies\_ratings2;

set TEST.movies\_ratings;

where rating > 3;

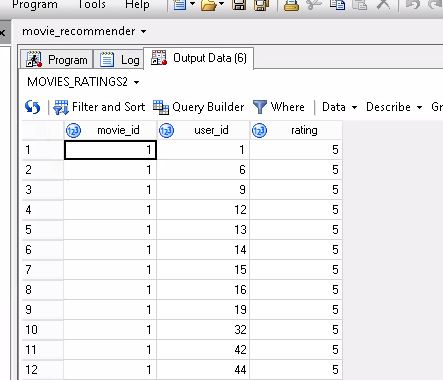
run;

\*\* sorting the dataset movies\_ratings2;

proc sort data=TEST.movies\_ratings2;

by movie\_id Descending rating user\_id;

run;



\*\* Creating coratings of movies if at least 50 users gives the ratings ;

proc sql ;

create table Test.Pairs as

select

a.movie\_id ,

b.movie\_id as movie\_id\_2 ,

a.rating ,

b.rating as rating\_2,

a.user\_id

from

Test.Movies\_Ratings a

join Test.Movies\_Ratings b on a.user\_id = b.user\_id

where

a.movie\_id < b.movie\_id

group by

1, 2

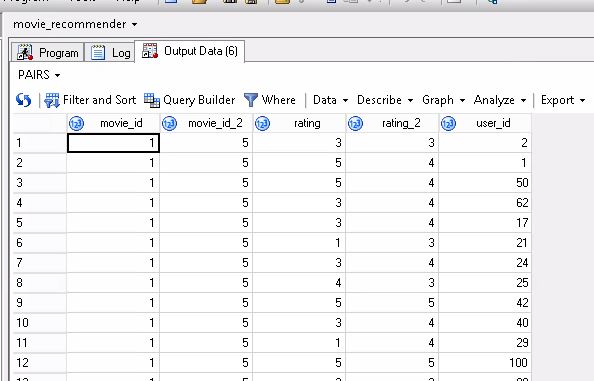
having

count(\*) >= 50

order by

1, 2 ;

quit ;

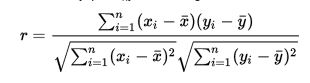


**Pearson Correlation**:

The **Pearson correlation coefficient** (**PCC**), also referred to as the **Pearson's *r*** or **Pearson product-moment correlation coefficient** (**PPMCC**), is a measure of the linear [correlation](https://en.wikipedia.org/wiki/Correlation) between two variables *X* and *Y*. It has a value between +1 and −1, where 1 is total positive linear correlation, 0 is no linear correlation, and −1 is total negative linear correlation. It is widely used in the sciences. It was developed by [Karl Pearson](https://en.wikipedia.org/wiki/Karl_Pearson) from a related idea introduced by [Francis Galton](https://en.wikipedia.org/wiki/Francis_Galton) in the 1880s.

Pearson's correlation coefficient is the [covariance](https://en.wikipedia.org/wiki/Covariance) of the two variables divided by the product of their [standard deviations](https://en.wikipedia.org/wiki/Standard_deviations). The form of the definition involves a "product moment", that is, the mean (the first moment about the origin) of the product of the mean-adjusted random variables; hence the modifier *product-moment* in the name.

Pearson's correlation coefficient when applied to a [sample](https://en.wikipedia.org/wiki/Sample_(statistics)) is commonly represented by the letter *r* and may be referred to as the *sample correlation coefficient* or the *sample Pearson correlation coefficient*. We can obtain a formula for *r* by substituting estimates of the covariances and variances based on a [sample](https://en.wikipedia.org/wiki/Statistical_sample) into the formula above. So if we have one dataset {*x*1,...,*xn*} containing *n* values and another dataset {*y*1,...,*yn*} containing *n* values then that formula for *r* is:{\displaystyle r={\frac {\sum \_{i=1}^{n}(x\_{i}-{\bar {x}})(y\_{i}-{\bar {y}})}{{\sqrt {\sum \_{i=1}^{n}(x\_{i}-{\bar {x}})^{2}}}{\sqrt {\sum \_{i=1}^{n}(y\_{i}-{\bar {y}})^{2}}}}}}



Where Xi and Yi are the ratings of the movie\_id and Movie\_id\_2 given by the users respectively.

\*\* calculating Pearson correlation using pairs table and Proc corr ;

proc corr data=Test.Pairs noPrint out=Test.Recommendations (where=(\_TYPE\_ in ('CORR')) ) pearson;

by movie\_id movie\_id\_2 ;

var rating ;

with rating\_2 ;

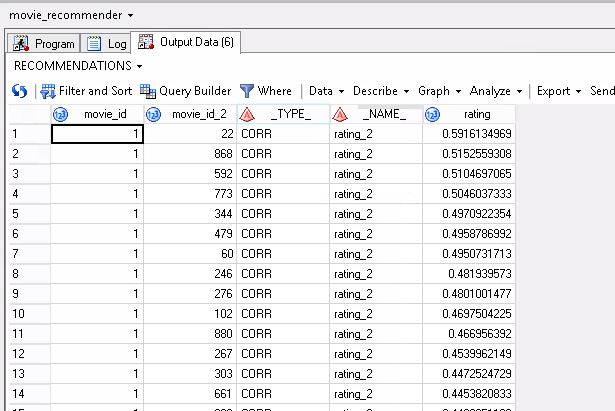
run ;

\*\* sorting recommendations by movie id ;

proc sort data=Test.Recommendations ;

by movie\_id descending rating ;

run ;



\*\*fetching top 3 movies corelating movies for each movie;

data Test.Recommendations1 ;

set Test.Recommendations ;

retain counter ;

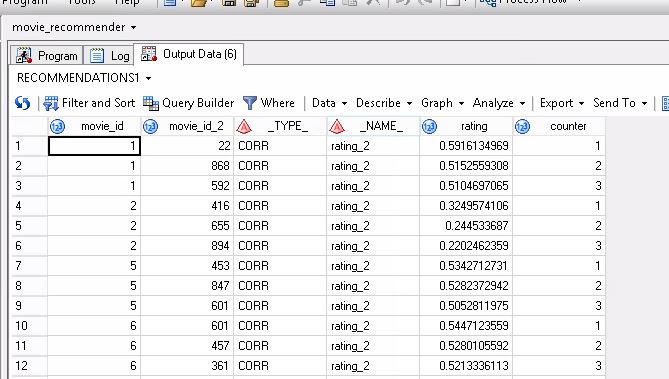
by movie\_id ;

if first.movie\_id then counter = 0 ;

counter + 1 ;

if counter <= 3 ;

run ;



\*\*Recommending movies to the users based on Movies\_ratings2 and Recommendations1 tables;

proc sql ;

create table Test.Recommendations2 as

select

a.user\_id ,

b.movie\_id\_2 as suggested\_movie ,

b.movie\_id as Based\_on\_Rating\_given\_to

from

Test.movies\_ratings2 a

join Test.Recommendations1 b on a.movie\_id = b.movie\_id;

quit ;

\*\*sorting finalrecommendations table;

proc sort data= Test.Recommendations2 nodup;

by user\_id suggested\_movie Based\_on\_Rating\_given\_to ;

run;

