Lecture 5 Example - Shell Workflow

All the following examples are based on the adult dataset from the UCI Machine Learning repository also known as "Census Income" dataset.

This data set is commonly used to predict whether income exceeds $50K/yr based on census data. With 48842 rows and 14 attributes, it is not a large dataset by far but will be sufficient to illustrate the examples.

Downloading the Data

Although the data is stored with the .data extension, it is a well-formatted CSV file.

```
curl http://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data > adult.data
```

Cleaning the Data

You'll notice that the file does not have a header line. The names of columns are also not within the file. You can add that header line by concatenating two files using the `cat` command. The header file is not in the original dataset and you need to create it. To do so, you can `echo` the comma separated list of column names into a `header.csv` file.

```
echo "age,workclass,fnlwgt,education,education-num,marital-status,occupation,relationship,race,sex,capital-gain,capital-loss,native-country,class" > header.csv
```

In this example, you used the shell redirection `>` character to dump the output of `cat` to the `adult.csv` file. The `>` will either create a file or replace its content entirely if the file already exists. Doubling the symbol `>>` will *append* the new content to an already existing file without erasing its content. Let's now add the `header.csv` file at the beginning of the `adult.data` file. At the same time, you rename `adult.data` to `adult.csv` since it is, after all, a CSV formatted file.

```
cat header.csv adult.data > adult.csv
```

Check that the first row of adult.csv contains the column names:

```
head -n 1 adult.csv
```

Inspecting and Cleaning the Data

Modify a file with `sed` - Another frequent data mining scenario happens when a file is corrupted or

badly formatted, such as with non UTF-8 characters or a misplaced comma. You can correct that file without actually opening it using the `sed` command.

The generic `sed` pattern is: `sed "s/<string to replace>/<string to replace it with>/g" <source_file> > <target_file>`.

The `adult.csv` dataset uses the `?` character to denote a missing value. You can replace that character with a better suited default value using `sed`. The empty string is preferable as an indication of a missing value as it will be interpreted as a `NaN` value when loading the data in a Pandas DataFrame.

```
grep ", ?," adult.csv | wc -l
```

This gives you a count of 2399 lines with at least one column with a missing value denoted by `?`. The following command will replace all the columns with `?` by an empty string. All the cells that only contain the `?`, followed by a space will now be truly empty.

```
sed "s/, ?,/,,/g" adult.csv >  adult_replaced.csv
```

Note that you use the column delimiter `,` in the source and target strings to avoid replacing legit question marks that could be present elsewhere in the dataset.

If we want to overwrite the changes and save the file back as `adult.csv`, we can simply run:

```
mv adult_replaced.csv adult.csv
```

The next command counts the number of duplicated lines in `adult.csv`.

```
sort adult.csv | uniq -d | wc -l
```

...and shows that there are 23 duplicates.

The next command takes the output of all lines with added repetition counts, sorts in reverse order and outputs the first 3 duplicates:

```
sort adult.csv | uniq -c | sort -r | head -n 3
```

There are many powerful options that can be obtained by combining `sort` and `uniq` with different flags. Use the `man sort` and `man uniq` pages to further explore these commands.

The great thing about CSV files and shell commands is that you can also work at the column level by using `cut` to select a particular column. `cut` takes two main flags: `-d` to specify the *column delimiter*

and `-f` to specify the *columns* you want to work on. In the following example, you use `cut` to find the number of unique values taken by the categorical variable `workclass` (column 2).

First select the column `workclass` and pipe to `head` to verify that you have the right column:

```
cut -d , -f 2 adult.csv | head -n 3
```

Now, to count uniques, you `sort` the output of `cut` and `pipe` the result to `uniq -c`

```
cut -d , -f 2 adult.csv | sort | uniq -c
```

Which tells you, for instance, that you have 1837 null values, and that the main class is by far the `Private` class with 22969 occurrences.

The command line above is similar to the `value_counts()` method applied to a DataFrame containing the `adult.csv` data.