## Section 1: Data Pipelines

The objective of this section is to design and implement a solution to process a data file on a regular interval (e.g. daily). Assume that there are 2 data files `dataset1.csv` and `dataset2.csv`, design a solution to process these files, along with the scheduling component. The expected output of the processing task is a CSV file including a header containing the field names.

You can use common scheduling solutions such as `cron` or `airflow` to implement the scheduling component. You may assume that the data file will be available at 1am everyday. Please provide documentation (a markdown file will help) to explain your solution.

Processing tasks:

- Split the `name` field into `first\_name`, and `last\_name`

- Remove any zeros prepended to the `price` field

- Delete any rows which do not have a `name`

- Create a new field named `above\_100`, which is `true` if the price is strictly greater than 100

\*Note: please submit the processed dataset too.\*

Thoughts on section 1 data pipelines: Basically, this is to test on extracting data from a data source, transform by cleansing the data to improve the quality and establish consistency and load the data into a target database.

**Split the `name` field into `first\_name`, and `last\_name`:**

Based on the data files `dataset1.csv` and `dataset2.csv` downloaded from Github, under the first column ‘name’, it seems that the first name and last name are jumbled up together as a single string.

We are supposed to split the string into separate first name and last name. There are about 5000 strings in the ‘name’ column for both datafiles. As there are no commas in between the ‘first name’ and ‘last name’, we need to insert a blank space or delimiter in order to achieve the task. It is also viewed as to create sub-strings using the split method from a single string.

For example: This name William Garcia is a single string from the ‘dataset2.csv’ in the ‘name’ field. We need to split the name to ‘first name’ and ‘last name’ whereby in between there is a whitespace which we will use to the string into substring with the following python command:

substring = 'William Garcia'

print (substring [0:substring.index(' ')])

**Remove any zeros prepended to the `price` field:**

Based on my understanding, this is to remove any leading zeros from a number given as a string. For example, we have this first row with ‘first\_name‘ = Daniel, ‘last\_name’ = Martin, given the ‘price’ = 0.30019336 and ‘above\_100’ = false.

We want the desired output value of the ‘price’ to be only .30019336 which is to remove the leading 0 from the number given as a string. As such, we can try to use the iteration statement to remove leading zeros in a string in python.

Example: A = ['0.30019336']

removeleading = []

for x in A:

while x[0] == "0":

x = x[1:]

removeleading.append(x)

print(removeleading)

**Delete any rows which do not have a `name`:**

As for this particular task, we are supposed to delete any rows which do not have a ‘name’. In this case, my understanding is that we are deleting any null values in the ‘name’ field which is erasing any redundant data since such rows only have values in the ‘price’ field and null in the ‘name’ field.

First, we will import the library pandas from python. Then, we will get the dataset by reading from the file directory that you have stored the file in. We will use the orders commands to display those object which is null. From there, it will show us the entire data frame with the true and false value. True value indicates whether that particular cell or the intersection of row and column is having null value and vice versa.

Since our dataset has about 5000 rows which will take a few seconds and messy on the screen, so I will use head function. Once the result is out, we will pay attention to those cell that contain the true value. If there are none, then the data set does not have any null values. If the return result table have true values in certain cell, we will move on to drop all the cells that have null or return true values.

To further group all the cells that are having null values in a particular row/column, we can use the sum function. With the return results, we are only interested in the ‘name’ field. Thus, if there are some returned result, we can then zoom in into the ‘name’ column and identify them by filtering out the results. If all cells have a value and does not contain null value, then it will not do anything.

Example: import pandas as pd

orders = pd.read\_csv('<file directory\dataset1>.csv',0)

orders[orders['Name'].isnull()]

orders.shape

orders.dropna(how = 'all').shape

**Create a new field named `above\_100`, which is `true` if the price is strictly greater than 100:**

For this task, we are told to insert a new column with header name ‘above\_100’ into our csv dataset. In this, we need to compare the data of each single cell in the ‘price’ field. We can use if & else statement and comparing each cell whether it is smaller than 100 or greater than 100. If it is greater than 100, then it will return a true statement. Then, if it is smaller than 100, it will return a false statement.

Example:

a = 258.1809089

b = 40.20849552

c = 119.0382607

if a > 100:

print("True")

else:

print("False")

if b > 100:

print("True")

else:

print("False")

if c > 100:

print("True")

else:

print("False")

Scheduling Component: We will be using Apache Airflow DAD in order to schedule the task whereby the data file will be available at 1am everyday.

Example:

from airflow.models.dag import DAG

from airflow import DAG

from airflow.operators.bash import BashOperator

default\_args = {

'owner': 'coder2j',

'retries': 5,

'retry\_delay': timedelta(minutes=5)

with = DAG (

default\_args=default\_args

dag\_id="dag\_with\_cron\_expression\_v01"

start\_date=datetime(2021, 1, 12)

schedule\_interval='@daily'

schedule\_interval=' 0 1 \* \* \*' #cron expression

) as dag:

task1 = BashOperator(

task\_id='task1'

bash\_command="echo dag with cron expression!"

)

task1