Conversion Details Documentation

Overview :

The purpose of this documentation is to help you understand the code. At the end, we will export a final file for us to import into conversion details template for further analysis for the campaign that you are analyzing.

Reminder :

Before you start to run this script, please make sure Python, the programming language used to write this script, and module pip have both been installed on your computer in order to avoid any errors. If you have not installed the required programs stated above, please see below links for the installation guide.

- How to install Python
- How to install pip

I would suggest you install ipython/Jupyter notebook to facilitate data analysis.

- How to install Jupyter Notebook

There are many packages/modules that have been built and shared in Python open source environment to let uers use as long as the packages are installed on their local computers. Below are the packages/modules (pandas, datetime, calendar, and os) that we will need in our script. If you encounter an error **ImportError : No module named "xyz"** while running the script, just open your command prompt and simply type in `pip install xyz` then you should be good to go!

Let's get started:

This snippet is to import all of the modules/packages in advance for our later use.

```
import pandas as pd
from pandas import Series
import datetime
import calendar
import os
```

On Jupyter Notebook, pandas set 20 visible columns as default. In order to make all the columns visible, we can simply input the below :

```
pd.set_option("display.max_columns", 50)
pd.get_option("display.max_columns")
```

This chunk of code is to ask you to input all the files you would like to combine together into a final file for data cleaning use. The limit for the number of input files here is set as 10, which is a total of 11 files since the first number starts at 0 in Python. **If you have more than 11 files needed to be combined, you can easily change the range number.** `file_size = list(range(10))` After this piece of code, you will have all the input files combined and assigned to a variable called "raw".

```
file_size = list(range(10))

for i in file_size:

    if i == 0:
        file_name = raw_input("Please type in path of the file > ")
        input_file = pd.read_csv(file_name)
        print "input_file shape : ", input_file.shape
        raw = input_file

        answer = raw_input("If there are other files, please type \"y\", otherwise,
press \"n\" > ")
        if answer == "y":
            continue
        else:
            print "The raw file comprises", i+1,"file.","The shape of the raw file
: ", raw.shape
            break

    else:
        file_name = raw_input("Please type in path of the file > ")
        input_file = pd.read_csv(file_name)
        print "input_file shape : ", input_file.shape
        raw = raw.append(input_file, ignore_index= True)

        answer = raw_input("If there are other files, please type \"y\", otherwise,
press \"n\" > ")
        if answer == "y":
            continue
        else:
            print "The raw file comprises", i+1,"files.", "The shape of the raw
file : ", raw.shape
            break
```

The original HD coversion details report from TTD consists of 99 columns but, for our analysis, not all of them are useful. So, let's get rid of the unnecessary ones by assigning useful columns to a variable called columns and slicing from raw.

```
columns = ["Conversion Time","TDID","Conversion Id","# Impressions","# Display
Clicks",
                    "Tracking Tag Name","First Impression Time","First Impression
Campaign Name",
                    "First Impression Ad Group Name","Last Impression Time","Last
```

```
Impression Campaign Name",
                    "Last Impression Ad Group Name","Last Impression Site","Last
Impression Country","Last Impression Metro",
                    "Attribution Model","XDIDs","First Impression Device
Type","Last Impression Device Type",
                    "Conversion Device Type"]


raw = raw[columns]

print raw.shape
```

This step is to discard all the rows with 0 impression and to reset the index of rows to avoid potential issues occurring in following phases.

```
print len(raw["# Impressions"])

raw = raw[raw["# Impressions"] != 0]
print len(raw["# Impressions"])

raw = raw.reset_index(drop=True)
print len(raw["# Impressions"])
```

In order to avoid confusion regarding to Conversion Device Path, here we set First and Last Impression Device Types as "x" once # Impressions as 1

```
raw.loc[raw["# Impressions"] == 1, "First Impression Device Type"] = " x "
raw.loc[raw["# Impressions"] == 1, "Last Impression Device Type"] = " x "
```

Let's have a sneak peak of current data

```
raw.head()
```

Mathematical calculation cannot be applied to Object data types of data in Python. To make the calculation work, we should convert the data types of timsestamp columns from Object to Time. Here is the step for data type conversion as well as for replacing minutes and seconds with 0 to make our following calculation easier.

```
conversion_time_list = []

for i in raw["Conversion Time"]:
    t = datetime.datetime.strptime(i,"%Y-%m-%d %H:%M:%S.%f")
    t = datetime.datetime.replace(t,minute = 0, second = 0, microsecond = 0)
```

```
    conversion_time_list.append(t)


last_impression_list = []

for i in raw["Last Impression Time"]:
    t = datetime.datetime.strptime(i,"%Y-%m-%d %H:%M:%S.%f")
    t = datetime.datetime.replace(t,minute = 0, second = 0, microsecond = 0)
    last_impression_list.append(t)

first_impression_list = []

for i in raw["First Impression Time"]:
    t = datetime.datetime.strptime(i,"%Y-%m-%d %H:%M:%S.%f")
    t = datetime.datetime.replace(t,minute = 0, second = 0, microsecond = 0)
    first_impression_list.append(t)


raw["cal_conversion_time"] = Series(conversion_time_list)
raw["cal_last_impression_time"] = Series(last_impression_list)
raw["cal_first_impression_time"] = Series(first_impression_list)
```

Now, we have all timestamp columns with correct data type available for calculation. It's time to calculate interval time within "First Impression","Last Impression", and "Conversion". At the end of this snippet, we will get the impression lag times formatted in days with one decimal which is calculated by hour/24 plus day.

```
raw["Last To Convert"] = pd.to_datetime(raw["cal_conversion_time"]) -
pd.to_datetime(raw["cal_last_impression_time"])
raw["First To Last"] = pd.to_datetime(raw["cal_last_impression_time"]) -
pd.to_datetime(raw["cal_first_impression_time"])
raw["First To Convert"] = pd.to_datetime(raw["cal_conversion_time"]) -
pd.to_datetime(raw["cal_first_impression_time"])

list_day = []
list_time = []

for row in raw["Last To Convert"]:
    date = str(row).split(" days ")
    list_day.append(date[0])
    list_time.append(date[1])

raw["Last_To_Convert_Day"] = Series(list_day)
raw["Last_To_Convert_Time"] = Series(list_time)


list_day = []
list_time = []

for row in raw["First To Last"]:
    date = str(row).split(" days ")
    list_day.append(date[0])
    list_time.append(date[1])
```

```
raw["First_To_Last_Day"] = Series(list_day)
raw["First_To_Last_Time"] = Series(list_time)


list_day = []
list_time = []


for row in raw["First To Convert"]:
    date = str(row).split(" days ")
    list_day.append(date[0])
    list_time.append(date[1])

raw["First_To_Convert_Day"] = Series(list_day)
raw["First_To_Convert_Time"] = Series(list_time)



day_ratio_list = []

for i in raw["Last_To_Convert_Time"]:
    t = datetime.datetime.strptime(i, "%H:%M:%S")
    day_ratio = round(float(t.hour) / 24, 1)
    day_ratio_list.append(day_ratio)

raw["Last_To_Convert_ratio"] = Series(day_ratio_list)


day_ratio_list = []

for i in raw["First_To_Last_Time"]:
    t = datetime.datetime.strptime(i, "%H:%M:%S")
    day_ratio = round(float(t.hour) / 24, 1)
    day_ratio_list.append(day_ratio)

raw["First_To_Last_ratio"] = Series(day_ratio_list)


day_ratio_list = []

for i in raw["First_To_Convert_Time"]:
    t = datetime.datetime.strptime(i, "%H:%M:%S")
    day_ratio = round(float(t.hour) / 24, 1)
    day_ratio_list.append(day_ratio)

raw["First_To_Convert_ratio"] = Series(day_ratio_list)

raw["Last_To_Convert_Day"] = pd.to_numeric(raw["Last_To_Convert_Day"])
raw["Last_To_Convert_ratio"] = pd.to_numeric(raw["Last_To_Convert_ratio"])
raw["Last_To_Convert"] = raw["Last_To_Convert_Day"] + raw["Last_To_Convert_ratio"]


raw["First_To_Last_Day"] = pd.to_numeric(raw["First_To_Last_Day"])
raw["First_To_Last_ratio"] = pd.to_numeric(raw["First_To_Last_ratio"])
raw["First_To_Last"] = raw["First_To_Last_Day"] + raw["First_To_Last_ratio"]
```

```
raw["First_To_Convert_Day"] = pd.to_numeric(raw["First_To_Convert_Day"])
raw["First_To_Convert_ratio"] = pd.to_numeric(raw["First_To_Convert_ratio"])
raw["First_To_Convert"] = raw["First_To_Convert_Day"] +
raw["First_To_Convert_ratio"]
```

After calculating impression lag times, let's dig deeper from our data.

- Conversion Device Path : Replacing "other" device type with "PC" and adding a "->" in between three impression device columns.

```
raw[["First Impression Device Type","Last Impression Device Type","Conversion
Device Type"]] = raw[["First Impression Device Type","Last Impression Device
Type","Conversion Device Type"]].replace("Other","PC")


raw["Conversion Device Path"] = raw["First Impression Device Type"] + "->" +
raw["Last Impression Device Type"] + "->" + raw["Conversion Device Type"]
```

- DOW : Day of Week

```
day_list = []

for row in raw["Conversion Time"]:
    test = datetime.datetime.strptime(str(row),"%Y-%m-%d %X.%f").date()
    day = datetime.datetime.weekday(test)
    dow = calendar.day_name[day]
    day_list.append(dow)

raw["DOW"] = Series(day_list)
```

- HOD : Hour of Day

```
HOD = []
for row in raw["cal_conversion_time"]:
    hour = str(row).split(" ")
    hour = hour[1]
    hour = str(hour).split(":")
    HOD.append(hour[0])

raw["HOD"] = Series(HOD)
```

- Ad Group Path

```
raw["Ad Group First to Last Imps Path"] = raw["First Impression Ad Group Name"] +
"->" + raw["Last Impression Ad Group Name"]
```

Last but not the least, we can make our final file leaner by excluding unnecessary columns created during the data cleaning process i.e "Last_To_Conver_Day".

```
raw_final = raw[["Conversion Time","TDID","Conversion Id","# Impressions","#
Display Clicks",
                    "Tracking Tag Name","First Impression Time","First Impression
Campaign Name",
                    "First Impression Ad Group Name","Last Impression Time","Last
Impression Campaign Name",
                    "Last Impression Ad Group Name","Last Impression Site","Last
Impression Country",
                    "Last Impression Metro", "Attribution Model","XDIDs","First
Impression Device Type",
                    "Last Impression Device Type","Conversion Device Type",
                    "First_To_Convert","First_To_Last","Last_To_Convert",
"Conversion Device Path",
                    "Ad Group First to Last Imps Path","Campaign Conversion Path"
"DOW","HOD"]]
```

Let's have a sneak peak of our final file's structure before export

```
print raw_final.shape
raw_final.head(5)
```

- Tracking Tag Filtering

If you would like to look into the specific tracking tags, type in "y" and the name of tracking tags. Otherwise, type in "n" and skip the process.

```
x = raw_input("Would you want to filter the final data by certain tracking tag ? ")

if x == "y":
    tag = raw_final["Tracking Tag Name"].unique()

    for i in tag:
        print "Tracking Tag : ", i

    index = raw_input("Please type in the tracking tag you would like to include in
the final file > ")

    final = raw_final[raw_final["Tracking Tag Name"] == index]

    x = raw_input("Are there any other tracking tags ? ")

    while x == "y":
        index = raw_input("Please type in the tracking tag you would like to
include in the final file > ")
        p = raw_final[raw_final["Tracking Tag Name"] == index]
```

```
        final = final.append(p, ignore_index= True)
        x = raw_input("Are there any other tracking tags ? ")

    else:
        print final.shape

else:
    final = raw_final
    print final.shape
```

This step is to see what tracking tag will be included in the final file, which is helpful especially you have specific tags filtered from previoud step.

```
print final["Tracking Tag Name"].unique()
```

Finally, it's time to harvest the effort!!

```
name = raw_input("Please name your file with the file type .xlsx> ")

print "The file", name, "is generating..."

writer = pd.ExcelWriter(name)
final.to_excel(writer, index= False)
writer.save()

print "The file", name, "is saved under path", os.getcwd()
```