Goal: This scripts goal is to take in two csv files and generate a csv file that combines the two, and contains statistics that help differentiate the differences between Image21 and Text21.

The script begins by importing pandas, numpy and datetime.

# import pandas and numpy to generate dataframes and perform calculations

import pandas as pd

import numpy as np

from datetime import datetime

* Pandas is used for creating dataframes that can be used for calculating statistics and generating/importing data from csv files.
* Numpy is required by pandas to generate dataframes and perform calculations on these dataframes
* Datetime is used to convert the log data’s time stamps into datetime that allows for simple time calculation

Jump ahead to where the script begins calling functions.

The first function called is csv\_to\_df().

# Create dataframes out of the two csv files and add their headers

df\_image = csv\_to\_df('imagept21.csv') #image21

df\_text = csv\_to\_df('text21.csv') #text21

csv\_to\_df() takes in a string value that contains a csv file.

def csv\_to\_df(csv\_file):

    # generate dataframe from csv

    # sep: Delimiter to use, names: the column headers in order

    df = pd.read\_csv(csv\_file, sep=',', names=["time", "user", "site", "scheme", "mode", "event", "event details", "data"])

    #drop columns that aren't needed

    df = df.drop(columns=['site', 'data', 'scheme', 'mode'])

    #Remove any rows that are not a login event or enter event with event detail start (event/start and login/success or fail)

    df = df.loc[(df['event'] == 'login') | ((df['event'] == 'enter')  & (df['event details'] == 'start'))]

    return df

The function starts by opening the csv file and converting the data within it into a dataframe. It separates the data by commas as specified in ‘sep’. At the same time, it applies the headers to the dataframe (as the csv file does not contain any).

The next part will drop unnecessary columns, as they aren’t required to generate our resulting csv file.

Finally, the function will remove any rows that is not a ‘login ‘event or not an ‘enter’ event with the event detail of ‘start’. And return the resulting dataframe to its callers.

The next function call is calculate\_time\_df().

# Obtain a data frame with the time taken to complete logins

df\_image = calculate\_time\_df(df\_image)

df\_text = calculate\_time\_df(df\_text)

This function will take a dataframe as a parameter. Then will generate a dataframe that contains the amount of time taken for a login for each login.

def calculate\_time\_df(df):

    #variables

    start = None # the initial login time

    user = None # The user id of the user for the current login calculation

    temp = [] # list to hold each row of the new dataframe

    # Fill empty dataframe

    for x in df.index: # Iterate through each row

        if df.at[x, "event"] == "enter" and df.at[x, "event details"] == "start": #if the current row is event enter and event detail start

            start = df.at[x, "time"]

            user = df.at[x, "user"]

        else: #Is a login event

            if start != None and user != None:

                if user == df.at[x, "user"]: #ensure that its the same user

                    # convert string to datetime

                    date\_time\_start = datetime.strptime(start, '%Y-%m-%d %H:%M:%S')

                    date\_time\_end = datetime.strptime(df.at[x, "time"], '%Y-%m-%d %H:%M:%S')

                    # obtain the total time taken in seconds

                    total\_time = (date\_time\_end - date\_time\_start).total\_seconds()

                    # create a row with data

                    temp.append((user, df.at[x, "event details"], total\_time))

                #Reset start time and current user

                start = None

                user = None

    # Create a dataframe and return it

    return pd.DataFrame(temp, columns = ["userid", "login result", "time"])

The function first initializes some variables:

* Start – used to identify the current login’s start time
* User – used to identify the current login’s user
* Temp[] – used to store each row of data for the new dataframe

For each row in the given dataframe, we perform 2 checks:

* If the current row’s event is ‘enter’ and event details is ‘start’, we know that the user has started the login process. So, we store the user’s start time and user id into ‘start’ and ‘user’
* Otherwise it’s a login event. First, we would check if the login event has started (in case of faulty data). Then we check to make sure it’s the same user as the one that started the event (once again, checking for faulty data). Knowing that there is no faulty data, we calculate the time taken for the login, and add a new row into ‘temp’. Outside the user check, we reset the variables ‘user’ and ‘start’ to none.

Once the for loop is done, we generate a dataframe from temp, specifying the headers as "userid", "login result", and "time".

Next function call is calculate\_stats\_df().

# Generate dataframe with mean value for success, and mean value for failure for each user

df\_image = calculate\_stats\_df(df\_image)

df\_text = calculate\_stats\_df(df\_text)

This function is used for generating a dataframe using pandas functions to perform some statistics on a given dataframe. These statistics will match our resulting csv’s requirements:

* Column with total number of logins
* Columns for number of success and failures logins
* Columns for total time taken to login for success and failure

def calculate\_stats\_df(df):

    #split the dataframe into two dataframes for success and failure

    df\_success = df[df["login result"] == "success"]

    df\_fail = df[df["login result"] == "failure"]

    #drop result columns

    df = df.drop(['login result'], axis = 1)

    df\_success = df\_success.drop(['login result'], axis = 1)

    df\_fail = df\_fail.drop(['login result'], axis = 1)

    #calculate number of logins (create dataframes)

    df\_total\_count = df.groupby(['userid'], as\_index=False).agg({"time":"count"}).rename(columns={"time" : "total logins"})    # total count

    df\_success\_count = df\_success.groupby(['userid'], as\_index=False).agg({"time":"count"}).rename(columns={"time" : "successful logins"}) #success count

    df\_fail\_count = df\_fail.groupby(['userid'], as\_index=False).agg({"time":"count"}).rename(columns={"time" : "unsuccessful logins"}) #failed count

    # mean of time (create dataframes)

    df\_success\_mean = df\_success.groupby(['userid'], as\_index=False).agg({"time":"mean"}).rename(columns={"time" : "avg login time success (s)"})

    df\_fail\_mean = df\_fail.groupby(['userid'], as\_index=False).agg({"time":"mean"}).rename(columns={"time" : "avg login time failed (s)"})

    # merge (merge left in case there are empty data)

    resulting\_df = df\_total\_count.merge(df\_success\_count, how='left', on = "userid").merge(df\_fail\_count, how='left', on = "userid").merge(df\_success\_mean, how='left', on = "userid").merge(df\_fail\_mean, how='left', on = "userid")

    #If there are any Nan values in the number of logins

    resulting\_df[["successful logins", "unsuccessful logins"]] = resulting\_df[["successful logins", "unsuccessful logins"]].fillna(value = 0)

    return resulting\_df

The function starts off by splitting the given dataframe based on the login result (either success or failure). Knowing what their login results are now, we drop it from the newly created dataframe and the given dataframe.

Next we count the number of logins. We apply the count on each dataframe, which creates a new dataframe where the ‘time’ column is overwritten with the new count column that we name:

* "total logins" for the given dataframe’s count
* “successful logins” for the dataframe with successful logins
* “unsuccessful logins” for the dataframe with uncessful logins

These new dataframes will be stored into df\_total\_count, df\_success\_count, and df\_fail\_count.

Now we calculate the mean of the time taken to login for successful and failure logins. Similar to count, it generates a new dataframe with a mean column that is named. The dataframes will be stored in df\_success\_mean and df\_fail\_mean.

Then we merge left on all dataframes together on the userid. Merge left in case that there are users that only failed login or only succeeded login.

In case there are NaN values from the merge, we fill each NaN in the count columns as 0. Then we return the resulting dataframe.

The next part is adding a column to identify the password scheme used.

#add column pwd scheme to the dataframes

#also pass the password scheme as the value for each row

df\_image.insert(1, "pwd scheme", scheme\_type\_1)

df\_text.insert(1, "pwd scheme", scheme\_type\_2)

This new column is added as the second column of the dataframes.

Next function call is join\_and\_sort().

# join the two dataframes

df\_result = join\_and\_sort(df\_image, df\_text)

This function will join the two dataframes together (image21 dataframe and text21 dataframe) to create the resulting dataframe required to generate the csv file.

def join\_and\_sort(df1, df2):

    #join the two dataframes

    df = pd.concat([df1, df2])

    #sort by userid

    return df.sort\_values(["userid"])

The function starts off by combining the two dataframes into one. Then it returns the combined dataframe after sorting by user id.

The final part of the script will convert the resulting dataframe into a csv file with its headers.

#Generate csv file

df\_result.to\_csv('combined.csv', index = False, header=True)