UCSF Data Scientist Challenge

Outline

The aim of this report is to analyse a dataset composed by the measurements of HIV status of all individuals in 32 communities in Kenya and Uganda. There are 4 time periods (t=0,1,2,3) where if a person attend a community health clinic (CHC), his HIV status is measured, if HIV positive, the viral load will be measured and he may be given antiretroviral therapy (ART). If a person does not attend the CHC, a tracking team will attempt to find the person and perform the same functions (test for HIV, measure viral load, possibly provide ART).

The paper opens with a summary of the analysis and outcome for each of the three points of the Data Challenge. A more detailed description, and the relative coding can be found in each dedicated section.

To perform the analysis Python 3.6 has been used with specific libraries for Data analysis and visualization, like: *pandas*, *numpy*, *matplotlib* and *seaborn*.

Summary

Question 1

- For each of the 32 communities in the data and for each of 4 time points, report the the proportion of patients who have an unsuppressed viral load.

To address question 1 and calculate the proportion of patients with an unsuppressed viral load (variable name: `prop_unsupp`), I created a collection of functions operating at different levels: community or patience. Below the list with a short description:

`read_input_files`: Upload each `community_t.csv` and `ViralLoads.csv` file in a *dictionary* of *pandas dataframes*.

`select_VL`: After the *merging* with viral load measurements per patient, for each community and time point:

1. find the *starting* (`chcdate`) and *ending* (`trdate`) dates of the measurements campaign;
2. select the correct `viral load` value according to associated `date`;

`calc_unsup`: Define the variable `unsupp` according to the `HIV` and `VL` values.

`save_results`: Calculate the `prop_unsupp` variable and save results in a pandas dataframe.

After a last data *reshaping* to fulfill the requested format, the final `unsupp.csv` file has been created with a loop over all dataframes inside the dictionary.

Here the code.

Question 2

- Pretending these were real data, are there any data quality problems in the dataset that the team would need to investigate? What decisions did you make in addressing these problems?

To address this point, I tried to understand a bit more in details the data. I wrote a function, `closer_look_at_community` to check the presence of possible data discrepacy and visualize general information like:

- the age of tested peoples according to their gender;
- how many peoples are HIV positive;
- the percentage of HIV positive that are treated (ART=1) with an antiretroviral therapy;

The analysis reveals that sometime there are some errors inputing the starting date. For example, for the *Bugamba* community at time *t=0* the starting date is first of Genuary of 1899. This could affect the selection of the correct `VL` measurment. A possible solution is identify and remove this lines. Anyway, we noticed that these lines have a `braceletid` that does not correspond to any `braceletid` in the `ViralLoad` dataframe. So merging the dataframe, as requested for following analysis steps, solve the problem.

Another point that draws my attention is the distribution of the viral load (VL) values. Values ranges from low, where the main part of distribution is, to sparse really high values. They could be outliers, due to errors during the measurement procedure, or really very high concentration of viral loads. Checking on litterature, I found that a few weeks after infection, HIV viral load increases to very high level (also many millions of copies/mL), then, as the immune system fights back, viral load usually drops to lower levels (50.000-10000) when HIV treatement is started. Finally, treatment should reduce viral load to less than 50 copies/mL within 3 months.

I also found a sample of 550 peoples having HIV negative, but for which the VL has been measured, resulting also in very high values. This sample could be voluntary introduced to check the realibility of HIV test, as a *control* sample. Count how many time an HIV negative patient presents a non minimal viral load numeber of copies (*false negative rate*).

Or, more simply, the presence of peoples HIV negative in a sample of only HIV positve could be an error. This sample could be rejected from the following analysis with a condition on HIV value.

Here more details about the analysis.

Question 3

- Suppose we changed our data simulation so that all patients who are HIV positive at time 1, 2, or 3 are immediately treated with antiretroviral therapy (ART). The data generating process is otherwise unchanged (including treatment at time 0). In the resulting data, what would be the total population proportion of patients with an unsuppressed viral load at time 3? Provide a single estimate and a 95% confidence interval. Very briefly describe your methodology.

The basic idea to answer the question is to use our data to calculate the effectiveness of the antiretroviral therapy (ART) at each time point, tracking what happens to excatly the same sample of patients. In other words, we are going to calculate the probability that the viral load keeps unsuppressed (unsupp=1) after one, two, and three time periods after the beginning of the ART (p_1 , p_2 , p_3). Then, we count ALL NEW patients with HIV positive at t=1, and t=2. Under the assumption that all these new patients will immediately start the ART, we multiply their counts by the opportune *probability* to estimate how many of them will have (or not) a viral load at t=3, i.e. after two and one time periods, respectively. Instead, for the patients that started the ART at t=0 we know from the data how many still have unsupp=1 at t=3.

The statistical approach is based on the properties of the Binomial distribution. Indeed, the *mean* of the distributon tells us the mean number of patients with an unsuppressed viral load (unsupp= 1) after each time points.

$$\mu = np$$

where *n* is the total population treated (ART=1) at a certain time point and *p* represents the ineffectiveness of the antiretroviral therapy.

Instead, the *standard deviation* gives us the uncentainty on the result.

$$\sigma = \sqrt{np(1-p)}$$

As first step, we calculate p_1 after one temporal step (t=0 -> t=1) tracking the history of peoples treated at t=0. We need to calculate how many still have a viral load (unsupp=1) at t=1. Similarly, checking exactly the same sample of patients at t=2 and t=3, we will know p_2 and p_3 after two (t=0 -> t=2) and three (t=0 -> t=3) steps, respectively.

As second step, *n* comes from real data at t=0, instead *n* at t=1, and t=2 would be ALL new `HIV` positive, according to the new simulated hipothesis.

The TOTAL average number of peoples with an unsuppressed viral load at t=3, will be the sum of the real number of patients sick at t=3 that have been treated at t=0, plus the *estimated means* at t=1 and t=2:

$$\mu_{tot} = realsicks_{t=0} + \mu_{t=1} + \mu_{t=2}$$

The final standard deviation at t=3 is calculated adding in quadrature the errors coming from our estimations at t=1 and t=2. The number of siks peoples coming from t=0 is not an estimation, but a *real* number, so it does not have an ussociated uncertainty.

$$\sigma_{tot} = \sqrt{\sigma_{t=1}^2 + \sigma_{t=2}^2}$$

In conclusion, we found that in the hypothesis that ALL `HIV` positive would be treated, the total popoluation proportion of patients with an unsuppressed viral load at time t=3 is 27.7%, with an uncertainty of 0.3% at 95% of confidence level.

More details about the analysis could clarify the procedure.

Challenge question 1

Here the code to address question 1 and create the file `unsupp.csv`.

In [2]:
```
# Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import glob
import math

def read_input_files(d):
    # Function to read input files and create a data frame dictionary
    import glob
    communities_list = glob.glob("*.csv")

    for file in communities_list:
        name = file.split('.')[0]
        d[name] = pd.read_csv(file)
    return d

def select_VL(df):
    # Function to select valid Viral Loads
    # Convert dates columns
    df.chcdate = pd.to_datetime(df.chcdate)
    df.trdate = pd.to_datetime(df.trdate)
    df.date = pd.to_datetime(df.date)

    # Find min and max dates
    chcstart = df.chcdate.min()
    trkend = df.trdate.max()

    # Select rows with date inside the range
    df[(df.date >= chcstart) & (df.date <= trkend)]

    # Caluculate the time difference with chcstart
    df['time_diff'] = df.date - chcstart

    # Order df so the minimum is the first of the duplicates
    df.sort_values(by='time_diff', ascending=True, inplace=True)

    # Drop working columns
    df.drop(['time_diff'], axis=1, inplace=True)

    # Finally, get only the first of the duplicates and output the result
    df.drop_duplicates(subset='searchid', keep='first', inplace=True)
    return df

def calc_unsup(row):
   # Function to calculate unsup
   return 1 if ((row['HIV'] == 1) & (row['VL'] > 500)) else 0


def save_results(count, name, df, df_res):
    ## Function to calculate the mean and save in a data frame
    # Calculate the mean
    prop_unsup = df['unsupp'].mean()
```

```
    # Populate the final df
    community = name.split('_')[0]
    time = name.split('_')[1]

    ## Save results in df_res
    df_res.loc[count, ['community']] = community
    df_res.loc[count, ['time']] = time
    df_res.loc[count, ['prop_unsup']] = prop_unsup

    return df_res

def reshaping_data(name, df, dfs_list):
    # Function to create a df with all info for all communities
    community = name.split('_')[0]
    time = name.split('_')[1]
    df.insert(0, 'community', community)
    df['time'] = time

    # Append dfs in a list
    dfs_list.append(df)

    return dfs_list

def organize_data(output_csv):
    # Common variables for input and output and further analysis
    d={}
    df_res = pd.DataFrame(columns=['community','time','prop_unsup'])
    count=0
    dfs_list = []
    peoples_list = []

    # Read input files
    read_input_files(d)

    for name, df in d.items():
        if(name != "ViralLoads"):
            ## Save patience IDS for statistical analysis
            peoples = d[name].searchid
            peoples_list.append(peoples)

            # Merge dfs
            df_merged = d[name].merge(d['ViralLoads'], on='braceletid')

            # Call the function to select valid VL
            select_VL(df_merged)

            ## Calculate unsupp
            df_merged['unsupp'] = df_merged.apply(calc_unsup, axis=1)

            ## Finally calculate the mean and save in df_res
            count+=1
            save_results(count, name, df_merged, df_res)

            ##Reshaping df for further analysis
            reshaping_data(name, df_merged, dfs_list)
```

```
    # Reshaping for the final output
    res = df_res.pivot(index='community', columns='time', values='prop_unsu
p')
    res.to_csv(output_csv)

    ## Concatenate dfs to further analysis
    dfs_list = pd.concat(dfs_list, axis=0)

    return dfs_list

df_all = organize_data('~/unsupp.csv')
```

Challenge question 2

General look at the community

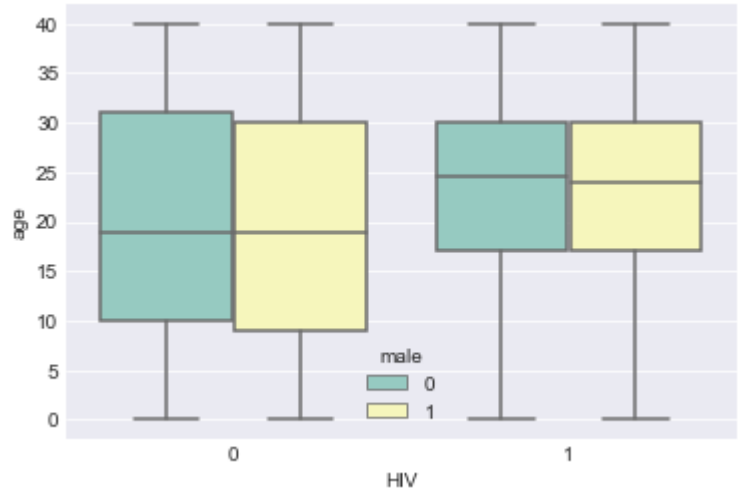
Here a function to make some plots on common variable like `HIV`, `ART`, `age` and `male`.

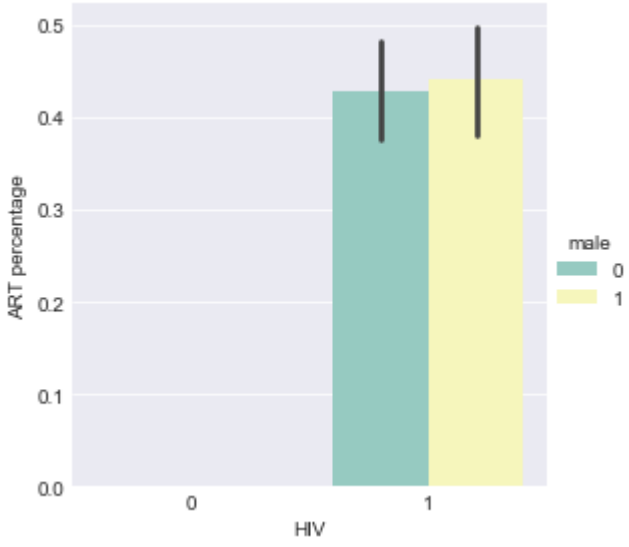
In [23]:
```
def closer_look_at_community(name, time):
    # Function to take a closer look at the single community data at a given t
ime
    file_name = '{}_{}.csv'.format(name, time)
    df = pd.read_csv(file_name)

    # Convert dates columns
    df.chcdate = pd.to_datetime(df.chcdate)
    df.trdate = pd.to_datetime(df.trdate)

    # Some plots to visualize
    sns.set_style("darkgrid")
    sns.boxplot(x="HIV", y="age", hue="male", data=df, palette="Set3")

    g = sns.factorplot(x="HIV", y="ART", hue="male", data=df, kind="bar", pale
tte="Set3")
    g.despine(left=True)
    g.set_ylabels("ART percentage")

    fig, axs = plt.subplots(ncols=2)
    sns.countplot(x="HIV", hue="male", data=df, palette="Set3", ax=axs[0])
    sns.countplot(x="ART", hue="male", data=df, palette="Set3", ax=axs[1])

    # Max and min dates
    chcstart = df.chcdate.min()
    trkend = df.trdate.max()
    print(chcstart, trkend)
    print(df.sort_values('chcdate', ascending=True).head(10))
    print(df.sort_values('trdate', ascending=False).head(10))

closer_look_at_community('Bugamba', '0')
```

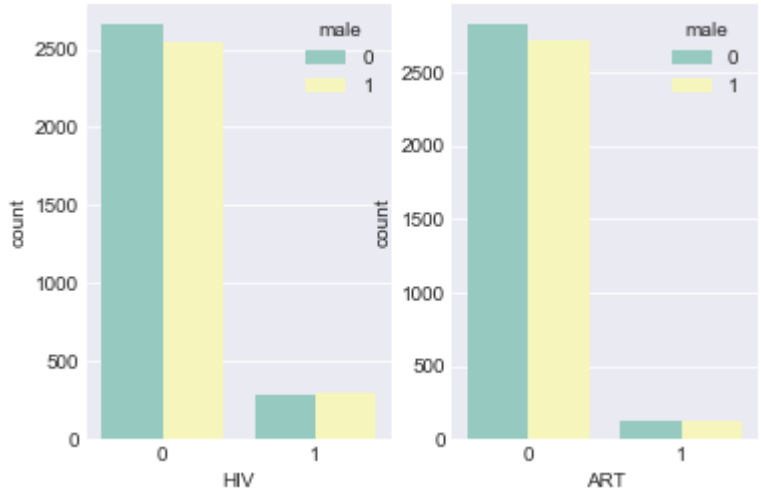
1899-01-01 00:00:00 2014-04-09 00:00:00

	searchid	HIV	ART	chcdate	trdate	braceletid	age	male
4937	251503	0	0	1899-01-01	NaT	2417075501	23	0
4759	108376	0	0	1899-01-01	NaT	2327645386	38	1
4100	566180	0	0	1899-01-01	NaT	2480463142	0	0
2305	807636	0	0	1899-01-01	NaT	2473049990	2	1
1036	423960	0	0	1899-01-01	NaT	2476522880	22	0
221	875594	0	0	1899-01-01	NaT	2394150199	1	0
2848	592343	0	0	1899-01-01	NaT	2542207058	37	1
592	593419	0	0	2013-12-06	NaT	2363652138	12	1
5594	578400	0	0	2013-12-06	NaT	2471548499	0	0
4486	233459	0	0	2013-12-06	NaT	2474992602	13	1

	searchid	HIV	ART	chcdate	trdate	braceletid	age	male
3971	402714	0	0	NaT	2014-04-09	2479547469	0	0
51	142846	0	0	NaT	2014-04-09	2354362437	22	1
228	956485	0	0	NaT	2014-04-09	2359038639	1	1
108	260608	0	0	NaT	2014-04-09	2424854859	12	1
32	535054	0	0	NaT	2014-04-08	2497649872	4	0
3924	865206	0	0	NaT	2014-04-08	2523842261	5	1
3708	127314	0	0	NaT	2014-04-08	2522768267	17	1
4326	694871	0	0	NaT	2014-04-08	2494712445	2	0
3310	806775	0	0	NaT	2014-04-08	2476889864	34	0
1087	310478	1	1	NaT	2014-04-07	2538580064	35	0







Wrong chcdate

As shown from the above lines, there are some mistakes in `chcdate`. There are six lines where `chcdate` is 1899-01-01. This is clearly an error, also looking at the other values of the variable and at the `trddate` max values.

A possible solution is to check if the `braceletid` corresponding to a wrong `chcdate` is inside the *ViralLoad* dataframe or not. If not, the simple *merging* of the two dataframe would solve the inconsistency.

In [243]:
```
#check if the wrong chcdate could be recovered by braceletid
df_viralload = pd.read_csv('ViralLoads.csv')
df_viralload.loc[df_viralload.braceletid == 2542207058]
```

Out[243]:

	braceletid	VL	date

In [244]:
```
# test a normal one
df_viralload.loc[df_viralload.braceletid == 2400879956]
```

Out[244]:

	braceletid	VL	date
23801	2400879956	633	2014-03-21

Closer look at viral load measurements

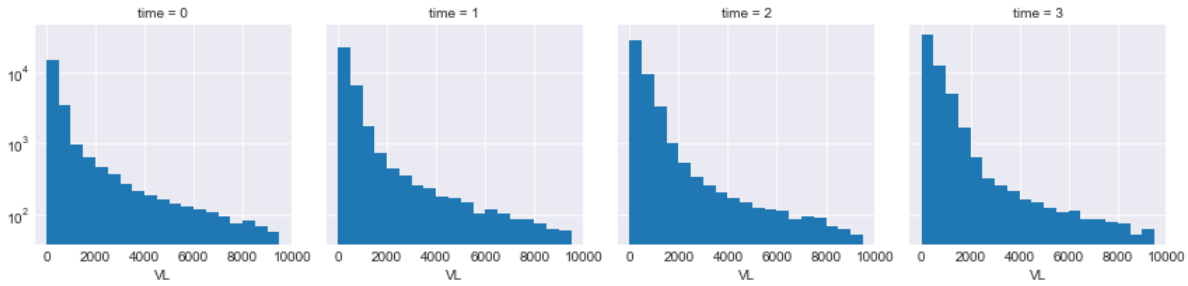
After merging the `community` and the `ViralLoad` dataframes it is possible to check the data consistency also about new variables. First of all, the general dataframe has the following structure:

In [5]:
```
# Taaking a look at the general dataframe with ALL communities and times
df_all.head()
```

Out[5]:

	ART	HIV	VL	age	braceletid	chcdate	community	date	male	searchid	time
561	0	1	149	31.0	2466813600	2013-12-07	Bugamba	2013-12-07	0.0	394219	0
206	0	1	486	11.0	2422054048	2013-12-12	Bugamba	2013-12-12	0.0	983549	0
91	0	1	316	7.0	2339738023	2013-12-12	Bugamba	2013-12-12	0.0	217101	0
255	0	1	40	19.0	2412370042	2013-12-13	Bugamba	2013-12-13	0.0	999292	0
488	1	1	40	29.0	2453161587	2013-12-13	Bugamba	2013-12-13	1.0	806060	0

In [51]:
```
# Check the VL values
sns.set_style("darkgrid")
g = sns.FacetGrid(df_all, col="time",)
g = g.map(plt.hist, "VL", log=True, bins=range(0, 10000, 500))
```

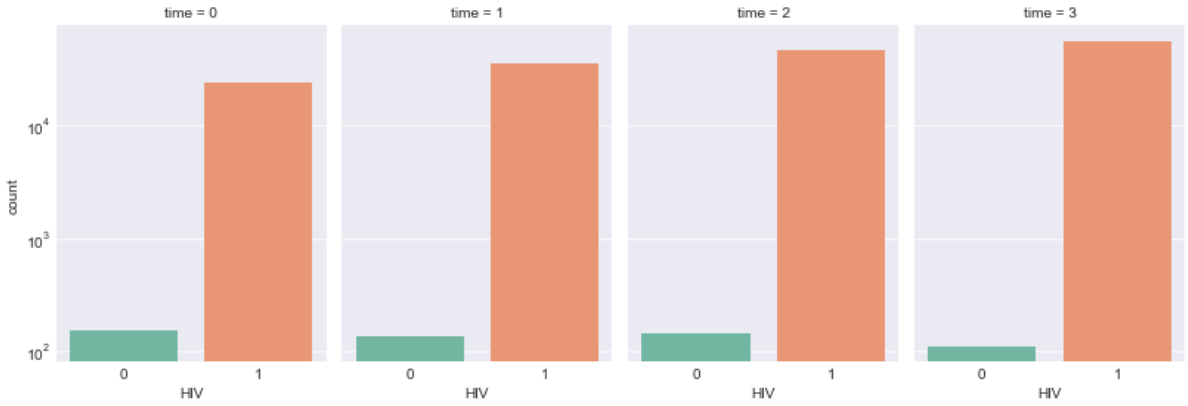


The VL distributions present a lot of values really high at each time. Those does not seems outliers, because the distribution is continuosly populated, but are those *possible* values or errors?

HIV negative, with a measured Viral Load

Check the number of total peoples HIV positive at different time.

In [25]:
```
# How many people with HIV=1 at different times
g = sns.factorplot(x="HIV", col="time", data=df_all, kind="count", palette="Se
t2", log="True", size=4, aspect=.7);
```



Values of `HIV = 0` are strange here. Because in principle, only peoples HIV positive need to be checked by viral load. We do not know if this is a sample of peoples introduced to study the HIV test *false negative* or simply an error. But let investigate.

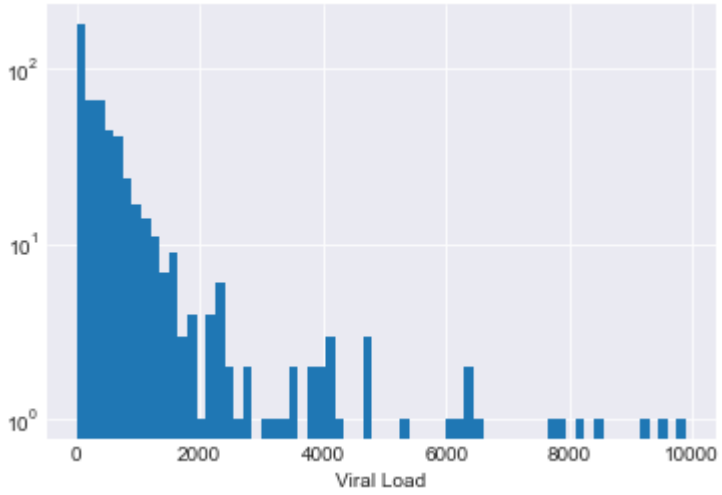
In [5]:
```
# NON sick? possible confusing data
total_nosicks = df_all[df_all.HIV == 0].count()['HIV']
nosicks_at_0 = df_all[(df_all.HIV == 0) & (df_all.time == '0')].count()['HIV']
nosicks_at_1 = df_all[(df_all.HIV == 0) & (df_all.time == '1')].count()['HIV']
nosicks_at_2 = df_all[(df_all.HIV == 0) & (df_all.time == '2')].count()['HIV']
nosicks_at_3 = df_all[(df_all.HIV == 0) & (df_all.time == '3')].count()['HIV']
print(total_nosicks, nosicks_at_0, nosicks_at_1, nosicks_at_2, nosicks_at_3)
```

```
550 156 137 146 111
```

We could check the values of viral loads (`VL`) for those peoples. If the `VL` distribution is not peaked at 40 (min value), the HIV positive/negative test could lose reliability.

In [80]:
```
# plot the VL value for HIV negative peoples
q = "HIV == 0"
df_all.query(q)['VL'].hist(log=True, bins=range(0, 10000,150)).set_xlabel("Viral Load")
```

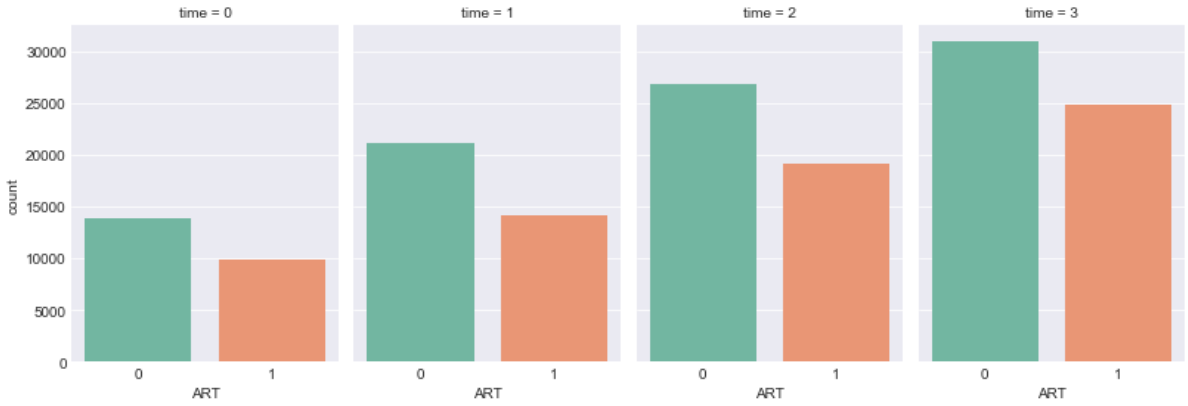
Out[80]: Text(0.5,0,'Viral Load')



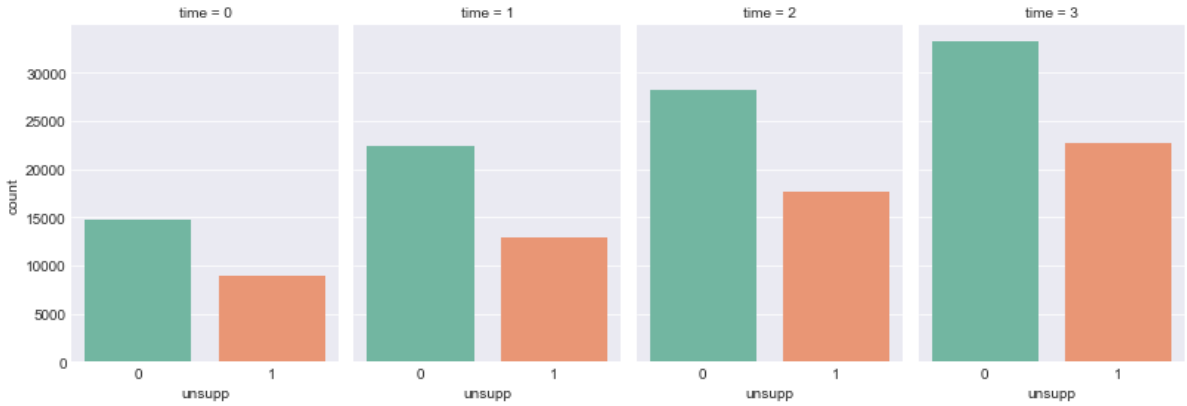
The above distribution shows a lot of very high VL values, not only the minimum value (40). This sample does not need to be removed because, according the unsupp definition independently from the VL value, HIV negative have unsupp 0.

To conclude, we could take a look also at how many peoples have been treated at different times and how many have an unsuppressed viral loads.

In [27]:
```
# How many people with ART=1 at different times
g = sns.factorplot(x="ART", col="time", data=df_all, kind="count", palette="Set2", size=4, aspect=.7);
```



In [28]:
```
# How many people with unsupp=1 at different times
g = sns.factorplot(x="unsupp", col="time", data=df_all, kind="count", palette=
"Set2", size=4, aspect=.7);
```



Challenge question 3

In order to address question 3, we need to find, first of all, the effectiveness of antiretroviral treatement (probability of success). I mean, we need do calculate how many patients still have `unsuppressed viral load` (`unsupp`= 1) after the treatment in each of different campaigns (`time` = 0,1,2,3).

In order to find what happens to treated patiens, first of all, we need to calculate the total number of people treated at t=0 and count them.

In [6]:
```
#Total population treated at t=0
total_treated_t0 = df_all[(df_all.ART == 1) & (df_all.time == '0')].count()['A
RT']
total_treated_t0
```

Out[6]:
```
9873
```

Now, if we count how many peoples are `HIV` positive at different `time`, we can calculate the percentage of patience that have been treated.

In [7]:
```
# Some numbers
total_sicks = df_all[df_all.HIV == 1].count()['HIV']
sicks_at_0 = df_all[(df_all.HIV == 1) & (df_all.time == '0')].count()['HIV']
sicks_at_1 = df_all[(df_all.HIV == 1) & (df_all.time == '1')].count()['HIV']
sicks_at_2 = df_all[(df_all.HIV == 1) & (df_all.time == '2')].count()['HIV']
sicks_at_3 = df_all[(df_all.HIV == 1) & (df_all.time == '3')].count()['HIV']
print(total_sicks, sicks_at_0, sicks_at_1, sicks_at_2, sicks_at_3)
```

```
160231 23550 35109 45793 55779
```

So, at t=0, the 42% (9873/23550) have been treated (ART=1).

Now, to investigate the effectiveness of the cure we need to trace what happens to the same patient at t=1, t=2 and t=3. Specifically. we are going to calculate the probsbility to keep having an unsuppressed viral load after one (from t=0 to t=1), two (from t=0 to t=2) or three temporal slots, since the beginning of the treatement.

In [8]:
```
# Count how many have unsupp=1 at t=1 in list_treated_t0
treated_t0 = df_all[(df_all.ART == 1) & (df_all.time == '0')]
list_treated_t0 = treated_t0['searchid'] ## check for unique()
sicks_t1 = df_all[(df_all.unsupp == 1) & (df_all.time == '1') & (df_all.search
id.isin(list_treated_t0))]
list_sicks_t1 = sicks_t1['searchid']
nosicks_t1 = df_all[(df_all.unsupp == 0) & (df_all.time == '1') & (df_all.sear
chid.isin(list_treated_t0))]
list_nosicks_t1 = nosicks_t1['searchid']
print(list_sicks_t1.count(), list_nosicks_t1.count())
```

```
4970 4897
```

In [9]:
```
#Same at t=2
sicks_t2 = df_all[(df_all.unsupp == 1) & (df_all.time == '2') & (df_all.search
id.isin(list_treated_t0))]
list_sicks_t2 = sicks_t2['searchid']
nosicks_t2 = df_all[(df_all.unsupp == 0) & (df_all.time == '2') & (df_all.sear
chid.isin(list_treated_t0))]
list_nosicks_t2 = nosicks_t2['searchid']
print(list_sicks_t2.count(), list_nosicks_t2.count())
```

```
4728 5139
```

In [10]:
```
# Same at t=3
sicks_t3 = df_all[(df_all.unsupp == 1) & (df_all.time == '3') & (df_all.search
id.isin(list_treated_t0))]
list_sicks_t3 = sicks_t3['searchid']
nosicks_t3 = df_all[(df_all.unsupp == 0) & (df_all.time == '3') & (df_all.sear
chid.isin(list_treated_t0))]
list_nosicks_t3 = nosicks_t3['searchid']
print(list_sicks_t3.count(), list_nosicks_t3.count())
```

```
4553 5316
```

In [11]:
```
# Calculate the probability of success
p_onestep = list_sicks_t1.count()/total_treated_t0
p_twostep = list_sicks_t2.count()/total_treated_t0
p_threestep = list_sicks_t3.count()/total_treated_t0
print(p_onestep,p_twostep,p_threestep)
```

```
0.5033930922718526 0.47888179884533577 0.46115668996252407
```

So, the percentage (probablity of successes) of people that still have an `unsuppressed viral load` is different after 1,2, or 3 temporal slots. Respectively: t=1: p_onestep = 50.3% t=2: p_twostep = 47.9% t=3: p_threestep = 46.1%

Now, we need to repeat the same analysis to understand what happens to the patients treated at t1, t2 and t3. And check how many are new patiens, because on these there is the difference between real data and new simulated.

In [12]:
```
#Total population treated at t=1 and story
total_treated_t1 = df_all[(df_all.ART == 1) & (df_all.time == '1')].count()['A
RT']
treated_t1 = df_all[(df_all.ART == 1) & (df_all.time == '1')]
list_treated_t1 = treated_t1['searchid']

######
sicks_t2_treated_t1 = df_all[(df_all.unsupp == 1) & (df_all.time == '2') & (df
_all.searchid.isin(list_treated_t1))]
list_sicks_t2_treated_t1 = sicks_t2_treated_t1['searchid']
nosicks_t2_treated_t1 = df_all[(df_all.unsupp == 0) & (df_all.time == '2') & (
df_all.searchid.isin(list_treated_t1))]
list_nosicks_t2_treated_t1 = nosicks_t2_treated_t1['searchid']
######
sicks_t3_treated_t1 = df_all[(df_all.unsupp == 1) & (df_all.time == '3') & (df
_all.searchid.isin(list_treated_t1))]
list_sicks_t3_treated_t1 = sicks_t3_treated_t1['searchid']
nosicks_t3_treated_t1 = df_all[(df_all.unsupp == 0) & (df_all.time == '3') & (
df_all.searchid.isin(list_treated_t1))]
list_nosicks_t3_treated_t1 = nosicks_t3_treated_t1['searchid']

print(list_treated_t1.count(),list_sicks_t2_treated_t1.count(),list_nosicks_t2
_treated_t1.count(),list_sicks_t3_treated_t1.count(), list_nosicks_t3_treated_
t1.count())
```

```
14114 5450 8654 5217 8885
```

To find the final number, we need to check if the peoples treated at t=1 are the same treated at t=0 or there are new cases. So, let compare the two lists of the `serachid`.

In [13]:
```
#check if list_treated_t0 and list_treated_t1 are separated or not
new_treated_t1= treated_t1.loc[~treated_t1['searchid'].isin(list_treated_t0)]
list_new_treated_t1 = new_treated_t1['searchid']
print(list_new_treated_t1.count())
```

```
4247
```

So, at t=1, all patience coming from t=0 are treated plus a percentage of new `HIV` cases. Here the difference between real case and new simulation. In the simulation ALL new `HIV` positive would be treated. So let calculate

In [14]:
```
# How many new cases of HIV = 1 at t1
new_sicks_t1 = sicks_at_1 - sicks_at_0
# supposing (new simulation) at t1 ALL are treated. At t3 in mean, would hav
e unsupp =1
mean_sick_t3_from_t1 = new_sicks_t1 * p_twostep

# How many new cases of HIV = 1 at t2
new_sicks_t2 = sicks_at_2 - sicks_at_1
# supposing (new simulation) at t2 ALL are treated. At t3 in mean, would hav
e unsupp =1
mean_sick_t3_from_t2 = new_sicks_t2 * p_onestep
print(new_sicks_t1, new_sicks_t2)
```

```
11559 10684
```

Now the final number of peoples with an `unsuppressed viral load` at t=3 in the ipothesis that ALL new HIV cases would be treated is the following sum:

In [15]:
```
# Total number of sick at t3
sicks_t3_treated_t0 = list_sicks_t3.count()
total_sick_t3 = sicks_t3_treated_t0 + mean_sick_t3_from_t1 + mean_sick_t3_from
_t2
print(total_sick_t3)
```

```
15466.64651068571
```

About the 95% of confidence level for the incertanity of this estimation, this 95% means 2 *sigma* and the sigma of the final distribution comes from the square root of square sum of single sigmas for two and one time steps.

In [16]:
```
# Calculate the single standard deviations
sigma_t1 = math.sqrt(mean_sick_t3_from_t1*(1-p_twostep))
sigma_t2 = math.sqrt(mean_sick_t3_from_t2*(1-p_onestep))

# the total one is
sigma_tot = math.sqrt(math.pow(sigma_t1, 2) + math.pow(sigma_t2, 2))

print(sigma_t1, sigma_t2, sigma_tot)
```

```
53.70842518118659 51.68052819297548 74.53503826891117
```

Now, knowing that for a two tails test, the 95% of confidence level corresponds to 1.96 standard deviations from the mean, our confidence interval is:

In [23]:
```
# Calculate the confidence level
confidende_level_95 = 1.96*sigma_tot
confidende_level_95
```

Out[23]:
```
146.0886750070659
```

So the final number of patiens thath would have an unsuppressed viral load at t=3 at 95% of confidence level, in case ALL patieces would treated at t=1, t=2 or t=3 is 15467 +- 146.

Finally, knowing that the total number of peoples `HIV` positive at `time`=3 is `sicks_at_3`:

```
In [18]: sicks_at_3
```

```
Out[18]: 55779
```

the total popoluation proportion of patients with an unsuppressed viral load at time 3, would be:

```
In [19]: total_prop_unsupp = total_sick_t3/sicks_at_3
         total_prop_unsupp
```

```
Out[19]: 0.27728439933820453
```

with an uncertainty of:

```
In [24]: confidende_level_95/sicks_at_3
```

```
Out[24]: 0.002619062281630468
```

On the contrary, in the real case where approximatetly only the 40% of HIV positive population is treated with antiretroviral therapy at each time, the total proportion of peoples having an unsuppressed viral load at the end of the campaigns, is:

```
In [25]: real_total_prop_unsupp = df_all[(df_all.HIV == 1) & (df_all.time == '3')]['uns
         upp'].mean()
         real_total_prop_unsupp
```

```
Out[25]: 0.4060667993330823
```