Day4_Mapping_Exercice2_CORRECTED

September 10, 2024

0.1 Get some basic mapping stats with samtools flagstat

As we will mainly launch python code to analyse mapping results, the kernel of this jupyter book is Python3. So, to run linux commands, we have to add : * `%` to execute a single linux command * `%%bash` to execute a serie of linux commands

0.1.1 Merge individual flagstat files into an unique file `all_stat.csv` with python code

- Add the path of the directory that contains flagstat files in the following cell (line 8) `flagstat_dir = PATH_2_FLAGSTAT_DIRECTORY`
- Then, execute the two following cells

[2]:
```
# IMPORT PYTHON PACKAGE USED BY THE CODE
import os
import pandas as pd

# VARIABLE INITIALIZATION

## NAME OF THE DIRECTORY THAT CONTAINS FLAGSTAT FILES
flagstat_dir =  "/home/jovyan/SV_DATA/RESULTS/2_bam_flagstat/"

## NAME OF THE FILE THAT WILL CONTAIN ALL THE FLAGSTAT RESULTATS
stat_file = f"{flagstat_dir}/all_stat.csv"

# PRINT THE CONTENT OF 2 PREVIOUS VARIABLES INITIALIZED
print("DIRECTORY : ",flagstat_dir)
print("FINAL STAT FILE : ",stat_file)
```

```
DIRECTORY :  /home/jovyan/SV_DATA/RESULTS/2_bam_flagstat/
FINAL STAT FILE :  /home/jovyan/SV_DATA/RESULTS/2_bam_flagstat//all_stat.csv
```

[3]:
```
# OPEN THE FINAL FILE IN WHICH WE PRINT SOME STATS EXTRACTED
# FROM EACH INDIVIDUAL FILE GENERATED BY SAMTOOLS FLAGSTAT
with open(stat_file, 'w') as stat:
    # WRITE A HEADER LINE IN OUR STAT FILE
    header_line = "sample,mapped,paired,unmapped"
    stat.write(header_line)

    # READING EACH FILE OF THE FLGSTAT DIRECTORY
```

```
    for file in os.listdir(flagstat_dir):

        filen = flagstat_dir + "/" + file

        # If the word "flagstat" is in name of file
        if "flagstat" in file:

            # Extract sample name and save into a new variable newLine
            new_line = f"\n{file.split('.')[0]},"

            # OPEN AND READS FLAGSTAT FILE
            with open(filen, "r") as flagstat:

                # read file line by line
                for line in flagstat:
                    # remove the line skipper at the endo of the line
                    line = line.rstrip()
                    # Keep only line mapped, paired or singleton word
                    if ('mapped (' in line and not 'primary' in line) or␣
↪'paired (' in line or 'singleton' in line:
                        # get percentage value and save it into the varaible␣
↪called perc
                        perc = f"{line.split('(')[1].split('%')[0]}"
                        new_line += f"{perc},"
                # WRITE THE LINE ONCE THE FLAGSTAT FILE COMPLETELY READ
                stat.write(new_line.strip(","))
```

Display the content of the final stat file

[4]:
```
%ls -lt /home/jovyan/SV_DATA/RESULTS/2_bam_flagstat/ | head
```

```
total 76
-rw-r--r-- 1 jovyan users 425 Sep 10 14:20 all_stat.csv
-rw-r--r-- 1 jovyan users 525 Sep 10 13:13 H44.flagstat
-rw-r--r-- 1 jovyan users 519 Sep 10 13:10 G11.flagstat
-rw-r--r-- 1 jovyan users 525 Sep 10 13:09 E318.flagstat
-rw-r--r-- 1 jovyan users 521 Sep 10 13:06 E2.flagstat
-rw-r--r-- 1 jovyan users 524 Sep 10 13:05 D119.flagstat
-rw-r--r-- 1 jovyan users 521 Sep 10 13:03 C2.flagstat
-rw-r--r-- 1 jovyan users 524 Sep 10 13:02 C218.flagstat
-rw-r--r-- 1 jovyan users 521 Sep 10 12:59 B8.flagstat
```

[5]:
```
%cat $stat_file
```

```
sample,mapped,paired,unmapped
1613,96.40,94.64,0.31
4752,71.15,69.31,0.33
716,38.18,35.74,0.72
```

```
B1,95.36,93.96,0.26
5417,86.85,85.02,0.39
D119,89.54,86.60,1.01
C218,69.57,67.39,0.77
685,70.24,68.50,0.34
G11,86.77,85.72,0.19
H44,91.20,88.24,0.93
C2,92.51,91.08,0.30
1868,91.04,89.97,0.14
A8,92.51,91.08,0.40
B8,91.88,90.48,0.34
B218,N/A : N/A),N/A : N/A),N/A : N/A)
E2,95.19,93.71,0.36
A1,87.66,86.25,0.45
E318,81.34,78.60,0.94
```

Plot mapping rate per sample Load csv file into a panda dataframe

[6]:
```
df_bam_stat = pd.read_csv(stat_file, index_col=False, sep=",")
df_bam_stat
```

[6]:

	sample	mapped	paired	unmapped
0	1613	96.40	94.64	0.31
1	4752	71.15	69.31	0.33
2	716	38.18	35.74	0.72
3	B1	95.36	93.96	0.26
4	5417	86.85	85.02	0.39
5	D119	89.54	86.60	1.01
6	C218	69.57	67.39	0.77
7	685	70.24	68.50	0.34
8	G11	86.77	85.72	0.19
9	H44	91.20	88.24	0.93
10	C2	92.51	91.08	0.30
11	1868	91.04	89.97	0.14
12	A8	92.51	91.08	0.40
13	B8	91.88	90.48	0.34
14	B218	N/A : N/A)	N/A : N/A)	N/A : N/A)
15	E2	95.19	93.71	0.36
16	A1	87.66	86.25	0.45
17	E318	81.34	78.60	0.94

Basic stats

[7]:
```
# only print the values of the "mapped" column
print(df_bam_stat['mapped'])
```

```
0          96.40
1          71.15
```

```
2          38.18
3          95.36
4          86.85
5          89.54
6          69.57
7          70.24
8          86.77
9          91.20
10         92.51
11         91.04
12         92.51
13         91.88
14    N/A : N/A)
15         95.19
16         87.66
17         81.34
Name: mapped, dtype: object
```

[8]:
```python
# Display only the minimum, maximum and average rate of mapped reads
minimum = df_bam_stat["mapped"].min()
maximun = df_bam_stat["mapped"].max()
mean_flag = df_bam_stat["mapped"].mean()

print("\n######## BASIC STATS\n MAPPED")
print(f"\t%min : {minimum}\t %max : {maximun}\t %mean : {mean_flag}")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[8], line 4
      2 minimum = df_bam_stat["mapped"].min()
      3 maximun = df_bam_stat["mapped"].max()
----> 4 mean_flag = df_bam_stat["mapped"].mean()
      6 print("\n######## BASIC STATS\n MAPPED")
      7 print(f"\t%min : {minimum}\t %max : {maximun}\t %mean : {mean_flag}")

File /opt/conda/lib/python3.11/site-packages/pandas/core/series.py:6226, in␣
 ↪Series.mean(self, axis, skipna, numeric_only, **kwargs)
   6218 @doc(make_doc("mean", ndim=1))
   6219 def mean(
   6220     self,
   (…)
   6224     **kwargs,
   6225 ):
-> 6226     return NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)

File /opt/conda/lib/python3.11/site-packages/pandas/core/generic.py:11969, in␣
 ↪NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)
```

```
  11962 def mean(
  11963     self,
  11964     axis: Axis | None = 0,
   (…)
  11967     **kwargs,
  11968 ) -> Series | float:
> 11969     return self._stat_function(
  11970         "mean", nanops.nanmean, axis, skipna, numeric_only, **kwargs
  11971     )

File /opt/conda/lib/python3.11/site-packages/pandas/core/generic.py:11926, in␣
 ↪NDFrame._stat_function(self, name, func, axis, skipna, numeric_only, **kwargs
  11922 nv.validate_func(name, (), kwargs)
  11924 validate_bool_kwarg(skipna, "skipna", none_allowed=False)
> 11926 return self._reduce(
  11927     func, name=name, axis=axis, skipna=skipna, numeric_only=numeric_onl
  11928 )

File /opt/conda/lib/python3.11/site-packages/pandas/core/series.py:6134, in␣
 ↪Series._reduce(self, op, name, axis, skipna, numeric_only, filter_type, **kwd
   6129     # GH#47500 - change to TypeError to match other methods
   6130     raise TypeError(
   6131         f"Series.{name} does not allow {kwd_name}={numeric_only} "
   6132         "with non-numeric dtypes."
   6133     )
-> 6134 return op(delegate, skipna=skipna, **kwds)

File /opt/conda/lib/python3.11/site-packages/pandas/core/nanops.py:147, in␣
 ↪bottleneck_switch.__call__.<locals>.f(values, axis, skipna, **kwds)
    145         result = alt(values, axis=axis, skipna=skipna, **kwds)
    146 else:
--> 147     result = alt(values, axis=axis, skipna=skipna, **kwds)
    149 return result

File /opt/conda/lib/python3.11/site-packages/pandas/core/nanops.py:404, in␣
 ↪_datetimelike_compat.<locals>.new_func(values, axis, skipna, mask, **kwargs)
    401 if datetimelike and mask is None:
    402     mask = isna(values)
--> 404 result = func(values, axis=axis, skipna=skipna, mask=mask, **kwargs)
    406 if datetimelike:
    407     result = _wrap_results(result, orig_values.dtype, fill_value=iNaT)

File /opt/conda/lib/python3.11/site-packages/pandas/core/nanops.py:720, in␣
 ↪nanmean(values, axis, skipna, mask)
    718 count = _get_counts(values.shape, mask, axis, dtype=dtype_count)
    719 the_sum = values.sum(axis, dtype=dtype_sum)
--> 720 the_sum = _ensure_numeric(the_sum)
    722 if axis is not None and getattr(the_sum, "ndim", False):
```

```
    723     count = cast(np.ndarray, count)

File /opt/conda/lib/python3.11/site-packages/pandas/core/nanops.py:1693, in␣
 ↪_ensure_numeric(x)
   1690 elif not (is_float(x) or is_integer(x) or is_complex(x)):
   1691     if isinstance(x, str):
   1692         # GH#44008, GH#36703 avoid casting e.g. strings to numeric
-> 1693         raise TypeError(f"Could not convert string '{x}' to numeric")
   1694     try:
   1695         x = float(x)

TypeError: Could not convert string '96.4071.1538.1895.3686.8589.5469.5770.2486
 ↪7791.2092.5191.0492.5191.88N/A : N/A)95.1987.6681.34' to numeric
```

[15]:
```
df_bam_stat.drop([14], inplace=True)
df_bam_stat
```

[15]:

	sample	mapped	paired	unmapped
0	1613	96.40	94.64	0.31
1	4752	71.15	69.31	0.33
2	716	38.18	35.74	0.72
3	B1	95.36	93.96	0.26
4	5417	86.85	85.02	0.39
5	D119	89.54	86.60	1.01
6	C218	69.57	67.39	0.77
7	685	70.24	68.50	0.34
8	G11	86.77	85.72	0.19
9	H44	91.20	88.24	0.93
10	C2	92.51	91.08	0.30
11	1868	91.04	89.97	0.14
12	A8	92.51	91.08	0.40
13	B8	91.88	90.48	0.34
15	E2	95.19	93.71	0.36
16	A1	87.66	86.25	0.45
17	E318	81.34	78.60	0.94

Sort by sample name

[16]:
```
df_bam_stat_sorted=df_bam_stat.sort_values(by=['sample'])
df_bam_stat_sorted
```

[16]:

	sample	mapped	paired	unmapped
0	1613	96.40	94.64	0.31
11	1868	91.04	89.97	0.14
1	4752	71.15	69.31	0.33
4	5417	86.85	85.02	0.39
7	685	70.24	68.50	0.34

2	716	38.18	35.74	0.72
16	A1	87.66	86.25	0.45
12	A8	92.51	91.08	0.40
3	B1	95.36	93.96	0.26
13	B8	91.88	90.48	0.34
10	C2	92.51	91.08	0.30
6	C218	69.57	67.39	0.77
5	D119	89.54	86.60	1.01
15	E2	95.19	93.71	0.36
17	E318	81.34	78.60	0.94
8	G11	86.77	85.72	0.19
9	H44	91.20	88.24	0.93

Mapping Rate Plots with Python (Seaborn)

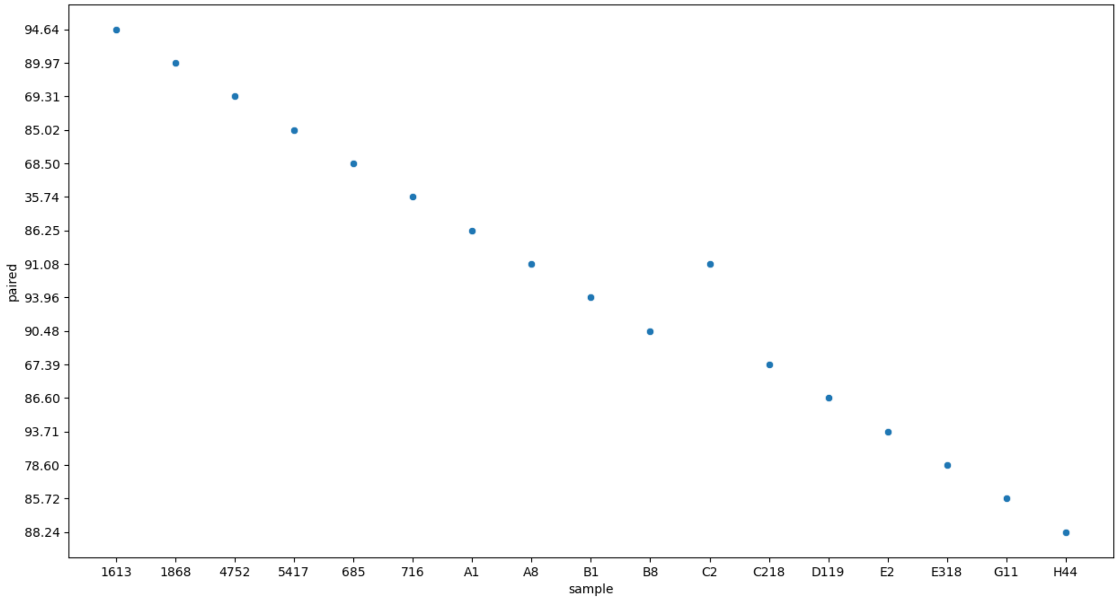
- Plot the mapping rate (Y-axis) for each sample (X-axis) using the `mapped`and the `sample` columns of the `df_bam_sorted`dataframe

[17]:
```
# Plot with seaborn
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize = (15,8))
sns.scatterplot(x="sample",y="paired", data=df_bam_stat_sorted)
```

```
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

[17]:
```
<Axes: xlabel='sample', ylabel='paired'>
```



- Plot the values of the three columns (mapped, paired, unmapped) for each sample

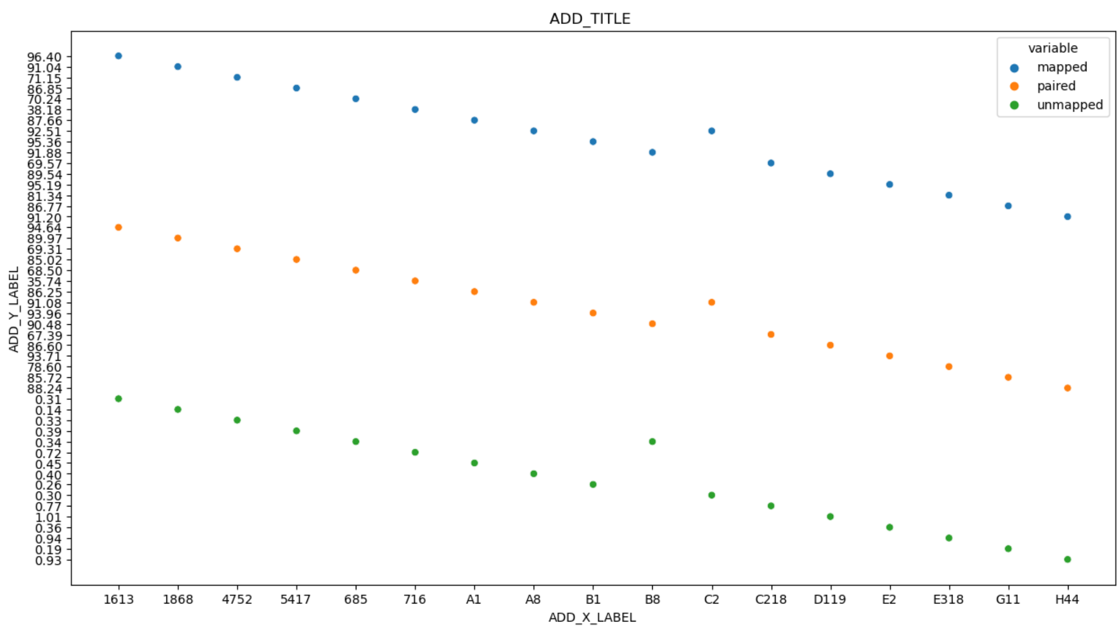
[18]:
```
# Plot with seaborn
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize = (15,8))
ax=sns.scatterplot(x="sample",y="value", hue='variable', data=pd.
 ↪melt(df_bam_stat_sorted, 'sample'))
ax.set_title("ADD_TITLE ")
ax.set_xlabel("ADD_X_LABEL")
ax.set_ylabel("ADD_Y_LABEL")
```

```
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
```

```
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

[18]: Text(0, 0.5, 'ADD_Y_LABEL')



[ ]: