

CA4

November 20, 2017

1 CA4

1.1 Introduction

The fourth assignment is based on transforming a large dataset into text format (over 5000 lines of text)

Dataset consist of the GitHub updates by different users.

I scrubed (cleaned) the data and placed it into the relevant objects.

After scrubbing dataset I gained 422 different sets of commit objects.

In this report I will analyse 422 commit objects that are stored in the list by using Python and will come up with interesting statistics.

My code for calculating the analysis will be documented and tested.

1.2 Python script preparation

```
In [1]: #set a directory
import os
os.chdir('D://DBS//5.Programming_for_big_data//github//pz_pbd//CA4')

#create a function which would read file
def read_file(changes_file):
    # use strip to strip out spaces and trim the line
    #strip each line in the inserted file, and
    #return after for loop have checked all file
    return [line.strip() for line in open(changes_file, 'r')]

In [2]: #create commit class
class Commit (object):

    #create commit instance variables
    def __init__ (self, revision, author, date, time, \
        number_of_lines, change_path = '', comment = ''):
        self.revision = revision
        self.author = author
        self.date = date
        self.time = time
        self.number_of_lines = number_of_lines
        self.change_path = change_path
```

```

        self.comment = comment

#create instance which return commits as a dictionary
def to_dic(self):
    return {'revision': self.revision, \
            'author': self.author, 'date': self.date, \
            'time': self.time, \
            'number of lines': self.number_of_lines, \
            'changed path': self.change_path, 'comment': self.comment}

#return instance variable
#return number of lines as a string, and join comments
#the reason why we need to join comments is
#because we can have multiple lines comments
def __repr__(self):
    return self.revision+ ',' + self.author + ',' + self.date + \
    ',' + self.time + ',' + str(self.number_of_lines) + ',' \
    + ' '.join(self.change_path) + ',' + ' '.join(self.comment) + '\n'

```

In [3]: `import pandas as pd`

```

#create function which returns data frame of commits
def to_df(data):
    commits = []
    for commit in data:
        commits.append(commit.to_dic())
    return pd.DataFrame(commits)

#get commits function will help us to break information from each commit
def get_commits(data):
    #each commit is separated by 72 hifins
    sep = 72*'- '
    #create a list to store each commit
    commits = []
    #first index starts in the line 0
    index = 0
    #while our index is less than data lenght
    while index < len(data):
        try:
            # find each line of the commit
            #and split by the separator /
            #index + 1 because line starts one line lower than index line
            details = data[index + 1].split('|')
            #call class commit and add details to the variable
            #commit as per class
            commit = Commit(details[0].strip(),
                             details[1].strip(),
                             details[2].strip().split(' ')[0],

```

```

        details[2].strip().split(' ')[1],
        #get number of lines and return as int
        int(details[3].strip().split(' ')[0]))
    #set file end index
    #build in function looks for empty line index and
    #return that index (use build in index function)
    #look for empty lines and stop looking once found
    change_file_end_index = data.index ('', index + 1)
    #changed path line starts from the third line
    commit.change_path = data[index +3 : change_file_end_index]
    #get comments
    commit.comment = data [change_file_end_index + 1:
        change_file_end_index + 1 + commit.number_of_lines]
    # add details to the list of commits
    commits.append(commit)
    #find for a next 72 hifins
    #look for next index in the file which starts with
    #72 hifins, and update index position
    index = data.index(sep, index + 1)
except IndexError:
    #then index equal to len of data program stops
    index = len(data)
#return all commits
return commits

```

1.2.1 Test Functions

To make sure that my functions work, I am going to test each of them separately.

Going to check:

- Length of the dataset
- Length of commit objects
- Who is the author at index 420 position in commits list
- Revision number at index 0 position in commits list
- Comments at index 23 position in commits list
- Changed paths at index 40 position in commits list
- Who is the author at index 9 position in the the data frame

```

In [4]: if __name__ == '__main__':
        #open the file - and read all lines.
        changes_file = 'changes_python.log'
        #assign all lines to data variable
        data = read_file(changes_file)
        #get commits from get_commits function and insert data variable
        commits = get_commits(data)
        #assign commits, to the data_frame variable by using to_df function
        data_frame = to_df(commits)

```

```
In [5]:      #print number of lines from the dataset
          print(len(data))
```

5255

```
In [6]:      #print length of commit objects
          #from this commits variable I will start forming my dataframe
          print(len(commits))
```

422

```
In [7]:      #print author at the 420 index position from the commit list
          print(commits[420].author)
```

Jimmy

```
In [8]:      #print revision at the 0 index position from the commit list
          print(commits[0].revision)
```

r1551925

```
In [9]:      #print comment at the 23 index position from the commit list
          print (commits[23].comment)
```

['SFR-108 : Create bilingual French/English translated Android application for SFR', '-----']

```
In [10]:     #print changed paths at the 40 index position from the commit list
          print (commits[40].change_path)
```

['A /cloud/personal/client-international/android/branches/android-15.2-solutions/libs/ui/res/1']

```
In [11]:     #print author at the 9 index position from the data frame
          print (data_frame['author'][9])
```

Vincent

1.3 Analysis by using Python

I will analyse dataset by using Python language.

To find interesting statistics in 422 commit objects I will use following libraries:

- Pandas
- Numpy

- Matplotlib (visualization)
- Seaborn (visualization)

In this report I will explain step by step how I got my results.

```
In [12]:      #import required libraries
            import pandas as pd
            import numpy as np
            import matplotlib.pyplot as plt
            %matplotlib inline
            #Seaborn is a Python visualization library based on matplotlib.
            #It provides a high-level interface for drawing attractive statistical graphics.
            import seaborn as sns
            plt.style.use('fivethirtyeight')
            import warnings
            warnings.filterwarnings('ignore')
```

Once I have imported all required libraries for my analysis I can start to analyse and explore data.

Build in function `.info()` shows that data frame has 422 entries and display all 7 column names. Below we can see that data frame doesn't have any missing values.

```
In [13]:      #display information about a dataset
            data_frame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 422 entries, 0 to 421
Data columns (total 7 columns):
author          422 non-null object
changed path    422 non-null object
comment         422 non-null object
date            422 non-null object
number of lines 422 non-null int64
revision        422 non-null object
time            422 non-null object
dtypes: int64(1), object(6)
memory usage: 23.1+ KB
```

To get more familiar with a dataset I will view first 5 lines by using build in function `.head()`

```
In [14]:      #display data frame 5 lines
            data_frame.head()
```

```
Out[14]:      author \
0          Thomas
1          Thomas
2        Vincent
3          Thomas
```

```

4 /OU=Domain Control Validated/CN=svn.company.net

                                changed path \
0 [A /cloud/personal/client-international/androi...
1 [M /cloud/personal/client-international/androi...
2 [M /cloud/personal/client-international/androi...
3 [M /cloud/personal/client-international/androi...
4 [M /cloud/personal/client-international/androi...

                                comment          date \
0 [Renamed folder to the correct name] 2015-11-27
1 [Removed unused webview.plan.management and we... 2015-11-27
2 [enable all clients] 2015-11-27
3 [Chnaged jira url to https] 2015-11-27
4 [[gradle-release] prepare for next development... 2015-11-27

number of lines revision      time
0              1 r1551925 16:57:44
1              1 r1551575 09:46:32
2              1 r1551569 09:38:09
3              1 r1551558 09:13:26
4              1 r1551504 07:05:41

```

Now once data frame is prepared for analysis I am going to check for unique futures within the data frame.

First off all I will investigate, how many unique authors are in the dataset by using **unique()** function.

```

In [15]: #print unique authors
         data_frame['author'].unique()

Out[15]: array(['Thomas', 'Vincent',
                '/OU=Domain Control Validated/CN=svn.company.net', 'Jimmy',
                'Freddie', 'Dave', 'ajon0002', 'murari.krishnan', 'Nicky', 'Alan'], dtype=object)

```

Once the list with unique authors are known, I can see that one of the authors name is **/OU=Domain Control Validated/CN=svn.company.net** and I am going to replace this name into **DCV** as a very long name can cause difficulties for further data exploration and visualisation.

```

In [16]: #replace author /OU=Domain Control Validated/CN=svn.company.net into DCV
         #this creates new series of values, so we need to
         #assign this new column to the correct column
         data_frame['author'] = data_frame['author']. \
         replace(['/OU=Domain Control Validated/CN=svn.company.net'], 'DCV')

In [17]: #print unique authors
         data_frame['author'].unique()

Out[17]: array(['Thomas', 'Vincent', 'DCV', 'Jimmy', 'Freddie', 'Dave', 'ajon0002',
                'murari.krishnan', 'Nicky', 'Alan'], dtype=object)

```

1.3.1 The biggest amount of commit objects by Author

Once I have a list of authors ready, I am going to investigate which author was the most active and had created the biggest number of commit objects in the dataset.

By using matplotlib and seaborn I will display authors who have generated the biggest amount of commit objects.

```
In [18]: #count how many different authors have commit objects and group by author
log_in_per = data_frame[['author' , 'date']].groupby(['author'], \
                                                    as_index = False).count()

#rename column names and print
log_in_per = log_in_per.rename(columns = {'author': 'Author', \
                                          'date': 'Commit count'})

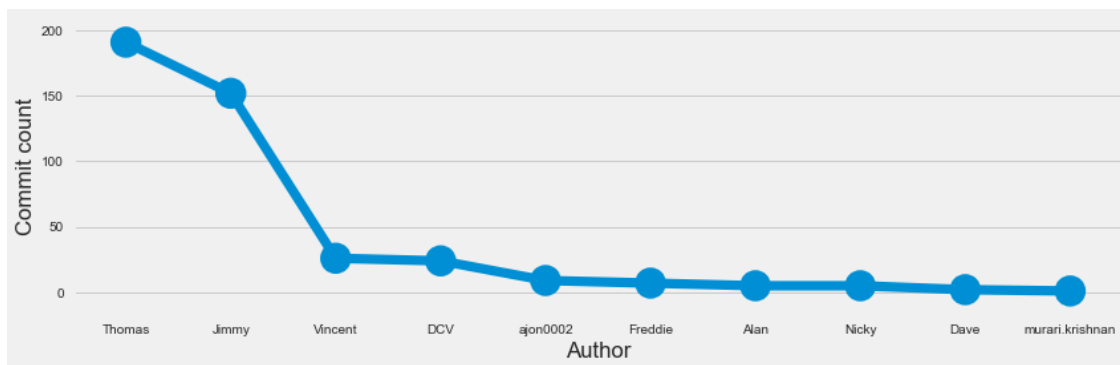
#sort values
log_in_per = log_in_per.sort_values(['Commit count'], ascending = [False])
log_in_per
```

```
Out[18]:
```

| | Author | Commit count |
|---|-----------------|--------------|
| 6 | Thomas | 191 |
| 4 | Jimmy | 152 |
| 7 | Vincent | 26 |
| 1 | DCV | 24 |
| 8 | ajon0002 | 9 |
| 3 | Freddie | 7 |
| 0 | Alan | 5 |
| 5 | Nicky | 5 |
| 2 | Dave | 2 |
| 9 | murari.krishnan | 1 |

```
In [19]: #create a factor plot
sns.factorplot('Author', 'Commit count', data = log_in_per, size = 4, \
              aspect = 3)
```

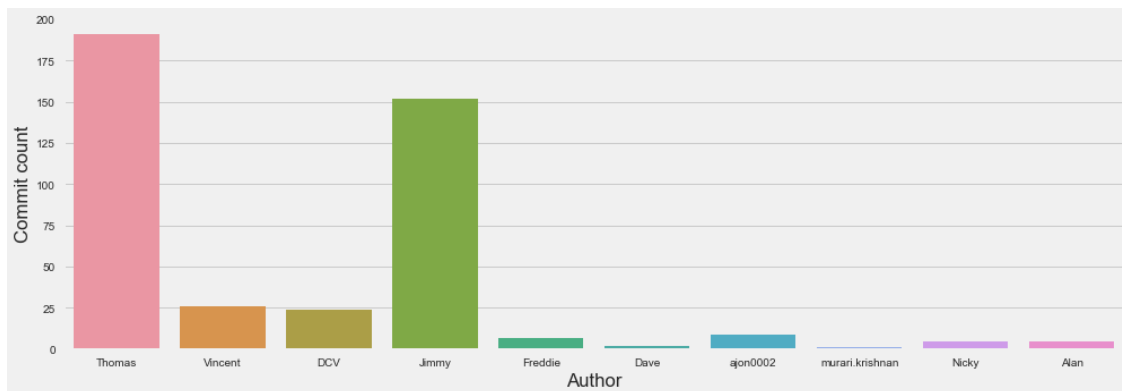
```
Out[19]: <seaborn.axisgrid.FacetGrid at 0x9e724a8>
```



```
In [20]: #create an axis for a plot
fig, (axis1) = plt.subplots(1,1,figsize=(15,5))

#display as a bar chart
ax = sns.barplot('Author', 'Commit count', data = log_in_per, order = \
                ['Thomas', 'Vincent', \
                 'DCV', 'Jimmy','Freddie', 'Dave', 'ajon0002', \
                 'murari.krishnan', 'Nicky', 'Alan'], ax = axis1)
ax.set (xlabel = 'Author', ylabel = 'Commit count')
plt.show
```

Out[20]: <function matplotlib.pyplot.show>



Once data visualisation are completed, we can see that author Thomas has the biggest amount of commit objects (191) and the smallest number of commit objects (1) belongs to author murari.krishnan.

To better understand data I created pivot table to display percentage of commit objects belonging to each author and will display visualisation in the pie chart.

```
In [21]: #display overall percentage of commits by author
commits = int(data_frame['date'].count())
#add new column to the data frame, and calculate percentage
log_in_per['Percentage'] = pd.Series.round \
                        (((log_in_per['Commit count']*100)/commits),2 )
log_in_per
```

```
Out[21]:
```

| | Author | Commit count | Percentage |
|---|----------|--------------|------------|
| 6 | Thomas | 191 | 45.26 |
| 4 | Jimmy | 152 | 36.02 |
| 7 | Vincent | 26 | 6.16 |
| 1 | DCV | 24 | 5.69 |
| 8 | ajon0002 | 9 | 2.13 |
| 3 | Freddie | 7 | 1.66 |
| 0 | Alan | 5 | 1.18 |
| 5 | Nicky | 5 | 1.18 |

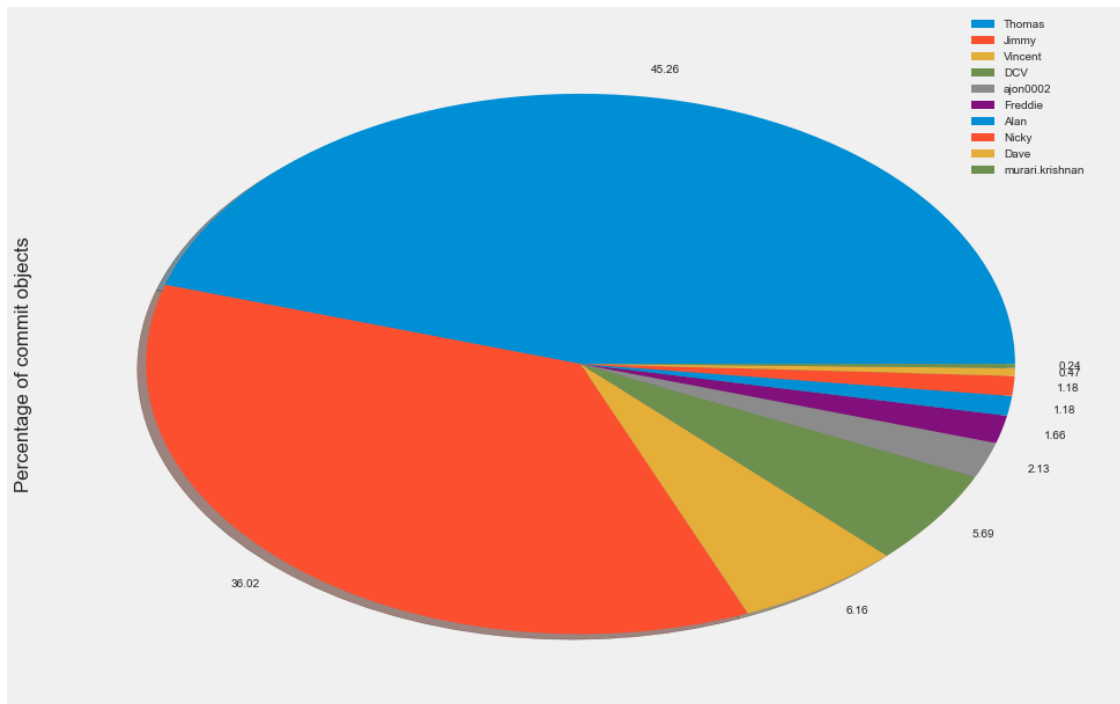
| | | | |
|---|-----------------|---|------|
| 2 | Dave | 2 | 0.47 |
| 9 | murari.krishnan | 1 | 0.24 |

```
In [22]: #create an axis for the pie chart
fig, (axis1) = plt.subplots(1,1,figsize=(15,10))
#plot relevant data frame
ax = log_in_per.plot.pie(y='Percentage', shadow = True, ax = axis1, \
                        labels = log_in_per['Percentage'])

#set a y label
ax.set ( ylabel = 'Percentage of commit objects')
#set a position of the legend to the left top corner
plt.legend(labels = log_in_per['Author'], bbox_to_anchor=(0.85,1.025), \
          loc="upper left")

plt.show
```

Out[22]: <function matplotlib.pyplot.show>



1.4 Number of comments per author and average number of comments per commit object

Once we know that author Thomas is the most active author and that the highest number of commit objects belongs to him, now it will be interesting to see if Thomas generated the biggest number of comment lines and if his average number of comments lines per commit object will be the highest.

I am going to analyse:

- How many comment lines each user have created
- Average number of comments per commit object grouped by author
- If user with the biggest amount of commit objects (Thomas) will have the highest average of commented lines per commit objec

```
In [23]: #check how many comment lines each author has created
number_of_lines = data_frame[['author' , 'number of lines']].groupby(['author'], \
                                                                    as_index = False).sum()

#rename columns
number_of_lines= number_of_lines.rename(columns = \
                                         {'author': 'Author', 'number of lines': 'Commented lines'})

#sort values in descending order
number_of_lines = number_of_lines.sort_values(['Commented lines'], \
                                              ascending = [False])

number_of_lines
```

```
Out[23]:
```

| | Author | Commented lines |
|---|-----------------|-----------------|
| 6 | Thomas | 234 |
| 4 | Jimmy | 154 |
| 7 | Vincent | 80 |
| 1 | DCV | 24 |
| 8 | ajon0002 | 24 |
| 3 | Freddie | 14 |
| 5 | Nicky | 14 |
| 0 | Alan | 8 |
| 2 | Dave | 2 |
| 9 | murari.krishnan | 1 |

```
In [24]: #merge two different data frames
#1 log in count & Commented lines
mergged_df = pd.merge( left = log_in_per, right = number_of_lines, \
                      left_on = 'Author', right_on = 'Author' )

mergged_df
```

```
Out[24]:
```

| | Author | Commit count | Percentage | Commented lines |
|---|-----------------|--------------|------------|-----------------|
| 0 | Thomas | 191 | 45.26 | 234 |
| 1 | Jimmy | 152 | 36.02 | 154 |
| 2 | Vincent | 26 | 6.16 | 80 |
| 3 | DCV | 24 | 5.69 | 24 |
| 4 | ajon0002 | 9 | 2.13 | 24 |
| 5 | Freddie | 7 | 1.66 | 14 |
| 6 | Alan | 5 | 1.18 | 8 |
| 7 | Nicky | 5 | 1.18 | 14 |
| 8 | Dave | 2 | 0.47 | 2 |
| 9 | murari.krishnan | 1 | 0.24 | 1 |

```
In [25]: #add new column to the data frame
mergged_df['Average number of comments per commit'] = \
pd.Series.round(mergged_df['Commented lines'] / mergged_df['Commit count'], 2)
mergged_df
```

```
Out [25]:
```

| | Author | Commit count | Percentage | Commented lines \ |
|---|-----------------|--------------|------------|-------------------|
| 0 | Thomas | 191 | 45.26 | 234 |
| 1 | Jimmy | 152 | 36.02 | 154 |
| 2 | Vincent | 26 | 6.16 | 80 |
| 3 | DCV | 24 | 5.69 | 24 |
| 4 | ajon0002 | 9 | 2.13 | 24 |
| 5 | Freddie | 7 | 1.66 | 14 |
| 6 | Alan | 5 | 1.18 | 8 |
| 7 | Nicky | 5 | 1.18 | 14 |
| 8 | Dave | 2 | 0.47 | 2 |
| 9 | murari.krishnan | 1 | 0.24 | 1 |

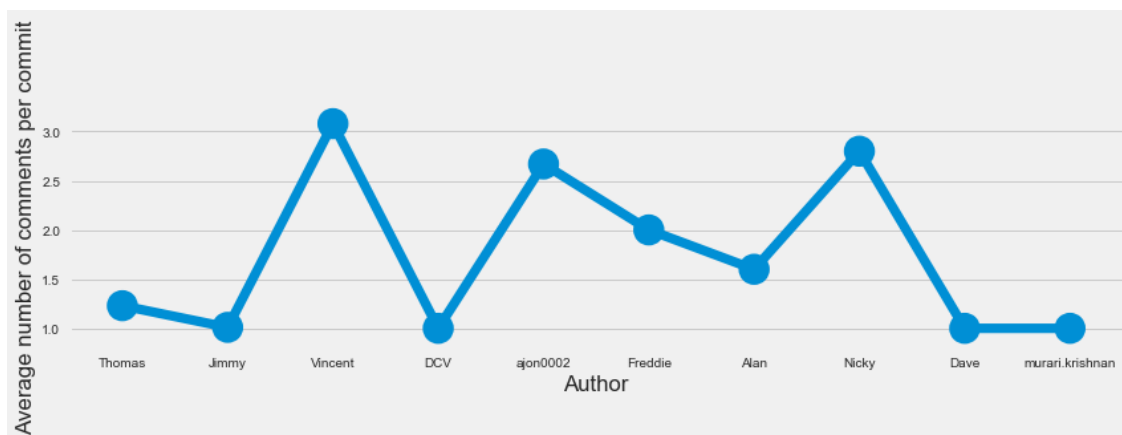
```

Average number of comments per commit
0      1.23
1      1.01
2      3.08
3      1.00
4      2.67
5      2.00
6      1.60
7      2.80
8      1.00
9      1.00

```

```
In [26]: #create a factor plot
sns.factorplot('Author', \
               'Average number of comments per commit', data = mergged_df, \
               size = 4, aspect = 3)
```

```
Out [26]: <seaborn.axisgrid.FacetGrid at 0x492c6a0>
```



Visualisation of the average number of comments per commit object shows that author Vincent has highest average of 3.08 (comments per commit object) while author Thomas who has the largest number of commit objects (191) has average of 1.23 comments per one commit object.

1.5 The favourit part of the day to create a commit object

We have investigated that the highest amount of commit objects is not necessary going to the lead author to a highest average of comments per one commit object.

Now we can have a look into time frame, when the most commit objects have been generated (In the first part of the day (Morning) between 00:00AM and 12:00PM or in the second part of the day (Evening) between 12:00PM - 00:00AM).

```
In [27]: #strip time into hours minutes and seconds
data_frame['hours'], data_frame['minutes'], data_frame['seconds'] = \
    data_frame['time'].str.split(':',2).str

#add additional column in the dataframe,
#0 = first half of the day, 1 = second half of the day
data_frame['day_time'] = data_frame['hours']

#repalce first half of the day values to 0
data_frame['day_time'] = data_frame['day_time'].replace(['00', '01', \
'02', '03', '04', '05', '06', '07', '08', '09', '10', '11'], '0')

#repalce second half of the day values to 1
data_frame['day_time'] = data_frame['day_time'].replace(['12', '13', \
'14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24'], '1')

In [28]: #count how many commits done in the morning and in the evening
count_daytime = data_frame[['day_time', 'revision']].groupby(['day_time'], \
    as_index = False).count()

#Replace value 0 into str Morning
count_daytime['day_time'] = count_daytime['day_time'].replace(['0'], 'Morning')
#replace value 1 into str Evening
count_daytime['day_time'] = count_daytime['day_time'].replace(['1'], 'Evening')
#connect two values for visualisation purpose
count_daytime['visual'] = count_daytime[['day_time', \
'revision']].apply(lambda x : '{} = {}'.format(x[0],x[1]), \
    axis=1)

#display
count_daytime

Out[28]:
```

| | day_time | revision | visual |
|---|----------|----------|---------------|
| 0 | Morning | 138 | Morning = 138 |
| 1 | Evening | 284 | Evening = 284 |

```
In [29]: #count how many comments made in the morning and howe many in the evening
number_of_lines_time = data_frame[['day_time', \
'revision']].groupby(['day_time'], as_index = False).sum()

#Replace value 0 into str Morning
number_of_lines_time['day_time'] = \
    number_of_lines_time['day_time'].replace(['0'], 'Morning')
#replace value 1 into str Evening
```

```

number_of_lines_time['day_time'] = \
    number_of_lines_time['day_time'].replace(['1'], 'Evening')
#connect two values for visualisation purpose
number_of_lines_time['visual'] = number_of_lines_time[['day_time' , \
'number of lines']].apply(lambda x : '{} = {}'.format(x[0],x[1]), \
                                                                    axis=1)

#display
number_of_lines_time

```

```

Out[29]:  day_time  number of lines      visual
0  Morning             173  Morning = 173
1  Evening             382  Evening = 382

```

In [30]:

```

#create sub plots
fig, (axis1,axis2) = plt.subplots(1,2,figsize=(15,6))

#create sns barplot
ax = sns.barplot(x='day_time', y='revision', \
                 data=count_daytime, ax=axis1, order=['Morning','Evening'])
ax.set (xlabel = 'Period of the day', ylabel = 'Count of commits created')

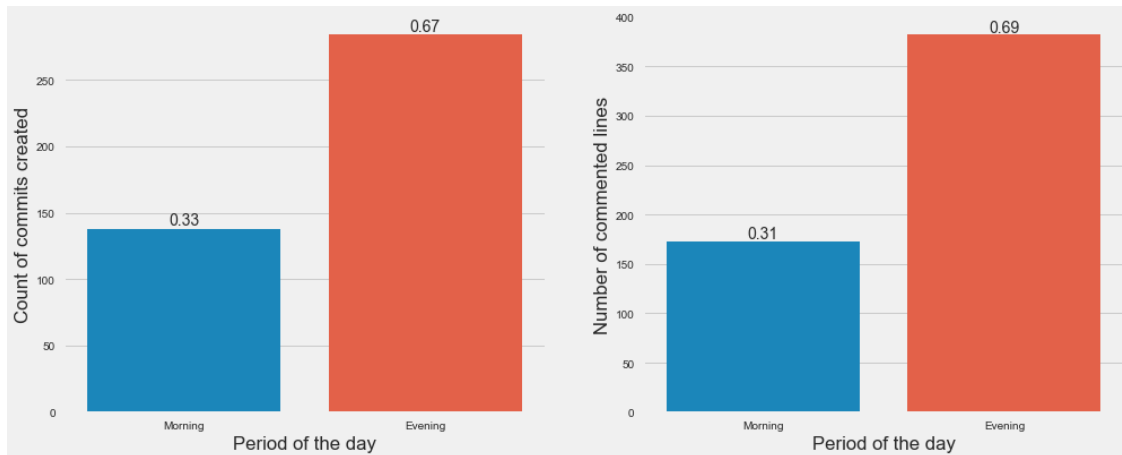
#total is the length of the commits
total = len(data_frame['revision'])
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x()+p.get_width()/2.,
            height + 3,
            '{:1.2f}'.format(height/total),
            ha="center")
plt.show

#create sns barplot
ax = sns.barplot(x='day_time', y='number of lines', \
                 data=number_of_lines_time, ax=axis2, \
                 order=['Morning','Evening'])
ax.set (xlabel = 'Period of the day', ylabel = 'Number of commented lines')

#total is a sum of commented lines in a dataframe
total = sum(number_of_lines_time['number of lines'])
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x()+p.get_width()/2.,
            height + 3,
            '{:1.2f}'.format(height/total),
            ha="center")
plt.show

```

Out[30]: <function matplotlib.pyplot.show>

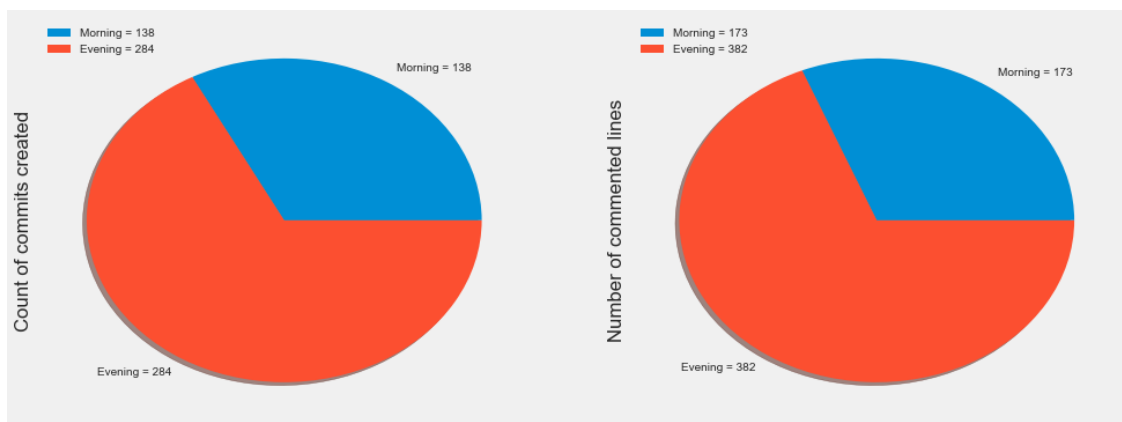


```
In [31]: #create sub plots
fig, (axis1,axis2) = plt.subplots(1,2,figsize=(15,6))

ax = count_daytime.plot.pie(y='revision', shadow = True, ax = axis1, \
                             labels = count_daytime['visual'])
ax.set ( ylabel = 'Count of commits created')
plt.show

ax = number_of_lines_time.plot.pie(y='number of lines', shadow = True, \
                                   ax = axis2, labels = number_of_lines_time['visual'])
ax.set ( ylabel = 'Number of commented lines')
plt.show
```

Out[31]: <function matplotlib.pyplot.show>



Visualisation represents that authors are more active and create more commit objects in the evening time. Overall 67 % of commit objects were created between 12:00PM and 00:00AM, while 69 % of comment lines were created as well in the same time frame.

It will be interesting to see if all authors have been more creative in the evening time. Lets check proportion of generated commit objects and commented lines by each author in the evening and morning time.

```
In [32]: #add additional column to the data frame
data_frame['day_time_str'] = \
data_frame['day_time']
#replace int 0 to str Morning
data_frame['day_time_str'] = \
data_frame['day_time_str'].replace(['0'], 'Morning')
#replace int 1 to str Evening
data_frame['day_time_str'] = \
data_frame['day_time_str'].replace(['1'], 'Evening')

#look for a size of each option (Evening, Morning) grouped by author
#and str and make a count
authors_time_commits = \
data_frame.groupby(['author' , 'day_time_str']).count()

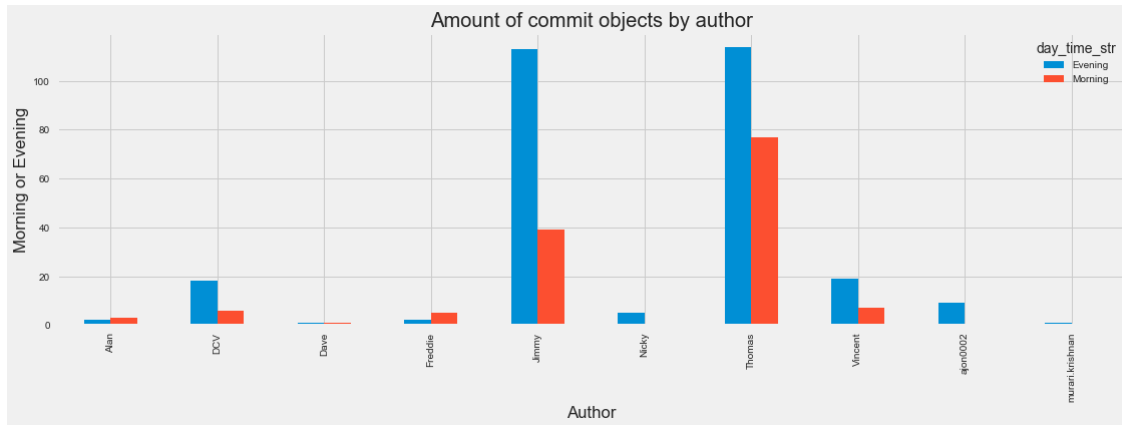
#with out hierachy, we need to unstack grouped data frame
authors_time_commits = \
data_frame.groupby(['author' , 'day_time_str']).count().unstack()
authors_time_commits = authors_time_commits['revision']
authors_time_commits
```

```
Out[32]: day_time_str      Evening  Morning
author
Alan                2.0         3.0
DCV                 18.0         6.0
Dave                1.0         1.0
Freddie             2.0         5.0
Jimmy              113.0        39.0
Nicky               5.0         NaN
Thomas             114.0        77.0
Vincent             19.0         7.0
ajon0002            9.0         NaN
murari.krishnan     1.0         NaN
```

```
In [33]: #create an axis for the plot
fig, (axis1) = plt.subplots(1,1,figsize=(17,5))
plot = authors_time_commits.plot.bar(stacked=False, \
                                     title="Amount of commit objects by author", ax = axis1)
plot.set_xlabel("Author")
```

```
plot.set_ylabel("Morning or Evening")
```

Out[33]: <matplotlib.text.Text at 0xe47ce48>



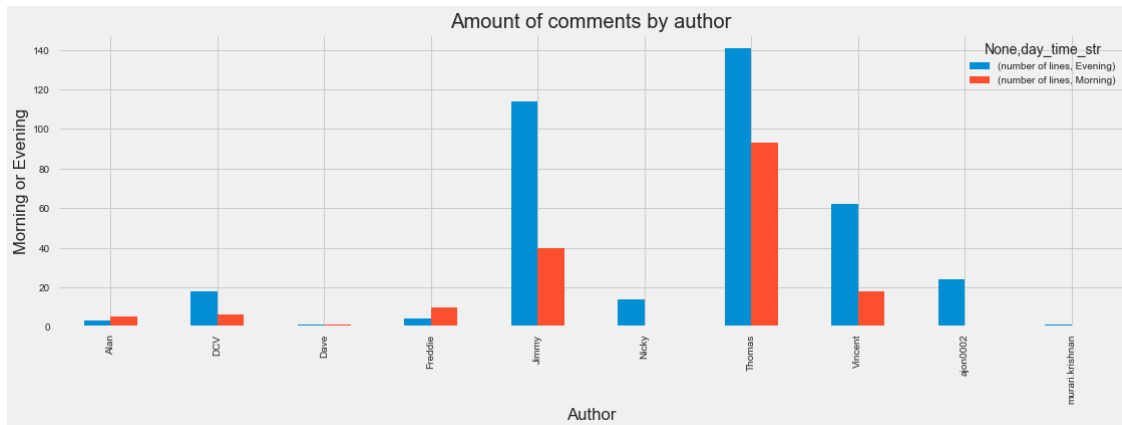
```
In [34]: #data frame grouped by author and sum of comments
authors_time_comments = \
data_frame.groupby(['author' , 'day_time_str']).sum()
#with out hierachy
authors_time_comments = \
data_frame.groupby(['author' , 'day_time_str']).sum().unstack()
authors_time_comments
```

```
Out[34]:
```

| | number of lines | |
|-----------------|-----------------|---------|
| day_time_str | Evening | Morning |
| author | | |
| Alan | 3.0 | 5.0 |
| DCV | 18.0 | 6.0 |
| Dave | 1.0 | 1.0 |
| Freddie | 4.0 | 10.0 |
| Jimmy | 114.0 | 40.0 |
| Nicky | 14.0 | NaN |
| Thomas | 141.0 | 93.0 |
| Vincent | 62.0 | 18.0 |
| ajon0002 | 24.0 | NaN |
| murari.krishnan | 1.0 | NaN |

```
In [35]: #create an axis for the plot
fig, (axis1) = plt.subplots(1,1,figsize=(17,5))
plot = authors_time_comments.plot.bar(stacked=False, \
title="Amount of comments by author", ax = axis1)
plot.set_xlabel("Author")
plot.set_ylabel("Morning or Evening")
```


Out[35]: <matplotlib.text.Text at 0xe6f6eb8>



Visualisations illustrate that from 10 authors only two (Alan & Freddie) have created more commit objects and generated more comment lines during the morning time than a night time. For this reason we can assume that the biggest amount of commit objects and comments were generated during the evening time between 12:00PM and 00:00AM.