

1. Scala's real-world project repository data

With almost 30k commits and a history spanning over ten years, Scala is a mature programming language. It is a general-purpose programming language that has recently become another prominent language for data scientists.

Scala is also an open source project. Open source projects have the advantage that their entire development histories -- who made changes, what was changed, code reviews, etc. -- are publicly available.

We're going to read in, clean up, and visualize the real world project repository of Scala that spans data from a version control system (Git) as well as a project hosting site (GitHub). We will find out who has had the most influence on its development and who are the experts.

The dataset we will use, which has been previously mined and extracted from GitHub, is comprised of three files:

1. `pulls_2011-2013.csv` contains the basic information about the pull requests, and spans from the end of 2011 up to (but not including) 2014.
2. `pulls_2014-2018.csv` contains identical information, and spans from 2014 up to 2018.
3. `pull_files.csv` contains the files that were modified by each pull request.

```
In [1]: # Importing pandas and pyplot
import pandas as pd
import matplotlib.pyplot as plt

# Loading in the data
pulls_one = pd.read_csv('datasets/pulls_2011-2013.csv')
pulls_two = pd.read_csv('datasets/pulls_2014-2018.csv')
pull_files = pd.read_csv('datasets/pull_files.csv')
```

2. Preparing and cleaning the data

First, we will need to combine the data from the two separate pull DataFrames.

Next, the raw data extracted from GitHub contains dates in the ISO8601 format. However, `pandas` imports them as regular strings. To make our analysis easier, we need to convert the strings into Python's `DateTime` objects. `DateTime` objects have the important property that they can be compared and sorted.

The pull request times are all in UTC (also known as Coordinated Universal Time). The commit times, however, are in the local time of the author with time zone information (number of hours difference from UTC). To make comparisons easy, we should convert all times to UTC.

```
In [2]: # Append pulls_one to pulls_two
pulls = pulls_two.append(pulls_one)
```

```
# Convert the date for the pulls object
pulls['date'] = pd.to_datetime(pulls['date'], utc=True)
```

3. Merging the DataFrames

The data extracted comes in two separate files. Merging the two DataFrames will make it easier for us to analyze the data in the future tasks.

```
In [3]: # Merge the two DataFrames
data = pulls.merge(pull_files, on='pid')
```

4. Is the project still actively maintained?

The activity in an open source project is not very consistent. Some projects might be active for many years after the initial release, while others can slowly taper out into oblivion. Before committing to contributing to a project, it is important to understand the state of the project. Is development going steadily, or is there a drop? Has the project been abandoned altogether?

The data used in this project was collected in January of 2018. We are interested in the evolution of the number of contributions up to that date.

For Scala, we will do this by plotting a chart of the project's activity. We will calculate the number of pull requests submitted each (calendar) month during the project's lifetime. We will then plot these numbers to see the trend of contributions.

- A helpful reminder of how to access various components of a date can be found in [this exercise of Data Manipulation with pandas](#)
- Additionally, recall that you can group by multiple variables by passing a list to `groupby()`. This video from [Data Manipulation with pandas](#) should help!

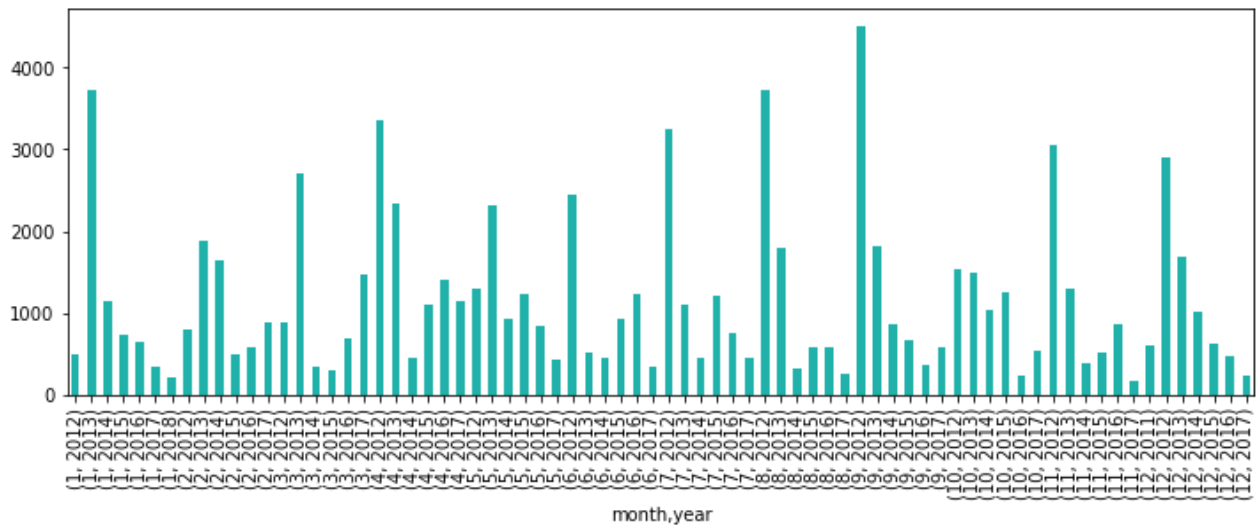
```
In [4]: %matplotlib inline

# Create a column that will store the month
data['month'] = data['date'].dt.month

# Create a column that will store the year
data['year'] = data['date'].dt.year

# Group by the month and year and count the pull requests
counts = data.groupby(['month', 'year'])['pid'].count()

# Plot the results
counts.plot(kind='bar', figsize = (12,4), color='LightSeaGreen')
plt.show()
```



5. Is there camaraderie in the project?

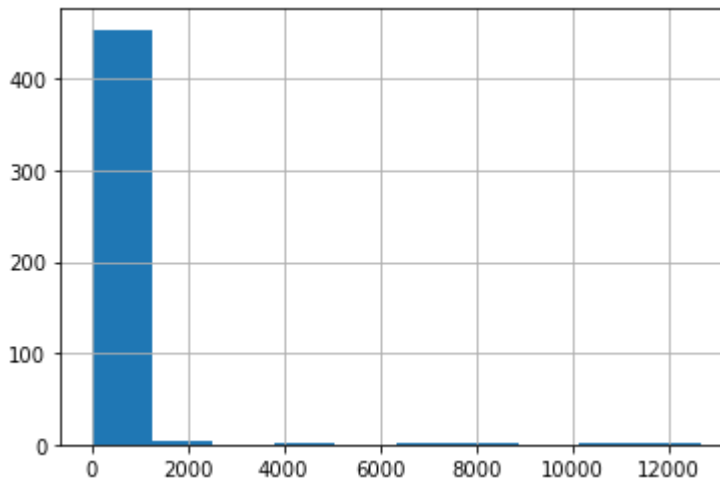
The organizational structure varies from one project to another, and it can influence your success as a contributor. A project that has a very small community might not be the best one to start working on. The small community might indicate a high barrier of entry. This can be caused by several factors, including a community that is reluctant to accept pull requests from "outsiders," that the code base is hard to work with, etc. However, a large community can serve as an indicator that the project is regularly accepting pull requests from new contributors. Such a project would be a good place to start.

In order to evaluate the dynamics of the community, we will plot a histogram of the number of pull requests submitted by each user. A distribution that shows that there are few people that only contribute a small number of pull requests can be used as an indicator that the project is not welcoming of new contributors.

```
In [5]: # Required for matplotlib
%matplotlib inline

# Group by the submitter
by_user = data.groupby('user')['pid'].count()

# Plot the histogram
by_user.hist()
plt.show()
```



6. What files were changed in the last ten pull requests?

Choosing the right place to make a contribution is as important as choosing the project to contribute to. Some parts of the code might be stable, some might be dead. Contributing there might not have the most impact. Therefore it is important to understand the parts of the system that have been recently changed. This allows us to pinpoint the "hot" areas of the code where most of the activity is happening. Focusing on those parts might not be the most effective use of our times.

```
In [6]: # Identify the last 10 pull requests
last_10 = pulls.nlargest(10, 'date')

# Join the two data sets
joined_pr = last_10.merge(pull_files, on='pid')

# Identify the unique files
files = set(joined_pr['file'])

# Print the results
files
```

```
Out[6]: {'LICENSE',
'doc/LICENSE.md',
'doc/License.rtf',
'project/VersionUtil.scala',
'src/compiler/scala/reflect/reify/phases/Calculate.scala',
'src/compiler/scala/tools/nsc/backend/jvm/BCodeHelpers.scala',
'src/compiler/scala/tools/nsc/backend/jvm/PostProcessor.scala',
'src/compiler/scala/tools/nsc/backend/jvm/analysis/BackendUtils.scala',
'src/compiler/scala/tools/nsc/profile/AsyncHelper.scala',
'src/compiler/scala/tools/nsc/profile/Profiler.scala',
'src/compiler/scala/tools/nsc/symtab/classfile/ClassfileParser.scala',
'src/compiler/scala/tools/nsc/typechecker/Contexts.scala',
'src/library/scala/Predef.scala',
'src/library/scala/concurrent/Lock.scala',
'src/library/scala/util/Properties.scala',
'src/reflect/scala/reflect/internal/pickling/ByteCodecs.scala',
'src/reflect/scala/reflect/internal/tpe/GlbLubs.scala',
'src/scaladoc/scala/tools/nsc/doc/html/page/Entity.scala',
'src/scalap/decoder.properties',
'test/files/neg/leibniz-liskov.check',
'test/files/neg/leibniz-liskov.scala',
```

```
'test/files/pos/leibniz-liskov.scala',
'test/files/pos/leibniz_liskov.scala',
'test/files/pos/parallel-classloader.scala',
'test/files/pos/t10568/Converter.java',
'test/files/pos/t10568/Impl.scala',
'test/files/pos/t10686.scala',
'test/files/pos/t5638/Among.java',
'test/files/pos/t5638/Usage.scala',
'test/files/pos/t9291.scala',
'test/files/run/t8348.check',
'test/files/run/t8348/TableColumn.java',
'test/files/run/t8348/TableColumnImpl.java',
'test/files/run/t8348/Test.scala'}
```

7. Who made the most pull requests to a given file?

When contributing to a project, we might need some guidance. We might find ourselves needing some information regarding the codebase. It is important direct any questions to the right person. Contributors to open source projects generally have other day jobs, so their time is limited. It is important to address our questions to the right people. One way to identify the right target for our inquiries is by using their contribution history.

We identified `src/compiler/scala/reflect/reify/phases/Calculate.scala` as being recently changed. We are interested in the top 3 developers who changed that file. Those developers are the ones most likely to have the best understanding of the code.

In [7]:

```
# This is the file we are interested in:
file = 'src/compiler/scala/reflect/reify/phases/Calculate.scala'

# Identify the commits that changed the file
file_pr = data.query('file == @file')

# Count the number of changes made by each developer
author_counts = file_pr.groupby('user').count()

# Print the top 3 developers
print(author_counts.nlargest(3, 'pid'))
```

	pid	date	file	month	year
user					
xeno-by	11	11	11	11	11
retronym	5	5	5	5	5
soc	4	4	4	4	4

8. Who made the last ten pull requests on a given file?

Open source projects suffer from fluctuating membership. This makes the problem of finding the right person more challenging: the person has to be knowledgeable *and* still be involved in the project. A person that contributed a lot in the past might no longer be available (or willing) to help. To get a better understanding, we need to investigate the more recent history of that particular part of the system.

Like in the previous task, we will look at the history of

```
src/compiler/scala/reflect/reify/phases/Calculate.scala.
```

```
In [8]: file = 'src/compiler/scala/reflect/reify/phases/Calculate.scala'

# Select the pull requests that changed the target file
file_pr = data.query('file == @file')

# Merge the obtained results with the pulls DataFrame
joined_pr = file_pr.merge(pulls, on=['pid', 'user', 'date'])

# Find the users of the last 10 most recent pull requests
users_last_10 = set(joined_pr.nlargest(10, 'date')['user'])

# Printing the results
users_last_10
```

```
Out[8]: {'bjornregnell', 'retronym', 'soc', 'starblood', 'xeno-by', 'zuvizudar'}
```

9. The pull requests of two special developers

Now that we have identified two potential contacts in the projects, we need to find the person who was most involved in the project in recent times. That person is most likely to answer our questions. For each calendar year, we are interested in understanding the number of pull requests the authors submitted. This will give us a high-level image of their contribution trend to the project.

```
In [9]: %matplotlib inline

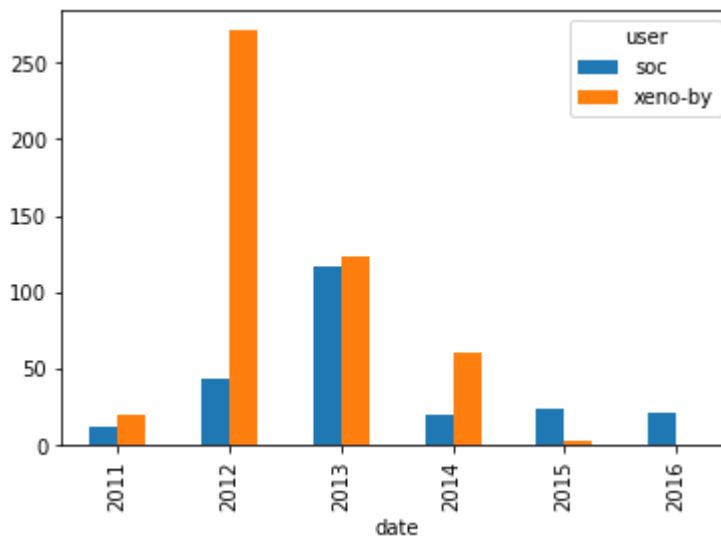
# The developers we are interested in
authors = ['xeno-by', 'soc']

# Get all the developers' pull requests
by_author = pulls.query('user == @authors')

# Count the number of pull requests submitted each year
counts = by_author.groupby(['user', by_author['date'].dt.year]).agg({'pid': 'count'}).r

# Convert the table to a wide format
counts_wide = counts.pivot_table(index='date', columns='user', values='pid', fill_value=0)

# Plot the results
counts_wide.plot(kind='bar')
plt.show()
```



10. Visualizing the contributions of each developer

As mentioned before, it is important to make a distinction between the global expertise and contribution levels and the contribution levels at a more granular level (file, submodule, etc.) In our case, we want to see which of our two developers of interest have the most experience with the code in a given file. We will measure experience by the number of pull requests submitted that affect that file and how recent those pull requests were submitted.

In [10]:

```
authors = ['xeno-by', 'soc']
file = 'src/compiler/scala/reflect/reify/phases/Calculate.scala'

# Select the pull requests submitted by the authors, from the `data` DataFrame
by_author = data.query('user == @authors')

# Select the pull requests that affect the file
by_file = by_author.query('file == @file')

# Group and count the number of PRs done by each user each year
grouped = by_file.groupby(['user', by_file['date'].dt.year]).count()['pid'].reset_index

# Transform the data into a wide format
by_file_wide = pd.pivot_table(grouped, values='pid', columns=['user'], index=['date'])

# Plot the results
by_file_wide.plot(kind='bar')
plt.show()
```

