**Поиск угроз с помощью Jupyter Notebook.**

**Часть 2. Базовый анализ данных с помощью Pandas**



В предыдущей статье я представил основные концепции Jupyter Notebook и показал, как развернуть базовую среду блокнотов. К этому моменту мы готовы запустить ещё кода Python и показать вам, насколько гибкими и динамичными могут быть блокноты для анализа событий безопасности.

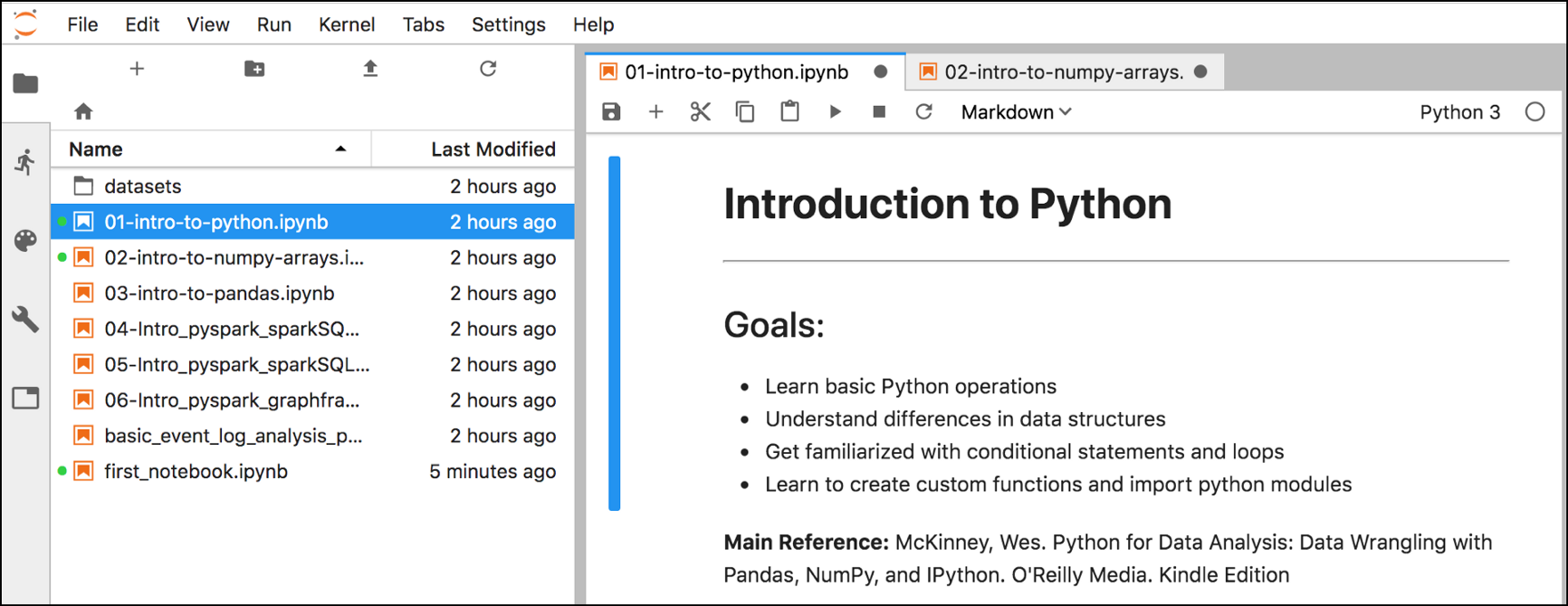
В этой статье я покажу вам, как читать и анализировать файл журнала безопасности в формате JSON с помощью библиотеки Python под названием Pandas.

Эта статья является частью из пяти статей. Остальные четыре части можно найти по следующим ссылкам:

* Поиск угроз с помощью Jupyter Notebook. Часть 1. Ваш первый Notebook.
* Поиск угроз с помощью Jupyter Notebook. Часть 3. Поиск Elasticsearch через Apache Spark.
* Поиск угроз с помощью Jupyter Notebook. Часть 4. SQL JOIN через Apache SparkSQL.
* Поиск угроз с помощью Jupyter Notebook. Часть 5. Документирование, совместное использование и запуск книг-охотников за угрозами!

**Требования**

* Предполагается, что вы читали предыдущую статью и открывали проект HELK, следуя указанным инструкциям.
* Основные навыки программирования на Python и базовые знания Python Numpy Arrays. Если вы новичок в структурах данных Python, потоках управления, циклах, функциях, модулях и массивах NumPy или хотели бы освежить свою память, я подобрал для вас две краткие вступительные записи. Откройте блокноты 01-intro-to-python.ipynb и 02-intro-to-numpy-arrays.ipynb.



Несмотря на то, что хочется сразу перейти к анализу журналов событий, я считаю, что важно понять основные концепции, лежащие в основе Pandas.

**Что такое Pandas?**

*Pandas — это пакет Python, предоставляющий быстрые, гибкие и выразительные структуры данных, предназначенные для того, чтобы сделать работу с «реляционными» или «помеченными» данными простой или интуитивно понятной. Pandas призван стать фундаментальным строительным блоком высокого уровня для практического анализа данных в реальном мире в Python.*

Объекты данных Pandas сильно зависят от высокопроизводительной структуры данных массива NumPy и обеспечивают гибкие возможности манипулирования данными типичных электронных таблиц и реляционных баз данных (таких как SQL).

**Что такое Numpy?**

* NumPy (сокр. от Numeric Python) является фундаментальным пакетом Python для научных вычислений.
* Он использует высокопроизводительную структуру данных, известную как n-мерный массив или ndarray, объект многомерного массива для эффективного вычисления массивов и матриц.

**Что такое Array?**

* Массивы Python — это структуры данных, в которых хранятся данные, аналогичные списку. Исключением является ограничение типа хранимых в них объектов. Элементы массива имеют одинаковый тип и индексируются набором натуральных чисел.
* Массив модуля python позволяет указывать тип массива во время создания объекта, используя код типа, который является одним символом.

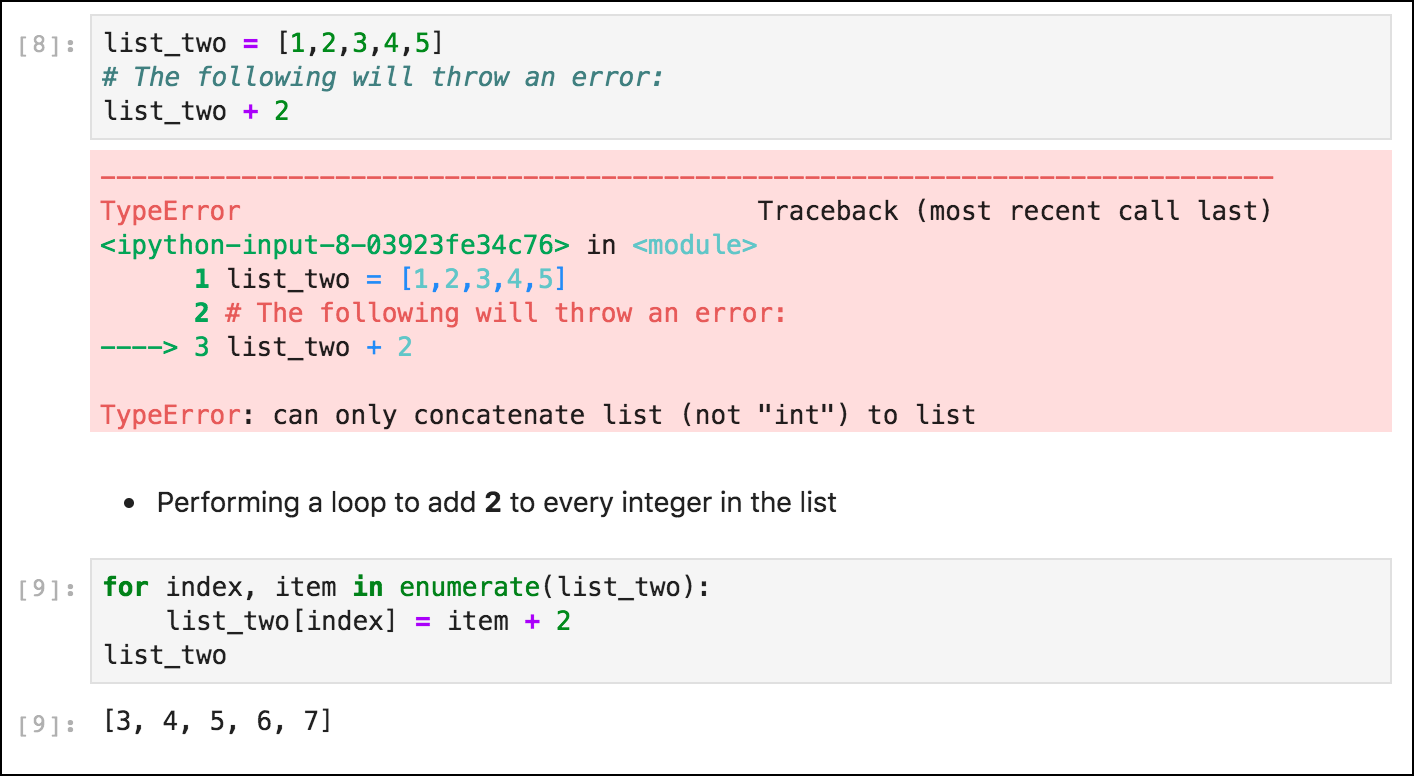


**NumPy N-мерный массив (ndarray)**

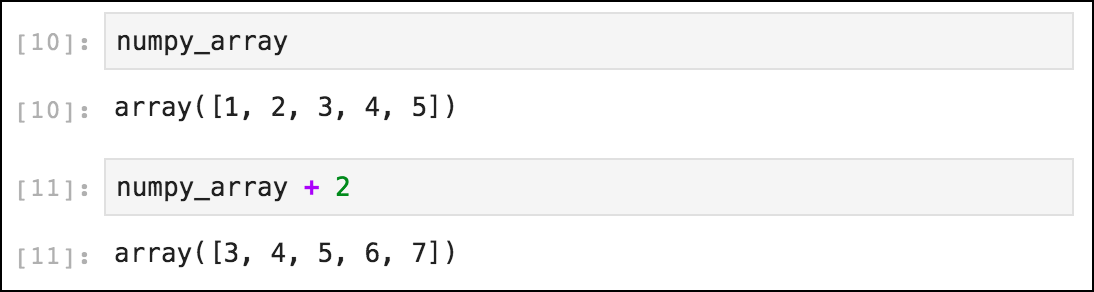
* Массив NumPy — это эффективный многомерный массив, обеспечивающий быстрые арифметические операции, ориентированные на массив.
* Как и любой другой массив Python, это контейнер для элементов того же типа.



* Как и в случае с другими объектами-контейнерами в Python, содержимое **ndarray** может быть доступно и изменено с помощью операций индексации или среза.
* Массивы NumPy обрабатывают векторизованные операции и более эффективны для хранения, чем другие встроенные структуры данных Python.
* Если мы пытаемся добавить число к каждому элементу списка, необходимо выполнить цикл, чтобы применить базовое добавление к каждому элементу.



* Если сделать то же самое с массивом NumPy, то вы сможете добавить число в объект массива, и оно будет применено к каждому элементу массива.



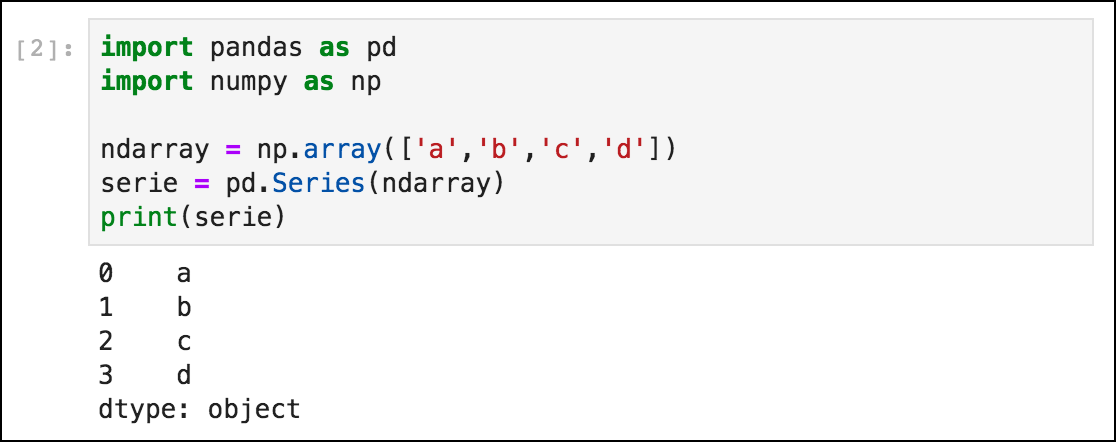
**Pandas и массивы NumPy**

Pandas использует и расширяет ряд функций на основе массивов из NumPy и предоставляет более выразительные средства представления и управления данными. Две основные структуры данных в Pandas:

* Series
* DataFrames

**Pandas Series**

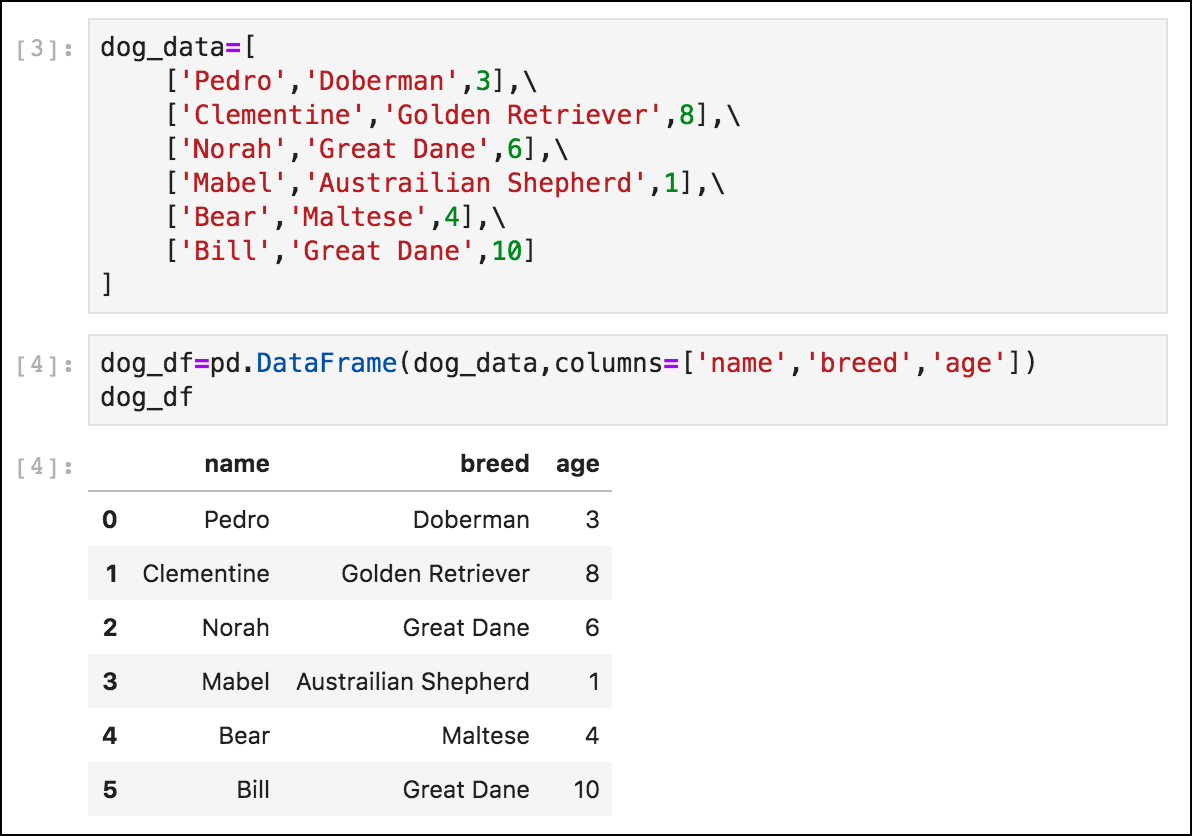
* Series — это одномерный помеченный индексированный массив, который может содержать любой тип данных, например целые числа, числа с плавающей запятой и строки.
* Series можно создать и инициализировать простым путем передачи NumPy ndarray.



* Как вы видели выше, индекс находится слева, а значения - справа. Мы не указывали индекс для данных, поэтому по умолчанию создается индекс, начинающийся с 0.

**Pandas DataFrames**

* DataFrame — это двумерная таблица, которая используется для представления данных в табличном формате.



* Представьте DataFrame как электронную таблицу с заголовками столбцов и строками.
* DataFrame имеет индекс строки и столбца.
* Pandas полагается на типы данных NumPy для числовых записей в фреймах данных. Например, мы можем напечатать тип отдельных записей, таких как первое значение в столбце age, используя iloc, и заметить, что он имеет тип NumPy.

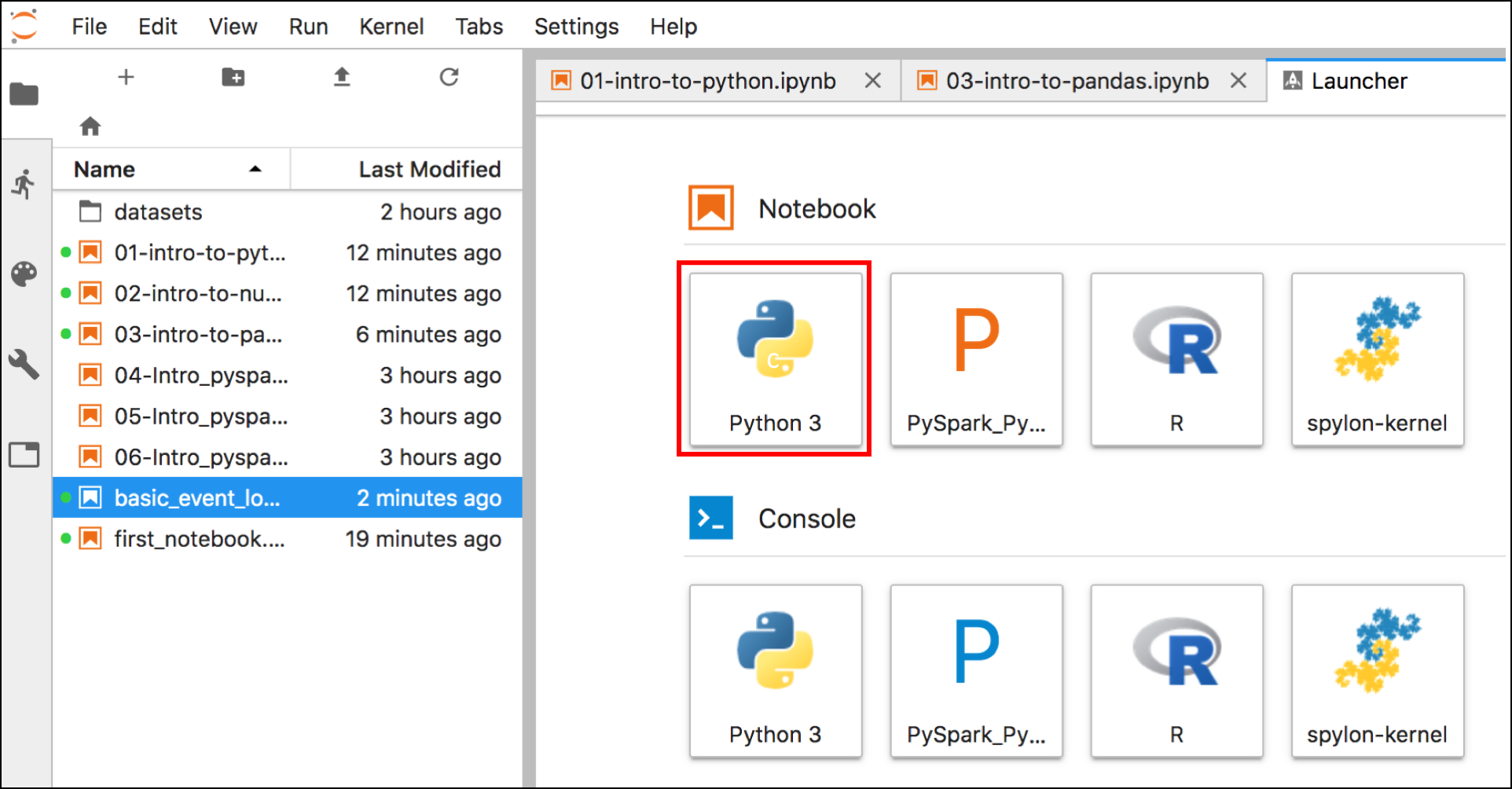


Теперь, когда мы понимаем основы и структуру данных Pandas, мы можем поиграть с журналами событий безопасности в табличном формате через DataFrames.

**Анализ основных событий**

Давайте создадим новый блокнот, изучим набор данных Mordor, обогатим его, визуализируем и отфильтруем интересные события. Решением этого блокнота называется basic\_event\_log\_analysis\_pandas, и оно доступно для вас. Я рекомендую воссоздать его, следуя приведенным ниже инструкциям, чтобы вы могли попрактиковаться.

**Создайте новый блокнот**



**Импортируйте библиотеки**

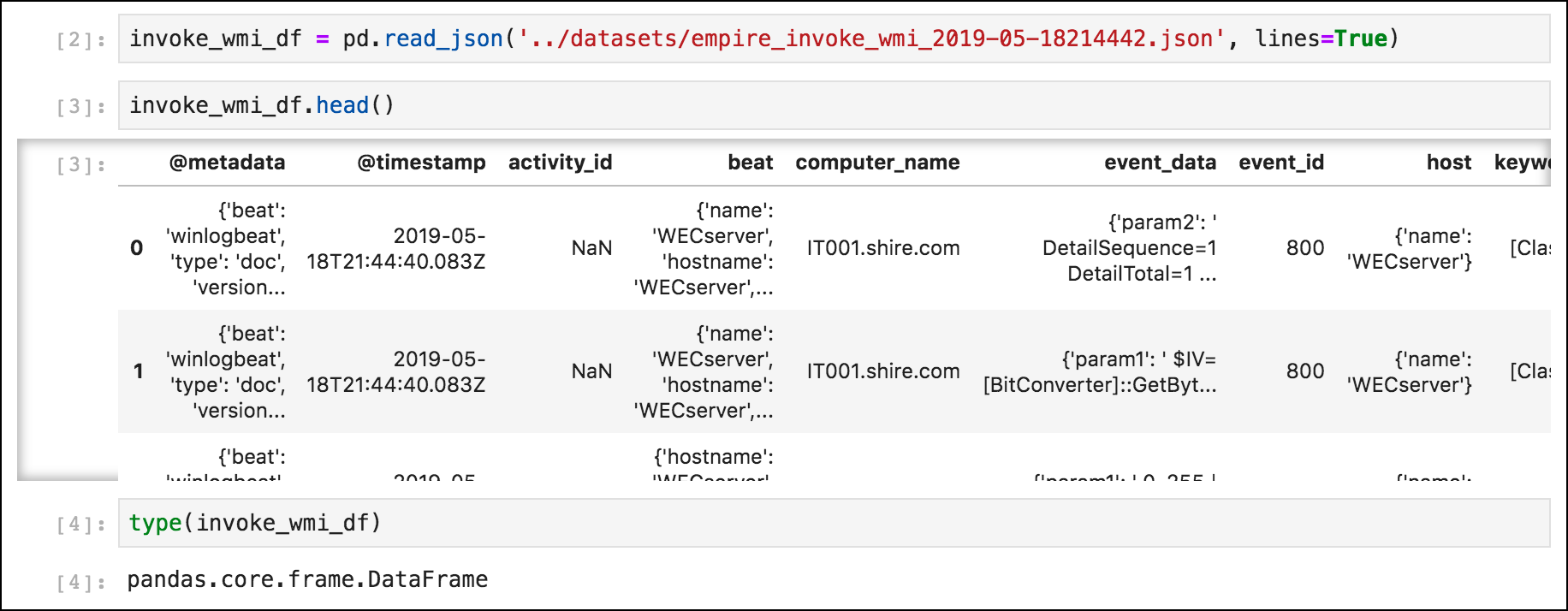
import pandas as pd  
from pandas.io.json import json\_normalize  
import json  
import matplotlib.pyplot as plt



**Прочитайте набор данных Mordor**

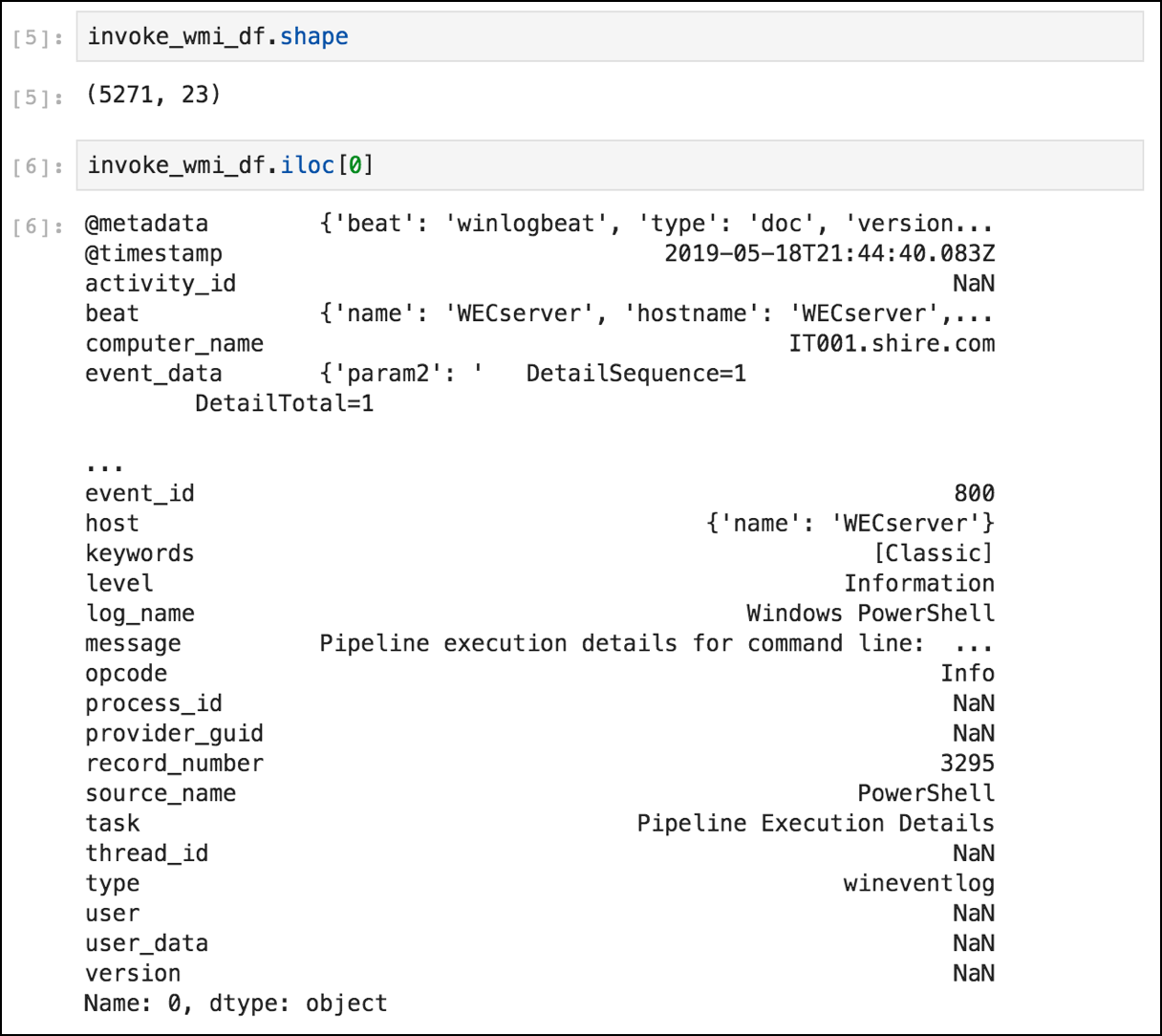
* Мы собираемся прочитать небольшой набор данных empire\_invoke\_wmi Mordor.
* Мы можем использовать метод read\_json для преобразования JSON-файла Mordor в DataFrame. Параметр lines имеет значение True, чтобы читать файл как объект json на строку.

invoke\_wmi\_df = pd.read\_json('datasets/empire\_invoke\_wmi\_2019-05-18214442.json', lines=True)



**Изучите данные**

* Далее, можно использовать shape для возврата кортежа, представляющего размерность DataFrame, который содержит 5271 ряд и 23 столбца.
* Кроме того, можно использовать iloc для выбора первой записи в DataFrame. На этом этапе мы получаем представление о схеме, которая находится в DataFrame. Помните, что наш набор данных содержит разрозненные источники данных. Поэтому я надеюсь, что схема сильно изменится для каждой записи или категории аудита.



**Выровняйте вложенные колонки**

* На предыдущем изображении мы видели несколько вложенных полей в наборе данных. Поэтому мы можем использовать json\_normalize, чтобы выровнять все эти столбцы.
* Сначала я перевожу DataFrame обратно в JSON с помощью метода to\_json.
* Затем загружаю результаты в виде структуры json, чтобы упорядочить их с помощью функции json\_normalize и получить взамен DataFrame.

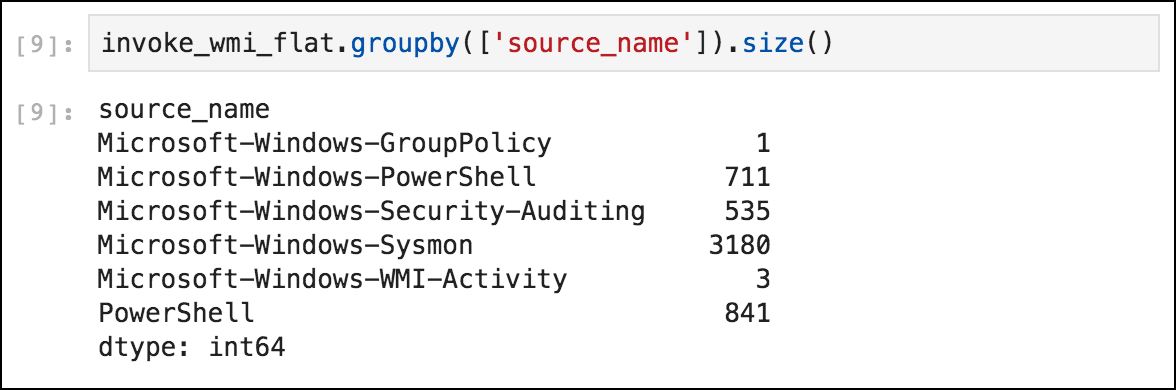
json\_struct = json.loads(invoke\_wmi\_df.to\_json(orient="records"))  
invoke\_wmi\_flat = json\_normalize(json\_struct)



**Определите доступные источники данных**

* Вы можете использовать метод pandas groupby и скомпоновать DataFrame по полю source\_name. Мне нравится делать это, так как можно узнать больше о доступных источниках данных, с которыми я могу поиграть.

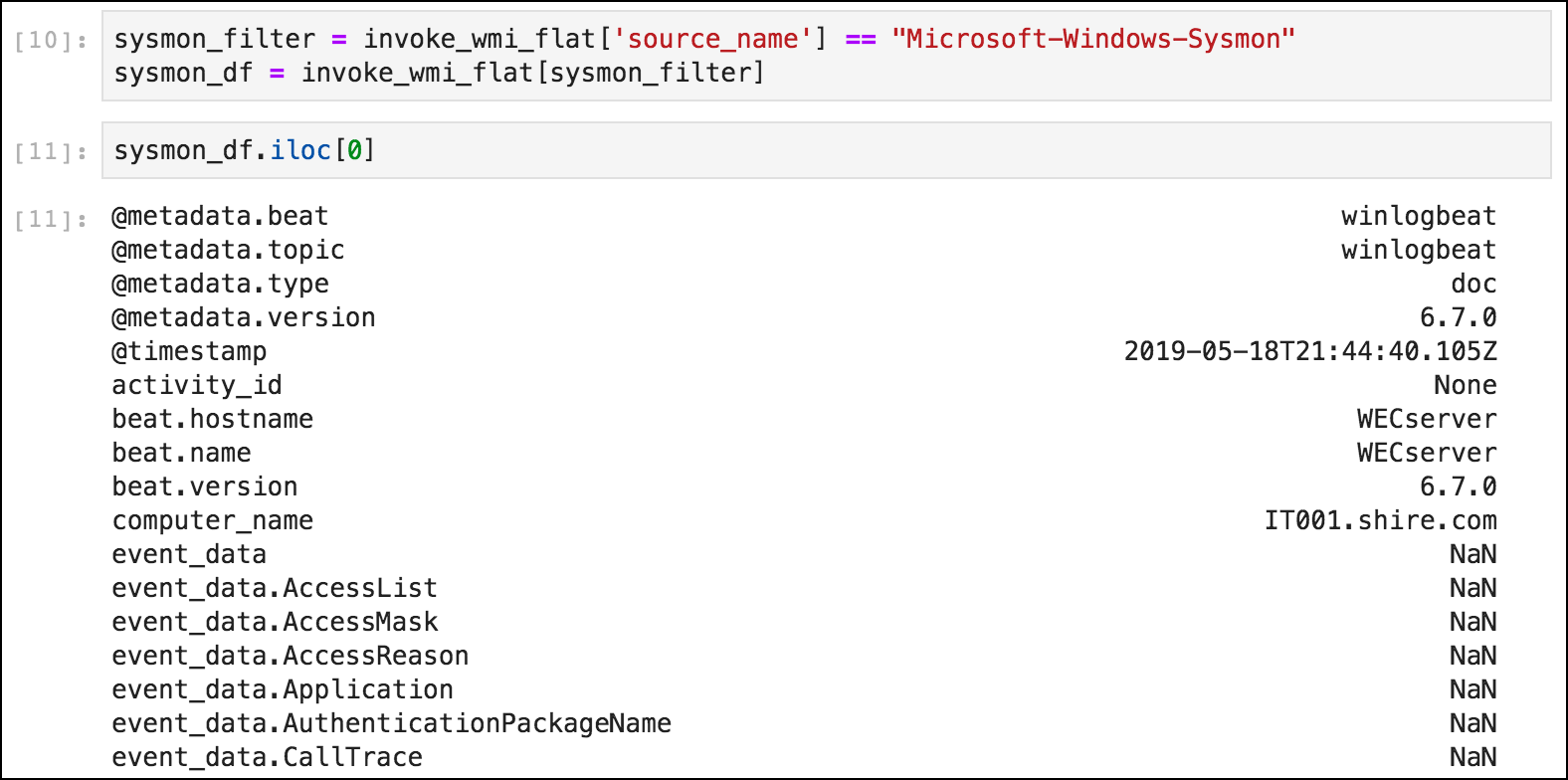
invoke\_wmi\_flat.groupby(['source\_name']).size()



**Отфильтруйте события Sysmon**

* Теперь можно фильтровать наш основной DataFrame по конкретному источнику данных.
* Для начала я решил отфильтровать только события Microsoft-Windows-Sysmon.
* Как вы можете видеть ниже, есть ещё несколько полей со значениями NaN. Это потому, что схема данных для каждого события в Sysmon отличается.

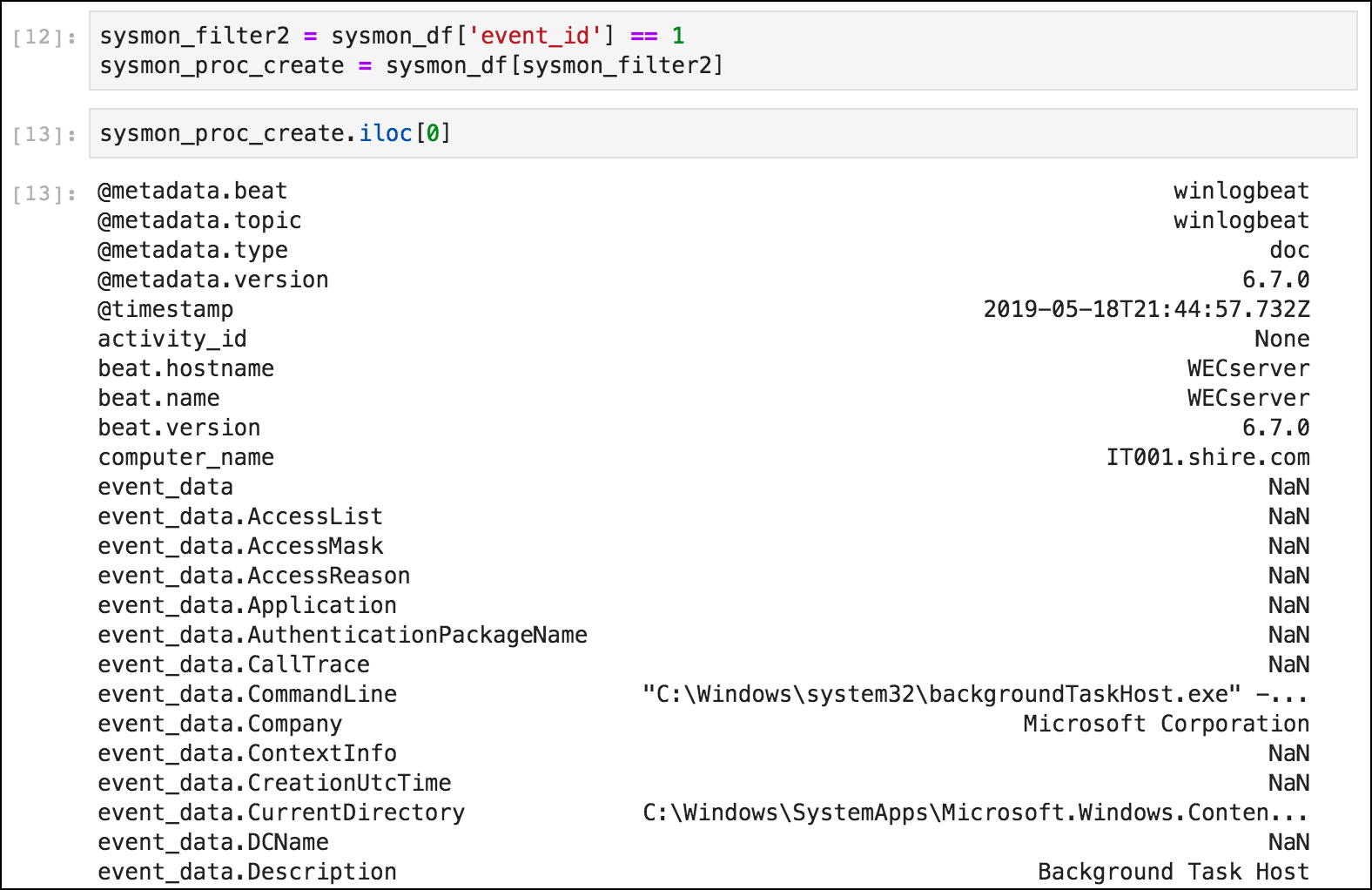
sysmon\_filter = invoke\_wmi\_flat['source\_name'] == "Microsoft-Windows-Sysmon"  
sysmon\_df = invoke\_wmi\_flat[sysmon\_filter]



**Отфильтруйте по событиям создание процессов Sysmon (EID 1)**

* Теперь, когда у нас есть Sysmon DataFrame, мы можем фильтровать эти события по определённым идентификаторам событий. Давайте сделаем это, передав ему фильтр event\_id == 1.

sysmon\_filter2 = sysmon\_df['event\_id'] == 1  
sysmon\_proc\_create = sysmon\_df[sysmon\_filter2]



**Удалите пустые поля в ProcessCreate Dataframe**

* После фильтрации Sysmon DataFrame по одному конкретному событию можно удалить столбцы со значениями NaN. Все записи, относящиеся к одному и тому же типу события (идентификатору события), должны содержать одну и ту же схему для лёгкой фильтрации полей, не принадлежащих схеме ProcessCreate.
* Узнать больше о схеме можно с помощью OSSEM Sysmon Event Id 1.

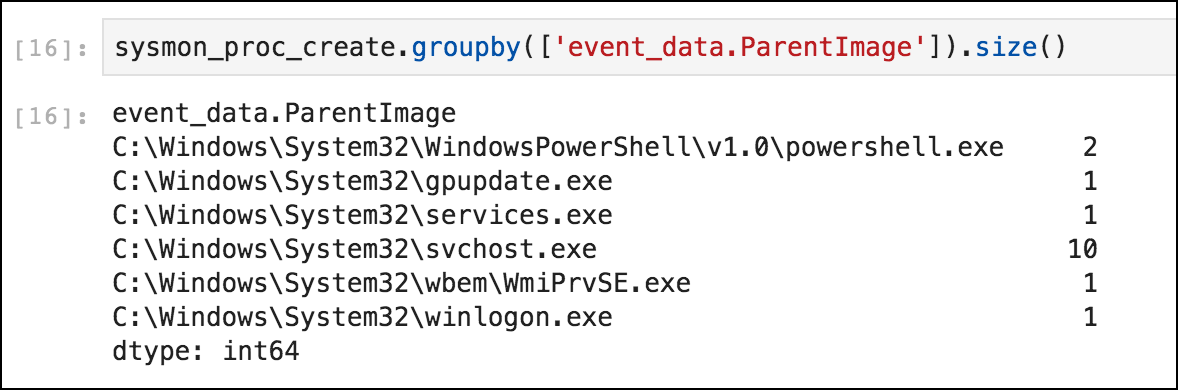
sysmon\_proc\_create = sysmon\_proc\_create.dropna(axis = 1, how = 'all')



**Скомпонуйте события по определённым полям**

* Давайте рассмотрим наши события ProcessCreate и сгруппируем их по ParentImage.
* Согласно набору данных, файл содержит боковое смещение по WMI для появления нового агента на целевой машине. Поэтому я решил сложить значения столбца ParentImage, чтобы узнать больше о родительских процессах, которые выполняли какое-то действие в конечной точке.

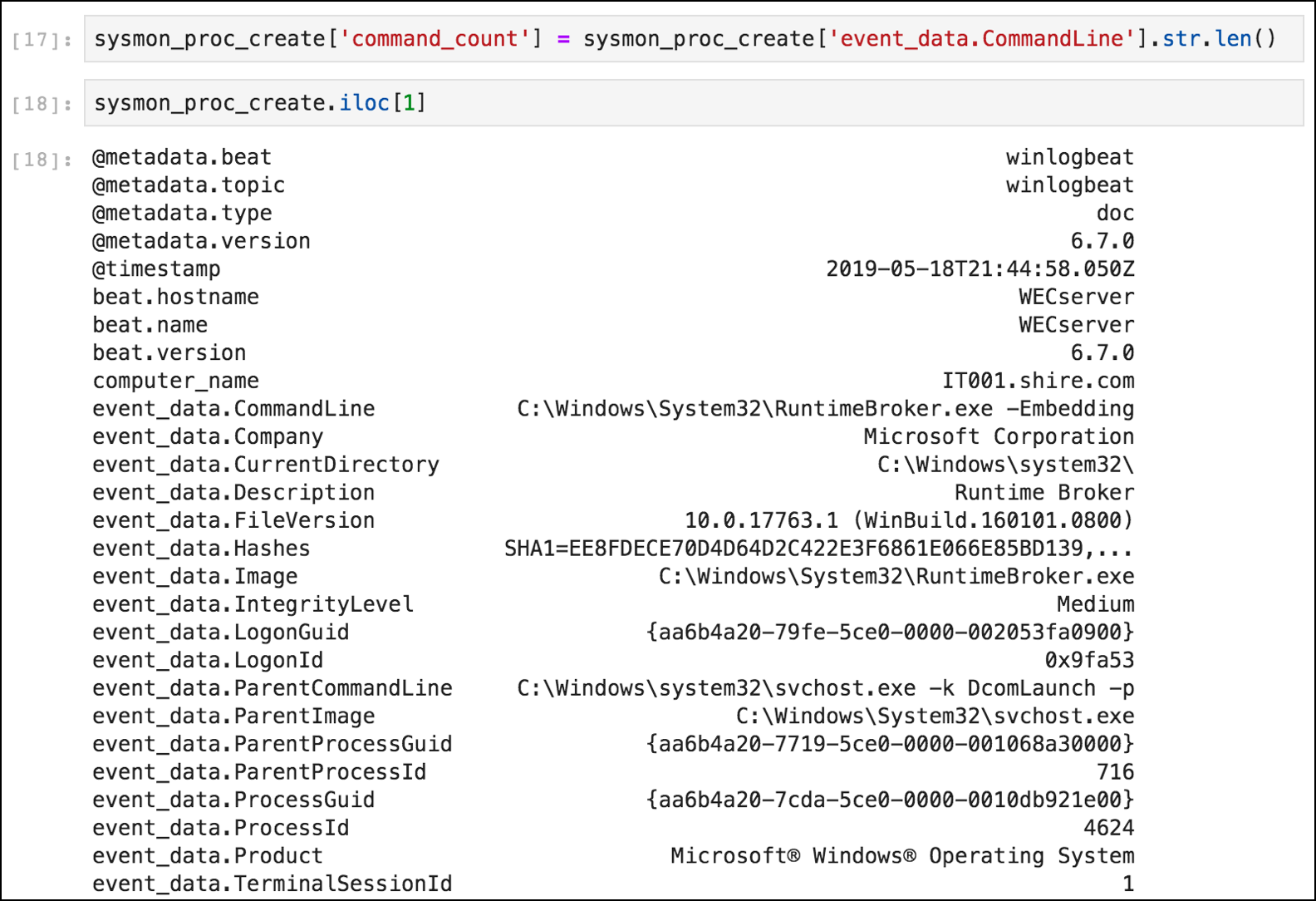
sysmon\_proc\_create.groupby(['event\_data.ParentImage']).size()

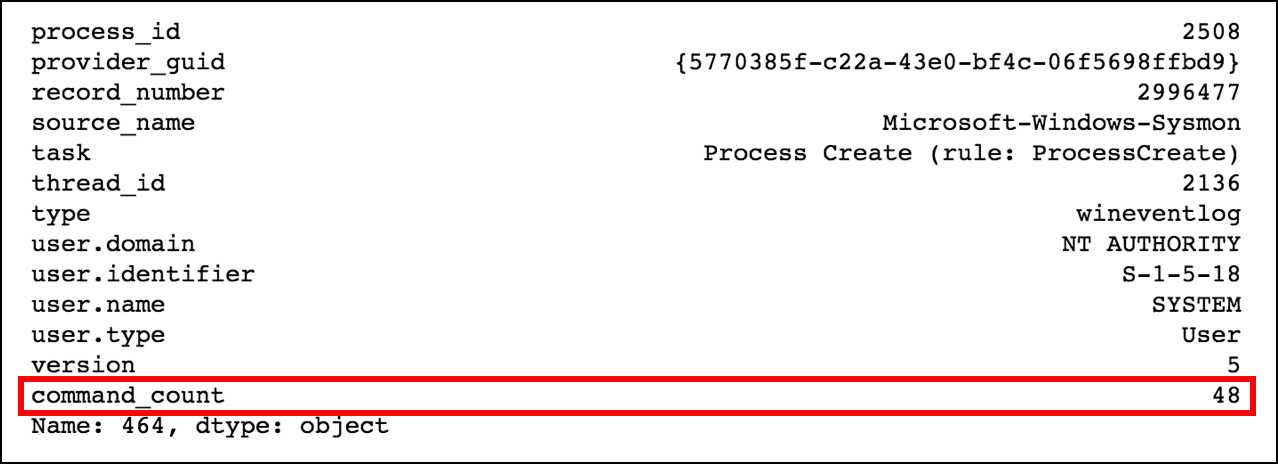


**Пополните данные**

* Один из основных шагов, которые мы можем сделать, — пополнить текущие события.
* Мы можем добавить новое поле в наш DataFrame и установить его с результатами вычисления общего количества символов, найденных в каждой CommandLine для каждой записи. Затем можно использовать это новое поле для поиска потенциальных выбросов.

sysmon\_proc\_create['command\_count'] = sysmon\_proc\_create['event\_data.CommandLine'].str.len()

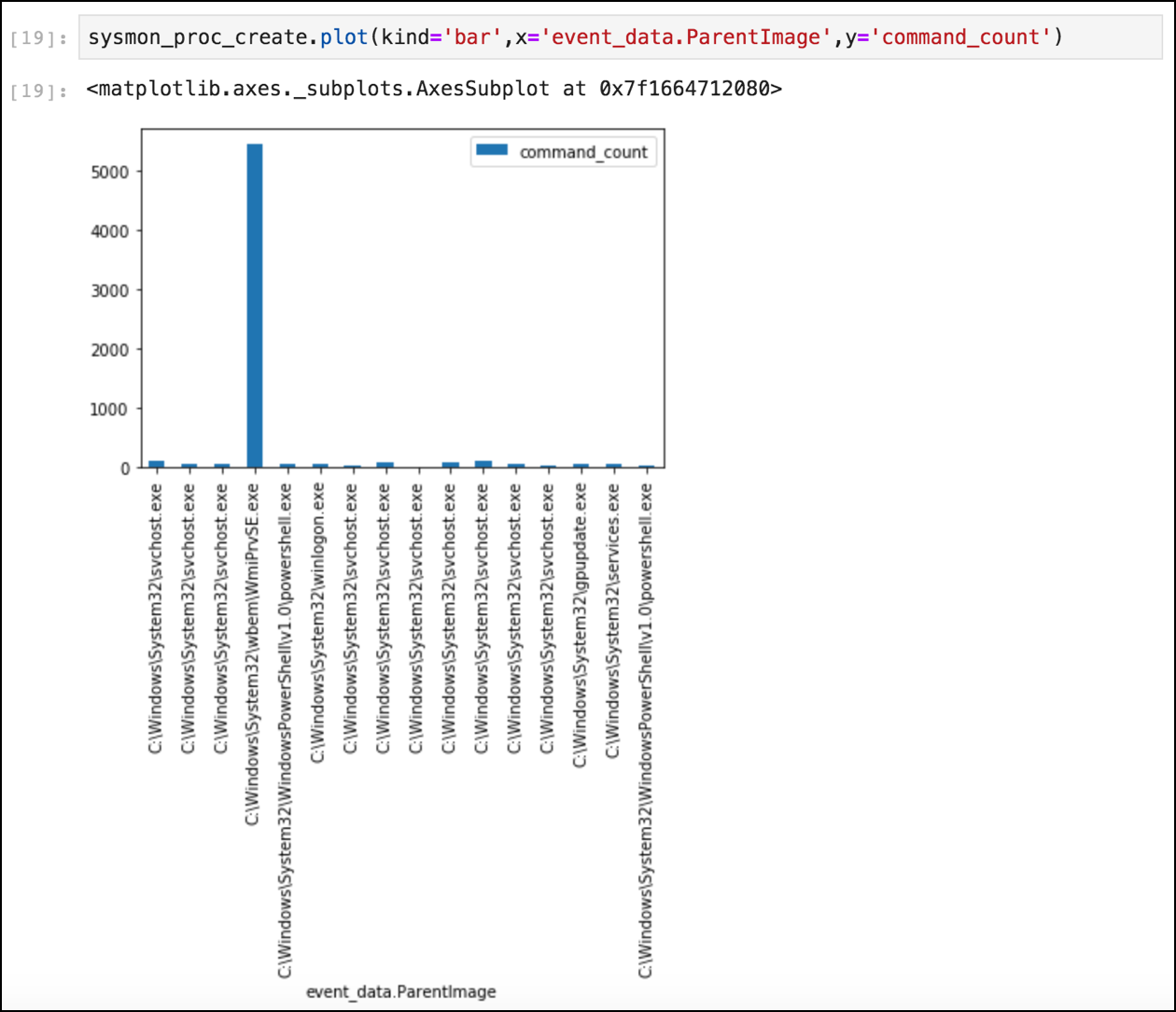




**Визуализируйте данные**

* Мы можем взять значения нового поля из каждой записи и визуализировать их с помощью базовой гистограммы.
* Можно сказать, что C:\Windows\System32\Wbem\WmiPrvSE.exe выполнил какое-либо действие с большим количеством символов в аргументах CommandLine по сравнению с другими процессами в сети.

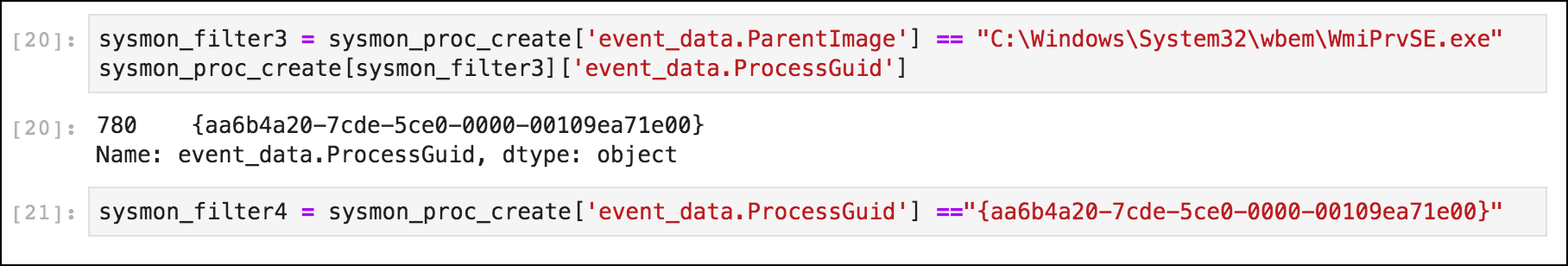
sysmon\_proc\_create.plot(kind='bar',x='event\_data.ParentImage',y='command\_count')



**Отфильтруйте по интересному событию**

* Мы можем пойти дальше и узнать больше о дочерних процессах, ответственных за большое количество символов в аргументах CommandLine.
* Затем можно взять ProcessGuid дочернего процесса и найти его.

sysmon\_filter3 = sysmon\_proc\_create['event\_data.ParentImage'] == "C:\Windows\System32\wbem\WmiPrvSE.exe"  
sysmon\_proc\_create[sysmon\_filter3]['event\_data.ProcessGuid']



**Покажите результаты**

* Для увеличения ширины столбцов наших результатов можно использовать Pandas option\_context manager, чтобы увидеть большие аргументы CommandLine.
* Потенциально мы нашли нашего злоумышленника.

with pd.option\_context('display.max\_colwidth', 2000):  
print(sysmon\_proc\_create[sysmon\_filter4]['event\_data.CommandLine'])



Я надеюсь, что эта статья была полезна для тех, кто только начинает использовать Jupyter Notebook. Кроме того, я надеюсь, что у вас появится интерес к работе с DataFrames для представления и анализа журналов событий безопасности.

Если вы хотите больше погрузиться в изучение дополнительных возможностей, предоставляемых Pandas, запустите доступный блокнот 03-intro-to-pandas. Также можно открыть блокнот basic\_event\_log\_analysis\_pandas, если хотите повторить каждый шаг, представленный в этой статье, и сравнить результаты.

В следующей статье мы продолжим анализировать журналы событий безопасности в табличном формате, но непосредственно из базы данных, а не из автономного файла JSON. Мы по-прежнему будем использовать тот же набор данных Mordor, но я сделаю так, чтобы данные поступали в реальном времени, были нормализованы и сохранились в базе данных Elasticsearch. Затем я покажу вам, как подключиться к базе данных Elasticsearch через Apache Spark для параллельной обработки данных, которые меняются в масштабах.

**Ссылки**

* <https://github.com/pandas-dev/pandas>
* <https://medium.com/@ericvanrees/pandas-series-objects-and-numpy-arrays-15dfe05919d7>
* McKinney, Wes. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O’Reilly Media. Kindle Edition
* <https://github.com/Cyb3rWard0g/OSSEM>
* <https://github.com/Cyb3rWard0g/mordor>
* <https://pandas.pydata.org/pandas-docs/stable/index.html>