

Original Article

Data Visualization with Python Pragmatic Eyes

Sameer Shukla

System Architect, HCL America

6201 Breeze Bay Pt, Apt 1634, Fort Worth, Texas (USA) - 76131

Abstract - Data Visualization helps understand the data's significance to technical and non-technical people. As we know, the data volume in today's scenario is huge in Petabytes. Architectures are quite different from before to support such huge data volume, and distributed systems play an important role. This paper focuses on adapting Visualization techniques in every phase of the Software Development Process, which can be built using Python and some graphical libraries like Matplotlib and Seaborn. We can find problems in our data model or event design using this.



Keywords - Microservices, Python, Visualization, Kafka, Cassandra, Kafka-Python, Matplotlib.

I. INTRODUCTION

Modern world applications are expected to be highly available and responsive and should be able to deal with Gigabytes, Terabytes, and Petabytes of data. To deal with such expectations, companies are shifting architecture from Monolith to Microservices and trying to scale individual services and incorporating distributed systems into their architecture, such as Kafka, which is required for the interaction between various services in the system. Even the database selection choice is also distributed, such as Cassandra. A picture is worth a thousand words; data visualization plays an important role as you can visually see the huge amounts of data in

graphs and charts and quickly make out where your application is going. Data visualization during the development phase also plays a vital role as you can find some critical problems in the systems. In this article, I am trying to showcase the importance of visualization during the development phase and other phases using Python and graphical libraries like Matplotlib, Seaborn, and Bokeh.

II. VISUALIZATION TOOLS AND LIBRARIES



Various lightweight graph libraries are available in the python world, such as Matplotlib, Seaborn, Bokeh, etc. These three are the most widely used; it doesn't matter whether the application you are developing is written in Java, Scala, or any language. For data visualization, I recommend using Python with the graph libraries mentioned above. The target here is to create a Visualization tool in very fewer lines of code irrespective of the language in which the application is written.

A. Python

We are going to use Python for our Visualization purpose, and the reason is plain and simple; here, I don't want to write hundreds of lines of code, and I don't want to write HTML, CSS, and bloat my code with JavaScript. Because my target here is to develop



a tool for debugging purposes and check application behavior with the kind of data I am receiving in the form of Pie Charts, Bar Graphs, Scatter Plots, etc., Python and the integration of graphical libraries like Matplotlib, Seaborn, and Bokeh are very simple. We can generate graphs and charts in less than 10 lines of code, irrespective of whether it is a Microservices environment, Cassandra, or Spark.

B. Matplotlib

It's a huge 2D plotting library, and you need to trial and error to finalize the plot of your use. For example, if you want to check the state-wise sales, you can go for bar graphs or pie charts; it depends on what kind of data you want to visualize. If you check the matplotlib gallery, you can find various charts and graphs like Line, Bars, Markers, Statistical plots, pies, polar charts, etc.

C. Seaborn

Seaborn is also a Python data visualization library that is based on Matplotlib. You can have statistical graphics with Seaborn.

D. Bokeh

Again, a Python-based data visualization library, and you can have interactive graphs. Bokeh renders its graphics using HTML and JavaScript, making it a great candidate for building web-based interactive dashboards.

III. USE CASE FOR VISUALIZATION: MICROSERVICES

Companies are developing microservices based on Kafka; each Microservice communicates with each other with the help of Kafka. Imagine one service writing messages on a Kafka topic, and other services will consume messages from the same topic. With Kafka in place, you can scale your microservices by simply increasing the number of brokers or by increasing the number of Partitions. You can develop microservice with the help of Kafka streams as well, where you can pull the entire data set from Kafka, then apply any reduce or map function according to your requirement and redirect the modified message to another topic for other services to consume

A. Advantage of Using Kafka

Scalability: In case of any node failure, Kafka quickly recovers itself, or you can add more brokers for better scalability.

High Performance: Since Kafka is distributed, high performance and scalability come naturally in using Kafka

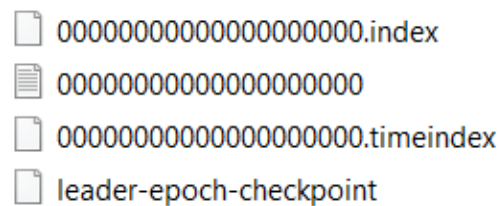
B. What is Kafka, and How data is written in Kafka

Kafka is a publish-subscribe messaging system; this use case is widely adopted in microservices development, where all the tiny microservices talk to each other with the help of Kafka; one microservice

can publish a message to another service to consume messages from the same topic.

C. Pragmatic Introduction to Kafka

Kafka is an open-source distributed data streaming platform, and it was developed at Linked. Over the period, Kafka evolved a lot and established itself as a tool for building a real-time data pipeline. There are various use cases where Kafka can be used; some are Messaging, Log Aggregation, Website Activity Tracking, Stream Processing, etc. Kafka's storage unit is a partition, and a partition is an immutable and ordered sequence of messages where every new message will be appended to the end. A partition is split across the multiple brokers, and the partition is simply a flat file on the disk. When writing happens in Kafka, it writes to an active segment; once the segment is full, a new segment is opened, which becomes the currently active segment.



```
00000000000000000000.index
00000000000000000000
00000000000000000000.timeindex
leader-epoch-checkpoint
```

In the image above, "00000000" is a segment this is where the messages written to Kafka are stored, ". index" file maintains the offset to the position in the segment.log file

D. VISUALIZE

Assuming we are developing microservices for the e-commerce application, we want to visualize the sales data based on State names and the number of units sold in each state. Our tool will be consuming messages with a different group-id from the topic in the form of Key-Value pair, ensuring that we are not touching the critical path in the microservices development pipeline, where Key is the name of State and Value is the number of units sold. For our tool development, we will be using a library based on Python named Kafka-Python and incorporating Matplotlib, Seaborn, or Bokeh, any visualization library of choice. Let's explore the code.

```
1 from kafka import KafkaConsumer
2
3 consumer = KafkaConsumer(
4     'activity',
5     bootstrap_servers=['localhost:9092'],
6     auto_offset_reset='earliest',
7     enable_auto_commit=True,
8     group_id='test-grp')
9
```

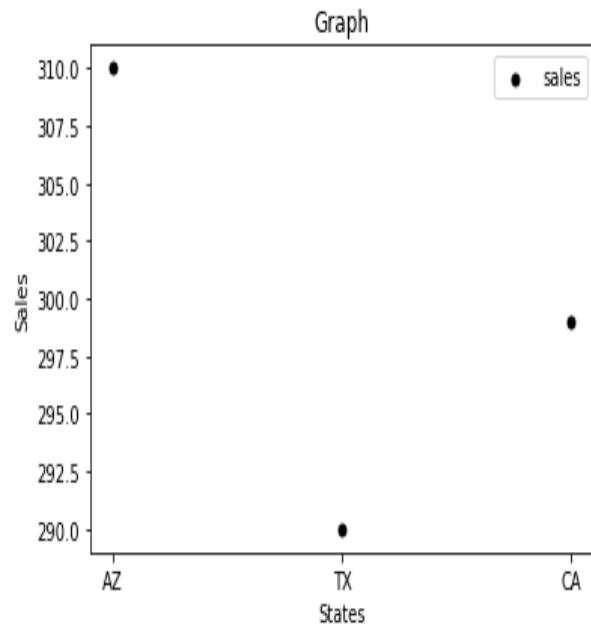
1. In the code above, we have imported KafkaConsumer as we will consume messages from the topic.
2. From line 3, we instantiated a consumer that takes a topic name 'activity,' server name, which is our localhost (Kafka Server running locally). The other 3 properties are Kafka-specific, indicating that we will start consuming messages from the earliest offset. Once the message is consumed, it is marked for auto-commit and gives a group id to our application.

```
import matplotlib.pyplot as plt
consumer = KafkaConsumer('activity', group_id=None)
state=[]
sales=[]
```

3. We are importing Matplotlib for plotting purposes and instantiating consumers along with lists, one representing state and the other representing sales.
4. The Last step is where we will iterate through the messages, and using the scatter plots, we will plot our data.

```
10 import matplotlib.pyplot as plt
11 consumer = KafkaConsumer('activity', group_id=None)
12 state=[]
13 sales=[]
14 for msg in consumer:
15     state.append(msg.key.decode())
16     sales.append(int(msg.value.decode()))
17     plt.scatter(state,sales, label='sales', color='k', s=25, marker='o')
18     plt.xlabel('States')
19     plt.ylabel('Sales')
20     plt.title('Graph')
21     plt.legend()
22     plt.show()
23
```

The X-axis indicates the state names, and Y-axis indicates Sales; let's see the graph for the following inputs TX;290, AZ; 310, CA; 299, WA; 301



IV. USE CASE FOR VISUALIZATION: CASSANDRA



Cassandra is a distributed and decentralized database. It is scalable and has no single point of failure; it is elastic and scalable. Distributed architecture and systems are the future because it is independently scalable and always available. We can easily add or remove nodes in Cassandra Cluster when the volume of data goes up or down. Cassandra's data will be queried based on the Partitioning Key column, and it's a Primary Key in Cassandra. You can consider it a Key in SortedMap;

the Key is nothing but a hash value. When the data is written in Cassandra, the Partitioner generates a hash value (token). This hash identifies the node/Partition to which the data will reside.

Similarly, during querying the data again, Hash Value (Token) will be calculated, Node/Partition will be identified, and data returned to the user. We need to select the Partitioning key column judiciously. We need to make sure that the Partitioning key is evenly distributed. Else we will have an unbalanced cluster, and the performance will be impacted big time. To avoid this, we need to ensure that our data model is correct and our configurations. This unbalanced Cluster problem is not easily identified because in Cassandra, aggregate functions like count (*) are not allowed, by allowed here, I mean if we have lots of data in our system, say a million records count (*) query will timeout, so there is no way to find this problem until our read queries are impacted. This one problem can be solved through our visualization tool, and we can have a graph/chart in place. It can very well inform us how our data is distributed and what type of data is in the system. I mean everything from Partitioning keys to clustering keys and other columns by data here.

A. Use case

We will explore an e-commerce application where we will see state/country-wise sales data in Bar graphs and Pie charts. We will see the top 10 states with the greatest number of sales through the Bar Graph, and with the Pie chart, we will see how much percentage of data each state/country has in the system.

B. Cassandra restrictions

- a. In Cassandra, we should not execute aggregate functions in the query like count (*) plus functions like Group By are not allowed in Cassandra
- b. Because of its architecture, no Like queries are allowed because Partitioner generates a token value, and we cannot simply fire like query on tokens.
- c. No range query on Partitioning key, again because Partitions are nothing but tokens and are unordered, because of this, we cannot execute range queries on Partitioning key. No Sorting because data in Cassandra is scattered to SSTables, and sorting means reading all those SSTables and executing our sort; it's a very heavy process that's why it's not recommended to use sorting in Cassandra.

Because of these limitations, we cannot directly use something like Python-Cassandra and execute our visualization tool; the approach that we should follow here is to pull the data from Cassandra to a flat file (sales.csv) and give this file as an input to Pandas (Python library for data analysis) and then we can

execute our data visualization after sorting, filtering and all the other necessary operations. It's a slightly off routine process.

C. Pandas



Pandas is an excellent library for data analysis; some of its features mentioned below make it an excellent package for data analysis.

- a. Allows the use of rows and labels
- b. Easy handling of NaN values.
- c. Can load data from different formats into Data Frames.
- d. It can merge different datasets.

Super easy integration with Matplotlib and Seaborn

D. Checking Country wise sales.

First, we will organize our data.

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
In [12]: # Read file to Dataframe
df = pd.read_csv('sales.txt', sep=';', names=['country', 'productid', 'units sold'])
df.head()
```

```
Out[12]:
```

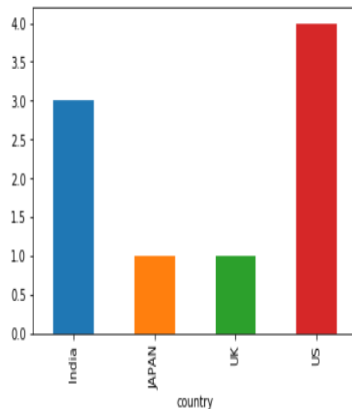
	country	productid	units sold
0	India	2090	20
1	US	2032	100
2	India	3003	30
3	US	4032	10
4	UK	3024	34

As you can see above, we have used Pandas; the three imports you can see above are for NumPy, pandas, and matplotlib. NumPy is the package for

scientific computing, Pandas is our library right now for data analysis purposes, and matplotlib is used for plotting purposes. First, we will read the data into the data frame and give names to our columns, and in the second line, `df.head()` we are just checking our data; the `head()` method will print the first 5 lines from the file. Now we are all set to perform a group-by, sort kind of aggregate operations and check them in the graphs and charts for our visualization purposes.

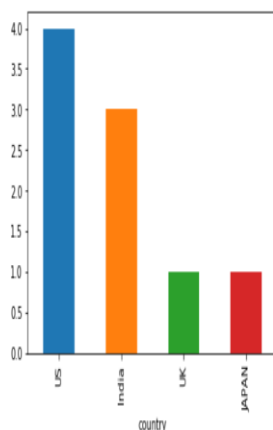
```
In [21]: # Country wise sales
grpByCountry = df.groupby('country').size()
```

```
In [22]: grpByCountry.plot(kind='bar',x='country',y='units sold')
plt.show()
```



As you can see in line 21*, we are grouping the data by country and checking the total sales done against each country, but these bars are random. Let's sort them in descending order and plot them again.

```
In [24]: grpByCountry.sort_values(ascending=False).plot(kind='bar',x='country',y='units sold')
plt.show()
```

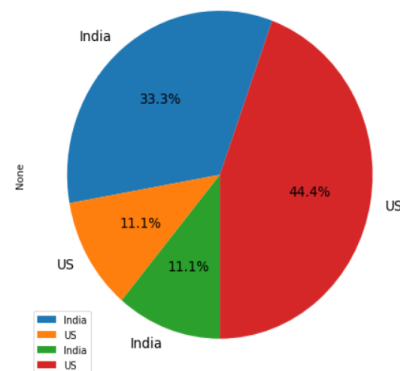


We simply sort the grab country data frame in descending order and plot it. Look at the ease of use these libraries bring into the picture, and we are not writings here hundreds of lines of code; we are not dealing with div issues in HTML or styling issues in CSS no JavaScript errors we see here; we are simply able to focus on our goal of Visualizing the data in the system. Pie Charts: If we want to check how

much % sales are done by each country, we can use pie charts; during the service or any application development, you have all the internal data with you, and you can very well check the data distribution in percentages, which gives you a clear idea whether your Partitioning key is balanced or not or exactly how your data is distributed in the system.

```
In [53]: plt.figure(figsize=(16,8))
ax1 = plt.subplot(121, aspect='equal')
grpByCountry.plot(kind='pie', y = 'units sold', ax=ax1, autopct='%1.1f%%',
startangle=70, shadow=False, labels=df['country'], Legend = True, fontsize=14)

ax2 = plt.subplot(122)
plt.axis('off')
plt.show()
```



V. CONCLUSION

Here I have tried to show how we can visualize the data during the development phase only; I have not written any fancy JS, HTML, or CSS files or not wasted hours and days on the development of such utility. It doesn't matter whether we are developing microservices using Play, Scala, or Java, SpringBoot we can very easily use Python and its charting libraries for our debugging or identifying the data patterns in the application purposes.

REFERENCES

- [1] Kafka-Python: <https://docs.confluent.io/4.1.2/clients/confluent-kafka-python/index.html>
- [2] Cassandra: <https://docs.confluent.io/4.1.2/clients/confluent-kafka-python/index.html>
- [3] Matplotlib: <https://matplotlib.org/gallery.html>
- [4] Images: <https://unsplash.com>
- [5] Apache Kafka: <https://kafka.apache.org/>
- [6] Pandas: <https://pandas.pydata.org/>
- [7] Seaborn: <https://seaborn.pydata.org/>
- [8] Bokeh: <https://bokeh.pydata.org/en/latest/>