Rolling Correlation Analysis of Stock Market Indices Using Big Data Tools

1.

Introduction

What is a Stock Market Index?

- A number calculated based on share prices of a list of important companies
- A representation for the market performance

What is a Correlation Coefficient?

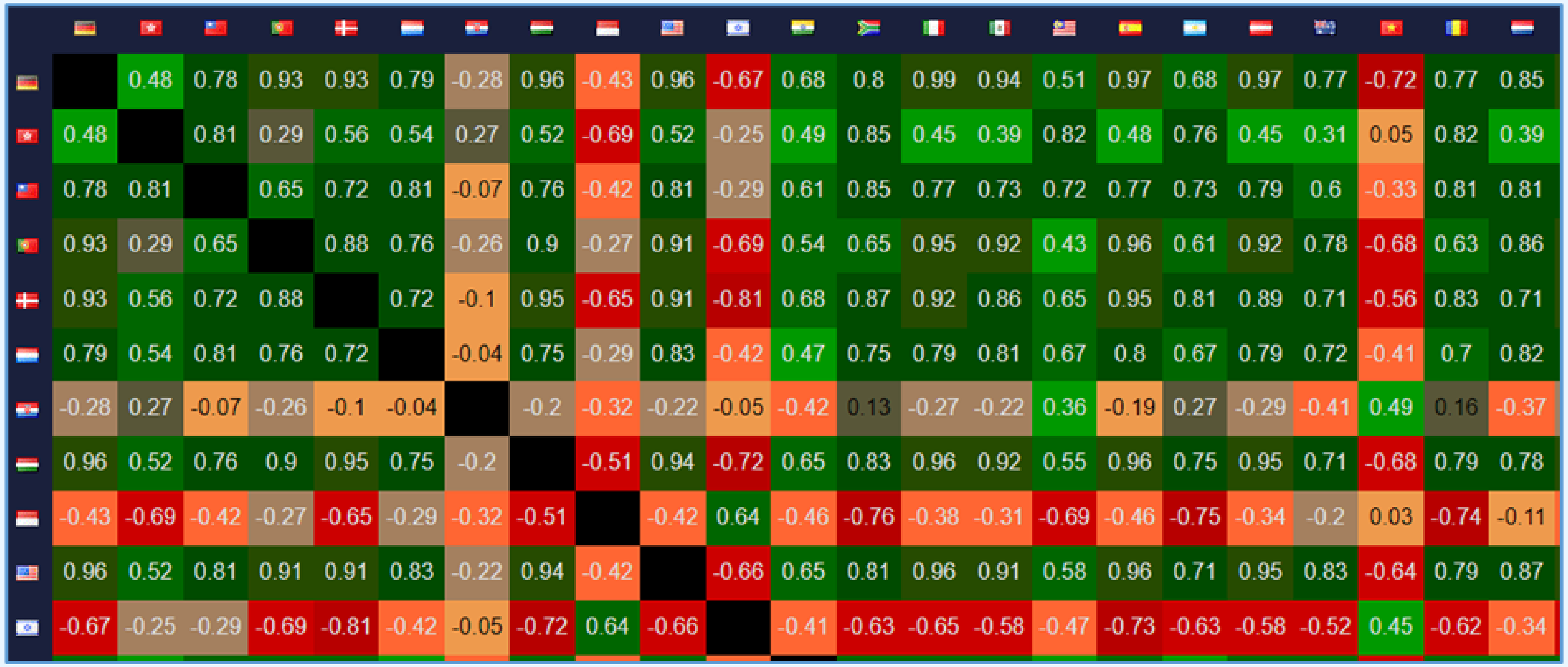
- A metric to measure the strength of two indices moving in tandem with one another
- Cor. coef. = 1 means two indices will move by the same amount, and in the same direction
- Cor. coef. = -1 means two indices will move by the same amount, but in the opposite direction

What is a rolling correlation coefficient?

- A rolling correlation, akin to a moving average, is simply the moving correlation coefficient between two time series over a specified window width.

- For example, with a window width of 24 months, the 24-month rolling correlation as at December 2020 is the correlation coefficient between two time series using data from January 2019 until December 2020 (i.e. 24 months).

Example of Correlation Coefficients between Stock Market Indices



	0.48	0.78	0.93	0.93	0.79	-0.28	0.96	-0.43	0.96	-0.67	0.68	0.8	0.99	0.94	0.51	0.97	0.68	0.97	0.77	-0.72	0.77	0.85
0.48		0.81	0.29	0.56	0.54	0.27	0.52	-0.69	0.52	-0.25	0.49	0.85	0.45	0.39	0.82	0.48	0.76	0.45	0.31	0.05	0.82	0.39
0.78	0.81		0.65	0.72	0.81	-0.07	0.76	-0.42	0.81	-0.29	0.61	0.85	0.77	0.73	0.72	0.77	0.73	0.79	0.6	-0.33	0.81	0.81
0.93	0.29	0.65		0.88	0.76	-0.26	0.9	-0.27	0.91	-0.69	0.54	0.65	0.95	0.92	0.43	0.96	0.61	0.92	0.78	-0.68	0.63	0.86
0.93	0.56	0.72	0.88		0.72	-0.1	0.95	-0.65	0.91	-0.81	0.68	0.87	0.92	0.86	0.65	0.95	0.81	0.89	0.71	-0.56	0.83	0.71
0.79	0.54	0.81	0.76	0.72		-0.04	0.75	-0.29	0.83	-0.42	0.47	0.75	0.79	0.81	0.67	0.8	0.67	0.79	0.72	-0.41	0.7	0.82
-0.28	0.27	-0.07	-0.26	-0.1	-0.04		-0.2	-0.32	-0.22	-0.05	-0.42	0.13	-0.27	-0.22	0.36	-0.19	0.27	-0.29	-0.41	0.49	0.16	-0.37
0.96	0.52	0.76	0.9	0.95	0.75	-0.2		-0.51	0.94	-0.72	0.65	0.83	0.96	0.92	0.55	0.96	0.75	0.95	0.71	-0.68	0.79	0.78
-0.43	-0.69	-0.42	-0.27	-0.65	-0.29	-0.32	-0.51		-0.42	0.64	-0.46	-0.76	-0.38	-0.31	-0.69	-0.46	-0.75	-0.34	-0.2	0.03	-0.74	-0.11
0.96	0.52	0.81	0.91	0.91	0.83	-0.22	0.94	-0.42		-0.66	0.65	0.81	0.96	0.91	0.58	0.96	0.71	0.95	0.83	-0.64	0.79	0.87
-0.67	-0.25	-0.29	-0.69	-0.81	-0.42	-0.05	-0.72	0.64	-0.66		-0.41	-0.63	-0.65	-0.58	-0.47	-0.73	-0.63	-0.58	-0.52	0.45	-0.62	-0.34

(Source: Macroaxis, 2022)

Problem Statement & Motivation

- Most publicly available correlation coefficients are static over a specified timeframe, but they can change rapidly (e.g. during financial turmoil)
  → Trend of rolling correlation coefficients for a moving window (e.g. 24 months) could provide additional insights to investors

- Limited past studies on stock market indices using big data tools
  → Use big data tools (Apache Hive and Apache PySpark) to perform rolling correlation analysis on stock market indices

Aim

- To perform rolling correlation analysis between stock market indices using big data tools

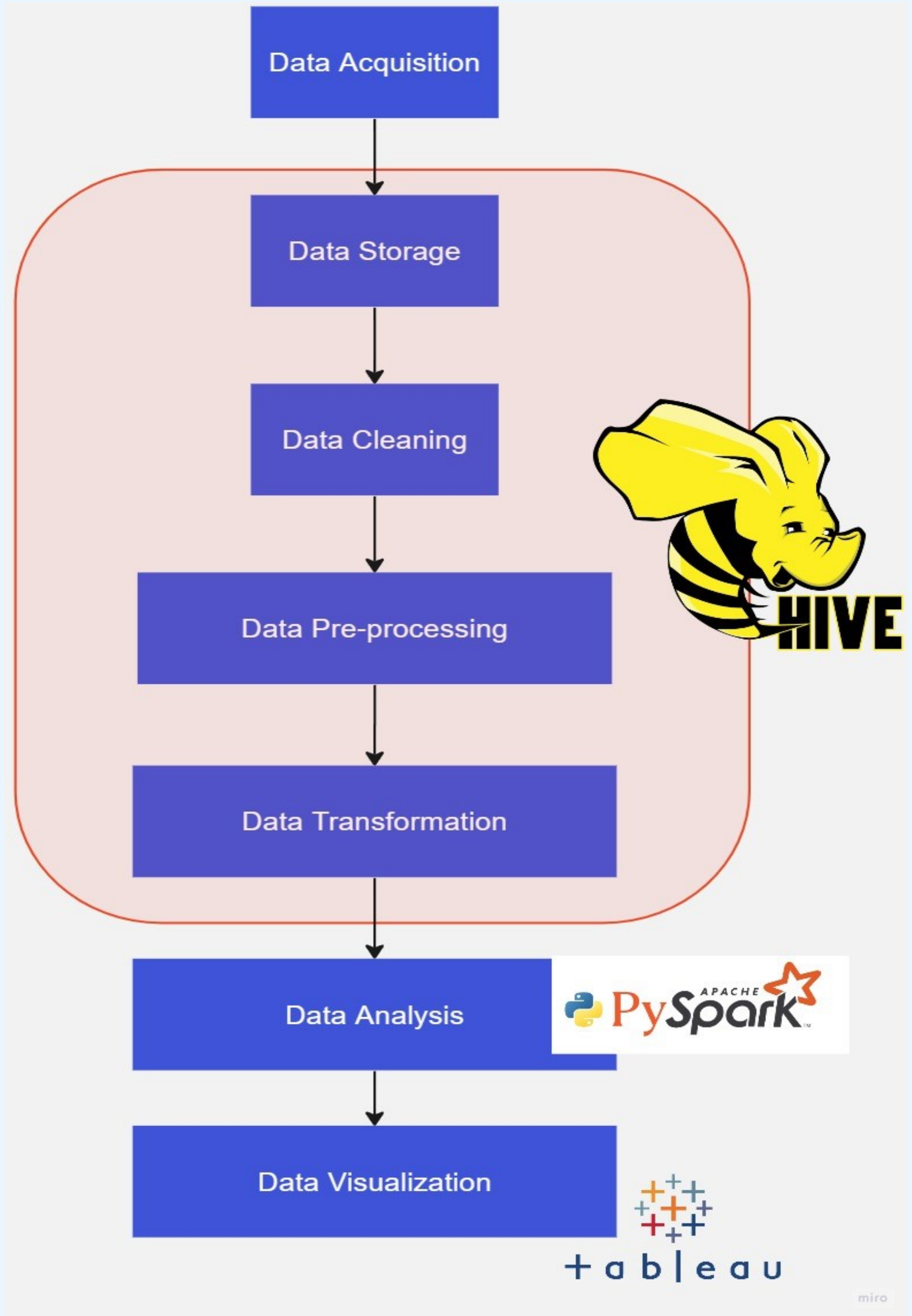
Research Questions & Objectives

Can big data tools be used to perform data access on stock market indices?	To determine the feasibility of selected data access tools for all specified tasks in data cleaning, data pre-processing and data transformation
Can big data tools be used to perform data analysis on stock market indices?	To compute the rolling correlation coefficients between stock market indices using big data tool
How does the correlation between stock markets change over time?	To analyse the rolling correlation coefficients between stock markets

2.

Methodology

Overview


Data Acquisition
Data Storage
Data Cleaning
Data Pre-processing
Data Transformation
Data Analysis
Data Visualization
HIVE
APACHE
PySpark
tableau
miro

Data Acquisition

Description of dataset

Title	Stock Exchange Data
Source	Cody. (2021). Kaggle.
Link	https://www.kaggle.com/datasets/mattiuzc/stock-exchange-data
Period	Year 1965 till 2021
Description	Daily price data for indices tracking stock exchanges from all over the world (United States, China, Canada, Germany, Japan, and more). The data was all collected from Yahoo Finance, which had several decades of data available for most exchanges. Prices are quoted in terms of the national currency of where each exchange is located.
Filename	indexData.csv
Number of rows	112,457 rows
Number of columns	8 columns

Data Acquisition

Contents of dataset

Column	Description	Data type	Range
Index	Ticker symbol for index	String	14 stock markets
Date	Date of observation	Date	05-Jan-1965 to 03-Jun-2021
Open	Opening price	Float	54.87 to 68,775.06
High	Highest price during trading day	Float	54.87 to 69,403.75
Low	Lowest price during trading day	Float	54.87 to 68,516.99
Close	Close price adjusted for splits	Float	54.87 to 68,775.06
Adj Close	Adjusted close price adjusted for both dividends and splits	Float	54.87 to 68,775.06
Volume	Number of shares traded during trading day	Integer	0 to 94,403,740,000

Data Acquisition

Stock market indices dataset

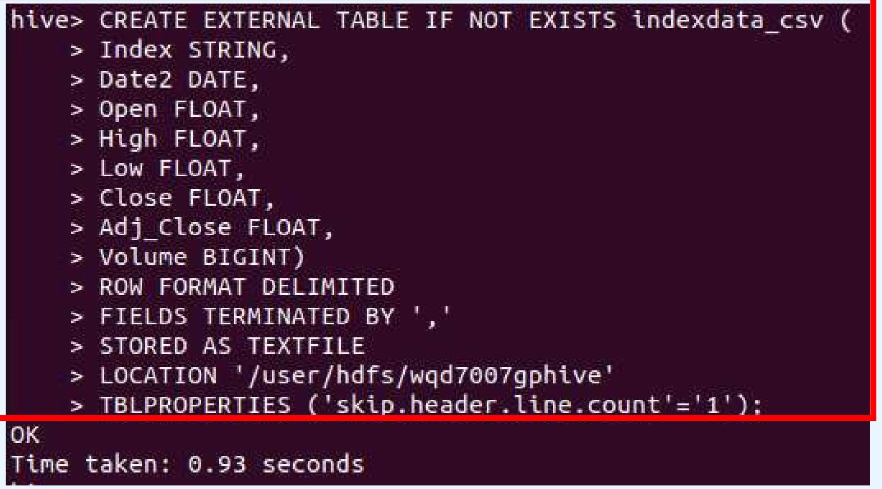
Index symbol	Stock market index
000001.SS	Shanghai Stock Exchange Composite Index, China
399001.SZ	Shenzhen Stock Exchange Component Index, China
GDAXI	DAX Performance Index, Germany
GSPTSE	S&P TSX Composite Index, Canada
HSI	Hang Seng Index, Hong Kong
IXIC	NASDAQ Composite Index, United States
J203.JO	Johannesburg All Share Index, South Africa
KS11	KOSPI Composite Index, Korea
N100	Euronext 100 Index, Europe
N225	Nikkei 225 Index, Japan
NSEI	National Stock Exchange Fifty Index, India
NYA	New York Stock Exchange Composite Index, United States
SSMI	Swiss Market Index, Switzerland
TWII	Taiwan Weighted Index, Taiwan

ETL Process using Apache Hive

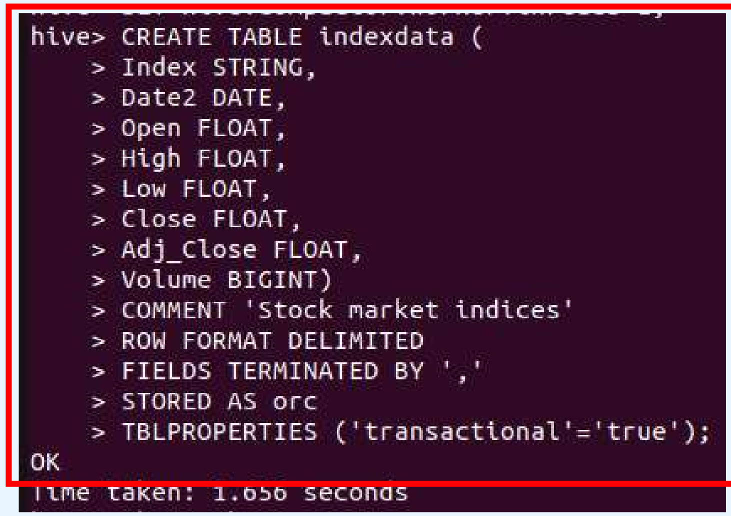
Step	Description
	Data Storage
1	Save raw dataset (indexData.csv) into HDFS
2	Load raw dataset into tool
	Data Cleaning
3	Count rows before deletion
4	Delete rows with missing values
5	Count rows after deletion
6	Delete “Adj Close” and “Volume” columns
	Data Pre-Processing
7	Add “Year”, “Month”, “Day” columns based on “Date” column
8	Add “Year-Month” column
	Data Transformation for Monthly Data
9	For all “Index” names, average all the indices (“Open”, “High”, “Low”, “Close”) by “Year-Month”
10	Query average monthly “Close” column
11	Export queried result as csv

ETL Process using Apache Hive Data Storage

1) Create external table and table in Apache HIVE:

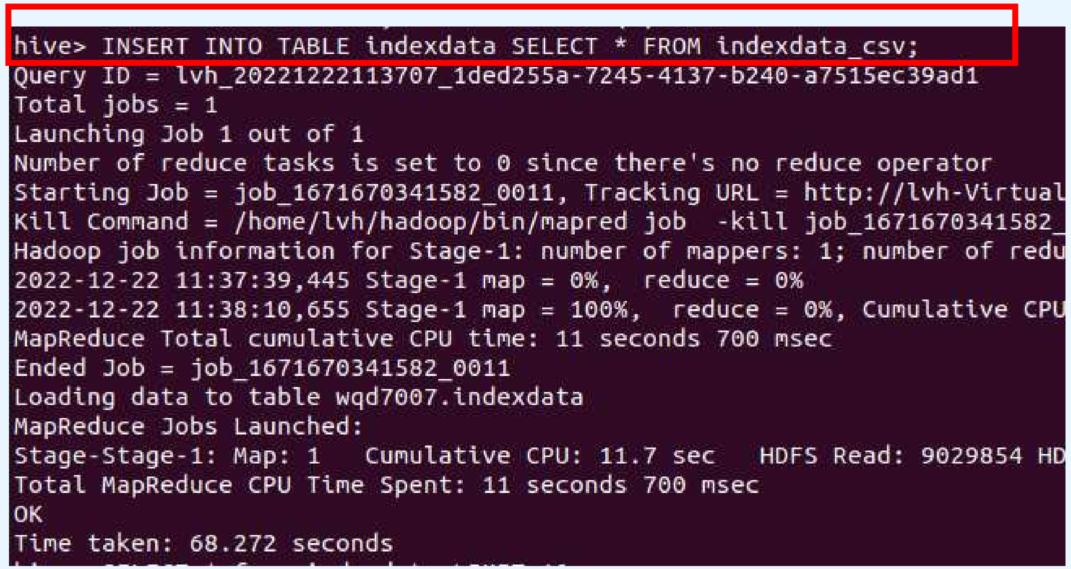


```
hive> CREATE EXTERNAL TABLE IF NOT EXISTS indexdata_csv (
    > Index STRING,
    > Date2 DATE,
    > Open FLOAT,
    > High FLOAT,
    > Low FLOAT,
    > Close FLOAT,
    > Adj_Close FLOAT,
    > Volume BIGINT)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE
    > LOCATION '/user/hdfs/wqd7007gphive'
    > TBLPROPERTIES ('skip.header.line.count'='1');
OK
Time taken: 0.93 seconds
```



```
hive> CREATE TABLE indexdata (
    > Index STRING,
    > Date2 DATE,
    > Open FLOAT,
    > High FLOAT,
    > Low FLOAT,
    > Close FLOAT,
    > Adj_Close FLOAT,
    > Volume BIGINT)
    > COMMENT 'Stock market indices'
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS orc
    > TBLPROPERTIES ('transactional'='true');
OK
Time taken: 1.656 seconds
```

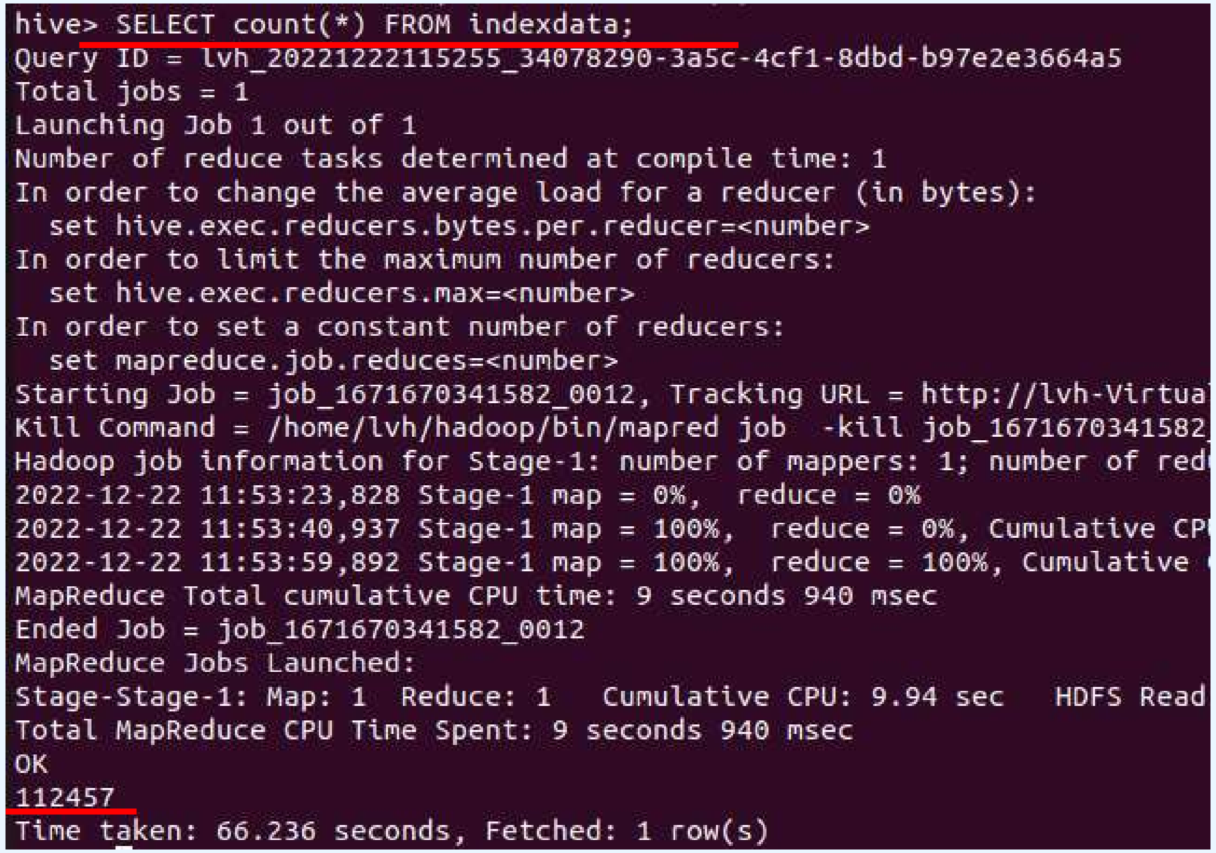
2) Load Raw Dataset into Apache HIVE:



```
hive> INSERT INTO TABLE indexdata SELECT * FROM indexdata_csv;
Query ID = lvh_20221222113707_1ded255a-7245-4137-b240-a7515ec39ad1
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1671670341582_0011, Tracking URL = http://lvh-Virtual
Kill Command = /home/lvh/hadoop/bin/mapred job  -kill job_1671670341582_
Hadoop job information for Stage-1: number of mappers: 1; number of redu
2022-12-22 11:37:39,445 Stage-1 map = 0%,  reduce = 0%
2022-12-22 11:38:10,655 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU
MapReduce Total cumulative CPU time: 11 seconds 700 msec
Ended Job = job_1671670341582_0011
Loading data to table wqd7007.indexdata
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 11.7 sec   HDFS Read: 9029854 HD
Total MapReduce CPU Time Spent: 11 seconds 700 msec
OK
Time taken: 68.272 seconds
```

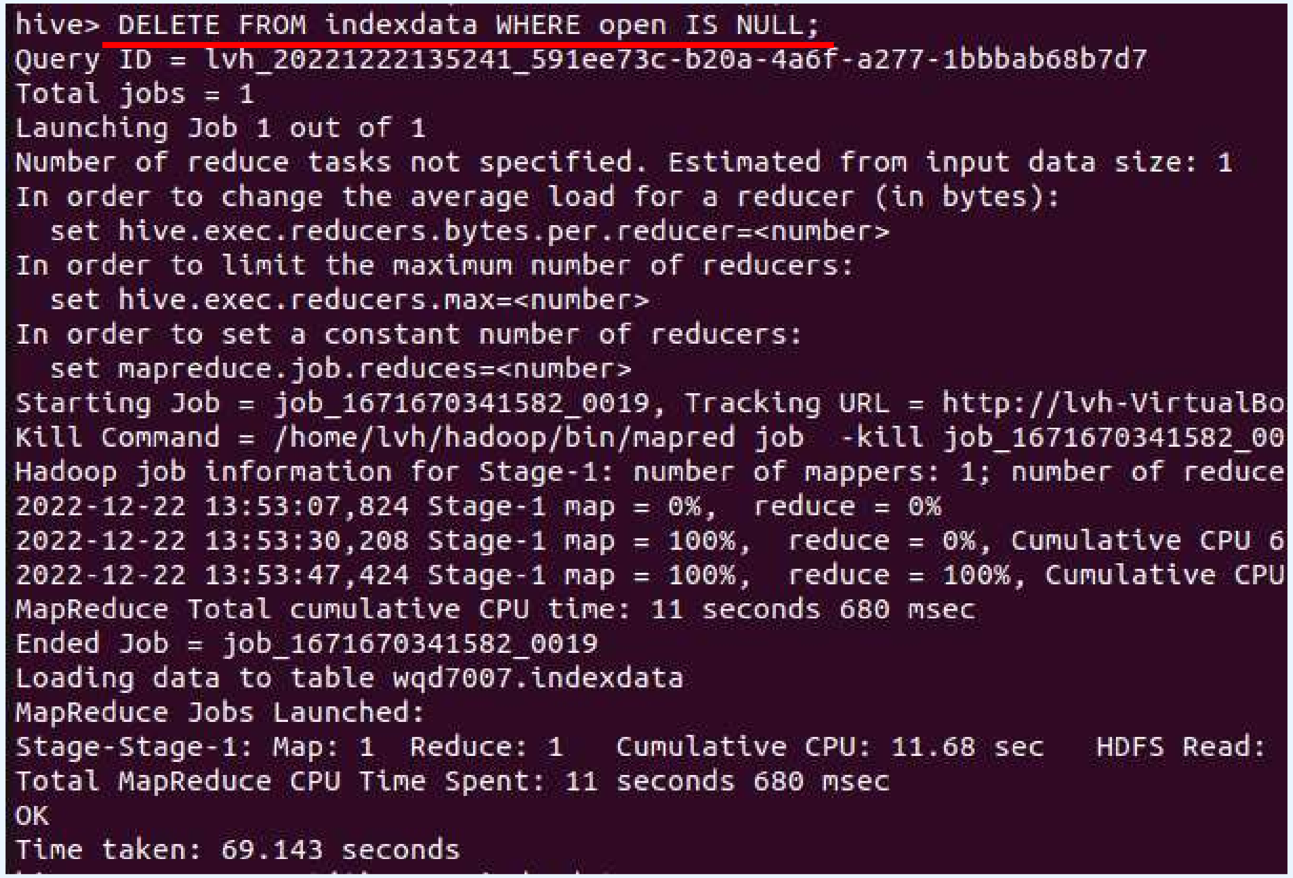
ETL Process using Apache Hive
Data Cleaning

3) Count rows before deletion [112,457 rows]



```
hive> SELECT count(*) FROM indexdata;
Query ID = lvh_20221222115255_34078290-3a5c-4cf1-8dbd-b97e2e3664a5
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1671670341582_0012, Tracking URL = http://lvh-Virtual
Kill Command = /home/lvh/hadoop/bin/mapred job  -kill job_1671670341582
Hadoop job information for Stage-1: number of mappers: 1; number of red
2022-12-22 11:53:23,828 Stage-1 map = 0%,  reduce = 0%
2022-12-22 11:53:40,937 Stage-1 map = 100%,  reduce = 0%, Cumulative CP
2022-12-22 11:53:59,892 Stage-1 map = 100%,  reduce = 100%, Cumulative
MapReduce Total cumulative CPU time: 9 seconds 940 msec
Ended Job = job_1671670341582_0012
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 9.94 sec   HDFS Read
Total MapReduce CPU Time Spent: 9 seconds 940 msec
OK
112457
Time taken: 66.236 seconds, Fetched: 1 row(s)
```

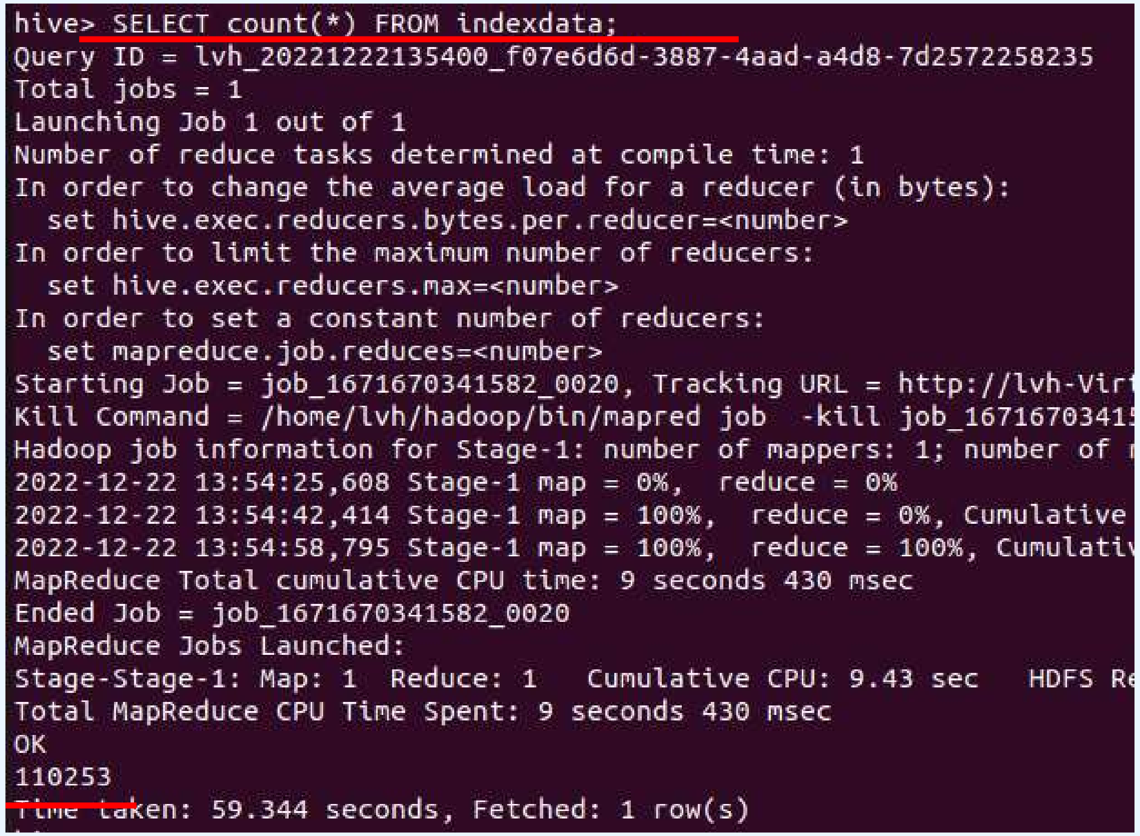
4) Delete rows with missing values,



```
hive> DELETE FROM indexdata WHERE open IS NULL;
Query ID = lvh_20221222135241_591ee73c-b20a-4a6f-a277-1bbbab68b7d7
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1671670341582_0019, Tracking URL = http://lvh-VirtualBo
Kill Command = /home/lvh/hadoop/bin/mapred job  -kill job_1671670341582_00
Hadoop job information for Stage-1: number of mappers: 1; number of reduce
2022-12-22 13:53:07,824 Stage-1 map = 0%,  reduce = 0%
2022-12-22 13:53:30,208 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 6
2022-12-22 13:53:47,424 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU
MapReduce Total cumulative CPU time: 11 seconds 680 msec
Ended Job = job_1671670341582_0019
Loading data to table wqd7007.indexdata
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 11.68 sec   HDFS Read:
Total MapReduce CPU Time Spent: 11 seconds 680 msec
OK
Time taken: 69.143 seconds
```

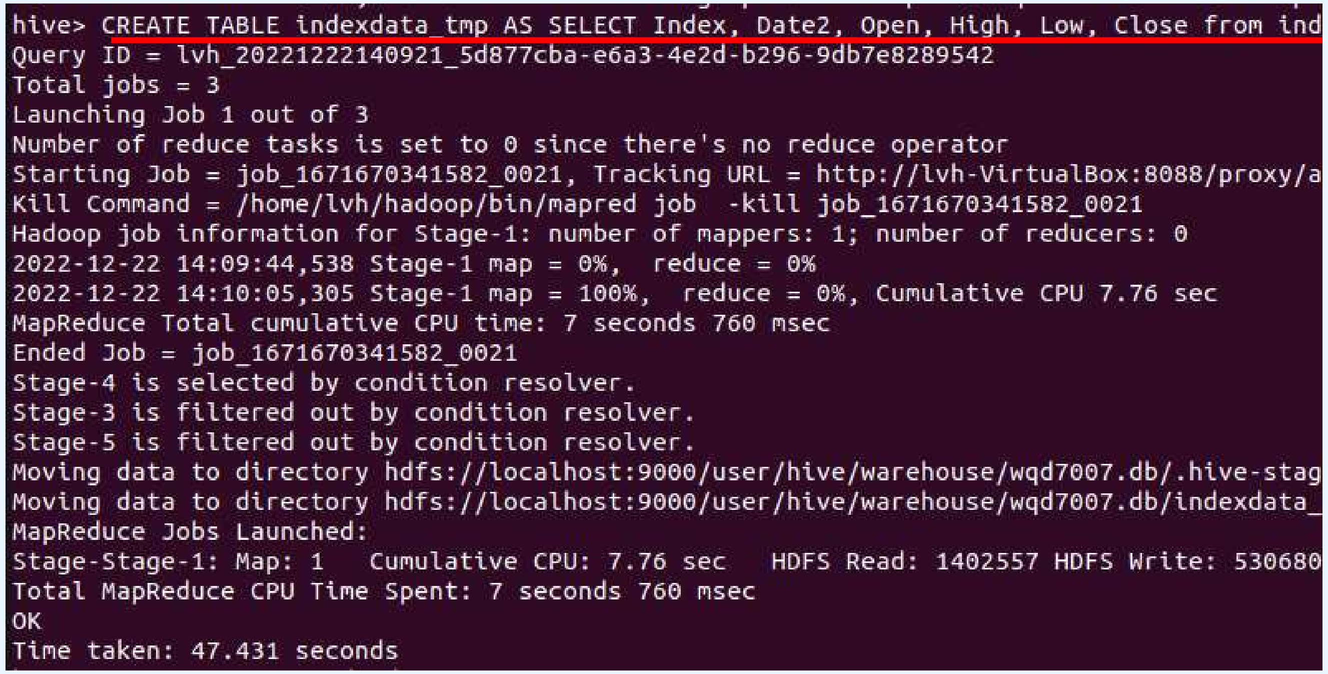
ETL Process using Apache Hive
Data Cleaning

5) Count rows after deletion [110,253 rows]



```
hive> SELECT count(*) FROM indexdata;
Query ID = lvh_20221222135400_f07e6d6d-3887-4aad-a4d8-7d2572258235
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1671670341582_0020, Tracking URL = http://lvh-Virt
Kill Command = /home/lvh/hadoop/bin/mapred job  -kill job_16716703415
Hadoop job information for Stage-1: number of mappers: 1; number of r
2022-12-22 13:54:25,608 Stage-1 map = 0%,  reduce = 0%
2022-12-22 13:54:42,414 Stage-1 map = 100%,  reduce = 0%, Cumulative
2022-12-22 13:54:58,795 Stage-1 map = 100%,  reduce = 100%, Cumulativ
MapReduce Total cumulative CPU time: 9 seconds 430 msec
Ended Job = job_1671670341582_0020
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 9.43 sec   HDFS Re
Total MapReduce CPU Time Spent: 9 seconds 430 msec
OK
110253
Time taken: 59.344 seconds, Fetched: 1 row(s)
```

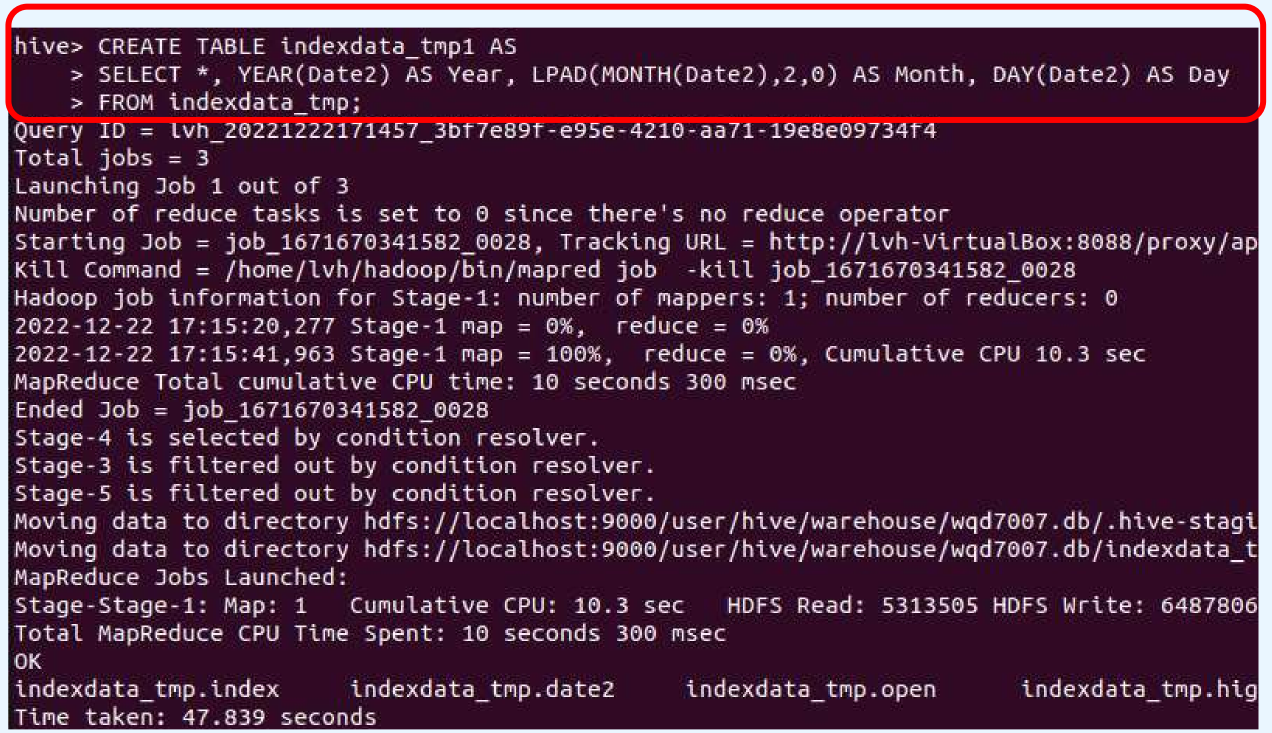
6) Delete “Adj Close” and “Volume” columns



```
hive> CREATE TABLE indexdata_tmp AS SELECT Index, Date2, Open, High, Low, Close from ind
Query ID = lvh_20221222140921_5d877cba-e6a3-4e2d-b296-9db7e8289542
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1671670341582_0021, Tracking URL = http://lvh-VirtualBox:8088/proxy/a
Kill Command = /home/lvh/hadoop/bin/mapred job  -kill job_1671670341582_0021
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2022-12-22 14:09:44,538 Stage-1 map = 0%,  reduce = 0%
2022-12-22 14:10:05,305 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 7.76 sec
MapReduce Total cumulative CPU time: 7 seconds 760 msec
Ended Job = job_1671670341582_0021
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to directory hdfs://localhost:9000/user/hive/warehouse/wqd7007.db/.hive-stag
Moving data to directory hdfs://localhost:9000/user/hive/warehouse/wqd7007.db/indexdata_
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 7.76 sec   HDFS Read: 1402557 HDFS Write: 530680
Total MapReduce CPU Time Spent: 7 seconds 760 msec
OK
Time taken: 47.431 seconds
```

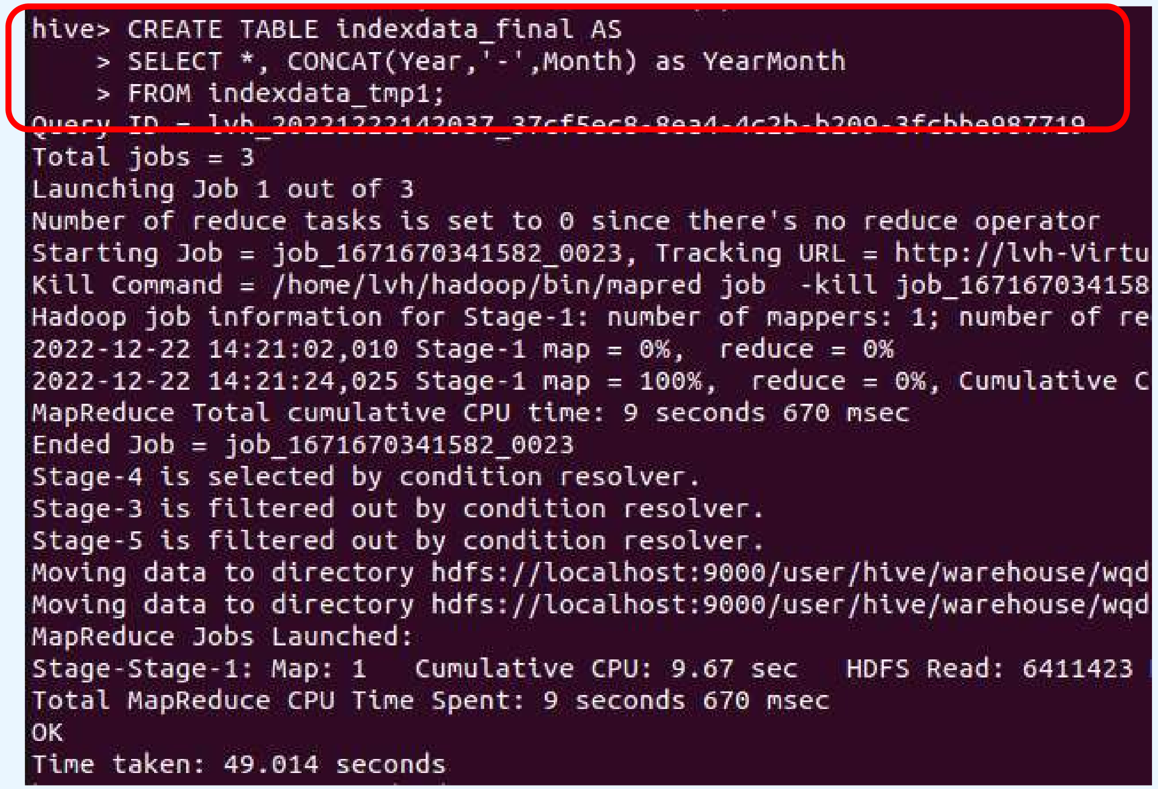
ETL Process using Apache Hive
Data Preprocessing

7) Add “Year”, “Month”, “Day” columns based on “Date” column



```
hive> CREATE TABLE indexdata_tmp1 AS
    > SELECT *, YEAR(Date2) AS Year, LPAD(MONTH(Date2),2,0) AS Month, DAY(Date2) AS Day
    > FROM indexdata_tmp;
Query ID = lvh_20221222171457_3bf7e89f-e95e-4210-aa71-19e8e09734f4
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1671670341582_0028, Tracking URL = http://lvh-VirtualBox:8088/proxy/ap
Kill Command = /home/lvh/hadoop/bin/mapred job  -kill job_1671670341582_0028
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2022-12-22 17:15:20,277 Stage-1 map = 0%,  reduce = 0%
2022-12-22 17:15:41,963 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 10.3 sec
MapReduce Total cumulative CPU time: 10 seconds 300 msec
Ended Job = job_1671670341582_0028
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to directory hdfs://localhost:9000/user/hive/warehouse/wqd7007.db/.hive-stagi
Moving data to directory hdfs://localhost:9000/user/hive/warehouse/wqd7007.db/indexdata_t
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 10.3 sec   HDFS Read: 5313505 HDFS Write: 6487806
Total MapReduce CPU Time Spent: 10 seconds 300 msec
OK
indexdata_tmp.index     indexdata_tmp.date2     indexdata_tmp.open      indexdata_tmp.hig
Time taken: 47.839 seconds
```

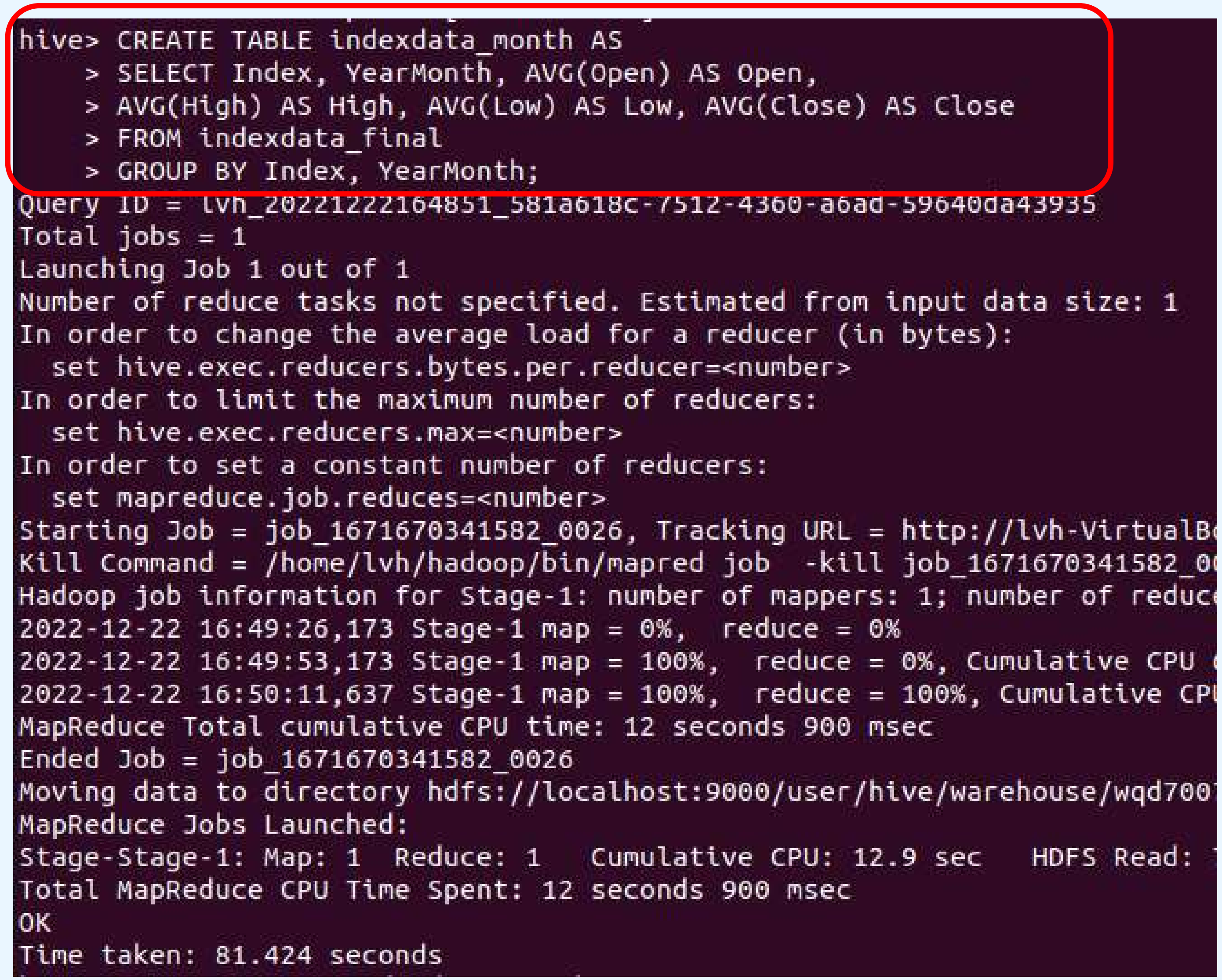
8) Add “Year-Month” column



```
hive> CREATE TABLE indexdata_final AS
    > SELECT *, CONCAT(Year,'-',Month) as YearMonth
    > FROM indexdata_tmp1;
Query ID = lvh_20221222142037_37cf5ec8-8ea4-4c2b-b209-3fcbbe987719
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1671670341582_0023, Tracking URL = http://lvh-Virtu
Kill Command = /home/lvh/hadoop/bin/mapred job  -kill job_167167034158
Hadoop job information for Stage-1: number of mappers: 1; number of re
2022-12-22 14:21:02,010 Stage-1 map = 0%,  reduce = 0%
2022-12-22 14:21:24,025 Stage-1 map = 100%,  reduce = 0%, Cumulative C
MapReduce Total cumulative CPU time: 9 seconds 670 msec
Ended Job = job_1671670341582_0023
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to directory hdfs://localhost:9000/user/hive/warehouse/wqd
Moving data to directory hdfs://localhost:9000/user/hive/warehouse/wqd
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 9.67 sec   HDFS Read: 6411423
Total MapReduce CPU Time Spent: 9 seconds 670 msec
OK
Time taken: 49.014 seconds
```

ETL Process using Apache Hive
Data Transformation for Monthly Data

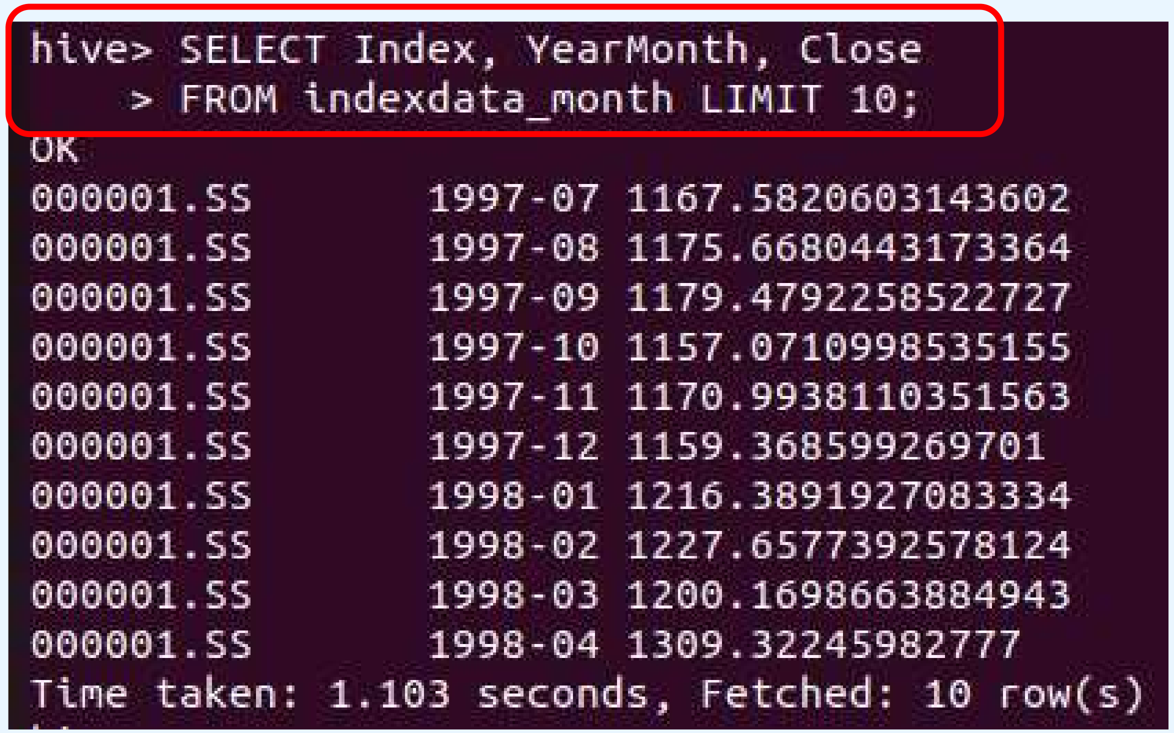
9) For all "Index" names, average all the indices ("Open", "High", "Low", "Close") by "Year-Month"



```
hive> CREATE TABLE indexdata_month AS
    > SELECT Index, YearMonth, AVG(Open) AS Open,
    > AVG(High) AS High, AVG(Low) AS Low, AVG(Close) AS Close
    > FROM indexdata_final
    > GROUP BY Index, YearMonth;
Query ID = lvh_20221222164851_581a618c-7512-4360-a6ad-59640da43935
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1671670341582_0026, Tracking URL = http://lvh-VirtualBo
Kill Command = /home/lvh/hadoop/bin/mapred job  -kill job_1671670341582_00
Hadoop job information for Stage-1: number of mappers: 1; number of reduce
2022-12-22 16:49:26,173 Stage-1 map = 0%,  reduce = 0%
2022-12-22 16:49:53,173 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU
2022-12-22 16:50:11,637 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU
MapReduce Total cumulative CPU time: 12 seconds 900 msec
Ended Job = job_1671670341582_0026
Moving data to directory hdfs://localhost:9000/user/hive/warehouse/wqd700
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 12.9 sec   HDFS Read:
Total MapReduce CPU Time Spent: 12 seconds 900 msec
OK
Time taken: 81.424 seconds
```

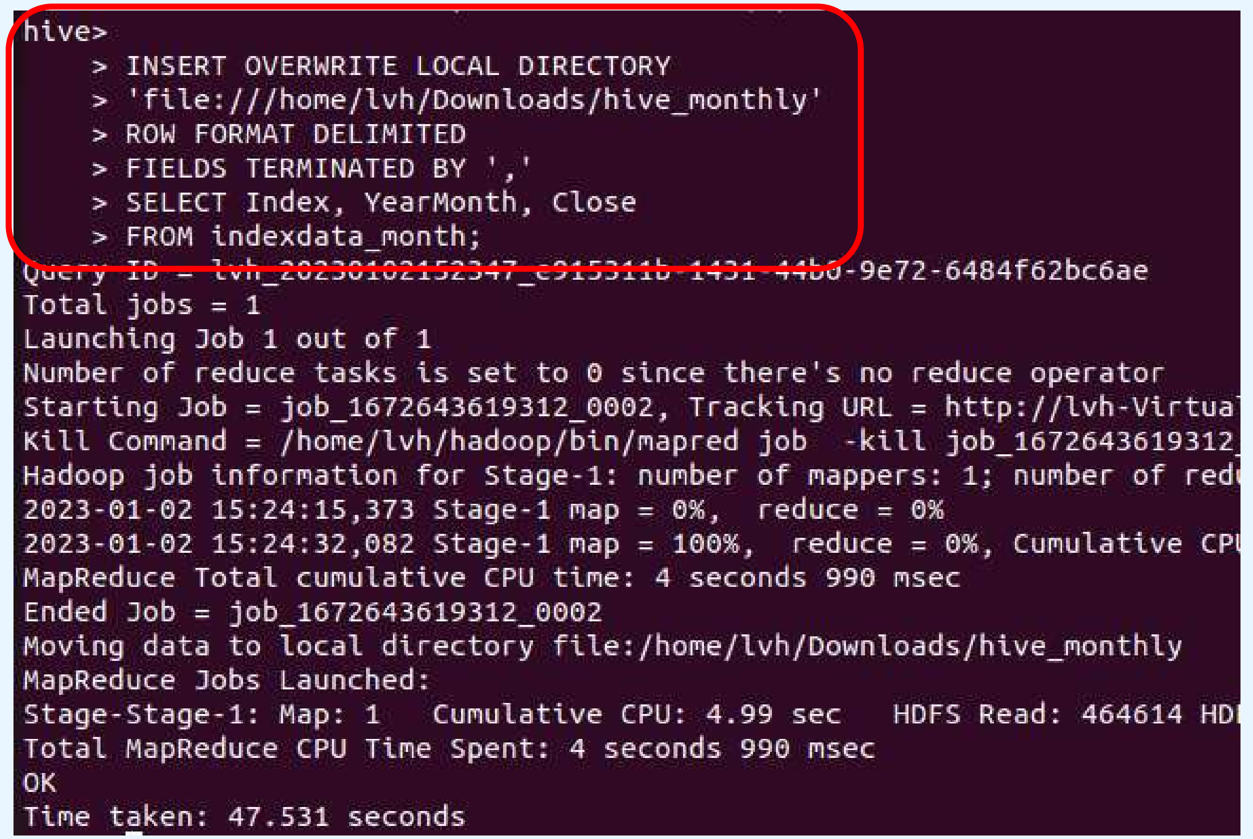
ETL Process using Apache Hive Data Transformation for Monthly Data

10) Query average monthly “Close” column.



```
hive> SELECT Index, YearMonth, Close
    > FROM indexdata_month LIMIT 10;
OK
000001.SS       1997-07 1167.5820603143602
000001.SS       1997-08 1175.6680443173364
000001.SS       1997-09 1179.4792258522727
000001.SS       1997-10 1157.0710998535155
000001.SS       1997-11 1170.9938110351563
000001.SS       1997-12 1159.368599269701
000001.SS       1998-01 1216.3891927083334
000001.SS       1998-02 1227.6577392578124
000001.SS       1998-03 1200.1698663884943
000001.SS       1998-04 1309.32245982777
Time taken: 1.103 seconds, Fetched: 10 row(s)
```

11) Export queried result as csv and ready for analysis.



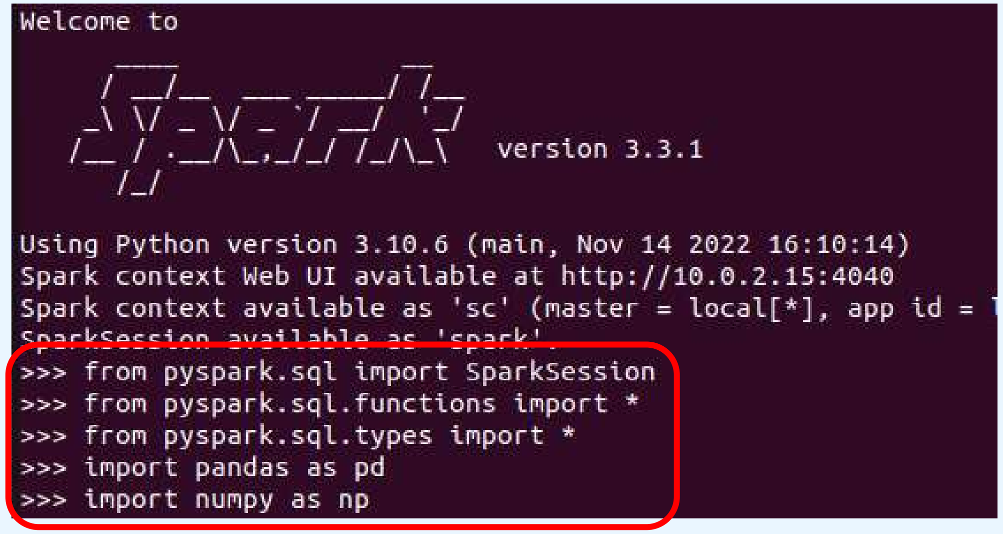
```
hive>
    > INSERT OVERWRITE LOCAL DIRECTORY
    > 'file:///home/lvh/Downloads/hive_monthly'
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > SELECT Index, YearMonth, Close
    > FROM indexdata_month;
Query ID = lvh_20230102152347_e915311b-1431-44b6-9e72-6484f62bc6ae
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1672643619312_0002, Tracking URL = http://lvh-Virtua
Kill Command = /home/lvh/hadoop/bin/mapred job  -kill job_1672643619312
Hadoop job information for Stage-1: number of mappers: 1; number of red
2023-01-02 15:24:15,373 Stage-1 map = 0%,  reduce = 0%
2023-01-02 15:24:32,082 Stage-1 map = 100%,  reduce = 0%, Cumulative CP
MapReduce Total cumulative CPU time: 4 seconds 990 msec
Ended Job = job_1672643619312_0002
Moving data to local directory file:/home/lvh/Downloads/hive_monthly
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 4.99 sec   HDFS Read: 464614 HD
Total MapReduce CPU Time Spent: 4 seconds 990 msec
OK
Time taken: 47.531 seconds
```

Procedure for Rolling Correlation Analysis with PySpark

Step	Description
1	Import the queried result from data access stage above (csv) into PySpark
2	For ease of analysis, spread rows into different index columns using “pivot” function, i.e. “Year-Month”, “Index”, “AvgClose” → “Year-Month”, “Index0”, “Index1”, “Index2”, …, “Index13”
3	Compute pairwise 24-month rolling correlation coefficients for each pair of stock market indices (altogether 91 pairs from 14 stock market indices)
4	Export all pairwise rolling correlation coefficients into separate csv files for further analysis / visualisation

Rolling Correlation Analysis with PySpark

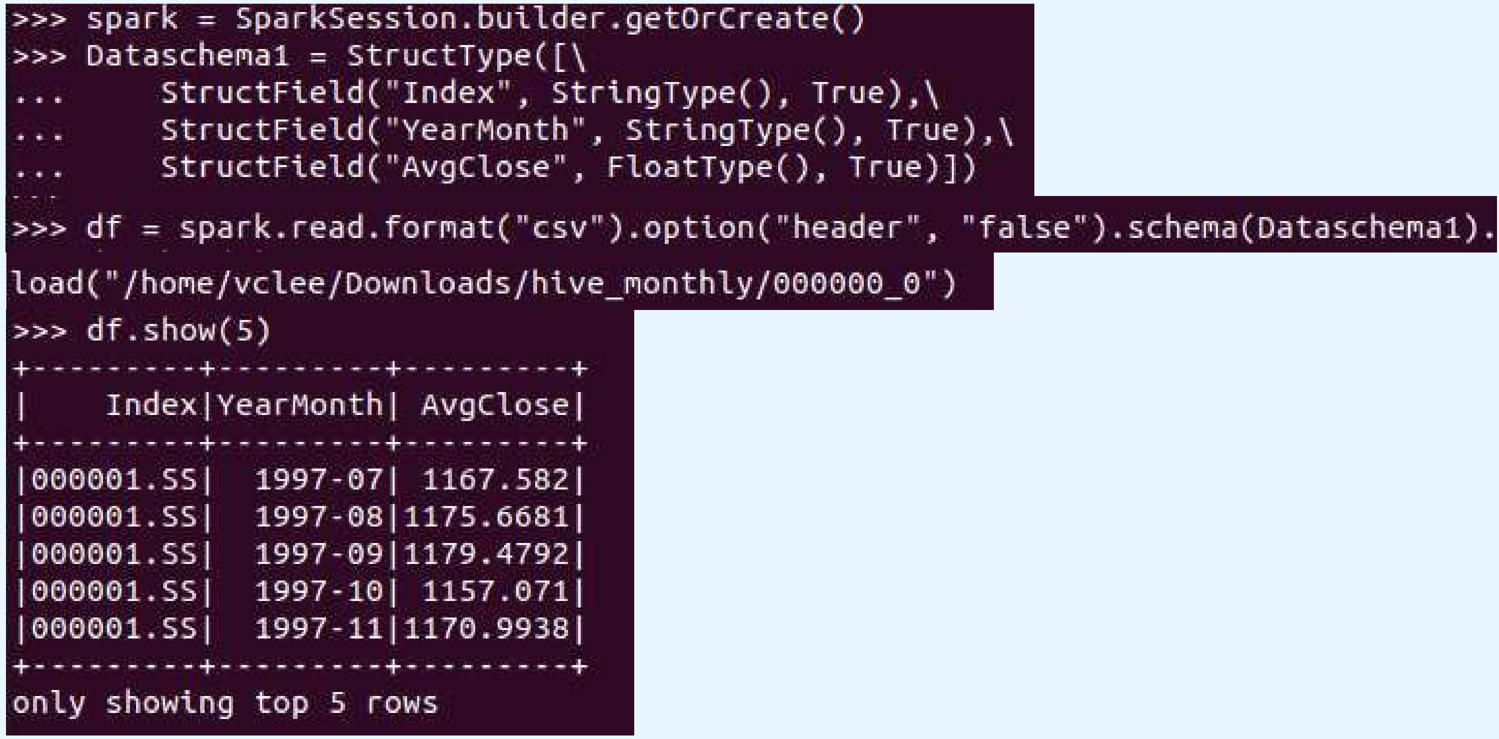
1) Load Python Libraries in Apache PySpark.



```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.3.1
      /_/

Using Python version 3.10.6 (main, Nov 14 2022 16:10:14)
Spark context Web UI available at http://10.0.2.15:4040
Spark context available as 'sc' (master = local[*], app id =
SparkSession available as 'spark'.
>>> from pyspark.sql import SparkSession
>>> from pyspark.sql.functions import *
>>> from pyspark.sql.types import *
>>> import pandas as pd
>>> import numpy as np
```

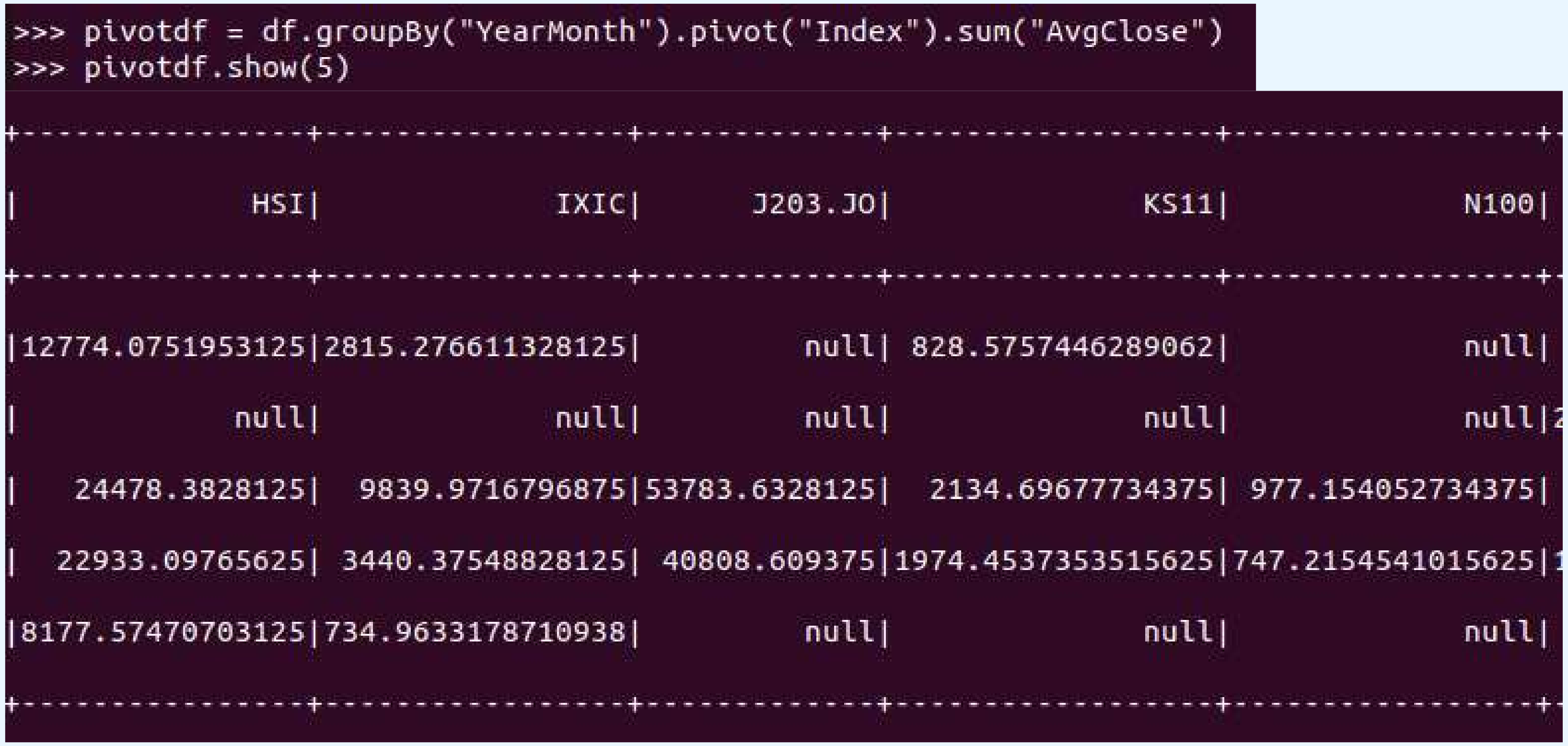
2) Import the queried result from Apache Hive (csv) into PySpark



```
>>> spark = SparkSession.builder.getOrCreate()
>>> Dataschema1 = StructType([\
...     StructField("Index", StringType(), True),\
...     StructField("YearMonth", StringType(), True),\
...     StructField("AvgClose", FloatType(), True)])
>>> df = spark.read.format("csv").option("header", "false").schema(Dataschema1).
load("/home/vclee/Downloads/hive_monthly/000000_0")
>>> df.show(5)
+---------+---------+---------+
|    Index|YearMonth| AvgClose|
+---------+---------+---------+
|000001.SS|  1997-07| 1167.582|
|000001.SS|  1997-08|1175.6681|
|000001.SS|  1997-09|1179.4792|
|000001.SS|  1997-10| 1157.071|
|000001.SS|  1997-11|1170.9938|
+---------+---------+---------+
only showing top 5 rows
```

Rolling Correlation Analysis with PySpark

3) For ease of analysis, spread rows into different index columns using “pivot” function,
i.e. “Year-Month”, “Index”, “AvgClose” → “Year-Month”, “Index0”, “Index1”, “Index2”, …, “Index13”



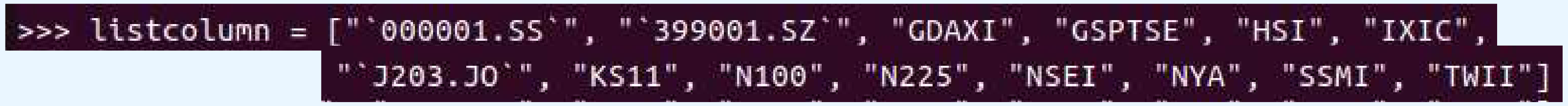
```
>>> pivotdf = df.groupBy("YearMonth").pivot("Index").sum("AvgClose")
>>> pivotdf.show(5)
```

```
+----------------+-----------------+-------------+------------------+-----------------+-
|             HSI|             IXIC|      J203.JO|              KS11|             N100|
+----------------+-----------------+-------------+------------------+-----------------+-
|12774.0751953125|2815.276611328125|         null| 828.5757446289062|             null|
|            null|             null|         null|              null|             null|2
|   24478.3828125|  9839.9716796875|53783.6328125|  2134.69677734375| 977.154052734375|
|  22933.09765625| 3440.37548828125| 40808.609375|1974.4537353515625|747.2154541015625|1
|8177.57470703125|734.9633178710938|         null|              null|             null|
+----------------+-----------------+-------------+------------------+-----------------+-
```

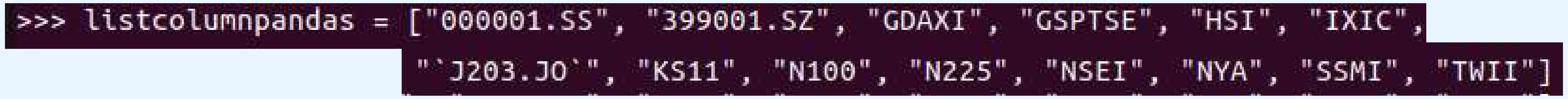
Rolling Correlation Analysis with PySpark

4) Compute pairwise 24-month rolling correlation coefficients for each pair of stock market indices (altogether 91 pairs from 14 stock market indices)

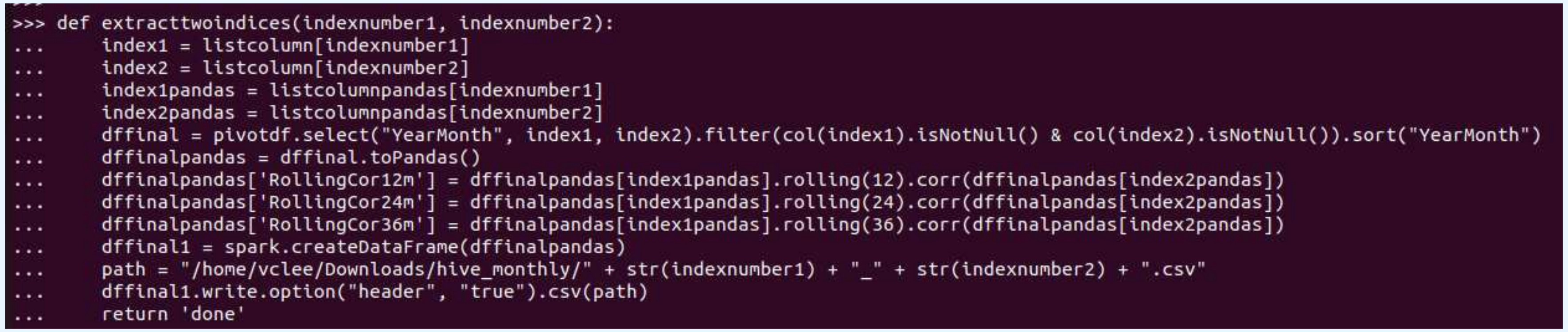
Construct user-defined function “extracttwoindices” to perform the analysis:



```
>>> listcolumn = ["`000001.SS`", "`399001.SZ`", "GDAXI", "GSPTSE", "HSI", "IXIC",
                  "`J203.JO`", "KS11", "N100", "N225", "NSEI", "NYA", "SSMI", "TWII"]
```

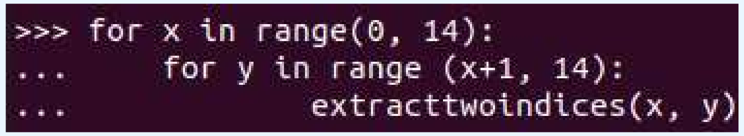


```
>>> listcolumnpandas = ["000001.SS", "399001.SZ", "GDAXI", "GSPTSE", "HSI", "IXIC",
                        "`J203.JO`", "KS11", "N100", "N225", "NSEI", "NYA", "SSMI", "TWII"]
```



```
>>> def extracttwoindices(indexnumber1, indexnumber2):
...     index1 = listcolumn[indexnumber1]
...     index2 = listcolumn[indexnumber2]
...     index1pandas = listcolumnpandas[indexnumber1]
...     index2pandas = listcolumnpandas[indexnumber2]
...     dffinal = pivotdf.select("YearMonth", index1, index2).filter(col(index1).isNotNull() & col(index2).isNotNull()).sort("YearMonth")
...     dffinalpandas = dffinal.toPandas()
...     dffinalpandas['RollingCor12m'] = dffinalpandas[index1pandas].rolling(12).corr(dffinalpandas[index2pandas])
...     dffinalpandas['RollingCor24m'] = dffinalpandas[index1pandas].rolling(24).corr(dffinalpandas[index2pandas])
...     dffinalpandas['RollingCor36m'] = dffinalpandas[index1pandas].rolling(36).corr(dffinalpandas[index2pandas])
...     dffinal1 = spark.createDataFrame(dffinalpandas)
...     path = "/home/vclee/Downloads/hive_monthly/" + str(indexnumber1) + "_" + str(indexnumber2) + ".csv"
...     dffinal1.write.option("header", "true").csv(path)
...     return 'done'
```

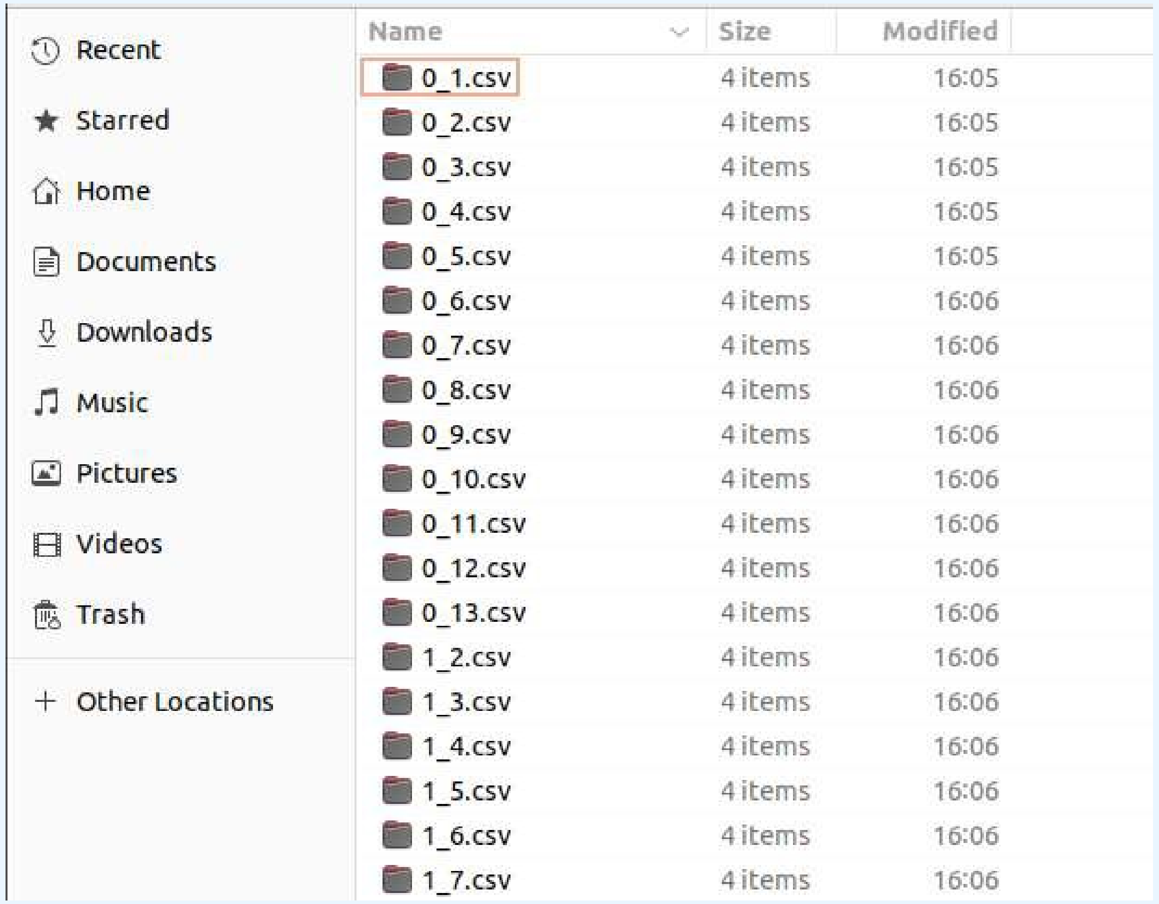
Use “for” loop and “extracttwoindices” function to compute 91 pairs of rolling correlation coefficients:



```
>>> for x in range(0, 14):
...     for y in range (x+1, 14):
...             extracttwoindices(x, y)
```

Rolling Correlation Analysis with PySpark

5) Export all pairwise rolling correlation coefficients into separate csv files for further analysis / visualisation



Name	Size	Modified
0_1.csv	4 items	16:05
0_2.csv	4 items	16:05
0_3.csv	4 items	16:05
0_4.csv	4 items	16:05
0_5.csv	4 items	16:05
0_6.csv	4 items	16:06
0_7.csv	4 items	16:06
0_8.csv	4 items	16:06
0_9.csv	4 items	16:06
0_10.csv	4 items	16:06
0_11.csv	4 items	16:06
0_12.csv	4 items	16:06
0_13.csv	4 items	16:06
1_2.csv	4 items	16:06
1_3.csv	4 items	16:06
1_4.csv	4 items	16:06
1_5.csv	4 items	16:06
1_6.csv	4 items	16:06
1_7.csv	4 items	16:06

3.

Results & Discussion: Rolling Correlation Analysis

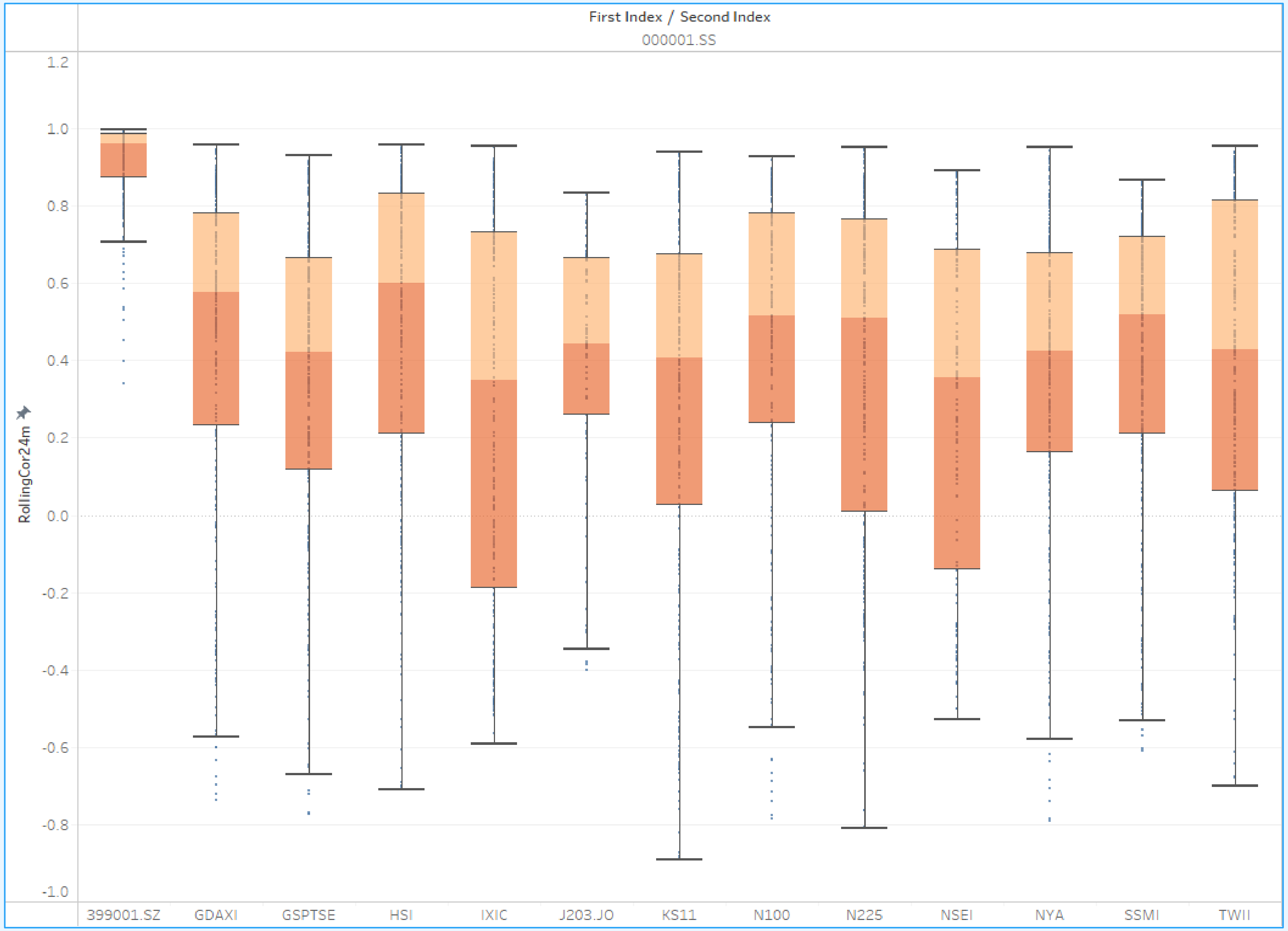
Emerging economies of Asia

- Shanghai Stock Exchange Composite Index, China
- Shenzhen Stock Exchange Component Index, China
- National Stock Exchange Fifty Index, India

Global developed economies

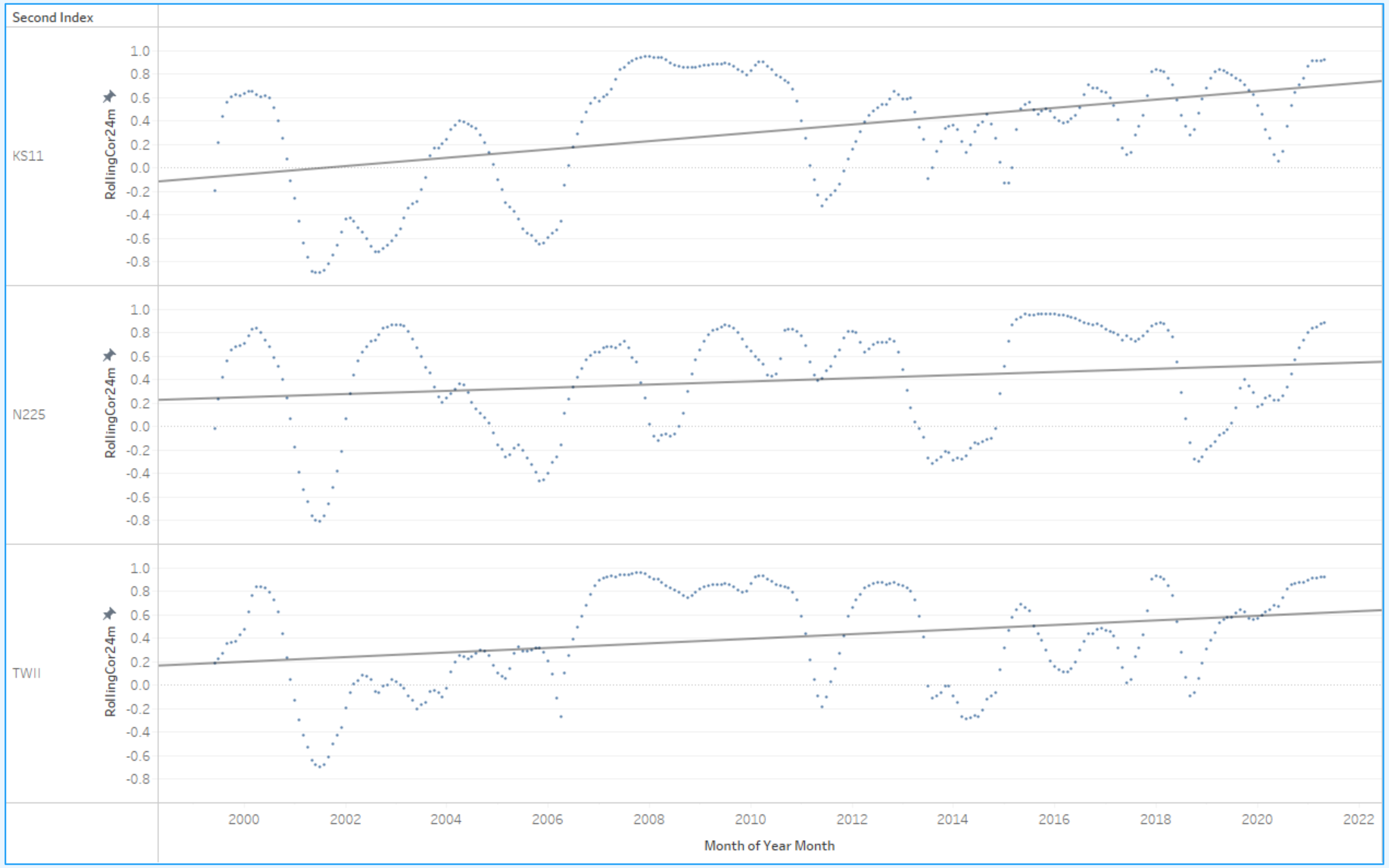
- NASDAQ Composite Index, United States
- Hang Seng Index, Hong Kong

Shanghai Stock Exchange Composite Index, China



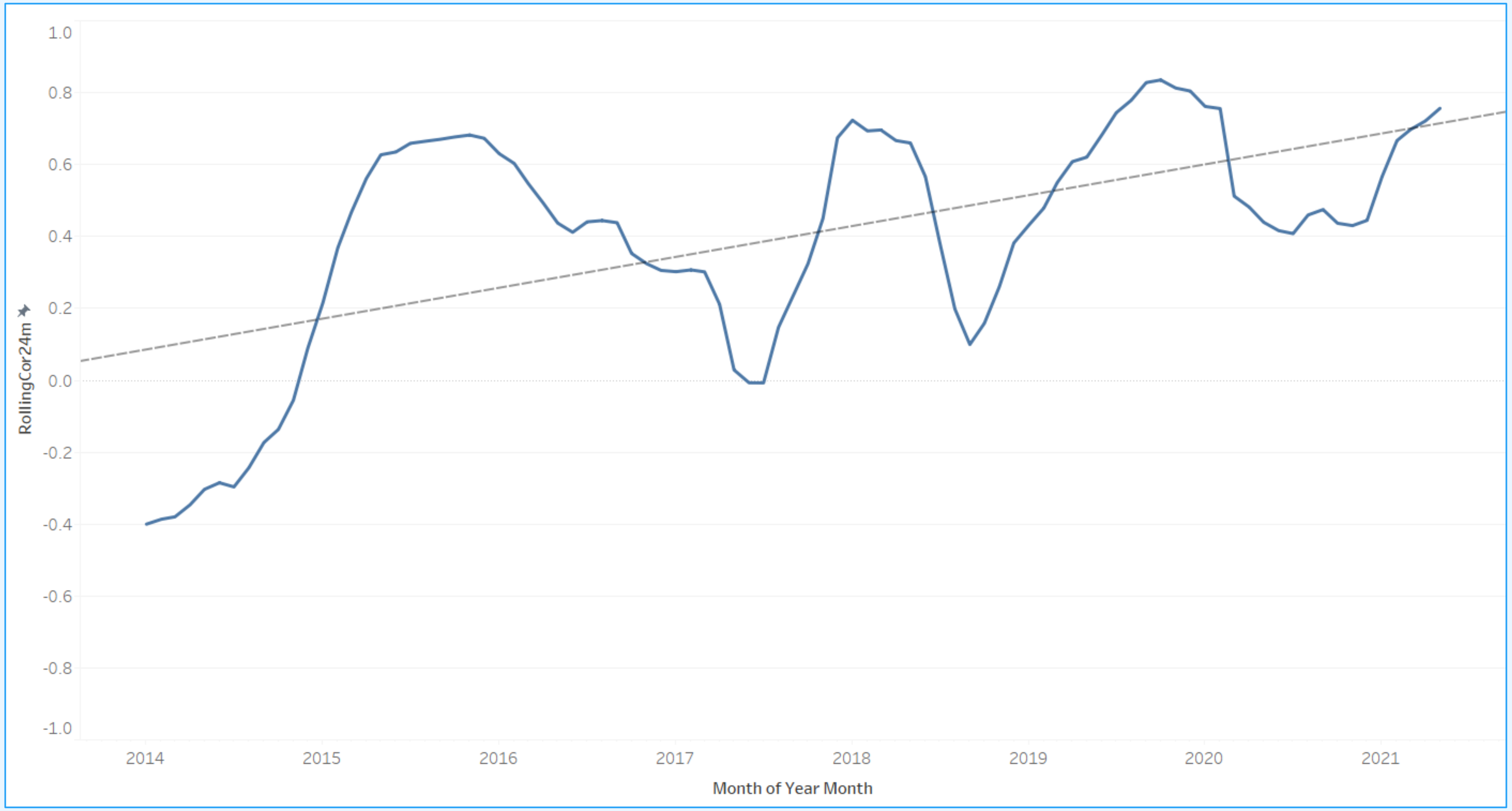
- Except Shenzhen of China, Shanghai was moderately positively correlated with others (median +0.3 to +0.6).
- Germany (+0.576) and Hong Kong (+0.6) had the highest median
- Spread between Shanghai and Shenzhen was small, because both are from Mainland China and subject to similar financial regulations and government policies
- Otherwise, the spreads of Shanghai with other indices were wide, suggesting substantial volatility over time.

Shanghai Stock Exchange Composite Index, China



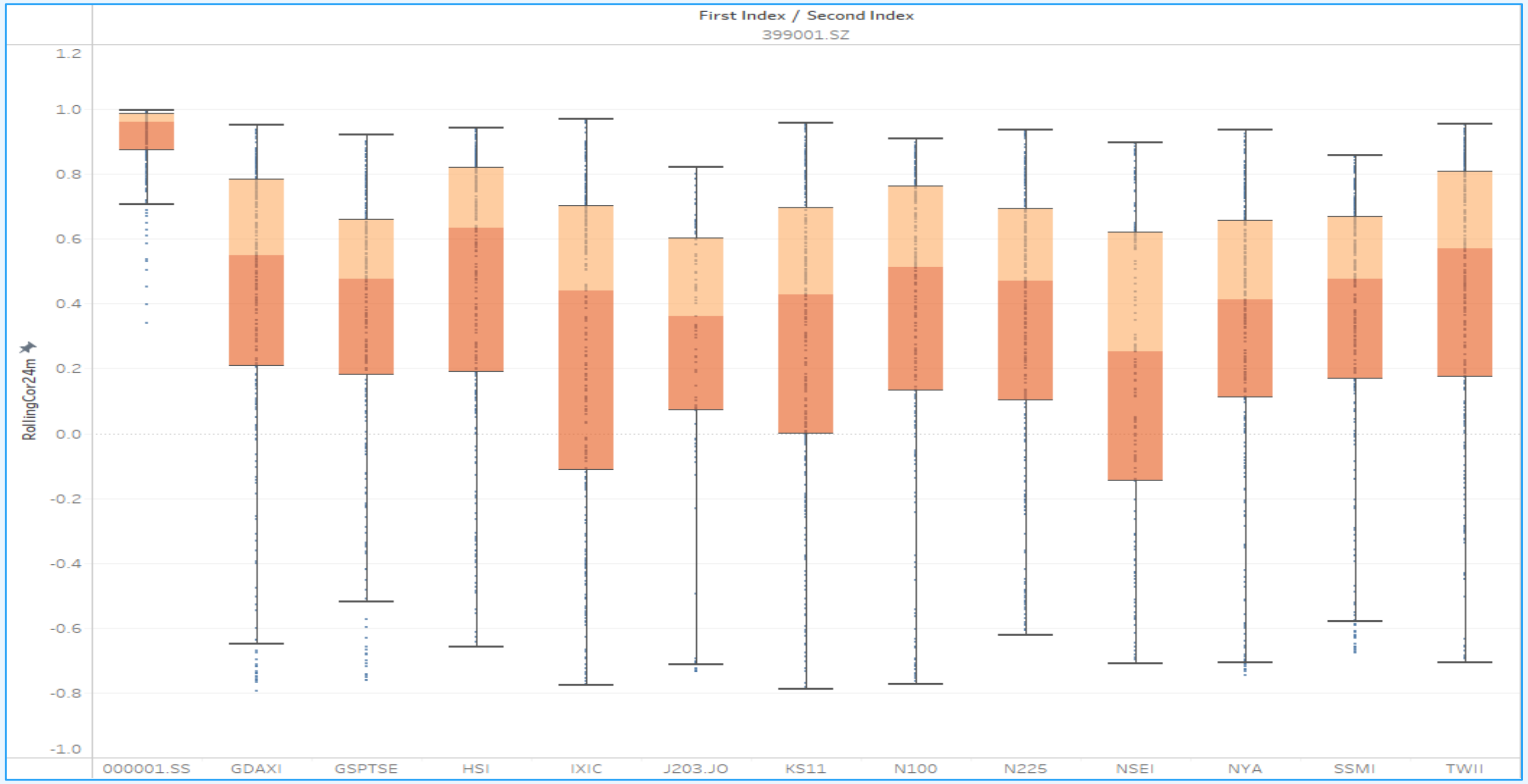
- Increasing rolling correlation with other East Asian indices over time:
    - ➢ KS11 of Korea
    - ➢ N225 of Japan
    - ➢ TWII of Taiwan
- According to Shimizu (2021), since the 1997 Asian economic crisis, the economies in East Asia have integrated more closely, supported by signing of various trade agreements, and holding of economic summits

Shanghai Stock Exchange Composite Index, China



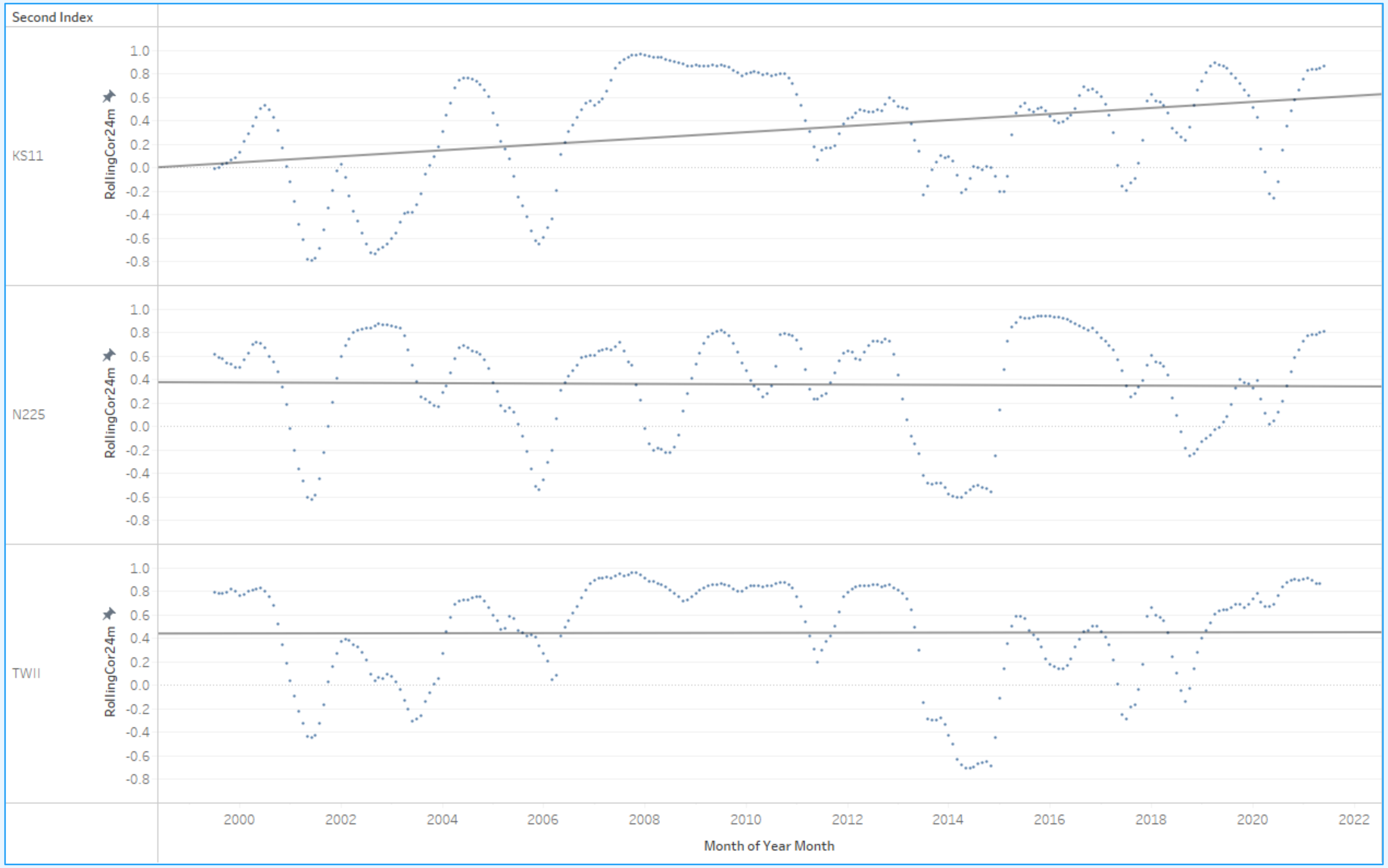
- Increasing rolling correlation with South Africa over time
- Corroborates with findings by Ahmed and Huo (2018):
  - ➢ Stronger Sino-African partnership in trades and investments in recent years, and consequently leading to closer ties between their share indices.

Shenzhen Stock Exchange Composite Index, China



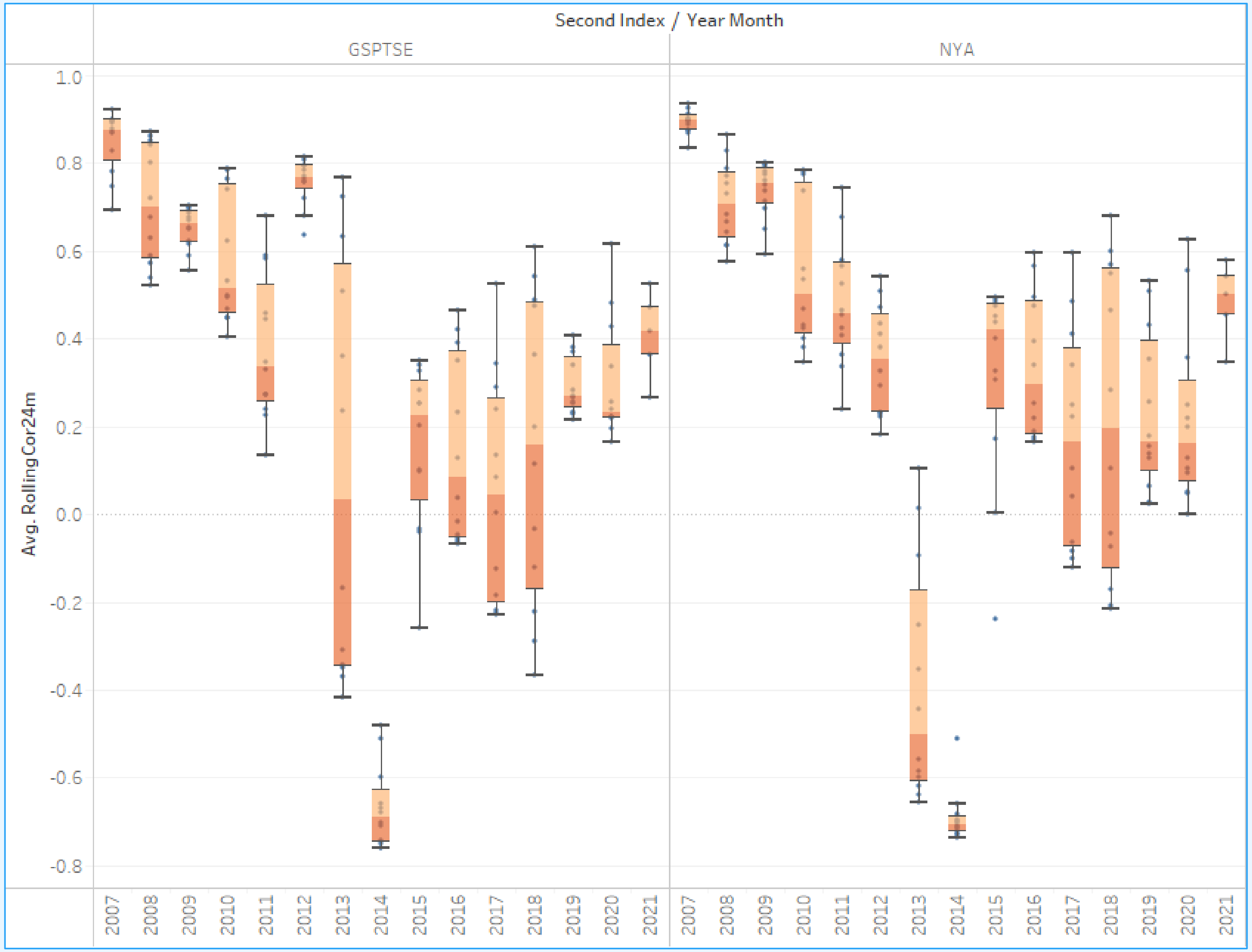
- Similar to Shanghai Index

Shenzhen Stock Exchange Composite Index, China



- Increasing rolling correlation with Korea over time.
- However, contrary to Shanghai, Shenzhen did not show apparent positive trend in the rolling correlation with Japan and Taiwan

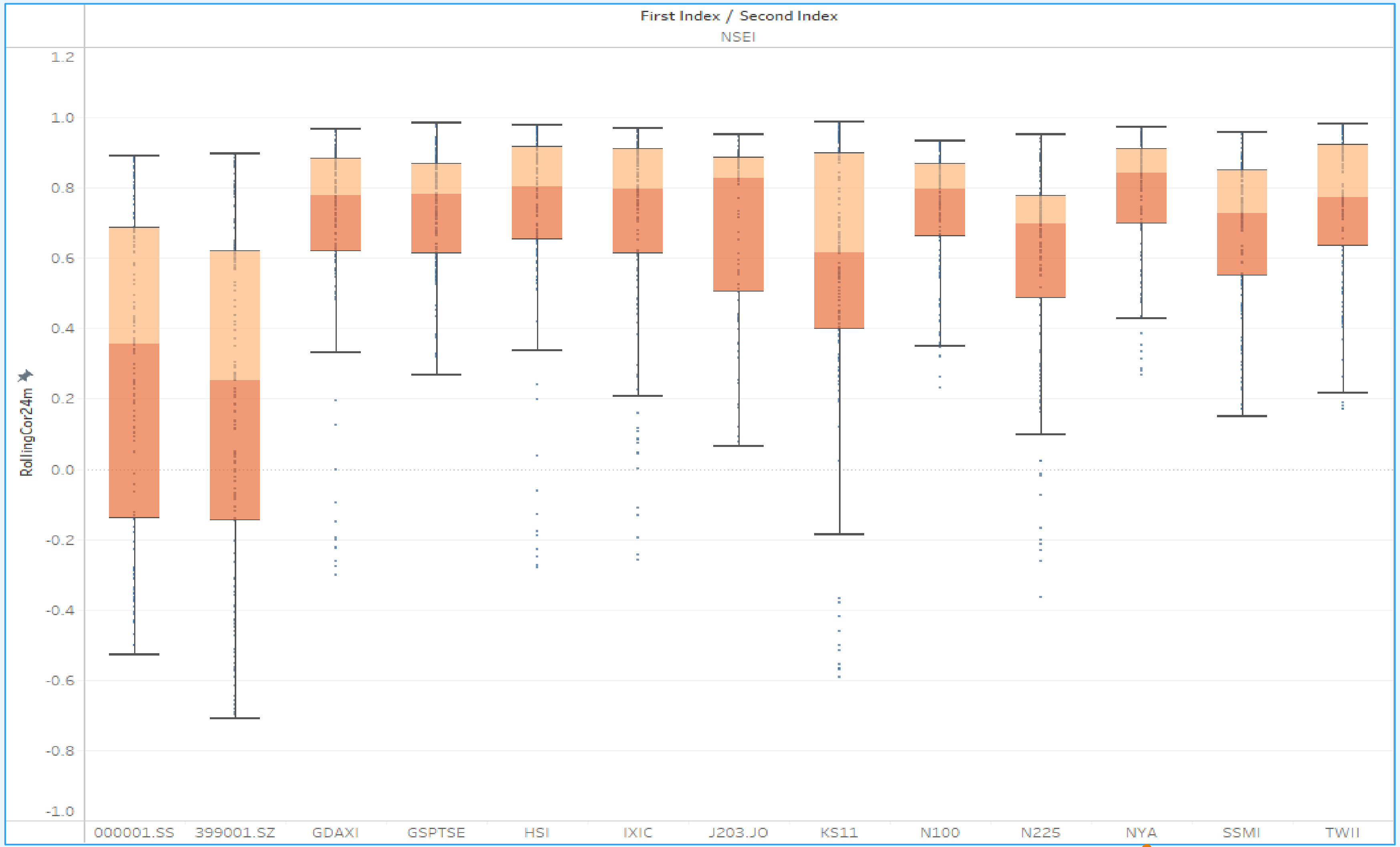
Shenzhen Stock Exchange Composite Index, China



Rolling correlation coefficients between Shenzhen and GSPTSE of Canada, and NYA of US:

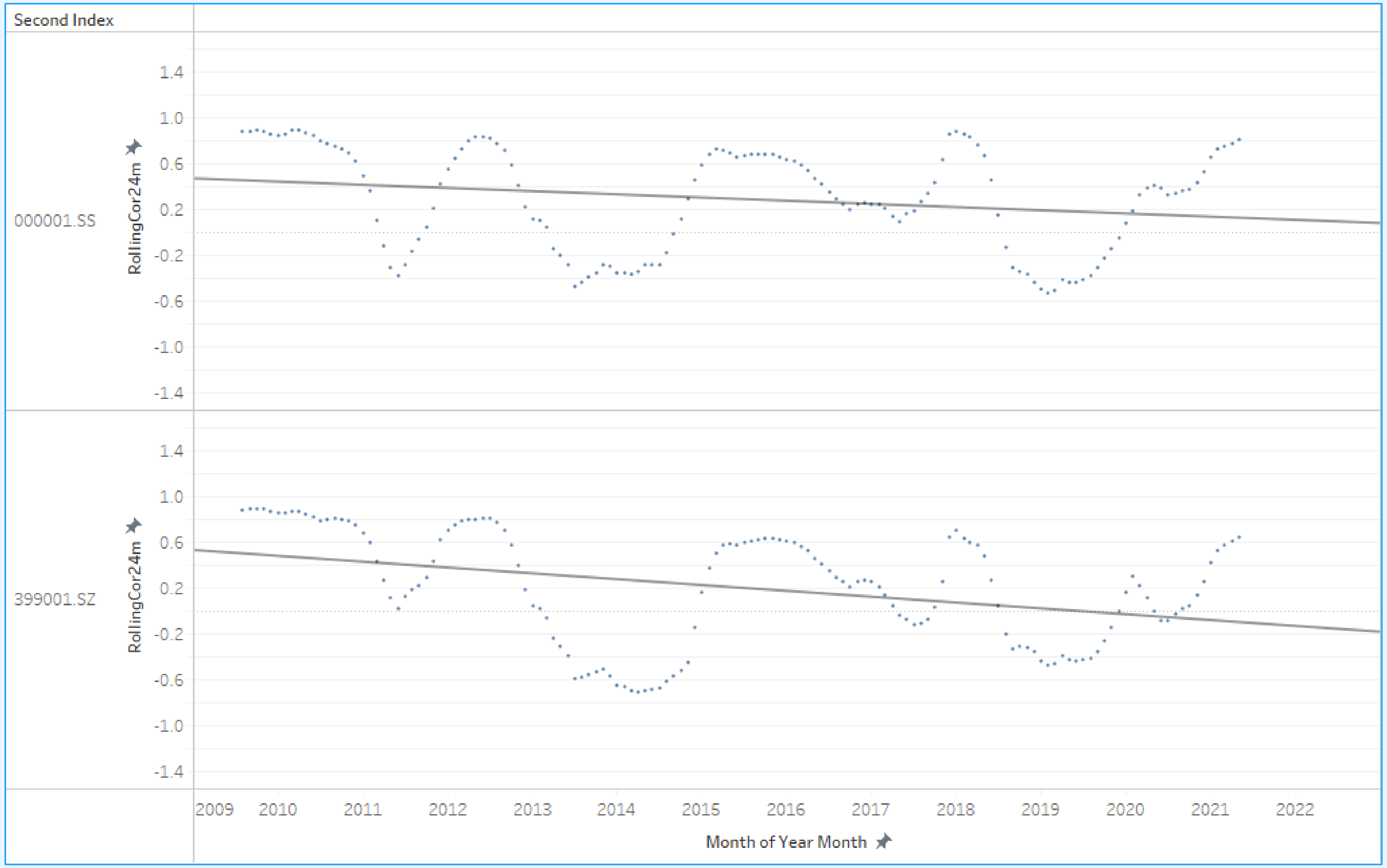
- Year 2013 and 2014 were abnormal years due to the first sign of economic slowdown in China after 30 years of rapid economic growth (Wong, 2015).
- Shenzhen, unlike its Shanghai counterpart, was showing an apparent decreasing correlation with the NYA and GSPTSE since 2015 after a series of political and economic disputes
    - ➢ South China Sea disputes circa 2015 (Whitlock, 2015),
    - ➢ Failed negotiations of free-trade agreements between China and Canada (Vanderklippe, 2020),
    - ➢ The US–China Trade War that began in July 2018, with US imposing the first round of tariffs on Chinese imports, and
    - ➢ Substantial deterioration of Canada–China relations with the arrest of Huawei CFO Meng Wanzhou in December 2018 (Asia Pacific Foundation of Canada, 2019).

National Stock Exchange Fifty Index, India



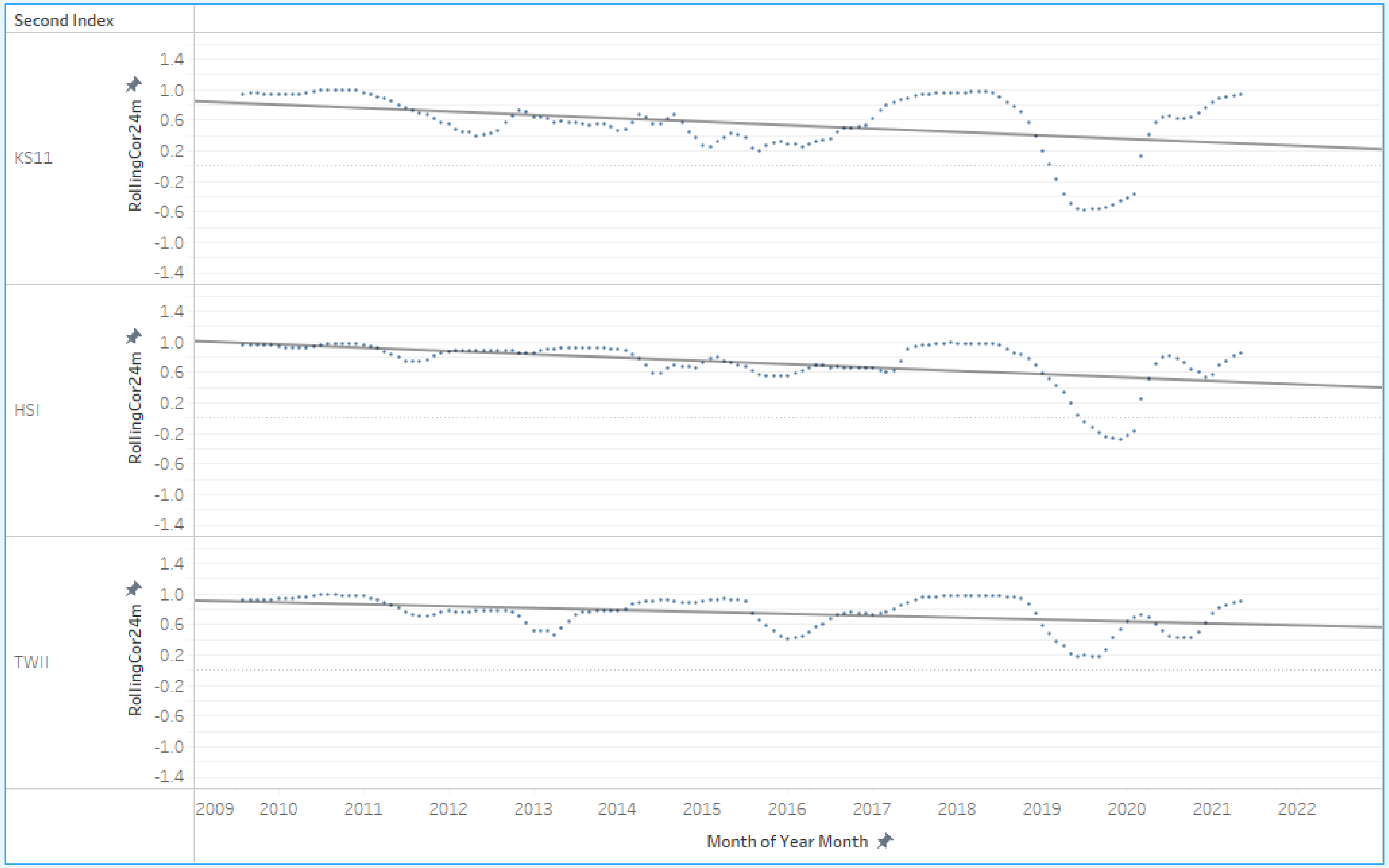
- NSEI was lowly correlated with indices of Mainland China (median +0.2 to +0.4)
- According to Kumar (2020), India and China have small bilateral relationships with little foreign investment inflows and outflows.
- NSEI was highly correlated with indices from other countries / territories (median > +0.7), except Korea (median = +0.615) and Japan (median = +0.698).
- This indicates NSEI is an integral part of the global economy.

National Stock Exchange Fifty Index, India



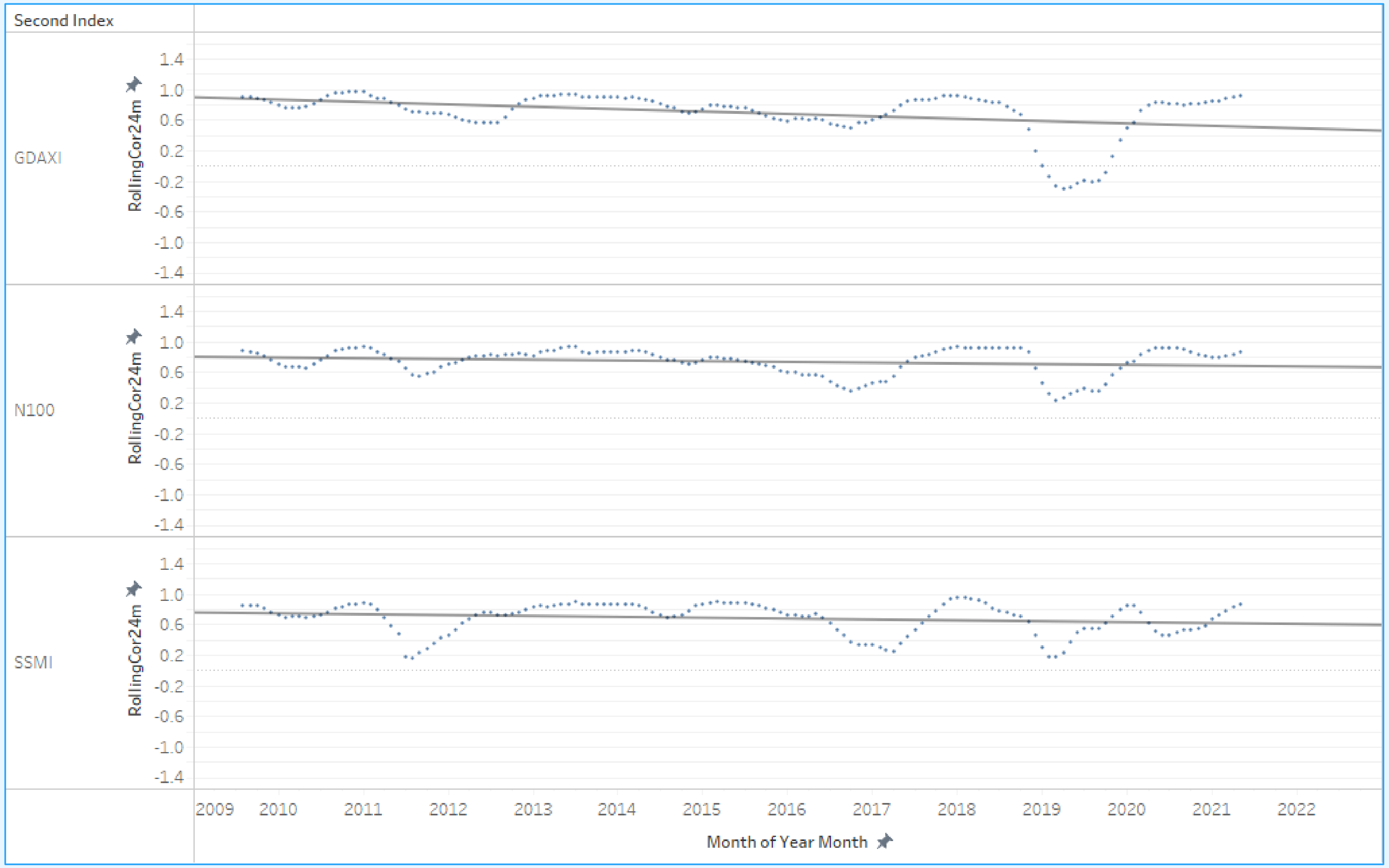
- Decreasing rolling correlation with Shanghai and Shenzhen of Mainland China
- As the border dispute in recent years continues, Kumar (2020) expects the Chinese and Indian economies will continue to decouple. As such, the rolling correlation between these indices is expected to continue its downward trajectory.

National Stock Exchange Fifty Index, India



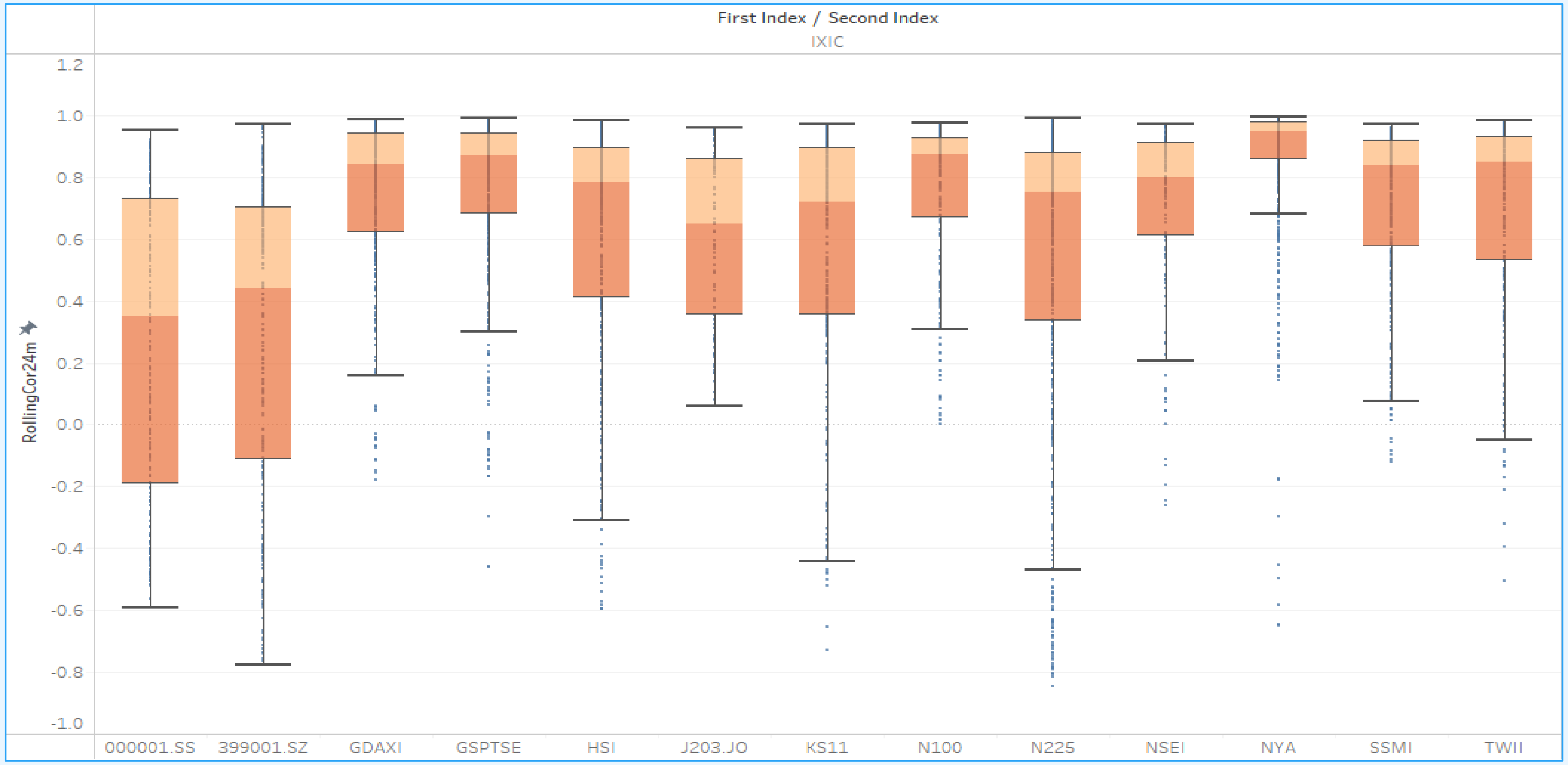
- Decreasing rolling correlation with other East Asia economies except Japan, i.e. Hong Kong, Korea, and Taiwan
- Downtrend of rolling correlation between NSEI and these indices could be due to the spillover effect of their increasing correlation with Mainland China.

National Stock Exchange Fifty Index, India



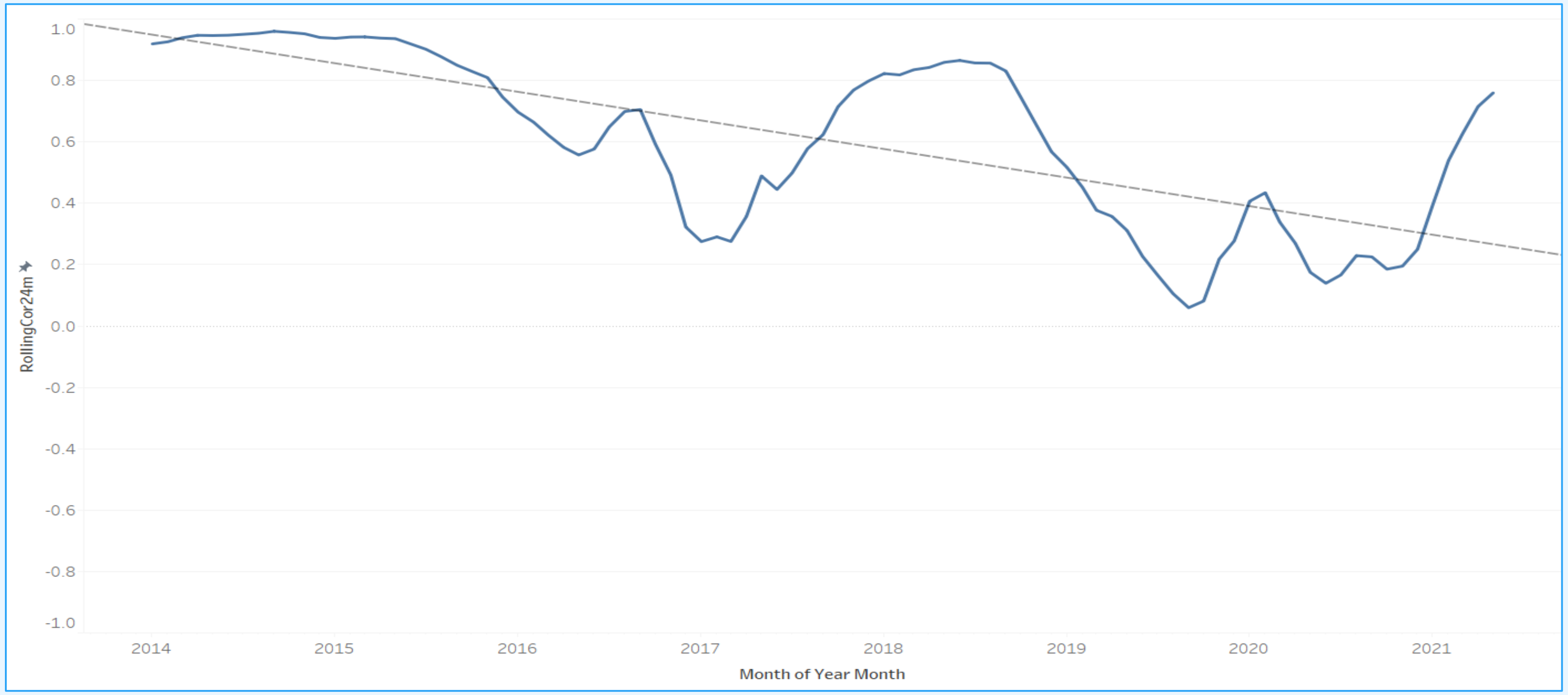
- Slightly decreasing rolling correlation with European indices, namely GDAXI of Germany, N100 of Europe, and SSMI of Switzerland

NASDAQ Composite Index, United States



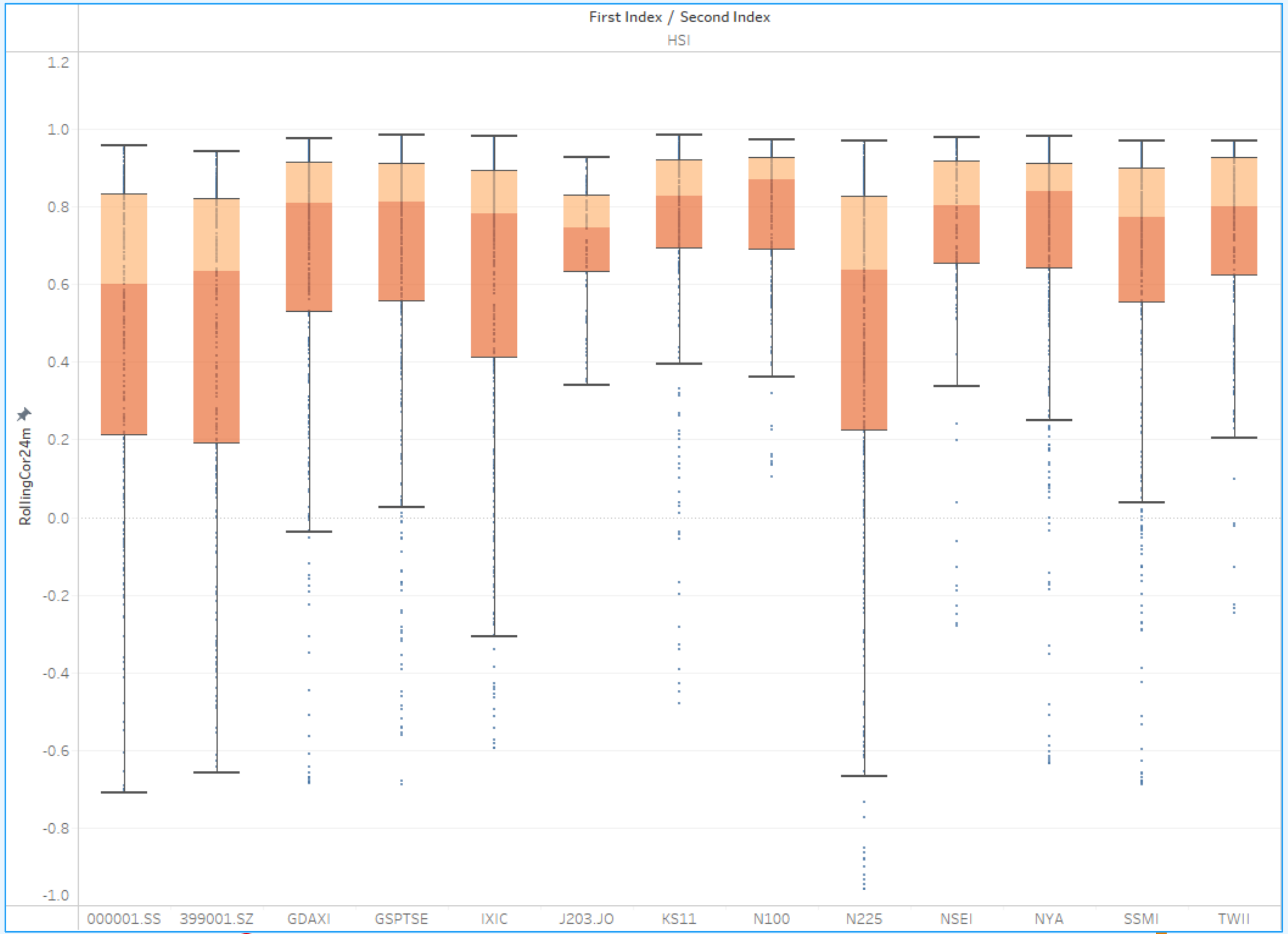
- All indices were highly correlated with NASDAQ (median > +0.7), except Shanghai, Shenzhen and South Africa

NASDAQ Composite Index, United States



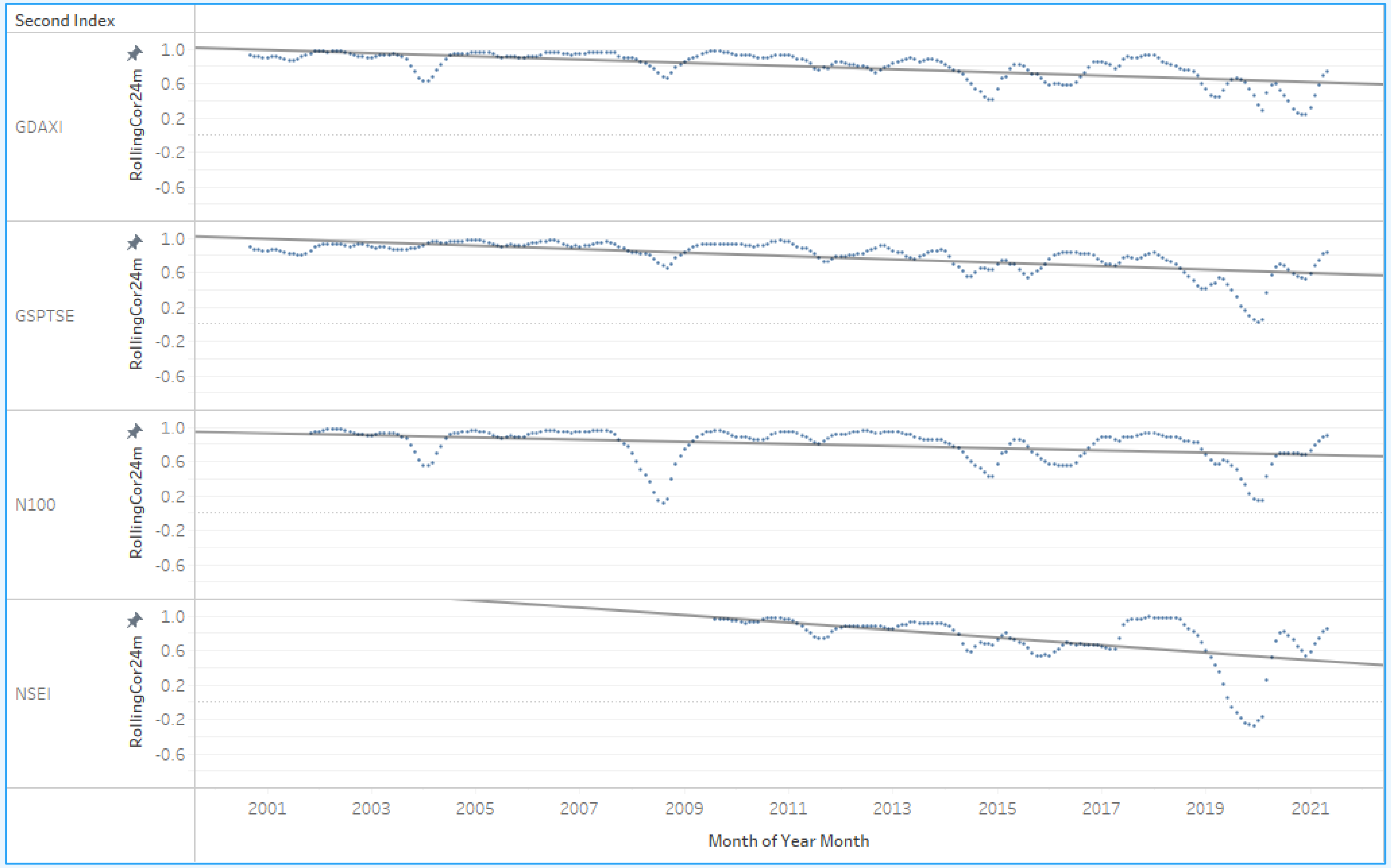
- One notable movement is the plummeting rolling correlation coefficient of NASDAQ against South Africa
- The decreasing rolling correlation highlights the flatlined influence of the US on South Africa, as discussed in Sheehy (2022).

Hang Seng Index, Hong Kong



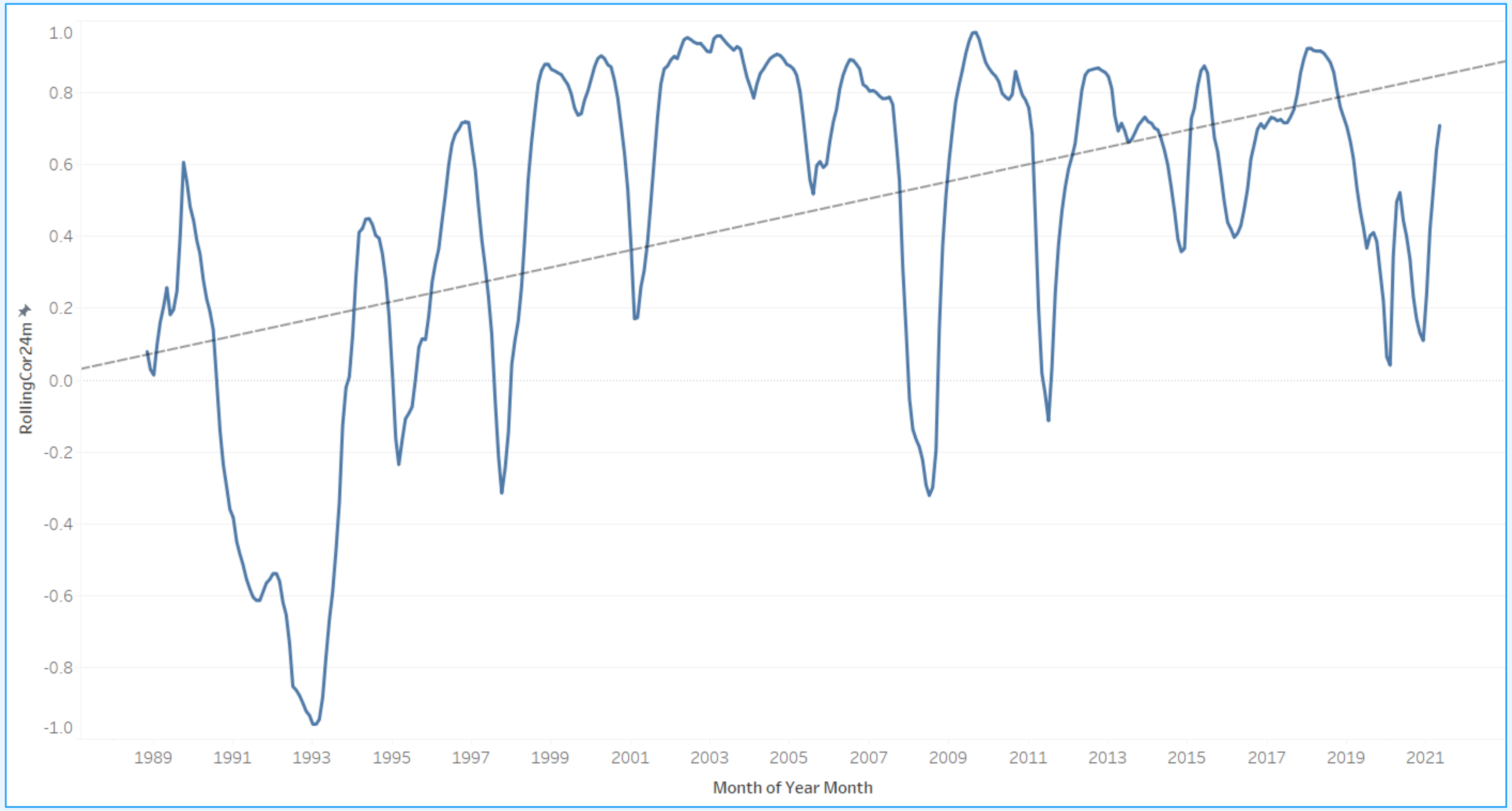
- Being a global financial hub, the median rolling correlation HSI with ex-Mainland-Chinese indices were high (> +0.7), except with N225 of Japan
- As a Special Administrative Region of China, Hong Kong at the same time also exhibited the highest median rolling correlation with the two Mainland-Chinese indices (> +0.6) compared with other indices

Hang Seng Index, Hong Kong



- In the recent five years, the rolling correlation coefficient of HSI with GDAXI of Germany, GSPTSE of Canada, N100 of Europe, and NSEI of India were trending down

Hang Seng Index, Hong Kong



- On the other hand, its rolling correlation coefficient with N225 of Japan showed substantial increase after the 1998 Asia Financial Crisis

4.

Conclusion & Future Works

Conclusion

- Rolling correlation analysis of global stock markets were effectively performed by PySpark, aided by Tableau as the data visualisation tool
- Business insights can be derived from the analysis to make educated decisions

Future Works

- To execute Hive with Apache Tez or Apache Spark
- To conduct rolling correlation analysis with other window widths (e.g. 12m, 36m)
- To build a model to predict future rolling correlation.

Thanks!

Any questions?