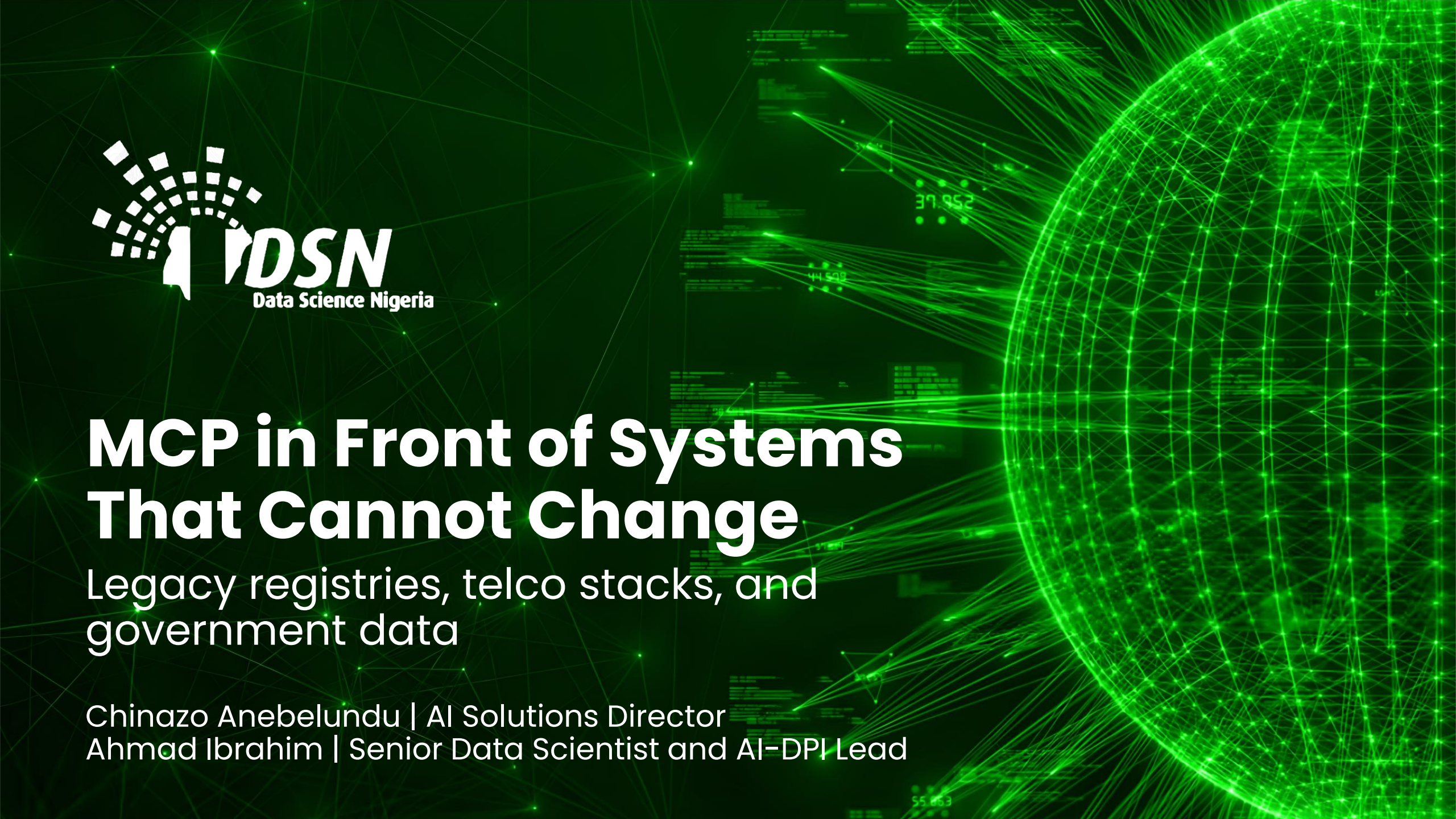


MCP in Front of Systems That Cannot Change

Legacy registries, telco stacks, and government data

Chinazo Anebelundu | AI Solutions Director
Ahmad Ibrahim | Senior Data Scientist and AI-DPI Lead



Agenda

01 What tutorials teach, and what they assume

02 Start thin: wrap the old system, don't rebuild it

03 Many government clients on one platform

04 Security rules that come from contracts

05 Tool descriptions need review, just like code

06 Earning the trust of government IT teams

What MCP docs and tutorials teach

Official quickstart: a weather server in one hour

1

Pick an open API

Public weather API. No login.

2

Write two tools

get_alerts, get_forecast

3

Comment = description

Your comment is what the AI reads

4

Run it on your laptop

One user, one machine

Back on slide 12

What it teaches well

How client, server and tools fit

What it quietly assumes

Open API, one user, no audit, full control

Source: "Build an MCP server" quickstart, modelcontextprotocol.io

Our reality: the old system cannot change

What the tutorial assumes

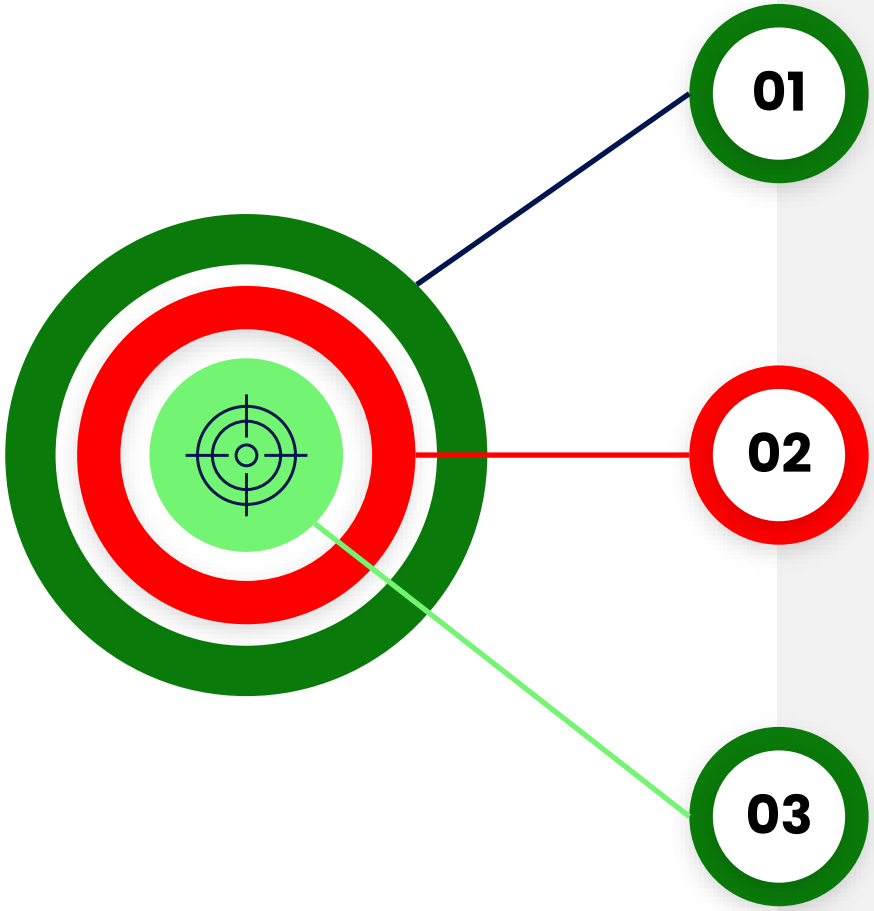
- ✗ Add endpoints anytime
- ✗ Change the database fast
- ✗ Data goes anywhere
- ✗ Easy security approval

What we actually get

- ✓ Read-only or nightly exports
- ✓ Months-long change queues
- ✓ Data must stay in-country
- ✓ Security fears AI-chosen calls

Build around the old system, never inside it

Three projects, same problem



Company assistant
Multi-agent, in the client's cloud

Locked: core records
We get: read-only copies

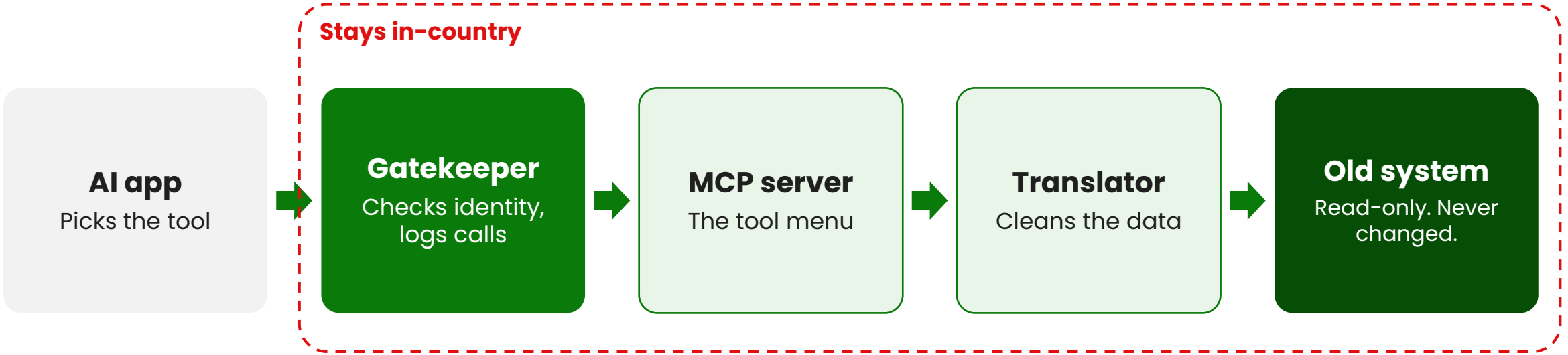
Telco translation
Live call and SMS translation

Locked: network and billing
We get: edge hooks

Government search
In-country records search

Locked: agency records
We get: read-only, file drops

The design we always end up with



1. **The AI sees only the menu**
   No agency, no permissions, no raw data
2. **Hard rules live in code**
   User, agency, limits: fixed in code
3. **The old system notices nothing new**
   Just normal read requests

Start thin. Add layers only when forced.

Never rebuild. Each step up costs more.

Where most tools should stay

Direct wrap

One tool per endpoint. Fast, but messy.

01

Clean-up layer

Rename fields, plain errors.

02

Own copy

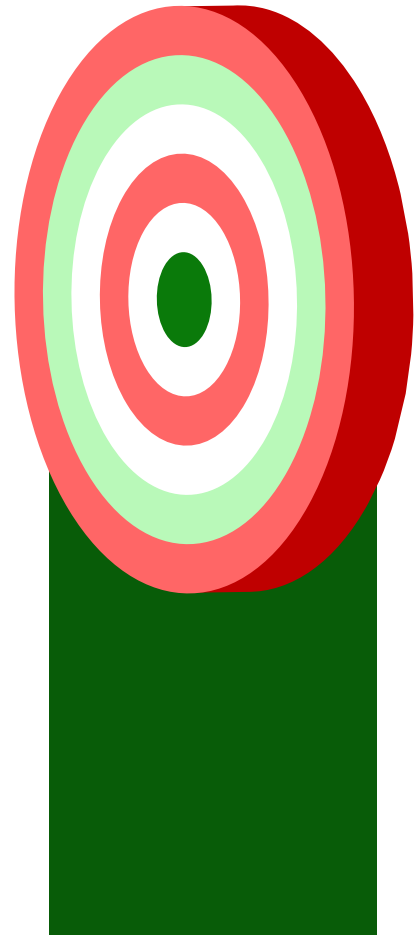
Your copy, for fast search.

03

Own service

Built beside it, not instead.

04



When are you forced to move up?

Warning sign	What you see	Move to
Business rules creep in	"If status is 3, then..."	Own service
Can't combine data	AI joins results itself	Own copy
Too slow	Voice needs < 1 second	Own copy + cache
Errors keep growing	New codes every release	Own service
You need to write	AI must update records	Own service + approval
Systems disagree	One citizen, three IDs	Own service

Can't review it in one sitting? It's a service now.

Many government clients, one platform

1

Login decides the agency

From login, never from the AI

2

Separate data, shared code

Own data and keys, one codebase



No sharing by default

Only with a signed agreement

4

Each agency sees its own tools

Only tools its agreement allows

3

Security rules come from contracts

Contracts, not tickets

Each clause becomes a control

"Who saw what data?"

Log user, agent, agency

"Give only the access needed"

Read-only tools, per-tool checks



"Data stays in the country"

Model, logs, indexes: local

"Use our login system"

Client's own login

What we record for every tool call

```
{
  "ts": "2026-03-14T09:21:07Z",
  "tenant": "agency-04",
  "principal": "user:officer-1182",
  "tool": "registry.lookup_record",
  "contract_version": "2.3.0",
  "contract_hash": "sha256:9b1e4c",
  "args_hash": "sha256:51aa07",
  "decision": "allow",
  "result": {"rows": 1, "class": "PII"}
}
```

1. **Record the tool's version**
   Explains why behaviour changed

2. **Hide the private values**
   Hash inputs, hide values

3. **Record it at the gatekeeper**
   MCP headers name the tool

One sentence changed the AI's behaviour

Before

`search_documents`

"Search policy documents by keyword."

After

`search_documents`

"Search policy documents. Use for any question about programmes, eligibility or status."

No code changed. No schema changed.

1

The AI picked the wrong tool

Search, not the live registry

2

The answers still sounded confident

From outdated documents

3

Nothing flagged it

Checks only looked at code

Lesson: the AI reads tool descriptions as instructions. Review them like code.

How we now review tool descriptions

1. **Keep it in the code repo**
   One versioned file
2. **Show the wording change**
   Old vs new, side by side
3. **Re-test the AI**
   Rerun tests, flag tool changes
4. **Get two approvals**
   Owner + one user team
5. **Number it and lock it**
   Gateway blocks unapproved
6. **Roll out slowly**
   One agency first

How big is a description change?

Change	Example	Size	Who must approve
Typo fix	"retreive" to "retrieve"	Small	Tool owner
Any wording change	Add "use for status questions"	Medium	Owner, users, re-test
New optional input	Add a language option	Medium	Owner, re-test
Rename or remove input	id becomes record_id	Big	Full review
Read-only becomes write	readOnlyHint: true to false	Big	Security, owner

Wording is never small: other teams' agents obey it.

Earning the trust of government IT teams

Stage 1
AI suggests, people act
Staff run AI suggestions

Stage 2
Read-only, approved tools
Read-only, IT off switch

Stage 3
Human approves each action
A person approves changes

Stage 4
Limited independence
Granted by IT, revocable

What actually convinced them

Hard limits
AI never sets agency or limits

Replay any answer
Trace every call and version

Their own language
Controls in their policy terms

Takeaways

1. Wrap first. Add layers only when forced.
2. Code decides user and agency, never the AI.
3. Contract rules become call logs.
4. Review tool descriptions like code.
5. Build trust step by step.

Let's discuss

How does your team review them today?

Thank You

- Data Science Nigeria
- DSNai - Data Science Nigeria
- Dsn_ai_network
- DSNai_Data Science Nigeria/ Data Scientists Network
- www.datasciencenigeria.org
- www.dsn.foundation

