BDS Workshop

Prototyping an Integrated AI Application

From Business Idea to Working POC in 2 Hours

Christian Nielsen
Eskil Olav Andersen, PhD

Aalborg University Business School
Innovation, Knowledge, and Economic Development (IKE)
AI Denmark (AIDK)
eoa@business.aau.dk

November 7, 2025

Agenda

Introduction

Today's Challenge

Build a **working proof-of-concept** application

Your Mission:

- Create a Streamlit app
- Demonstrate clear business value
- Runs locally (no deployment)
- 5-minute stakeholder pitch

Time Available:

- 2 hours of development
- Access to full dataset
- LLMs as co-pilots
- Teacher support

Key Point: Focus: Demonstrate value, not perfect code

Workshop Timeline & Deliverables

What You'll Deliver:

1. **Working Streamlit App** - Runs on your laptop, demonstrates your solution
2. **5-Minute Pitch** - Problem, demo, technical approach, insights
3. **Clear Value Proposition** - Who is this for? What problem does it solve?

Success = Working Prototype with Clear Value

What Success Looks Like:

- ✓ App runs without errors
- ✓ Solves a clear business problem
- ✓ You can explain it convincingly
- ✓ Components work together
- ✓ Generates actionable insights

What's NOT Required:

- × Production-ready code
- × Perfect error handling
- × Beautiful UI design
- × Scalable architecture
- × Comprehensive testing

Can you pitch it convincingly in 3-5 minutes?

POC vs Production: Different Mindsets

Proof of Concept

- **Timeline:** Hours to days
- **Goal:** Show it works
- **Quality:** Good enough
- **Scope:** Core idea only
- **Users:** Stakeholder-s/demo
- **Focus:** *Does this solve the problem?*

If POC succeeds →

Production System

- **Timeline:** Weeks to months
- **Goal:** Ship to users
- **Quality:** Robust, tested
- **Scope:** Full feature set
- **Users:** Real end-users
- **Focus:** *Scalable & maintainable*

Necessary Steps

Step 1: Define Your Value Proposition

Start with clarity on WHO and WHY:

Who?
Apartment seekers?
Landlords?
Investors?
Platform users?

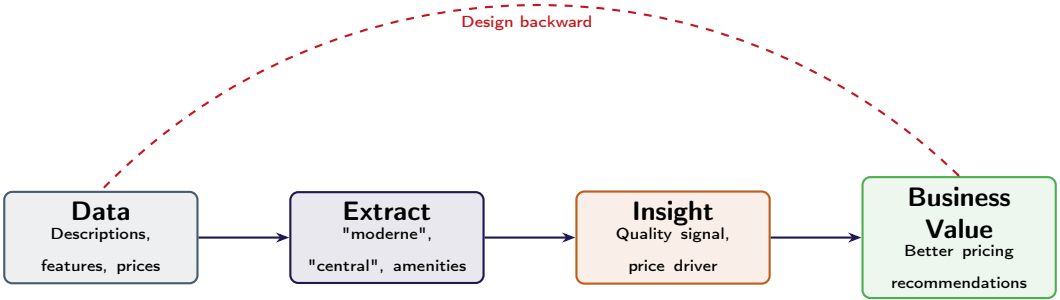
What Problem?
Finding apartments?
Pricing decisions?
Market insights?
Search quality?

Why Better?
Faster?
More accurate?
New insights?
Better UX?

Key Point: Clear value proposition = Easier to build & easier to pitch

Step 2: Map Data → Features → Value

Work backward from the value you want to create:

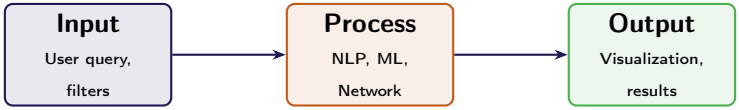


Example:

- **Goal:** Recommend similar apartments
- **Need:** Similarity measure (cosine, network distance)
- **Requires:** Feature vectors or embeddings from descriptions
- **Therefore:** Extract text features OR use semantic embeddings

Step 3: Design Modular Architecture

Break your app into composable components:



Why Modular?

Benefits:

- Test components independently
- Debug in isolation
- Replace/improve one part
- Easier to explain

Integration Strategy:

- Build components first
- Test with dummy data
- Connect incrementally
- Validate each connection

Important: Components should be composable and testable independently

At 1 Hour: STOP and Evaluate

You should have:

- ✓ Data loads successfully
- ✓ One analysis component works (NLP/M-L/Network)
- ✓ Basic Streamlit app runs

If not, consider:

- Simplify scope
- Use CSV from EDA notebook
- Ask teachers for suggestions
- Adjust ambition level

Dataset and Contextualization

Dataset Overview

Boligportal Aalborg Dataset

Scale & Structure:

- 1,330 rental apartments
- Scraped November 2025
- Rich structured fields
- Danish text descriptions
- Complete metadata

Key Fields:

- Rent, size, rooms, floor
- Amenities (elevator, balcony, parking)
- Location (street, postal code)
- Description (avg 1,049 characters)

What You Have:

- `boligportal_aalborg.jsonl`
- EDA notebook (full exploration)
- CSV export (if preferred)
- Example notebook (clustering, recommendations)

Data Quality:

- ✓ 0% missing critical fields
- ✓ Complete descriptions
- ✓ Clean, ready to use

Example Value Propositions

Different users, different problems, different solutions:

Apartment Seekers:
"Find apartments matching my preferences"
→ Semantic search + recommendations + similarity networks

Landlords:
"Is my apartment priced correctly?"
→ Price prediction + comparable analysis + market positioning

Market Analysts:
"What defines each market segment?"
→ Clustering + feature analysis + segment profiling

Investors:
"Which neighborhoods offer best value?"
→ Location analysis + price-quality ratios + trend identification

Technical Components

Component 1: Unlocking Insights from Text

Descriptions hold rich details. Start simple, escalate as needed.

- **Basic (Fast & Reliable): Keyword Searching**
  e.g., `"altan" in description.lower()`
  - ✓ **Pro:** Easy, fast, great for binary features (has_dishwasher).
  - × **Con:** Misses synonyms (*balkon*) and context (*ikke støjende* vs *rolig*).

- **Intermediate (Semantic Power): Embeddings**
  e.g., using `sentence-transformers`
  - ✓ **Pro:** Captures meaning. Find "quiet modern" from "peaceful renovated".
  - × **Con:** Heavier computation than simple keywords.

- **Advanced (Generative AI): Large Language Models (LLMs)**
  Use for complex extraction (JSON of amenities), summarization, or classification (e.g., "vibe").

LLM Trade-offs: API vs. Local

API (OpenAI, Google)

- ✓ Extremely powerful
- ✓ Simple to use
- × Network latency
- × Potential costs/limits

Local (Llama, Mistral)

- ✓ No network lag, private
- ✓ Free to run (on own hardware)
- × Requires setup & GPU
- × Models can be less capable

Use ML to answer core business questions directly from your features.

- **Clustering (e.g., K-Means)**
  Business Question: "What market segments exist?"
  → Groups similar apartments to create profiles like "Luxury Penthouses" or "Student Studios".

- **Regression (e.g., Random Forest)**
  Business Question: "Is this apartment priced correctly?"
  → Predicts rent from features to power a "fair deal" finder or landlord pricing tool.

- **Recommendation (e.g., Cosine Similarity)**
  Business Question: "If a user likes this, what else will they like?"
  → Finds the "closest" apartments based on features to improve user discovery.

Networks provide a unique, visual way to understand connections.

Why Use Networks? They excel at showing the *structure* of relationships hidden in the data.

- **Amenity Co-occurrence Network**
  Nodes = amenities, Edges = they appear together often.
  → Reveals what defines a "premium" apartment (e.g., "elevator", "balcony", and "parking" form a tight cluster).
- **Apartment Similarity Network**
  Nodes = apartments, Edges = high feature similarity.
  → Identifies direct competitors, finds comparables, and reveals market sub-clusters.
- **Location Network**
  Nodes = apartments, Edges = on the same street.
  → Analyze hyper-local market dynamics and price variations within a neighborhood.

Note: This can be a powerful differentiator for your pitch, but make sure your core value proposition is working first. It's an extension, not a requirement.

Component 4: Designing a User-Centric Interface

Your UI's job: Make your analysis interactive and your insights understandable.

- **Pattern 1: The Interactive Filter**
  The user configures inputs to see a specific slice of the data.
  - **Layout:** Controls in a sidebar, results (charts, map, tables) in the main area.
  - **Use Case:** Perfect for dashboards and general data exploration.
- **Pattern 2: The "Drill-Down" Explorer**
  The user starts with a high-level view and clicks to reveal more detail.
  - **Layout:** A single, rich visualization (like a map) is the primary interface. Selecting a point reveals its details.
  - **Use Case:** Ideal for geographic analysis or finding specific items in a large dataset.
- **Pattern 3: The "What-If" Simulator**
  The user adjusts parameters to see how a model's prediction changes in real-time.
  - **Layout:** Dominated by sliders and input boxes that feed directly into your model.
  - **Use Case:** Excellent for demonstrating a price predictor or a recommendation engine.

Design Tip: Start by asking: "What question is my user trying to answer?" Then, build the simplest possible interface to help them find that answer.

Integration: Building Modular, Maintainable Systems

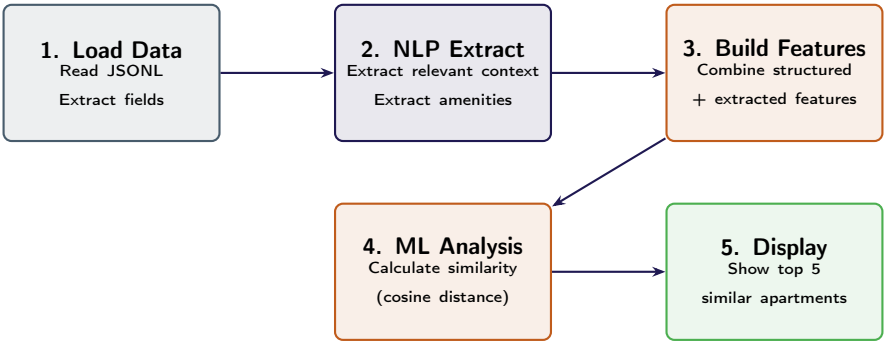
Good integration means components can be modified or replaced without breaking the whole system.

Principles of Modular Integration:

- **Clear Input/Output Contracts**
  Each component should have well-defined inputs and outputs.
  - *Example:* NLP component takes raw text → returns feature dictionary
  - ✓ You can swap NLP methods (regex → LLM) without touching other code
- **Separation of Concerns**
  Keep data processing, analysis, and presentation in separate functions/files.
  - *Example:* `data.py`, `analysis.py`, `app.py`
  - ✓ Debug ML model without worrying about UI, fix UI without touching ML
- **Test Components Independently**
  Build and validate each component with dummy/sample data first.
  - *Example:* Test clustering on 50 apartments before running on full dataset
  - ✓ Catch bugs early, iterate faster, integration is easier
- **Incremental Integration**
  Connect components one at a time, testing after each connection.
  - *Example:* Data loading → Add NLP → Test → Add ML → Test → Add UI
  - ✓ Always have a working version, know exactly what broke if something fails

Conceptual Integration Flow

How components work together: A simple apartment recommendation system

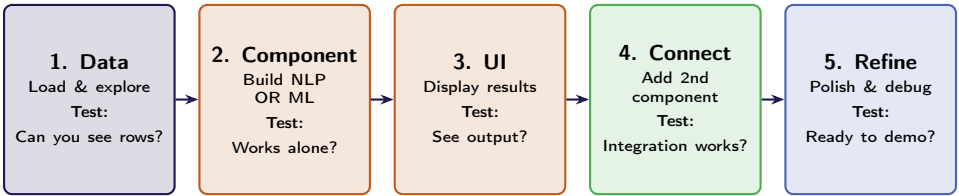


Key Point: Each box is an independent function/module. You can:

- Replace step 2 (regex → LLM) without touching steps 4-5
- Swap step 4 (cosine → network distance) without changing steps 1-3
- Improve step 5 (table → interactive map) without touching analysis

Development Strategy: Build Incrementally

The key to success: Always have something working



Core Principle: Each step work independently and as intended. Test before moving forward. Never integrate broken components.

Evaluation and Execution

Stakeholder Pitch Format (5 minutes)

Your pitch should tell a clear story: Problem → Solution → Impact

1 minute
Problem & Value
Who is this for?
What problem?
Why does it matter?

2-3 minutes
LIVE Demo
Show the app
Key features
Main insights

1 minute
How It Works
Technical approach
Components
Integration

30 seconds
Next Steps
Key insights
Extensions
Potential impact

Do:

- Practice your demo beforehand
- Have backup screenshots
- Focus on business value
- Be concise and clear

Don't:

- Read code line-by-line
- Apologize for what's missing
- Go over time
- Debug live on stage

Evaluation Criteria

How your POC will be assessed:

Business Value - Clear problem, relevant solution, articulated value

Technical Execution - App works, appropriate methods, integrated

Presentation - Clear pitch, effective demo

Insight Quality - Novel findings

Requirements and Enhancements

What you need to deliver, and how you can stand out:

Core Requirements (Everyone Must Deliver):

- ✓ **Text Analysis:** Extract features from descriptions (keywords, LLM extraction, embeddings)
- ✓ **ML/Analysis:** One working component (clustering, regression, recommendations, or network)
- ✓ **Application:** Working Streamlit app with interactivity
- ✓ **Value Proposition:** Clear problem statement and demo-ready pitch

Ways to Enhance (Choose What Fits Your Solution):

- **Deeper Integration:** Components genuinely enhance each other (e.g., network metrics as ML features)
- **Sophisticated Analysis:** Advanced NLP (semantic search), complex ML (ensemble models), or network analysis
- **Multiple Components:** Combine NLP + ML + Network in meaningful ways
- **Novel Insights:** Discover patterns or approaches beyond the obvious
- **Polished Experience:** Intuitive UI, compelling visualizations, professional presentation

Final Reminders

Before you start building, remember these principles:

- **Start with Business Value, Not Technology**
  Ask "What problem am I solving?" before "What algorithm should I use?"
- **Sketch Before Coding**
  5 minutes planning architecture = 30 minutes saved debugging
- **Test Components Independently**
  Don't wait until integration to see if things work
- **Use the Checkpoint to Course-Correct**
  At 1 hour: Stop, evaluate, adjust if needed
- **Prepare Demo Backup**
  Screenshots in case app breaks during presentation
- **Focus on Storytelling**
  Why does this matter? Who benefits? What changes?