# Module 9 Summary: ML Deployment

Data Science with Python – BSc Course

# What We Covered

**Module 9: ML Deployment (L41–L44)**

We took your trained models from prototype to production across 4 lessons:

- **L41 – Model Serialization:** Save and load models with joblib, pickle, ONNX
- **L42 – FastAPI:** Build REST APIs to serve predictions
- **L43 – Streamlit Dashboards:** Create interactive web apps in pure Python
- **L44 – Cloud Deployment:** Deploy apps to the cloud with Streamlit Cloud

**The Journey:**

- Train model → serialize → API → dashboard → cloud
- From local notebook to production URL in 4 lessons

**You now have the complete deployment pipeline from training to production**

# L41 Recap: Model Serialization

**Problem Solved:** Save trained models to disk and load them for deployment without retraining.

**Key Takeaways:**

- **joblib:** Best for sklearn models (fast, compression support)
- **pickle:** Universal Python serializer (but slower for ML)
- **ONNX:** Framework-agnostic, best for production inference
- **Metadata:** Always save features, version, scores with your model

**Code Pattern:**

- Save: `joblib.dump(model, 'model.joblib', compress=3)`
- Load: `model = joblib.load('model.joblib')`
- Best practice: Save entire pipeline (scaler + model), not just model

**joblib for sklearn. pickle for general Python. ONNX for production. Always save metadata!**

# L42 Recap: FastAPI

**Problem Solved:** Serve ML predictions via a web API that any application can consume.

**Key Takeaways:**

- **FastAPI:** Modern, fast, auto-documented Python API framework
- **Pydantic schemas:** Validate input/output automatically
- **Load once:** Load model at startup, predict on every request
- **Swagger docs:** Generated automatically at `/docs`

**Code Pattern:**

- `@app.on_event("startup"): load_model()`
- `@app.post("/predict", response_model=Output)`
- `def predict(input: Input): return model.predict(...)`
- Run: `uvicorn main:app --reload`

**FastAPI + Pydantic + joblib = production ML API. Swagger docs at /docs.**

# L43 Recap: Streamlit Dashboards

**Problem Solved:** Create interactive web apps for data exploration and ML predictions using only Python.

**Key Takeaways:**

- **Streamlit:** Python script → web app (no HTML/CSS/JS)
- **Widgets:** Sliders, selectboxes, file uploads for user input
- **Caching:** `@st.cache_data` for data, `@st.cache_resource` for models
- **Layouts:** Sidebar, columns, tabs for organization

**Code Pattern:**

- `st.title('Dashboard')`
- `ticker = st.sidebar.selectbox('Stock', tickers)`
- `st.plotly_chart(fig)`
- Run: `streamlit run app.py`

**Streamlit: from Python script to web app in minutes. Widgets return values automatically.**

# L44 Recap: Cloud Deployment

**Problem Solved:** Deploy ML apps to the cloud so anyone can access them via a URL.

**Key Takeaways:**

- **Streamlit Cloud:** Free, easy deployment from GitHub
- **requirements.txt:** List all dependencies with pinned versions
- **Secrets management:** Never commit API keys to git
- **Monitoring:** Check logs and analytics for deployed apps

**Deployment Workflow:**

- Test locally → create requirements.txt → push to GitHub
- Connect repo on Streamlit Cloud → configure secrets → deploy
- Share URL with users

**Push to GitHub → deploy on Streamlit Cloud → share URL. That simple.**

# Key Concepts Mastered

**The Complete Deployment Pipeline:**

- **Serialization:** Preserve trained models for reuse
- **APIs:** Programmatic access to model predictions
- **Dashboards:** Interactive UIs for non-technical users
- **Cloud:** Global accessibility and automatic scaling

**Production Best Practices:**

- Version your models (filename with date/version)
- Cache expensive operations (data loading, model inference)
- Handle errors gracefully (validation, try/except, status codes)
- Monitor deployed apps (logs, analytics, error tracking)

**These skills transform you from researcher to production ML engineer**

# Finance Applications

**Real-World Use Cases:**

- **Risk Dashboards:** Streamlit app showing portfolio VaR, stress tests, exposures
- **Trading APIs:** FastAPI serving buy/sell signals to algorithmic traders
- **Client Portals:** Web apps for clients to explore fund performance
- **Compliance Tools:** Deployed apps for regulatory reporting and monitoring

**Industry Examples:**

- Goldman Sachs Marquee: API platform for model predictions
- BlackRock Aladdin: Cloud-deployed risk models at scale
- Stripe ML: Fraud detection API serving millions of calls daily

**Deployment is where ML creates business value in finance**

# Tools & Code Patterns

**Your Deployment Toolkit:**

| Tool | Use Case |
|---|---|
| `joblib.dump/load` | Save/load sklearn models efficiently |
| `FastAPI + Pydantic` | Build type-safe REST APIs |
| `uvicorn` | ASGI server for FastAPI apps |
| `Streamlit` | Interactive dashboards in pure Python |
| `Streamlit Cloud` | Free hosting for Streamlit apps |
| `requirements.txt` | Dependency management |
| `pipreqs` | Auto-generate requirements.txt |
| `.streamlit/secrets.toml` | Secure API key management |

**Memorize these patterns – you'll use them in every ML project**

# What's Next: Module 10 – Capstone & Ethics

**The Final Module:**

- **L45 – Project Work 1:** Start your end-to-end ML project
- **L46 – Project Work 2:** Continue building and refining
- **L47 – ML Ethics:** Bias detection, fairness, explainability, compliance
- **L48 – Final Presentations:** Present your complete project

**What You'll Build:**

- Complete ML pipeline: data → training → API → dashboard → cloud
- Finance application: portfolio optimization, risk prediction, or trading strategy
- Deployed live demo with URL for your resume

**Plus Ethics:** Responsible AI in finance – bias, fairness, explainability, regulation

**Technical skills plus ethical awareness make you a complete data scientist**

## Module 9 Complete!

You can now take any ML model from idea to production.

Ready for Module 10: Capstone & Ethics