# Session 4.1: Integration of Business Analytics and Enterprise AI Architecture: End-to-End Decision Systems

## THE BUSINESS SITUATION

- A model that lives in an analyst's notebook creates no value; value comes only when its predictions trigger real actions, every day, without a person running the code
- The card issuer from Session 3.4 wants its fraud model to score each night's transactions automatically and hold the risky ones
- The head of operations asks: how do we put the fraud model to work every night, and how will we know when it starts to fail? Today, you save a trained model, reuse it on new data, and build the checks that warn you when the world changes

Today's flow: 60 min lecture, then 120 min hands-on lab

## EXPECTED OUTCOMES

### By the end of today, you will be able to:

1. ***"How does a model get from training to daily use?"***
   Map an end-to-end decision system (data in, checks, model, action), and save and reload a model with `joblib`
2. ***"Is new data still like the training data?"***
   Detect data drift (a change in the incoming data that makes a model less reliable) by comparing averages
3. ***"What if something breaks at night?"***
   Monitor latency and accuracy from a run log, and handle a missing model file (`FileNotFoundError`) with a safe fallback

# Session Map: Follow the Numbers

**Today's question:** *How do we put the fraud model to work every night, and how will we know when it starts to fail?*

## 1 Why it matters
Lecture, part 1

1. **Session Purpose**
   answer today's question
2. **Learning Outcomes**
   3 skills you will show today
3. **Business Context**
   From a Model to a Decision System
4. **Data Basics**
   Saving a Model and Watching It

→

## 2 Learn the tools
Lecture, part 2

5. **Lab Dataset**
   3 CSVs
6. **Keywords**
   6 terms
7. **Core Foundations**
   5 ideas, one at a time
8. **IDE Setup**
   `pip install scikit-learn joblib`
9. **Code Examples**
   3 demos + Quick Checks
10. **References**
    4 official docs

→

## 3 Practice
Lab

11. **Exercise 1: Guided**
    Read the Run Log
12. **Exercise 2: Applied**
    Survive a Missing Model File
13. **Exercise 3: Advanced**
    One Function for the Whole Night

# From a Model to a Decision System

## KEY POINTS

- A decision system has four stages, run every night as one batch (a group of records processed together):
- Batch vs real-time:
  - **Batch** scoring runs on a schedule, such as every night, for many records at once. It is cheaper, and fine when decisions can wait a few hours.
  - **Real-time** scoring answers one request within milliseconds, for example while a customer waits at the checkout. It is faster but costs more to build and run.
- Models get worse over time, because customers and markets change.
  - **Data drift:** the new data no longer looks like the training data (for example, purchases are much larger in the holiday season)
  - **Performance decay:** accuracy on recent, checked cases falls

## SEE IT ON REAL DATA

| Stage | What happens | Example |
|---|---|---|
| Ingestion | collect the day's data | 120,000 card transactions |
| Validation | check it before use | no missing values, all columns present |
| Scoring | the model gives each record a probability | fraud probability from 0 to 1 |
| Action | a business rule turns scores into actions | hold every transaction at 0.5 or above |

# Saving a Model and Watching It

**KEY POINTS**

- Training is slow and done rarely; scoring is fast and done often.
  - `joblib.dump(model, "models/fraud_model.joblib")` saves it
  - `joblib.load("models/fraud_model.joblib")` loads it back, ready to predict, with no retraining
- A simple drift check compares each feature's average in the new batch with its average in the training data:
  - change in percent = (new average - training average) / training average x 100
  - Our rule: flag a feature when the change is more than 20% in either direction
- A run log records every night's stages with their record counts and latency (how long a stage took, in milliseconds; 1,000 ms = 1 second).

# The Data for Today

## FILES

- `data/session_03_04_dataset.csv`
- `data/session_04_01_new_batch.csv`
- `data/session_04_01_dataset.csv`

Run log columns:

## COLUMNS

| | |
|---|---|
| `Run_Date` | The night of the run, `2024-12-01` to `2024-12-05`. |
| `Pipeline_Stage` | `Ingestion`, `Validation`, `Scoring`, or `Action`. |
| `Records_Count` | Records processed by the stage. For `Action`, the number of held transactions. |
| `Latency_Ms` | Time the stage took, in milliseconds. |
| `Model_Accuracy` | Accuracy on that night's investigated cases. Filled only for the `Scoring` stage; empty (`NaN`) for the others. |

# 6 Terms You Need Today

**TERM 1**

`End-to-End Decision System`

**What:** ingestion, validation, scoring, and action joined into one automatic run

**Why:** turns predictions into business actions without manual steps

**TERM 2**

`Batch vs Real-Time Scoring`

**What:** many records on a schedule vs one record on request

**Why:** the business deadline decides which one is worth its cost

**TERM 3**

`Model Persistence` (`joblib`)

**What:** saving a trained model to a file and loading it later

**Why:** train once, score every night

**TERM 4**

`Data Drift`

**What:** new data that no longer looks like the training data

**Why:** a model is only reliable on data like the data it learned from

**TERM 5**

`Monitoring`

**What:** tracking counts, latency, and accuracy for every run

**Why:** catches a failing system before customers do

**TERM 6**

`Fallback`

**What:** a simple, safe rule used when the model is not available

**Why:** the business keeps running when something breaks

# How It Works

7.1

## Train once, use many times

**Remember:** save the whole pipeline once; load it every night.

7.2

## Probabilities, not just yes or no

**Remember:** use probabilities when the cut-off is a business decision.

7.3

## Data drift

**Remember:** compare new averages with training averages, in both directions.

7.4

## Monitoring

**Remember:** log every run, then watch the trends, not just single nights.

7.5

## When something breaks

**Remember:** plan for failure, and always report which method made the decisions.

# 7.1 Train once, use many times: On Real Data

**SEE IT ON REAL DATA**

| Step | Code | Result |
|---|---|---|
| Save | `joblib.dump(model, "models/fraud_model.joblib")` | a small file, about 1.7 KB |
| Load | `loaded = joblib.load("models/fraud_model.joblib")` | the same trained model |
| Check | `(model.predict_proba(X) == loaded.predict_proba(X)).all()` | `True`: identical predictions |

**Remember:** save the whole pipeline once; load it every night.

# Train Once, Save to a File

YourName_S41_demo_1.py — Input → Process → Output

```
# 1. Input: the investigated transactions from Session 3.4
import os
import joblib
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
train = pd.read_csv("data/session_03_04_dataset.csv")
features = ["Transaction_Amount_USD", "Foreign_IP_Flag",
            "Failed_Login_Attempts_Count"]

# 2. Process: scale + logistic regression, trained in one Pipeline
model = Pipeline([
    ("scale", StandardScaler()),
    ("classify", LogisticRegression()),
])
model.fit(train[features], train["Actual_Fraud_Binary"])

# 3. Output: save the trained model as a file
os.makedirs("models", exist_ok=True)
joblib.dump(model, "models/fraud_model.joblib")
print("Saved: models/fraud_model.joblib")
print("Training accuracy:", model.score(train[features], train["Actual_Fraud_Binary"]))
```

## HOW IT WORKS

- First time only, in the Terminal: `pip install scikit-learn joblib`
- The training data is the 16 investigated transactions from Session 3.4; the model uses 3 features.
- `Pipeline([("scale", StandardScaler()), ("classify", LogisticRegression())])` joins two steps into one object: first standardize, then classify.

## READ THE OUTPUT

```
Saved: models/fraud_model.joblib
Training accuracy: 0.9375
```

- The file now exists in the `models` folder, visible in Panel `[1]`.

# Quick Check 1: `os.path.exists()`

NEW TOOL

### `os.path.exists()`

`os.path.exists(path)` returns `True` if the file exists. Why: the nightly job should confirm the model is there before it starts.

YOUR TASK

### Fill in the Blank

Add one line to part **3. Output**:

```
print(os.path._____("models/fraud_model.joblib"))
```

**Check:** the Terminal shows `True`.

# The Nightly Job: Load, Score, Decide

YourName_S41_demo_2.py — Input → Process → Output

```
# 1. Input: the saved model and tonight's new transactions
import joblib
import pandas as pd
model = joblib.load("models/fraud_model.joblib")
batch = pd.read_csv("data/session_04_01_new_batch.csv")
features = ["Transaction_Amount_USD", "Foreign_IP_Flag",
            "Failed_Login_Attempts_Count"]

# 2. Process: fraud probability, then the business rule (hold at 0.5 or above)
batch["Fraud_Probability"] = model.predict_proba(batch[features])[:, 1].round(3)
batch["Decision"] = "Pass"
batch.loc[batch["Fraud_Probability"] >= 0.5, "Decision"] = "Hold"

# 3. Output: print and save tonight's decisions
print(batch[["Transaction_ID", "Fraud_Probability", "Decision"]])
batch.to_csv("data/nightly_decisions.csv", index=False)
```

## HOW IT WORKS

- `joblib.load(...)` brings back the trained pipeline: no training happens here, so this script runs in a fraction of a second.
- `model.predict_proba(batch[features])[:, 1]` gives each new transaction its fraud probability (section 7.2).
- `batch["Decision"] = "Pass"` sets every row to "Pass"; `batch.loc[batch["Fraud_Probability"] >= 0.5, "Decision"] = "Hold"` changes only the risky rows.

## READ THE OUTPUT

```
  Transaction_ID  Fraud_Probability Decision
0     N-TXN-9001              0.862     Hold
1     N-TXN-9002              0.070     Pass
...
```

- 4 of 8 transactions are held, all foreign purchases above $2,800.

# Quick Check 2: Run date stamp

NEW TOOL

## Run date stamp

`from datetime import date` and `date.today()` give today's date. Why: every decision file must show when the decisions were made, for audits.

YOUR TASK

## Fill in the Blank

Add `from datetime import date` to part **1. Input**, and one line at the end of part **2. Process**: `batch["Run_Date"] = str(date._____())`

**Check:** the saved `data/nightly_decisions.csv` has a `Run_Date` column with today's date, written as `YYYY-MM-DD`.

# Drift Check: Does Tonight Look Like Training?

YourName_S41_demo_3.py — Input → Process → Output

```
# 1. Input: training data and tonight's batch
import pandas as pd
train = pd.read_csv("data/session_03_04_dataset.csv")
batch = pd.read_csv("data/session_04_01_new_batch.csv")
features = ["Transaction_Amount_USD", "Foreign_IP_Flag",
            "Failed_Login_Attempts_Count"]

# 2. Process: compare the averages, feature by feature
drift = pd.DataFrame({
    "Train_Avg": train[features].mean().round(2),
    "Batch_Avg": batch[features].mean().round(2),
})
drift["Change_Pct"] = (
    (drift["Batch_Avg"] - drift["Train_Avg"]) / drift["Train_Avg"] * 100
).round(1)
drift["Drift_Flag"] = drift["Change_Pct"] > 20

# 3. Output: the drift report
print(drift)
```

## HOW IT WORKS

- `train[features].mean()` and `batch[features].mean()` give each feature's average in the two files.
- Putting both into `pd.DataFrame({...})` lines them up by feature name, one row per feature.
- `Change_Pct` is the change in percent from the training average (section 7.3).

## READ THE OUTPUT

- Tonight's purchases are on average 86% larger than the training purchases: amount has drifted.

# Quick Check 3: `.abs()`

NEW TOOL

`.abs()`

`column.abs()` removes the minus sign. Why: a feature that **falls** by 30% has drifted too; `> 20` alone misses it.

YOUR TASK

## Fill in the Blank

In part **2. Process**, change the flag line to `drift["Drift_Flag"] = drift["Change_Pct"].___() > 20`

**Check:** the flags do not change for tonight's batch, but a change of `-30.0` would now be flagged as well.

# Where to Look Things Up

REFERENCE

## Scikit-Learn Model Persistence Guide (Joblib Documentation)

https://scikit-learn.org/stable/model_persistence.html

REFERENCE

## Python Library: os.path (Checking Files and Paths)

https://docs.python.org/3/library/os.path.html

REFERENCE

## MLflow Official Documentation: Open Source Platform for the Machine Learning Lifecycle

https://mlflow.org/docs/latest/index.html

REFERENCE

## Sculley et al.: Hidden Technical Debt in Machine Learning Systems (NeurIPS Standards)

https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf

# Three Exercises

EXERCISE 1: GUIDED

## Read the Run Log

**Goal:** Find the nights in the run log that need attention

**File:**
`YourName_S41_monitoring.py`

**Check:** alerts on `2024-12-04` and `2024-12-05`

EXERCISE 2: APPLIED

## Survive a Missing Model File

**Goal:** Keep the nightly job working without the model file

**File:**
`YourName_S41_practice.py`

**Check:** fallback holds 3; the model holds 4

EXERCISE 3: ADVANCED

## One Function for the Whole Night

**Goal:** One function for the whole night: validate, score, act, report

**File:**
`YourName_S41_challenge.py`

**Check:** `held: 4`, `needs_review: True`

Full steps for all three exercises: sections 11-13 of the course page.