# Final Year Project: Air Traffic Control (ATC) — Data Science & ML

**Goal:** Build a prototype ATC decision-support system that ingests simulated/real flight + weather data, visualizes traffic, predicts delays/congestion, detects anomalies, and recommends optimized routing/ landing sequences.

---

## 1. Project Overview (What you'll build)

- **Real-time data pipeline** (or simulated real-time): ingest position (ADS-B/radar-like), flight plans, and weather.
- **Data storage & processing**: cleaned, time-aligned datasets; feature engineering pipeline.
- **Dashboards**: map-based traffic visualization, runway/taxi status, congestion heatmaps.
- **ML modules**: delay prediction, ETA correction, anomaly detection, and simple sequencing/ optimization (heuristic or RL prototype).
- **User interface**: web-based dashboard for controllers (demo only — not full-fledged safety-critical system).
- **Evaluation & report**: quantitative metrics, ablation studies, demo script.

---

## 2. Minimum Scope (MVP) — what to complete for a solid final project

1. Static dataset ingestion and preprocessing.
2. Dashboards: live-simulated playback on map.
3. One predictive model (arrival delay or ETA correction).
4. One anomaly detection model for out-of-pattern trajectories.
5. Short optimization demo: greedy landing-sequencing that reduces average waiting time.
6. Final report, presentation, and demo video.

---

## 3. Suggested Tech Stack

- **Data ingestion / streaming:** Python, Pandas, Apache Kafka (optional) or simple socket/CSV replay script.
- **Storage:** PostgreSQL (timeseries extension optional) or Parquet files; SQLite for prototype.
- **Processing:** Python (Pandas, NumPy), PySpark if scale is large.
- **ML:** scikit-learn, XGBoost/LightGBM, TensorFlow/PyTorch (for LSTM/seq models or RL).
- **Anomaly detection:** scikit-learn (IsolationForest), PyOD, or Autoencoders in PyTorch.
- **Web UI / Dashboard:** Streamlit or Dash for faster prototyping; Leaflet or Mapbox for maps if using JS.

- **Visualization:** Plotly, Folium (for interactive maps), Matplotlib.
- **Dev & Deployment:** Git, Docker (optional), Heroku or Streamlit Sharing for demo.

---

## 4. Week-by-Week Step-by-Step Plan (12-week schedule)

### Week 1 — Project setup & research

- Define scope & deliverables (use MVP above).
- Search and gather datasets (see Appendix for dataset ideas).
- Setup repo, virtualenv, and basic README.
- Draw system architecture diagram.

### Week 2 — Data collection & simulation

- If real ADS-B data unavailable, find/open datasets or write a simulator that replays CSVs with timestamps.
- Store raw data in a folder or DB.

### Week 3 — Data cleaning & EDA

- Clean missing values, synchronize timestamps, convert coordinates.
- EDA: traffic counts, busiest hours, route heatmaps.
- Save cleaned datasets.

### Week 4 — Build visualization & playback

- Build a simple map playback to show flight tracks over time.
- Add controls: play/pause, speed, filter by aircraft.

### Week 5 — Feature engineering

- Create features: speed, heading, distance to airport, scheduled vs actual departure, weather features (wind, visibility).
- Aggregate features per flight and per time-window.

### Week 6 — First predictive model (ETA / delay)

- Train baseline model (linear regression / Random Forest) to predict arrival delay.
- Evaluate with MAE, RMSE, $R^2$.
- Save best model.

### Week 7 — Anomaly detection

- Implement Isolation Forest or autoencoder to flag unusual trajectories or sudden altitude changes.
- Create a dashboard panel to list flagged flights.

### Week 8 — Optimization prototype

- Implement a greedy sequencing algorithm for landing slots (minimize total delay/holding time).
- Compare metrics with naive FIFO.

### Week 9 — Integrate models into pipeline

- Hook models to the live-simulated playback: predictions and anomaly flags appear in UI.

### Week 10 — Evaluation, ablation, and robustness

- Run experiments, produce tables & plots of model performance across subsets (weather vs no-weather).
- Sensitivity tests: missing data, noisy positions.

### Week 11 — Write report & prepare slides

- Draft chapters: Introduction, Related Work, Data, Methods, Results, Discussion, Conclusion.
- Create slides and demo script.

### Week 12 — Final polishing & demo recording

- Fix UI bugs, ensure reproducibility, package environment (requirements.txt or Dockerfile).
- Record a 5–10 minute demo video and rehearse presentation.

---

## 5. Data: Where to get it / How to simulate

- **Open datasets (examples to search):** ADS-B data archives, OpenSky Network, FAA datasets, Eurocontrol datasets, NOAA weather data.
- **If blocked:** create a simulator that generates flights using realistic waypoints and noise. Simulate weather effects (wind delay, hold patterns).
- **Fields needed:** timestamp, aircraft_id/callsign, lat, lon, altitude, groundspeed, heading, flight_plan_id, scheduled_arrival, actual_arrival (if available), wind_speed/direction, visibility.

---

## 6. System Architecture (high-level)

1. **Data source** (CSV replay / API / simulator)
2. **Ingestion** (Python script or Kafka producer) → timestamped records
3. **Storage** (Parquet / PostgreSQL)
4. **Processing & Feature Store** (Pandas/Spark pipeline)
5. **ML Models** (training pipeline + model serving endpoint OR in-app inference)
6. **Dashboard** (Streamlit/Dash) — Map + Model outputs
7. **Evaluation/Reporting** (notebook + figures)

Sketch this diagram in your report.

---

## 7. Key Implementation Details & Example Code Snippets

### 1) Simple CSV replay (simulate streaming)

```
import time
import pandas as pd

df = pd.read_csv('flights.csv', parse_dates=['timestamp'])
start = df['timestamp'].min()
for _, row in df.sort_values('timestamp').iterrows():
    # emulate real-time delay
    time.sleep(0.01)  # speed up for demo
    process_record(row.to_dict())
```

### 2) Feature: distance-to-runway (Haversine)

```
from math import radians, sin, cos, sqrt, atan2

def haversine(lat1, lon1, lat2, lon2):
    R = 6371e3
    phi1, phi2 = map(radians, [lat1, lat2])
    dphi = radians(lat2-lat1)
    dlambda = radians(lon2-lon1)
    a = sin(dphi/2)**2 + cos(phi1)*cos(phi2)*sin(dlambda/2)**2
    return 2*R*atan2(sqrt(a), sqrt(1-a))
```

### 3) Simple delay prediction (scikit-learn)

```
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(n_estimators=100)
model.fit(X_train, y_train)
preds = model.predict(X_test)
```

### 4) Anomaly detection (IsolationForest)

```
from sklearn.ensemble import IsolationForest
clf = IsolationForest(contamination=0.01)
clf.fit(X_train)
anomaly_score = clf.decision_function(X_test)
anomaly_flag = clf.predict(X_test)  # -1 anomaly, 1 normal
```

---

## 8. Evaluation Metrics

- **Prediction:** MAE, RMSE, $R^2$, calibration plots.
- **Anomaly detection:** precision@k, recall, confusion matrix (if labeled anomalies exist), qualitative inspection.
- **Optimization:** average waiting time, total delay, fuel burn proxy (time × fuel rate).
- **System:** latency of inference (ms), throughput (records/sec).

---

## 9. Deliverables (what to submit)

- GitHub repo with code + instructions (README).
- `requirements.txt` or `environment.yml` / Dockerfile.
- Cleaned dataset or simulation script (if dataset cannot be shared).
- Final report (PDF) with diagrams, tables, discussion.
- Presentation slides (5–10 slides) and demo video (5–10 minutes).
- Short user guide to run the demo locally.

---

## 10. Extensions & Extra Credit Ideas

- Train an LSTM/transformer to predict trajectories or ETA.
- Use Reinforcement Learning for landing sequencing (simple environment only).
- Integrate real-time weather API for live demos.
- Add multi-airport coordination and flow management.
- Add voice-simulated controller instructions (NLP).

---

## 11. Risks, Ethics & Limitations (important to include in report)

- This project is a **prototype** and not certified for operational ATC use.
- Safety-critical systems require rigorous certification; avoid claiming operational readiness.
- Discuss biases in datasets (geographic, seasonal) and data privacy (aircraft identifiers).

---

## 12. Report Structure (suggested)

1. Title & Abstract
2. Introduction & Motivation
3. Related Work
4. Data Description
5. System Architecture
6. Methods (Feature engineering, Models)
7. Experiments & Results
8. Discussion (limitations & ethics)

9. Conclusion & Future Work
10. Appendix (code snippets, instructions)

---

## Appendix — Quick checklist before demo

- [ ] Repo runs with `pip install -r requirements.txt`.
- [ ] Demo script: `python run_demo.py`.
- [ ] Dataset or simulator ready and documented.
- [ ] UI shows map + predictions + anomaly list.
- [ ] Results & plots in report.

---

Good luck — you can follow this guide strictly, or pick and choose features depending on your timeline. If you want, I can now: - Generate a **template repo structure** (file/folder layout + README). - Create a **Streamlit dashboard template** with map playback. - Provide **starter code** for model training and evaluation.

Which of those would help most next?