TECHNICAL MANUAL

# SL 4G

USAREUR-AF

## TM-40G — MAVEN SMART SYSTEM (MSS)

*Specialist Course Manual*

HEADQUARTERS
UNITED STATES ARMY EUROPE AND AFRICA
(USAREUR-AF)
Wiesbaden, Germany



**26 MARCH 2026**

DRAFT — UNOFFICIAL — NOT FOR OPERATIONAL USE

# TM-40G — MAVEN SMART SYSTEM (MSS)

**Forward:** ORSA analysts translate operational questions into mathematical models and translate results into commander-actionable recommendations. This manual provides the technical procedures for conducting that work on MSS/Foundry using Code Workspaces. **Prereqs:** SL 1, Maven User; SL 2, Builder; SL 3, Advanced Builder; Data Literacy Technical Reference (all required). Quantitative background equivalent to graduate-level statistics, and working proficiency in R or Python, required before beginning this manual; CONCEPTS_GUIDE_TM40G_ORSA (read before this manual). *HQ USAREUR-AF · v1.0 · 2026 · DISTRIB: USG only · AUTH: C2DAO/UDRA v1.1*

**WARNING: Analytical products without uncertainty quantification may cause commanders to make decisions with false precision. Confidence intervals, sensitivity ranges, and explicit model limitations are MANDATORY components of every ORSA deliverable presented to a commander. CAUTION: Connecting Foundry datasets to external tools (JDLM, spreadsheet exports, R/Python libraries outside the Foundry environment) must be coordinated with C2DAO. Unauthorized data movement violates UDRA v1.1 data governance requirements. NOTE: ORSA analysts are advisors, not decision-makers. The output of every model is an input to a commander's judgment, not a substitute for it. Design your products accordingly.**

## CHAPTER 1 — INTRODUCTION: ORSA ROLE, SCOPE, AND ANALYTICAL PRODUCT STANDARDS

**BLUF:** ORSA analysts translate operational questions into mathematical models and translate results into commander-actionable recommendations. This manual provides the technical procedures for conducting that work on MSS/Foundry using Code Workspaces.

### 1-1. ORSA Specialist Manual

This manual provides task-level instruction for ORSA officers, NCOs, and quantitative analysts conducting operations research and systems analysis on the Maven Smart System (MSS). MSS is the USAREUR-AF enterprise AI/data platform built on Palantir Foundry.

**SL 4G covers** configuring and using Code Workspaces (R and Python environments) within Foundry; connecting Foundry datasets to Code Workspace analysis environments; statistical modeling: regression, classification, and model validation for readiness and logistics applications; time series forecasting: readiness trend analysis and logistics demand forecasting using ARIMA/SARIMA patterns; Monte Carlo

simulation: risk analysis for COA comparison, logistics planning uncertainty quantification; optimization: linear programming for resource allocation and scheduling against operational constraints; wargame and exercise analysis: data collection architecture, aggregation pipelines, outcome measurement; building analytical decision support products in Quiver and Contour for commander consumption; uncertainty communication standards: confidence intervals, sensitivity analysis, briefing product standards; and connecting MSS data to external ORSA tools following C2DAO-approved data handling procedures.

**SL 4G does NOT cover** general Foundry pipeline development — see SL 3; ontology design — see SL 3; non-analytical Workshop applications — see SL 2 and SL 3; machine learning model deployment to production — see SL 4M; or data architecture and schema design — see SL 3 and SL 4H.

## 1-2. Curriculum Position, Advanced Track, and WFF Context

> **NOTE**
>
> The **EUCOM Thunderforge AI Planning Ecosystem** (DIU/Scale AI contract, 2024–25) employs AI agents to simulate wargaming and planning scenarios at theater level, enabling AI-augmented MDMP and compressing planning timelines. ORSA analysts trained on MSS are essential participants in Thunderforge-class initiatives — the quantitative decision-support skills in SL 4G (and advanced methods in SL 5G) directly enable the human-machine teaming required for AI-assisted wargaming and COA analysis.

**Prerequisite:** SL 3 (Advanced Builder) is REQUIRED. Completion of SL 2 and SL 1 is assumed. A graduate-level quantitative background (statistics, R or Python) is required independently of the TM series.

**Advanced track:** Upon completing SL 4G, qualified ORSAs should pursue **SL 5G (Advanced ORSA)** for advanced topics including Bayesian methods, complex simulation design, multi-objective optimization, and campaign analysis support to USAREUR-AF operational planning.

**Peer specialist tracks:** The ORSA collaborates closely with the ML Engineer (SL 4M) — the ORSA owns the analytical question and decision product; the MLE owns continuously-operating prediction pipelines at scale. Coordinate with SL 4H (AI Engineer) when wrapping ORSA analysis in automated LLM synthesis workflows. Coordinate with SL 4L (Software Engineer) for production-grade pipeline implementation and OSDK-backed delivery interfaces.

**WFF awareness:** ORSA analysts on MSS serve as quantitative decision support across all Warfighting Functions. WFF-qualified users (SL 4A through SL 4F — Intelligence, Fires, Movement and Maneuver, Sustainment, Protection, and Mission Command) are primary consumers of ORSA products. Design analytical products with WFF staff officers as the intended audience: commanders want decision-grade products, not statistical reports. The ORSA's decomposition framework (Decision → Information → Data → Method) applies regardless of which WFF the tasker originates from.

## 1-2a. ORSA in the Decision Optimization Discipline

The primary reference for the ORSA role in data teams is the XVIII ABC experience:

**Forney, Herrmann, and Steele** ("Fighting with Live Data," February 2026) document ORSA employment at XVIII Airborne Corps, where the Data Scientist (49A/D4 ASI) is one of five core ODT roles. Their BDA visualization case study — prototype in 3 months, MVP in 6, POR handoff in 9 — demonstrates the operational tempo at which ORSA-trained analysts produce operationalized data products within a multifunctional team. This article documents the most directly applicable precedent — authored by a COL, MAJ, and CPT with direct responsibility for the XVIII ABC ODT pilot.

**Supplementary context:** Adkins ("Achieving Decision Dominance," January-February 2025) is a thought piece proposing terminology for data team employment. Adkins identifies FA 49 (ORSA) as one of three core functional areas for what he terms Decision Optimization Teams (DOTs). His shorthand — AFP, operationalized data, DOT — has entered common usage and provides useful framing, but the article is one officer's proposal, not doctrine.

**What this means for SL 4G graduates:** You are not a standalone analyst producing reports on request. In the ODT model, you are an embedded team member producing **Automated Fighting Products** — visualization tools connected to live data via automated pipelines that reduce staff burden and inform commander decisions. Your products must be operationalized (immediately actionable by the consumer) and your methods must be transparent (uncertainty quantification, model limitations, confidence intervals). The MSS platform is the delivery mechanism; your quantitative skills are the engine.

> **NOTE**
>
> Adkins names the Maven Smart System as an ASCC-level COP platform. The analytical products you build on MSS in SL 4G align with the AFP concept.

*Sources: Forney et al., "Fighting with Live Data," Military Review Online Exclusive (2026); Adkins, "Achieving Decision Dominance," Military Review 105, no. 1 (2025).*

## 1-3. The ORSA Role in USAREUR-AF

ORSA analysts provide the USAREUR-AF commander and staff with quantitative decision support at every phase of the operations process. Table 1-1 maps ORSA activities to staff functions.

**Table 1-1. ORSA Support to Staff Functions**

| Staff Function | ORSA Support | Typical Product |
|---|---|---|
| G2 (Intelligence) | Force correlation modeling, threat probability analysis | COA probability distributions, RED force effectiveness estimates |

| Staff Function | ORSA Support | Typical Product |
|---|---|---|
| G3 (Operations) | Wargame analysis, COA comparison, decision point analysis | Comparative utility matrices, maneuver optimization models |
| G4 (Logistics) | Supply demand forecasting, Class IX consumption modeling | Demand signal forecasts, stockage level recommendations |
| G7 (Training) | Readiness trend analysis, training effectiveness measurement | C-rating forecasts, training return-on-investment analysis |
| G8 (Finance) | Resource allocation optimization, budget modeling | Resource efficiency analyses, programming recommendations |
| C2DAO | Analytical methodology governance, product standards | ORSA methodology reviews, model documentation |

## 1-4. Prerequisites

Before beginning this manual, verify you meet all prerequisites:

**Platform prerequisites:** - ☐ SL 1 complete — can operate MSS, access datasets, use Workshop applications - ☐ SL 2 complete — can build basic Pipeline Builder transforms and Workshop applications - ☐ SL 3 complete — can design ontology models, advanced pipelines, and Contour analyses - ☐ Code Workspace access provisioned — confirm with unit MSS Administrator

**Quantitative prerequisites:** - ☐ Linear algebra: matrix operations, eigenvalues, least squares - ☐ Probability and statistics: distributions, hypothesis testing, confidence intervals, regression - ☐ Time series fundamentals: stationarity, autocorrelation, ARIMA - ☐ Optimization: linear programming, constraint formulation, objective functions - ☐ Proficiency in R (tidyverse, forecast, lpSolve) or Python (numpy, pandas, statsmodels, scipy) - ☐ Familiarity with simulation concepts (random number generation, sampling methods)

**NOTE**

This manual assumes graduate-level quantitative proficiency. Statistical procedures are described at an implementation level, not an introductory level. Analysts without the prerequisite background should complete formal coursework before applying these procedures to operational products.

## 1-5. Governing References

| Publication | Title | Relevance |
|---|---|---|
| DA PAM 600-3 | Officer Professional Development | Defines FA 49 (ORSA) career progression and qualifications |
| AR 5-11 | Management of Army M&S | M&S governance policy |
| DA PAM 5-11 | VV&A of Army Models and Simulations | Model credibility and accreditation standards |
| ATP 5-0.3 | Multi-Service TTP for Operation Assessment | Assessment methodology, MOE/MOP analytical frameworks |
| AR 71-9 | Warfighting Capabilities Determination | Capabilities-based assessments — core ORSA analytical work |
| ADP 5-0 | The Operations Process | Analytical framework for plan/prepare/execute/assess |
| FM 5-0 | Planning and Orders Production (Nov 2024) | MDMP, COA evaluation criteria schema, assessment framework (MOE/MOP/Indicators), force ratio |
| FM 3-60 | Targeting | CARVER target value analysis methodology (Appendix G) |
| Army CIO Memorandum | Data and Analytics Policy (April 2024) | Data governance authority |

## 1-5a. Strategic Guidance

The following are strategic guidance documents — not doctrine — that inform MSS training design and operational context.

| Document | Authority | Relevance |
|---|---|---|
| UDRA v1.1 | Unified Data Reference Architecture (February 2025) | Technical reference architecture |

## 1-6. Analytical Product Standards

Every ORSA product delivered to a commander or staff officer must meet the following standards. These are not stylistic preferences — they are analytical integrity requirements.

**Standard 1 — Data Provenance:** Every product must document the source datasets, the date range of input data, and any data quality limitations discovered during validation.

**Standard 2 — Assumption Documentation:** Every model must explicitly state its assumptions. If a logistics forecast assumes constant consumption rates, that assumption must appear in the product.

**Standard 3 — Uncertainty Quantification:** Point estimates must be accompanied by confidence intervals or ranges. Forecasts must include prediction intervals. Simulation outputs must report distributions, not just means.

**Standard 4 — Validation Evidence:** Classification and regression models must report out-of-sample performance metrics. Forecasts must report backtesting error on historical data. Simulation parameters must be calibrated to observed data.

**Standard 5 — Sensitivity Analysis:** Products that inform significant resource decisions must include sensitivity analysis showing how conclusions change when key assumptions change.

**Standard 6 — Plain-Language Summary:** Every technical product must include a BLUF paragraph in plain language stating the finding, the confidence level, and the recommended commander action or decision point.

**Standard 7 — Reproducibility:** Analysis code must be stored in the Code Workspace repository. Any analyst on the team must be able to re-run the analysis from the stored code and data and reproduce the same result.

## 1-7. Assessment Framework: MOE, MOP, and Indicators

**BLUF:** ORSA analysts design and maintain the assessment framework that connects observable data to commander decisions. FM 5-0 (Nov 2024), Chapter 8 defines the three-tier structure: Indicators feed Measures of Performance and Measures of Effectiveness; together these answer whether the force is accomplishing its mission.

**Table 1-2. Assessment Measure Types (FM 5-0, Chapter 8)**

| Measure Type | Question Answered | Nature | Example (Readiness Context) |
|---|---|---|---|
| **MOE** (Measure of Effectiveness) | "Did we achieve the desired effect?" | Qualitative or quantitative criteria tied to end state conditions | Percentage of BCTs at C1 readiness within 120 days |
| **MOP** (Measure of Performance) | "Did we complete the task to standard?" | Task completion criteria — binary or scalar | Number of PMCS cycles completed per reporting period |
| **Indicator** | "What observable data feeds the MOE/MOP?" | Reportable data point collected from operations | Daily equipment FMC rate reported via GCSS-Army |

**NOTE**

MOPs tell you whether the force executed the task. MOEs tell you whether executing the task produced the intended effect. An ORSA product that reports only MOPs (task counts) without linking them to MOEs (operational outcomes) provides activity reporting, not assessment. Design every assessment product with explicit MOE-MOP-Indicator linkage.

### COA Evaluation Criteria Schema (FM 5-0, Table 5-2)

When building COA evaluation models on MSS, structure each criterion as a data object with the following fields:

| Field | Definition | Example |
|---|---|---|
| **Short Title** | Concise label for the criterion | `TIME_TO_OBJECTIVE` |
| **Definition** | What the criterion measures, stated as a question | "How quickly does the COA achieve the decisive point?" |
| **Unit of Measure** | Quantifiable unit used for comparison | Hours |
| **Benchmark** | Standard or threshold against which COAs are compared | 72 hours (OPORD planning factor) |
| **Formula** | Calculation method for deriving the measure from raw data | `(Phase 2 end time) − (LD time)` |

**WARNING**

Do not allow COA evaluation criteria to become ad hoc. Every criterion the staff uses for COA comparison must have all five fields defined before the wargame begins. Undisciplined criteria produce undisciplined analysis — and undisciplined decisions.

Implement this schema as a Foundry object type or structured dataset so that evaluation criteria are version-controlled and reusable across planning cycles. See Chapter 5 for Monte Carlo methods that propagate uncertainty through these criteria during COA comparison.

---

## CHAPTER 2 — CODE WORKSPACES FOR ORSA

**CODE EXAMPLES:** Python data pipeline and transform patterns referenced in this chapter are available in the local development shim at `data_skills/13_foundry_patterns/python_transforms.py` and `data_skills/13_foundry_patterns/incremental_transforms.py`. Statistics and ML reference

modules are in `skills/data_skills/05_statistics/` and `skills/data_skills/06_machine_learning/`. Activate the venv: `source skills/data_skills/.venv/bin/activate`.

**BLUF:** Code Workspaces provide R and Python environments with direct access to Foundry datasets. Proper workspace configuration is the foundation for all ORSA work on MSS.

## 2-1. Code Workspace Overview

Foundry Code Workspaces provide containerized development environments (Jupyter notebooks, RStudio-analog interfaces, or direct Python/R scripting) with native access to Foundry datasets, transforms, and the Foundry file system. For ORSA, Code Workspaces are the primary development and analysis environment.

**Key capabilities for ORSA:**

- Direct dataset read/write via the Foundry Datasets API
- Spark-backed computation for large datasets
- Package installation from approved package repositories
- Version-controlled notebooks and scripts
- Integration with Foundry Pipeline Builder for productionizing analysis
- Scheduled execution for recurring analytical products

## 2-2. Task: Configure a New ORSA Code Workspace

**CONDITIONS:** You have SL 3 proficiency, MSS access, and a defined analytical task requiring Code Workspace development. A Foundry project folder has been provisioned for your ORSA team.

**STANDARDS:** Workspace is configured, connects to at least one Foundry dataset, passes a smoke test, and is documented in the team repository.

**EQUIPMENT:** MSS access (browser-based), appropriate dataset read permissions.

**PROCEDURE:**

1. Navigate to your ORSA project folder in Foundry. Select **New** > **Code Workspace**.
2. Name the workspace using the convention: `ORSA_[TASK]_[YYYYMM]` (e.g., `ORSA_ReadinessForecast_202603`).
3. Select the appropriate environment kernel:
4. **Python:** Select the latest Python 3.x kernel with the `foundry-ml` or `foundry-analytics` profile if available
5. **R:** Select the R kernel — note that not all Foundry deployments include R; confirm availability with your MSS Administrator

6. In the workspace, install required packages. For Python:

```
# requirements block — place at top of notebook
# These are installed in the workspace environment
# Do NOT pip install during runtime in production notebooks

# Standard ORSA Python stack
import numpy as np
import pandas as pd
import scipy.stats as stats
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
import matplotlib
matplotlib.use("Agg")  # headless — required for Foundry notebook environments
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score, TimeSeriesSplit
from sklearn.metrics import classification_report, mean_absolute_error
from scipy.optimize import linprog
```

For R:

```
# R package loading block
library(tidyverse)
library(forecast)    # ARIMA/SARIMA, accuracy metrics
library(tseries)     # stationarity tests (adf.test)
library(lpSolve)     # linear programming
library(caret)       # classification workflows
library(Metrics)     # mae, rmse, mape
library(ggplot2)     # visualization
library(lubridate)   # date handling for time series
```

1. Connect to a Foundry dataset. In Python:

```
from foundry import foundry_client

# Initialize Foundry client — uses workspace token automatically
client = foundry_client()

# Load a dataset into a pandas DataFrame
# Replace with your actual dataset RID or path
readiness_df = client.datasets.load_dataset(
    dataset_rid="ri.foundry.main.dataset.YOUR-DATASET-RID",
    branch="master"
).to_pandas()

# Validate load
print(f"Loaded {len(readiness_df)} rows, {len(readiness_df.columns)} columns")
print(readiness_df.dtypes)
print(readiness_df.head(3))
```

In R:

```
library(foundry)

# Initialize client
client <- foundry_client()

# Load dataset
readiness_df <- client$datasets$load_dataset(
  dataset_rid = "ri.foundry.main.dataset.YOUR-DATASET-RID",
  branch = "master"
) |> as_tibble()

cat(sprintf("Loaded %d rows, %d columns\n", nrow(readiness_df), ncol(readiness_df)))
glimpse(readiness_df)
```

1. Run a smoke test: verify row counts match expected volume, spot-check key fields for expected value ranges, check for null rates on critical columns.
2. Create a `README.md` cell or notebook section documenting:
3. Analytical task description
4. Source datasets and dataset RIDs
5. Date range of analysis
6. Known data quality issues
7. Point of contact for questions

**NOTE**

Foundry dataset RIDs are the stable, version-controlled identifiers for datasets. Always store RIDs in your notebook header, not just friendly names. Friendly names can change; RIDs do not.

**CAUTION**

Never write credentials, tokens, or connection strings into notebook cells. Code Workspaces use workspace-level token injection. If you find yourself typing a password or API key, stop — you are doing it wrong. Contact your MSS Administrator.

## 2-3. Task: Manage Packages in the ORSA Workspace

**CONDITIONS:** You need a package not available in the default workspace environment.

**STANDARDS:** Package is installed, documented in the workspace requirements file, and does not conflict with existing environment packages.

**EQUIPMENT:** MSS access (browser-based), Code Workspace with active environment, MSS Administrator contact for approved package list.

**PROCEDURE:**

1. Check whether the package is available in the approved package repository. Not all PyPI/CRAN packages are available in the Foundry-managed environment. Contact your MSS Administrator for the approved list.
2. If available, add the package to the workspace's `requirements.txt` (Python) or `renv.lock` / `DESCRIPTION` file (R) rather than installing inline with `pip install` or `install.packages()`. Inline installation does not persist across workspace restarts.
3. For Python, the `requirements.txt` approach:

```
# requirements.txt — ORSA workspace
numpy>=1.24
pandas>=2.0
scipy>=1.10
statsmodels>=0.14
scikit-learn>=1.3
matplotlib>=3.7
seaborn>=0.12
lpSolve>=5.6  # if using R binding via rpy2
```

1. Rebuild the workspace environment after updating requirements. In most Foundry deployments, this is done via the workspace settings panel.
2. Document any non-standard packages in your notebook README with justification for their use.

> **NOTE**
>
> Package versions matter for reproducibility. Pin major and minor versions in requirements.txt. A forecast model built on statsmodels 0.13 may behave differently on 0.14. The version you used when producing an operational product must be recoverable.

## 2-4. Task: Write Output Datasets Back to Foundry

**CONDITIONS:** Your analysis produces results that need to be stored in Foundry for use by Workshop applications, Contour dashboards, or other consumers.

**STANDARDS:** Output dataset is written to the correct project location, schema is documented, and output can be read by downstream consumers.

**EQUIPMENT:** Code Workspace with active Foundry client session, write permissions to the target output dataset location.

**PROCEDURE:**

1. Prepare the output DataFrame with clean, documented column names. Use lowercase with underscores. Never use spaces in column names for Foundry output datasets.
2. Write the output in Python:

```
# Write results DataFrame back to Foundry
output_df = pd.DataFrame({
    "unit_uic": forecast_df["uic"],
    "forecast_month": forecast_df["month"],
    "predicted_c_rating": forecast_df["prediction"],
    "lower_bound_90": forecast_df["lower_90"],
    "upper_bound_90": forecast_df["upper_90"],
    "model_version": "ARIMA_202603_v1",
    "generated_date": pd.Timestamp.now().isoformat()
})

# Write to Foundry dataset
client.datasets.write_dataset(
    dataset_rid="ri.foundry.main.dataset.OUTPUT-DATASET-RID",
    dataframe=output_df,
    mode="overwrite"  # or "append" for incremental products
)

print(f"Wrote {len(output_df)} rows to output dataset")
```

1. Verify the write by reading the dataset back and confirming row counts and schema match expectations.
2. Update the dataset description in Foundry with the analysis date, model version, and source notebook path.

---

# CHAPTER 3 — STATISTICAL MODELING

**BLUF:** Statistical models on MSS provide the quantitative foundation for readiness forecasting, risk classification, and logistics analysis. All models require documented assumptions, validation evidence, and uncertainty reporting before delivery to commanders.

## 3-1. Regression for Readiness Forecasting

Regression models predict a continuous outcome (e.g., readiness score, equipment availability percentage) from predictor variables (maintenance days, personnel fill, training days, parts availability). The primary application in USAREUR-AF is predicting near-term C-rating trajectories based on observable leading indicators.

### 3-1a. Task: Build a Linear Regression Readiness Model

**CONDITIONS:** You have a Foundry dataset containing historical unit readiness records (minimum 24 months), including C-rating numerical equivalents and candidate predictor variables.

**STANDARDS:** Model achieves acceptable out-of-sample RMSE (defined by the analyst based on operational tolerance), assumptions are verified, and output includes prediction intervals.

**EQUIPMENT:** Code Workspace with Python or R, readiness history dataset.

**PROCEDURE:**

Python implementation:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, root_mean_squared_error
from sklearn.preprocessing import StandardScaler
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
import statsmodels.api as sm

# --- 1. Load and inspect data ---
df = readiness_df.copy()
df["report_date"] = pd.to_datetime(df["report_date"])
df = df.sort_values("report_date")

# --- 2. Feature engineering ---
# Lag features: last month's readiness as predictor
df["c_rating_lag1"] = df.groupby("uic")["c_rating_numeric"].shift(1)
df["c_rating_lag2"] = df.groupby("uic")["c_rating_numeric"].shift(2)
df["rolling_3mo_avg"] = (
    df.groupby("uic")["c_rating_numeric"]
    .transform(lambda x: x.rolling(3, min_periods=2).mean())
)

# Drop rows with NaN from lag creation
df_model = df.dropna(subset=[
    "c_rating_lag1", "c_rating_lag2", "rolling_3mo_avg",
    "personnel_fill_pct", "equipment_availability_pct",
    "maintenance_backlog_days", "c_rating_numeric"
]).copy()

# --- 3. Define features and target ---
feature_cols = [
    "c_rating_lag1",
    "c_rating_lag2",
    "rolling_3mo_avg",
    "personnel_fill_pct",
    "equipment_availability_pct",
    "maintenance_backlog_days"
]
```

```
X = df_model[feature_cols]
y = df_model["c_rating_numeric"]

# --- 4. Temporal train/test split (never random for time series) ---
# Use last 6 months as holdout — never use random split for temporal data
cutoff = df_model["report_date"].quantile(0.8)
X_train = X[df_model["report_date"] < cutoff]
X_test  = X[df_model["report_date"] >= cutoff]
y_train = y[df_model["report_date"] < cutoff]
y_test  = y[df_model["report_date"] >= cutoff]

print(f"Train: {len(X_train)} obs | Test: {len(X_test)} obs")

# --- 5. Scale features ---
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled  = scaler.transform(X_test)

# --- 6. Fit OLS via statsmodels for full inference output ---
X_train_sm = sm.add_constant(X_train_scaled)
model = sm.OLS(y_train, X_train_sm).fit()
print(model.summary())

# --- 7. Evaluate on holdout ---
X_test_sm = sm.add_constant(X_test_scaled)
y_pred = model.predict(X_test_sm)

mae  = mean_absolute_error(y_test, y_pred)
rmse = root_mean_squared_error(y_test, y_pred)
print(f"\nHoldout MAE:  {mae:.3f}")
print(f"Holdout RMSE: {rmse:.3f}")

# --- 8. Generate prediction intervals (approximate, OLS) ---
predictions = model.get_prediction(X_test_sm)
pred_summary = predictions.summary_frame(alpha=0.10)  # 90% CI
# pred_summary contains: mean, mean_se, mean_ci_lower, mean_ci_upper,
#                        obs_ci_lower (prediction interval), obs_ci_upper

# --- 9. Save residual diagnostic plot ---
residuals = y_train - model.fittedvalues
fig, axes = plt.subplots(1, 2, figsize=(12, 4))
axes[0].scatter(model.fittedvalues, residuals, alpha=0.4)
axes[0].axhline(0, color="red", linestyle="--")
axes[0].set_xlabel("Fitted Values")
axes[0].set_ylabel("Residuals")
axes[0].set_title("Residuals vs. Fitted")
sm.qqplot(residuals, line="45", ax=axes[1], alpha=0.4)
axes[1].set_title("Q-Q Plot of Residuals")
plt.tight_layout()
plt.savefig("readiness_model_diagnostics.png", dpi=150)
plt.close()
print("Diagnostic plot saved.")
```

R implementation:

```
library(tidyverse)
library(caret)
library(Metrics)

# --- 1. Prepare data ---
df <- readiness_df |>
  mutate(report_date = as.Date(report_date)) |>
  arrange(uic, report_date) |>
  group_by(uic) |>
  mutate(
    c_rating_lag1    = lag(c_rating_numeric, 1),
    c_rating_lag2    = lag(c_rating_numeric, 2),
    rolling_3mo_avg  = zoo::rollmean(c_rating_numeric, 3, fill = NA, align = "right")
  ) |>
  ungroup() |>
  drop_na(c_rating_lag1, c_rating_lag2, rolling_3mo_avg,
          personnel_fill_pct, equipment_availability_pct,
          maintenance_backlog_days, c_rating_numeric)

# --- 2. Temporal split ---
cutoff <- quantile(df$report_date, 0.8)
train <- df |> filter(report_date < cutoff)
test  <- df |> filter(report_date >= cutoff)

# --- 3. Fit model ---
model <- lm(
  c_rating_numeric ~ c_rating_lag1 + c_rating_lag2 + rolling_3mo_avg +
    personnel_fill_pct + equipment_availability_pct + maintenance_backlog_days,
  data = train
)
summary(model)

# --- 4. Evaluate ---
preds <- predict(model, newdata = test, interval = "prediction", level = 0.90)
test_results <- test |>
  mutate(
    predicted      = preds[, "fit"],
    lower_90       = preds[, "lwr"],
    upper_90       = preds[, "upr"]
  )

cat(sprintf("Holdout MAE:  %.3f\n", mae(test_results$c_rating_numeric,
test_results$predicted)))
cat(sprintf("Holdout RMSE: %.3f\n", rmse(test_results$c_rating_numeric,
test_results$predicted)))
```

**CAUTION: Never use random train/test splits for temporal data. A random split allows the model to train on future data and test on past data, producing optimistically biased performance estimates. Always split chronologically: train on earlier periods, test on later periods.**

## 3-2. Classification for Risk Analysis

Classification models predict a categorical outcome — most commonly: will a unit drop below C-3 readiness in the next 30/60/90 days? Classification enables targeted commander action before a readiness failure occurs.

### 3-2a. Task: Build a Readiness Drop Risk Classifier

**CONDITIONS:** Readiness history dataset with sufficient instances of readiness degradation events to train a classifier (minimum 50 positive cases recommended). Binary outcome: `risk_flag` = 1 if unit dropped below threshold within N days, 0 otherwise.

**STANDARDS:** Model F1 score and precision/recall on holdout set are documented. Class imbalance is addressed. Confusion matrix is included in product.

**EQUIPMENT:** Code Workspace with Python, readiness history dataset with labeled degradation events.

**PROCEDURE:**

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
from sklearn.utils.class_weight import compute_class_weight
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
import numpy as np

# --- 1. Construct binary risk label ---
# risk_flag = 1 if unit C-rating dropped below C3 within next 60 days
df_model["risk_flag"] = (
    df_model.groupby("uic")["c_rating_numeric"]
    .transform(lambda x: x.shift(-2) < 3)  # -2 = 60-day lookahead at monthly data
    .astype(int)
)

df_clf = df_model.dropna(subset=["risk_flag"]).copy()

feature_cols = [
    "c_rating_lag1", "c_rating_lag2", "rolling_3mo_avg",
    "personnel_fill_pct", "equipment_availability_pct",
    "maintenance_backlog_days"
]
X_clf = df_clf[feature_cols]
y_clf = df_clf["risk_flag"]

print(f"Class balance: {y_clf.value_counts().to_dict()}")

# --- 2. Temporal split ---
cutoff = df_clf["report_date"].quantile(0.8)
X_tr = X_clf[df_clf["report_date"] < cutoff]
X_te = X_clf[df_clf["report_date"] >= cutoff]
y_tr = y_clf[df_clf["report_date"] < cutoff]
y_te = y_clf[df_clf["report_date"] >= cutoff]
```

```
# --- 3. Handle class imbalance ---
classes = np.unique(y_tr)
weights = compute_class_weight("balanced", classes=classes, y=y_tr)
class_weight_dict = dict(zip(classes, weights))

# --- 4. Train classifier ---
clf = RandomForestClassifier(
    n_estimators=200,
    max_depth=6,         # limit depth to reduce overfitting on small datasets
    class_weight=class_weight_dict,
    random_state=42,
    n_jobs=-1
)
clf.fit(X_tr, y_tr)

# --- 5. Evaluate ---
y_pred = clf.predict(X_te)
y_prob = clf.predict_proba(X_te)[:, 1]

print("\nClassification Report:")
print(classification_report(y_te, y_pred, target_names=["No Risk", "Risk"]))
print(f"ROC-AUC: {roc_auc_score(y_te, y_prob):.3f}")
print("\nConfusion Matrix:")
print(confusion_matrix(y_te, y_pred))

# --- 6. Feature importance plot ---
importances = pd.Series(clf.feature_importances_, index=feature_cols).sort_values()
fig, ax = plt.subplots(figsize=(8, 4))
importances.plot(kind="barh", ax=ax)
ax.set_title("Feature Importances — Readiness Risk Classifier")
ax.set_xlabel("Importance")
plt.tight_layout()
plt.savefig("risk_classifier_importance.png", dpi=150)
plt.close()
```

**NOTE**

Class imbalance is common in readiness risk classification — readiness failures are relatively rare. A model that predicts "no risk" for every unit will achieve 90%+ accuracy on imbalanced data while being useless. Always report precision, recall, and F1 for the minority (risk) class, not just overall accuracy. Use `class_weight="balanced"` or oversample the minority class to address imbalance.

## 3-3. Model Validation Standards

All ORSA models delivered as operational products must meet the validation standards in Table 3-1.

**Table 3-1. Model Validation Requirements by Model Type**

| Model Type | Required Metrics | Minimum Acceptable | Additional Requirement |
|---|---|---|---|
| Linear Regression | RMSE, MAE, $R^2$, residual normality | Domain-specific (document threshold) | Residual plots, temporal split |
| Logistic Regression | AUC-ROC, F1 (risk class), Precision, Recall | AUC > 0.70 | Calibration plot, confusion matrix |
| Random Forest Classifier | Same as logistic | AUC > 0.70 | Feature importance, depth limit |
| Time Series Forecast | MAE, RMSE, MAPE, coverage of PI | MAPE < 15% for 30-day | Backtesting, prediction intervals |
| Simulation | Distribution of outputs, sensitivity to key params | No point-estimate-only output | Convergence check |

# CHAPTER 4 — TIME SERIES ANALYSIS

**BLUF:** Time series methods are the primary tool for readiness trend analysis and logistics demand forecasting. ARIMA and SARIMA models handle the seasonal patterns inherent in Army readiness cycles (EXEVAL windows, deployment cycles, training schedules).

## 4-1. Readiness Trend Analysis

### 4-1a. Task: Analyze Unit Readiness Trend

**CONDITIONS:** Monthly readiness data for a unit or formation, minimum 24 months. Analytical question: is readiness trending up, down, or stable? Are there seasonal patterns?

**STANDARDS:** Stationarity tested, trend characterized with confidence bounds, seasonal decomposition performed if applicable, and plot suitable for commander brief produced.

**EQUIPMENT:** Code Workspace with Python or R, readiness history dataset with at least 24 months of monthly C-rating records.

**PROCEDURE:**

```
import pandas as pd
import numpy as np
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller
```

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import warnings
warnings.filterwarnings("ignore")

# --- 1. Prepare time series ---
# Filter to single unit for trend analysis
unit_uic = "W12345"
ts_df = (
    readiness_df[readiness_df["uic"] == unit_uic]
    .sort_values("report_date")
    .set_index("report_date")
    [["c_rating_numeric"]]
)
ts_df.index = pd.DatetimeIndex(ts_df.index, freq="MS")  # monthly start

ts = ts_df["c_rating_numeric"]
print(f"Series length: {len(ts)} months")
print(f"Date range: {ts.index.min()} to {ts.index.max()}")

# --- 2. Test stationarity (Augmented Dickey-Fuller) ---
adf_result = adfuller(ts.dropna(), autolag="AIC")
print(f"\nADF Statistic: {adf_result[0]:.4f}")
print(f"p-value: {adf_result[1]:.4f}")
print("Stationarity: REJECT non-stationarity (series is stationary)"
      if adf_result[1] < 0.05
      else "FAIL TO REJECT non-stationarity (series may have unit root — consider
differencing)")

# --- 3. Seasonal decomposition ---
if len(ts) >= 24:  # need 2+ full cycles for decomposition
    decomp = seasonal_decompose(ts, model="additive", period=12)
    fig, axes = plt.subplots(4, 1, figsize=(12, 10))
    decomp.observed.plot(ax=axes[0], title="Observed")
    decomp.trend.plot(ax=axes[1], title="Trend")
    decomp.seasonal.plot(ax=axes[2], title="Seasonal")
    decomp.resid.plot(ax=axes[3], title="Residual")
    plt.suptitle(f"Readiness Decomposition — UIC {unit_uic}", y=1.01)
    plt.tight_layout()
    plt.savefig(f"readiness_decomp_{unit_uic}.png", dpi=150)
    plt.close()
    print("Decomposition plot saved.")

# --- 4. ACF/PACF plots for model order identification ---
fig, axes = plt.subplots(1, 2, figsize=(12, 4))
plot_acf(ts.dropna(), lags=20, ax=axes[0], title="ACF")
plot_pacf(ts.dropna(), lags=20, ax=axes[1], title="PACF")
plt.tight_layout()
plt.savefig(f"readiness_acf_pacf_{unit_uic}.png", dpi=150)
plt.close()
```

## 4-2. Task: Build an ARIMA Readiness Forecast

**CONDITIONS:** Stationarity analysis complete. Model order (p, d, q) identified from ACF/PACF or auto-selection. Forecast horizon defined (30/60/90 days or 1/2/3 months ahead).

**STANDARDS:** Model selected by AIC, backtested on holdout, forecast includes 80% and 90% prediction intervals, and commander-ready plot produced.

**EQUIPMENT:** Code Workspace with Python or R, readiness time series dataset (minimum 24 monthly observations per unit), stationarity analysis output from Task 4-1a.

**PROCEDURE:**

Python — auto-order selection and forecast:

```
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
import itertools

# --- 1. Auto-select ARIMA order by AIC (grid search) ---
p_range = range(0, 4)
d_range = range(0, 2)
q_range = range(0, 4)

best_aic = np.inf
best_order = None

for p, d, q in itertools.product(p_range, d_range, q_range):
    try:
        mod = ARIMA(ts, order=(p, d, q))
        res = mod.fit()
        if res.aic < best_aic:
            best_aic = res.aic
            best_order = (p, d, q)
    except Exception:
        continue

print(f"Best ARIMA order: {best_order}  |  AIC: {best_aic:.2f}")

# --- 2. Fit final model ---
model = ARIMA(ts, order=best_order)
fitted = model.fit()
print(fitted.summary())

# --- 3. Backtest: re-fit on 80% of data, evaluate on 20% ---
n = len(ts)
train_end = int(n * 0.8)
ts_train = ts.iloc[:train_end]
ts_test  = ts.iloc[train_end:]

backtest_model = ARIMA(ts_train, order=best_order).fit()
forecast_steps = len(ts_test)
backtest_forecast = backtest_model.get_forecast(steps=forecast_steps)
backtest_mean = backtest_forecast.predicted_mean
```

```
backtest_ci   = backtest_forecast.conf_int(alpha=0.10)  # 90% CI

bt_mae  = mean_absolute_error(ts_test, backtest_mean)
bt_mape = np.mean(np.abs((ts_test - backtest_mean) / ts_test)) * 100
print(f"\nBacktest MAE:  {bt_mae:.3f}")
print(f"Backtest MAPE: {bt_mape:.1f}%")

# --- 4. Generate forward forecast ---
forecast_horizon = 3  # months
fcast = fitted.get_forecast(steps=forecast_horizon)
fcast_mean = fcast.predicted_mean
fcast_ci80 = fcast.conf_int(alpha=0.20)  # 80% PI
fcast_ci90 = fcast.conf_int(alpha=0.10)  # 90% PI

print("\nForward Forecast:")
for i, (idx, val) in enumerate(fcast_mean.items()):
    lo90 = fcast_ci90.iloc[i, 0]
    hi90 = fcast_ci90.iloc[i, 1]
    print(f"  {idx.strftime('%Y-%m')}: {val:.2f}  [90% PI: {lo90:.2f} – {hi90:.2f}]")

# --- 5. Commander-ready plot ---
fig, ax = plt.subplots(figsize=(12, 5))
ts.plot(ax=ax, label="Historical", color="navy", linewidth=1.5)
fcast_mean.plot(ax=ax, label="Forecast", color="darkorange", linewidth=2, linestyle="-
-")
ax.fill_between(
    fcast_ci80.index,
    fcast_ci80.iloc[:, 0], fcast_ci80.iloc[:, 1],
    alpha=0.4, color="orange", label="80% Prediction Interval"
)
ax.fill_between(
    fcast_ci90.index,
    fcast_ci90.iloc[:, 0], fcast_ci90.iloc[:, 1],
    alpha=0.2, color="orange", label="90% Prediction Interval"
)
ax.axhline(y=3, color="red", linestyle=":", linewidth=1, label="C-3 Threshold")
ax.set_xlabel("Date")
ax.set_ylabel("C-Rating (Numeric)")
ax.set_title(f"Readiness Forecast — UIC {unit_uic}  |  ARIMA{best_order}")
ax.legend(loc="lower left")
plt.tight_layout()
plt.savefig(f"readiness_forecast_{unit_uic}.png", dpi=150)
plt.close()
print("Forecast plot saved.")
```

R implementation using the `forecast` package:

```
library(forecast)
library(tseries)
library(ggplot2)

# --- 1. Build ts object ---
ts_data <- ts(
  readiness_unit$c_rating_numeric,
```

```
  start = c(2024, 1),  # year, month
  frequency = 12
)

# --- 2. Stationarity test ---
adf_result <- adf.test(na.omit(ts_data))
cat(sprintf("ADF p-value: %.4f\n", adf_result$p.value))

# --- 3. Auto-fit ARIMA (auto.arima handles order selection) ---
model <- auto.arima(
  ts_data,
  seasonal = TRUE,     # consider SARIMA
  stepwise = FALSE,    # full search (slower but more thorough)
  approximation = FALSE,
  ic = "aic"
)
summary(model)

# --- 4. Backtest with time series cross-validation ---
errors <- tsCV(ts_data, forecastfunction = function(x, h) forecast(auto.arima(x), h =
h), h = 3)
cat(sprintf("CV RMSE (3-mo horizon): %.3f\n", sqrt(mean(errors^2, na.rm = TRUE))))

# --- 5. Forward forecast ---
fc <- forecast(model, h = 3, level = c(80, 90))
print(fc)

# --- 6. Commander-ready plot ---
autoplot(fc) +
  geom_hline(yintercept = 3, linetype = "dotted", color = "red") +
  labs(
    title = paste("Readiness Forecast —", unit_uic),
    subtitle = sprintf("ARIMA%s — Backtest RMSE: %.3f", arimaorder(model),
sqrt(mean(errors^2, na.rm = TRUE))),
    x = "Date", y = "C-Rating (Numeric)"
  ) +
  theme_bw()
ggsave(paste0("readiness_forecast_", unit_uic, ".png"), width = 10, height = 5, dpi =
150)
```

### 4-3. Task: Logistics Demand Forecasting (Class IX)

**CONDITIONS:** Monthly Class IX demand data (parts requisitions, consumption by NSN or commodity group) for a formation, minimum 18 months. Analytical question: forecast demand to support pre-positioning and stock level decisions.

**STANDARDS:** Seasonal patterns identified, SARIMA model fitted (Army maintenance cycles are seasonal), forecast includes uncertainty bounds, output feeds logistics planning product.

**EQUIPMENT:** Code Workspace with Python, logistics demand dataset (minimum 18 months of monthly Class IX consumption records).

**PROCEDURE:**

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
import itertools

# --- 1. Prepare demand series ---
demand_ts = (
    logistics_df
    .groupby("report_month")["parts_demand_qty"]
    .sum()
    .sort_index()
)
demand_ts.index = pd.DatetimeIndex(demand_ts.index, freq="MS")

# --- 2. Identify seasonal period ---
# Army Class IX demand peaks pre-EXEVAL, post-deployment return, and pre-winter
# Assume annual seasonality (period=12) unless data suggests otherwise
# Inspect ACF at lag 12 to confirm

# --- 3. Grid search for SARIMA(p,d,q)(P,D,Q,12) ---
p_d_q   = list(itertools.product(range(3), range(2), range(3)))
P_D_Q   = list(itertools.product(range(2), range(2), range(2)))
best_aic = np.inf
best_params = None

for (p, d, q), (P, D, Q) in itertools.product(p_d_q, P_D_Q):
    try:
        mod = SARIMAX(
            demand_ts,
            order=(p, d, q),
            seasonal_order=(P, D, Q, 12),
            enforce_stationarity=False,
            enforce_invertibility=False
        )
        res = mod.fit(disp=False)
        if res.aic < best_aic:
            best_aic = res.aic
            best_params = ((p, d, q), (P, D, Q, 12))
    except Exception:
        continue

print(f"Best SARIMA: {best_params[0]} x {best_params[1]}  |  AIC: {best_aic:.2f}")

# --- 4. Fit and forecast ---
final_model = SARIMAX(
    demand_ts,
    order=best_params[0],
    seasonal_order=best_params[1],
    enforce_stationarity=False,
    enforce_invertibility=False
).fit(disp=False)

forecast_horizon = 6
forecast = final_model.get_forecast(steps=forecast_horizon)
forecast_mean = forecast.predicted_mean
```

```
forecast_ci   = forecast.conf_int(alpha=0.10)

# Round to whole units for logistics planning
forecast_output = pd.DataFrame({
    "forecast_month":  forecast_mean.index,
    "forecast_demand": forecast_mean.values.round(0).astype(int),
    "lower_90":        forecast_ci.iloc[:, 0].clip(lower=0).round(0).astype(int),
    "upper_90":        forecast_ci.iloc[:, 1].round(0).astype(int)
})
print(forecast_output.to_string(index=False))

# --- 5. Plot ---
fig, ax = plt.subplots(figsize=(12, 5))
demand_ts.plot(ax=ax, label="Historical Demand", color="navy")
forecast_mean.plot(ax=ax, color="darkorange", linestyle="--", label="Forecast")
ax.fill_between(
    forecast_ci.index,
    forecast_ci.iloc[:, 0].clip(lower=0),
    forecast_ci.iloc[:, 1],
    alpha=0.3, color="orange", label="90% Prediction Interval"
)
ax.set_xlabel("Month")
ax.set_ylabel("Parts Demand (Qty)")
ax.set_title("Class IX Demand Forecast — 6-Month Horizon")
ax.legend()
plt.tight_layout()
plt.savefig("class_ix_demand_forecast.png", dpi=150)
plt.close()
```

**NOTE**

Logistics demand forecasting should incorporate known operational events as exogenous variables when available. A planned EXEVAL or DEFENDER exercise will spike Class IX demand. Use SARIMAX (the X = exogenous) to include planned event indicators as predictors alongside the time series component.

## CHAPTER 5 — MONTE CARLO SIMULATION AND RISK ANALYSIS

**BLUF:** Monte Carlo simulation quantifies decision risk by modeling uncertainty across all input variables simultaneously. For ORSA, the primary applications are COA comparison under uncertainty and logistics planning risk. Monte Carlo produces distributions of outcomes, not point estimates — this is its essential value for commander decision support.

## 5-1. COA Comparison Under Uncertainty

When comparing courses of action, every input parameter carries uncertainty. Monte Carlo simulation propagates that uncertainty through the model and produces a distribution of outcomes for each COA, enabling commanders to see not just expected values but risk profiles.

### 5-1a. Task: Conduct Monte Carlo COA Risk Analysis

**CONDITIONS:** Two or more COAs defined with associated input parameters and uncertainty ranges. Utility function or outcome metric defined (e.g., days to operational capability, sustainment cost, personnel risk).

**STANDARDS:** Minimum 10,000 simulation runs per COA, input distributions documented and calibrated to historical data or SME bounds, output includes probability of exceedance curves and summary statistics, sensitivity analysis performed.

**EQUIPMENT:** Code Workspace with Python, COA parameter definitions, historical data or SME-derived distribution bounds for each uncertain input.

**PROCEDURE:**

```
import numpy as np
import pandas as pd
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from scipy import stats

np.random.seed(42)  # reproducibility — document seed in product
N_SIMULATIONS = 50_000  # 50K runs — sufficient for stable percentile estimates

# ============================================================================
# SCENARIO: COA A vs COA B — Time to Full Operational Capability (days)
# Both COAs achieve the same objective via different approach routes and
# sustainment configurations. Key uncertain inputs:
#   - Transit time (affected by route conditions, traffic, weather)
#   - Equipment prep time (affected by maintenance readiness state)
#   - Personnel marshaling time (affected by notification time, leave status)
#   - Sustainment class III(B) delivery (affected by HEMTT availability)
# ============================================================================

# --- COA A input distributions ---
# (distribution type, parameters) — calibrated to historical exercise data
coa_a = {
    "transit_days":       stats.norm(loc=3.2, scale=0.8),       # ~3.2 days avg
    "equip_prep_days":    stats.gamma(a=3.0, scale=0.9),        # right-skewed, min 0
    "personnel_days":     stats.uniform(loc=0.5, scale=1.5),    # 0.5–2.0 days
    "sustainment_days":   stats.norm(loc=1.5, scale=0.5)        # ~1.5 days avg
}

# --- COA B input distributions ---
coa_b = {
```

```
    "transit_days":       stats.norm(loc=2.1, scale=1.4),       # shorter avg, higher
variance
    "equip_prep_days":    stats.gamma(a=4.0, scale=1.0),        # more equipment-
intensive
    "personnel_days":     stats.uniform(loc=0.5, scale=1.0),
    "sustainment_days":   stats.norm(loc=2.8, scale=0.8)        # longer sustainment
pipeline
}

def simulate_coa(inputs: dict, n: int) -> np.ndarray:
    """Sample all input distributions and sum to get total FOC time."""
    total = np.zeros(n)
    for component, distribution in inputs.items():
        samples = distribution.rvs(size=n)
        samples = np.clip(samples, 0, None)  # no negative times
        total += samples
    return total

results_a = simulate_coa(coa_a, N_SIMULATIONS)
results_b = simulate_coa(coa_b, N_SIMULATIONS)

# --- Summary statistics ---
def summarize(name: str, results: np.ndarray) -> None:
    print(f"\n{name}:")
    print(f"  Mean:          {results.mean():.2f} days")
    print(f"  Std Dev:       {results.std():.2f} days")
    print(f"  P10 / P50 / P90: {np.percentile(results, 10):.2f} / "
          f"{np.percentile(results, 50):.2f} / {np.percentile(results, 90):.2f}")
    print(f"  P(FOC <= 7 days): {(results <= 7).mean()*100:.1f}%")
    print(f"  P(FOC <= 10 days): {(results <= 10).mean()*100:.1f}%")

summarize("COA A", results_a)
summarize("COA B", results_b)

# --- Probability of exceedance ---
thresholds = np.linspace(0, 20, 200)
poe_a = [(results_a > t).mean() for t in thresholds]
poe_b = [(results_b > t).mean() for t in thresholds]

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Distribution comparison
axes[0].hist(results_a, bins=80, density=True, alpha=0.6, label="COA A",
color="steelblue")
axes[0].hist(results_b, bins=80, density=True, alpha=0.6, label="COA B",
color="darkorange")
axes[0].axvline(np.mean(results_a), color="steelblue", linestyle="--", linewidth=1.5,
label=f"COA A mean: {np.mean(results_a):.1f}d")
axes[0].axvline(np.mean(results_b), color="darkorange", linestyle="--", linewidth=1.5,
label=f"COA B mean: {np.mean(results_b):.1f}d")
axes[0].set_xlabel("Days to FOC")
axes[0].set_ylabel("Probability Density")
axes[0].set_title("COA Outcome Distributions (N=50,000)")
axes[0].legend()
```

```
# Probability of exceedance curves
axes[1].plot(thresholds, poe_a, color="steelblue", linewidth=2, label="COA A")
axes[1].plot(thresholds, poe_b, color="darkorange", linewidth=2, label="COA B")
axes[1].axvline(7, color="red", linestyle=":", label="7-day planning threshold")
axes[1].set_xlabel("Days to FOC")
axes[1].set_ylabel("P(FOC exceeds threshold)")
axes[1].set_title("Probability of Exceedance Curves")
axes[1].legend()
axes[1].grid(True, alpha=0.3)

plt.suptitle("Monte Carlo COA Comparison — FOC Timeline Risk", fontsize=13)
plt.tight_layout()
plt.savefig("coa_monte_carlo.png", dpi=150)
plt.close()
print("\nCOA comparison plot saved.")
```

## 5-2. Sensitivity Analysis

Sensitivity analysis identifies which input uncertainties drive the most outcome variance — telling commanders where reducing uncertainty has the highest value.

### 5-2a. Task: Conduct Tornado Sensitivity Analysis

**CONDITIONS:** Monte Carlo model complete. Analyst needs to identify the highest-impact uncertain inputs for a given COA.

**STANDARDS:** Tornado chart produced showing rank-ordered input contributions to output variance. Correlation coefficients between each input and output documented.

**EQUIPMENT:** Code Workspace with Python, Monte Carlo simulation results and stored component samples from Task 5-1a.

**PROCEDURE:**

```
# Tornado chart via Spearman rank correlation
# (robust to non-normal input distributions)
from scipy.stats import spearmanr

# Sample each input independently and compute correlation with output
component_samples = {}
for component, distribution in coa_a.items():
    component_samples[component] = distribution.rvs(size=N_SIMULATIONS).clip(0, None)

# Recompute output with stored samples
output = sum(component_samples.values())

correlations = {}
for component, samples in component_samples.items():
    r, p = spearmanr(samples, output)
    correlations[component] = r

# Sort by absolute correlation
```

```
sorted_corr = dict(sorted(correlations.items(), key=lambda x: abs(x[1])))

# Tornado chart
fig, ax = plt.subplots(figsize=(8, 5))
components = list(sorted_corr.keys())
values     = list(sorted_corr.values())
colors     = ["steelblue" if v > 0 else "firebrick" for v in values]
ax.barh(components, values, color=colors, alpha=0.8)
ax.axvline(0, color="black", linewidth=0.8)
ax.set_xlabel("Spearman Rank Correlation with Total FOC Time")
ax.set_title("Sensitivity Analysis — COA A (Tornado Chart)")
ax.set_xlim(-1, 1)
for i, v in enumerate(values):
    ax.text(v + 0.02 if v >= 0 else v - 0.02, i,
            f"{v:.2f}", va="center", ha="left" if v >= 0 else "right", fontsize=9)
plt.tight_layout()
plt.savefig("coa_a_tornado.png", dpi=150)
plt.close()
print("Tornado chart saved.")
```

### 5-3. Logistics Planning Risk — Supply Stockage Level Analysis

**CONDITIONS:** Historical Class IX demand data available. Commander requires a stockage recommendation that provides acceptable risk of stockout over a given resupply interval.

**STANDARDS:** Stockage level recommendation accompanied by explicit probability of stockout at recommended level. Safety stock calculation documented.

**EQUIPMENT:** Code Workspace with Python, historical daily Class IX demand records, commander-specified service level target.

**PROCEDURE:**

```
# Model daily demand as a distribution, simulate over resupply interval
# Stockage recommendation at specified service level (e.g., 90% probability of no
stockout)

# Input parameters (calibrate from historical data)
mean_daily_demand  = 12.4   # units/day — from historical data
std_daily_demand   = 3.8    # units/day
resupply_interval  = 7      # days between resupply
lead_time_days     = 2      # order-to-receipt lead time
service_level      = 0.90   # 90% — acceptable stockout risk = 10%

N_SIM = 100_000

# Simulate demand over (resupply interval + lead time)
total_demand_period = resupply_interval + lead_time_days
daily_demands = np.random.normal(
    loc=mean_daily_demand,
    scale=std_daily_demand,
    size=(N_SIM, total_demand_period)
)
```

```
daily_demands = np.clip(daily_demands, 0, None)
total_demands = daily_demands.sum(axis=1)

# Recommended stockage level at desired service level
recommended_level = np.percentile(total_demands, service_level * 100)

print(f"\nClass IX Stockage Level Analysis")
print(f"  Mean demand over {total_demand_period}-day window:
{total_demands.mean():.1f} units")
print(f"  Std Dev: {total_demands.std():.1f} units")
print(f"  Recommended stockage at {service_level*100:.0f}% service level:
{recommended_level:.0f} units")
print(f"  Safety stock (above mean): {recommended_level - total_demands.mean():.0f}
units")
print(f"  Probability of stockout at recommended level: {(total_demands >
recommended_level).mean()*100:.1f}%")

# Stockout risk curve
levels = np.arange(int(total_demands.mean() * 0.7), int(total_demands.max() * 1.1))
stockout_prob = [(total_demands > lvl).mean() * 100 for lvl in levels]

fig, ax = plt.subplots(figsize=(10, 4))
ax.plot(levels, stockout_prob, color="navy", linewidth=2)
ax.axvline(recommended_level, color="green", linestyle="--",
           label=f"Recommended ({service_level*100:.0f}% SL): {recommended_level:.0f}
units")
ax.axhline(10, color="red", linestyle=":", label="10% stockout risk threshold")
ax.set_xlabel("Stockage Level (Units)")
ax.set_ylabel("Probability of Stockout (%)")
ax.set_title("Class IX Stockout Risk Curve")
ax.legend()
plt.tight_layout()
plt.savefig("class_ix_stockout_risk.png", dpi=150)
plt.close()
```

**NOTE**

The service level in stockage analysis is a commander's decision, not an analyst's assumption. Present the risk curve (Chapter 9) and let the commander select the acceptable risk level. The analyst's role is to quantify the tradeoff between stockage cost and stockout risk — not to make the risk tolerance decision.

## 5-4. CARVER Target Value Analysis

**BLUF:** CARVER is a structured scoring model for prioritizing targets, assets, or systems by six doctrinal factors. FM 3-60, Appendix G defines the methodology. CARVER produces a weighted numerical ranking directly implementable as a Foundry dataset and scoring pipeline on MSS.

**Table 5-4. CARVER Factor Definitions (FM 3-60, Appendix G)**

| Factor | Question | Score Range |
|---|---|---|
| **C** — Criticality | How essential is this target to the adversary's capability? | 1–5 |
| **A** — Accessibility | Can the force reach/affect the target given terrain, defenses, and standoff? | 1–5 |
| **R** — Recuperability | How quickly can the adversary restore the target's function after attack? | 1–5 |
| **V** — Vulnerability | Is the target susceptible to the available means of attack? | 1–5 |
| **E** — Effect | What collateral or cascading effects result from engaging the target? | 1–5 |
| **R** — Recognizability | Can the force positively identify the target under expected conditions? | 1–5 |

**Scoring procedure:**

1. Define the target set as rows in a Foundry dataset. Each row represents one potential target, node, or system.
2. Score each target on all six factors (1 = lowest, 5 = highest priority contribution). Use SME panels or structured elicitation — do not allow a single analyst to score unilaterally.
3. Apply weights if the commander prioritizes certain factors (e.g., doubling the weight on Criticality for a time-sensitive targeting cycle). Document weight rationale.
4. Compute CARVER total: `CARVER_score = wC*C + wA*A + wR*R + wV*V + wE*E + wR2*R2`. Rank targets by descending score.
5. Present the ranked list with factor breakdowns so the commander can see why each target ranks where it does — the decomposition is as valuable as the total score.

> **NOTE**
>
> CARVER is a prioritization tool, not a targeting decision. The ranked output informs the targeting working group; it does not replace commander authority or legal review. See SL 4B (Fires) for the targeting methodology that consumes CARVER products.

Implement CARVER as a reusable Foundry object type with six numeric properties (C, A, R, V, E, R2), a weights configuration, and a computed total. This allows the targeting working group to re-score rapidly as the operational picture changes.

# CHAPTER 6 — OPTIMIZATION

**BLUF:** Optimization methods find the best allocation of constrained resources across competing demands. Army applications include personnel assignment, equipment deployment, maintenance scheduling, and route planning. Linear programming is the foundational technique; present results with explicit constraint documentation so commanders understand what trade-offs the model enforces.

## 6-1. Resource Allocation with Linear Programming

### 6-1a. Task: Optimize Personnel Assignment Across Units

**CONDITIONS:** Multiple units with personnel requirements and fill priorities. Limited available personnel pool. Analyst must recommend optimal assignment to maximize aggregate readiness improvement subject to constraints.

**STANDARDS:** Objective function and all constraints documented in plain language before formulation. Solution validated against constraints manually (spot check). Sensitivity/shadow price analysis performed.

**EQUIPMENT:** Code Workspace with Python or R, unit readiness data, commander-approved priority weights and constraint values (minimum fills, maximum capacities, total available personnel).

**PROCEDURE:**

Python using scipy.optimize.linprog:

```
import numpy as np
from scipy.optimize import linprog
import pandas as pd

# ============================================================================
# SCENARIO: Assign 45 available personnel to 5 units to maximize aggregate
# readiness improvement. Each unit has different readiness gains per assigned
# personnel (marginal returns), minimum fill requirements, and maximum capacity.
# ============================================================================

# Units: A, B, C, D, E
n_units = 5
unit_names = ["1-1 CAV", "2-1 IN", "3-1 FA", "4-1 EN", "5-1 LOG"]

# Readiness improvement per additional person assigned (marginal value)
# Higher = more valuable to assign to that unit
readiness_gain_per_person = np.array([2.1, 1.8, 2.5, 1.2, 1.6])

# Constraints:
total_personnel = 45           # total available to assign
min_assignment  = np.array([5, 4, 3, 2, 3])   # unit minimum fills
max_assignment  = np.array([15, 12, 12, 10, 12])  # unit capacity limits
```

```
# --- Formulation ---
# Decision variables: x[i] = number of personnel assigned to unit i
# Objective: MAXIMIZE sum(readiness_gain_per_person[i] * x[i])
#            (linprog minimizes, so negate the objective)
c = -readiness_gain_per_person  # negate for minimization

# Inequality constraints (A_ub @ x <= b_ub):
# Total personnel: sum(x) <= total_personnel
A_ub = np.ones((1, n_units))
b_ub = np.array([total_personnel])

# Equality constraint: sum(x) == total_personnel (use all available personnel)
A_eq = np.ones((1, n_units))
b_eq = np.array([total_personnel])

# Bounds: min_assignment[i] <= x[i] <= max_assignment[i]
bounds = [(min_assignment[i], max_assignment[i]) for i in range(n_units)]

# --- Solve ---
result = linprog(
    c=c,
    A_ub=A_ub,
    b_ub=b_ub,
    bounds=bounds,
    method="highs"  # HiGHS solver — recommended in scipy >= 1.9
)

if result.success:
    assignments = np.round(result.x).astype(int)
    total_readiness_gain = -result.fun

    print("OPTIMAL PERSONNEL ASSIGNMENT")
    print("=" * 50)
    for i, (unit, assigned) in enumerate(zip(unit_names, assignments)):
        gain = readiness_gain_per_person[i] * assigned
        print(f"  {unit}: {assigned} personnel  |  Readiness gain: {gain:.1f}")
    print(f"\nTotal personnel assigned: {assignments.sum()}")
    print(f"Total readiness gain:     {total_readiness_gain:.1f}")

    # --- Sensitivity: what if we had 5 more personnel? ---
    b_eq_plus5 = np.array([total_personnel + 5])
    result_plus5 = linprog(c, A_ub=A_ub, b_ub=np.array([total_personnel + 5]),
                           bounds=bounds, method="highs")
    if result_plus5.success:
        marginal_value = (-result_plus5.fun - total_readiness_gain) / 5
        print(f"\nSensitivity: Each additional person beyond {total_personnel}")
        print(f"  adds ~{marginal_value:.2f} readiness points (shadow price)")
else:
    print(f"Optimization failed: {result.message}")
```

R implementation using lpSolve:

```
library(lpSolve)
library(tibble)

# Objective: maximize readiness gain per person * assignment quantity
# (lpSolve uses lp() which can maximize directly)

n_units <- 5
unit_names <- c("1-1 CAV", "2-1 IN", "3-1 FA", "4-1 EN", "5-1 LOG")
readiness_gain_per_person <- c(2.1, 1.8, 2.5, 1.2, 1.6)
total_personnel <- 45
min_assignment  <- c(5, 4, 3, 2, 3)
max_assignment  <- c(15, 12, 12, 10, 12)

# Constraint matrix:
# Row 1: sum(x) == total_personnel (equality)
f.con <- matrix(rep(1, n_units), nrow = 1)
f.dir <- "="
f.rhs <- total_personnel

# Bounds as additional inequality rows
upper_bounds <- diag(n_units)  # x[i] <= max_assignment[i]
lower_bounds <- diag(n_units)  # -x[i] <= -min_assignment[i]

f.con.full <- rbind(f.con, upper_bounds, -lower_bounds)
f.dir.full <- c("=", rep("<=", n_units), rep("<=", n_units))
f.rhs.full <- c(total_personnel, max_assignment, -min_assignment)

result <- lp(
  direction   = "max",
  objective.in = readiness_gain_per_person,
  const.mat   = f.con.full,
  const.dir   = f.dir.full,
  const.rhs   = f.rhs.full,
  all.int     = TRUE  # integer assignments
)

if (result$status == 0) {
  assignments <- result$solution
  cat("OPTIMAL PERSONNEL ASSIGNMENT\n")
  cat(rep("=", 50), "\n", sep = "")
  for (i in seq_along(unit_names)) {
    cat(sprintf("  %-12s %d personnel  |  Readiness gain: %.1f\n",
                unit_names[i], assignments[i],
                readiness_gain_per_person[i] * assignments[i]))
  }
  cat(sprintf("\nTotal readiness gain: %.1f\n", result$objval))
} else {
  cat("Optimization failed.\n")
}
```

## 6-2. Maintenance Scheduling Optimization

**CONDITIONS:** Maintenance backlog with multiple systems, limited mechanic-hours per day, prioritized by operational urgency. Optimize maintenance schedule to maximize operational availability within scheduling window.

**STANDARDS:** Schedule accounts for all capacity constraints, priority weights justified and documented, solution presented as a Gantt-style allocation table for direct use by maintenance officer.

**EQUIPMENT:** Code Workspace with Python, maintenance backlog dataset (system ID, hours required, deadline, priority), maintenance officer confirmation of daily mechanic-hour capacity and commander priority weights.

**PROCEDURE:**

```
import pandas as pd
import numpy as np
from scipy.optimize import linprog

# Maintenance schedule optimization (simplified)
# Decision: which systems to schedule in which maintenance windows (days)
# to maximize priority-weighted availability

# Systems requiring maintenance
systems = pd.DataFrame({
    "system_id":     ["A1", "A2", "A3", "B1", "B2", "C1"],
    "priority_wt":   [5, 3, 4, 2, 3, 5],       # commander priority 1-5
    "mechanic_hrs":  [8, 4, 12, 6, 3, 10],      # hours required to complete
    "deadline_day":  [3, 5, 7, 4, 7, 5],        # must complete by day N
})

n_systems = len(systems)
n_days    = 7
capacity_per_day = 16  # mechanic-hours available per day

print("Maintenance Scheduling Problem:")
print(systems.to_string(index=False))
print(f"\nCapacity: {capacity_per_day} mechanic-hours/day over {n_days} days")
print(f"Total demand: {systems['mechanic_hrs'].sum()} mechanic-hours")
print(f"Total capacity: {capacity_per_day * n_days} mechanic-hours")

# Binary decision variables: x[i,d] = 1 if system i scheduled on day d
# This is a binary integer program — use scipy milp or greedy heuristic
# For small problems, greedy priority-first scheduling is operationally readable

# Greedy approach: schedule by priority, earliest available day within deadline
schedule = []
daily_load = {d: 0 for d in range(1, n_days + 1)}

for _, row in systems.sort_values("priority_wt", ascending=False).iterrows():
    scheduled = False
    for day in range(1, int(row["deadline_day"]) + 1):
        if daily_load[day] + row["mechanic_hrs"] <= capacity_per_day:
```

```
            schedule.append({
                "system_id": row["system_id"],
                "scheduled_day": day,
                "mechanic_hrs": row["mechanic_hrs"],
                "priority_wt": row["priority_wt"]
            })
            daily_load[day] += row["mechanic_hrs"]
            scheduled = True
            break
    if not scheduled:
        schedule.append({
            "system_id": row["system_id"],
            "scheduled_day": None,
            "mechanic_hrs": row["mechanic_hrs"],
            "priority_wt": row["priority_wt"]
        })

schedule_df = pd.DataFrame(schedule)
print("\nProposed Maintenance Schedule:")
print(schedule_df.to_string(index=False))

unscheduled = schedule_df[schedule_df["scheduled_day"].isna()]
if len(unscheduled) > 0:
    print(f"\nWARNING: {len(unscheduled)} system(s) could not be scheduled within
deadline:")
    print(unscheduled[["system_id", "mechanic_hrs",
"priority_wt"]].to_string(index=False))
    print("  --> Commander action required: extend window, add capacity, or
reprioritize.")
```

**NOTE**

Optimization models are only as good as their objective function and constraint set. Before briefing a schedule to a maintenance officer, verify: (1) all constraints are captured (not just mechanic hours — also part availability, bay space, special tool requirements), (2) priority weights reflect current commander guidance, and (3) the model has not over-simplified a real-world scheduling problem. The model provides a starting point, not a binding schedule.

## CHAPTER 7 — WARGAME AND EXERCISE ANALYSIS

**BLUF:** Exercise analysis is one of the highest-value ORSA activities in USAREUR-AF. Rigorous wargame data collection and analysis enables commanders to measure training effectiveness, validate TTPs, and identify pre-deployment capability gaps before they manifest in operations.

## 7-0a. Force Ratio and Relative Combat Power

**NOTE — Doctrinal Responsibility:** ORSA analysts are doctrinally responsible for quantitative force ratio calculations and relative combat power assessments (FM 5-0, Table 5-4). These are structured data products that feed directly into the MDMP — specifically COA development and COA analysis (wargaming). The staff relies on ORSA to produce defensible, quantified comparisons of friendly-to-enemy capability, not subjective estimates.

**Force ratio** expresses the quantitative relationship between opposing forces along a specified dimension (personnel strength, combat vehicles, fires platforms, etc.). **Relative combat power** extends force ratio by applying qualitative weighting factors — training level, morale, terrain advantage, equipment condition — to produce a more operationally relevant comparison.

**ORSA implementation on MSS:**

1. Build a force ratio dataset with columns: `unit_id`, `echelon`, `dimension` (e.g., armor, infantry, fires), `friendly_count`, `enemy_count`, `ratio`, `assessment_date`.
2. Apply combat power weighting factors from the staff estimate. Document every weighting factor, its source (intelligence assessment, commander guidance, historical data), and its uncertainty range.
3. Compute ratios by WFF, by phase, and by geographic area of operations. Present as a structured table, not a single aggregate number — aggregation hides the asymmetries that drive operational risk.
4. Include sensitivity analysis: show how the ratio changes if key intelligence estimates are wrong by 10%, 25%, or 50%. Commanders need to know where the assessment is fragile.

> **WARNING**
>
> Force ratios are not predictive models — they are descriptive inputs to commander judgment. A 3:1 ratio does not guarantee success. Present ratios with explicit caveats about what the numbers include and exclude. Historical force ratio thresholds (e.g., "3:1 for attack") are planning factors, not physical laws.

Feed force ratio products into the COA evaluation criteria schema (Section 1-7, Table 1-2) and the Monte Carlo COA comparison framework (Section 5-1) to produce integrated decision support.

## 7-1. Exercise Data Architecture

Good exercise analysis begins before the first vehicle moves. ORSA analysts must design data collection architecture during the MDMP planning phase, not during execution.

### 7-1a. Task: Design Exercise Data Collection Architecture

**CONDITIONS:** Exercise planning is in progress. OPORD issued or FRAGO forthcoming. ORSA analyst tasked to design metrics framework and data collection architecture for post-exercise analysis.

**STANDARDS:** Data collection plan coordinated with OC/T teams and J3/G3, Foundry datasets provisioned before exercise execution, data schema documented and versioned, collection instruments tested in tabletop before execution.

**EQUIPMENT:** MSS access with Foundry dataset provisioning rights, Appendix C schema templates, coordination access to G3/J3 and OC/T leadership.

**PROCEDURE:**

1. Define commander's analytical requirements. Work from the commander's intent and the training objectives. Every data collection effort must answer a question the commander needs answered. Table 7-1 provides a standard starting framework.

**Table 7-1. Standard Exercise Metric Categories**

| Category | Example Metrics | Collection Method | Foundry Dataset |
|---|---|---|---|
| Timeline adherence | Planned vs. actual phase line times, decision point times | OC/T direct observation, SIGACT log | `exercise_timeline_events` |
| Logistics performance | Class III/V consumption vs. plan, resupply cycle times, stockout events | S4/BSB data, OC/T logs | `exercise_logistics_actuals` |
| C2 effectiveness | Decision latency, OPORD issuance times, subordinate compliance | Battle rhythm observation | `exercise_c2_events` |
| Fires effectiveness | Time from request to effects, percent missions on time | FSE records | `exercise_fires_log` |
| Casualty/DNBI | Simulated casualty rates, MEDEVAC response times | OC/T, MEDIC | `exercise_casualty_events` |
| Red force performance | RED tempo, objective seizure times, attrition exchange | OC/T (RED cell) | `exercise_red_force_actuals` |

1. Provision Foundry datasets for each collection category before exercise execution. Use the schema in Appendix C as a starting template. Data must have consistent primary keys across all datasets (at minimum: `exercise_id`, `event_timestamp`, `unit_uic`).
2. Designate data entry points. Options in order of data quality:
3. **Option A (preferred):** OC/Ts enter data directly into a MSS Workshop collection application on tablet during execution
4. **Option B:** OC/T paper logs, digitized to Foundry within 4 hours of each phase
5. **Option C:** After-action compilation (lowest fidelity — timestamps unreliable)

6. Test the data collection instrument in a tabletop 72 hours before execution. Verify all data entry paths write correctly to Foundry, primary keys are consistent, and the OC/T team understands their data entry responsibilities.

**CAUTION: Data collected on paper and retroactively entered into Foundry degrades significantly in timestamp accuracy. For timeline analysis, Option C (retroactive compilation) renders time-between-events calculations unreliable. Push for Option A or B during planning.**

### 7-2. Task: Build Exercise Aggregation Pipeline

**CONDITIONS:** Exercise data has been collected across multiple Foundry datasets. Analyst needs to aggregate, clean, and compute derived metrics for post-exercise analysis.

**STANDARDS:** Pipeline is idempotent (can be re-run without corrupting results), data quality validation is embedded, all derived metrics are documented with calculation logic.

**EQUIPMENT:** Code Workspace with Python, read access to all exercise collection datasets (timeline, logistics, C2 events), planned phase timeline for comparison.

**PROCEDURE:**

```
import pandas as pd
import numpy as np
from datetime import timedelta

# --- 1. Load exercise datasets ---
timeline_df   = client.datasets.load_dataset(rid_timeline).to_pandas()
logistics_df  = client.datasets.load_dataset(rid_logistics).to_pandas()
c2_df         = client.datasets.load_dataset(rid_c2_events).to_pandas()

# --- 2. Standardize timestamps ---
for df in [timeline_df, logistics_df, c2_df]:
    df["event_timestamp"] = pd.to_datetime(df["event_timestamp"], utc=True)

# --- 3. Data quality validation ---
def validate_dataset(df: pd.DataFrame, name: str, required_cols: list) -> bool:
    """Validate required columns and flag missing/null critical fields."""
    issues = []
    for col in required_cols:
        if col not in df.columns:
            issues.append(f"MISSING COLUMN: {col}")
        elif df[col].isnull().any():
            null_count = df[col].isnull().sum()
            issues.append(f"NULL VALUES in {col}: {null_count} rows
({null_count/len(df)*100:.1f}%)")
    if issues:
        print(f"\nData Quality Issues — {name}:")
        for issue in issues:
            print(f"  WARNING: {issue}")
        return False
    print(f"  {name}: {len(df)} rows — validated OK")
    return True
```

```
validate_dataset(timeline_df, "Timeline Events",
                 ["exercise_id", "event_timestamp", "unit_uic", "event_type",
"phase"])
validate_dataset(logistics_df, "Logistics Actuals",
                 ["exercise_id", "event_timestamp", "unit_uic", "supply_class",
"quantity"])

# --- 4. Compute timeline performance metrics ---
# Planned vs. actual phase crossing times
planned_times = pd.DataFrame({
    "phase":          ["PHASE_I_START", "PHASE_II_START", "PHASE_III_START", "EXEX"],
    "planned_time":   pd.to_datetime([
        "2026-03-15 06:00:00+00:00",
        "2026-03-15 12:00:00+00:00",
        "2026-03-16 06:00:00+00:00",
        "2026-03-17 18:00:00+00:00"
    ])
})

actual_phase_times = (
    timeline_df[timeline_df["event_type"] == "PHASE_CROSSING"]
    .groupby("phase")["event_timestamp"]
    .first()
    .reset_index()
    .rename(columns={"event_timestamp": "actual_time"})
)

timeline_comparison = planned_times.merge(actual_phase_times, on="phase", how="left")
timeline_comparison["delta_minutes"] = (
    (timeline_comparison["actual_time"] - timeline_comparison["planned_time"])
    .dt.total_seconds() / 60
)
print("\nTimeline Performance:")
print(timeline_comparison[["phase", "planned_time", "actual_time",
"delta_minutes"]].to_string(index=False))

# --- 5. Compute logistics metrics ---
# Resupply cycle time: time between request and delivery
resupply_requests  = logistics_df[logistics_df["event_type"] ==
"RESUPPLY_REQUEST"].copy()
resupply_deliveries = logistics_df[logistics_df["event_type"] ==
"RESUPPLY_DELIVERY"].copy()

resupply_cycles = resupply_requests.merge(
    resupply_deliveries,
    on=["exercise_id", "unit_uic", "resupply_id"],
    suffixes=("_request", "_delivery")
)
resupply_cycles["cycle_time_hrs"] = (
    (resupply_cycles["event_timestamp_delivery"] -
resupply_cycles["event_timestamp_request"])
    .dt.total_seconds() / 3600
)
```

```
print("\nResupply Cycle Time Summary:")
print(resupply_cycles["cycle_time_hrs"].describe().round(2))

# --- 6. Compile summary metrics for commander brief ---
summary_metrics = {
    "exercise_id":                  timeline_df["exercise_id"].iloc[0],
    "total_events_captured":        len(timeline_df),
    "phase_i_delta_minutes":
timeline_comparison.loc[timeline_comparison["phase"]=="PHASE_I_START",
"delta_minutes"].values[0] if len(timeline_comparison) > 0 else None,
    "mean_resupply_cycle_hrs":      resupply_cycles["cycle_time_hrs"].mean(),
    "p90_resupply_cycle_hrs":
resupply_cycles["cycle_time_hrs"].quantile(0.90),
    "n_stockout_events":            (logistics_df["event_type"] ==
"STOCKOUT").sum(),
}

print("\nSummary Metrics for Commander Brief:")
for k, v in summary_metrics.items():
    print(f"  {k}: {v}")
```

### 7-3. Task: Generate Post-Exercise Report

**CONDITIONS:** Exercise aggregation pipeline complete. Summary metrics computed. Commander requires an AAR-quality analytical product within 24 hours of EXEX.

**STANDARDS:** Report includes quantitative findings for each training objective, comparison to standard/benchmark where available, identified trends, and BLUF recommendations.

**EQUIPMENT:** Code Workspace with Python, summary metrics dataset from Task 7-2, training objectives and published standards for comparison, Quiver or Contour access for visualization delivery.

**PROCEDURE:**

1. Structure the report using the standard AAR framework: Planned vs. Actual, Sustains, Improves.
2. For each training objective, provide:
3. The metric used to measure it
4. The measured result with uncertainty/variability
5. Comparison to previous exercise or published standard
6. Color-coded status (MET/PARTIALLY MET/NOT MET) with quantitative basis
7. Use the summary statistics computed in paragraph 7-2. Include histograms and CDF plots for time-based metrics (cycle times, decision latency) rather than means alone. A mean that masks a heavy tail (one catastrophic delay) is misleading.
8. Export the report product to a Foundry dataset for Contour/Workshop visualization, and generate a static PDF for distribution to commanders without MSS access.

# CHAPTER 8 — DECISION SUPPORT PRODUCTS

**BLUF:** ORSA analytical products are only valuable if commanders can use them. Quiver and Contour on MSS provide the visualization layer that translates technical analysis into commander-ready decision support. Build products for the consumer — not for other analysts.

## 8-1. Principles of Commander-Ready Visualization

Before building any analytical dashboard or visualization, internalize the following design principles:

**Principle 1 — Answer One Question Per View.** Each Quiver chart or Contour workbook section should answer a single, specific question. "What is the readiness trend for 1st Brigade?" is a question. "Readiness overview dashboard" is not.

**Principle 2 — Lead with BLUF.** The most important finding should appear in the top-left of any product, in plain language, before any charts. A commander scanning a product for 30 seconds should know the main point.

**Principle 3 — Show Uncertainty Visually.** Every forecast chart must show confidence intervals. Every percentage must show sample size. Confidence bands, error bars, and range shading are not optional decoration.

**Principle 4 — Red/Amber/Green with Quantitative Basis.** Traffic light colors are operationally useful only when the thresholds are defined, documented, and consistent. Never apply RAG status based on analyst intuition.

**Principle 5 — Commander Time is the Constraint.** A product a commander will use in 30 seconds beats a comprehensive product they will not open. Build for the shortest effective decision time, not analytical completeness.

## 8-2. Task: Build a Readiness Forecast Dashboard in Quiver

**NOTE — Palantir Developers reference:** *Quiver | How to Build an Analysis in Palantir Foundry* — Covers the end-to-end workflow for creating a Quiver analysis from scratch: connecting data sources, building charts, and publishing a workbook. Watch before your first Quiver build if you have not used the tool before. Available on the Palantir Developers YouTube channel (@PalantirDevelopers).

**CONDITIONS:** Readiness forecast model output dataset exists in Foundry (from Chapter 4 procedure). Commander requires an always-current forecast view integrated into the existing readiness dashboard.

**STANDARDS:** Dashboard displays current C-ratings, 90-day forecast, prediction intervals, and highlights units at risk of C3 or below within the forecast horizon. Refreshes automatically on data update.

**EQUIPMENT:** MSS access with Quiver authoring rights, forecast output dataset (from Task 4-2), read access to current readiness dataset.

**PROCEDURE:**

1. In Foundry, navigate to the project Quiver workspace. Create a new workbook: `ORSA_Readiness_Forecast_[BRIGADE_NAME]`.
2. Connect the forecast output dataset (`orsa_readiness_forecast`) as the primary data source.
3. **Sheet 1 — Formation Summary (Commander View):**
4. Add a **Pivot table** showing: rows = unit UIC/name, columns = current month + next 3 months, values = predicted C-rating (mean)
5. Apply **conditional formatting:** green = C1/C2, amber = C3, red = C4/C5 (configure threshold values in the format rules, not hardcoded)
6. Add a **KPI tile** at top: count of units projected to be below C3 within 90 days

**NOTE — Palantir Developers reference:** *Quiver | Calculating KPIs for Time Series Data in Palantir Foundry* — Demonstrates how to compute and display KPI metrics derived from time-series data in a Quiver workbook, including rolling aggregations and threshold-based indicators. Directly applicable to readiness forecast KPI tiles and trend charts in this dashboard. Available on the Palantir Developers YouTube channel (@PalantirDevelopers).

1. **Sheet 2 — Unit Forecast Detail (Staff View):**
2. Add a **Line chart** with: x-axis = month, y-axis = C-rating
3. Series 1: Historical actuals (solid navy line)
4. Series 2: Forecast mean (dashed orange line)
5. Band: lower_bound_90 to upper_bound_90 (shaded orange, 30% opacity)
6. Reference line: y=3 (C3 threshold, dashed red)
7. Connect a **Unit selector** parameter widget to filter by UIC
8. Add a **Last Updated** text tile that shows the `generated_date` from the dataset

**NOTE — Palantir Developers reference:** *Quiver | How to Use Parameters in Your Analysis* — Covers creating and wiring parameter widgets (dropdowns, date pickers, text inputs) to drive dynamic filtering across charts in a Quiver workbook. Directly applicable to the Unit selector parameter in Sheet 2. Available on the Palantir Developers YouTube channel (@PalantirDevelopers).

1. **Sheet 3 — ORSA Notes (Analytical Metadata):**
2. Free text tile documenting: model type (ARIMA order), training data range, backtest MAPE, known data quality issues, date of last model refresh
3. This sheet is for staff analysts to validate the product — do not show on commander briefing
4. Set **refresh schedule** to align with the upstream readiness data pipeline update cadence (typically weekly).

**NOTE**

Quiver workbooks connected to live Foundry datasets automatically reflect new data when the underlying dataset updates — provided the dataset schema (column names and types) remains stable. If the ORSA pipeline changes column names, Quiver visualizations will break. Version control your output dataset schema and communicate schema changes to all consumers before implementing them.

**NOTE — Palantir Developers reference:** *Quiver | How to Navigate the Dependency Graph and Expand your Analysis* — Shows how to use Quiver's dependency graph view to trace upstream data lineage, identify which datasets feed a given chart, and extend an analysis by branching from existing nodes. Use this when diagnosing why a workbook stopped updating or when expanding a readiness dashboard to incorporate additional data sources. Available on the Palantir Developers YouTube channel (@PalantirDevelopers).

## 8-3. Task: Build an Analytical Contour Workbook for COA Support

**NOTE — Palantir Developers reference:** *Contour | Building a Year Over Year Sales Dashboard* — Demonstrates how to structure a Contour workbook for period-over-period comparison, including calculated fields for delta/variance, reference lines, and layout for executive audiences. The YoY comparison pattern translates directly to readiness trend analysis (current quarter vs. prior quarter) and COA outcome comparisons. Available on the Palantir Developers YouTube channel (@PalantirDevelopers).

**CONDITIONS:** G3 requires a product supporting COA comparison briefing. Monte Carlo results from Chapter 5 are stored in a Foundry dataset.

**STANDARDS:** Contour workbook presents probability of exceedance curves, distribution comparison, key percentiles table, and BLUF recommendation — all in a format suitable for direct projection during a briefing.

**EQUIPMENT:** MSS access with Contour authoring rights, Monte Carlo simulation results dataset (from Task 5-1a), sensitivity analysis results dataset (from Task 5-2a).

**PROCEDURE:**

1. Create a Contour workbook: `ORSA_COA_Risk_Analysis_[OPERATION]`.
2. Connect the Monte Carlo results dataset (e.g., `orsa_coa_monte_carlo_results` with columns: `coa_id`, `simulation_run`, `outcome_value`).
3. **Section 1 — BLUF:** Text section at top of document. Complete this section last, after analyzing the data. State: which COA has lower expected time/cost/risk, the probability each COA meets the planning threshold, and the key uncertainty driver from the sensitivity analysis.
4. **Section 2 — Outcome Distribution Comparison:**
5. Histogram (overlapping, 50% opacity) of outcome values for each COA

6. Vertical reference line at planning threshold
7. Table below chart: Mean, P10, P50, P90 for each COA, and P(meets threshold) for each COA

**NOTE — Palantir Developers reference:** *Quiver | How to Perform Ad-Hoc Aggregations* — Covers how to perform on-the-fly group-by aggregations (sum, count, percentile) in Quiver without modifying the underlying dataset or pipeline. Applicable when you need to quickly roll up simulation run outputs by COA for summary statistics during exploratory analysis before publishing a formal Contour product. Available on the Palantir Developers YouTube channel (@PalantirDevelopers).

1. **Section 3 — Probability of Exceedance:**
2. Line chart: x = outcome value, y = P(outcome > x), separate lines for each COA
3. Shade the region right of planning threshold in red
4. **Section 4 — Sensitivity (Tornado Chart):**
5. Horizontal bar chart from sensitivity analysis dataset
6. Label bars with Spearman correlation values
7. Export a static snapshot PDF for the briefing package. In Contour: File > Export > PDF. Name it: `ORSA_COA_[DATE]_[CLASSIFICATION].pdf`.

---

# CHAPTER 9 — COMMUNICATING UNCERTAINTY

**BLUF:** Uncertainty communication is the most consequential ORSA skill. The correct forecast, incorrectly communicated, can lead to worse decisions than no forecast at all. Every ORSA briefing product must be designed to give commanders accurate calibration of analytical confidence.

## 9-1. Confidence Intervals vs. Prediction Intervals

Many analysts confuse confidence intervals (CI) and prediction intervals (PI). For commander briefings, this distinction matters.

**Table 9-1. Confidence Interval vs. Prediction Interval**

| Concept | What it bounds | When to use | Typical width |
|---|---|---|---|
| **Confidence Interval** | Where the true mean of the population lies | When briefing the average behavior across many units | Narrower (grows smaller with more data) |
| **Prediction Interval** | Where the next individual observation will fall | When briefing what will happen to a specific unit | Always wider than CI; does not shrink to zero |

| Concept | What it bounds | When to use | Typical width |
|---|---|---|---|
| **Credible Interval (Bayesian)** | Where the parameter lies given the data and prior | When using Bayesian modeling | Context-dependent |

**WARNING: Never brief a confidence interval as if it were a prediction interval. A 90% CI for the mean readiness of a brigade does NOT mean that 90% of individual units will fall within that range. Briefing a CI to a commander who thinks they are hearing a PI may cause them to under-plan for variability. State explicitly which type of interval you are reporting.**

## 9-2. Briefing Standards for ORSA Products

**Table 9-2. Required Uncertainty Language by Product Type**

| Product Type | Required Statement | Example Language |
|---|---|---|
| Point forecast | Type of interval, confidence level, horizon | "The model forecasts 847 Class IX parts in April, with a 90% prediction interval of 710–990 parts." |
| Readiness C-rating forecast | Prediction interval, explicit model limitations | "2-1 IN is forecast at C2 in 90 days. The 80% prediction interval spans C1 to C3. This forecast degrades beyond 60 days; treat 90-day outlook as directional." |
| COA risk estimate | Percentile framing, P(meets threshold) | "COA A has a 73% probability of achieving FOC within 7 days. COA B has a 58% probability." |
| Classification (risk flag) | Precision, recall, and what false positives/negatives mean operationally | "This model correctly identifies 81% of units that will degrade (recall). Of units it flags, 67% actually degrade (precision). Verify flagged units with SME review." |
| Optimization output | Constraint sensitivity, assumption limitations | "This schedule is optimal given the stated constraints. If mechanic availability drops below 12 hours/day, systems A3 and C1 will miss their deadlines." |

## 9-3. Task: Conduct Sensitivity Analysis for Commander Brief

**CONDITIONS:** You have a forecast or model result that will inform a significant resource or operational decision. Commander has asked how confident you are in the result.

**STANDARDS:** Sensitivity analysis shows how the key conclusions change across a range of assumption values. Output is a table or chart the commander can interpret in under 30 seconds.

**EQUIPMENT:** Code Workspace with Python, the analytical model or forecast being assessed, documented key assumptions and their uncertainty ranges.

**PROCEDURE:**

```
import numpy as np
import pandas as pd
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt

# --- Scenario: Logistics demand forecast sensitivity ---
# Key assumption: daily consumption rate
# How does the 90% stockage recommendation change as this assumption varies?

service_level      = 0.90
resupply_interval  = 7
lead_time_days     = 2
std_daily_demand   = 3.8  # held constant for this sensitivity
N_SIM              = 50_000

# Vary the mean daily demand assumption from -20% to +20%
base_mean = 12.4
sensitivity_range = np.linspace(base_mean * 0.80, base_mean * 1.20, 9)

results = []
for mean_demand in sensitivity_range:
    total_demand_period = resupply_interval + lead_time_days
    daily_demands = np.random.normal(mean_demand, std_daily_demand,
                                     size=(N_SIM, total_demand_period))
    daily_demands = np.clip(daily_demands, 0, None)
    total = daily_demands.sum(axis=1)
    recommended = np.percentile(total, service_level * 100)
    results.append({
        "assumed_daily_demand": round(mean_demand, 1),
        "assumption_vs_base":   f"{(mean_demand/base_mean - 1)*100:+.0f}%",
        "recommended_stockage": int(np.round(recommended)),
        "safety_stock":         int(np.round(recommended - total.mean()))
    })

sensitivity_df = pd.DataFrame(results)
print("SENSITIVITY TABLE — Stockage Recommendation vs. Consumption Rate Assumption")
print(sensitivity_df.to_string(index=False))

# Highlight: how much does the recommendation change across the full range?
delta = sensitivity_df["recommended_stockage"].max() -
sensitivity_df["recommended_stockage"].min()
print(f"\nStockage range across ±20% consumption assumption: {delta} units")
print(f"This represents a
{delta/sensitivity_df.loc[4,'recommended_stockage']*100:.1f}% planning buffer if
assumption is uncertain.")

# Plot
fig, ax = plt.subplots(figsize=(9, 4))
ax.plot(sensitivity_df["assumed_daily_demand"],
        sensitivity_df["recommended_stockage"],
        marker="o", color="navy", linewidth=2)
ax.axvline(base_mean, color="green", linestyle="--", label=f"Base estimate:
{base_mean}")
ax.fill_between(
```

```
    sensitivity_df["assumed_daily_demand"],
    sensitivity_df["recommended_stockage"] - 5,
    sensitivity_df["recommended_stockage"] + 5,
    alpha=0.15, color="navy"
)
ax.set_xlabel("Assumed Mean Daily Demand (units/day)")
ax.set_ylabel("Recommended Stockage Level (units)")
ax.set_title("Sensitivity: Stockage Recommendation vs. Consumption Assumption\n(90%
Service Level, 7-day Resupply Interval)")
ax.legend()
plt.tight_layout()
plt.savefig("stockage_sensitivity.png", dpi=150)
plt.close()
print("Sensitivity plot saved.")
```

### 9-4. Briefing Posture for ORSA Analysts

When briefing ORSA products to commanders and staff, apply the following standards:

1. **Lead with the answer, not the method.** Commanders need the finding and what to do with it. Methodology is a backup when questioned. Do not spend the first three slides on your model architecture.
2. **State confidence explicitly and early.** "I am highly confident in this finding" and "This is directional only — treat with caution" are both valid statements. Both must appear in the product. Commanders cannot calibrate decisions without knowing your confidence level.
3. **Anticipate the second question.** After "what does this tell me," the commander will ask "what happens if you're wrong?" Have your sensitivity analysis ready.
4. **Distinguish what the data shows from what you recommend.** The data showed X. Given X, the analyst recommends Y. The commander decides Z. Be clear about which statement you are making at any given moment.
5. **Never revise an analytical finding under social pressure.** If a commander or staff officer challenges your result because they do not like it, your response is to examine the challenge technically: are there data issues? Are there factors your model did not capture? If the technical basis is sound, hold the finding. Analysts who change results for social reasons destroy the credibility of the entire ORSA function.

## APPENDIX A — ORSA PRODUCT STANDARDS CHECKLIST

Use this checklist before submitting any ORSA product to a commander or staff officer.

### A-1. Data Quality

- ☐ Source datasets identified with dataset RIDs and date ranges
- ☐ Null rates on critical fields documented
- ☐ Outliers investigated and disposition documented (removed/retained with justification)
- ☐ Data quality limitations stated in product

### A-2. Model Documentation

- ☐ Model type and configuration documented (ARIMA order, classifier parameters, LP formulation)
- ☐ All assumptions stated explicitly
- ☐ Training data described (time period, unit coverage)
- ☐ Potential sources of bias identified

### A-3. Validation Evidence

- ☐ Out-of-sample performance metrics computed (temporal holdout, not random split)
- ☐ Performance metrics appropriate to model type (Table 3-1)
- ☐ Backtesting results documented for time series models
- ☐ Confusion matrix and precision/recall documented for classifiers

### A-4. Uncertainty Quantification

- ☐ Confidence intervals or prediction intervals included for every point estimate
- ☐ Interval type (CI vs. PI) explicitly labeled
- ☐ Sensitivity analysis performed for key uncertain inputs
- ☐ Model limitations stated in product

### A-5. Reproducibility

- ☐ Analysis code stored in Code Workspace repository
- ☐ Package versions documented
- ☐ Random seed documented for simulation products
- ☐ Source dataset RIDs in code header

### A-6. Commander-Ready Presentation

- ☐ BLUF paragraph in plain language (no technical jargon)

- ☐ Key finding appears in first 30 seconds of review
- ☐ RAG status thresholds defined, documented, and consistent
- ☐ Uncertainty shown visually (confidence bands, prediction intervals)
- ☐ Recommendation clearly distinguished from finding

### A-7. Governance

- ☐ C2DAO coordination complete for new data products
- ☐ Classification markings verified on all deliverables
- ☐ External tool data sharing approved by C2DAO (if applicable)
- ☐ Product version labeled and dated

## APPENDIX B — STATISTICAL QUICK REFERENCE: ARMY-RELEVANT USE CASES

### B-1. Choosing the Right Method

**Table B-1. Statistical Method Selection Guide**

| Analytical Question | Data Type | Recommended Method | Key Assumption to Verify |
|---|---|---|---|
| Will this unit's readiness degrade in 60 days? | Binary outcome, tabular features | Logistic Regression or Random Forest Classifier | Sufficient positive cases (n > 50 events) |
| How many Class IX parts will we need next month? | Continuous, time-ordered | SARIMA or SARIMAX | Stationarity; seasonal period identified |
| Which COA minimizes risk of missing FOC date? | Continuous output, uncertain inputs | Monte Carlo Simulation | Input distributions calibrated to data |
| How should I allocate 45 mechanics across 6 units? | Continuous decision vars, linear constraints | Linear Programming | Objective function is linear; constraints are linear |
| Is this year's readiness trend better than last year? | Two groups, continuous outcome | t-test or Mann-Whitney U (if non-normal) | Independence, equal variance (t-test); sample size |

| Analytical Question | Data Type | Recommended Method | Key Assumption to Verify |
|---|---|---|---|
| Does training days correlate with C-rating improvement? | Two continuous vars | Pearson correlation (normal) or Spearman (non-normal) | Check for outliers; correlation ≠ causation |
| What percentage of units are C3 or below? | Binary (threshold applied to continuous) | Descriptive statistics + CI for proportion | Sample representativeness |
| Is readiness improving since the new maintenance SOP? | Before/after, continuous | Interrupted time series (ITSA) | No other confounding changes at same time |

## B-2. Sample Size Rules of Thumb

**Table B-2. Minimum Sample Size Guidance**

| Application | Minimum n | Notes |
|---|---|---|
| Linear regression (OLS) | 10× number of predictors | Shrink model if n is insufficient |
| Logistic regression | 10-20 events per predictor | "Events" = the minority class (risk cases) |
| Random Forest classifier | 50+ minority class events | Below this, ROC-AUC is unreliable |
| ARIMA time series | 24+ observations (monthly) | 36+ recommended for seasonal models |
| Monte Carlo simulation | 10,000+ runs | 50,000+ for stable tail percentiles (P90+) |
| t-test | 30+ per group | Below 30, verify normality; use Welch's t |
| Correlation | 30+ | Instability below 30 with typical Army data variance |

## B-3. Army C-Rating Numeric Equivalents

For quantitative analysis, convert C-ratings to numeric values per the standard mapping:

**Table B-3. C-Rating to Numeric Mapping**

| C-Rating | Numeric Value | Interpretation |
|---|---|---|
| C1 | 4 | Fully capable — meet all METL tasks |
| C2 | 3 | Mostly capable — minor deficiencies |
| C3 | 2 | Marginally capable — significant deficiencies |
| C4 | 1 | Not capable — cannot perform METL tasks |

| C-Rating | Numeric Value | Interpretation |
|---|---|---|
| C5 | 0 | Forming/not reportable |

**NOTE**

The numeric mapping treats C-ratings as ordinal, not interval. A move from C3 to C2 is not necessarily the same "distance" as C2 to C1. For regression modeling, verify this assumption or use ordinal logistic regression. For most ORSA applications, the ordinal-to-numeric mapping is operationally adequate.

## B-4. Common Errors in ORSA Products

### Table B-4. Common Errors and Corrections

| Error | Consequence | Correct Approach |
|---|---|---|
| Random train/test split on time series data | Optimistic performance estimate; model "trains on the future" | Always split chronologically |
| Reporting CI as PI | Under-estimated unit-level variability | Report PI for individual unit forecasts |
| Accuracy metric on imbalanced classifier | Model appears accurate while being useless | Report precision, recall, F1 for minority class |
| Point estimate without uncertainty | False precision in commander decision | Always accompany with CI/PI/range |
| Correlation presented as causation | Incorrect causal inference | Explicitly state correlation does not imply causation; discuss confounders |
| Over-fit model (high train, low test performance) | Model fails on new data | Regularization, simpler model, cross-validation |
| Ignoring seasonality in time series | Poor forecast accuracy for Army data | Use SARIMA; inspect ACF at lag 12 |
| LP with too few constraints | Unrealistic solution | Validate solution against domain knowledge; add missing constraints |

# APPENDIX C — WARGAME DATA COLLECTION TEMPLATES

## C-1. Timeline Event Schema

Use this schema for the `exercise_timeline_events` Foundry dataset.

**Table C-1. Timeline Event Dataset Schema**

| Field | Type | Description | Required |
|---|---|---|---|
| `exercise_id` | string | Unique exercise identifier (e.g., "DEFENDER_26_MAIN") | Y |
| `event_id` | string | Unique event UUID | Y |
| `event_timestamp` | timestamp (UTC) | DTG of event occurrence | Y |
| `unit_uic` | string | Reporting unit UIC | Y |
| `event_type` | string | Controlled vocabulary (PHASE_CROSSING, DECISION_POINT, SIGNIFICANT_ACT, CASUALTY_EVENT) | Y |
| `phase` | string | Exercise phase label | Y |
| `planned_time` | timestamp (UTC) | Planned DTG for this event (null if unplanned) | N |
| `event_description` | string | Free text description (max 500 chars) | N |
| `oc_t_id` | string | OC/T badge number who recorded event | Y |
| `data_entry_method` | string | "LIVE_APP", "PAPER_RETROACTIVE", or "RECONSTRUCTION" | Y |
| `created_at` | timestamp (UTC) | System-generated on insert | Y |

## C-2. Logistics Actuals Schema

**Table C-2. Logistics Actuals Dataset Schema**

| Field | Type | Description | Required |
|---|---|---|---|
| `exercise_id` | string | Exercise identifier | Y |

| Field | Type | Description | Required |
|---|---|---|---|
| `event_id` | string | Unique event UUID | Y |
| `event_timestamp` | timestamp (UTC) | DTG of event | Y |
| `unit_uic` | string | Requesting unit | Y |
| `event_type` | string | Controlled vocabulary (RESUPPLY_REQUEST, RESUPPLY_DELIVERY, STOCKOUT, CONSUMPTION_RECORD) | Y |
| `supply_class` | string | Supply class (I, II, III, IIIB, IV, V, VII, VIII, IX) | Y |
| `nsn_or_commodity` | string | NSN or commodity group code | N |
| `quantity` | float | Quantity (units per supply class convention) | Y |
| `resupply_id` | string | Links request and delivery records | Y (for request/delivery pairs) |
| `planned_quantity` | float | Planned issue quantity (null if unplanned event) | N |
| `oc_t_id` | string | Recording OC/T | Y |

## C-3. Pre-Exercise Data Validation Checklist

Before exercise execution begins, verify:

- ☐ All schemas provisioned in Foundry dev branch
- ☐ Primary keys ( `exercise_id` , `unit_uic` ) consistent across all datasets
- ☐ OC/T data entry application tested and connected to correct datasets
- ☐ Controlled vocabulary lists (event_type, supply_class) distributed to OC/T team
- ☐ Fallback paper forms formatted consistently with schema
- ☐ ORSA analyst on-call contact distributed to OC/T team leads
- ☐ Pipeline to aggregate raw events to summary metrics tested on synthetic data
- ☐ Output dashboard visible and functioning in Workshop/Quiver

## APPENDIX D — PROFESSIONAL READING LIST

Curated articles from Army professional journals and military publications. These supplement doctrinal references with contemporary operational perspectives.

| Source | Title | Date | Relevance |
|---|---|---|---|
| Small Wars Journal | "Accelerating Decision-Making: Integrating AI into the Modern Wargame" | Feb 2026 | AI-enabled wargaming |
| Small Wars Journal | "AI-Enabled Wargaming at CGSC" | Jan 2026 | Wargaming implications for PME |
| Military Review | "AI as a Combat Multiplier: Using AI to Unburden Army Staffs" | Sep 2024 | AI decision support for staffs |
| Military Review | "Taking a Data-Centric Approach to Unit Readiness" | 2024 | Analytics in BCT readiness |
| Green Notebook | "Understanding Weapons of Math Destruction" | Jul 2024 | Algorithmic bias and risk |

## GLOSSARY

**AAR (After-Action Review)** — Structured review of exercise or operational events against standards, conducted to identify sustains and improves.

**ADF Test (Augmented Dickey-Fuller Test)** — Statistical test for unit root in a time series; used to assess stationarity before fitting ARIMA models. p-value < 0.05 rejects the null hypothesis of a unit root.

**AIC (Akaike Information Criterion)** — Model selection criterion that penalizes model complexity; lower AIC indicates better fit relative to model complexity. Used for ARIMA order selection.

**ARIMA (Autoregressive Integrated Moving Average)** — Time series forecasting model combining autoregressive (AR), differencing (I), and moving average (MA) components. Parameterized as ARIMA(p, d, q).

**AOR (Area of Responsibility)** — The geographic area for which a commander has authority and accountability.

**Binary Classification** — Supervised learning task predicting one of two outcomes (e.g., readiness degrades / does not degrade).

**BLUF (Bottom Line Up Front)** — Army writing standard requiring the main point to appear first, before supporting detail.

**C-Rating (C1–C5)** — Army unit readiness rating. C1 = fully capable; C5 = forming/not reportable. Defined by AR 220-1.

**Calibration (Model)** — Verification that predicted probabilities match observed frequencies. A well-calibrated model that says "70% probability" should be right 70% of the time across many such predictions.

**C2DAO (C2 Data and Analytics Office)** — USAREUR-AF governance authority for data standards, analytical product standards, and MSS/Foundry policy. Coordination authority for new data products and external tool connections.

**Confidence Interval (CI)** — Range that contains the true population parameter with specified probability. Bounds the mean, not individual observations. Narrows with more data.

**COA (Course of Action)** — A potential plan of action evaluated during the military decision-making process (MDMP).

**Cross-Validation** — Model validation technique using multiple train/test splits to estimate out-of-sample performance. Time series cross-validation must respect temporal order.

**DNBI (Disease and Non-Battle Injury)** — Casualties resulting from illness or injury not caused by enemy action; a key metric in medical/logistics readiness analysis.

**ETL (Extract, Transform, Load)** — The process of extracting data from source systems, transforming it for analysis, and loading it to a destination dataset.

**EXEVAL (External Evaluation)** — Formal assessment of a unit's mission capability conducted by evaluators external to the unit.

**F1 Score** — Harmonic mean of precision and recall. Balanced metric for classifiers, especially valuable with class imbalance. F1 = 2 × (precision × recall) / (precision + recall).

**FOC (Full Operational Capability)** — The condition at which a unit or system can perform all assigned functions per its doctrine and design.

**Foundry** — Palantir data platform underlying MSS. Provides dataset management, pipeline orchestration, ontology, and Code Workspaces.

**HiGHS Solver** — High-performance linear programming solver embedded in scipy >= 1.9. Recommended for LP formulations in Python ORSA work.

**Idempotent** — Property of a pipeline or operation: running it multiple times produces the same result as running it once. Required for all production ORSA pipelines.

**Interrupted Time Series Analysis (ITSA)** — Quasi-experimental design for assessing the effect of an intervention on a time series outcome (e.g., did a new maintenance SOP change the readiness trend?).

**JDLM (Joint Deployment Logistics Model)** — Army simulation tool for deployment logistics analysis. C2DAO coordination required before connecting MSS data to JDLM.

**Lead Time (Logistics)** — Time between placement of a requisition and receipt of the item. A key parameter in stockage level modeling.

**Linear Programming (LP)** — Optimization method for finding the maximum or minimum of a linear objective function subject to linear constraints.

**MAPE (Mean Absolute Percentage Error)** — Forecast accuracy metric: mean of |actual - forecast| / actual × 100. Expressed as a percentage.

**MDMP (Military Decision-Making Process)** — The Army's seven-step planning process: receipt of mission, mission analysis, COA development, COA analysis, COA comparison, COA approval, orders production.

**METL (Mission Essential Task List)** — The set of collective tasks a unit must be able to perform to accomplish its wartime mission.

**Monte Carlo Simulation** — Computational technique that models uncertainty by sampling from input distributions and aggregating the resulting output distribution over many iterations.

**MSN (Mission)** — The assigned task and its purpose.

**MSS (Maven Smart System)** — The USAREUR-AF enterprise AI/data platform, built on Palantir Foundry. The primary analytical environment for ORSA work described in this manual.

**OPDATA (Operational Data)** — Data generated by or supporting operational activities. Handled with appropriate classification and governance controls.

**OLS (Ordinary Least Squares)** — Standard linear regression estimation method minimizing the sum of squared residuals.

**ORSA (Operations Research/Systems Analysis)** — The application of quantitative methods to provide commanders with analytical decision support. ORSA officers (FA49) are the primary practitioners.

**OC/T (Observer/Controller/Trainer)** — Personnel assigned to observe and evaluate units during exercises.

**Overfitting** — Model condition where high training-set performance does not generalize to new data. Diagnosed by large gap between training and holdout performance metrics.

**P-value** — Probability of observing a test statistic as extreme as the observed value, given the null hypothesis is true. Does not measure practical significance.

**Prediction Interval (PI)** — Range that will contain the next individual observation with specified probability. Always wider than the confidence interval for the mean.

**Precision (Classifier)** — Of the units the model flagged as "at risk," the fraction that actually were at risk. Precision = TP / (TP + FP).

**Recall (Classifier)** — Of all units that were actually at risk, the fraction the model correctly flagged. Recall = TP / (TP + FN). Also called "sensitivity."

**ROC-AUC** — Area under the receiver operating characteristic curve. Measures classifier discrimination ability across all threshold values. 0.5 = random; 1.0 = perfect.

**SARIMA** — Seasonal ARIMA. Extends ARIMA with seasonal AR, differencing, and MA components. Parameterized as SARIMA(p, d, q)(P, D, Q, s) where s = seasonal period (12 for monthly Army data).

**SARIMAX** — SARIMA with exogenous (external) predictor variables. Used when known operational events (EXEVAL, deployment) affect the time series.

**Sensitivity Analysis** — Systematic evaluation of how model outputs change in response to changes in model inputs or assumptions.

**Service Level (Logistics)** — The probability that demand is met from stock during a resupply cycle without stockout. Common Army targets: 85–95% depending on criticality.

**Shadow Price (LP)** — The rate of change of the optimal objective value with respect to a one-unit change in a constraint right-hand-side value. Indicates the marginal value of relaxing a constraint.

**Spearman Rank Correlation** — Non-parametric correlation measure that ranks both variables before computing Pearson correlation. Robust to non-normal distributions and outliers.

**Stationarity** — Property of a time series where statistical properties (mean, variance, autocorrelation) do not change over time. Required for standard ARIMA modeling.

**Tornado Chart** — Horizontal bar chart ranking input variables by their impact on an output, used in sensitivity analysis. Resembles a tornado (widest bars at top).

**UDRA (Unified Data Reference Architecture)** — v1.1 (February 2025). The Army's authoritative data architecture reference. Governs data standards, interoperability, and platform alignment for Army data programs.

**UIC (Unit Identification Code)** — The unique alphanumeric identifier assigned to each Army unit.

**Uncertainty Quantification** — The systematic characterization of sources and magnitude of uncertainty in analytical models and their outputs.

---

*This publication supersedes all previous ORSA-specific guidance for MSS/Foundry Code Workspace operations in the USAREUR-AF AOR.*

*For questions or recommended changes to this publication, contact the USAREUR-AF C2DAO, Wiesbaden, Germany.*

*Reference: learn-data.armydev.com for current MSS platform documentation.*

*Army CIO Memorandum, Data and Analytics Policy, April 2024. Unified Data Reference Architecture (UDRA) v1.1, February 2025.*

**DoD and Army Strategic References:**

- **JADC2 Strategy Summary (March 2022)** — Cross-domain data integration strategy for Joint All-Domain Command and Control