DS 5110: Introduction to Data Management and Processing

Fall 2025

Athlete Load Management & Performance Optimization Platform

Final Project Progress Report

Team Members:

Harsha Prakash
Samuel Greeman

1 Dataset Description

Dataset Link: Not applicable - synthetic data generated programmatically

Dataset Description: We generated a synthetic dataset using Python's Random library to simulate 50 athletes over 6 months. The dataset contains approximately 5,000 records covering training sessions, injury events, and recovery metrics. Each athlete has daily training load measurements and periodic injury occurrences based on realistic probability distributions.

Dataset Suitability: Synthetic data was chosen to ensure full control over the relationships between training loads and injury occurrences while avoiding privacy concerns. This approach allows us to validate our models with known ground truth and ensure reproducibility for academic evaluation.

Listing 1: Data Generation Code

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

# Set random seed for reproducibility
np.random.seed(42)
random.seed(42)

# Number of entries
n_entries = 5000

# Generate athlete IDs (50 athletes with multiple sessions each)
athlete_ids = [f"ATH_{str(i).zfill(3)}" for i in range(1, 51)]

# Training session types
session_types = ['Endurance', 'Strength', 'Speed', 'HIIT', 'Recovery
    ', 'Skills', 'Match/Competition']

# Muscle groups
muscle_groups = ['Hamstring', 'Quadriceps', 'Calf', 'Groin', 'Lower
    Back', 'Shoulder', 'Knee', 'Ankle', 'Hip Flexor', 'Achilles', '
    None']

# Injury types
injury_types = ['Strain', 'Sprain', 'Tear', 'Inflammation', '
    Tendinitis', 'Fracture', 'Contusion', 'None']

# Recovery status
recovery_status = ['Full Recovery', 'Progressing', 'Setback', '
    Chronic', 'No Injury']

# Generate base date
```

```
start_date = datetime(2024, 1, 1)

# Initialize lists to store data
data = {
    'session_id': [],
    'athlete_id': [],
    'date': [],
    'session_type': [],
    'duration_minutes': [],
    'intensity_level': [],
    'avg_heart_rate': [],
    'max_heart_rate': [],
    'distance_km': [],
    'sprint_count': [],
    'acute_load': [],
    'chronic_load': [],
    'acwr': [],  # Acute:Chronic Workload Ratio
    'training_monotony': [],
    'training_strain': [],
    'rpe': [],  # Rating of Perceived Exertion (1-10)
    'sleep_hours': [],
    'sleep_quality': [],
    'fatigue_score': [],
    'muscle_soreness': [],
    'injury_history_count': [],
    'days_since_last_injury': [],
    'current_injury': [],
    'injury_type': [],
    'muscle_group_affected': [],
    'injury_severity': [],  # 1-10 scale
    'recovery_days': [],
    'recovery_progress': [],
    'medical_clearance': [],
    'predicted_risk_score': [],
    'high_risk_flag': []
}

# Generate data
session_count = 0
for i in range(n_entries):
    session_count += 1

    # Basic session info
    data['session_id'].append(f"SES_{str(session_count).zfill(5)}")
    athlete_id = random.choice(athlete_ids)  # Random athlete
       selection for multiple sessions
    data['athlete_id'].append(athlete_id)
```

```
# Date (distributed over 6 months)
days_offset = random.randint(0, 180)
session_date = start_date + timedelta(days=days_offset)
data['date'].append(session_date.strftime('%Y-%m-%d'))

# Session details
session_type = random.choice(session_types)
data['session_type'].append(session_type)

# Duration varies by session type
if session_type == 'Recovery':
    duration = np.random.normal(45, 10)
elif session_type == 'Match/Competition':
    duration = np.random.normal(90, 15)
elif session_type == 'HIIT':
    duration = np.random.normal(60, 10)
else:
    duration = np.random.normal(75, 15)
data['duration_minutes'].append(max(30, min(120, duration)))

# Intensity (1-10 scale)
if session_type == 'Recovery':
    intensity = np.random.uniform(2, 5)
elif session_type in ['Match/Competition', 'HIIT']:
    intensity = np.random.uniform(7, 10)
else:
    intensity = np.random.uniform(5, 8)
data['intensity_level'].append(round(intensity, 1))

# Heart rate data
base_hr = random.randint(60, 80)
if session_type == 'Recovery':
    avg_hr = base_hr + random.randint(20, 40)
    max_hr = avg_hr + random.randint(15, 30)
elif session_type in ['Match/Competition', 'HIIT', 'Speed']:
    avg_hr = base_hr + random.randint(80, 100)
    max_hr = avg_hr + random.randint(20, 40)
else:
    avg_hr = base_hr + random.randint(50, 80)
    max_hr = avg_hr + random.randint(15, 35)

data['avg_heart_rate'].append(min(195, avg_hr))
data['max_heart_rate'].append(min(210, max_hr))

# Distance (varies by session type)
if session_type in ['Endurance', 'Match/Competition']:
```

```
            distance = np.random.normal(8, 2)
        elif session_type in ['Speed', 'HIIT']:
            distance = np.random.normal(5, 1.5)
        elif session_type == 'Recovery':
            distance = np.random.normal(3, 1)
        else:
            distance = np.random.normal(4, 1.5)
        data['distance_km'].append(max(0.5, round(distance, 2)))

        # Sprint count
        if session_type in ['Speed', 'Match/Competition', 'HIIT']:
            sprints = random.randint(10, 40)
        elif session_type == 'Recovery':
            sprints = 0
        else:
            sprints = random.randint(3, 15)
        data['sprint_count'].append(sprints)

        # Load metrics
        acute_load = duration * intensity * random.uniform(0.8, 1.2)
        data['acute_load'].append(round(acute_load, 2))

        chronic_load = acute_load * random.uniform(0.7, 1.1)
        data['chronic_load'].append(round(chronic_load, 2))

        acwr = acute_load / chronic_load if chronic_load > 0 else 1.0
        data['acwr'].append(round(acwr, 2))

        # Training monotony and strain
        monotony = random.uniform(1.0, 3.5)
        data['training_monotony'].append(round(monotony, 2))
        data['training_strain'].append(round(acute_load * monotony, 2))

        # RPE (Rating of Perceived Exertion)
        rpe = intensity + random.uniform(-1, 1)
        data['rpe'].append(max(1, min(10, round(rpe, 1))))

        # Sleep and recovery metrics
        sleep_hrs = np.random.normal(7.5, 1.2)
        data['sleep_hours'].append(max(4, min(10, round(sleep_hrs, 1))))

        sleep_qual = random.randint(1, 10)
        data['sleep_quality'].append(sleep_qual)

        # Fatigue score (inverse relationship with sleep quality)
        fatigue = 10 - (sleep_qual * 0.6 + (sleep_hrs / 10) * 4) +
           random.uniform(-1, 1)
```

```
data['fatigue_score'].append(max(1, min(10, round(fatigue, 1))))

# Muscle soreness
soreness = intensity * 0.8 + random.uniform(-2, 2)
data['muscle_soreness'].append(max(0, min(10, round(soreness, 1)
   )))

# Injury history
injury_hist_count = random.choices([0, 1, 2, 3, 4, 5], weights
   =[30, 30, 20, 10, 7, 3])[0]
data['injury_history_count'].append(injury_hist_count)

# Days since last injury
if injury_hist_count > 0:
    days_since = random.randint(0, 365)
else:
    days_since = 999  # No previous injury
data['days_since_last_injury'].append(days_since)

# Current injury status (higher risk with recent injuries, high
   load, low recovery)
risk_factors = 0
if days_since < 30:
    risk_factors += 3
elif days_since < 90:
    risk_factors += 2
elif days_since < 180:
    risk_factors += 1

if acwr > 1.5:
    risk_factors += 2
elif acwr > 1.3:
    risk_factors += 1

if fatigue > 7:
    risk_factors += 2

if sleep_hrs < 6:
    risk_factors += 1

if monotony > 2.5:
    risk_factors += 1

# Determine if current injury exists
injury_prob = min(0.9, risk_factors * 0.08)
has_injury = random.random() < injury_prob
```

```
data['current_injury'].append('Yes' if has_injury else 'No')

if has_injury:
    injury_type = random.choice([it for it in injury_types if it
        != 'None'])
    muscle_group = random.choice([mg for mg in muscle_groups if
       mg != 'None'])
    severity = random.randint(3, 10)
    recovery_days_needed = severity * random.randint(3, 7)
    recovery_prog = random.choice(['Progressing', 'Setback', '
       Chronic']) if severity > 5 else random.choice(['
       Progressing', 'Full Recovery'])
    medical_clear = 'No' if recovery_prog in ['Setback', '
       Chronic'] else random.choice(['Yes', 'No'])
else:
    injury_type = 'None'
    muscle_group = 'None'
    severity = 0
    recovery_days_needed = 0
    recovery_prog = 'No Injury'
    medical_clear = 'Yes'

data['injury_type'].append(injury_type)
data['muscle_group_affected'].append(muscle_group)
data['injury_severity'].append(severity)
data['recovery_days'].append(recovery_days_needed)
data['recovery_progress'].append(recovery_prog)
data['medical_clearance'].append(medical_clear)

# Predicted risk score (0-100)
risk_score = (
    (acwr - 0.8) * 20 +
    fatigue * 4 +
    (10 - sleep_hrs) * 5 +
    monotony * 8 +
    (injury_hist_count * 5) +
    (max(0, 90 - days_since) / 90 * 20) +
    soreness * 2 +
    random.uniform(-5, 5)
)
risk_score = max(0, min(100, risk_score))
data['predicted_risk_score'].append(round(risk_score, 2))

# High risk flag
data['high_risk_flag'].append('High Risk' if risk_score > 65
   else 'Low Risk')
```

```
# Create DataFrame
df = pd.DataFrame(data)

# Save to CSV
df.to_csv('athlete_load_management_dataset.csv', index=False)

print(f"Dataset created successfully with {len(df)} entries!")
print(f"\nDataset shape: {df.shape}")
print(f"\nFirst few rows:")
print(df.head())
print(f"\nDataset info:")
print(df.info())
print(f"\nSummary statistics:")
print(df.describe())
print(f"\nInjury distribution:")
print(df['current_injury'].value_counts())
print(f"\nRisk distribution:")
print(df['high_risk_flag'].value_counts())
```

2 Tools and Methodologies

We are using PostgreSQL for database management, Python 3.9 for data processing and analysis, and Scikit-learn for machine learning models. PostgreSQL handles our relational data with proper constraints and foreign keys. Python with Pandas manages data manipulation and feature engineering. Scikit-learn provides Random Forest implementation for injury risk prediction. We chose these tools because they are industry-standard, well-documented, and covered in our coursework.

3 Preliminary Timeline

- **Week 1 (Completed):** Database design and synthetic data generation
- **Week 2 (Current):** Implement database schemas in PostgreSQL
- **Week 3:** Complete all 8 SQL analytical reports
- **Week 4:** Feature engineering and data preparation for ML
- **Week 5:** Train and evaluate Random Forest model
- **Week 6:** Final integration, testing, and documentation
- **Week 7:** Prepare presentation and submit final deliverables

4 Team Member Contributions

Harsha Prakash (Health Data Science) designed the medical database schema including injuries, recovery metrics, and risk predictions tables. He generated synthetic medical data with realistic injury patterns and recovery timelines. He will create 4 SQL reports focused on injury analysis and implement the injury risk prediction model.

Samuel Greeman (Sports Data Science) designed the performance database schema including training sessions and load calculations tables. He generated synthetic training data with realistic workload patterns. He will create 4 SQL reports focused on performance metrics and implement ACWR calculations.

Shared Work: Both members collaborated on the overall database design, ensuring proper integration through athlete IDs and consistent date ranges. We jointly developed the data generation framework and will work together on model evaluation and final documentation.

5 Progress and Next Steps

Progress to Date: We have completed the database design with ER diagrams and successfully generated 5,000+ synthetic records across medical and performance domains. The data generation scripts are documented and reproducible using fixed random seeds.

Next Steps:

- Implement database schemas in PostgreSQL (this week)
- Begin writing SQL queries for analytical reports
- Start feature engineering for machine learning

Challenges: We are behind our original schedule but have adjusted our scope to ensure completion. We've removed complex features like real-time streaming and dashboards to focus on core requirements: database implementation, SQL reports, and basic ML model.