Streamlined Workplan

Project Overview

Process 1,000+ motorater Excel files containing time series gait data from 3 treatment groups (vehicle/Baseline, LoDose, HiDose) across 7 timepoints (baseline/0, 7dpi-42dpi) with 5 runs per animal per timepoint. Goal: extract comprehensive features, identify injury-responsive features, and develop composite recovery scores.

Data Structure

- **Files**: `[DateCode]_[AnimalID]_[XDPI]_[RunNumber]_out.xlsx` (stored in Box, downloaded locally)
- **Sheet 2**: 46 precomputed features (rows = hundredths of seconds)
- **Sample size**: ~13 animals per treatment group
- **Animal Data Key**: Excel with `FileName`, `AnimalID`, `Treatment Groups` (Baseline, LoDose, HiDose), `Timepoint (days)` (0, 7, 14, 21, 28, 35, 42), `Trial`, `Condition`
- **Existing baseline**: Summary stats and cycle-based features from `integrated_data.R`

Workflow Architecture

Phase 1: Data Loading & Feature Extraction

File: `scripts/01_extract_features.py`

Input: Data directory (all Excel files) + Animal Data Key file

Process:

1. Load Animal Data Key to map `animal_id` → `treatment_group`
2. Load all Excel files, parse metadata from filenames
3. Extract Sheet 2 (features) for each file
4. Apply `trim_flat_edges()` to remove flatline sections - reuse logic from `integrated_data.R`
5. Feature Extraction:

- **Method A (current):** Summary stats (mean, sd, min, max, range, IQR, CV) + cycle-based features (avg_cycle_length, avg_cycle_max, avg_cycle_min, regularity_score) - reuse logic from `integrated_data.R` and `matthew_pipeline.R`
- **5.1. Python libraries for time-series feature extraction:**
  - **Method B**: TSFresh library to extract features for each of 46 original features using `tsfresh.extract_features()` with default parameters (very comprehensive, around 800 features per original time-series ~ total = 46 x 800)
  - **Method C**: Catch22: other Python feature-based transformer that extracts 22 distinct time-series features per orginal time-series (reduces redunancy, maximizes interpretability ~ total = 46 x 22)
- **5.2. Additional features:**
  - **Optionally add custom features after primary methods (A-C) for left-right coordination**: peak averages, bilateral synch (cross-correlation between left/right paired features), bilateral asymmetry metrics (absolute/percent difference)

6. **Validation**: Plot sample signals with detected peaks/cycles to validate extraction (visual check only)

Output: `outputs/extracted_features.parquet` - one row per original file with:

- Metadata: `animal_id`, `treatment_group`, `timepoint`, `run_number`, `condition`
- All extracted features from selected method

Key Functions:

- `load_and_extract_features(data_dir, key_file)` : Main pipeline
- `extract_baseline_features(time_series)` : Summary stats + cycle features
- `extract_tsfresh_features(time_series)` : TSFresh extraction
- `add_custom_features(time_series)` : Peak averages, bilateral synch, asymmetry
- `validate_extraction(sample_files)` : Plot signals with peaks for validation

Phase 2: Feature Selection

File: `scripts/02_feature_selection.py`

Goal: Identify features most affected by injury (baseline vs 7dpi) that may show recovery patterns.

2.1: TSFresh Built-in Selection

- TSFresh will gives us around 46 x 800 features. We need to use `tsfresh.select_features()` as a first step to removes irrelevant/redundant features using statistical tests,
- Use target = baseline vs 7dpi binary classification (features affected by injury)

- Alternatively, also use target = 7dpi vs 43dpi binary classification (features that recovery), and take both injury and recovery sensitive features

2.2: Additional Selection Methods (after TSFresh selection)

Method A: Classification-Based Feature Selection

- Train Random Forest classifier on baseline vs 7dpi binary classification
- Extract feature importances and select top N features (e.g., top 50-100)
- Alternative: L1-regularized logistic regression (Lasso) for feature selection (if want smaller feature set)
- More suitable than Method B because Cohen's d tests each feature independently, while RF captures feature interactions. RF also down-weights redundant features while Cohen's d can have redundant features.

Method B: Cohen's d Effect Size - logic in `LDA.R`

- Compute Cohen's d for baseline vs 7dpi: `d_injury`
- Compute Cohen's d for baseline vs 42dpi: `d_recovery`
- Select features with large injury effect: `|d_injury| > threshold` (use data-driven threshold, e.g., top 25% of effect sizes)
- Optionally rank by recovery pattern: `|d_recovery| < |d_injury|` (shows improvement)

Method C: Features selected by both A and B

Output: `outputs/selected_features.csv` with feature names and selection method (A - C)

Team Split: 1 person implements method A, 1 person implements method B

Phase 3: Baseline Normalization

File: `scripts/03_baseline_normalization.py`

Goal: Normalize features by each animal's baseline, maintaining interpretability.

Approach: **Percent Change from Baseline**

- Formula: `(value - baseline_mean) / baseline_mean * 100`
- Interpretation: % change from baseline (intuitive for recovery)

Validation:

- Check for no extreme explosions `(max(abs(normalized_value)) > 1000%)` and `normalized_value_at_baseline ≈ 0%`

Aggregation:

- After normalization, aggregate across runs: mean per (animal, timepoint)
- Output: `outputs/normalized_features_per_animal.parquet`

Note: Start with percent change. If problem, consider z-score by baseline as alternative, but evaluate based on interpretability of results.

Key Functions:

- `normalize_by_baseline(features_df)` : Apply percent change normalization
- `validate_normalization(normalized_df)` : Check distributions and trajectories
- `aggregate_across_runs(normalized_df)` : Mean per animal×timepoint

Phase 4: Visualization & Dimensionality Reduction

File: `scripts/04_visualization.py`

PCA - reuse `PCA.R` logic:

- Use selected features from Phase 2
- Apply Yeo-Johnson transformation + scaling
- Plot PC1 vs PC2, PC1 vs PC3 colored by treatment group and timepoint
- Plot recovery trajectories: PC1 or LD1 over timepoints

LDA - reuse `LDA.R` logic:

- Use selected features from Phase 2
- PCA dimensionality reduction before LDA
- Plot LD1 scores over timepoints

Connectivity Visualization:

- Correlation network between top features (nodes = features, edges = correlations > threshold)

Outputs: `outputs/pca_results.pdf`, `outputs/lda_results.pdf`, `outputs/connectivity_network.pdf`

Phase 5: Composite Recovery Score Development

File: `scripts/05_recovery_scores.py`

Method A: LD1 Score (existing)

Method B: Composite Walking Score (existing from Ali Lab)

- Use PC1 and PC2 from PCA using selected features (alternatively experiment with more PCs)
- Define centroids: baseline (0) and injured (7dpi)
- Project animals onto line between centroids
- CWS = normalized distance along projection (0 = injured, 1 = healthy)

Method C: Mahalanobis Distance

- Compute distance from each animal to healthy baseline distribution
- Smaller distance = better recovery
- **Note:** accounts for feature correlations, handles different variances, interpretable

Method D: Supervised Regression Residuals

- Train regression model predicting timepoint from features
- Residual = actual - predicted (smaller residual = better recovery trajectory)
- **Note:** can overfit (limited data) and less interpretible

Evaluation:

- Recovery curve (including predicted with LOO, see logic in `LDA.R`)
- Treatment group differences (HiDose > LoDose > Baseline expected)

Output: `outputs/recovery_scores.csv`

Phase 6: Feature Interpretation

File: `scripts/06_feature_interpretation.py`

- Correlate individual features with recovery scores
- ANOVA across treatment groups (at each timepoint)
- Post-hoc tests (Tukey HSD) to identify significant group differences
- Visualize: overlay recovery scores with top contributing features

Output: `outputs/feature_drivers.csv`, `outputs/interpretation_plots.pdf`

Workflow Organization

Team 1 - Feature Extraction (Phase 1)

- **Nikhita**: Steps 1-4 and Methods A (baseline) & B
- **Person A**:

    - **Method C**: just existing library
    - **Section 5.2.**: Main work, custom coordination features
- **Inputs:** Raw Excel files + Animal Key
- **Outputs:** `outputs/extracted_features.parquet`

Team 2 - Feature Selection (Phase 2)

- **Nikhita**: 2.1
- **Persons B & C**:
    - **Method A:** Random Forest / L1 Logistic feature importance
    - **Method B:** Cohen's *d* effect size
- **Inputs:** For now, existing baseline features (Nikhita will send integrated csv)
- **Outputs:** `outputs/selected_features.csv`
- **Parallel:** while Team 1 expands features, Team 2 can prototype selection on existing baseline features, then re-run on full set once ready.

Team 3 - Recovery Score (Phase 5)

- **Nikhita**: Methods A & B
- **Persons D & E**:
    - **Method C:** Mahalanobis Distance
    - **Method D:** Supervised Regression Residuals
- **Inputs:** For now, existing baseline features + Cohen's d selection (Nikhita will send integrated csv with selected baseline features)
- **Outputs:** `outputs/recovery_score.csv`
- **Parallel:** Team 3 will prototype recovery scores for any input feature set, and we will later update with best feature extraction + selection set.

All Teams (Phase 6)

- A normalization and recovery scoring pipelines are ready:
    - Correlate individual features with recovery scores
    - Run ANOVA + post-hoc tests
    - Visualize top contributing features
- **Outputs:**
    - `outputs/feature_drivers.csv`
    - `outputs/interpretation_plots.pdf`

File Structure

```
scripts/
├── 01_extract_features.py          # Combined data loading + feature
extraction
├── 02_feature_selection.py         # TSFresh, RF/lasso, Cohen's d
├── 03_baseline_normalization.py    # Normalize for each animal's baseline
├── 04_visualization.py             # PCA, LDA, connectivity plots
├── 05_recovery_scores.py           # LD1, CWS, Mahalanobis, Regression
residuals
└── 06_feature_interpretation.py    # Driver analysis

outputs/
├── extracted_features.parquet       # One row per file, all features
├── selected_features.csv            # Selected feature list
├── normalized_features_per_animal.parquet
├── recovery_scores.csv
└── [various plots]
```

Modular Pipeline:

Each run of the pipeline:

1. Loads the data
2. Applies one of the three **feature extraction** methods
3. Applies one of the two **feature selection** methods
4. Normalizes by baseline
5. Computes one of the three **recovery scores**
6. Logs outputs and metrics for comparison

Each run saves a config log (`outputs/run_log.json`) summarizing which combination (feature extraction, selection, score) was used — for later comparison.