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 containing all Motorater 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, and standardize column names.
3. Extract Sheet 2 (features) for each file
4. Apply `trim_flat_edges()` to remove flatline sections - reused logic from `integrated_data.R`

5. Feature Extraction:
    - **Method A (current)**: Summary statistics + cycle-based features:
        - **Summary stats:** mean, standard deviation, min, max, range, IQR, coefficient of variation (CV)
        - **Cycle features:** average cycle length, average cycle max, average cycle min, regularity score
        - Peak/trough detection includes filtering out unusually small peaks or unusually high troughs
    - **Method B (planned)**: TSFresh library for automated, comprehensive feature extraction (~800 features per original time series)
    - **Method C (planned)**: Catch22 library for interpretable time-series features (22 features per original time series)
    - **Additional optional features (any method)**:
        - Peak averages
        - Bilateral synchronization (cross-correlation of left/right paired features)
        - Bilateral asymmetry metrics (absolute/percent difference)
6. **Validation (`--visualize [num_files]` mode)**:
    - Randomly select files (or specify number) to plot signals with detected peaks/cycles
    - Saves plots in `outputs/cycle_plots`
    - Also extracts the same features from the visualized files and saves to `visualized_features_methodA.csv`

Output:

- Normal extraction: `outputs/extracted_features_method{A/B/C}.csv` or `outputs/extracted_features.parquet`
- Visualization mode (`--visualize`): plots + CSV in `outputs/cycle_plots/`

Key Function	Description
`trim_flat_edges(x)`	Removes flatline sections from a time series.
`extract_features_methodA(x, feat_name, return_peak_info=False)`	Computes summary statistics and cycle-based features.
`extract_features_methodB(x, feat_name)`	Placeholder, will compute TSFresh features.
`extract_features_methodC(x, feat_name)`	Placeholder, will compute Catch22 features.
`extract_file_features(df, method="A")`	Extracts all feature columns from a dataframe.
`extract_dir_features(data_dir, key_file, method="A/B/C")`	Main pipeline to process a directory of files using the chosen extraction method.

Key Function	Description
`visualize file cycles(file_path, output_dir=None)`	Plots signals with peaks/troughs for validation.

Notes:

- Cycle detection uses `scipy.signal.find_peaks` with prominence and outlier filtering
- Supports optional cycle feature visualization via `--visualize [num_files]`
- Designed to easily switch between Method A, B, or C depending on desired feature set

Phase 2: Data Transformation & Feature Selection

File: `scripts/02_transform_select_features.py`

Goal: Clean, normalize, and select features most affected by injury (baseline vs 7dpi), potentially showing recovery trends.

Input: `outputs/extracted_features_methodA.csv` (from Phase 1)

Steps:

2.1 Data Transformation (Preprocessing)

Before feature selection, the dataset is transformed to ensure consistent scaling and comparability across animals.

1. **Mean Imputation**
    - Missing values are imputed using the mean of other runs for the same `(animal_id, treatment_group, timepoint_days)` group.
    - Ensures each animal's time series remains internally consistent.
2. **Baseline Normalization**
    - Each feature is divided by its animal's baseline mean (`timepoint_days == 0`).
    - Preserves interpretability: normalized values > 1 indicate increase from baseline, < 1 indicate decrease.
    - Extreme values (|x| > 1000) are imputed using the mean of non-extreme runs for that same animal and timepoint.
3. **Skew Adjustment & Scaling**
    - Applies **Yeo-Johnson transformation** to features with |skew| > 1.
    - Then standardizes all features to **mean = 0, std = 1** using `StandardScaler` .
    - Produces consistent feature ranges for selection models.

2.2: TSFresh Built-in Selection (if TSFresh used to extract)

- 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

2.3: Feature Selection Methods

After preprocessing and preliminary TSFresh selection, one of three selection methods is applied.

- **Method A: Classification-Based Selection (planned)**
    - Train a **Random Forest classifier** (baseline vs 7dpi).
    - Select top N features based on feature importance.
    - Captures feature interactions and down-weights redundancy.
    - Alternative: **L1-regularized logistic regression (LASSO)** for sparser selection.
- **Method B: Cohen's d Effect Size (current)**
    - Compute Cohen's *d* for each feature:
    - `d_injury` : baseline → 7dpi
    - `d_recovery` : 7dpi → 42dpi
    - Select features with strong injury effect (`|d_injury| ≥ 0.8`) and moderate recovery change (`|d_recovery| ≥ 0.2`).
    - Visualize top 50 features by injury effect size in a heatmap.
- **Method C: Intersection (planned)**
    - Retains only features selected by both A and B, capturing robust injury-recovery sensitivity.

Outputs:

- `outputs/transformed_selected_features_method{A/B/C}.csv` → dataset with selected features + metadata
- `outputs/feature_info_method{A/B/C}.csv` → summary of feature statistics (e.g., effect sizes)
- `outputs/methodB_effectsize_heatmap.png` → visualization of top effect size features

Key Function	Description
`mean_impute_by_group(df)`	Imputes missing values using group means
`normalize_by_animal_baseline(df)`	Normalizes by each animal's baseline mean
`skew_and_scale(df)`	Adjusts for skew and standardizes all features

Key Function	Description
`select_features_methodB(df, threshold)`	Computes Cohen's *d* and selects features
`preprocess_and_select(input_file, method)`	Main unified pipeline
`select_features_methodA/C(df)`	Placeholders for future selection methods

Phase 3: Visualization & Dimensionality Reduction

File: `scripts/03_PCA_and_LDA.py`

Goal: Visualize recovery trajectories and group separations using PCA and LDA applied to the transformed & selected features from Phase 2.

Input: `outputs/transformed_selected_features_method{A/B/C}.csv` (cleaned, baseline-normalized, skew-adjusted, scaled, and selected features with metadata)

Steps:

1. **Load input**
   - Read transformed & selected features CSV produced by Phase 2. Metadata columns are preserved.
2. **PCA** (`run_pca_analysis`)
   - Fit PCA on selected features (optionally specify `n_components`).
   - Produce PC scores with metadata and save PC recovery curves and loadings plots.
3. **LDA** (`run_lda_analysis`)
   - Optionally reduce dimensionality using PCA (provided PCA model or compute internally).
   - Train LDA to separate:
     - **Injury axis:** baseline (0 dpi) vs injury (7 dpi)
     - **Treatment axis:** treatment groups at 42 dpi
   - Combine axes with a weighting factor (`treatment_weight`) to produce LD1 scores.
   - Save LD1 results and recovery/feature-loading plots.
4. **Save outputs**
   - PCA results folder: `outputs/PCA_results/`
   - LDA results folder: `outputs/LDA_results/`

Outputs (examples)

- `outputs/PCA_results/PC_scores.csv`
- `outputs/PCA_results/PC_loadings.png`
- `outputs/PCA_results/PC1_recovery_curves.png`
- `outputs/LDA_results/LD_scores.csv`

- `outputs/LDA_results/LD1_scatter_allruns.png`
- `outputs/LDA_results/LD1_animal_trajectories.png`
- `outputs/LDA_results/LD1_loadings.png`

Key Function	Description
`run pca analysis(df, n components=None, output_folder='outputs')`	Performs PCA on feature columns, saves PC scores and PCA visualizations (PC recovery curves, PC loadings). Returns `(pca_model, pc_df, explained_var, feature_cols)`.
`run lda analysis(df, output folder='outputs', pca variance threshold=0.8, treatment weight=0.3, pca_model=None)`	Performs LDA to separate injury (0 vs 7 days) and treatment groups (at 42 days). Optionally uses PCA components for dimensionality reduction. Saves LD1 scores and visualizations. Returns `(lda_model, ld_df, feature_cols, n_pca_components)`.

Notes:

- LDA can use PCA components if provided, but can also operate directly on features.
- LD1 scores combine injury and treatment axes for visualization of recovery and treatment effects.

Phase 4: Composite Recovery Score Development

File: `scripts/05_recovery_score.py`

Input:

- `outputs/PCA_results/PCA_scores.csv` (for Methods A–C)
- `outputs/transformed_selected_features_method{A/B/C}.csv` (for Method D)

Method A: LD1 Score (existing, in PCA script for now)

- Uses LD1 scores from LDA (can optionally reduce dimensionality with PCA components)
- Reflects combined injury and treatment axes

Method B: Composite Walking Score (CWS, existing from Ali Lab MATLAB script in Slack)

- Uses PC1 and PC2 from PCA using selected features (more PCs optional)
- Define centroids: baseline (0 dpi) and injured (7 dpi)
- Project animals onto line between centroids
- CWS = normalized distance along projection (0 = injured, 1 = healthy)
- Interpretation possible via PC loadings

Method C: Mahalanobis Distance

- Compute distance from each animal to healthy baseline distribution in PC space

- Smaller distance = better recovery
- Accounts for feature correlations, handles different variances
- Interpretation possible via PC loadings

Method D: Supervised Regression Residuals

- Train regression model predicting timepoint from transformed & selected features
- Residual = actual - predicted (smaller residual = better recovery trajectory)
- More directly interpretable per feature
- Can overfit with limited data

Evaluation:

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

Output: `outputs/recovery_scores.csv`

Phase 5: Feature Interpretation

File: `scripts/05_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, 6 and Step 5 Methods A (baseline) & B
- **Person A**:
  - **Step 5 Method C**: just existing library
  - **Step 5 Additional Features**: Main work, custom coordination features
- **Input**: Data directory containing all Motorater Excel files + Animal Data Key file
- **Output**:
  - Normal extraction: `outputs/extracted_features_method{A/B/C}.csv` or `outputs/extracted_features.parquet`

- Visualization mode (`--visualize`): plots + CSV in `outputs/cycle_plots/`

Team 2 - Feature Selection (Phase 2)

- **Nikhita**: 2.1, 2.2, and 2.3 Method B (Cohen's *d* effect size)
- **Persons B**: 2.3 Method B Random Forest / L1 Logistic feature importance
- **Inputs:** `outputs/extracted_features_methodA.csv`
- **Outputs:**
  - `outputs/transformed_selected_features_method{A/B/C}.csv` → dataset with selected features + metadata
  - `outputs/feature_info_method{A/B/C}.csv` → summary of feature statistics (e.g., effect sizes)
  - `outputs/methodB_effectsize_heatmap.png` → visualization of top effect size features
- **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 4)

- **Nikhita**: Method A, will try B as well
- **Persons C, D, & E**:
  - **Method B:** CWS (reference MATLAB script provided in Slack)
  - **Method C:** Mahalanobis Distance
  - **Method D:** Supervised Regression Residuals
- **Inputs:**
  - Methods A-C: `outputs/PCA_results/PCA_scores.csv`
  - Method D: `outputs/transformed_selected_features_method{A/B/C}.csv`
- **Outputs:** `outputs/recovery_score.csv`
- **Parallel:** Team 3 will prototype recovery scores for current input feature/PC set, then update.

All Teams (Phase 5)

- 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
├── 02_transform_select_features.py
├── 03_PCA_and_LDA.py
├── 04_recovery_scores.py
└── 05_feature_interpretation.py

outputs/
├── extracted_features_method{A/B/C}.csv or .parquet            # From
Phase 1
├── cycle_plots/    # From Phase 1: If run script with visualize mode
├── transformed_selected_features_method{A/B/C}.csv  # From Phase 2
├── feature_info_method{A/B/C}.csv      # From Phase 2
├── methodB_effectsize_heatmap.png      # From Phase 2
├── LDA_results/                        # From Phase 3
├── PCA_results/                        # From Phase 3
├── recovery_scores.csv                 # From Phase 4
└── [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 three **feature selection** methods
4. Normalizes by baseline
5. Computes one of the four **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.