NFL Run/Pass Prediction Project Summary

Scope
This project builds a complete machine learning pipeline to predict whether an NFL offensive play will be a run or a pass *before the snap*. The original workflow started as a Kaggle notebook and has been refactored into a clean, modular Python package called *nfl_run_pass*. The package:
• loads and filters NFL play-by-play data
• engineers football-specific pre-snap features
• preprocesses and scales the data
• trains and tunes a logistic regression model
• evaluates performance
• saves model artifacts for reuse and deployment

What We Are Predicting
The target variable is *is_pass*, a binary label where:
• 1 = the play is a pass
• 0 = the play is a run

The model only uses pre-snap information to avoid leakage: down, distance, field position, score differential, time remaining, formation indicators (shotgun, no-huddle), and basic game context.

Modules in the nfl_run_pass Library

1. config.py
Central configuration for the entire project. It defines:
• data paths (raw CSV, artifacts directory)
• target and label settings (e.g., play_type column, is_pass)
• feature lists (which base and engineered features to use)
• train/test split parameters (test size, random seed, stratification)
• logistic regression hyperparameters and GridSearchCV settings
By centralizing these values, all other modules read from one source of truth instead of hard-coding constants.

2. data_loading.py
Handles the transition from raw CSV to a modeling-ready dataframe. It:
• loads the raw NFL play-by-play CSV
• filters to a specific season (e.g., 2023)
• keeps only run and pass plays based on the play_type column
• creates the binary target column *is_pass*
• exposes a convenience function *load_and_prepare_run_pass_data* that performs all steps in one call

3. features.py
Responsible for feature engineering and building the final feature matrix. It:
• checks for required raw columns (yardline_100, goal_to_go, ydstogo, qtr, score_differential, etc.)
• creates engineered pre-snap features, including:
  – is_red_zone, is_goal_to_go
  – short/medium/long_ydstogo buckets
  – shotgun, no_huddle
  – score_differential and is_trailing / is_tied / is_leading
  – is_fourth_qtr, late_half
  – is_home_offense
• constructs (X, y) via *build_feature_matrix*, using the configured feature list from config.py

4. preprocessing.py
Implements the data preparation steps before modeling. It:
• builds the feature matrix (X, y) from the prepared dataframe
• handles missing values (numeric median imputation, then fills remaining NaNs)
• splits into train/test sets with stratification according to configuration
• scales features using StandardScaler (fit on train, applied to test)
• exposes *prepare_train_test_data*, which returns X_train, X_test, y_train, y_test, scaler, and feature_cols

5. models.py
Encapsulates the model training logic, focusing on logistic regression. It:
• constructs a base LogisticRegression model (with configurable max_iter)
• reads the hyperparameter grid from config.py
• trains either:
  – a simple logistic regression model, or
  – a tuned model via GridSearchCV over C and class_weight
• returns the fitted model plus metadata (best parameters, best cross-validated score, and full CV results)

6. evaluation.py
Provides a consistent way to evaluate the trained model. It:
• computes metrics on both train and test sets:
  – accuracy
  – precision
  – recall
  – F1 score
  – ROC-AUC when probability scores are available
• builds a 2x2 confusion matrix on the test set (run vs pass)
• returns a structured dictionary summarizing train and test performance

7. pipeline.py
Orchestrates the entire ML workflow end-to-end. It:
• calls data_loading to load and prepare the season's run/pass dataset
• calls preprocessing to build features, split, and scale
• calls models.train_log_reg_model to fit (and optionally tune) the classifier
• calls evaluation.evaluate_run_pass_model to compute metrics
• saves the trained model, scaler, and feature column names to disk as artifacts
• returns a single results object containing the model, metrics, artifact paths, and a snapshot of the config
This is the main entry point when we want to retrain the model or rerun the full experiment in one step.

8. tuning.py
Adds standalone hyperparameter tuning utilities separate from the main pipeline. It:
• runs GridSearchCV over logistic regression hyperparameters using the project config
• allows more focused experiments on tuning without changing the main training pipeline
• provides comparison utilities (e.g., baseline vs tuned model performance on the test set)

Overall
Together, these modules form a reusable, team-friendly Python library for NFL run/pass prediction. Teammates can call the high-level pipeline for full training and evaluation, or drop into individual modules for deeper experimentation with data, features, models, and tuning strategies.