WiDS Global Datathon 2026

A best-practice Kaggle agenda from exploratory analysis to stacked ensembles

Competition focus: wildfire threat to evacuation zones within 12h, 24h, 48h, and 72h using only the first 5 hours of signal.

EDA • validation • model testing • tuning • calibration • stacking • learning from experts

Workflow

- Audit data
- Build CV
- Test models
- Tune and stack

Executive summary

Objective
Build a robust local workflow that mirrors the competition metric, then improve through calibrated modeling and disciplined ensembling.

Winning pattern
Start simple, verify validation, expand model families, tune only after leakage checks, and ensemble only when OOF gains persist.

End state
A reproducible submission factory with stable CV, error slices, calibrated outputs, and a final stack.

- Treat the public leaderboard as a weak signal; trust repeated out-of-fold gains first.
- Favor model diversity over brute-force tuning on a single learner.
- Keep every transformation train-fold safe and reproducible.
- Borrow ideas from public notebooks, then revalidate them locally.

Core rule
Do not move to heavy tuning or complex stacking until the local metric and leakage checks are trustworthy.

Competition brief

- Predict the probability that a wildfire threatens an evacuation zone within 12h, 24h, 48h, and 72h.
- Use only the first five hours after ignition / initial perimeter observation.
- Output calibrated, monotone probabilities across all four horizons.
- Support emergency response: prioritization plus confidence.

Why it matters

This is a survival-style forecasting problem, not a plain one-shot binary classification task.

Operational meaning

The best model must rank urgent fires well and also deliver probabilities responders can trust.

Implication
Your entire pipeline should optimize both discrimination and calibration; a single “high AUC” mindset is too narrow for this Kaggle.

What makes this competition unusual

Right-censoring

Many fires never hit the target zone during follow-up. The model must learn from incomplete outcomes, not only observed hits.

Multi-horizon output

Each row produces four related probabilities. Outputs must be monotone and coherent with a survival curve.

Hybrid scoring

Public summaries describe a hybrid metric that rewards both ranking quality and calibration quality.

Early-signal regime

You only get the first five hours of signal, so feature engineering must squeeze information from initial geometry, motion, and distance.

Response
Design validation and post-processing as first-class citizens of the project.

Agenda to tackle the competition

Workstreams 1–5

- Reproducible setup
- Data audit and schema checks
- EDA and leakage hunt
- Local metric and fold design
- Baseline model ladder

Workstreams 6–10

- Feature engineering
- Hyperparameter tuning
- Bagging / blending / stacking
- Learning from experts
- Submission strategy and final checklist

Roadmap: a disciplined competition cadence

Sprint 0

Setup repo, schema checks, and a first valid submission.

Sprint 1

Finish EDA, target interpretation, drift review, and leakage memo.

Sprint 2

Run baseline ladder across all fold-safe model families.

Sprint 3

Engineer feature families and run clean ablations.

Sprint 4

Tune only promoted models and add bagging + calibration.

Sprint 5

Blend / stack, validate robustness, and finalize controlled submissions.

Cadence rule

Each sprint ends with a go / no-go decision based on out-of-fold evidence, not on leaderboard excitement.

Rules and guardrails

Competition guardrails

- Encode schema validation before your first serious submission.
- IDs must align exactly; probabilities must remain in [0,1].
- Horizon outputs must be monotone within each row.
- Log every submission with code version and OOF score.

Process guardrails

- Freeze one baseline notebook or script that always runs end-to-end.
- Separate feature generation from modeling and post-processing.
- Review leakage every sprint, not only at the end.
- Keep a submission budget; do not probe the public leaderboard blindly.

Timeline note

Public summaries indicate the competition opened on Jan 28, 2026, has an entry/team-merge deadline in late April, and a final submission deadline on May 1, 2026.

Success criteria

Healthy local CV

Repeated mean/std improvements across folds and seeds, not one lucky run.

Healthy public LB

Moves broadly in the same direction as OOF when you make meaningful changes.

Healthy process

Config-driven, reproducible, and easy to audit.

Minimum bar

- Trustworthy local metric
- Clean baseline submission
- At least one strong model family
- Stable fold design

Stretch bar

- Calibrated ensemble
- Lower fold variance than best single model
- Defensible feature story
- Slice-aware error analysis

Reproducible project setup

Foundation

- Create a clear repo layout: data/, features/, src/, notebooks/, configs/, submissions/.
- Version the feature pipeline separately from model configs and post-processing rules.
- Store one canonical fold assignment and reuse it everywhere.
- Track experiments with a simple registry: run name, model, feature set, OOF, LB, notes.

Suggested stack

- pandas or polars for wrangling
- scikit-survival or lifelines for survival baselines
- LightGBM / XGBoost / CatBoost for gradient boosting
- Optuna for tuning
- MLflow or a clean CSV log for tracking

Data audit and schema checks

First checks

- ID uniqueness and exact column contract across train and test.
- Type audit for every numeric and categorical field.
- Null patterns and structural missingness.
- Target columns, event definition, and follow-up time semantics.
- Any train/test schema mismatch before modeling begins.

Public clues

- Kaggle data page snippets indicate four CSV files.
- The same snippet suggests roughly 83 columns in total.
- Use those as expectations only; verify everything directly after download.
- Write a schema validator early and rerun it before every final submission.

Survival framing in plain language

Event

A fire comes within the operational threat threshold of an evacuation zone after the 5-hour feature window.

Time

Measure time-to-hit from t0+5h, not from t0. That distinction matters for local evaluation.

Censoring

If no hit is observed within the window, the row is right-censored rather than simply labeled negative.

Prediction target

Model a coherent survival function or horizon-specific risk curve, then convert it into the required 12h / 24h / 48h / 72h probabilities.

Timeline cue

t0 → first 5h of features → prediction time → evaluate risk at 12h / 24h / 48h / 72h.

Local metric first

Public summaries describe a hybrid score that mixes ranking quality and calibration quality.

Hybrid score ≈ 0.3 × C-index + 0.7 × (1 − weighted Brier score)

What it means

- Ranking quality matters: urgent fires should be ordered correctly.
- Calibration matters: the probabilities themselves must be trustworthy.
- A model can rank well and still lose on poor calibration.

Action

- Write a local evaluator and unit-test it on synthetic edge cases.
- Use it in every fold run before trusting any score.
- Keep the post-processing protocol constant when comparing models.

Cross-validation design

Principles

- Use one canonical fold assignment for every experiment.
- Balance event/censoring structure as much as possible across folds.
- If multiple rows belong to one incident or geography, group them to avoid bleed-through.
- Keep preprocessing, calibration, and meta-modeling strictly fold-local.

What good looks like

- Fold scores are stable enough to compare models.
- No duplicated entity leaks across train/valid splits.
- Train-fold preprocessing mirrors future inference exactly.
- The same CV contract survives into blending and stacking.

Backbone rule
If the fold logic is weak, every later gain is suspect.

Leakage threat model

Future information

Any feature that uses behavior beyond the allowed 5-hour window is a red flag.

Fold contamination

Global imputers or encoders fitted on all data can leak validation information.

Group bleed

Rows from the same fire or geography should not straddle train and validation blindly.

Leaderboard probing

Repeated public-LB-driven tweaks can overfit the visible split.

Golden rule

If a feature seems too predictive to be true, assume it is leaking until you can prove both time safety and fold safety.

02

EDA that matters

Use exploratory analysis to discover leakage risks, survival structure, feature quality, and high-value modeling slices.

EDA objectives

What EDA should answer

- What is the exact target behavior and censoring structure?
- Which feature families are stable and which are brittle?
- Where are the biggest leakage and drift risks?
- Which slices are easy, hard, or operationally critical?

Deliverables

- Data memo with schema, missingness, drift, and leakage notes.
- Slice inventory for later error analysis.
- Prioritized feature backlog.
- Fixed fold assignment and tested evaluator.

Outcome and censoring analysis

Questions to answer

- How many rows hit within each horizon?
- How many are censored early versus fully observed?
- Are short times-to-hit dominating the score?
- Is the event / censor mix balanced across folds?

Why it matters

- Class imbalance and censoring shape your loss landscape.
- Some models may rank well but mis-handle mid-horizon calibration.
- This analysis tells you whether the competition is mostly about near-term triage or broader forecast quality.

Train vs test drift

Drift review

- Compare summary statistics, missingness, and quantile bins for important features.
- Flag columns whose behavior changes materially between train and test.
- Separate true test shift from train-only artifacts caused by joins or target handling.
- Prefer stable feature families in the first strong baseline.

Output

- A drift watchlist.
- A list of stable core features.
- A list of risky features that require caution or calibration support.
- A better explanation for OOF vs leaderboard disagreements.

Missingness and anomalies

Patterns to separate

- Random sparse nulls
- Systematic nulls by slice or operational state
- Impossible values or unit bugs
- Extreme outliers that may destabilize models

Response

- Use explicit dtype maps and robust imputers.
- Add null flags when missingness may itself be informative.
- Clip or winsorize only with a documented rule.
- Keep all cleaning steps reproducible inside each fold.

Geospatial sanity checks

What to validate

- Distance to evacuation zones and how that distance changes.
- Relative heading or spread toward the nearest zone.
- Whether centroid-based distances hide asymmetric spread.
- Whether near-zone rows behave like near-certain short-horizon risks.

Modeling implications

- Distance features can be extremely strong.
- They may also create near-perfect separation in some regimes.
- That means you should check calibration carefully and search for second-order interactions instead of trusting raw distance alone.

Correlation and redundancy

Approach

- Group features into families: distance, geometry, dynamics, metadata, interactions.
- Within each family, identify redundant columns and keep short, strong representatives.
- When two features tell the same story, prefer the more stable and interpretable one.

Payoff

- Lower variance.
- Faster tuning.
- Cleaner interpretation.
- Simpler feature ablations later in the competition.

Slice discovery

Useful slices

- Very near zones vs far zones
- Fast-moving vs slow-moving fires
- Rows with rich dynamics vs rows with sparse dynamics
- Rows with heavy missingness vs clean rows

Why slice analysis matters

- It tells you where the model really fails.
- It turns feature engineering into targeted improvement instead of random complexity.
- It helps decide whether a model deserves a place in a final ensemble.

Exit criteria for the exploratory phase

Deliverable 1

A data memo with schema, drift, missingness, and leakage findings.

Deliverable 2

A prioritized feature backlog with must-build and nice-to-have ideas.

Deliverable 3

A fixed fold assignment and a tested local evaluator.

Only then move on

Starting the model ladder before these artifacts exist usually creates false leads and wasted tuning effort.

03

Model ladder

Test a diverse set of baselines first, then invest only in the families that repeatedly improve out-of-fold score.

Baseline ladder

Level 0

Constant or horizon-prior submission. Use it to debug schema and evaluator.

Level 1

Distance-only or simple geometry-only model. Use it to see how far obvious signal already goes.

Level 2

Classical survival baseline: Cox/AFT or a simple survival-style benchmark.

Level 3

Tree-based survival and discrete-time boosting. These are likely the core competitive families.

Baseline family 1: classical survival

Why keep them

- They are disciplined baselines for a survival problem.
- They help interpret which engineered features matter.
- Their outputs can diversify a later stack even if they do not win outright.

What not to do

- Do not overinvest if nonlinear models clearly dominate.
- Do not confuse interpretability with leaderboard ceiling.
- Do keep their OOF predictions if they add calibration diversity.

Baseline family 2: tree-based survival

Candidates

- Random Survival Forest for a strong nonparametric benchmark.
- Gradient Boosting Survival Analysis for flexible survival-aware boosting.
- Evaluate both raw survival curves and horizon probabilities after calibration.

What to watch

- Fold variance.
- Calibration quality at required horizons.
- Whether the model behaves sensibly on sparse or censored slices.

Baseline family 3: discrete-time boosting

Why this family is attractive

- Boosters are flexible, fast, and Kaggle-friendly.
- A discrete-time reformulation adapts them to multi-horizon risk.
- They often become core leaderboard models when paired with good calibration.

Risks

- A strong booster can overfit the public leaderboard if validation is weak.
- Post-processing matters: calibration and monotonicity are part of the final model.
- Compare LightGBM, XGBoost, and CatBoost only under the same fold and post-process contract.

Calibration layer

Calibration workflow

- Fit calibrators on out-of-fold predictions only.
- Try simple methods first: isotonic or logistic calibration per horizon.
- Repair monotonicity after calibration.
- Check calibration by slice, not only globally.

Why it matters here

- Public metric summaries place heavy weight on calibration quality.
- A good ranker can become a strong leaderboard model only after probability repair.
- Never compare models under different post-processing protocols.

Model comparison matrix

Compare on

- OOF mean
- OOF std
- Ranking quality
- Calibration quality
- Slice behavior

Promote if

- Gain is repeated
- Variance stays acceptable
- Adds complementary error profile

Drop if

- Gain appears only on public LB
- Model is unstable across folds
- Benefit disappears after calibration or repair

Promotion rule before tuning

Promote

Repeated OOF gain, acceptable variance, and a different error pattern from the current best model.

Hold

Looks promising but unstable; keep in the zoo, but do not spend major tuning budget yet.

Drop

Improves only the public leaderboard or only a narrow, unimportant slice.

Budget rule

A good stack is built from promoted models, not from every notebook you ran.

04

Feature engineering

Convert domain intuition into train-fold-safe features that improve both urgency ranking and probability calibration.

Raw feature families

Must organize first

- Distance to zones
- Geometry and shape
- Movement or spread dynamics
- Context or metadata
- Engineered interactions

Why organize by family

- It makes ablation cleaner.
- It makes redundancy pruning easier.
- It reduces tuning chaos.
- It helps later stacking because you know which models rely on which signal families.

Physics-informed and geospatial features

Good ideas

- Relative motion toward the nearest evacuation zone.
- Projected advance under current heading and spread proxies.
- Distance normalized by current geometry.
- Shape descriptors that capture directional threat.

Execution rule

- Build a small geometry library.
- Test new features on hand-checked rows before mass generation.
- Document units and direction conventions.
- Keep every transform time-safe and fold-safe.

Temporal dynamics from the first five hours

Potential features

- Perimeter growth inside the 5-hour window
- Directional drift
- Rate-of-change summaries
- Flags showing whether dynamic information is actually observed

Important caution

- Public notes suggest many fires may have only one observation.
- That means dynamic features can collapse to zero for large subsets.
- Treat “no dynamic information” as its own signal rather than just noise.

Interactions and threshold features

Promising patterns

- Distance × spread
- Distance thresholds such as near / mid / far
- Shape × heading combinations
- Log or rank transforms for heavy-tailed variables

Restraint rule

- Use interactions surgically.
- Start with domain-legible combinations.
- If a feature cannot be explained in one sentence, it probably belongs later in the queue.

Feature selection and regularization

Selection strategy

- Run family-wise ablations first.
- Then use model-specific importance to prune obvious noise.
- Prefer a slightly smaller stable set to a massive fragile one.

Regularization strategy

- Regularize linear models aggressively.
- Tune depth, leaves, and minimum-data settings for trees and boosters.
- Keep features that improve mean OOF, reduce variance, or help critical slices.

Safe feature generation checklist

Time safety

Does the feature use only information available by t0+5h?

Fold safety

Is every fit or encoding operation performed inside the training fold only?

Schema safety

Will the feature exist for every test row with the same definition?

Ablation safety

Does the feature help repeatedly in OOF rather than by luck?

Cheapest bug

The cheapest bug to fix is the one you detect at feature construction time.

05

Tuning, bagging, and stacking

Use compute strategically: first tune stable base learners, then ensemble for variance reduction and complementary error correction.

Hyperparameter tuning strategy

Tuning discipline

- Tune only promoted models.
- Use a small, high-signal search space first.
- Optimize on mean OOF while monitoring fold dispersion and key slices.
- Retest the winning configuration with multiple seeds.

Three phases

- Pilot search: find direction.
- Focused search: refine the best region.
- Robustness reruns: test whether the gain survives.
- Stop when additional trials no longer compound.

Search spaces by model family

RSF / GBSA

- Trees, depth, min leaf
- Balance bias and variance
- Keep depth modest early

Gradient boosters

- Leaves, depth, learning rate, min data
- Powerful but easy to overfit
- Pair with calibration

Post-process

- Calibrator strength
- Monotonicity repair
- Blending weights

Bagging and seed ensembling

Why bagging is attractive

- It reduces variance without requiring a full stack.
- It often improves probability smoothness.
- It is a low-risk way to strengthen already good models.

How to do it

- Bag only strong base learners.
- Vary random seed, subsample, or mild hyperparameters.
- Measure the marginal gain of each added member.

Blend vs stack

Simple average

- Use when all base models are similarly strong.
- Best first ensemble baseline.

Weighted blend

- Use when a few models clearly dominate.
- More control, but weight overfit is possible.

Stacking

- Use only when models are genuinely complementary.
- Highest ceiling, highest risk.

Rule of thumb
Start with a weighted blend of your top 2–4 stable models. Stack only if OOF residual patterns clearly differ.

Recommended stacking architecture

Shared folds

One fold contract for all models.

Base learners

Classical survival + tree survival + discrete-time booster.

OOF features

Horizon probabilities and a few disagreement summaries.

Meta-model

Simple regularized blender, not an exotic second-stage monster.

Best practice
Most stacking gains come from complementary base signals and clean OOF construction, not from a fancy meta-learner.

Out-of-fold predictions and meta-features

Save these

- Raw OOF horizon probabilities for every base model.
- A few compact summaries such as mean probability and model disagreement.
- Optional slice flags from EDA if they help the stack choose wisely.

Discipline

- Never leak in-fold predictions into the meta-model.
- Start with a heavily regularized linear combiner.
- Only move to nonlinear stacking if clear gains remain.

Post-processing and probability repair

Required steps

- Calibrate horizon probabilities on OOF predictions.
- Enforce monotonicity row-wise.
- Clip final probabilities to [0,1].
- Validate schema again before submission.

Why it deserves time

- Post-processing is part of the model in this competition.
- It can convert a good ranker into a better hybrid-score model.
- It reduces the risk of invalid or unstable submissions.

06

Learning from experts and competing smart

Use public notebooks and discussions as accelerators: borrow ideas, validate them locally, and preserve leaderboard discipline.

How to mine notebooks and discussions productively

Good habits

- Track ideas in a structured log: feature idea, model idea, validation idea, post-process idea.
- Prefer notebooks that explain failure modes, not only leaderboard screenshots.
- Re-implement promising ideas cleanly inside your own fold framework.

What to borrow from public work

- Metric-aware validation.
- Physics-informed features.
- LightGBM-style survival reformulations.
- Bagged hybrid survival ensembles.

Submission strategy and leaderboard hygiene

Smart behavior

- Use submissions for genuine checkpoints, not tiny tweaks.
- Treat public score as a noisy fold, not the truth.
- If public LB disagrees with repeated OOF, investigate drift or variance before reacting.

Decision rules

- OOF up + LB up: promote carefully.
- OOF up + LB flat/down: inspect shift, do not panic.
- OOF down + LB up: suspect overfit.
- OOF flat + LB spike: retest before trusting it.

Final checklist and next steps

- Lock a fold assignment and a local evaluator you trust.
- Finish EDA with a written drift, leakage, and slice memo.
- Build the baseline ladder and keep only promoted models.
- Tune the stable core, then add bagging, calibration, and finally stacking.
- Validate schema, monotonicity, and probability bounds before every final submission.
- Learn from experts, but only adopt ideas that improve your own OOF.

Immediate next move
Start with data audit, target reconstruction, and one clean baseline submission.

Outcome if executed well
A professional Kaggle workflow that keeps improving without losing scientific discipline.