Hackathon Brief Document

Intelligent Insurance Bundle Recommendation

DataQuest

Phase I: 7 Hours · Model Development
Phase II: 3 Hours · Productization & Deployment

Contents

1 Hackathon Overview

The insurance sector increasingly relies on data-driven decision-making to personalize offers, reduce risk, and improve customer satisfaction. Choosing the right insurance contract remains a complex task for both clients and brokers, involving multiple parameters such as client profiles, characteristics, risk exposure, and budget constraints.

This hackathon challenges participants to design and implement an **intelligent recommender system** that suggests the most suitable insurance contracts for clients based on their profile data.

Participants will work through a **two-phase competition**, combining Machine Learning performance with business and deployment readiness. This structure reflects the real-world constraints and operational requirements faced by modern insurance companies.

Competition Structure:

- **Phase I** (7 hours) — Model development, feature engineering, and leaderboard submissions.
- **Phase II** (3 hours) — API development, deployment, technical report, and final pitch.

2 The Problem

Your objective is to build a predictive model that can accurately determine which `Purchased_Coverage_Bundle` a prospective customer will ultimately choose based on data from a global insurance brokerage network.

This is a **multi-class classification problem** with **10 classes** (labeled `0`–`9`). Participants must analyze the provided training data, identify the underlying behavioral and demographic patterns, and apply their findings to predict the exact insurance bundle for each user in the test set. How you approach data preparation, feature engineering, and model selection is entirely up to you.

3 Data Definitions

The dataset contains detailed logs of customer profiles, policy preferences, underwriting timelines, and broker interactions.

- **Training set:** 60,868 rows (features + target column).
- **Test set:** 15,218 rows (features only — you predict the target).

Identifiers & Target

- `User_ID`: Unique identifier for the customer.
- `Purchased_Coverage_Bundle`: **[TARGET]** The final insurance package selected (integer, 10 classes).

Demographics & Financials

- `Adult_Dependents / Child_Dependents / Infant_Dependents`: Number of individuals covered.
- `Estimated_Annual_Income`: The estimated yearly income of the household.
- `Employment_Status`: Working arrangement of the primary applicant.

- `Region_Code`: Anonymized geographic location.

Customer History & Risk Profile

- `Existing_Policyholder`: If the user already has an active policy.
- `Previous_Claims_Filed`: Number of prior claims.
- `Years_Without_Claims`: Consecutive years without filing a claim.
- `Previous_Policy_Duration_Months`: Duration of prior insurance policy.
- `Policy_Cancelled_Post_Purchase`: Whether the user canceled shortly after buying.

Policy Details & Preferences

- `Deductible_Tier`: Out-of-pocket deductible level.
- `Payment_Schedule`: Premium payment frequency (e.g., Monthly, Annual).
- `Vehicles_on_Policy`: Number of vehicles covered.
- `Custom_Riders_Requested`: Number of special coverage add-ons.
- `Grace_Period_Extensions`: Number of times the payment deadline was extended.

Temporal & Operational

- `Policy_Start_Year / Policy_Start_Month / Policy_Start_Week / Policy_Start_Day`: When the policy began.
- `Days_Since_Quote`: Days between quote issuance and data snapshot.
- `Underwriting_Processing_Days`: Days taken to underwrite the policy.
- `Policy_Amendments_Count`: Number of post-purchase policy amendments.

Broker & Acquisition

- `Broker_ID`: Identifier for the servicing broker.
- `Employer_ID`: Identifier for the employer (group policies).
- `Broker_Agency_Type`: Type of brokerage agency.
- `Acquisition_Channel`: How the customer was acquired.

4 Evaluation & Scoring

4.1 Evaluation Metric

Submissions are evaluated using the **Macro F1-Score**. This metric calculates the F1-Score for each class independently and then takes the unweighted average. It is chosen to evaluate performance **equally across all insurance bundles**, regardless of how frequently they appear in the data.

$$\text{Macro F1} = \frac{1}{N}\sum_{i=1}^{N} 2 \times \frac{\text{Precision}_i \times \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i}$$

4.2 Final Score Calculation

Your raw Macro F1 is adjusted by two penalty multipliers based on model size and inference latency:

$$\texttt{final\_score} = \text{Macro F1} \times \underbrace{\max\left(0.5,\ 1 - \frac{\text{size}_{\text{MB}}}{200}\right)}_{\text{Size Penalty}} \times \underbrace{\max\left(0.5,\ 1 - \frac{\text{latency}_{\text{s}}}{10}\right)}_{\text{Latency Penalty}}$$

- **Size Penalty:** Models up to 200 MB are penalized linearly; the multiplier floors at 0.5 (i.e., a model $\geq$200 MB loses at most 50% of its F1 score).
- **Latency Penalty:** Inference latency up to 10 s is penalized linearly; the multiplier floors at 0.5.
- **Only the `predict()` function is timed.** Preprocessing and model loading happen before the clock starts.

Note

A model with perfect F1 (1.0), 50 MB size, and 1 s latency would score: $1.0 \times 0.75 \times 0.9 = 0.675$. Keep your models small and fast.

5 Submission Format

Submissions are uploaded as a `.zip` file to the judge platform. The ZIP must contain **exactly 3 files at the root level** (no nested folders):

```
submission.zip
|-- solution.py          # Your code (required)
|-- model.<ext>          # Your trained model (required, exactly
   one)
|-- requirements.txt     # Python dependencies (required)
```

5.1 `solution.py` — Required Interface

Your `solution.py` must export exactly three functions:

```python
def preprocess(df):          # Returns a pandas DataFrame
    ...

def load_model():             # Returns your loaded model object
    ...

def predict(df, model):       # Returns a DataFrame with User_ID
    ...                       # and Purchased_Coverage_Bundle
```

The `predict()` function must return a Pandas DataFrame with exactly two columns:

```
User_ID,Purchased_Coverage_Bundle
USR_060868,7
USR_060869,2
USR_060870,4
...
```

Important

- All test `User_ID`s must be present in your predictions. Missing IDs will cause the submission to fail.
- `Purchased_Coverage_Bundle` values must be integers (0–9).
- Do **not** drop the `User_ID` column in `preprocess()` — it is needed in `predict()`.

5.2 Model File

- Exactly **one** file named `model.<ext>`.
- Allowed extensions: `.joblib`, `.pkl`, `.onnx`, `.h5`, `.pb`, `.pt`, `.pth`, `.safetensors`, `.keras`, `.tflite`, `.bin`.

5.3 `requirements.txt`

- Plain package specifiers only (e.g., `pandas==2.1.4`).
- **Forbidden options:** `-extra-index-url`, `-index-url`, `-find-links`, `-trusted-host`, and similar pip flags will be rejected.
- Dependencies are installed with `-only-binary :all:` — packages that require compilation will fail.

6 Constraints & Resource Limits

Models run inside an isolated Docker container with strict resource caps.

Constraint	Description	Limit
Max Upload Size	ZIP file upload limit	**50 MB**
Max Uncompressed Size	Total extracted content	**500 MB**
Container Memory	RAM available at runtime	**1 GB**
Container CPU	CPU cores allocated	**1 core**
Execution Timeout	Total pipeline time	**120 s**
Max Submissions	Per team, lifetime	**20**
Rate Limit	Submission cooldown	**1 / min**

Important

No network access at runtime. Your code runs in a fully sandboxed container with no internet connectivity. All model weights, solution file, and dependencies must be included in the ZIP.

6.1 Pre-installed Packages

The following packages are pre-installed in the judge environment and **cannot be overridden**:

- `numpy 1.26.4`
- `scipy 1.11.4`
- `pandas 2.1.4`
- `scikit-learn 1.3.2`
- `xgboost 2.0.3`
- `lightgbm 4.6.0`
- `catboost 1.2.3`
- `torch 2.6.0` (CPU)
- `torchvision 0.21.0`
- `torchaudio 2.6.0`
- `tensorflow 2.14.0`
- `joblib 1.3.2`
- `tqdm 4.66.1`
- `pyyaml 6.0.1`
- `cloudpickle 3.0.0`

Note

The Python version in the judge is **3.10**. Ensure your code is compatible.

7 Security Rules

Submissions go through automated security validation. Violations cause **immediate rejection**.

- **Flat structure only** — no nested directories inside the ZIP.
- **No symlinks**, hidden files, or encrypted archives.
- **Allowed file extensions only** — text files (`.py`, `.txt`, `.json`, `.md`, `.csv`) and model files (see Section 5.2).
- **Zip bomb protection** — max compression ratio of 100:1 for files >1 KB.
- **No null bytes** in text files (binary-disguised-as-text is rejected).

8 Deliverables

8.1 Phase I: Model Development (7 Hours)

Focus: Model Performance and Accuracy

- **Submission Files:** `solution.py`, `model.<ext>`, and `requirements.txt` packaged as a `.zip`.
- Submit via the judge platform. Track your rank on the live leaderboard.
- **Bonus:** Additional points awarded for model explainability (e.g., SHAP, feature importances).

8.2 Phase II: Productization & Deployment (3 Hours)

Focus: Engineering, Deployment, and Presentation

Participants must submit a Git repository, a technical report, and deliver a presentation (including an optional demo).

Technical Report Requirements

- Complete system architecture diagram.
- Explanation of engineered features (Feature Engineering).
- Brief justification of model choice.
- **Bonus:** EDA notes, diagrams, and documentation of failed attempts.

API & Inference Requirements

- **Minimal API:** Must include at least 2 endpoints.
- **Interface:** A simple UI for inference.
- **Bonus (MLOps):** CI/CD pipelines (e.g., GitHub Actions), retraining pipelines, or caching strategies (e.g., Redis, in-memory).
- **Bonus (Deployment):** Live deployment of the Frontend and API.
- **Repo Quality:** Clean code, proper structure, and documentation.

9 Scoring Criteria

Total Score: 100 Points + 25 Bonus Points

Category	Details	Points
Model Performance	**(Total)**	**60**
	Feature Engineering Creativity	20
	Final Score (Macro F1 × Penalties) × 30	30
	Explainability (SHAP, feature importances, etc.)	10
Technical Report	Architecture, Feature explanation, Model justification	**10**
Pitch & Business	Presentation, Business Value, Demo	**15**
Web Application	**(Total)**	**15**
	Frontend Implementation	5
	API Implementation	10
Bonuses	MLOps, Deployment, Repo Quality	**+25**

Judging Criteria Overview

- **Code Quality** — clean, readable, well-structured code.
- **Innovation & Creativity** — novel approaches to feature engineering and modeling.
- **Adherence to Constraints** — respecting resource limits and submission format.

10 Leaderboard

- The leaderboard ranks teams by their **best final score** (highest across all submissions).
- Team names are **anonymized** to other participants — you can only see your own team name.
- A **blind period** activates during the final hour of the competition. During this window, the leaderboard freezes — scores are still recorded but rankings are no longer updated publicly.

11 How the Judge Works

Understanding the judge pipeline helps you avoid common submission failures.

1. **Upload & Security Check** — Your ZIP is validated for structure, extensions, symlinks, and zip bombs in an isolated sandbox.
2. **Extraction & Sanitization** — The ZIP is extracted; `requirements.txt` is scanned for forbidden pip options.
3. **Build** — Missing dependencies from `requirements.txt` are installed if needed.
4. **Run** — Your `solution.py` is executed in a **network-isolated** container. The runner calls `preprocess()`, then `load_model()`, then starts the timer and calls `predict()`.
5. **Score** — Your predictions are compared against ground truth using Macro F1, then adjusted by size and latency penalties.

Note

If any step fails, the submission is marked as failed with an error message visible on the platform. You do **not** lose a submission attempt on security check failures.

Good luck, and may the best model win!