**Software Design Document**

**Step 1: Define the purpose and scope**

**Purpose**

The purpose of this program is to analyze a health insurance dataset by managing, visualizing, and comparing groups of data.

* **Problem statement**: Health insurance datasets contain information such as BMI, # of children, smoking status, region, and medical charges. To better understand trends in healthcare costs, we need a program that analyzes these records.
* **Objectives**: Generate descriptive statistics, create simple text-based visualizations, and test hypotheses about the relationships between health factors and charges.
* **Value proposition**: This program provides clear insights into how demographic and lifestyle factors influence medical charges. By combining statistics, visuals, and hypothesis testing it will help to identify cost disparities and fairness across regions.

**Scope**

This program is limited to analyzing the given health insurance dataset. It covers data storage, descriptive statistics, text-based histograms, and group comparisons.

* **In-scope features:** Loading and storing N records in an object, calculating statistics (count, mean, std, min, max, percentiles), creating text-based histograms, sorting regions by average charges, and calculating linear regression.
* **Out-of-scope features:** Using external libraries to make a more visual graph, interactive dashboards, advanced machine learning, and predictive healthcare analytics beyond the provided tasks.
* **Milestones and timeline:**

Day 1–2: Dataset & Setup

* Load the dataset and create a custom object to store the records.
* Make sure data types (age, BMI, children, charges, etc.) are usable.

Day 3–4: Statistics & Basic Visualizations

* Implement summary stats (count, mean, std, min, percentiles, max).
* Build text-based histograms (ages horizontal, BMI vertical, smoker vs non-smoker).

Day 5–6: Group Comparisons & Hypotheses

* Counts by children, regions, and smokers.
* Test fairness and simple hypotheses (older vs younger charges, smokers vs non-smokers, regional differences, BMI ranges).

Day 7–8: Regression Analysis

* Implement simple linear regression for charges vs BMI, children, and region.
* Compute Pearson r values.
* Run predictions on the given new x-values.

Day 9–10: Integration & Final Report

* Organize all functions into one program.
* Document results clearly (stats tables, histograms, hypothesis conclusions, regression findings).
* Final testing and polishing.
* **Assumptions and constraints:** The dataset provided is **clean and accurate** (no missing, corrupted, or duplicate records). Attributes (age, BMI, children, charges, region, smoker) are already in usable formats (e.g., no need for preprocessing). Statistical comparisons (e.g., smokers vs non-smokers, south vs north) are based only on the given dataset, not external data.

**Step 2: Conduct requirements analysis**

**Functional requirements**

This section describes the specific behaviors and functions the software must perform. It details what the system does.

* **Use cases and user stories:** Detail how different users (actors) will interact with the system to achieve their goals. A user story format is common in Agile methodologies (e.g., "As a user, I want to log in so that I can access my dashboard").
  + **As a student**, I can load the first N rows into typed objects so I can work with strongly-typed fields (age, bmi, etc.).
  + **As a user**, I can see count/mean/std/min/25%/50%/75%/max for age, BMI, children, and charges.
  + **As a user**, I can view **text histograms** of age (horizontal) and BMI (vertical) to understand distributions without graphics.
  + **As a user**, I can see totals per children value and smoker vs non-smoker counts as a vertical histogram.
  + **As a reviewer**, I can check regional **fairness** (±5%) and evaluate domain hypotheses (e.g., smokers pay more, BMI bucket ranges).
  + **As a reviewer**, I can get **sorted regions by average charges**.
  + **As a student**, I can compute and apply **simple linear regressions** and **Pearson r** for Charges vs BMI / Children / Region and apply formulas to unseen values.
  + **As a student**, I can load the first N rows into typed objects so I can work with strongly-typed fields (age, bmi, etc.).
  + **As a user**, I can see count/mean/std/min/25%/50%/75%/max for age, BMI, children, and charges.
  + **As a user**, I can view **text histograms** of age (horizontal) and BMI (vertical) to understand distributions without graphics.
  + **As a user**, I can see totals per children value and smoker vs non-smoker counts as vertical histogram.
  + **As a reviewer**, I can check regional **fairness** (±5%) and evaluate domain hypotheses (e.g., smokers pay more, BMI bucket ranges).
  + **As a reviewer**, I can get **sorted regions by average charges**.
  + **As a student**, I can compute and apply **simple linear regressions** and **Pearson r** for Charges vs BMI / Children / Region and apply formulas to unseen values.
* **System workflows**: Map out the logical flow of the system. Visual aids like flowcharts can be helpful here.
  + **Start** → parse CLI args: path, N.
  + **Load** → read first N rows → ArrayList<Record>.
  + **Extract Columns** → ages/bmis/children/charges lists.
  + **Describe** → compute stats → print.
  + **Visualize (text)** → age horizontal hist; BMI vertical hist; smoker bars; children counts.
  + **Group & Compare** → children totals; fairness by region; region averages sorted.
  + **Hypothesis Tests** → 50+ vs ≤20; BMI bucket ranges; per-child trend; smoker mean & range; smoker south premium; smoker vs non-smoker BMI; smoker age distribution; young-smoke threshold; south vs north comparisons.
  + **Regression** → charges vs BMI/children/region → output a,b,r; apply to new x/y sets.
  + **Done**
* **Data handling:** Specify how data is processed, stored, and managed.
  + **Input**: insurance.csv (Kaggle format).
  + **Parsing**: Scanner line-by-line; header skipped; split by comma into Record.
  + **Storage**: ArrayList<Record> (first N records).
  + **Transformations**:
    - Extract columns into ArrayList<Double> for stats.
    - Grouping via HashMap/TreeMap (children counts, region sums).
    - Histograms via bin counts in int[], labels in String[].
  + **Computation**:
    - Stats: sorted ArrayList<Double> + quantile interpolation; sample std-dev.
    - Fairness: compare region proportions to uniform (25%) with ±5% tolerance.
    - Hypotheses: compute group means/ranges; monotonic checks across children.
    - Regressions: ordinary least squares; Pearson r; deterministic region encoding.
  + **Output**: Console text (tables, ASCII histograms, conclusions).

Non-functional requirements

* **Performance:** Requirements related to speed, response time, and resource usage.
  + **Time-to-Result** 
    - For N ≤ 1,500 rows (full Kaggle: 1,338), end-to-end runtime ≤ **1.0 s** on a typical laptop s*o* I/O dominates; computations are O(N) except percentile calculations requiring sort O(N log N).
* **Large N Bound**
* For N ≤ **250k** rows, runtime ≤ **20 s**, peak RSS ≤ **512 MB**.so use primitive-leaning data extraction, reuse arrays, avoid duplicating lists.
* **Algorithmic Complexity**
* O(N log N) (one sort per metric set).
* **I/O Throughput**
* ≥ **5 MB/s** effective CSV ingest for local disk so scanner + buffered reading; skip header once; single pass to populate domain objects.
* **CLI Startup**
* Cold start to first log line ≤ **200 ms**.
* **Security:** Measures needed to protect data, such as authentication, authorization, and encryption.
  + **Principle of Least Privilege**
    - Only reads files explicitly passed via CLI; no directory crawling..
  + **Path Handling**
    - Normalize/resolve path; reject control chars and null bytes; never echo raw path back without quoting.
  + **Data Minimization**
    - No persistent storage; no temp files. In-memory only; process lifetime equals analysis lifetime.
  + **PII Handling**
    - Dataset includes potential sensitive attributes. Output is aggregate **only**; never writes per-row dumps by default.
  + **Supply-Chain**
    - Pure JDK
    - Build with reproducible flags; source and binary checksums optional.
  + **Denial of Service Protections**
    - CLI guardrails: hard cap on N can be set , and --timeout for graceful abort.
* **Scalability**: The system's ability to handle an increased workload or number of users.
* **Configurable N**
* Mustallow N to be supplied via CLI; system operates identically for any N up to available memory.
* **Memory Footprint Controls**
* Switch “column mode” (store only needed columns for analyses) to reduce memory by ≥40**%** vs storing full and record objects and feature flag or CLI --columns age ,bmi, children ,charges.
* **Streaming Mode (Roadmap)**
* All metrics/hypotheses except percentiles can be computed streaming in O(1) space. SDD documents separable calculators to enable later refactor without redesign.
* **Reliability and availability**: How often the system is available and how it recovers from failure.
  + **Deterministic Output**
    - Given same file + N, outputs are byte-for-byte identical (no randomized tie-breaks).
* **Input Robustness**
  + - Blank lines should be ignored; malformed rows counted and if a required field can’t be parsed, row is skipped; run continues.
* **Failure Handling**
  + - File not found should be clear error and non-zero exit code.
    - Numeric edge cases should be division-by-zero.
  + **Test Coverage**
    - **≥85%** line coverage for statistics, histograms, grouping, regression, and fairness logic so unit tests + property
  + **Logging & Observability**
    - Info ,Warn , Debug
* **Usability:** How easy and intuitive the system is for the user.
  + **Output Readability**
    - Stable, aligned columns; thousand-separators for money; 2–4 decimal places where meaningful.
  + **Text-Only Visuals**
    - Horizontal/vertical histograms rendered with counts printed.
  + **U-4 Error Messages**
    - Actionable, single-line: ERROR: insurance.csv not found. Try: java InsuranceAnalysis <path> [N]
  + **Accessibility**
    - No reliance on color; all semantics expressed via text and symbols.
    - Locale-safe numeric formatting uses.
  + **Deterministic Sorting & Labels**
    - Regions and bins reported in consistent patterns.