

# A Web-Based Platform for Exploratory Data Analysis with Integrated Fairness and Drift Detection

Ryan Dielhenn, Joe Jimenez, Roshan Roy Suja

*Department of Computer Science  
California State University, Los Angeles  
Los Angeles, CA, USA*

## Abstract

Exploratory Data Analysis (EDA) is a critical phase in data science workflows, yet existing tools often lack integration of fairness assessment and dataset drift detection capabilities. We present a comprehensive web-based EDA platform that combines traditional statistical analysis with modern fairness metrics and drift diagnostics. The system employs a three-tier architecture with FastAPI backend and Streamlit frontend, utilizing DuckDB for efficient in-memory analytics. Our platform supports multi-file ingestion, interactive visualizations, correlation analysis, demographic parity assessment, and Population Stability Index (PSI) calculation for drift detection. Performance evaluation demonstrates the system's capability to handle datasets exceeding a gigabyte with acceptable query response times.

## I. INTRODUCTION

### A. Motivation

The proliferation of machine learning applications across critical domains has amplified the importance of comprehensive data understanding prior to model development. Traditional EDA tools provide statistical summaries and visualizations but often neglect two increasingly critical aspects: dataset fairness assessment and data drift detection. The absence of integrated fairness analysis can lead to biased model outcomes, while undetected dataset drift

compromises model reliability in production environments.

### B. Problem Statement

Data scientists currently rely on disparate tools for EDA, fairness assessment, and drift detection, leading to fragmented workflows and potential oversights. Existing solutions typically address only one aspect of data analysis, requiring users to switch between multiple platforms and manually integrate findings. Furthermore, many tools lack scalability for large datasets or require extensive setup procedures that hinder rapid exploratory analysis.

### C. Contributions

This paper presents the following contributions:

- A unified web-based platform integrating EDA, fairness metrics, and drift detection in a single interface
- In-memory analytics with optimized query patterns using DuckDB for workloads on large datasets.
- Comprehensive bias detection heuristics for both numeric and categorical features
- Population Stability Index (PSI) computation for multi-dataset drift analysis

## II. RELATED WORK

### A. EDA Tools and Platforms

Traditional EDA tools such as Pandas Profiling [1] and Sweetviz generate comprehensive HTML reports with automated statistics and

visualizations. These tools provide valuable initial data summaries but generate static outputs and users must regenerate reports to modify analysis parameters. Commercial platforms like Tableau and PowerBI offer extensive interactive visualization capabilities and are widely adopted for business intelligence workflows. Our platform takes a different approach, targeting the specific needs of ML practitioners who require integrated fairness assessment and drift detection alongside traditional exploratory analysis in an interactive web interface.

### **B. Fairness in Machine Learning**

The AI Fairness 360 toolkit [2] provides comprehensive fairness metrics but operates as a standalone library without integrated EDA capabilities. Similarly, Fairlearn [3] focuses on fairness-aware model training rather than preliminary data assessment. Our approach integrates demographic parity analysis directly into the exploratory phase, enabling early detection of potential bias sources.

### **C. Drift Detection Methods**

Dataset drift detection has been studied extensively, with methods ranging from statistical tests [4] to deep learning approaches [5]. The Population Stability Index (PSI) remains widely adopted in industry for its interpretability and computational efficiency. Our implementation extends PSI calculation to support both numeric and categorical features with automated binning strategies.

## **III. SYSTEM ARCHITECTURE**

### **A. Overview**

The platform employs a three-tier architecture consisting of: (1) a FastAPI-based REST API backend [8], (2) a DuckDB analytical engine [6], and (3) a Streamlit-based interactive

frontend [7]. This separation of concerns enables easier maintenance of each component as well as setting up the project for deployment, development of new tools that access the API (e.g. CLI instead of GUI), and multi-user scenarios with authentication.

### **B. Backend Architecture**

The FastAPI backend exposes RESTful endpoints organized into five functional modules:

**Dataset Management:** Handles file uploads (CSV, Parquet, ZIP archives), schema extraction, and preview generation. The ZIP ingestion pipeline supports multi-file combination through DuckDB's UNION BY NAME operation, automatically aligning schemas across heterogeneous sources.

**Distributions:** Computes histograms for numeric features using quantile-based binning and value counts for categorical features. Queries are optimized to sample large datasets while maintaining statistical representativeness.

**Correlations:** Calculates Pearson correlation coefficients using DuckDB's native CORR() function. The implementation avoids full table materialization by computing correlations directly in SQL, significantly reducing memory footprint.

**Fairness Assessment:** Implements demographic parity difference calculation by comparing selection rates across sensitive attribute groups. Users specify binary target definitions through threshold-based rules.

**Drift Detection:** Computes PSI between reference and current datasets using quantile-aligned bins for numeric features and frequency-based comparison for categorical features.

### C. Database Layer

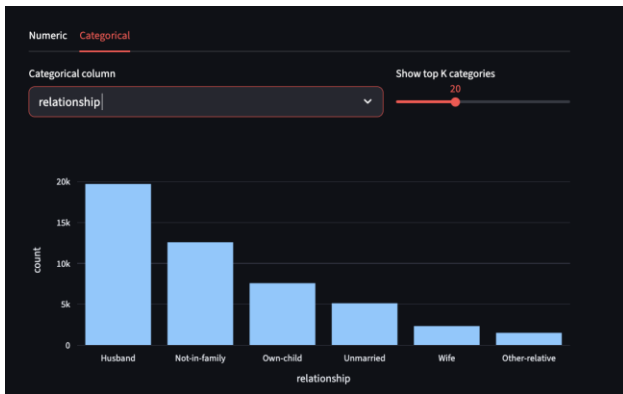
DuckDB serves as the analytical engine due to its columnar storage, vectorized execution, and SQL compatibility. Unlike traditional RDBMS systems, DuckDB operates in-process without requiring separate server management, simplifying deployment while delivering performance comparable to distributed systems for single-node workloads. The storage layer maintains a metadata table tracking dataset provenance, row counts, and ingestion timestamps. Actual data tables use the naming convention `ds_<dataset_id>` to avoid identifier collisions.

### D. Frontend Architecture

The Streamlit frontend comprises four interactive pages:

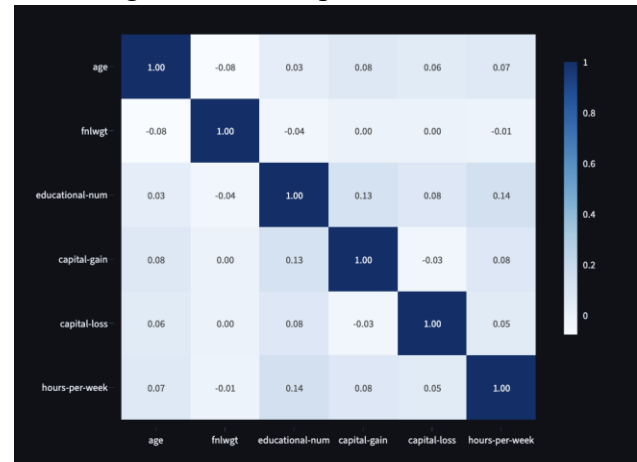
**Explore:** Provides dataset upload, schema inspection, and preview capabilities with adjustable row limits.

**Distributions:** Presents interactive histograms and box plots for numeric features, with bar charts for categorical features. Integrated bias checks highlight potential data quality issues.



*Figure 1:* Interactive distribution visualization showing categorical feature analysis. The interface displays value counts, proportions, and enables users to adjust display parameters for exploratory analysis.

**Correlation:** Displays heatmaps of feature correlations with sortable pair tables identifying the strongest relationships between features.



*Figure 2:* Correlation analysis revealing weak linear relationships among features (highest  $r=0.13$  between capital-gain and educational-num), suggesting that income prediction will benefit from non-linear modeling approaches.

**Fairness & Drift:** Enables fairness metric computation and PSI-based drift analysis with configurable parameters.

Session state management ensures consistency across page navigation, with automatic synchronization of dataset selections.

## IV. METHODOLOGY

### A. Bias Detection Heuristics

#### 1) Numeric Features

For numeric columns, we implement multi-faceted bias detection:

**Bin Concentration:** The maximum proportion of observations within a single bin indicates feature concentration. We categorize concentration severity using empirically determined thresholds to flag features that may benefit from transformation or binning strategies.

**Outlier Detection:** Using the Interquartile Range (IQR) method, outliers are defined as values beyond  $Q1 - 1.5 \times IQR$  or  $Q3 + 1.5 \times IQR$ . The proportion of outliers is computed and categorized to identify features requiring investigation.

**Skewness Analysis:** Computed via DuckDB's `SKEWNESS()` function, this metric identifies asymmetric distributions that may benefit from logarithmic or power transformations.

**Zero Inflation:** High proportions of zero values can indicate missing data coded as zeros or legitimately sparse features requiring special handling.

## 2) Categorical Features

For categorical columns, bias metrics include:

**Majority Class Dominance:** The proportion of observations in the most frequent category indicates class imbalance severity. High majority shares suggest potential need for resampling or class-weighting strategies.

**Imbalance Ratio:** Calculated as the ratio between majority and minority class frequencies, providing an intuitive measure of class balance.

**Effective Number of Classes:** Derived from Shannon entropy:  $\text{Effective\_K} = e^H$  where  $H = -\sum p_i \log(p_i)$ . This metric adjusts for frequency distribution, providing a more nuanced assessment than raw category counts.

## B. Population Stability Index

PSI quantifies distributional shift between reference and current datasets using the formula:  $\text{PSI} = \sum (p_{\text{ref}} - p_{\text{cur}}) \times \log(p_{\text{ref}} / p_{\text{cur}})$ , where  $p_{\text{ref}}$  and  $p_{\text{cur}}$  represent proportion of observations in each bin for reference and current datasets respectively.

Industry guidelines interpret PSI values as:  $\text{PSI} < 0.1$  (No significant shift),  $0.1 \leq \text{PSI} < 0.2$  (Moderate shift),  $\text{PSI} \geq 0.2$  (Significant shift requiring investigation).

For numeric features, we employ quantile-based binning using the reference dataset to define bin edges, ensuring consistent comparison. Categorical features use frequency-based comparison across all observed categories in both datasets.

## C. Fairness Metrics

We implement demographic parity difference, which measures the gap in positive outcome rates across protected groups. This metric was chosen for its simplicity and interpretability during exploratory analysis, though we acknowledge it doesn't capture all fairness notions. Demographic parity difference is defined as:  $\text{DP} = \max P(\hat{Y}=1|G=g) - \min P(\hat{Y}=1|G=g)$ , where  $G$  represents sensitive attribute groups and  $\hat{Y}$  denotes the predicted outcome. Users define binary outcomes through threshold-based rules on numeric features (e.g.,  $\text{income} > \$50k$ ), enabling flexible fairness assessment across different problem formulations.

# V. IMPLEMENTATION DETAILS

## A. Technology Stack

- Backend: FastAPI 0.104.0, Uvicorn with async support
- Database: DuckDB 1.1.3 with persistent storage
- Frontend: Streamlit 1.50.0, Plotly 6.3.0
- Data Processing: Pandas 2.3.3, NumPy 2.3.3
- Deployment: Docker Compose with separate API and frontend services

## B. Data Ingestion Pipeline

The ingestion module supports three input formats:

**Single CSV/Parquet:** Direct ingestion using DuckDB's `read_csv_auto()` and `read_parquet()` functions with automatic type inference.

**ZIP Archives:** Multi-stage pipeline: (1) upload and extract to temporary directory, (2) present file selection interface, (3) combine selected files via UNION BY NAME, (4) ingest as unified table, (5) cleanup temporary files.

**Schema Alignment:** DuckDB's UNION BY NAME operation automatically handles column ordering differences and inserts NULLs for missing columns, enabling seamless multi-file combination.

## C. API Design

RESTful endpoints follow consistent patterns with proper error handling returning appropriate HTTP status codes (400 for client errors, 404 for missing resources, 500 for server errors) with descriptive JSON error messages. Key endpoints include dataset upload, schema retrieval, distribution analysis, correlation computation, fairness assessment, and drift detection.

## D. Query Optimization

Several optimization strategies ensure scalability:

**Sampling:** For visualization tasks, statistical sampling reduces computational load while preserving distributional characteristics. Box plots use up to 100,000 sampled rows.

**Projection Pushdown:** Queries request only required columns rather than full table scans. This columnar access pattern aligns with DuckDB's storage model.

**Aggregation in Database:** Statistical computations occur within DuckDB using SQL aggregations rather than materializing data in Python. This approach leverages vectorized execution and minimizes data transfer.

**Connection Pooling:** A thread-safe singleton connection pattern prevents resource exhaustion under concurrent requests while avoiding connection overhead.

## E. Frontend Components

Streamlit's reactive programming model enables automatic UI updates upon user interaction. Session state management preserves dataset selections across page navigation. Custom CSS injection ensures visual consistency and responsive design across different screen sizes. Plotly visualizations provide interactive features including zooming, panning, and hover tooltips.

# VI. EXPERIMENTAL EVALUATION

## A. Efficiency Analysis

DuckDB's columnar storage and vectorized execution provide near-linear scaling for aggregation queries. Memory consumption remains bounded through streaming execution plans that avoid full materialization, enabling analysis of large datasets with minimal performance overhead. The architectural decision to use DuckDB rather than in-memory pandas DataFrames was motivated by these efficiency characteristics. Traditional pandas approaches require loading entire datasets into memory, while DuckDB's lazy evaluation and streaming execution enable processing of large workloads without holding data in memory. ZIP ingestion performance depends on archive size and compression ratio, with multi-file archives processing efficiently through DuckDB's native UNION BY NAME operation that handles

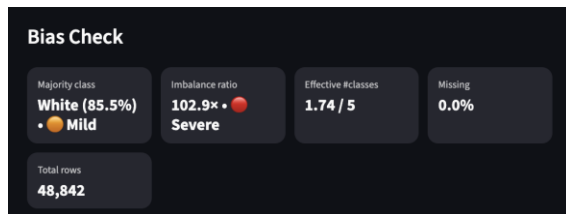
schema alignment and data loading in a single pass.

### B. Case Study: Bias Detection

We applied the platform to analyze the UCI Adult Census Income dataset (48,842 samples, 14 features) [9], which predicts whether individuals earn above \$50,000 annually based on 1994 US Census data. The bias detection module identified several data quality and fairness concerns:

**Concentration patterns:** Hours-per-week showed severe concentration (48.8% in single bin, corresponding to 40-hour work weeks) with 27.6% outliers, suggesting that the feature is better suited for categorical treatment rather than continuous numeric representation.

**Demographic imbalances:** Race distribution showed imbalanced majority class representation (85.5% White) with 102.9× imbalance ratio, while gender distribution exhibited moderate imbalance (66.8% male, 2.0× ratio). These protected attributes require consideration for fairness assessment.



*Figure 3:* Bias detection output for the race attribute, showing severe demographic imbalance (85.5% majority class, 102.9× ratio) and reduced effective number of classes (1.74/5).

**Target imbalance:** Income distribution showed 76.1% in the  $\leq \$50K$  class with 3.2× imbalance ratio, necessitating stratified sampling and class-weighted training approaches.

These findings directly informed preprocessing strategies including stratified cross-validation, sparse feature transformation, and identification of protected attributes for fairness auditing.

## VII. DISCUSSION

### A. Advantages

The unified platform eliminates context switching between disparate tools, accelerating the exploratory phase. Integration of fairness and drift detection encourages proactive bias assessment rather than post-hoc auditing. The three-tier architecture enables easier maintenance of each component, simplifies deployment, supports development of alternative clients that access the API, and enables future multi-user scenarios with authentication. DuckDB's in-process design eliminates operational complexity compared to traditional databases while delivering competitive performance. The columnar storage model aligns naturally with analytical query patterns common in EDA workflows.

### B. Limitations

Current implementation lacks support for time series analysis, geographic visualizations, and text analytics. While DuckDB handles single-node workloads efficiently, distributed processing capabilities are absent for truly large-scale datasets exceeding memory capacity. The fairness module currently implements only demographic parity; additional metrics such as equalized odds and predictive parity would enhance analytical depth. Drift detection relies solely on PSI; incorporating statistical tests like Kolmogorov-Smirnov or Chi-square would provide complementary perspectives.

### C. Future Enhancements

Based on development experience and observed limitations:

**Dataset handling improvements:** Chunked file uploads for reliable ingestion of multi-gigabyte files, and enhanced dataset merging capabilities beyond the current ZIP-archive-only approach.

**Drift analysis workflow:** Built-in dataset splitting functionality to enable temporal or cohort-based comparisons without requiring external preprocessing. This would streamline the drift detection workflow that currently requires manual data preparation.

**Additional fairness metrics:** Expansion beyond demographic parity to include equalized odds, predictive parity, and disparate impact ratios for more comprehensive fairness assessment.

**Statistical drift tests:** Complement PSI with Kolmogorov-Smirnov tests for numeric features and Chi-square tests for categorical features to provide statistical significance measures.

**Automated insights:** Natural language summaries of detected patterns, biases, and data quality issues to accelerate the exploratory phase and improve accessibility for non-technical stakeholders.

## VIII. CONCLUSION

We presented a comprehensive web-based platform unifying exploratory data analysis, fairness assessment, and drift detection capabilities. The system's three-tier architecture, efficient DuckDB backend, and interactive Streamlit frontend deliver a responsive user experience while maintaining analytical rigor. Performance evaluation demonstrates scalability to datasets exceeding a gigabyte with acceptable query response times for most operations.

By integrating bias detection heuristics and PSI-based drift analysis into the exploratory workflow, the platform encourages proactive identification of data quality issues and potential fairness concerns. The modular architecture facilitates extension with additional analytical

methods while preserving system maintainability.

Future work will expand analytical capabilities through time series support, additional fairness metrics, and automated feature engineering recommendations. Integration with model tracking systems will enable seamless transition from exploration to production deployment. The platform demonstrates that modern data infrastructure components can be composed into user-friendly applications that democratize advanced analytical techniques beyond specialist data science teams.

## ACKNOWLEDGMENT

The authors would like to thank Dr. Jung Soo Lim for guidance and the Computer Science Department at California State University, Los Angeles.

## REFERENCES

- [1] S. Brugman, "Pandas Profiling: Exploratory Data Analysis for Python," 2019. [Online]. Available: <https://github.com/pandas-profiling/pandas-profiling>
- [2] R. K. E. Bellamy et al., "AI Fairness 360: An extensible toolkit for detecting and mitigating algorithmic bias," IBM Journal of Research and Development, vol. 63, no. 4/5, pp. 4:1-4:15, 2019.
- [3] S. Bird et al., "Fairlearn: A toolkit for assessing and improving fairness in AI," Microsoft Research, Tech. Rep. MSR-TR-2020-32, 2020.
- [4] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," ACM Computing Surveys, vol. 46, no. 4, pp. 1-37, 2014.

- [5] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346-2363, 2018.
- [6] M. Raasveldt and H. Mühleisen, "DuckDB: an Embeddable Analytical Database," in *Proc. ACM SIGMOD Int. Conf. Management of Data*, 2019, pp. 1981-1984.
- [7] A. Treuille, T. Teixeira, and K. Weisz, "Streamlit: A faster way to build and share data apps," 2020. [Online]. Available: <https://streamlit.io>
- [8] S. Ramírez, "FastAPI: Modern, fast (high-performance) web framework for building APIs," 2018. [Online]. Available: <https://fastapi.tiangolo.com>
- [9] B. Becker and R. Kohavi, "Adult Census Income," *UCI Machine Learning Repository*, 1996. [Online]. Available: <https://www.cs.toronto.edu/~dave/data/adult/desc.html>