Machine Learning Final Presentation Documentation

Oliver Holmes, Manuel Merino, Mona Ashruf, Alzmani Abdulrahman, Joud Taher, Blanca Valdes

Business Goals and Technical Analysis

The project converts a 8 k-item, partially unstructured marketplace dataset of computer hardware into a pricing- and product-intelligence service delivered through a multi-page web application. Cleaning, semantic alignment and feature consolidation will transform Spanish-language raw feeds into an English-navigable asset that offers buyers transparent benchmarking and gives sellers, and the marketplace itself, evidence-based guidance for inventory and revenue decisions, eliminating manual spreadsheet work.

The application must realise three ML-enabled workflows specified in the brief. Descriptive analytics provide statistical overviews and K-means segmentation of market offerings; predictive analytics deliver LightGBM-based price estimates plus feature-importance explanations; and prescriptive analytics return the k most similar live offers via a K-nearest-neighbour search. These capabilities sit atop a full pipeline covering data collection, preparation, feature engineering, training, validation, deployment and logging, all shown in an English UI while preserving Spanish field content for fidelity.

The solution serves as a self-service decision cockpit: executives track market shifts, category managers verify price competitiveness, and consumers instantly compare configurations. Accelerating the insight-to-action loop is expected to sharpen price accuracy, lift conversion and safeguard margins, while the required user-feedback logger feeds continuous monitoring and retraining to keep model performance aligned with evolving market reality.

Data Analysis and Preparation

The data analysis phase began with a systematic ingest of the raw CSV feed—8,064 listings across 135 Spanish-named fields—directly loaded into a pandas DataFrame. Every header was mapped to a clear, snake_case English term using the provided glossary, ensuring consistency in downstream code and eliminating ambiguity about attribute semantics (e.g. duración_batería → battery_life_hours) .

Mixed-type columns were parsed to isolate pure numeric values and normalize units. Size fields such as "15.6\"" or "300 nits" were split into magnitude and unit, with all lengths converted to centimetres and luminance stripped to floats. Composite specs like "1920×1080" were parsed into width_px and height_px, then used to derive pixel_count and aspect_ratio, which served as foundational predictors for later modeling .

Exploratory data analysis revealed uneven distributions and outliers in key variables. Price exhibited right skew with a long tail of high-end configurations; RAM and storage showed clustering at standard capacities (8 GB, 16 GB, 256 GB, etc.). Histograms, box plots, and pairwise scatter plots guided the application of log transformations on skewed features and capping of extreme values at the 99th percentile, thereby reducing the undue influence of rare super-laptops.

Missingness was handled via a dual approach. Columns exceeding 30% nulls were dropped unless deemed domain-critical (e.g. GPU model). For the remainder, structural absences—such as desktops lacking battery life—were distinguished from true gaps by grouping records by product type (laptop, monitor, all-in-one). Group medians imputed numeric fields, group modes imputed categoricals, and boolean “was_missing” flags preserved patterns where absence itself could signal category or price tier . A final global median/mode fill closed any residual nulls.

The cleaned dataset was then partitioned into training and test sets (80/20 split, random_state=42), stratified by broad product category. This ensured that all subsequent modeling workflows—K-Means clustering, LightGBM regression, and KNN similarity search—operated on representative samples, facilitating reliable cross-validation and out-of-sample evaluation.

Feature Engineering and Feature Selection

Building on the cleaned data, we engineered features to capture latent performance attributes. From raw resolution and screen size, we computed pixel_density (pixels per square inch) to more precisely represent display sharpness. Price metrics such as

price_mean and price_spread distilled variability between listing minimums and maximums, informing both clustering stability and regression targets .

To surface non-linear effects, we created interaction terms like brightness × price_mean and performance ratios such as battery_life_hours / weight_kg. Right-skewed features (price, battery capacity) received log transforms to stabilize variance, improving linear separability for clustering and reducing heteroscedasticity for regression.

Categorical lists—most notably “connectivity” ports (HDMI, USB-C, VGA)—were split into separate binary flags via one-hot encoding. Ordinal features (e.g. processor generation) used label encoding to preserve rank, while high-cardinality categoricals (GPU model) were frequency-encoded to avoid sparse expansions. All transformations were encapsulated in a scikit-learn ColumnTransformer, ensuring identical pipelines for K-Means, LightGBM, and KNN .

With a rich feature set in place, we applied selection techniques to reduce noise and multicollinearity. A variance threshold filter removed near-constant columns; pairwise correlation analysis (dropping one of each pair with Pearson $|\rho| > 0.9$) collapsed redundant specs (e.g. width vs. diagonal). For the clustering workflow, we further applied PCA to capture 95% of variance in fewer components, enhancing cluster compactness and visualization clarity.

For the regression and KNN pipelines, univariate statistical tests (F-tests for continuous vs. price) and model-based importance metrics (LightGBM feature importances, Lasso coefficients) guided the final predictor set. We consciously retained features demonstrating consistent importance across folds, while category-aware pruning ensured that attributes critical to laptops (battery life) but irrelevant to monitors (touch capability) were handled appropriately.

These engineered and selected features formed the backbone of all three modeling approaches, balancing expressiveness with parsimony to optimize both predictive performance and computational efficiency.

Algorithm Selection, Model Training and Validation and Model Selection

The project implements three complementary modeling approaches, K-Means clustering for segmentation, LightGBM regression for price prediction, and K-Nearest Neighbors for similarity-based recommendations, to address descriptive, predictive, and prescriptive analytics goals, respectively. Each algorithm was chosen for its strengths in the given task: K-Means for its simplicity and interpretability in uncovering market segments; gradient-boosted trees (LightGBM) for their ability to model non-linear relationships and handle heterogeneous feature types in price estimation; and KNN for its intuitive distance-based retrieval of products most similar to a user's specified configuration.

For the K-Means workflow, a preprocessing pipeline first imputes missing numeric values using median imputation, encodes categorical variables via one-hot encoding, and standardizes all features to zero mean and unit variance. A ColumnTransformer groups these steps, and the transformed feature matrix feeds into scikit-learn's KMeans(n_clusters=2, random_state=42, n_init='auto'). The choice of k=2k=2k=2 was informed by initial exploratory runs,balancing compactness of clusters against interpretability,and validated via internal metrics (silhouette, Calinski-Harabasz, Davies-Bouldin). Once trained, each listing is assigned a cluster label, producing two coherent groups (e.g., basic vs. high-end hardware) that form the basis for dashboard segmentation.

The LightGBM price-prediction pipeline mirrors the clustering flow in its treatment of heterogeneous data. After isolating a target variable (precio_mean), the notebook defines numerical and categorical feature sets, imputes and scales numerical inputs, one-hot encodes categoricals, and assembles a full Pipeline ending in LGBMRegressor(). A grid search over hyperparameters (number of estimators, learning rate, max depth, etc.) is wrapped in GridSearchCV(cv=5), optimizing for mean squared error. The best hyperparameter combination is selected based on cross-validated performance on the training folds, then retrained on the full training set to yield the final regression model.

For the KNN regressor,used alongside the NearestNeighbors model,the notebook again constructs a preprocessing pipeline identical to that in LightGBM, ensuring comparability. A train/test split (80/20, random_state=42) partitions the data, and KNeighborsRegressor(n_neighbors=5) is fitted to the processed training set. Here,

k=5k=5k=5 was chosen as a standard starting point, trading off locality of neighborhood against stability of predictions; future iterations could tune this via grid search or cross-validation if needed. The resulting regressor provides point estimates for price, complementing the tree-based model.

Separately, the prescriptive workflow leverages NearestNeighbors(n_neighbors=6, metric='minkowski') to build a similarity index: by fitting to the same processed feature space (excluding the target), the model can retrieve the five closest peers for any query listing. This purely unsupervised neighbor search underpins the “Similar Offers” page, surfacing real market listings that match a user’s input configuration. The combined KNN regressor and NearestNeighbors pipelines thus deliver both a predictive and a descriptive facet of similarity.

Model selection across these three flows balances quantitative performance with interpretability and deployment simplicity. Clustering uses fixed k=2k=2k=2 for clarity, regression leverages automated hyperparameter tuning to maximize out-of-sample accuracy, and KNN defaults to a moderate neighborhood size to ensure responsive similarity lookups. Each final model is serialized (via joblib) alongside its preprocessing pipeline, guaranteeing reproducibility and seamless integration into the Streamlit application.

Performance Metrics Estimation

To quantify model quality, each workflow computes appropriate metrics reflecting its task: clustering validity measures for K-Means, regression error and explained variance for price models, and nearest-neighbor distance characteristics for similarity retrieval. These metrics not only assess standalone performance but also guide comparative evaluation and possible retraining triggers in production.

In the clustering workflow, three internal validity indices are computed for k=2k=2k=2: the silhouette score (~0.42) assesses cohesion vs. separation in cluster assignments, the Calinski-Harabasz index (~3232.8) gauges the ratio of between-cluster dispersion to within-cluster dispersion, and the Davies-Bouldin index (~0.56) measures average “similarity” (lower is better) between clusters. Together, they confirm that the two-cluster solution yields reasonably distinct groups, justifying its selection for market segmentation.

For the LightGBM regressor, out-of-sample performance is reported on the held-out test set: mean squared error (MSE), root mean squared error (RMSE ≈ 760 EUR), and R-squared (≈ 0.66) indicate that the model explains around two-thirds of price variance with typical prediction errors under 800 EUR. Training-set metrics (RMSE ≈ 684 EUR, R2≈0.69R^2≈0.69R2≈0.69) are slightly stronger, suggesting a modest degree of over-fitting but acceptable generalization given the real-world variability in hardware pricing.

The KNN regression model yields its own test RMSE (≈ 820 EUR) and R2R^2R2 (≈ 0.62), performance that is somewhat weaker than LightGBM's but still in a comparable range. This result highlights the bias–variance tradeoff inherent in KNN's fixed-neighborhood averaging: while simple, it can struggle with high-dimensional, heterogeneous feature spaces compared to tree-based learners. Nonetheless, its errors remain within an acceptable window for exploratory pricing estimates.

In the prescriptive NearestNeighbors setup, average neighbor distances and their distribution across the dataset are examined to ensure that retrieval remains meaningful (i.e., that "closest" items are indeed similar in feature space). While no single numeric metric captures user satisfaction directly, inspecting these distance histograms,and sampling specific lookup results,confirms that the five nearest offers share core specifications, validating the model for recommendation.

Collectively, these performance metrics inform both model selection and anticipated user experience: K-Means clusters are sufficiently cohesive to support segmentation, LightGBM delivers the most accurate price predictions for end-users, and KNN neighbor lists provide transparent, intuitive similarity recommendations. Future monitoring will track these metrics over time, triggering retraining whenever out-of-sample error or cluster validity degrades beyond predetermined thresholds.

Model Deployment and Model Serving

The deployment strategy follows the assignment's "browser / local server" mandate: the cleaned dataset, the three trained models, all Python source and a requirements.txt are zipped together with the Streamlit code so that evaluators can launch the web app offline with a single local command, without provisioning any cloud resources. Packaging everything for execution on a standard laptop satisfies the deliverable list that explicitly calls

for “data, code, generated models and streamlit application” and keeps reproducibility straightforward for grading.

Front-End Application

The Computer Analytics Dashboard is a web-based application designed to support users in exploring and evaluating computer market offerings, specifically laptops. It delivers a structured flow through descriptive, predictive, and prescriptive analytics, enabling both casual users and analysts to understand pricing trends, identify patterns, predict values, and receive tailored recommendations. The frontend is developed to be clean, intuitive, and responsive, with all interface labels in English while preserving the original Spanish values from the source dataset in data-driven fields.

Application Layout and Navigation

The interface is divided into four main sections, accessible through a fixed sidebar on the left. These sections correspond to the application's core functionality: Overview, Segmentation, Prediction, and Similar Offers. Each page is self-contained, offering interactive components and visualizations relevant to its analytical role. This separation promotes usability and reflects the Descriptive–Predictive–Prescriptive model underpinning the application.

Overview Page

Upon landing on the application, users are greeted by the Overview page. This section serves as a visual introduction to the marketplace by summarizing key statistics. A bar chart displays the overall price distribution, revealing how many laptops fall within various price brackets. Alongside this, a frequency chart depicts the distribution of screen sizes in the market. Additional visualizations include a categorical price breakdown across the most common product types, represented with a combination of percentiles, mean, and outliers and a donut chart showing the top 10 most represented brands in the dataset.

All visualizations use English headers and legends, but the actual product categories and brand names appear in their original Spanish form. This preserves the dataset’s integrity while maintaining a clear and internationally understandable interface.

Segmentation Page

The Segmentation page allows users to explore product clustering based on the K-Means algorithm. A scatter plot visually separates laptops into color-coded clusters, determined by user-selected feature dimensions for the x and y-axes (example: price and RAM) and the number of clusters (K). Below the chart, detailed summaries of each cluster are displayed, including the number of laptops, average price, average RAM, and screen size. Additionally, the most common product type is shown per cluster to assist users in interpreting group characteristics.

This section helps users identify product groupings, outliers, and potential market niches. The design supports comparative analysis by offering both a high-level visual overview and a precise numerical breakdown.

Prediction Page

The Prediction section enables users to estimate the price of a laptop based on a wide range of technical specifications. The form includes fields for memory size, storage, processor type, number of cores, GPU model, screen size, and several other physical and software attributes. Sliders, dropdowns, and text inputs are used appropriately to ensure that the interface remains user-friendly while accommodating diverse input types.

After a user submits the form, the application sends the data to a Light Gradient Boosting Machine (LightGBM) model that returns a price prediction. Alongside the predicted value, a horizontal bar chart indicates the relative importance of each feature in the model's decision, helping users understand what specifications most strongly influence price. A supplementary scatter plot is included to show real-world correlations, such as the relationship between RAM and price, offering further context.

Similar Offers Page

The final page of the application is dedicated to prescriptive analytics. Here, users can specify a configuration of interest and retrieve similar products from the dataset using a K-Nearest Neighbors (KNN) algorithm. The input form closely mirrors the one used in the Prediction page, promoting a consistent user experience. After submitting a query, the system returns a list of products with comparable specifications, ordered by similarity score.

Each recommended product is displayed with its model name, predicted price, actual price, and key specs like RAM, storage, and screen size.

Beneath the results, a detailed comparison table offers a row-by-row breakdown of the matched items. This table is organized to allow easy inspection of differences in features and prices across alternatives. This page helps users evaluate trade-offs and make informed decisions by comparing offers that match their priorities.

Interface and Localization

The frontend is fully English in terms of UI components, instructions, and labels. However, it preserves all original field values in Spanish, such as product categories (example: “Portátil multimedia”) and brand names, as they appear in the source dataset. This choice avoids mislabeling and maintains data authenticity while offering a globally readable interface.

Technical Integration

All data requests from the frontend are handled via API calls using the Axios library. These calls interact with backend models to fetch predictions and similar item recommendations. For example, when a user submits specifications for price prediction, the frontend issues a POST request to a backend endpoint (e.g., `/predict`) and receives a single price value in return. Similarly, the recommendation engine is triggered via another POST request (e.g., `/similar`) that returns a ranked list of matches.

Before making requests, the frontend ensures that all required fields are populated either by the user or through fallback defaults. This guarantees reliable model execution and prevents form submission errors. Form validation and input formatting are handled gracefully, promoting stability and usability.

Build and Deployment

To set up the frontend locally, developers should install the necessary dependencies using npm install. The development server can be started using npm run dev, and production builds are created with npm run build. The application uses environment variables to configure base URLs for backend APIs, which should be defined in a .env file at the root of the project.

Conclusion

This project successfully delivers a full-stack machine learning solution that transforms raw, semi-structured marketplace data into actionable insights through an interactive web application. Leveraging a robust data processing pipeline and state-of-the-art modeling techniques, the system enables descriptive, predictive, and prescriptive analytics tailored to the needs of different user types, from executives and category managers to end consumers.

Throughout the project, we gained hands-on experience with critical data science practices, including exploratory data analysis (EDA), feature engineering, and handling missing values. By working directly with real-world data, we learned how to identify patterns, uncover data quality issues, and apply domain-relevant transformations that significantly improved model performance. We developed a deeper understanding of how thoughtful data preprocessing can directly influence the reliability of downstream analytics.

Our team also grew in collaboration and project management skills, learning to coordinate effectively across different components of the pipeline—from backend data processing and machine learning modeling to frontend integration and user experience design. We adopted tools and workflows that supported code versioning, reproducibility, and peer reviews, preparing us for real-world team environments.

The final system integrates LightGBM for price prediction, K-Means for market segmentation, and KNN for similarity-based recommendations, ensuring that the application is both accurate and responsive to diverse user needs. Each component was selected based on thorough experimentation and validation, combining performance with interpretability.

Designed with usability in mind, the application features an English-language interface while preserving the semantic integrity of the original Spanish-language marketplace data. It replaces manual spreadsheet-based workflows with a scalable, intelligent platform that enhances price transparency, competitiveness, and operational efficiency.

Further, the inclusion of model explainability, user feedback logging, and localized insights positions the solution not just as a functional product, but as a robust foundation for future development. Overall, this project reflects our ability to apply machine learning in a practical, team-based setting—transforming complex data into real value for end users.