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 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.

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 (example: 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=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 (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), it 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=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.

Model selection across these three flows balances quantitative performance with interpretability and deployment simplicity. Clustering uses fixed k=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.

Performance Metrics Estimation

The performance evaluation of the three modeling approaches relied on metrics and visual diagnostics tailored to each task, ensuring that clustering coherence, prediction accuracy, and similarity retrieval transparency could all be assessed appropriately.

For K-Means clustering, the high-dimensional feature space was first reduced to two principal components via PCA. The resulting cluster assignments were then plotted on the same PC1–PC2 axes, with points colored by their assigned cluster and a separate graph was plotted with the points labeled according to their original tipo_de_producto categories. This showed that our EDA was successful given that the labels in tipo_de_producto stayed neatly within one of the clusters, without mixing across both of them. This visual inspection confirmed that the two clusters align meaningfully with intuitive product groupings, cluster 0 being 0 being top tier computers and Cluster 1 being lower-end computers, providing qualitative validation of segmentation.

The LightGBM regression model underwent five-fold cross-validation during hyperparameter tuning, optimizing for mean squared error. Final evaluation on the held-out test set produced a root mean squared error of approximately 520 EUR and an R-squared of 0.81, indicating that over eighty percent of the observed price variance is explained. Feature importance scores extracted from the trained ensemble highlighted processor generation, RAM capacity,

and display pixel density as the most influential predictors, guiding interpretation and user-facing explanations.

The K-Nearest Neighbors regressor, configured with k = 5, was assessed as a baseline predictive model by plotting predicted versus actual prices for the test set and examining the residual distribution. The scatter plot demonstrated that the neighbor-averaging approach captures the overall price trend but shows increased spread at the costlier and cheaper extremes. The residual plot revealed no systematic bias, supporting its use as a transparent reference for similarity-based recommendations.

Viewed together, these quantitative error metrics and qualitative visual diagnostics confirm that the K-Means clusters are coherent, that the LightGBM model achieves strong predictive accuracy, and that the KNN baseline offers an intuitive complement for prescriptive “five nearest offers.” This blend of performance measures ensures a robust, multi-faceted understanding of model behavior across descriptive, predictive, and prescriptive tasks.

Model Deployment and Model Serving

The application is deployed using a cloud-native architecture built on Google Cloud. It features a React-based frontend UI for user interaction, while the backend is developed in Python using libraries such as scikit-learn, pandas, and matplotlib for data processing, exploratory data analysis (EDA), and model training. The machine learning models—LightGBM, KMeans, and KNN—are trained and serialized using Joblib. These models are hosted as APIs via Google Cloud Functions, with deployment automated through GitHub Actions as part of a CI/CD pipeline triggered on pushes to the main branch. The serialized models are stored in Google Cloud Storage and are loaded by the APIs as needed. This setup enables scalable, maintainable model serving through cloud-based APIs, aligning with modern deployment best practices.

Front-End Application

The Computer Analytics Dashboard is a modern, cloud-native web application designed to analyze and interact with laptop market data. It offers users a comprehensive interface to explore price distributions, product segmentation, and predict laptop prices based on

technical specifications. The frontend is built using React.js, providing a dynamic and responsive user interface that integrates seamlessly with backend APIs hosted on Google Cloud.

The frontend is organized into three main sections: Overview, Segmentation, and Prediction. Navigation is provided through a fixed sidebar, which maintains consistent access to each module. The Overview page displays market trends through a set of visual components. These include a bar chart showing price distribution ranges, a screen size frequency chart, a box plot for price variation across top product types, and a donut chart highlighting the top 10 brands. Each visualization is designed to deliver clear insights into different dimensions of the laptop market.

The Segmentation page allows users to explore machine-learned product clusters. It features a scatter plot that visualizes two clusters formed using KMeans clustering on PCA-reduced data. These clusters help differentiate product groupings such as laptops and desktops. A second PCA plot is color-coded by product type, showing how different devices are distributed across the principal component axes. At the bottom, a cluster statistics section visually maps clusters to known categories, providing interpretability of the clustering process.

The Prediction page provides an interactive specification form where users can input device details such as RAM, storage, CPU model, core count, GPU, operating system, Bluetooth version, battery capacity, and screen specifications. The form dynamically adjusts numeric values through sliders and categorical selections. Upon submission, the frontend communicates with Google Cloud Functions that host serialized machine learning models, sending the input parameters and retrieving a price prediction. In addition to the prediction form, two analytical visualizations are displayed: a feature importance chart indicating which input variables most influence the model’s predictions, and a scatter plot showing the real-world relationship between RAM size and price.

The backend architecture supporting the dashboard is developed in Python, leveraging libraries such as scikit-learn, pandas, matplotlib, LightGBM, KMeans, and KNN. Models are trained offline and serialized using Joblib. These models are stored in Google Cloud Storage and are dynamically loaded by serverless APIs deployed through Google Cloud Functions. The system is designed with scalability and maintainability in mind. Code updates are integrated into a CI/CD pipeline powered by GitHub Actions, which automatically deploys updates to the APIs and model infrastructure upon each push to the main branch.

The frontend and backend communicate via REST APIs. When users interact with the prediction form or request data visualizations, the frontend sends structured input to the cloud function endpoints. These endpoints process the request using the appropriate model, return prediction values or chart data, and allow the frontend to update accordingly. Visualization assets such as charts may be rendered server-side using matplotlib and returned as images or chart-ready data.

The user interface is largely in English, though some labels and feature names (such as those in feature importance and product types) remain in Spanish. It is recommended to introduce language standardization or localization features to improve accessibility for global users. Additionally, while the prediction form is functional and integrated, it currently lacks a visible output box showing the predicted price, which is a recommended enhancement for user clarity.

The current frontend implementation is suitable for expansion. Suggested enhancements include adding a predicted price output section, enabling the comparison of multiple configurations, integrating a toggle for language selection, and displaying confidence intervals for predictions. For improved interpretability, additional visualizations such as price versus storage or CPU performance could be integrated alongside the existing RAM correlation chart.

This dashboard combines modern machine learning deployment practices with intuitive visual analytics, making it a robust tool for both technical and non-technical users. The React frontend, coupled with scalable backend services on Google Cloud, ensures the application remains responsive, maintainable, and ready for further enhancements.

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.

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.