University of East London

AIN SHAMS UNIVERSITY

FACULTY OF COMPUTER AND INFORMATION SCIENCES

AI Project

Project idea: **Smart Phones Price Prediction**

Team name: **Gradient Decents**

Team Members:

1-Mina Gerges Samaan	(Level 3-AI)	20231700252
2-Mina Maged Naem	(Level 3-AI)	20231700254
3-Youssef Ayman Abdulwahab	(Level 3-AI)	20231700261
4-Hussien Mohamed Ahmed	(Level 3-CSec)	20231700316
5-Abdulrahman Mahmoud Abdelmonem	(Level 3-CSec)	20231700322
6-Omar Sameh Abdelmonem	(Level 3-CSec)	20231700324

1. Project Overview

The goal of this project is to build a machine learning system that can sort cell phones into "expensive" and "non-expensive" groups based on their technical specs. The system was built using a method to deal with missing values and data cleaning.

1) Data Cleaning

- We checked the all the numerical columns are actually numerical as in (Float, int).
- We standardized all text to lowercase to avoid redundant categories caused by casing differences, like “POCO” and “Poco”.
- Unlike simple deletion, we handled Processor_Series column dynamically; Processor_Series was cleaned from non-numerical by replacing it with the mean of the column in the training dataset:
    - During training:
      Calculating the Mean of every single column and store it in the imputation values.
    - During test:
      Use the Mean of the training set.

- Performance_Tier was found to have 71% of the rows with the value “unknown”. The column was preserved and improved through predictive imputation using “HistGradientBoostingClassifier” using other features in the same dataset: [Clock_Speed_GHz, RAM Size GB, Core_Count, Refresh_Rate, fast_charging_power, Screen_Size, Resolution_Width, series_encoded, ram_tier_numeric] to maintain its high predictive power.
- We have capped the unrealistic entries in the “RAM Size GB” and “Storage Size GB” to a minimum value of 0.5 and 4 respectively.

2) Encoding Strategies

Since machine learning models require numerical input, we applied different encoding strategies based on the nature of the features:

- **Manual Binary Encoding:** we’ve manually mapped the target variable (price) and several hardware features like [NFC, memory_card_support, 5G, IR_Blaster, 4G, Dual_Sim, Vo5G] yes or no to value of 1 and 0 respectively.

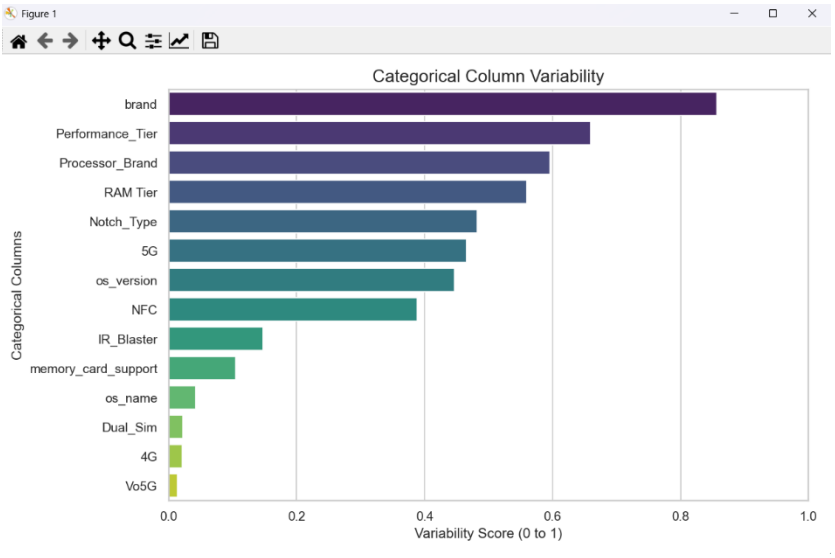
- **Ordinal Encoding:** For features with a logical hierarchy, such as RAM Tier and Performance_Tier, we used an OrdinalEncoder to map categories from "budget" to "flagship" as integers (0 to 4), ensuring the model recognizes the quality progression.

"unknown"→0 “budget"→1
"mid-range"→2 "high-end"→3
"flagship"→4

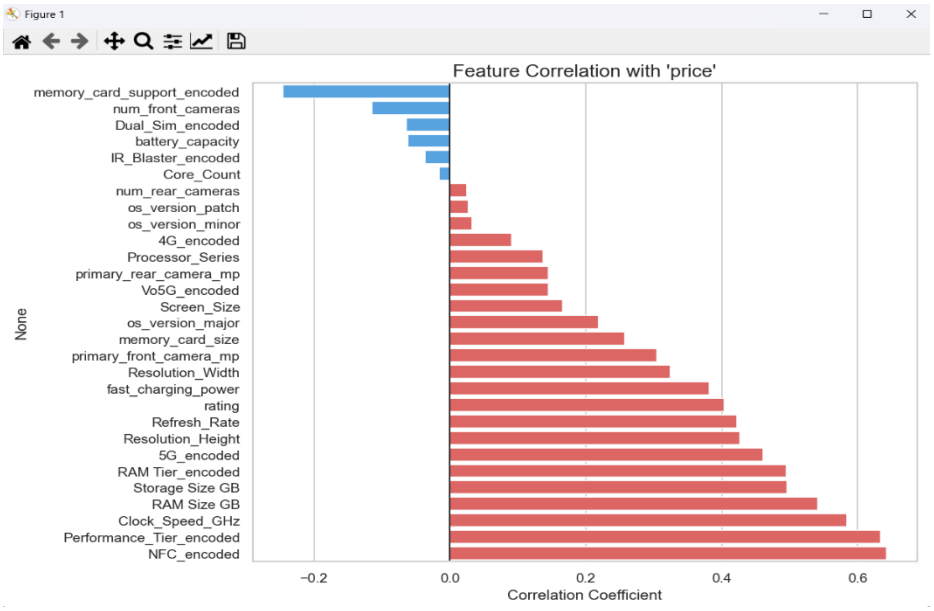
- **One-Hot Encoding:** For nominal features without a ranking, such as brand, Processor_Brand, and os_name, and Notch_Type. We used one hot encoding to prevent the model from assuming an incorrect mathematical relationship between different brands. We have also prevented new columns from raising in the test.

3) Analysis and visualization

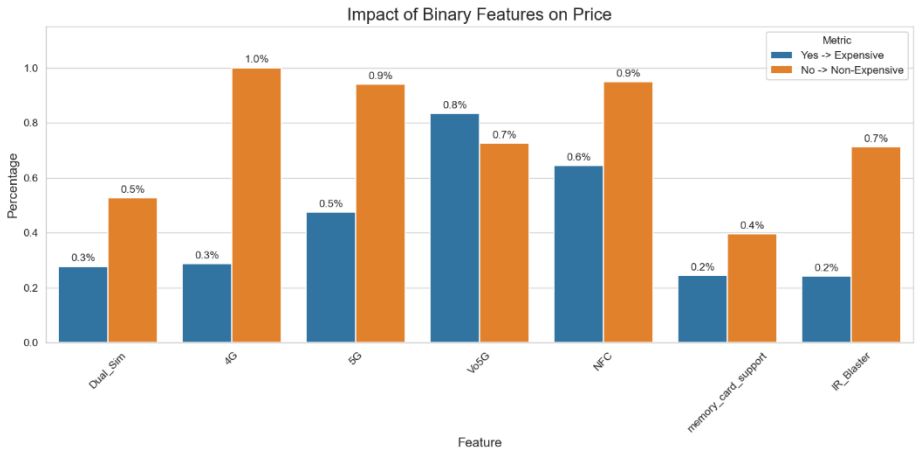
- To ensure the model learns from a clean "signal" rather than "noise," we have measured the variance across different features.



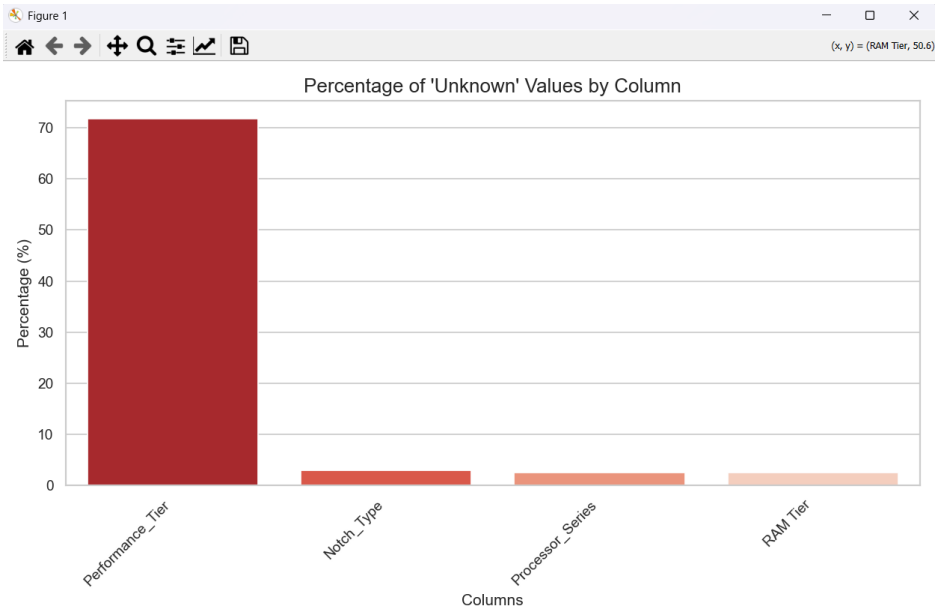
- We have measured the correlation between different features and the price



- We have visualized the relationship between the Boolean columns and the price.



- The Unique function in helper.py returns the unique values in each column
- show.py returns all of the values in a specific column
- The number of missing values measurement



4) Other Preprocessing Functionality

- Split “os_version” into major, minor, patch for example Old----------Removing v----Major---- Minor------Patch v10.1----------10.1----------10 --------- 1----------0 v2----------2 ----------2 --------- 0----------0
- Converting “memory_card_size” into TBs to GBs and removing “GB” and “TB”
- Standard Scaling given that huge differences between the numerical values, we had to standardize it.
- We’ve employed Sequential Feature Selection (SFS). This iterative process builds the model by adding one feature at a time, selecting only those that improve accuracy, until the optimal set of 30 features is reached. We utilize a lightweight Random Forest estimator to ensure the selection process remains computationally efficient.

2. Models Used

- **Logistic Regression**:

  A linear classifier that is fast to train and easy to interpret. It indicates the data is linearly separable, and we can get initial instincts about data without needing computationally expensive algorithms.

```
param_grid = [
    {
        'solver': ['liblinear'],
        'penalty': ['l1', 'l2'],
        'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
        'max_iter': [2000]
    },
    {
        'solver': ['lbfgs', 'newton-cg'],
        'penalty': ['l2'],
        'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
        'max_iter': [2000]
    }
]
```

- **SVM**:

  Particularly effective if the price categories have clear, distinct "gaps" between them but it’s computationally intensive.

```
param_grid = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
    'gamma': ['scale', 'auto', 0.1, 0.01, 0.001],
    'degree': [2, 3, 4]
}
```

- **KNN**:

  It depends on similar phone prices will have similar specifications (features). It doesn't "learn" a model but rather memorizes the training data.

```
param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11, 15],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}
```

- **Random Forest Classifier**:

  It builds many Decision Trees and merges their results to get a more accurate and stable prediction. It reduces the risk of overfitting that a single Decision Tree often suffers from.

```
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'criterion': ['gini', 'entropy']
}
```

- **XGBoost**:

  Unlike Random Forest (which builds trees in parallel), XGBoost builds trees sequentially, where each new tree tries to correct the errors of the previous one.

```
param_grid = {
    'n_estimators': [100, 300, 600],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 6, 10],
    'subsample': [0.8, 1.0]
}
```

---

3. Hyperparameter tunning

We've used cross-validation tightly integrated into the hyperparameter tuning process via GridSearchCV. By defining cv=5, we forced the algorithm to train and validate every parameter combination on multiple distinct folds of the data. This methodology is crucial because it ensures that the "best parameters" selected are mathematically robust and generalizable, rather than being the result of a lucky data split or overfitting to a specific subset of the training examples.

4. Final Conclusion

- This project successfully established a Machine Learning capable of classifying smartphone prices. By using simple data cleaning and advanced techniques, specifically the predictive imputation of the "Performance_Tier", we significantly maximized the utility of our available data, preserving over 70% of rows that would have otherwise been discarded or filled with noise.

- The comparative approach to modeling was critical to the accuracy. Starting with Logistic Regression provided a solid baseline and confirmed that price prediction involves complex, non-linear relationships that simpler models cannot fully capture. This justified our progression to more complex methods like Random Forest and XGBoost, which effectively handled these complexities.

- Finally, the rigorous application of GridSearchCV with Cross-Validation ensures that our results are not merely due to a fortunate data split. The final model is mathematically robust, generalizable to unseen data, and ready for deployment as a reliable pricing tool.