TALLER MODEL TRAINING

MARÍA FERNANDA TELLO VERGARA

2225338

JAVIER ALEJANDRO VERGARA

ETL

UNIVERSIDAD AUTÓNOMA DE OCCIDENTE

OCTUBRE 19 2024

CONTEXT

The goal of this project is to develop a machine learning model capable of predicting whether a product listed on a marketplace is new or used, using the MLA_100k.jsonlines dataset. After a thorough feature analysis, several machine learning models were implemented, including Random Forest, Gradient Boosting, K-Nearest Neighbors, Logistic Regression, and XGBoost. Accuracy metrics were evaluated, to determine the performance of each model. This analysis seeks to optimize the classification of product condition, which impacts pricing and sales strategies for sellers.

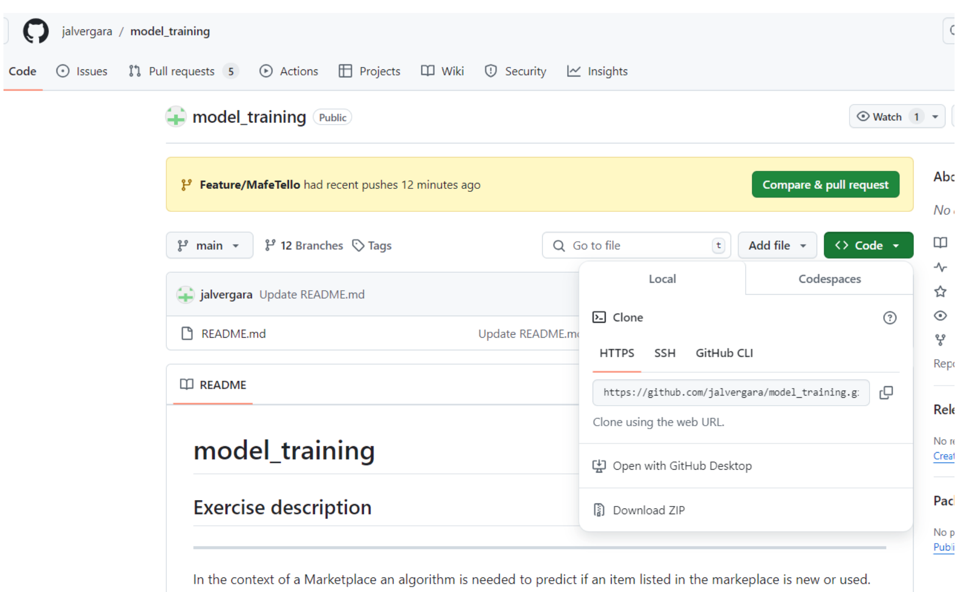
TOOLS

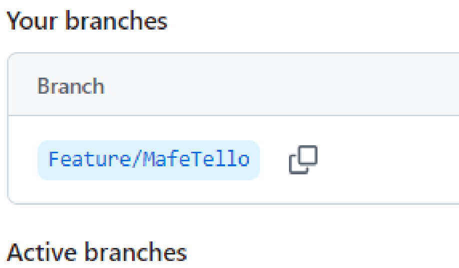
- Python
- Pandas
- NumPy
- Scikit-learn
- XGBoost
- LogisticRegression
- KNeighborsClassifier
- RandomForestClassifier
- GradientBoostingClassifier
- Matplotlib y Seaborn
- Jupyter Notebook/VSCode
- Joblib

STEP BY STEP

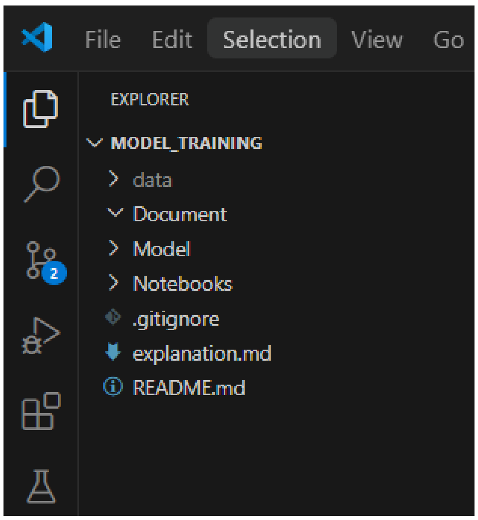
First, we clone this repository: https://github.com/jalvergara/model_training.git

and create a branch with the name: Feature/MafeTello.



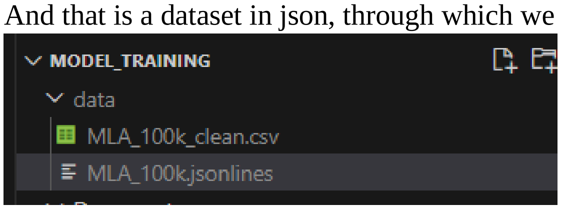


We created these folders to start working on the project.



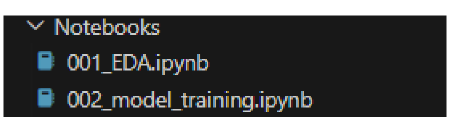
In the data folder we can see the file that we downloaded from the following link: https://drive.google.com/file/d/1mzk9g5StOsIvi8TBIVX5CObAsrAOWhcL/view

And that is a dataset in json, through which we are going to create a predictive model.

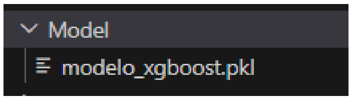


Additionally, it can be observed that we have another clean dataset after performing preprocessing, EDA and transformations on the data.

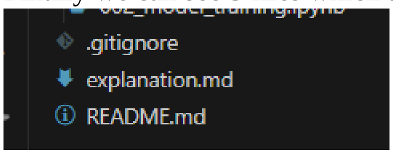
In the Notebooks folder we can see two files: 001_EDA.ipynb, where we perform some transformations on the data and visualizations. And 002_model_training.ipynb, where we perform the entire process to define the most appropriate model, and then download the .plk file.



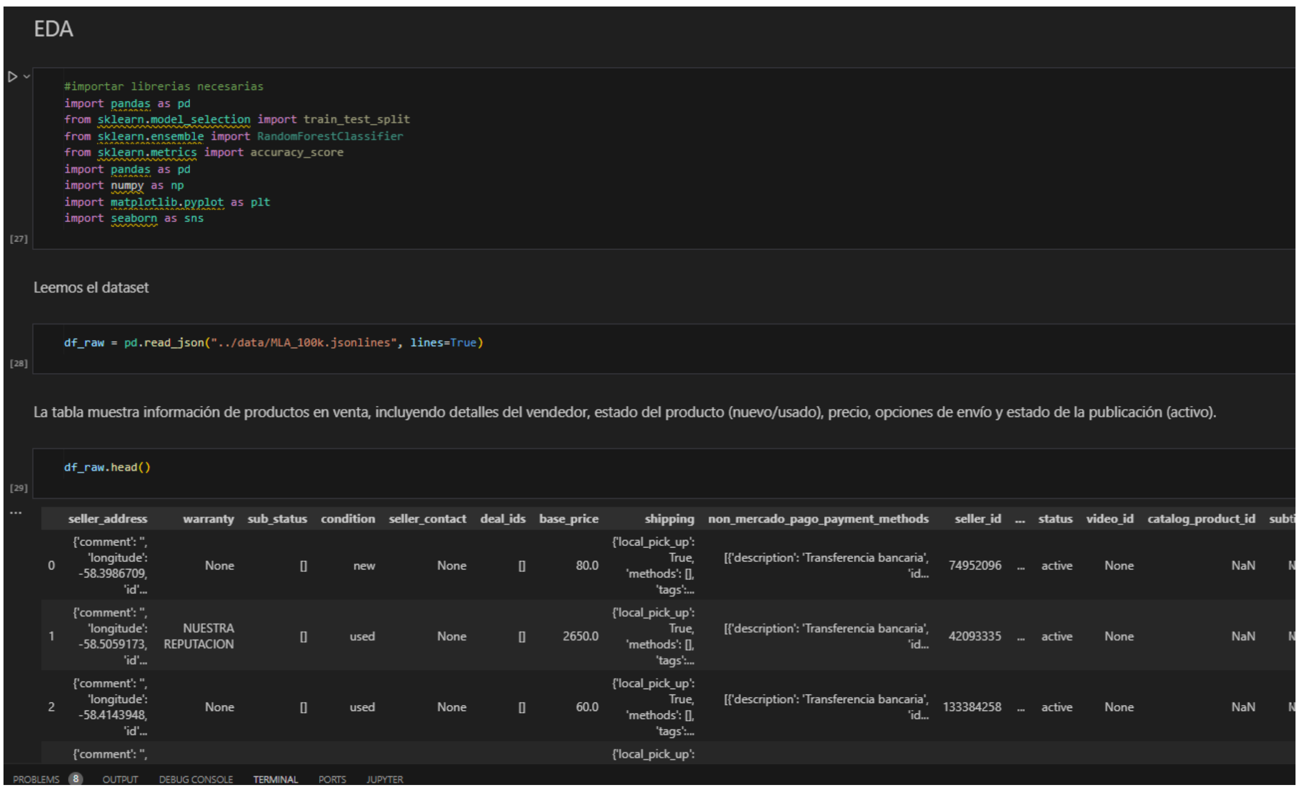
Then we can see in the Model folder the file that is generated after selecting the best model based on the evaluation metrics.



Finally we can see 3 files which are .gitignore, explanation.md and README.md.



EDA



EDA

```
#importar librerias necesarias
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Leemos el dataset

```
df_raw = pd.read_json("../data/MLA_100k.jsonlines", lines=True)
```

La tabla muestra información de productos en venta, incluyendo detalles del vendedor, estado del producto (nuevo/usado), precio, opciones de envío y estado de la publicación (activo).

```
df_raw.head()
```

	seller_address	warranty	sub_status	condition	seller_contact	deal_ids	base_price	shipping	non_mercado_pago_payment_methods	seller_id	...	status	video_id	catalog_product_id	subti
0	{'comment': '', 'longitude': -58.3986709, 'id'...	None	[]	new	None	[]	80.0	{'local_pick_up': True, 'methods': [], 'tags':...	[{'description': 'Transferencia bancaria', 'id...	74952096	...	active	None	NaN	N
1	{'comment': '', 'longitude': -58.5059173, 'id'...	NUESTRA REPUTACION	[]	used	None	[]	2650.0	{'local_pick_up': True, 'methods': [], 'tags':...	[{'description': 'Transferencia bancaria', 'id...	42093335	...	active	None	NaN	N
2	{'comment': '', 'longitude': -58.4143948, 'id'...	None	[]	used	None	[]	60.0	{'local_pick_up': True, 'methods': [], 'tags':...	[{'description': 'Transferencia bancaria', 'id...	133384258	...	active	None	NaN	N
	{'comment': '',							{'local_pick_up':							

In this part we can observe important information about the dataset.

[30]
```
df_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 48 columns):
 #   Column                         Non-Null Count   Dtype
---  ------                         --------------   -----
 0   seller_address                 100000 non-null  object
 1   warranty                       39104 non-null   object
 2   sub_status                     100000 non-null  object
 3   condition                      100000 non-null  object
 4   seller_contact                 2219 non-null    object
 5   deal_ids                       100000 non-null  object
 6   base_price                     100000 non-null  float64
 7   shipping                       100000 non-null  object
 8   non_mercado_pago_payment_methods 100000 non-null  object
 9   seller_id                      100000 non-null  int64
 10  variations                     100000 non-null  object
 11  location                       100000 non-null  object
 12  site_id                        100000 non-null  object
 13  listing_type_id                100000 non-null  object
 14  price                          100000 non-null  float64
 15  attributes                     100000 non-null  object
 16  buying_mode                    100000 non-null  object
 17  tags                           100000 non-null  object
 18  listing_source                 100000 non-null  object
 19  parent_item_id                 76989 non-null   object
...
 46  sold_quantity                  100000 non-null  int64
 47  available_quantity             100000 non-null  int64
dtypes: bool(2), datetime64[ns, UTC](2), float64(7), int64(4), object(33)
memory usage: 35.3+ MB
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

[31]
```
df_raw.describe().T
```

	count	mean	std	min	25%	50%	75%	max
base_price	100000.0	5.252423e+04	8.623127e+06	0.84	90.0	250.0	8.000000e+02	2.222222e+09
seller_id	100000.0	8.425269e+07	5.497257e+07	1304.00	39535905.5	76310627.0	1.325659e+08	1.946906e+08
price	100000.0	5.252433e+04	8.623127e+06	0.84	90.0	250.0	8.000000e+02	2.222222e+09
official_store_id	818.0	2.064438e+02	1.282530e+02	1.00	84.0	216.0	3.127500e+02	4.460000e+02
differential_pricing	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
original_price	143.0	1.593342e+03	2.245798e+03	120.00	449.0	858.0	1.500000e+03	1.399900e+04
catalog_product_id	11.0	3.727643e+06	1.884698e+06	94404.00	3050701.5	5093232.0	5.103216e+06	5.434513e+06
subtitle	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
initial_quantity	100000.0	3.509337e+01	4.210762e+02	1.00	1.0	1.0	2.000000e+00	9.999000e+03
sold_quantity	100000.0	2.396990e+00	4.268508e+01	0.00	0.0	0.0	0.000000e+00	8.676000e+03
available_quantity	100000.0	3.484238e+01	4.208084e+02	1.00	1.0	1.0	2.000000e+00	9.999000e+03

In this image we can see the dimensions of the original dataset.

Dimensiones del dataset

[33]
```
print(f'Número de filas: {df_raw.shape[0]}')
print(f'Número de columnas: {df_raw.shape[1]}')
```

```
Número de filas: 100000
Número de columnas: 48
```

Nombres de columnas y tipos de datos

```python
print(df_raw.columns)
print(df_raw.dtypes)
```

[34]

```
Index(['seller_address', 'warranty', 'sub_status', 'condition',
       'seller_contact', 'deal_ids', 'base_price', 'shipping',
       'non_mercado_pago_payment_methods', 'seller_id', 'variations',
       'location', 'site_id', 'listing_type_id', 'price', 'attributes',
       'buying_mode', 'tags', 'listing_source', 'parent_item_id',
       'coverage_areas', 'category_id', 'descriptions', 'last_updated',
       'international_delivery_mode', 'pictures', 'id', 'official_store_id',
       'differential_pricing', 'accepts_mercadopago', 'original_price',
       'currency_id', 'thumbnail', 'title', 'automatic_relist', 'date_created',
       'secure_thumbnail', 'stop_time', 'status', 'video_id',
       'catalog_product_id', 'subtitle', 'initial_quantity', 'start_time',
       'permalink', 'geolocation', 'sold_quantity', 'available_quantity'],
      dtype='object')
seller_address                       object
warranty                             object
sub_status                           object
condition                            object
seller_contact                       object
deal_ids                             object
base_price                          float64
shipping                             object
non_mercado_pago_payment_methods     object
seller_id                             int64
variations                           object
location                             object
...
geolocation                          object
sold_quantity                         int64
```

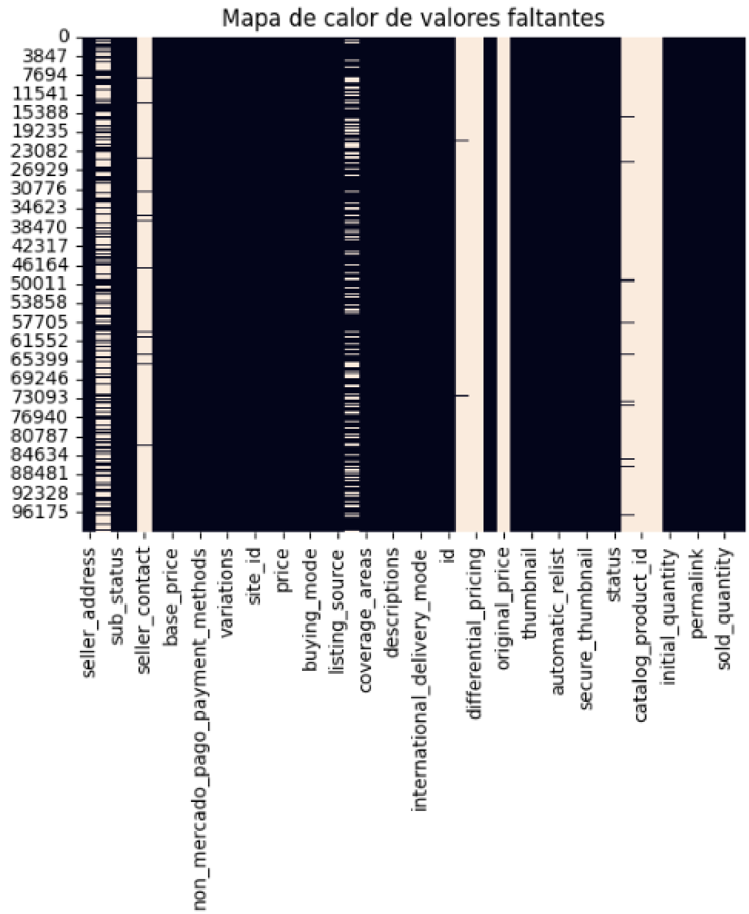
Data cleansing, handling missing values, and visualization

```python
# Revisar valores faltantes
print(df_raw.isnull().sum())

# Visualización de valores faltantes
sns.heatmap(df_raw.isnull(), cbar=False)
plt.title("Mapa de calor de valores faltantes")
plt.show()
```

[35]

```
seller_address                          0
warranty                            60896
sub_status                              0
condition                               0
seller_contact                      97781
deal_ids                                0
base_price                              0
shipping                                0
non_mercado_pago_payment_methods        0
seller_id                               0
variations                              0
location                                0
site_id                                 0
listing_type_id                         0
price                                   0
attributes                              0
buying_mode                             0
tags                                    0
listing_source                          0
parent_item_id                      23011
coverage_areas                          0
category_id                             0
descriptions                            0
last_updated                            0
international_delivery_mode             0
...
geolocation                             0
sold_quantity                           0
available_quantity                      0
dtype: int64
```



Some columns like seller_address and warranty, have many missing values, which could affect the analysis and the model to be built.

This heatmap shows that several columns have quite a few missing values, such as seller_address, seller_contact, and parent_item_id. So, we will remove them later in data preprocessing.

En este paso seleccionamos algunas columnas para eliminar, como identificadores que no nos brindan información relevante y columnas que tienen muchos datos nulos.

```
columns = ["warranty",
           "seller_contact",
           "deal_ids",
           "seller_id",
           "site_id",
           "listing_type_id",
           "parent_item_id",
           "category_id",
           "id",
           "seller_address",
            "sub_status",
            "variations",
            "location",
            "attributes",
            "tags",
            "listing_source",
            "coverage_areas",
            "international_delivery_mode",
            "pictures",
            "thumbnail",
            "automatic_relist",
            "secure_thumbnail",
            "permalink",
            "geolocation",
            "title",
           "official_store_id",
           "differential_pricing",
           "original_price",
           "currency_id",
           "video_id",
           "shipping",
           "non_mercado_pago_payment_methods",
           "descriptions",
           "catalog_product_id",
           "subtitle"]

df_cleaned = df_raw.drop(columns=(columns))
```

```
num_filas, num_columnas = df_cleaned.shape

print(f"el DataFrame tiene {num_filas} filas y {num_columnas} columnas.")
[37]
```

```
el DataFrame tiene 100000 filas y 13 columnas.
```

convertimos las columnas de fecha a datetime y luego al formato YYYYMMDD

```
# Convertir las columnas de fecha a datetime y luego al formato YYYYMMDD
fecha_columnas = ['last_updated', 'date_created', 'start_time', 'stop_time']

for col in fecha_columnas:
    df_cleaned[col] = pd.to_datetime(df_cleaned[col]).dt.strftime('%Y%m%d')

# Mostrar las columnas transformadas
print(df_cleaned[fecha_columnas])
[38]
```

```
      last_updated date_created start_time stop_time
0         20150905     20150905   20150905  20151104
1         20150926     20150926   20150926  20151125
2         20150909     20150909   20150909  20151108
3         20151005     20150928   20150928  20151204
4         20150828     20150824   20150824  20151023
...            ...          ...        ...       ...
99995     20150928     20150928   20150928  20151127
99996     20150911     20150911   20150911  20151110
99997     20150906     20150906   20150906  20151105
99998     20150818     20150818   20150818  20151017
99999     20150921     20150921   20150921  20151120

[100000 rows x 4 columns]
```

We check how many rows and columns we have left.

```
num_filas, num_columnas = df_cleaned.shape

print(f"el DataFrame tiene {num_filas} filas y {num_columnas} columnas.")
[37]
```

```
el DataFrame tiene 100000 filas y 13 columnas.
```

Observamos las variable numéricas.

```
numericas = df_cleaned.select_dtypes(include=np.number).columns.tolist()
[39]
```

Verificamos los tipos de datos de cada columna para aplicar One-Hot Encoding a las columnas categóricas.

```
print(df_cleaned.dtypes)
[40]
```

```
condition                 object
base_price               float64
price                    float64
buying_mode               object
last_updated              object
accepts_mercadopago         bool
date_created              object
stop_time                 object
status                    object
initial_quantity           int64
start_time                object
sold_quantity              int64
available_quantity         int64
dtype: object
```

Finlamente aplicamos One-Hot Encoding a todas las columnas categóricas.

```python
numericas = pd.get_dummies(df_cleaned, drop_first=True)

# Mostrar las primeras filas del DataFrame codificado
print(numericas.head())
```

[42]

```
   base_price   price  accepts_mercadopago  initial_quantity  sold_quantity  \
0        80.0    80.0                 True                 1              0
1      2650.0  2650.0                 True                 1              0
2        60.0    60.0                 True                 1              0
3       580.0   580.0                 True                 1              0
4        30.0    30.0                 True                 1              0

   available_quantity  condition_used  buying_mode_buy_it_now  \
0                   1           False                    True
1                   1            True                    True
2                   1            True                    True
3                   1           False                    True
4                   1            True                    True

   buying_mode_classified  last_updated_20141113  ...  start_time_20151006  \
0                   False                  False  ...                False
1                   False                  False  ...                False
2                   False                  False  ...                False
3                   False                  False  ...                False
4                   False                  False  ...                False

   start_time_20151007  start_time_20151008  start_time_20151009  \
0                False                False                False
1                False                False                False
2                False                False                False
...
3                False                False                False
4                False                False                False

[5 rows x 841 columns]
```

In this part we apply One-Hot Encoding to the categorical columns of the clean DataFrame. Now we have a DataFrame with numeric values that includes the encoded variables.

Convertir las clases categóricas 'new' y 'used' a valores numéricos (0 - 1).

```python
df_cleaned['condition'] = df_cleaned['condition'].map({'new': 1, 'used': 0})

# Verifica las primeras filas para confirmar
print(df_cleaned[['condition']].head())
```

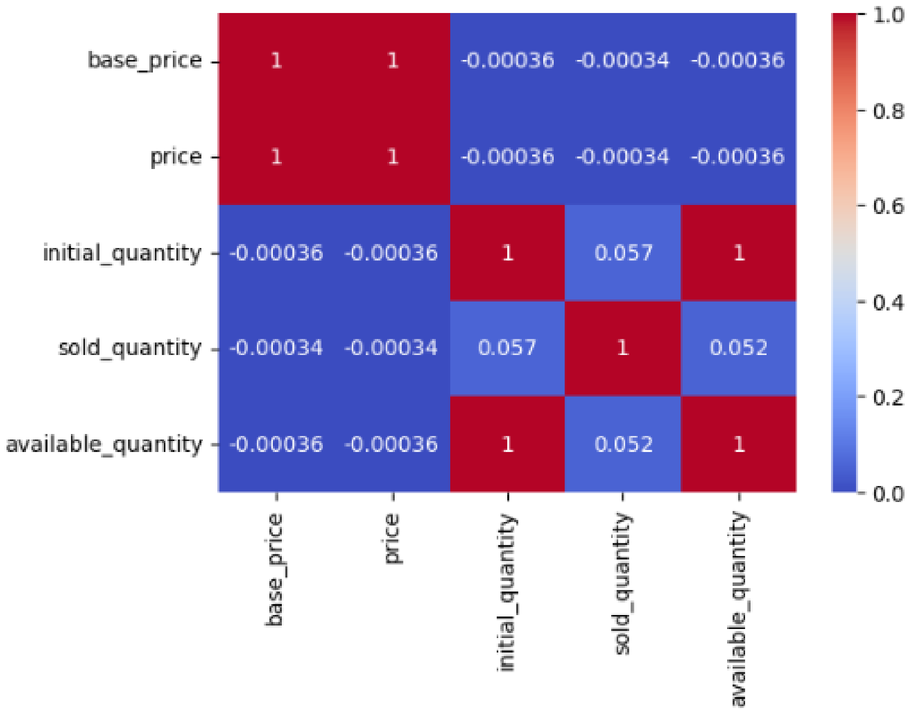
[43]

```
   condition
0          1
1          0
2          0
3          1
4          0
```

Ahora la columna tiene valores binarios que vamos a utilizar para el entrenamiento de modelos.

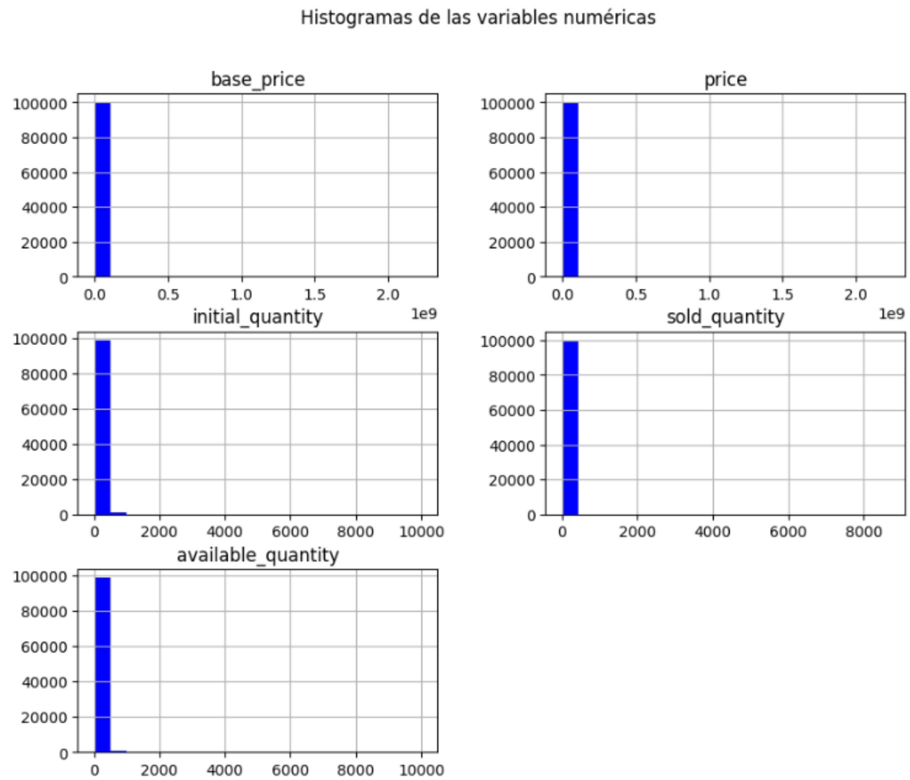
EDA VISUALIZATIONS

Calculate the correlation matrix of the numerical variables.



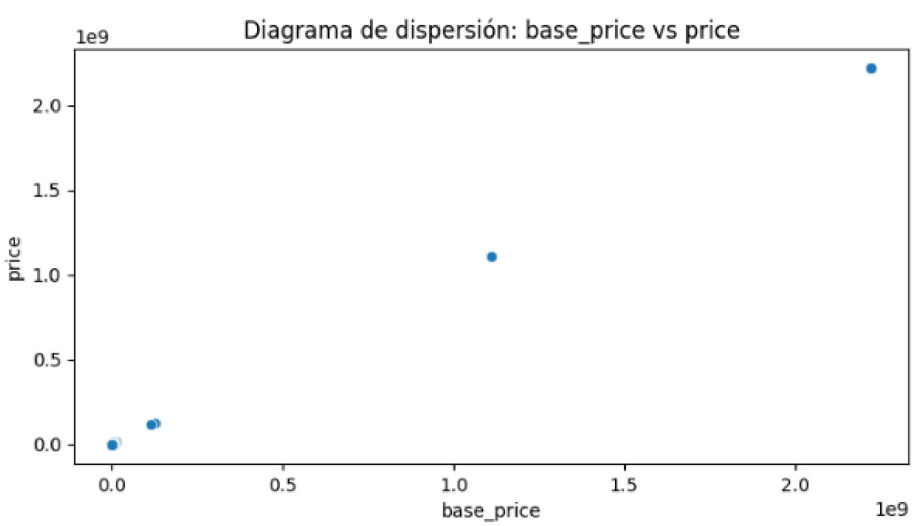
The correlation matrix shows that there are no strong relationships between the numerical variables in the dataset. The correlations between variables such as initial_quantity, sold_quantity, and available_quantity is very weak, with values close to 0. This suggests that these variables are not strongly related to each other, which is favorable for the model, as there is no significant risk of multicollinearity.

histograms for all numeric columns.



The histograms show that the numerical variables (base_price, price, initial_quantity, sold_quantity, and available_quantity) are concentrated near 0, indicating that most values are low. Long tails towards higher values are also observed, suggesting the presence of outliers. This implies that most products have low prices and small quantities, while only a few have exceptionally high values.

We create scatter plots between the first 4 pairs of numerical variables, where we can see some relationships between prices.



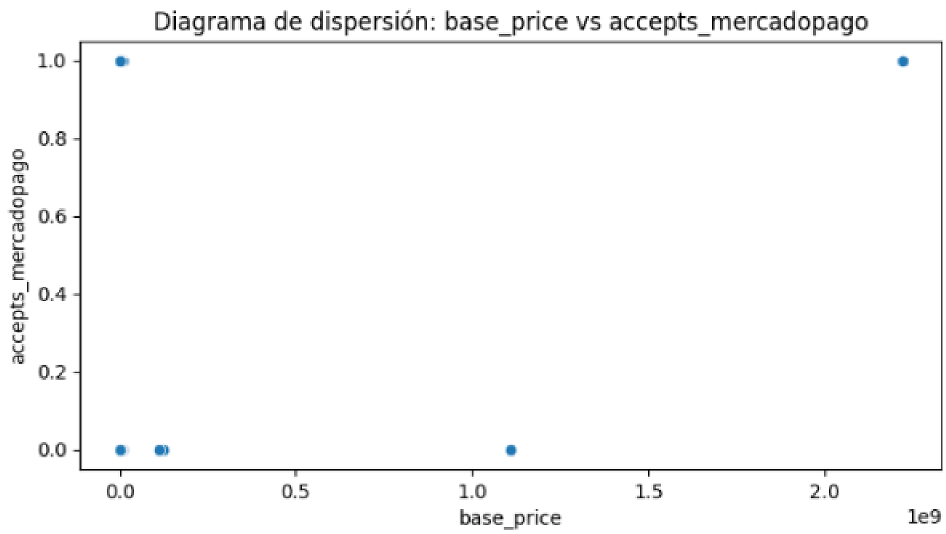


Diagrama de dispersión: base_price vs accepts_mercadopago
accepts_mercadopago
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0
1e9
base_price

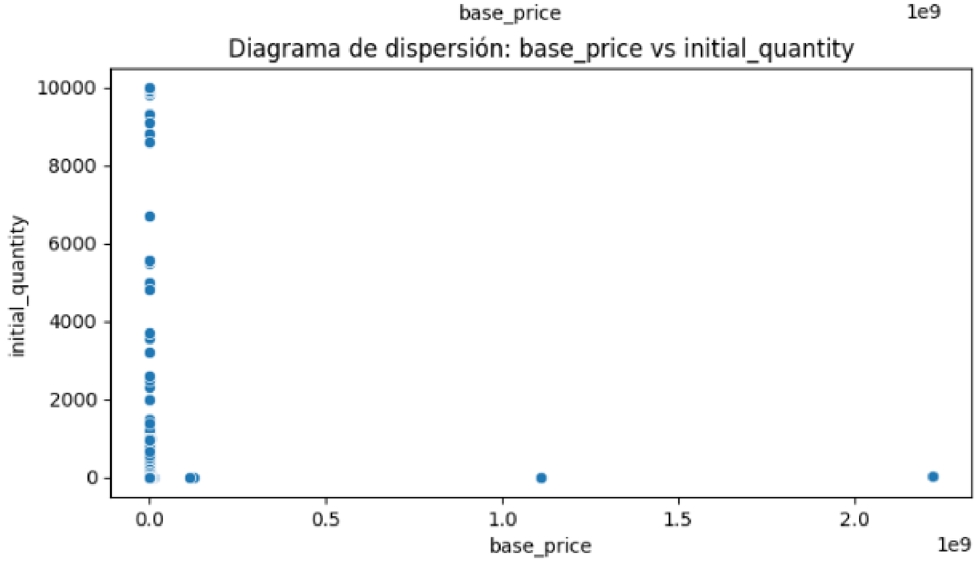


Diagrama de dispersión: base_price vs initial_quantity
initial_quantity
10000
8000
6000
4000
2000
0
0.0
0.5
1.0
1.5
2.0
1e9
base_price

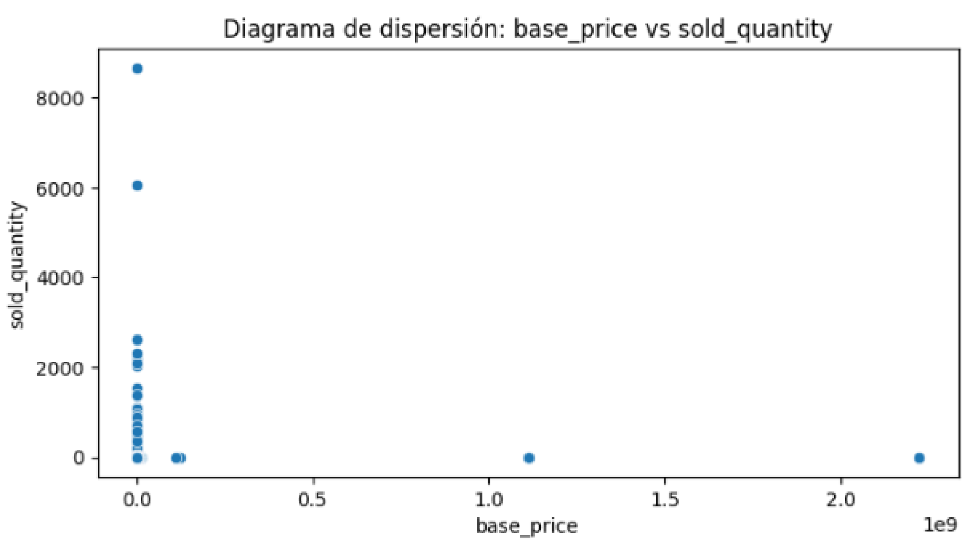
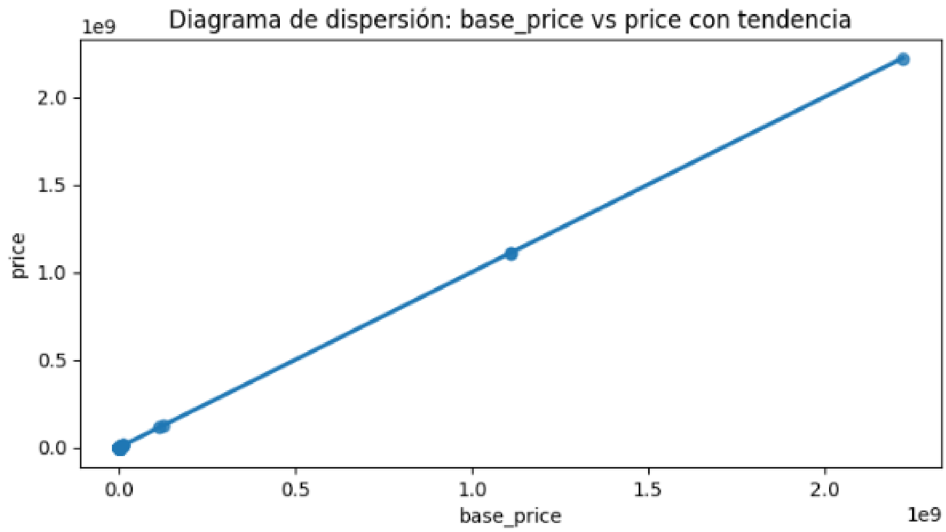
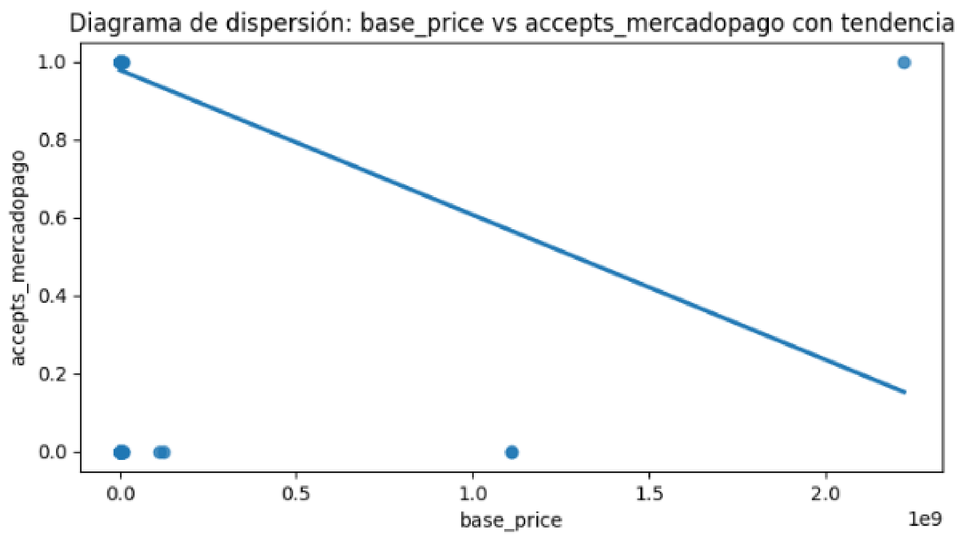


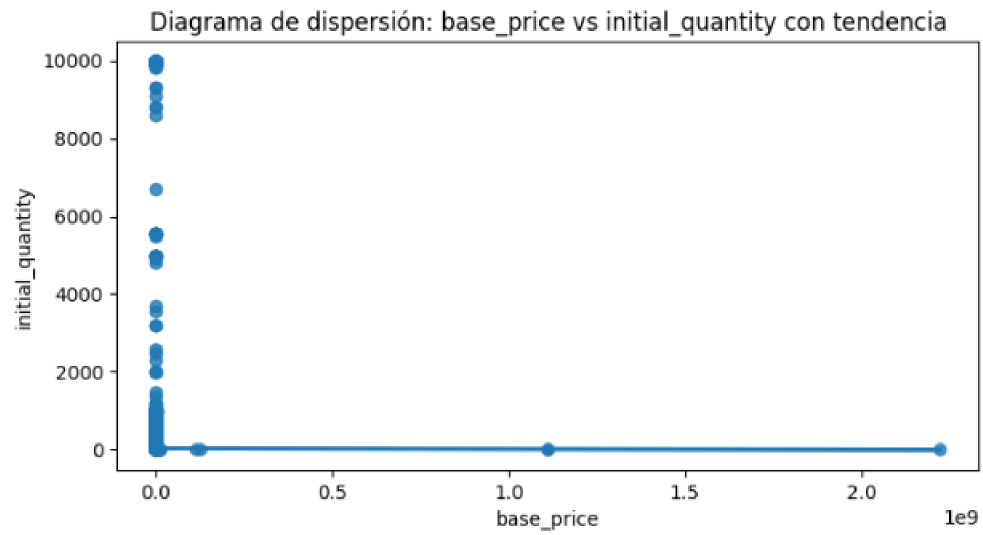
Diagrama de dispersión: base_price vs sold_quantity
sold_quantity
8000
6000
4000
2000
0
0.0
0.5
1.0
1.5
2.0
1e9
base_price

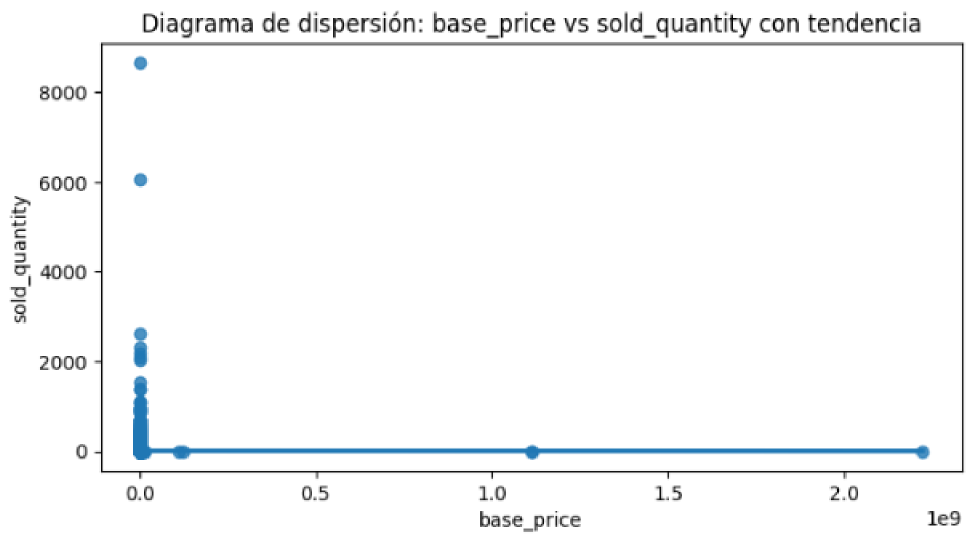
Scatter plots show that most products have low base prices, with few high-priced outliers. There is a clear relationship between base price and final price, although in some cases it is not proportional, suggesting discounts or differentiated pricing strategies. The variables for initial quantity and quantity sold show that most products have low quantities, with no strong correlation with base price, although some more expensive products also manage to sell large quantities.

We created other histograms to visualize trends.









The graphs show that there is a strong linear relationship between the base price and the final price, indicating that the final price increases proportionally to the base price. However, there is no clear relationship between the base price and variables such as accepts_mercadopago, initial_quantity, and sold_quantity, as they do not seem to be significantly correlated. The initial and sold quantities do not directly depend on the base price, and the acceptance of payments by MercadoPago does not show a clear trend in relation to the price either.

Finally, we save the clean dataset with the appropriate transformations.

```
df_cleaned.to_csv("../data/MLA_100k_clean.csv", index=False)
```

MODEL TRAINING

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import json
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, roc_auc_score
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, roc_auc_score
import joblib
```

[20] ✓ 0.0s

Leemos el dataset limpio

```python
df = pd.read_csv('../data/MLA_100k_clean.csv')
```

[4] ✓ 0.0s

We review the columns in the dataset and then generate the models.

```python
df.head()
```

✓ 0.0s

	condition	base_price	price	buying_mode	last_updated	accepts_mercadopago	date_created	stop_time	status	initial_quantity	start_time	sold_quantity	available_quantity
0	1	80.0	80.0	buy_it_now	20150905	True	20150905	20151104	active	1	20150905	0	1
1	0	2650.0	2650.0	buy_it_now	20150926	True	20150926	20151125	active	1	20150926	0	1
2	0	60.0	60.0	buy_it_now	20150909	True	20150909	20151108	active	1	20150909	0	1
3	1	580.0	580.0	buy_it_now	20151005	True	20150928	20151204	active	1	20150928	0	1
4	0	30.0	30.0	buy_it_now	20150828	True	20150824	20151023	active	1	20150824	0	1

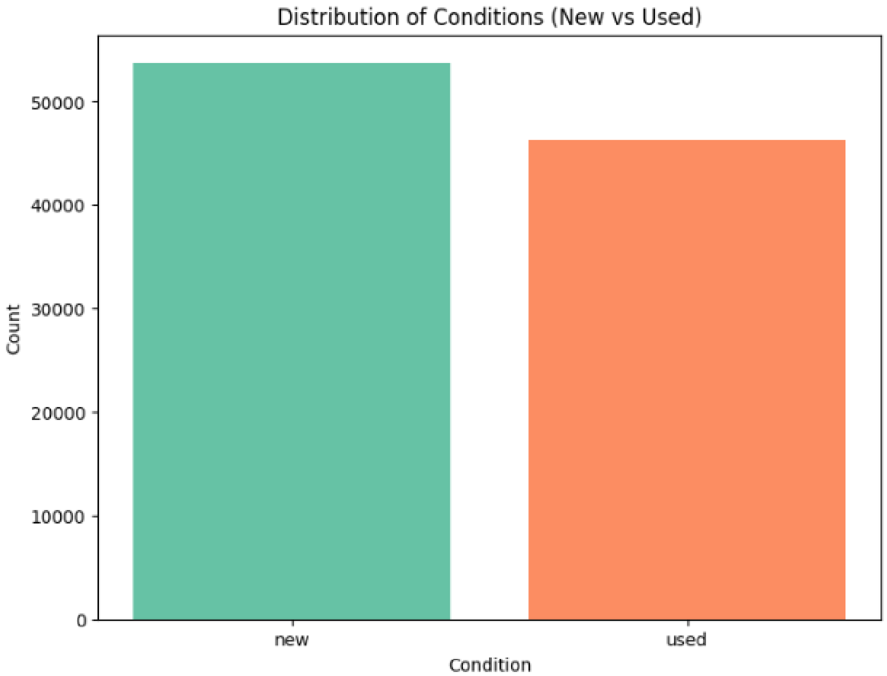
Converting categorical variables to dummy variables (One-Hot Encoding).

```python
df_encoded = pd.get_dummies(df, drop_first=True)

# Definir X (todas las columnas excepto la columna objetivo) e y (columna objetivo)
X = df_encoded.drop('condition', axis=1)
y = df_encoded['condition']

# Dividir el dataset en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Bar chart to see the distribution of conditions.



This bar chart indicates that there are more new products in the dataset than used products. Although the difference is not huge, there is a trend towards a higher supply of new products compared to used ones.

Generate the prediction models using different algorithms: Logistic Regression, RandomForestClassifier, XGBClassifier, Gradient Boosting and K-Nearest Neighbors (KNN).

WE TRAIN THE MODELS.

logistic regression.

```
# Entrenar el modelo de Regresión Logística
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)

# Hacer predicciones
y_pred_log = log_reg.predict(X_test)

# Calcular la precisión
accuracy_log = accuracy_score(y_test, y_pred_log)
print(f'Accuracy Logistic Regression: {accuracy_log}')
```

✓ 1.8s

```
Accuracy Logistic Regression: 0.7095
```

RandomForestClassifier.

```
# Crear el modelo Random Forest
rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)

# Entrenar el modelo
rf_clf.fit(X_train, y_train)

# Hacer predicciones
y_pred_rf = rf_clf.predict(X_test)

# Evaluar el modelo
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f'Accuracy Random Forest: {accuracy_rf}')
```

✓ 7.5s

```
Accuracy Random Forest: 0.76685
```

XGBClassifier.

```
# Crear el modelo XGBoost
xgb_clf = XGBClassifier(eval_metric='mlogloss')

# Entrenar el modelo
xgb_clf.fit(X_train, y_train)

# Hacer predicciones
y_pred_xgb = xgb_clf.predict(X_test)

# Evaluar el modelo
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
print(f'Accuracy XGBoost: {accuracy_xgb}')
```

✓ 0.3s

```
Accuracy XGBoost: 0.78515
```

Gradient Boosting.

```
# Entrenar Gradient Boosting
gb_clf = GradientBoostingClassifier(random_state=42)
gb_clf.fit(X_train, y_train)
y_pred_gb = gb_clf.predict(X_test)

# Evaluar Gradient Boosting
print("Gradient Boosting - Accuracy:", accuracy_score(y_test, y_pred_gb))
```

✓ 16.2s

```
Gradient Boosting - Accuracy: 0.76995
```

K-Nearest Neighbors (KNN)

```
# Entrenar KNN
knn_clf = KNeighborsClassifier(n_neighbors=5)
knn_clf.fit(X_train, y_train)
y_pred_knn = knn_clf.predict(X_test)

# Evaluar KNN
print("KNN - Accuracy:", accuracy_score(y_test, y_pred_knn))
[32] ✓ 4.2s
... KNN - Accuracy: 0.72995
```

Compare the results.

```
Accuracy Logistic Regression: 0.7095
Accuracy Random Forest: 0.76685
Accuracy XGBoost: 0.78515
Accuracy Gradient Boosting: 0.76995
Accuracy KNN: 0.72995
```

After training the five models: Logistic Regression, RandomForestClassifier, XGBClassifier, Gradient Boosting and K-Nearest Neighbors (KNN). To select the best one, we use the accuracy metric.

In this case, the XGBoost model has had the best performance in terms of accuracy **(0.78515).**

This indicates that it is the most effective in correctly classifying the instances in the dataset.

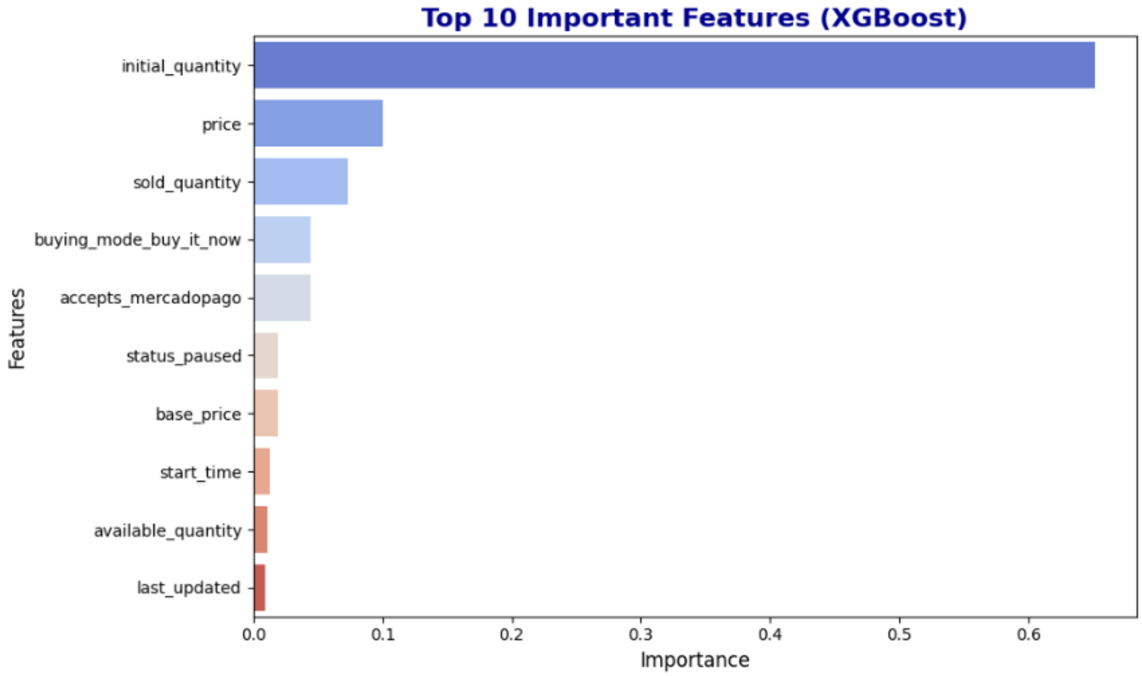
Save the XGBoost model as a .pkl file

```
joblib.dump(xgb_clf, '../Model/modelo_xgboost.pkl')

print("Modelo XGBoost guardado como 'modelo_xgboost.pkl'")
✓ 0.0s
Modelo XGBoost guardado como 'modelo_xgboost.pkl'
```

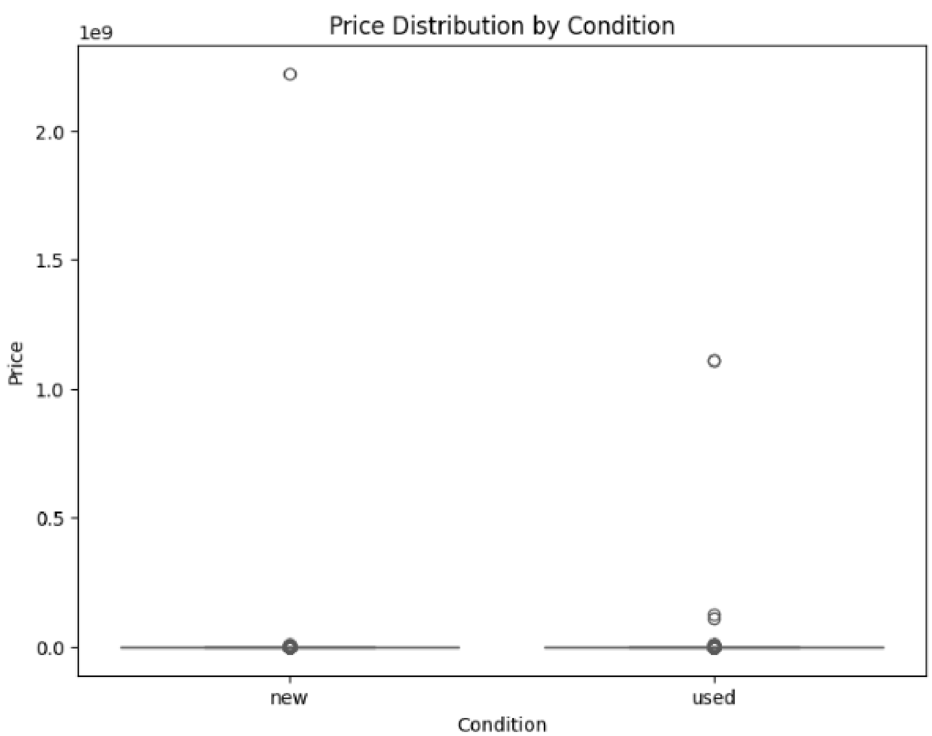
We made some graphics.

Graph of variables influencing the behavior of the model.



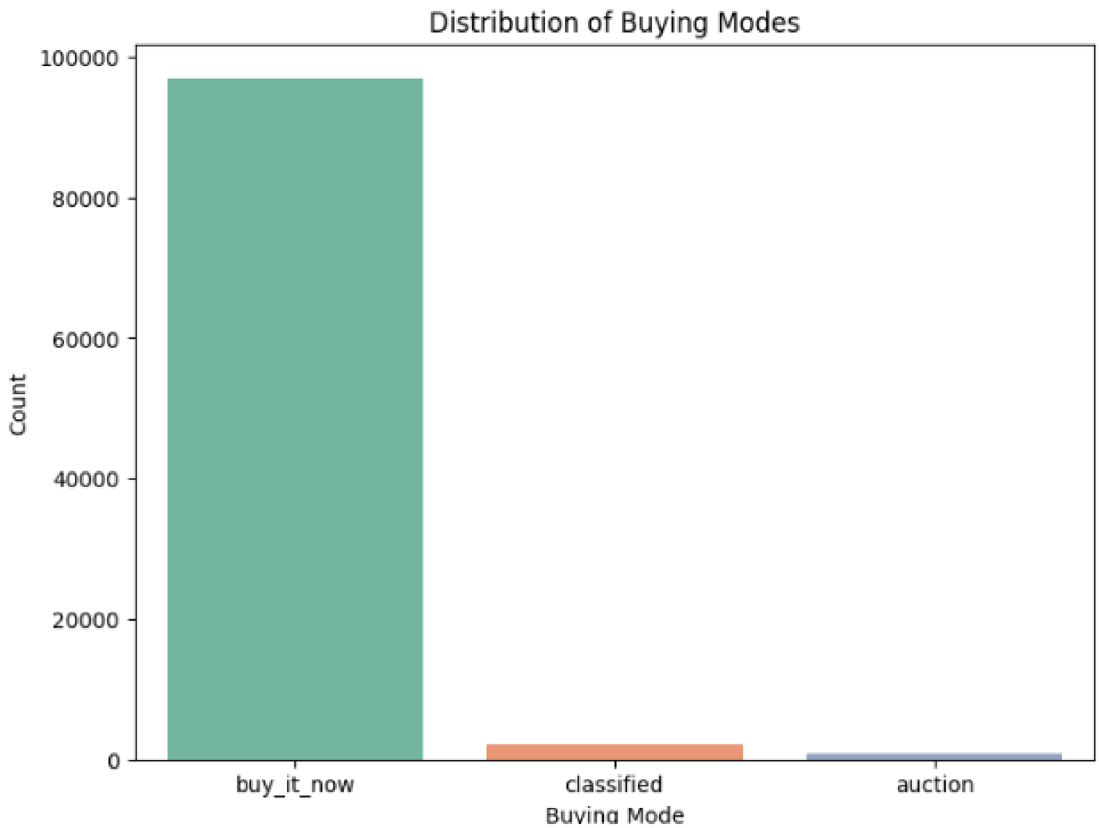
The most influential feature in the model is 'initial_quantity', with an importance close to 0.6. This means that it plays a crucial role in the model's decisions. In general, the higher the importance value of a feature, the more relevant it is to the model's performance. This implies that any variation in 'initial_quantity' can significantly affect the predictions generated by the model.

Price distribution chart according to product condition (new or used).



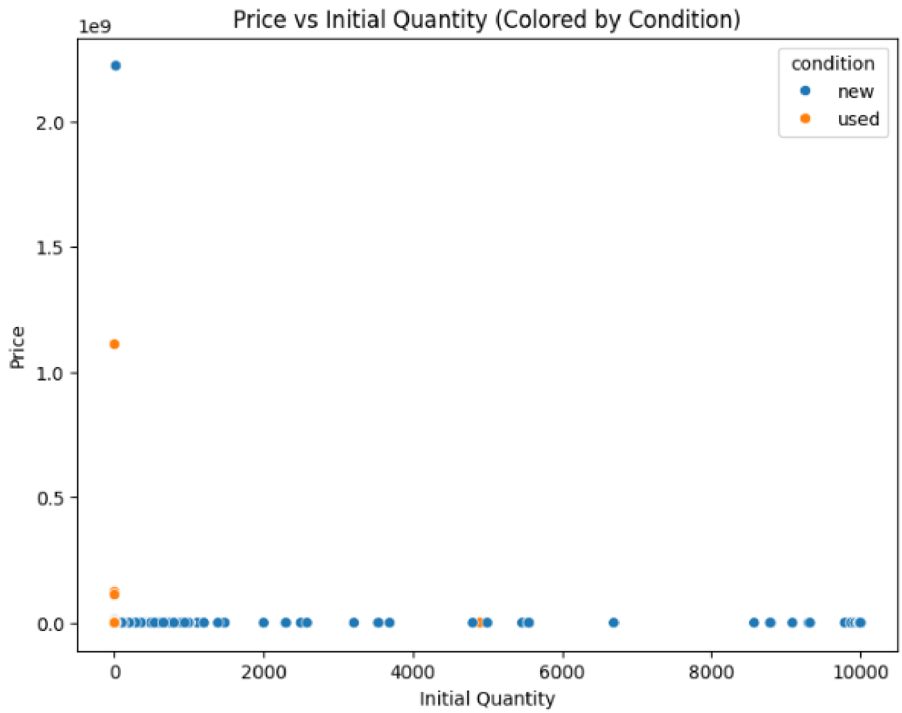
This chart shows that most products, both new and used, have very low prices, but there are some products with extremely high prices (outliers). There are no major differences in the central distribution of prices between new and used products.

Bar chart for buying mode.



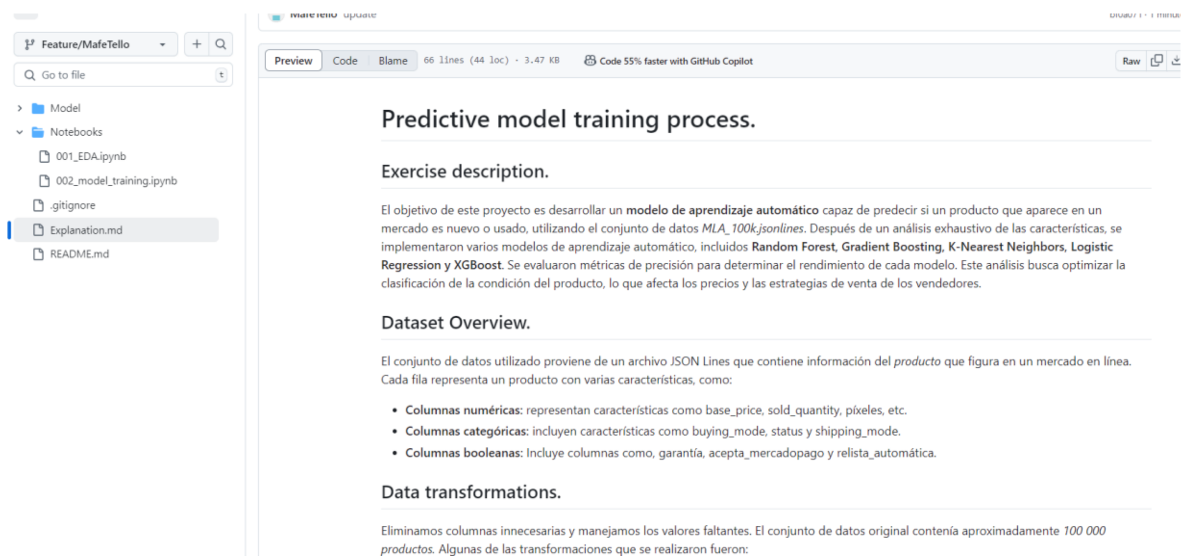
This chart shows that the vast majority of products are sold under the "buy it now" option, while "classified" and "auction" options are not as common.

Relationship between two numerical variables: Price and the initial quantity of products.



This chart shows the relationship between price and initial quantity of products, differentiating between new (in blue) and used (in orange) products. Most products have a low price and relatively small initial quantities. However, there are some outliers with extremely high prices. Most of these points correspond to new products.

Then I created the explanation.md file to add the explanation of everything that had been done in the project.



Finally, we uploaded all the folders to the GitHub repository!