Documentación Trabajo final Economía y Ciencia de Datos

EAE253 - 1

Grupo 15:
Tomás Lazcano
Tomás Maiza
Diego Manriquez
Renato Pirisi
Martín Yarur

Aplicación de la Solución

La página web es una herramienta interactiva construida con Streamlit que permite a los usuarios predecir el precio de laptops basándose en sus características técnicas, visualizar el rendimiento de los modelos utilizados (Random Forest, KNN, y Lasso) y explorar datos relacionados con las laptops para obtener insights relevantes (pestaña análisis exploratorio).

La idea es mantener una estructura de la página web con un inicio para dar la bienvenida al proyecto. Luego otra con la predicción donde el usuario pueda ingresar características de laptops (como tamaño de pantalla, RAM, GPU, etc.) y con base de estos inputs, obtener la predicción del precio con los modelos previamente "entrenados". La pestaña siguiente incluye el rendimiento de los modelos con gráficos y métricas mostrando la eficacia de los modelos. La última pestaña contendrá el análisis exploratorio donde se describen los datos originales para construir los modelos y expone las relaciones más importantes entre características técnicas y precios.

La solución descrita está implementada en Python y se utilizan varias bibliotecas clave, como streamlit, pandas y scikit-learn. Donde el código principal (app.py) sigue una estructura que utiliza módulos independientes uno al otro pero que termina aportando cada uno en el funcionamiento para facilitar comprensión y replicación. Los pasos principales son:

La carga de base de datos desde SQLite y se preprocesa utilizando funciones de módulo externo (carga_datos) y codificaciones de variables categóricas a través de one-hot encoding (cómo con GPU_brand).

```
# procesar_datos()  ->  Agrega y modifica columnas a partir del csv original
df = load_data()
X_train, X_test, y_train, y_test, scaler, feature_names = preprocess_data(df)
```

Luego sigue el entrenamiento de modelos donde se entrenan usando los datos preprocesados donde random forest captura relaciones no lineales, KNN aprovecha similitudes entre laptops con configuración similar y lasso que identifica las más relevantes.

```
# Entrenar los modelos
knn, rf, lasso_grid_search = train_models(X_train, y_train)
```

Finalmente los usuarios ingresan características de una laptop mediante la interfaz de streamlit y los modelos predicen el precio en función de los inputs.

```
# Predicción
input_data = pd.DataFrame({
    'Inches': [inches],'ResolutionNumber': [resolution_number],'RAM_GB': [ram_gb],'HDD_space': [hdd_space],
    'SSD_space': [ssd_space],'Weight_KG': [weight_kg],'TypeName_Notebook': [1 if tipo_laptop == 'Notebook' else 0],
    'TypeName_Netbook': [1 if tipo_laptop == 'Netbook' else 0],'TypeName_Workstation': [1 if tipo_laptop == 'Workstation' else
    'TypeName_Gaming': [1 if tipo_laptop == 'Gaming' else 0],'TypeName_2 in 1 Convertible': [1 if tipo_laptop == '2 in 1 Conver
    'TypeName_Ultrabook': [1 if tipo_laptop == 'Ultrabook' else 0], 'GPU_brand_Nvidia': [1 if gpu_brand == 'NVIDIA' else 0],
    'GPU_brand_AMD': [1 if gpu_brand == 'AMD' else 0],'GPU_brand_Intel': [1 if gpu_brand == 'Intel' else 0]})
price_knn, price_rf, price_lasso = predict_price(
    (knn, rf, lasso_grid_search), input_data, scaler, ['Inches', 'ResolutionNumber', 'RAM_GB', 'HDD_space', 'SSD_space', 'Weight_KG'],
    ['TypeName_Notebook', 'TypeName_Netbook', 'TypeName_Workstation', 'TypeName_Gaming', 'TypeName_2 in 1 Convertible', 'TypeName_Ultrab
`
```

Carga de datos:

El archivo carga_datos.py posee 9 funciones principales asociadas con la carga de datos desde el CSV descargado de kaggle a una base SQlite, con el objetivo de crear columnas nuevas a partir de información existente que facilita el análisis posterior.

El proceso comienza con la carga de datos desde un archivo CSV a un data frame de Pandas mediante la función **cargar_datos(filepath)**. **Procesar_screen_resolution(df)** toma la columna ScreenResolution del data frame y genera dos columnas nuevas, una para identificar si la pantalla es táctil ("Touchscreen"), y otra para obtener el área de la resolución en píxeles. **Procesar_cpu(df)** clasifica las CPUs en tres marcas principales ("Intel",

“AMD” y “Samsung”), dependiendo del nombre del procesador. **Procesar_ram(df)** convierte la información de memoria RAM en valores numéricos enteros , expresados en gigabytes. **Procesar_memoria(df)** desglosa “Memory” en diferentes tipos de almacenamiento (“HDD_space”, “SSD_space”, “Flash_space” y “Hybrid_space”), para obtener los almacenamientos por separado. **Procesar_gpu(df)** clasifica las GPUs en las marcas “Nvidia”, “AMD”, “Intel”, “ARM” o “Unknown”. **Procesar_peso(df)** convierte los valores de peso a kilogramos en formato float. **Procesar_op_sys(df)** estandariza los nombres de los sistemas operativos, agrupándolos en categorías como “Windows”, “Mac” y “Otro”. **Guardar_datos(df, database_path)** almacena la información en una base de datos SQLite. En esta base se guardan tanto los datos completos como una tabla con la versión acotada que incluye columnas seleccionadas relevantes para el análisis. Finalmente, **procesar_datos()** utiliza todas las funciones anteriormente definidas para obtener la base de SQlite para el análisis. Esta función puede ser llamada ejecutando el código carga_datos.py, o importándola en otros módulos.

Análisis exploratorio (exploratorio.py):

En las instancias iniciales del proyecto, es necesario realizar una serie de procesos en donde se describan las variables de la base y sus movimientos conjuntos, con tal de conocer mejor los datos y usarlos más provechosamente.

En primer lugar, se procedió a echar un vistazo a la forma de la base de datos y a obtener la estadística descriptiva de la variable de interés (*price_euros*), para lo cual utilizamos el comando *head* y *describe.* Además, se generó un histograma donde se muestra la distribución del precio de los computadores presentes en la base (Anexo 1). Esto último, utilizando la librería Seaborn.

En segundo lugar, se construyó una matriz de correlación con el fin de observar cómo varían conjuntamente las variables numéricas de la base de datos Anexo 2).

En tercer lugar, se generó un gráfico de caja para analizar la distribución de precios (Price_euros) según las categorías de “Hybrid_space” (Anexo 3), haciendo nuevamente uso de la librería Seaborn. Esto permite visualizar la mediana, el rango intercuartil y los *outliers* de cada categoría. Con esta misma estructura de código, se construyeron también gráficos (de caja) similares para otras variables como Company y Ram_GB (Anexos 4 & 5, respectivamente), entre otras.

Por último, se construyó un gráfico de dispersión para visualizar cómo distribuye el precio de los computadores para diferentes resoluciones de pantalla (variable *ScreenResolution*) y tener una idea de cuántos modelos hay para cada valor que tome la resolución (Anexo 6). Esto puede ayudar a identificar tendencias, patrones o agrupamientos (por ejemplo, dispositivos con precios altos y resoluciones específicas). Análogo a lo hecho con los gráficos de caja, este código se reutilizó para crear un gráfico (de dispersión) para la variable *Memory (Anexo 7).*

Para estas dos variables en específico no fue posible realizar gráficos de caja, pues cuentan con un numero de categorías inconvenientemente alto para este fin.

Bibliografía:
código de histograma, matriz de correlación (heatmap) y gráficos de cajas:
- https://www.kaggle.com/code/abonaplata/analisis-exploratorio-de-datos-con-python?scriptVersionId=17204829&cellId=21

código de gráfico de dispersión:
- https://www.kaggle.com/code/mohamedhaithamyamani/laptop-price-prediction-with-eda?scriptVersionId=197820240&cellId=14

Entrenamiento de modelos (predicciones.py):

load_data(): carga los datos desde la base de datos y utiliza una consulta en SQL para recuperar solo las columnas relevantes para entrenar los modelos y se saca una observación ya que era la única en tener CPU_brand=samsung, lo que podía entorpecer los modelos. Luego las variables categoricas son transformadas a dummies

preprocess_data(df): Se le entrega el dataframe resultante de load_data() y divide las observaciones en una muestra de entrenamiento y una de validación, separando por las variables predictoras y la predicha. Luego estandariza los datos numéricos predictores para la muestra de validación y entrenamiento. Finalmente retorna cada una de las submuestras y el nombre de las variables predictoras.

train_models(X_train,y_train): Crea los modelos knn y random forest con hyperparámetros predefinidos y además crea el modelo lasso sin definir el hyperparámetro. Luego entrena los modelos y se utiliza validación cruzada para encontrar el apha que minimice el RMSE.

predict_price(models, input_data, scaler, numeric_columns, dummy_columns): Recibe de input las nuevas variables explicativas, las estandariza y retorna las predicciones de los modelos entrenados anteriormente de los precios para un computador con dichas especificaciones.

plot_model_performance(df, y_test, y_pred_knn, y_pred_rf, y_pred_lasso, rf_model, lasso_cv): Genera gráficos interactivos con el uso de plotly, para analizar y comparar el desempeño de los 3 modelos. Esta función es usada en app.py para generar gráficos. Los resultados se presentan en 7 visualizaciones que muestran diferentes aspectos de los modelos y su rendimiento.

Primero, un scatterplot con la comparación entre valores reales y predicciones de los 3 modelos. También incluye una línea que representa la perfección en las predicciones (donde predicción = valor real), lo que permite evaluar visualmente la precisión.

Segundo, la importancia de las características para Random Forest presentadas en un gráfico de barras horizontales, ordenados de mayor a menor importancia, lo que facilita identificar qué variables tienen mayor peso en las predicciones.

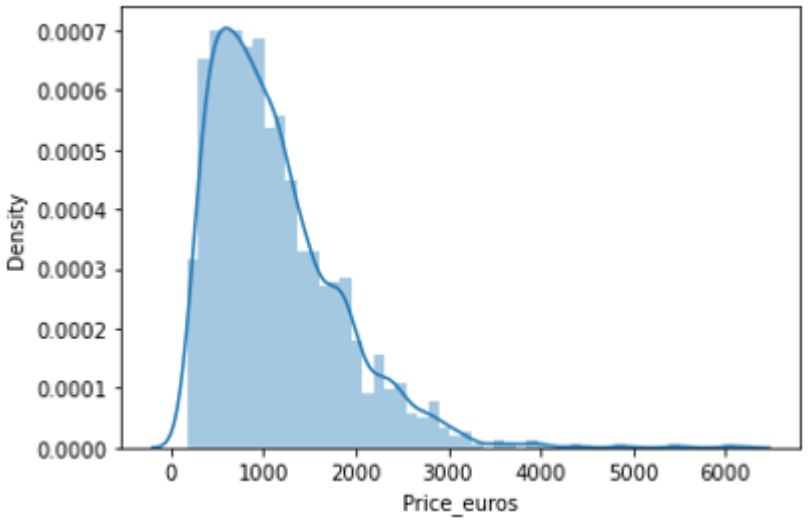
Tercero, la regularización en Lasso, que muestra cómo varía el error cuadrático medio (MSE) con diferentes valores de alpha en el modelo, utilizando validación cruzada. Este gráfico ayuda a interpretar el efecto de la regularización y resalta el alpha óptimo seleccionado por el modelo.

Cuarto a sexto, el desempeño de R^2 para cada modelo mediante gráficos circulares. Esto proporciona una manera visual de entender cuánto de la variación de los datos que se explica por cada modelo y cuánto queda sin explicar.

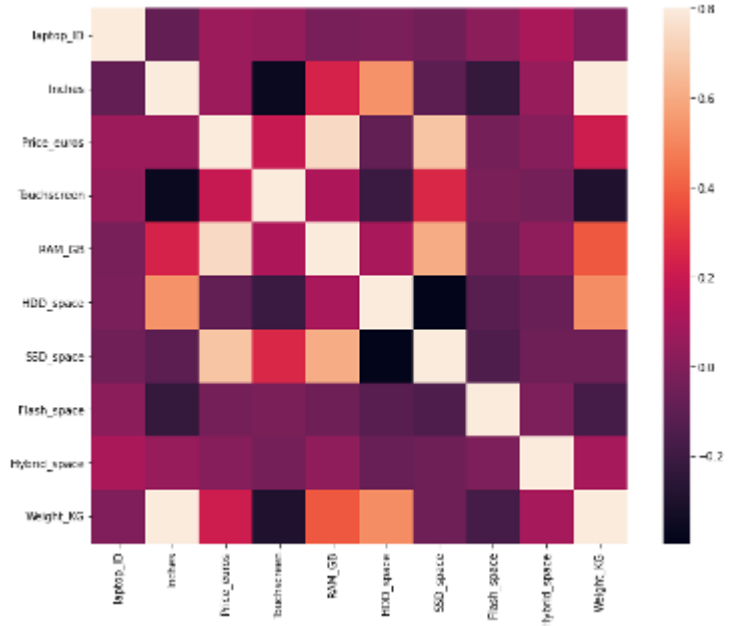
Séptimo, la raíz cuadrada de los errores cuadráticos medios (RMSE) de los modelos en un gráfico de barras. Este análisis permite medir la magnitud promedio de los errores de predicción, siendo útil para identificar cuál modelo es más preciso.

Anexos

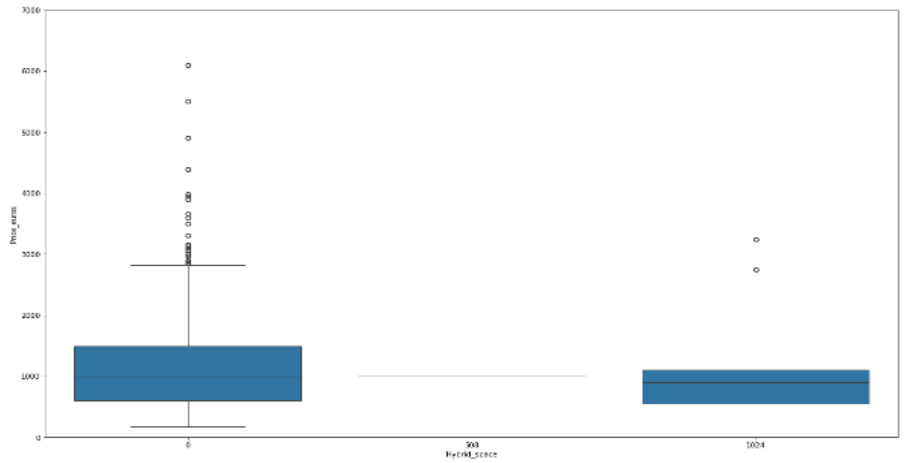
- **Anexo 1: Histograma precio computadoras**



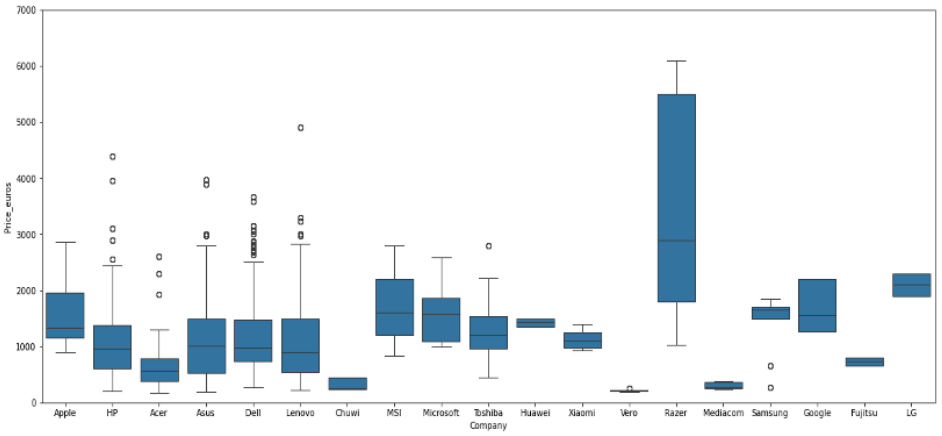
- **Anexo 2: Matriz de correlaciones**



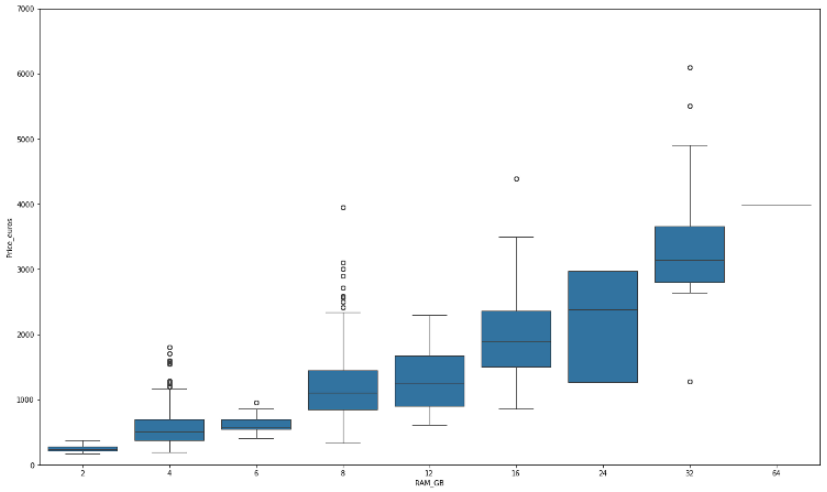
- **Anexo 3: gráfico de caja para Price_euros y Hybrid_space:**



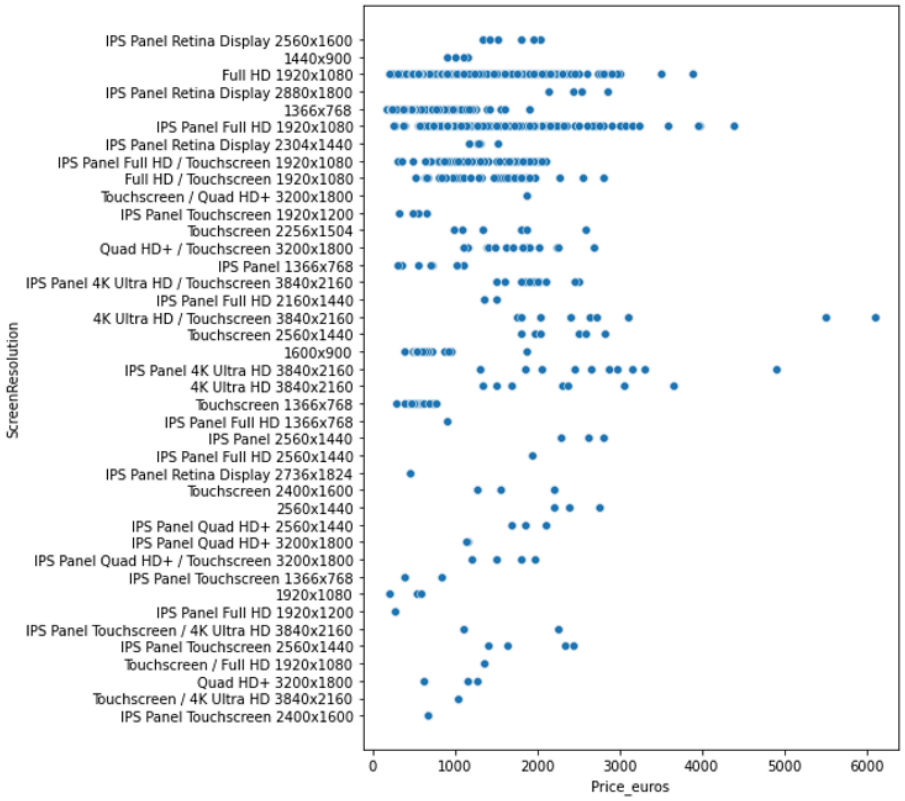
- **Anexo 4: gráfico de caja para Price_euros y Company:**



- **Anexo 5: gráfico de caja para Price_euros y RAM_GB:**



- **Anexo 6: gráfico Price_euros y ScreenResolution:**



- **Anexo 7: gráfico Price_euros y Memory:**