



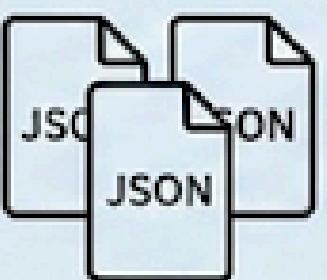
HUNTING SCAMS ON WALLAPOPO

A DATA PIPELINE AND FRAUD DETECTION CHALLENGE



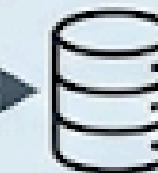
INICIALIZACIÓN DEL PROYECTO

FASE 1:
PREPARACIÓN DE INFRAESTRUCTURA
(Elasticsearch)



ilm_policy.json
index_template.json
index_alias.json

Configurar ES
(PUT requests)



Elasticsearch Listo
(Índice: lab10310.wallapop)

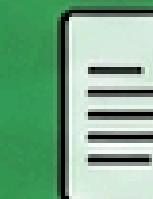
FASE 2:
RECOLECCIÓN DE DATOS HISTÓRICOS
(Entrenamiento)



FASE 3:
GENERACIÓN DE BASE DE CONOCIMIENTO
(Estadísticas)



Ejecutar analyst_poller.py
(Descarga 30 días)



Datos Crudos
(wallapop_raw_full_*.json)



Ejecutar regex_analyzer.py
(Procesamiento y Cálculo)



Archivo de Estadísticas
(market_stats.json)



PREPARACIÓN COMPLETADA

Datos listos para el Poller diario y la Ingesta

PIPELINE DIARIO



RESUMEN

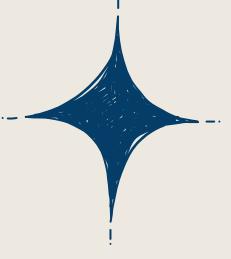
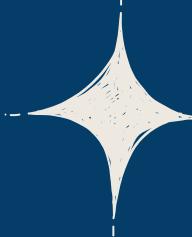
elastalert/
 > rules/
 > high_risk.yaml
 > config.yaml

ingestion/
 > bulk_ingest.py
 > ilm_policy.json
 > index_alias.json
 > index_template.json

kibana/
 > dashboard.ndjson

poller/
 > analist_poller.py
 > poller.py
 > regex_analyzer.py

market_stats.json
run_pipeline.sh
README.md
requirements.txt

- 
- 
- 
- 1** Orquestación: Un script Bash (`run_pipeline.sh`) gestiona el ciclo de vida.
 - 2** Recolección y procesamiento (`poller+regex_analyzer`): Descarga y analiza datos con heurística de riesgo y extrae especificaciones (CPU, RAM) del texto sucio mediante expresiones regulares (regex).
 - 3** Ingesta (`bulk_ingest`): Carga eficiente en Elasticsearch de los datos recolectados.
 - 4** Visualización & Alertas: Kibana y ElastAlert2 permiten visualizar y notificar posibles fraudes.

MARKET STATS



Technology & electronics > Computing: computers & tablets > Laptops (ID: 10310)

1 ANALIST_POLLER.PY

(Recolección de Datos) Se conecta a Wallapop y descarga miles de anuncios "en crudo" (raw data) para tener datos históricos para analizar. Guardados en wallapop_raw_full_YYYYMMDD_HHMM.json.

2 REGEX_ANALYZER.PY

(Generación de Estadísticas) Lee el archivo wallapop_raw_full_*.json, procesa todos los anuncios, calcula las medias de precio y desviaciones estándar para cada categoría, y genera el archivo market_stats.json.

3 POLLER.PY

(Detección en Tiempo Real) Usando el archivo market_stats.json, junto con más reglas, decide si un producto es sospechoso o no.

ANALIST_POLLER.PY

CONFIGURACIÓN DE RECOLECCIÓN

```
CATEGORY_ID = "24200"
TARGET_SUB_ID = "10310" # Portátiles
MAX_ITEMS_LIMIT = 50000
DAYS_TO_RETRIEVE = 30
SAVE_INTERVAL_MINUTES = 10
```

HEADERS: MAKE_REQUEST

```
Host: api.wallapop.com
Accept: application/json, text/plain, /*
User-Agent: Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36...
X-DeviceOS : O
X-Requested-With: XMLHttpRequest
```

PARAMS: RUN_COLLECTOR

```
"category_id": CATEGORY_ID,
"subcategory_ids": TARGET_SUB_ID,
"source": "side_bar_filters",
"country_code": "ES",
"order_by": "newest",
"latitude": "40.4168",
"longitude": "-3.7038",
```

```
# analist_poller.py (Resumido)

def make_request(url):
    """Gestión de red robusta: Reintentos + Backoff exponencial"""
    for attempt in range(3):
        try:
            time.sleep(1.5 ** attempt) # Espera incremental
            resp = requests.get(url, headers=FAKE_HEADERS)
            if resp.status_code == 429: # Rate Limit
                time.sleep(60)
                continue
            return resp
        except:
            continue
    return None

def get_item_details_full(item_id):
    """Deep Fetch: Obtiene datos críticos que no están en la búsqueda normal"""
    url = f"https://api.wallapop.com/api/v3/items/{item_id}"
    resp = make_request(url) # Usamos nuestra función segura
    return resp.json() if resp and resp.status_code == 200 else {}

def run_collector():
    """Bucle principal de recolección masiva"""
    all_items = []

    while len(all_items) < 50000:
        # 1. Obtener página de resultados
        items_batch = api.search(category="Laptops", time_filter="30_days")

        for item in items_batch:
            # 2. DEEP FETCH: Llamada extra por cada artículo
            details = get_item_details_full(item["id"])

            # 3. Enriquecer con datos ocultos (condición real, métricas...)
            item.update(details)
            all_items.append(item)

        # 4. Guardado atómico periódico
        save_checkpoint(all_items, "wallapop_dataset.json")
```

REGEX_ANALYZER.PY

FUNCIONES DE UTILIDAD (PRECIOS Y TEXTO)

```
>clean_price()  
>try_extract_hidden_price()  
>is_match()  
>smart_truncate_spam()  
>sanitize.hardware_ambiguities()
```

FUNCIONES DE ANÁLISIS DE HARDWARE

```
>detect_condition_from_data()  
>apply_category_constraints()  
>is_valid_ram()  
>clean_cpu_string()  
>clean_gpu_string()  
>extract_ram()  
>extract_specs_regex()
```

FUNCIONES DE CLASIFICACIÓN Y ORQUESTACIÓN

```
>classify_prime_category()  
>get_prioritized_specs_and_category()  
>determine_market_segment()  
>process_data()
```

```
# regex_analyzer.py (Resumido)  
  
def extract_specs_regex(text):  
    """Extrae hardware usando patrones complejos (Regex)"""  
    # Detecta "16GB", "8gb RAM", pero evita "128GB SSD"  
    ram = re.findall(r'\b(\d+)\s*(?:gb|gigas?)\b(?:!\s*ssd)', text)  
  
    # Detecta i7, Ryzen 5, M1...  
    cpu = re.findall(r'\b(core\s*-?)?(i[3579])\b', text)  
  
    return {"ram": max(ram), "cpu": clean_cpu(cpu)}  
  
def get_prioritized_specs_and_category(title, desc):  
    """Función principal de análisis"""  
    # 1. Limpieza de Spam (elimina listas de keywords falsas)  
    clean_desc = smart_truncate_spam(desc)  
  
    # 2. Extracción de Specs  
    specs = extract_specs_regex(title + " " + clean_desc)  
  
    # 3. Clasificación (Gaming, Workstation, Apple...)  
    category = classify_category(specs)  
  
    return specs, category
```

REGEX_ANALYZER.PY

FUNCIONES DE UTILIDAD (PRECIOS Y TEXTO)

```
>clean_price()  
>try_extract_hidden_price()  
>is_match()  
>smart_truncate_spam()  
>sanitize.hardware_ambiguities()
```

FUNCIONES DE ANÁLISIS DE HARDWARE

```
>detect_condition_from_data()  
>apply_category_constraints()  
>is_valid_ram()  
>clean_cpu_string()  
>clean_gpu_string()  
>extract_ram()  
>extract_specs_regex()
```

FUNCIONES DE CLASIFICACIÓN Y ORQUESTACIÓN

```
>classify_prime_category()  
>get_prioritized_specs_and_category()  
>determine_market_segment()  
>process_data()
```

```
{  
    "APPLE": {  
        "LIKE_NEW": {  
            "mean": 653.52,  
            "median": 590.0,  
            "stdev": 446.55,  
            "count": 1673,  
            "components": {  
                "cpu": {  
                    "APPLE M1": {  
                        "mean": 593.78,  
                        "median": 550.0,  
                        "stdev": 178.42,  
                        "count": 351  
                    },  
                    "APPLE M3": {  
                        "mean": 849.68,  
                        "median": 800.0,  
                        "stdev": 264.02,  
                        "count": 145  
                    },  
                    "APPLE M2": {  
                        "mean": 725.14,  
                        "median": 700.0,  
                        "stdev": 169.06,  
                        "count": 235  
                    },  
                    "INTEL I5": {  
                        "mean": 326.67,  
                        "median": 300.0,  
                        "stdev": 127.18,  
                        "count": 151  
                    },  
                    "INTEL I7": {  
                        "mean": 524.17,  
                        "median": 480.0,  
                        "stdev": 271.7,  
                        "count": 179  
                    },  
                    "AMD RYZEN 7": {  
                        "mean": 619.72,  
                        "median": 599.5,  
                        "stdev": 217.1,  
                        "count": 80  
                    },  
                    "INTEL I5": {  
                        "mean": 413.8,  
                        "median": 385.0,  
                        "stdev": 202.47,  
                        "count": 80  
                    }  
                }  
            }  
        }  
    }  
    "GAMING": {  
        "USED": {  
            "mean": 607.38,  
            "median": 500.0,  
            "stdev": 466.62,  
            "count": 621,  
            "components": {  
                "cpu": {  
                    "INTEL I9": {  
                        "mean": 1654.84,  
                        "median": 1520.0,  
                        "stdev": 837.43,  
                        "count": 25  
                    },  
                    "INTEL I7": {  
                        "mean": 524.17,  
                        "median": 480.0,  
                        "stdev": 271.7,  
                        "count": 179  
                    },  
                    "INTEL I5": {  
                        "mean": 413.8,  
                        "median": 385.0,  
                        "stdev": 202.47,  
                        "count": 80  
                    }  
                }  
            }  
        }  
    }  
}
```

RUN_PIPELINE.SH

Código resumido

```
# 1. Ejecutar recolección (Poller)
echo "[*] Ejecutando Poller..."
python3 poller/poller.py

# 2. Verificar si se generaron datos
if [ -f "$JSON_FILE" ]; then
    # 3. Cargar en Elasticsearch
    python3 ingestion/bulk_ingest.py "$JSON_FILE"

    # 4. Limpieza si hubo éxito
    rm "$JSON_FILE"
fi
```

POLLER.PY

```
# poller.py - Parte 1: Ingesta Inteligente

def run_smart_poller():
    params = {
        "category_id": "24200", # Informática
        "time_filter": "today", # Solo anuncios frescos
        "longitude": "-3.7038", "latitude": "40.4168"
    }

    while len(all_items) < LIMIT:
        # 1. Petición a la API pública
        response = make_request("https://api.wallapop.com/api/v3/search", params)
        items = response.json()["items"]

        for item in items:
            # 2. Corrección de precios "fake" (ej: "Vendo por 1€" -> busca 500€ en desc.)
            if item["price"]["amount"] < 5:
                real_price = regex_analyzer.try_extract_hidden_price(item["title"], item["description"])
                if real_price: item["price"]["amount"] = real_price

            # 3. Deep Fetch para obtener el estado REAL (Nuevo/Usado)
            # La búsqueda normal no dice si es "Reacondicionado" o "Como nuevo"
            details = get_item_details_full(item["id"])
            real_condition = details.get("type_attributes", {}).get("condition")

            # 4. Calcular riesgo y guardar
            risk_data = calculate_risk_base(item, force_condition=real_condition)
            item["enrichment"] = risk_data
            all_items.append(item)
```

POLLER.PY

CÁLCULO DEL FACTOR DE RIESGO

Implementa un sistema de scoring multi-factor que combina:

- Análisis estadístico de precios (Z-scores)
- Heurísticas de comportamiento sospechoso
- Comparación con valores de mercado por componente

Resultado del análisis con estructura:

- risk_score (int):

Puntuación 0-100 (mayor = más riesgo)

- risk_factors (List[str]):

Lista de factores de riesgo detectados

- market_analysis (Dict):

Detalles del análisis de mercado

Algoritmo de puntuación:

- Precio estadísticamente bajo ($Z < -1.5$): +30 puntos
- Anomalía extrema ($Z < -2.5$): +40 puntos adicionales
- Precio < 40% del valor estimado: +20 puntos
- Descripción corta (< 30 chars) con precio alto: +15 puntos
- Contacto externo (WhatsApp, teléfono): +30 puntos

```
# poller.py - Parte 2: Cálculo de Riesgo

def calculate_risk_base(item, condition):
    score = 0
    factors = []

    # 1. Analizar texto para sacar Hardware (i7, 16GB...) y Categoría (Gaming...)
    specs, category, _ = regex_analyzer.get_prioritized_specs_and_category(item["title"],
                                                                           item["description"])

    # 2. Buscar estadísticas de referencia para ese hardware exacto
    # MARKET_STATS es un JSON pre-cargado con precios medios reales
    stats = MARKET_STATS.get(category, {}).get(condition, {})

    # 3. Calcular Z-Score (Desviación del precio respecto a la media)
    # Si Z < -1.5 significa que es estadísticamente demasiado barato
    z_score = (item["price"] - stats["mean"]) / stats["stdev"]

    if z_score < -1.5:
        score += 30
        factors.append(f"Statistically Cheap (Z={z_score:.2f})")

    if z_score < -2.5: # Anomalía extrema
        score += 40
        factors.append("EXTREME Price Anomaly")

    # 4. Detectar comportamientos sospechosos (Heurística)
    if "whatsapp" in item["description"] or "600000000" in item["description"]:
        score += 30
        factors.append("External Contact")

    return {"risk_score": min(score, 100), "risk_factors": factors}
```

BULKING, KIBANA & ELAST ALERTS



<http://kibana.lan:5601>

4 BULK_INGEST.PY

Lee los ficheros JSON diarios generados por el poller y los envia a Elasticsearch de forma masiva y eficiente usando la API _bulk. Su función es transformar los registros de texto en documentos indexados y consultables.

5 ILM_POLICY.JSON, INDEX_ALIAS.JSON, INDEX_TEMPLATE.JSON

Arquitectura de almacenamiento. Configuran el ciclo de vida para rotar índices antiguos (ILM) , nombre estable de escritura (Alias) y definen los tipos de datos (como geo-puntos o precios) para asegurar que Kibana visualice todo correctamente (Template).

6 HIGH_RISK.YAML

Regla de detección que ElastAlert ejecuta periódicamente contra Elasticsearch

BULK_INGEST.PY

CONFIGURACIÓN

```
ES_URL = "http://localhost:9200"  
INDEX_ALIAS = "lab10310.wallapop"  
BATCH_SIZE = 1000
```

Se envían en lotes de 1000 documentos por petición HTTP para no saturar la memoria.

Cada documento se compone de dos líneas, una de acción (indexar) y otra con los datos del documento.

Flujo:

1. Leer línea por línea (streaming)
2. Parsear cada línea como JSON
3. Agrupar en lotes de BATCH_SIZE documentos
4. Enviar cada lote a Elasticsearch
5. Procesar documentos restantes al finalizar

```
def send_batch(lines: List[str]) -> None:  
    # Envía un lote de líneas NDJSON a Elasticsearch  
    bulk_body = "\n".join(lines) + "\n"  
    try:  
        response = requests.post(f"{ES_URL}/_bulk", data=bulk_body.encode("utf-8"),  
                                 headers={"Content-Type": "application/x-ndjson"}, timeout=60 # Timeout extendido para lotes grandes)  
        if response.status_code == 200:  
            resp = response.json()  
  
def bulk_ingest(json_file_path: str) -> None:  
    file_path = Path(json_file_path)  
    # Variables de control para el procesamiento por lotes  
    current_batch = [] # Buffer temporal para el lote actual  
    doc_count = 0 # Contador de documentos en el lote actual  
    total_processed = 0 # Contador total de documentos procesados  
    try:  
        with file_path.open("r", encoding="utf-8") as f:  
            for line in f:  
                line = line.strip()  
                # Intentar parsear la línea como JSON  
                try:  
                    doc = json.loads(line)  
                    # Construir la estructura bulk: acción + documento  
                    # La acción especifica el índice destino  
                    action = {"index": {"_index": INDEX_ALIAS}}  
                    current_batch.append(json.dumps(action))  
                    current_batch.append(json.dumps(doc, ensure_ascii=False))  
                    doc_count += 1  
                    # Cuando alcanzamos el tamaño del lote, enviamos a Elasticsearch  
                    if doc_count >= BATCH_SIZE:  
                        send_batch(current_batch)  
                        total_processed += doc_count  
                        current_batch = [] # Limpiar buffer para el siguiente lote  
                        doc_count = 0  
                print(f"[*] Proceso finalizado. Total documentos procesados: {total_processed}")
```

ILM_POLICY.JSON INDEX_ALIAS.JSON

ilm_policy.json (Política de Ciclo de Vida)
Define las "reglas de caducidad" de los datos para gestionar el espacio en disco automáticamente.

- Rotación (Hot): Crea un índice nuevo si el actual supera 1GB o 1 día de antigüedad (Rollover).
- Limpieza (Delete): Borra automáticamente los datos que tengan más de 30 días.

index_alias.json (Inicialización del Alias)
Crea el primer índice físico y el Alias para la escritura.

- Abstracción: Permite que los scripts envíen datos siempre al nombre genérico lab10310.wallapop sin preocuparse de si el índice actual es el -000001, -000002, etc.
- Escritura: Marca este primer índice como el destino inicial de los datos (is_write_index: true).

```
PUT _ilm/policy/lab10310-wallapop-rotation
{
  "policy": {
    "phases": {
      "hot": {
        "actions": {
          "rollover": {
            "max_size": "1gb",
            "max_age": "1d"
          }
        }
      },
      "delete": {
        "min_age": "30d",
        "actions": {
          "delete": {}
        }
      }
    }
  }
}
```

```
PUT lab10310.wallapop-000001
{
  "aliases": {
    "lab10310.wallapop": {
      "is_write_index": true
    }
  }
}
```

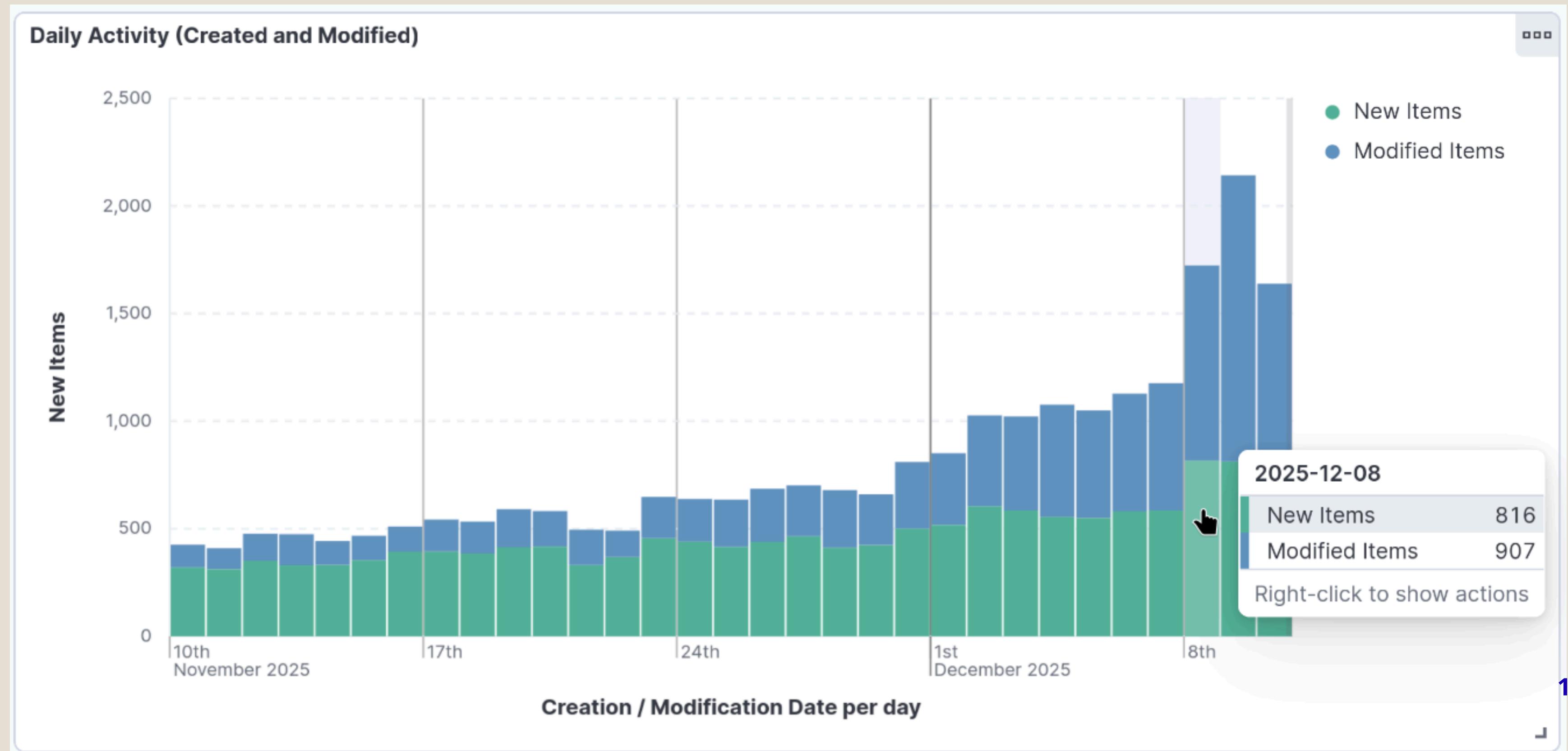
INDEX_TEMPLATE.JSON

Define cómo se debe guardar la información cada vez que se crea un nuevo índice (diariamente o por rotación), asegurando que los datos sean coherentes y buscables.

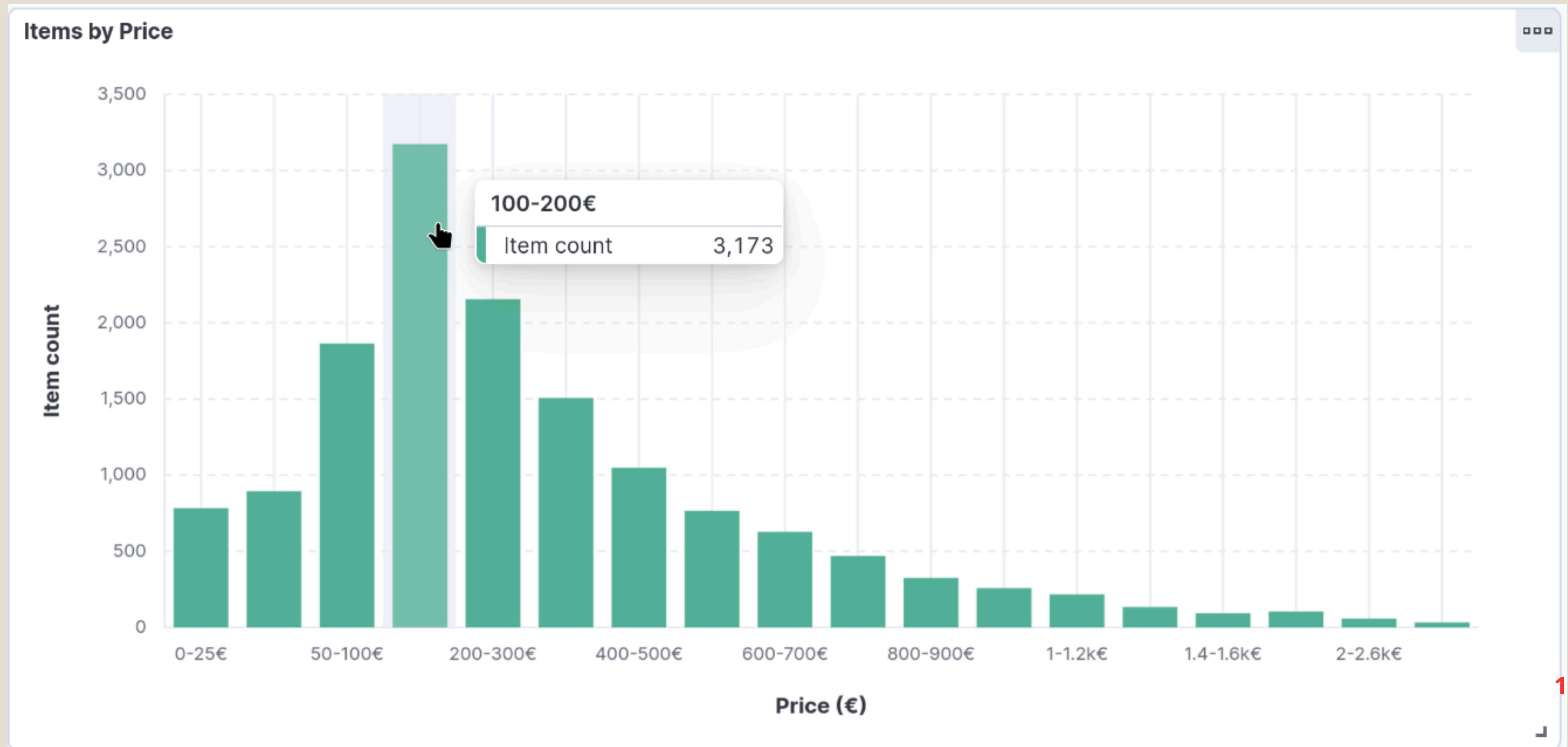
- Estructura de Datos (Mapping): Obliga a que cada campo tenga el formato correcto:
 - price: Número decimal (para poder hacer rangos).
 - location.geo: Coordenada geográfica (para mapas).
 - timestamps: Fecha real (para líneas de tiempo).
- Automatización: Asocia automáticamente los nuevos índices con la política de ciclo de vida (lab10310-wallapop-rotation) para que se roten y borren solos sin intervención manual.

```
PUT _index_template/lab10310-wallapop-template
{
  "index_patterns": ["lab10310.wallapop*"],
  "template": {
    "settings": {
      "index.lifecycle.name": "lab10310-wallapop-rotation",
      "index.lifecycle.rollover_alias": "lab10310.wallapop",
      "number_of_shards": 1,
      "number_of_replicas": 0
    },
    "mappings": {
      "dynamic_templates": [
        {
          "strings_as_keywords": {
            "match_mapping_type": "string",
            "mapping": {
              "type": "keyword",
              "ignore_above": 256
            }
          }
        }
      ],
      "properties": {
        "id": { "type": "keyword" },
        "title": {
          "type": "text",
          "fields": {
            "keyword": { "type": "keyword", "ignore_above": 256 }
          }
        },
        "description": { "type": "text" },
        "price": {
          "properties": {
            "amount": { "type": "float" },
            "currency": { "type": "keyword" }
          }
        },
        "category_id": { "type": "keyword" },
        "user_id": { "type": "keyword" },
        "web_slug": { "type": "keyword" },
        "created_at": { "type": "date" },
        "modified_at": { "type": "date" },
        "modified_date": { "type": "date" },
        "location": {
          "properties": {
            "geo": { "type": "geo_point" },
            "latitude": { "type": "float" },
            "longitude": { "type": "float" },
            "city": { "type": "keyword" },
            "postal_code": { "type": "keyword" },
            "country_code": { "type": "keyword" }
          }
        },
        "timestamps": {
          "properties": {
            "crawl_timestamp": { "type": "date" }
          }
        },
        "enrichment": {
          "properties": {
            "risk_score": { "type": "integer" },
            "risk_factors": { "type": "keyword" },
            "market_analysis": {
              "properties": {
                "detected_category": { "type": "keyword" },
                "composite_z_score": { "type": "float" },
                "estimated_market_value": { "type": "float" },
                "components_used": { "type": "keyword" },
                "specs_detected": {
                  "properties": {
                    "cpu": { "type": "keyword" },
                    "ram": { "type": "keyword" },
                    "gpu": { "type": "keyword" }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

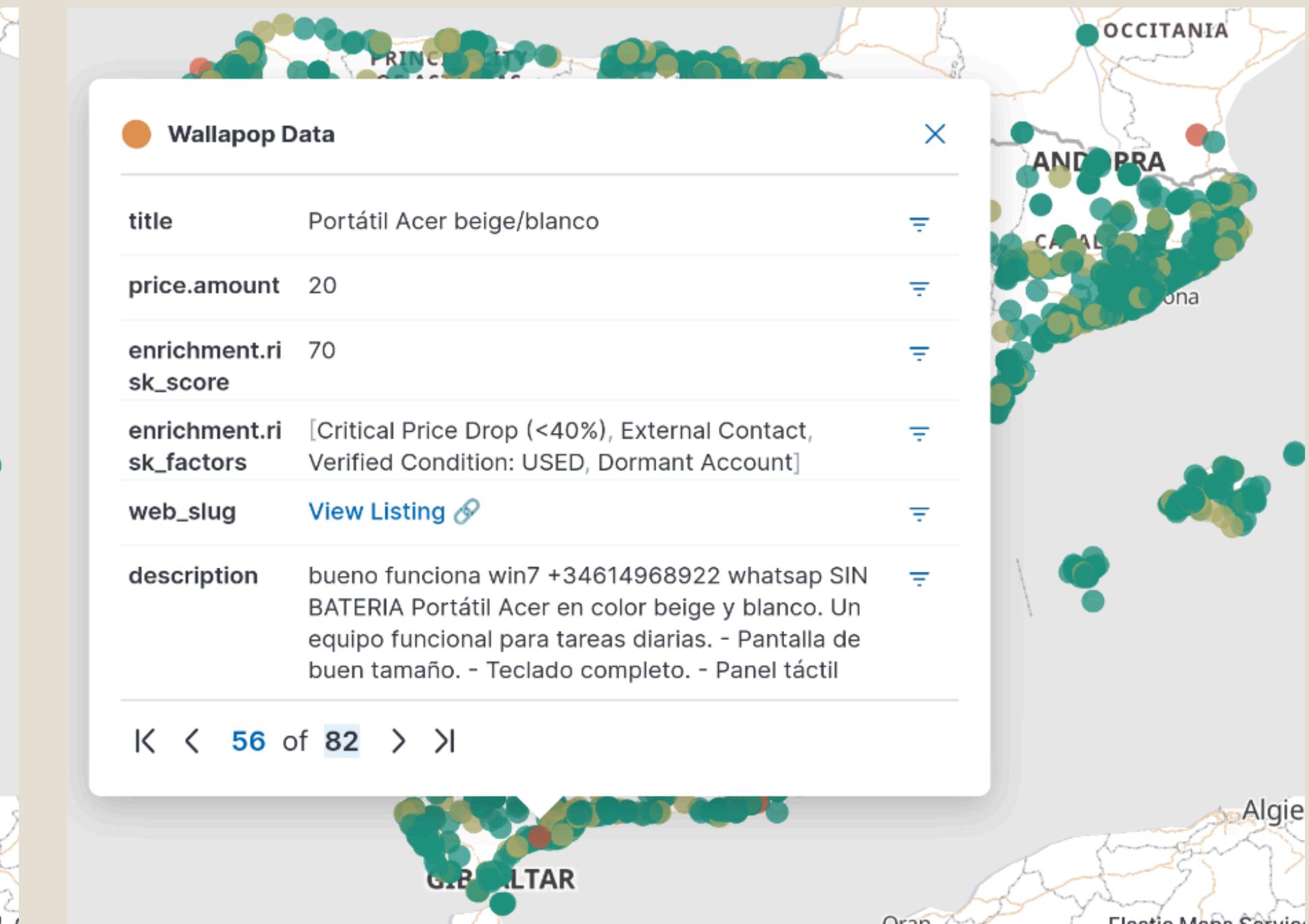
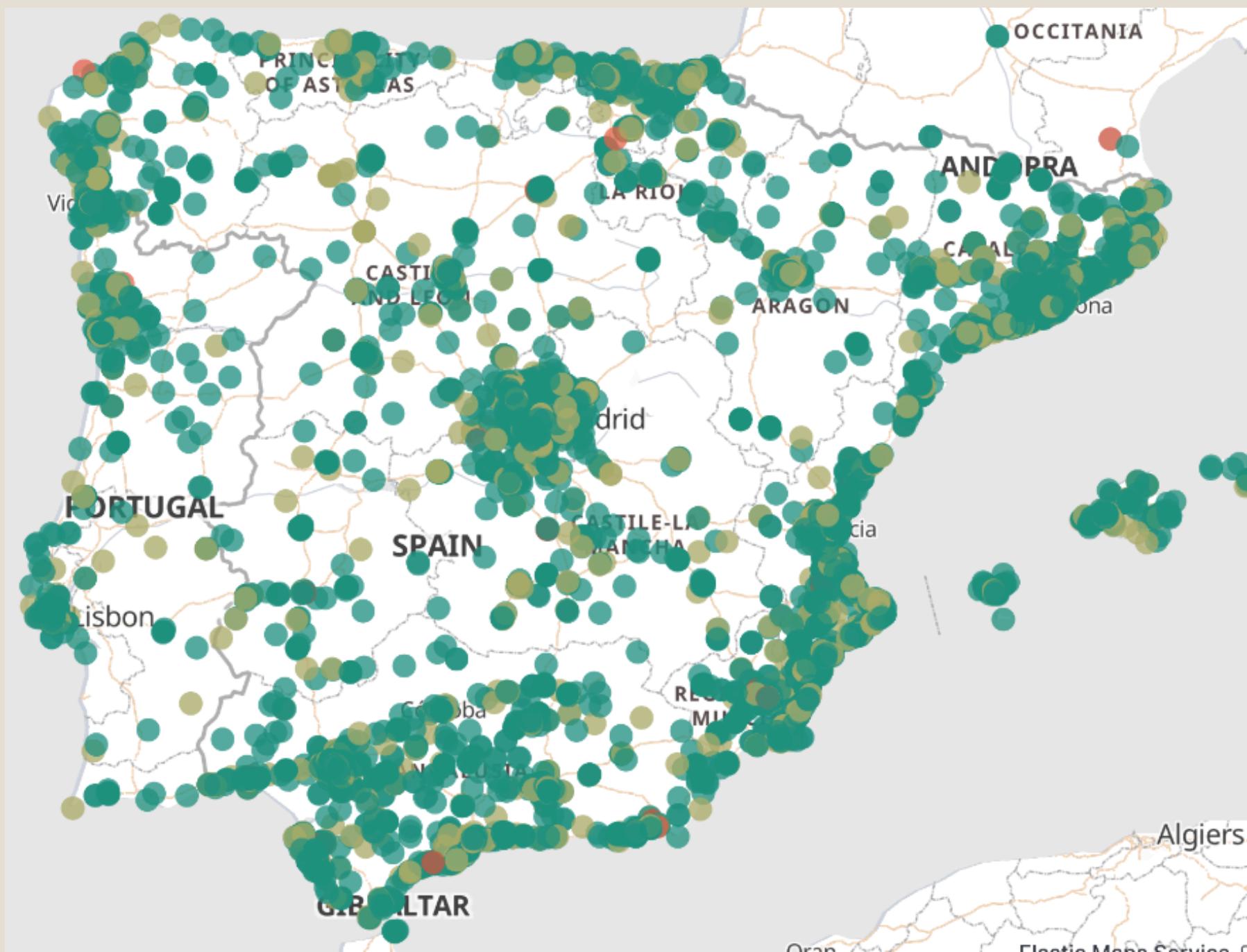
DASHBOARD: ACTIVIDAD DIARIA



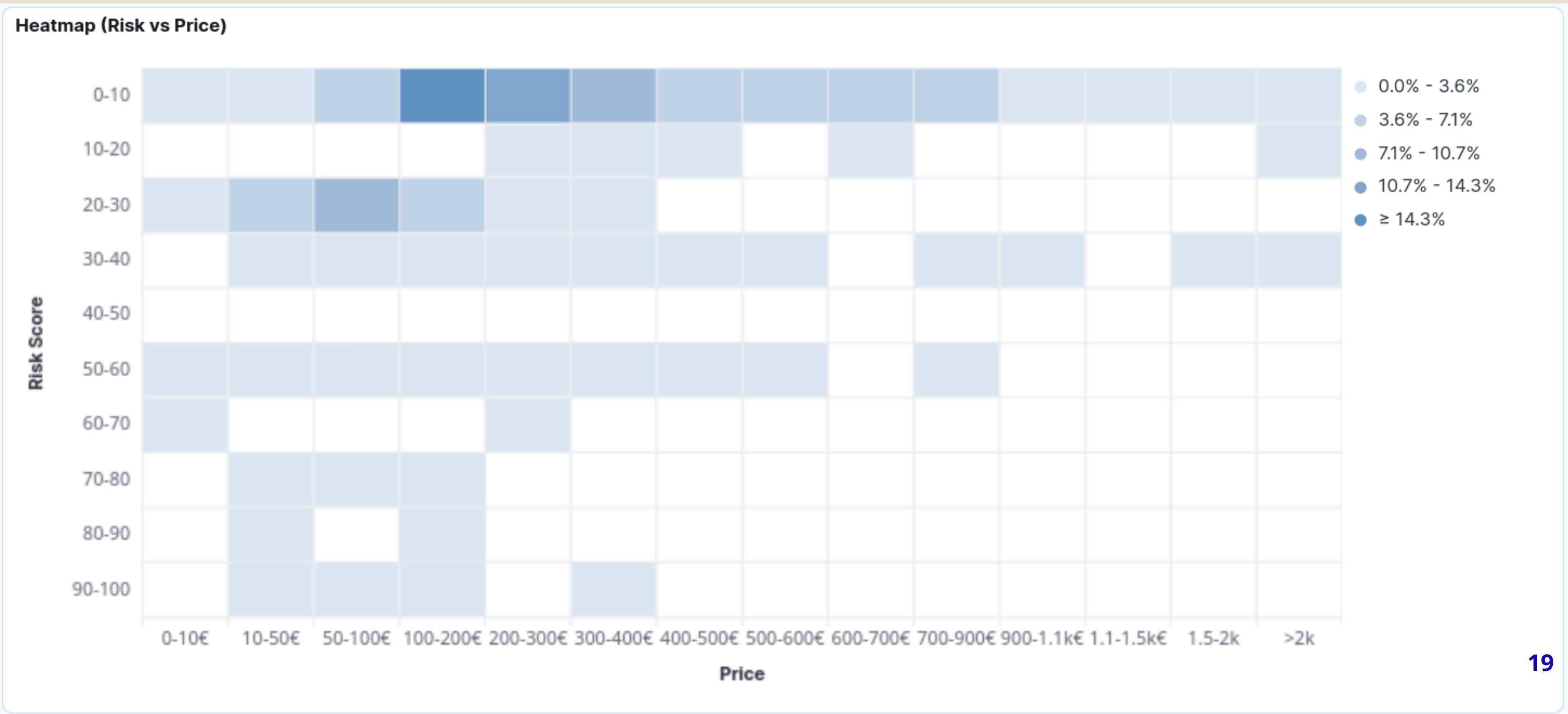
DASHBOARD: HISTOGRAMA DE PRECIOS



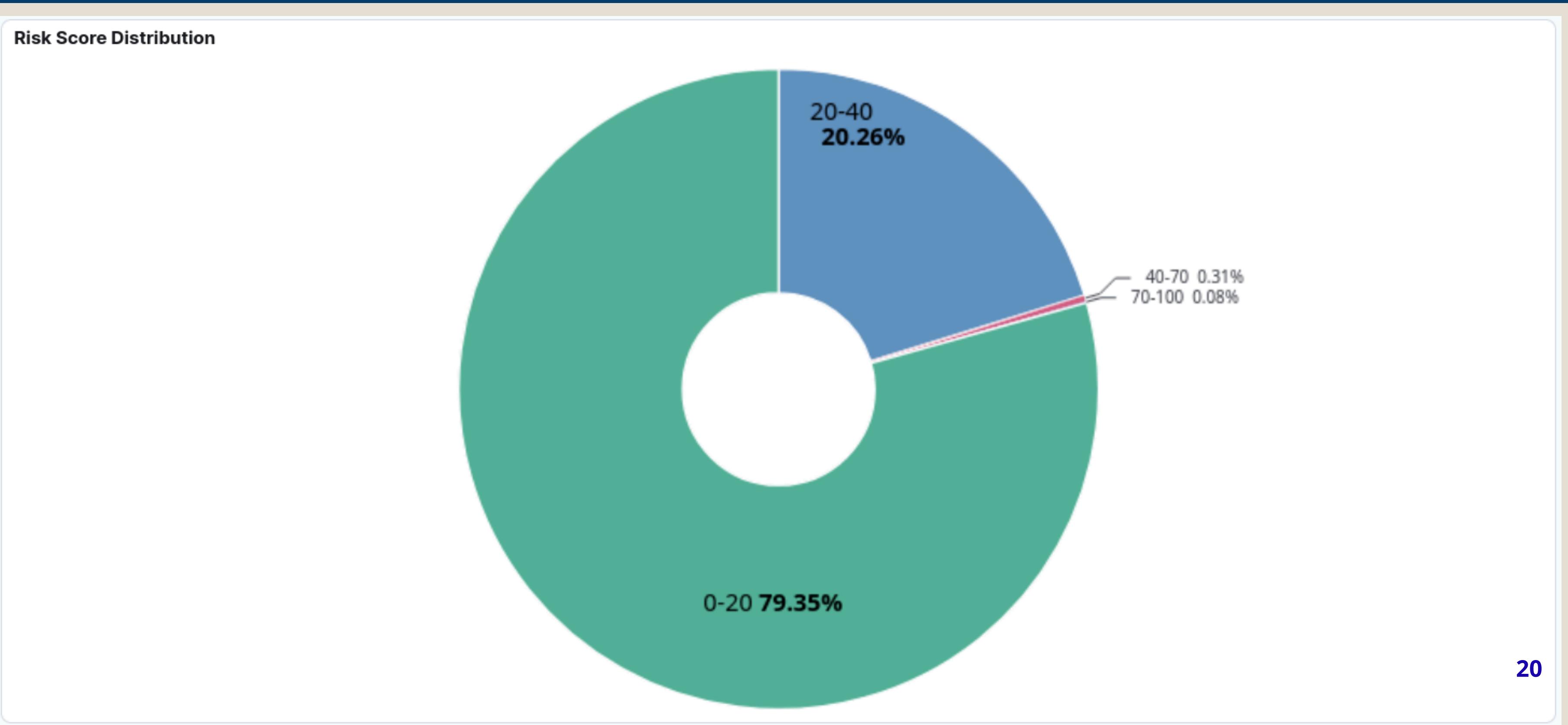
GEO MAPA CON FACTOR DE RIESGO



HEATMAP (FACTOR DE RIESGO VS PRECIO)



DISTRIBUCIÓN DEL FACTOR DE RIESGO



ANUNCIOS CON MAYOR FACTOR DE RIESGO

Top Risk Score Items						
Risk Score ↓	Title	Price	Timestamp	Link	Risk Factors	
90	MacBook Air M2 Dorado	100	Dec 5, 2025 @ 16:36:52.007	View Listing	[Critical Price Drop (<40%), EXTREME Price Anomaly, External Contact Attempt]	21
90	Caja MacBook Air 13 M2 Gris/Plata	20	Nov 16, 2025 @ 14:13:39.882	View Listing	[Critical Price Drop (<40%), EXTREME Price Anomaly, External Contact Attempt]	21
90	Lenovo ThinkPad T450 Negro	80	Dec 8, 2025 @ 14:02:31.321	View Listing	[Critical Price Drop (<40%), EXTREME Price Anomaly, External Contact Attempt]	21
90	Pantalla Apple A2779 Original	300	Dec 8, 2025 @ 21:31:46.401	View Listing	[Critical Price Drop (<40%), EXTREME Price Anomaly, External Contact Attempt]	21
80	MacBook Pro 13 2012 Intel Core i5 4G	150	Dec 9, 2025 @ 23:49:36.513	View Listing	[Critical Price Drop (<40%), External Contact Attempt]	21
80	HP M01	125	Dec 10, 2025 @ 16:19:07.472	View Listing	[Critical Price Drop (<40%), New User (<48h), Statistical Anomaly]	21
80	MSI GL63 8RD en Venta Varios Repuest	15	Dec 5, 2025 @ 18:07:26.130	View Listing	[Critical Price Drop (<40%), External Contact Attempt]	21
70	Portátil Lenovo Negro Pequeño	30	Dec 8, 2025 @ 13:16:54.445	View Listing	[Critical Price Drop (<40%), No Reputation / Dormant, External Contact Attempt]	21
70	Portátil Acer beige/blanco	20	Dec 2, 2025 @ 08:56:50.206	View Listing	[Critical Price Drop (<40%), External Contact Attempt]	21
70	Acer Chromebook Plus 514 Ryzen 5 Ra	150	Dec 10, 2025 @ 12:39:44.606	View Listing	[EXTREME Price Anomaly, Statistically Cheap (Z-Score > 2.5)]	21
70	Samsung RC530 - Intel Core i7 - 500GI	99	Nov 26, 2025 @ 12:37:13.380	View Listing	[Critical Price Drop (<40%), No Reputation / Dormant, External Contact Attempt]	21
60	Portátil ASUS Gaming Negro	280	Dec 10, 2025 @ 14:32:35.266	View Listing	[External Contact Attempt, Statistically Cheap (Z-Score > 2.5)]	21

LIMITACIONES Y CONCLUSIONES

Logros del Proyecto:

- Implementación exitosa de un pipeline E2E (Recolección → Ingesta → Visualización).

Limitaciones Actuales:

- Ceguera Contextual: Las expresiones regulares buscan patrones, no entienden el significado.
 - Ejemplo: "Pantalla para MacBook Pro" se detecta erróneamente como un portátil con "Categoría: MacBook Pro".
- Extracción Frágil: La variabilidad humana al escribir (faltas de ortografía, abreviaturas no estándar) hace que muchas características legítimas se pierdan o clasifiquen mal.
- Todo esto produce falsos positivos (o falsos negativos).

¿Solución? → Integración de LLMs para análisis semántico del texto
y clasificación.

