

Proyecto final Análisis Masivos

Cristian Arturo Paz Alvarez

Alonso Esteban Amaya Gil

Facultad de ingeniería, Universitaria Agustiniana

Machine Learning

Profesor Juan Sebastián Martínez Conejo

9 de septiembre, 2025

Resumen

Este proyecto aborda la dificultad de predecir el valor de las cartas del Juego de Cartas Coleccionables de Pokémon (Pokémon TCG) ante la falta de datos históricos en el momento de su lanzamiento. El objetivo es construir un modelo de *Machine Learning* para estimar la probabilidad de que una carta sea económicamente valiosa, basándose en atributos intrínsecos como su rareza, tipo y jugabilidad. Se utilizó un enfoque de clasificación binaria y un *dataset* de 19,500 registros obtenido de la API de Pokémon TCG. El análisis exploratorio de datos (EDA) reveló que la rareza, el número de ataques y el daño promedio son variables claves relacionadas con el precio. Para manejar el desbalance de clases, se definió la variable objetivo binaria (carta 'cara' o 'no cara') utilizando el percentil 95 del precio de mercado. A pesar de retos como datos faltantes y valores atípicos, el proyecto logró consolidar un conjunto de datos limpio y documentado, listo para la fase de entrenamiento del modelo.

Palabras clave: Pokémon TCG, *Machine Learning*, análisis predictivo, clasificación, *dataset*.

Contenido

| | |
|---|-----------|
| Resumen | 2 |
| Introducción..... | 4 |
| Planteamiento del problema y justificación | 5 |
| Objetivos | 6 |
| Objetivo general | 6 |
| Objetivos específicos..... | 6 |
| Marco teórico..... | 7 |
| Metodología..... | 8 |
| Base de datos seleccionada y fuente | 8 |
| Configuración y despliegue..... | 8 |
| Definición de la variable objetivo..... | 8 |
| Análisis de volumen y calidad | 9 |
| Retos encontrados | 9 |
| Resultados iniciales (EDA) | 9 |
| Preprocesamiento y limpieza de datos..... | 10 |
| Referencias | 20 |
| Glosario de términos | 21 |

Introducción

El mercado de cartas coleccionables de Pokémon TCG presenta una gran dinámica, donde el valor de cada carta fluctúa según su rareza, jugabilidad y demanda. Predecir el precio de una carta nueva al momento de su lanzamiento es un desafío, ya que no se cuenta con histórico de variaciones ni datos en tiempo real. Este proyecto plantea la construcción de un modelo de Machine Learning que permita, a partir de atributos estáticos de las cartas (rareza, tipo, ataques, habilidades), estimar la probabilidad de que una carta sea valiosa. El objetivo es ofrecer una herramienta útil tanto para coleccionistas como para jugadores y comerciantes.

Planteamiento del problema y justificación

El mercado de cartas de Pokémon TCG es muy dinámico, y el valor de las cartas nuevas a menudo es impredecible en su lanzamiento debido a la falta de datos históricos y en tiempo real. Esto representa un desafío significativo para coleccionistas, jugadores y comerciantes, ya que la incertidumbre sobre el valor de una carta dificulta la toma de decisiones informadas sobre su adquisición.

La justificación de este proyecto radica en la necesidad de una herramienta predictiva que, basándose en los **atributos intrínsecos de las cartas** (como rareza, tipo y jugabilidad), pueda estimar la probabilidad de que una carta se vuelva valiosa. La construcción de este modelo de *Machine Learning* no solo solventará el problema práctico de anticipar el valor de las cartas, sino que también servirá como un caso de estudio sobre la aplicación de técnicas de aprendizaje automático en mercados coleccionables.

Objetivos

Objetivo general

Construir un modelo de *Machine Learning* que prediga la probabilidad de que una carta de Pokémon TCG pertenezca al segmento de mayor valor económico.

Objetivos específicos

- **Integrar** en un *dataset* maestro los atributos de las cartas con sus precios.
- **Definir** una variable objetivo binaria que clasifique cartas 'caras' y 'no caras'.
- **Realizar** un análisis exploratorio que permita identificar patrones de valor.
- **Preparar** el *dataset* limpio y documentado para entrenar el modelo en el siguiente corte.

Marco teórico

El presente estudio se basa en los fundamentos del aprendizaje supervisado y la clasificación binaria, paradigmas centrales en el campo de la ciencia de datos. A continuación, se revisan los conceptos teóricos y los factores de mercado que justifican la aplicación de estas técnicas en el análisis predictivo de cartas coleccionables.

Aprendizaje Supervisado y Clasificación

El **aprendizaje supervisado** es un enfoque de *Machine Learning* en el que un algoritmo aprende a partir de un conjunto de datos etiquetados para predecir resultados en datos nuevos. En esta rama, la **clasificación binaria** es una tarea predictiva específica que divide los datos en dos clases posibles: 'carta cara' o 'carta no cara'. Para evaluar el rendimiento de estos modelos se emplean métricas especializadas como **ROC-AUC**, **PR-AUC** y el **F1-score**, que son cruciales para manejar el desbalance de clases, una problemática común en este tipo de análisis (Géron, 2022).

Factores de Valor en Pokémon TCG

El valor económico de una carta de Pokémon TCG es una variable compleja que está influenciada por múltiples factores. Raschka (2020) señala que la **rareza** de una carta es un predictor fundamental de su valor potencial, lo cual se alinea con la hipótesis de este estudio. Además de la rareza, la **jugabilidad** (su utilidad en el metajuego competitivo) y el **atractivo para los coleccionistas** también son atributos intrínsecos que contribuyen de manera significativa a la demanda y, por ende, al precio de una carta.

En resumen, el marco teórico de este proyecto se cimienta en el aprendizaje supervisado para resolver una tarea de clasificación binaria, utilizando métricas robustas para enfrentar el desbalance de clases. La revisión de la literatura confirma que atributos como la rareza y la jugabilidad son variables clave para predecir el valor de las cartas, lo que valida la aproximación metodológica de este estudio.

Metodología

Base de datos seleccionada y fuente

El dataset empleado corresponde a cartas de Pokémon TCG, con 19.500 registros y tablas auxiliares de ataques, habilidades, debilidades, resistencias y precios. La fuente de datos es la API de Pokémon TCG y archivos extraídos en formatos CSV y Parquet.

Configuración y despliegue

El procesamiento de datos se realizó en Python, empleando librerías como Pandas y Scikit-learn. Se integraron las distintas tablas en un archivo maestro llamado `ptcg_master_clean.parquet`. Se aplicaron procesos de limpieza de cadenas, imputación de nulos y creación de nuevas variables agregadas como el número de ataques, daño promedio, número de habilidades, debilidades y resistencias.

Definición de la variable objetivo

Se seleccionó la columna 'market' (precio de TCGPlayer) como referencia principal. A partir de ella se calculó el percentil 95 (p_{95}) para establecer la etiqueta binaria: 1 = Carta cara ($\geq p_{95}$) y 0 = Carta no cara ($< p_{95}$).

Análisis de volumen y calidad

El dataset maestro consolidado cuenta con 32.643 filas y 62 columnas. Entre las variables con mayor cantidad de valores nulos se encuentran 'level' (86%), 'n_abilities' (81%) y 'rules' (75%). Se definieron variables cuantitativas (número de ataques, daño promedio, número de habilidades, debilidades y resistencias, coste de retirada) y cualitativas (rareza, supertipo, subtipos, tipos, set).

Retos encontrados

- Desbalance de clases: pocas cartas entran en la categoría de 'caras'.
- Datos faltantes en atributos como nivel o habilidades.
- Outliers: precios extremadamente altos en rarezas especiales.
- Alta cardinalidad: gran número de sets y subtipos distintos.

Resultados iniciales (EDA)

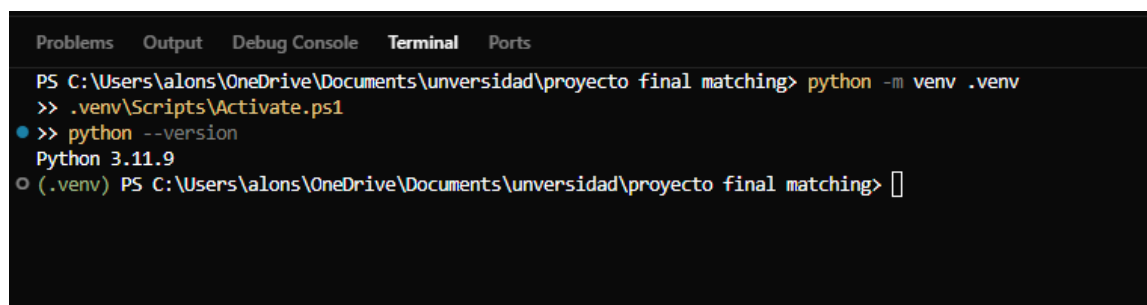
El análisis exploratorio inicial mostró que rarezas como 'Rare Holo Star' y 'Rare Shining' concentran precios medios muy altos. La mayoría de rarezas comunes y poco comunes tienen precios bajos. Se observa que variables como el número de ataques, el daño máximo y la rareza guardan relación con el precio. El dataset maestro final quedó consolidado en el archivo `ptcg_master_clean.parquet`, listo para el siguiente corte.

Preprocesamiento y limpieza de datos

1. Configuración del entorno de trabajo

Para garantizar un entorno de desarrollo aislado y reproducibilidad en el proyecto, se configuró un entorno virtual en Python. Tal como se ilustra en el **Comando 1**, se utilizó el comando `python -m venv .venv` para crear el entorno, y posteriormente se activó con el *script* correspondiente. El proyecto se ejecutó con la versión de Python 3.11.9.

Comando 1



```

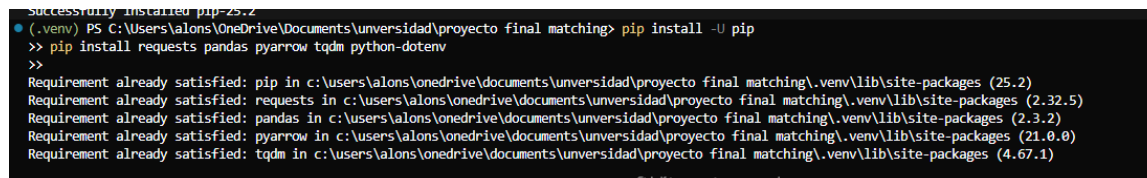
Problems  Output  Debug Console  Terminal  Ports
PS C:\Users\alons\OneDrive\Documents\universidad\proyecto final matching> python -m venv .venv
>> .venv\Scripts\Activate.ps1
● >> python --version
Python 3.11.9
○ (.venv) PS C:\Users\alons\OneDrive\Documents\universidad\proyecto final matching>

```

2. Instalación de librerías y dependencias

Tras la configuración del entorno virtual, se procedió a instalar las librerías necesarias para la extracción y el procesamiento de los datos. Se utilizó el gestor de paquetes de Python pip para instalar las librerías requests, pandas, pyarrow, tqdm y python-dotenv. Como se muestra en el **Comando 2**, estas herramientas son esenciales para la extracción de datos de la API, la manipulación de los *datasets* y la gestión de variables de entorno, respectivamente

Comando 2



```

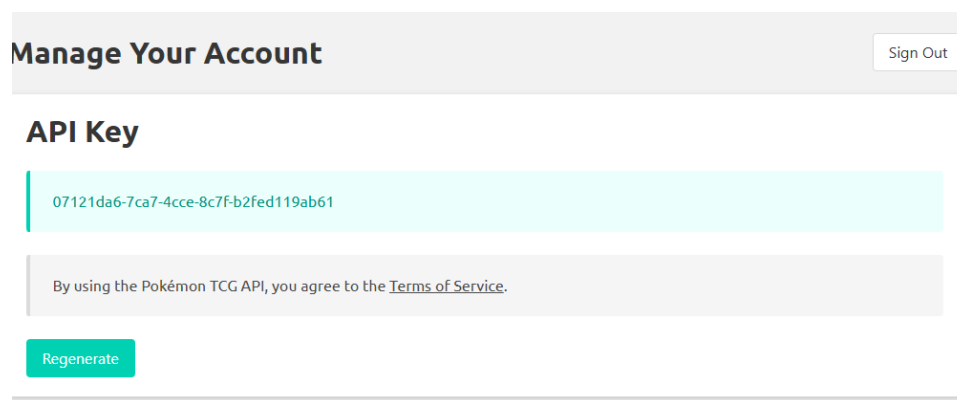
Successfully installed pip-25.2
(.venv) PS C:\Users\alons\OneDrive\Documents\universidad\proyecto final matching> pip install -U pip
>> pip install requests pandas pyarrow tqdm python-dotenv
>>
Requirement already satisfied: pip in c:\users\alons\onedrive\documents\universidad\proyecto final matching\.venv\lib\site-packages (25.2)
Requirement already satisfied: requests in c:\users\alons\onedrive\documents\universidad\proyecto final matching\.venv\lib\site-packages (2.32.5)
Requirement already satisfied: pandas in c:\users\alons\onedrive\documents\universidad\proyecto final matching\.venv\lib\site-packages (2.3.2)
Requirement already satisfied: pyarrow in c:\users\alons\onedrive\documents\universidad\proyecto final matching\.venv\lib\site-packages (21.0.0)
Requirement already satisfied: tqdm in c:\users\alons\onedrive\documents\universidad\proyecto final matching\.venv\lib\site-packages (4.67.1)

```

3. Conexión a la API y obtención de la clave de acceso

Para poder extraer los datos, se generó una clave de acceso (API Key) a través del panel de control de **Pokémon TCG Developers**. Esta clave, visible en la **Ilustración 1**, es un identificador único que se utiliza para autenticar y autorizar las peticiones a la API. Esta clave se almacenó en una variable de entorno para evitar que fuera expuesta en el código. La documentación oficial de la API se consultó en la dirección: <https://dev.pokemontcg.io/dashboard>.

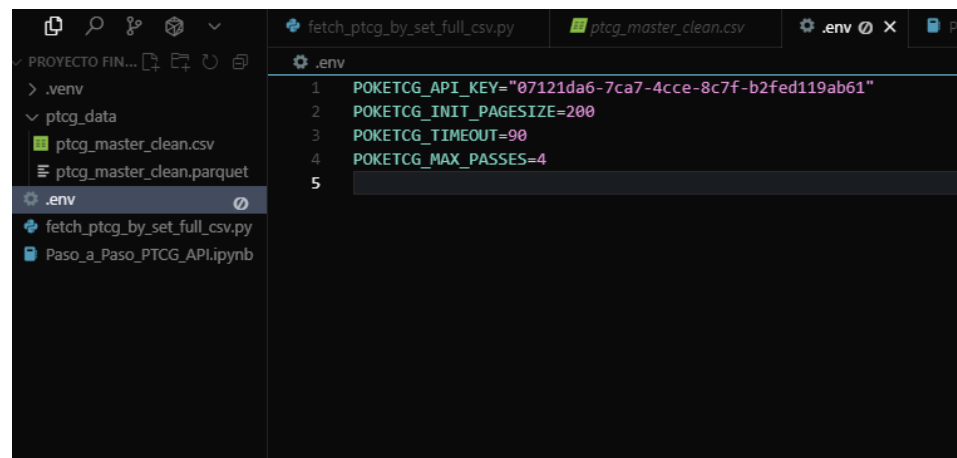
Ilustración 1



4. Configuración de variables de entorno

Como se muestra en el **Bloque de código 1**, la clave de acceso de la API y otros parámetros de configuración se almacenaron en un archivo de variables de entorno (.env) para garantizar que las credenciales sensibles no queden expuestas directamente en el código del proyecto. Esta práctica de seguridad es fundamental para la gestión de proyectos de *software*.

Bloque de código 1



5. Importación de librerías

El proceso de extracción de datos se inició con la importación de las librerías y módulos necesarios. Como se observa en el **Bloque de código 2**, se importaron librerías obligatorias como **requests** para realizar peticiones HTTP a la API, **pandas** para la manipulación y análisis de datos en formato de *dataframe*, y **tqdm** para visualizar el progreso de las operaciones. Adicionalmente, se adiciono el módulo **dotenv** para cargar las variables de entorno configuradas previamente.

Bloque de código 2

```
import os
import time
import random
from typing import Dict, Any, List, Set

import requests
import pandas as pd
from tqdm import tqdm
try:
    from dotenv import load_dotenv
    load_dotenv()
except Exception:
    pass
from requests.adapters import HTTPAdapter, Retry
```

6. Configuración de parámetros de extracción

Para el proceso de extracción de datos, se definieron parámetros clave que se visualizan en el **Bloque de código 3**. Estos incluyen la URL base de la API, la clave de acceso y los encabezados de autenticación. Además, se establecieron parámetros como el tiempo de espera para las peticiones (REQ_TIMEOUT), el tamaño inicial de las páginas (INIT_PAGESIZE) y el número máximo de pasadas (MAX_PASSES) para optimizar la descarga de datos.

Bloque de código 3

```
BASE = "https://api.pokemontcg.io/v2"
SETS_URL = f"{BASE}/sets"
CARDS_URL = f"{BASE}/cards"

API_KEY = os.getenv("POKETCG_API_KEY", "").strip()
HEADERS = {"X-API-Key": API_KEY} if API_KEY else {}

REQ_TIMEOUT = float(os.getenv("POKETCG_TIMEOUT", "60"))
INIT_PAGESIZE = int(os.getenv("POKETCG_INIT_PAGESIZE", "250"))
MAX_PASSES = int(os.getenv("POKETCG_MAX_PASSES", "3"))

OUT_DIR = "ptcg_data"
os.makedirs(OUT_DIR, exist_ok=True)
OUT_CSV = os.path.join(OUT_DIR, "ptcg_master_clean.csv")
```

7. Manejo de errores y reintentos automáticos

Para garantizar la solidez de la extracción, se instancio una función para crear una sesión HTTP con reintentos automáticos para errores de conexión o de servidor. Como se muestra en el **Bloque de código 4**, se configuró un adaptador HTTP con un número definido de reintentos (total=8), un factor de *backoff* y una lista de códigos de estado de error ([429, 500, 502, etc.]) que activan la lógica de reintento. Además, se diseñó una función para manejar específicamente los *timeouts* de conexión, lo que asegura que el proceso de descarga sea resiliente ante interrupciones de red.

Bloque de código 4

```
def make_session() -> requests.Session:
    s = requests.Session()
    retries = Retry(
        total=8, connect=8, read=8,
        backoff_factor=1.2,
        status_forcelist=[429, 500, 502, 503, 504],
        allowed_methods=["GET"],
        raise_on_status=False,
    )
    adapter = HTTPAdapter(max_retries=retries, pool_connections=30, pool_maxsize=30)
    s.mount("https://", adapter)
    s.mount("http://", adapter)
    return s

SESSION = make_session()

def safe_get(url: str, params: Dict[str, Any], headers: Dict[str, str],
            max_retries: int = 5, backoff: float = 1.5) -> requests.Response:
    """GET con reintentos adicionales para timeouts (Read/Connect)."""
    attempt = 0
    while True:
        attempt += 1
        try:
            return SESSION.get(url, params=params, headers=headers, timeout=REQ_TIMEOUT)
        except (requests.exceptions.ReadTimeout, requests.exceptions.ConnectTimeout):
            if attempt >= max_retries:
                raise
            time.sleep((backoff ** attempt) + random.uniform(0, 0.7))
```

8. Funciones auxiliares para la limpieza de datos

Se crearon funciones de apoyo para el procesamiento de los datos. El **Bloque de código 5** muestra dos funciones auxiliares: una llamada `list_to_str`, que se encarga de convertir listas de datos a cadenas de texto separadas por punto y coma, facilitando su almacenamiento en formato CSV. La otra función, `expected_from_set`, permite obtener el número total esperado de cartas por cada set, esto nos permite monitorear el progreso y la completitud de la extracción de datos.

Bloque de código 5

```
def list_to_str(x):
    if not x:
        return ""
    return ";".join(map(str, x))

def expected_from_set(s: dict) -> int:
    return s.get("total") or s.get("printedTotal") or 0
```

9. Lógica principal del crawler

La lógica principal de la extracción de datos se implementó en una función esencial que itera sobre los sets de cartas disponibles. Como se detalla en el **Bloque de código 6**, el proceso busca evitar duplicados y maneja la paginación de los resultados de la API. Se establece una secuencia de tamaños de página (pagesize) para optimizar la descarga, y se incorpora una lógica para detectar si ya se han obtenido todas las cartas de un set. Además, se utiliza un conjunto (seen) para garantizar la no duplicación de registros, lo que asegura que cada carta se procese una sola vez.

Bloque de código 6

```

104 def fetch_cards_of_set(set_id: str, start_page_size: int = INIT_PAGE_SIZE) -> List[Dict[str, Any]]:
105     cards: List[Dict[str, Any]] = []
106     seen: Set[str] = set()
107
108     # Secuencia de pageSize (sin duplicados):
109     sizes = []
110     for ps in [start_page_size, 250, 200, 150, 100, 50, 25]:
111         if ps not in sizes:
112             sizes.append(ps)
113
114     for page_size in sizes:
115         page = 1
116         empty_streak, MAX_EMPTY = 0, 3
117         while True:
118             params = {"q": f"set.id:{set_id}", "page": page, "pageSize": page_size}
119             r = safe_get(CARDS_URL, params=params, headers=HEADERS)
120             if r.status_code == 404:
121                 break # fin de páginas en este set
122             if r.status_code != 200:
123                 # cambiamos page_size; probamos siguiente tamaño
124                 break
125             batch = r.json().get("data", []) or []
126             if not batch:
127                 empty_streak += 1
128                 if empty_streak >= MAX_EMPTY:
129                     break
130             else:
131                 empty_streak = 0
132                 added = 0
133                 for item in batch:
134                     cid = item.get("id")
135                     if cid and cid not in seen:
136                         seen.add(cid)
137                         cards.append(item)
138                         added += 1
139                 page += 1
140             # si ya obtuvimos algo, pasamos al siguiente set;
141             # posibles huecos se cubrirán en otras pasadas
142             if cards:
143                 break
144         return cards
145

```

10. Función para Recolección Optimizada de Cartas por Conjunto

La función `crawl_all_by_set` como se detalla en el **Bloque de código 7** implementa un proceso iterativo para recolectar cartas únicas desde múltiples conjuntos, utilizando estructuras como conjuntos y diccionarios para evitar duplicados y llevar control del progreso. A través de varios ciclos controlados por un número máximo de pases (`max_passes`), descarga nuevas cartas solo si aún no se ha alcanzado la cantidad esperada, optimizando así el rendimiento. Este enfoque refleja buenas prácticas en scraping y manejo de datos, como el uso eficiente de estructuras y condiciones de parada, lo que permite una recolección robusta y escalable como se detalla en el **Bloque de código 7**.

Bloque de código 7

```

147 def crawl_all_by_set(max_passes: int = MAX_PASSES) -> List[Dict[str, Any]]:
148     sets = fetch_sets()
149     print(f"Sets detectados: {len(sets)}")
150
151     global_seen: Set[str] = set()
152     global_cards: Dict[str, Dict[str, Any]] = {}
153     expected = {s["id"]: expected_from_set(s) for s in sets}
154     collected = {s["id"]: 0 for s in sets}
155     improved = True
156     pass_num = 0
157     while improved and pass_num < max_passes:
158         pass_num += 1
159         improved = False
160         print(f"\n=== PASADA {pass_num}/{max_passes} ===")
161         for s in sets:
162             sid = s["id"]
163             exp = expected.get(sid, 0)
164             prev = collected.get(sid, 0)
165             # si ya cumplimos el esperado (si existe), saltamos
166             if exp and prev >= exp:
167                 continue
168
169             cards = fetch_cards_of_set(sid, start_pagesize=INIT_PAGESIZE)
170             added = 0
171             for c in cards:
172                 cid = c.get("id")
173                 if cid and cid not in global_seen:
174                     global_seen.add(cid)
175                     global_cards[cid] = c
176                     added += 1
177             collected[sid] = prev + added
178             if exp:
179                 print(f"{sid:8s} +{added:3d} (acum~{collected[sid]:4d} / esp={exp:4d})")
180             else:
181                 print(f"{sid:8s} +{added:3d} (acum~{collected[sid]:4d})")
182             if added > 0:
183                 improved = True
184     print(f"\nTOTAL CARTAS ÚNICAS DESCARGADAS: {len(global_cards)}")
185     return list(global_cards.values())

```

11. Transformación de Datos de Cartas a Formato Tabular Usando Python

La función `flatten_to_dataframe` como se detalla en el **Bloque de código 8** convierte una lista de diccionarios representando cartas en un DataFrame de pandas, organizando y aplanando información compleja como tipos, legalidades, atributos del set y precios de mercado. Utiliza estructuras de control y métodos de extracción seguros (`.get`) para manejar campos anidados o perdidos, evitando errores durante la conversión. Además, agrega múltiples precios desde la plataforma TCGPlayer por variante (bajo, medio, alto, etc.), lo que permite un análisis económico detallado. Esta función es esencial para estructurar datos semiestructurados en un formato analítico, lo anterior se usa también en procesos de minería de datos y visualización detalla en el **Bloque de código 8**.

Bloque de código 8

```
def flatten_to_dataframe(cards: List[Dict[str, Any]]) -> pd.DataFrame:
    rows = []
    for c in tqdm(cards, desc="Flatten", unit="card"):
        set_info = c.get("set", {}) or {}
        leg = c.get("legalities", {}) or {}
        set_leg = set_info.get("legalities", {}) or {}

        # Agregados de precios TCGPlayer por variantes
        tcgp = (c.get("tcgplayer") or {}).get("prices") or {}
        tcg_low, tcg_mid, tcg_high, tcg_market, tcg_directLow = [], [], [], [], []
        for _, p in tcgp.items():
            tcg_low.append(p.get("low"))
            tcg_mid.append(p.get("mid"))
            tcg_high.append(p.get("high"))
            tcg_market.append(p.get("market"))
            tcg_directLow.append(p.get("directLow"))

        mkp = (c.get("cardmarket") or {}).get("prices") or {}

        rows.append({
            "id": c.get("id"),
            "name": c.get("name"),
            "supertype": c.get("supertype"),
            "subtypes": list_to_str(c.get("subtypes")),
            "level": c.get("level"),
            "hp": c.get("hp"),
            "types": list_to_str(c.get("types")),
            "evolvesFrom": c.get("evolvesFrom"),
            "evolvesTo": list_to_str(c.get("evolvesTo")),
            "rules": list_to_str(c.get("rules")),
            "convertedRetreatCost": c.get("convertedRetreatCost"),
            "retreatCost_count": len(c.get("retreatCost") or []),
            "number": c.get("number"),
            "artist": c.get("artist"),
            "rarity": c.get("rarity"),
            "nationalPokedexNumbers": list_to_str(c.get("nationalPokedexNumbers")),
            # Legalidades
            "legalities.standard": leg.get("standard"),
            "legalities.expanded": leg.get("expanded"),
            # set.*
            "set.id": set_info.get("id"),
            "set.name": set_info.get("name"),
            "set.series": set_info.get("series"),
            "set.printedTotal": set_info.get("printedTotal"),
            "set.total": set_info.get("total"),
            "set.releaseDate": set_info.get("releaseDate"),
            "set.ptcgoCode": set_info.get("ptcgoCode"),
            "set.legalities.standard": set_leg.get("standard"),
            "set.legalities.expanded": set_leg.get("expanded"),
```


12. Normalización y Conversión Numérica de Datos Económicos de Cartas

En la parte final de la función `flatten_to_dataframe`, donde se agregan campos adicionales al `DataFrame`, especialmente relacionados con precios máximos desde `TCGPlayer` y valores de mercado desde `CardMarket` como se detalla en el **Bloque de código 9**. Se utilizan funciones `max()` para obtener los valores más altos entre múltiples variantes de precios, filtrando solo aquellos que son numéricos. Luego, se construye un `DataFrame` con todos los datos recopilados y se identifican columnas que deben convertirse a valores numéricos (por ejemplo, precios y costos). Esta conversión se realiza de forma segura usando `pd.to_numeric` con el parámetro `errors="coerce"`, lo que permite tratar errores sin interrumpir la ejecución. Este enfoque es fundamental para preparar datos para análisis estadístico, asegurando consistencia en los tipos de datos y facilitando operaciones posteriores como gráficos o modelado como detalla en el **Bloque de código 9**.

Bloque de código 9

```

"legalities.expanded": leg.get("expanded"),
# set.*
"set.id": set_info.get("id"),
"set.name": set_info.get("name"),
"set.series": set_info.get("series"),
"set.printedTotal": set_info.get("printedTotal"),
"set.total": set_info.get("total"),
"set.releaseDate": set_info.get("releaseDate"),
"set.ptcgoCode": set_info.get("ptcgoCode"),
"set.legalities.standard": set_leg.get("standard"),
"set.legalities.expanded": set_leg.get("expanded"),
# precios TCGPlayer (máximos por variantes)
"tcg_low_max": max([x for x in tcg_low if isinstance(x, (int, float))], default=None),
"tcg_mid_max": max([x for x in tcg_mid if isinstance(x, (int, float))], default=None),
"tcg_high_max": max([x for x in tcg_high if isinstance(x, (int, float))], default=None),
"tcg_market_max": max([x for x in tcg_market if isinstance(x, (int, float))], default=None),
"tcg_directLow_max": max([x for x in tcg_directLow if isinstance(x, (int, float))], default=None),
# precios CardMarket
"cm_trendPrice": mkp.get("trendPrice"),
"cm_averageSellPrice": mkp.get("averageSellPrice"),
"cm_lowPrice": mkp.get("lowPrice"),
"cm_germanProLow": mkp.get("germanProLow"),
"cm_suggestedPrice": mkp.get("suggestedPrice"),
"cm_reverseHoloSell": mkp.get("reverseHoloSell"),
"cm_reverseHoloLow": mkp.get("reverseHoloLow"),
"cm_reverseHoloTrend": mkp.get("reverseHoloTrend"),
"cm_avg1": mkp.get("avg1"),
"cm_avg7": mkp.get("avg7"),
"cm_avg30": mkp.get("avg30"),
"cm_reverseHoloAvg1": mkp.get("reverseHoloAvg1"),
"cm_reverseHoloAvg7": mkp.get("reverseHoloAvg7"),
"cm_reverseHoloAvg30": mkp.get("reverseHoloAvg30"),
})

df = pd.DataFrame(rows)
# Convertir a numérico donde aplica (sin romper)
numeric_cols = [
    "convertedRetreatCost",
    "tcg_low_max", "tcg_mid_max", "tcg_high_max", "tcg_market_max", "tcg_directLow_max",
    "cm_trendPrice", "cm_averageSellPrice", "cm_lowPrice", "cm_germanProLow",
    "cm_suggestedPrice", "cm_reverseHoloSell", "cm_reverseHoloLow", "cm_reverseHoloTrend",
    "cm_avg1", "cm_avg7", "cm_avg30", "cm_reverseHoloAvg1", "cm_reverseHoloAvg7", "cm_reverseHoloAvg30"
]
for col in numeric_cols:
    if col in df.columns:
        df[col] = pd.to_numeric(df[col], errors="coerce")
return df

```

13. Ejecución Principal para la Exportación de Cartas Pokémon a CSV

La función `main()` como se detalla en el **Bloque de código 10** es el punto de entrada del script, y coordina el flujo completo de ejecución: desde la descarga de cartas Pokémon hasta su exportación como archivo CSV. Primero, informa si no se dispone de una clave API, indicando que el proceso funcionará con limitaciones. Luego, llama a la función `crawl_all_by_set` para obtener todas las cartas disponibles, y a `flatten_to_dataframe` para convertirlas a un DataFrame de pandas estructurado. Finalmente, guarda este DataFrame como un archivo CSV en formato UTF-8, completando el proceso de transformación de datos desde una API hacia un formato tabular.

Bloque de código 10

```
def main():
    print("== Pokémon TCG → CSV por set ==")
    if not API_KEY:
        print("[AVISO] Sin API Key: funcionará, pero con límites más bajos.")
    print(f"Timeout={REQ_TIMEOUT}s | pageSize inicial={INIT_PAGESIZE} | max_passes={MAX_PASSES}")

    cards = crawl_all_by_set(max_passes=MAX_PASSES)
    print(f"Cartas únicas descargadas: {len(cards)}")

    df = flatten_to_dataframe(cards)
    print(f"DataFrame final: filas={len(df)}, cols={df.shape[1]}")
    df.fillna("").to_csv(OUT_CSV, index=False, encoding="utf-8")
    print(f"✓ Guardado CSV: {OUT_CSV}")

if __name__ == "__main__":
    main()
```

14. Conversión de Archivos Parquet a CSV con Pandas

La lógica que se detalla en el **Bloque de código 11** realiza una operación sencilla pero fundamental en flujos de datos: la conversión de archivos en formato Parquet a CSV usando la biblioteca pandas. Se leen dos archivos Parquet `ptcg_clean.parquet` y `ptcg_features.parquet` y se convierten directamente en archivos `.csv` mediante el método `.to_csv()`, asegurando que no se incluya el índice (`index=False`).

Bloque de código 11

```
#pasar parquet a csv
pd.read_parquet("ptcg_data/ptcg_clean.parquet").to_csv("ptcg_data/ptcg_clean.csv", index=False)
pd.read_parquet("ptcg_data/ptcg_features.parquet").to_csv("ptcg_data/ptcg_features.csv", index=False)
```

15. Ejecución de Preprocesamiento y Generación de Conjuntos de Datos para Modelado

El resultado de la ejecución completa de un script de preprocesamiento en Python como se detalla en la **Ilustración 2**, diseñado para preparar datos de cartas Pokémon antes de su uso en modelos de aprendizaje automático. Se cargan archivos CSV, se eliminan duplicados, se analizan nulos y se genera un resumen (ptcg_nulls_summary.csv). Posteriormente, se construyen conjuntos de entrenamiento y prueba a partir de los datos limpios, y se extraen características (features) y etiquetas (target). Estos conjuntos se guardan en formato .parquet, optimizado para lectura/escritura eficiente. Todo el proceso es informado paso a paso en consola, lo que refleja buenas prácticas de trazabilidad y control en la ingeniería de datos.

Ilustración 2

```
(.venv) PS C:\Users\alons\OneDrive\Documents\universidad\proyecto final matching> & "c:/Users/alons/OneDrive/Documents/universidad/proyecto final matching/.venv/Scripts/python.exe" "c:/Users/alons/OneDrive/Documents/universidad/proyecto final matching/01_preprocess_ptcg.py"
[LOAD] ptcg_data/ptcg_master_clean.csv -> shape=(19500, 54)
[DEDUP] eliminados=0 | actual=(19500, 54)
[INFO] Resumen de nulos guardado en: ptcg_data/ptcg_nulls_summary.csv
[SAVE] Limpio: ptcg_data/ptcg_clean.parquet | shape=(19500, 54)
[TARGET] Modo: regression | Columna y creada.
[SAVE] Features: ptcg_data/ptcg_features.parquet | shape=(19500, 151)
[SAVE] Train: ptcg_data/ptcg_train.parquet | (15600, 151)
[SAVE] Test : ptcg_data/ptcg_test.parquet | (3900, 151)

[Preprocesamiento completo.]
[LOAD] ptcg_data/ptcg_master_clean.csv -> shape=(19500, 54)
[DEDUP] eliminados=0 | actual=(19500, 54)
[INFO] Resumen de nulos guardado en: ptcg_data/ptcg_nulls_summary.csv
[SAVE] Limpio: ptcg_data/ptcg_clean.parquet | shape=(19500, 54)
[TARGET] Modo: regression | Columna y creada.
[SAVE] Features: ptcg_data/ptcg_features.parquet | shape=(19500, 151)
[SAVE] Train: ptcg_data/ptcg_train.parquet | (15600, 151)
[SAVE] Test : ptcg_data/ptcg_test.parquet | (3900, 151)

[Preprocesamiento completo.]
[INFO] Resumen de nulos guardado en: ptcg_data/ptcg_nulls_summary.csv
[SAVE] Limpio: ptcg_data/ptcg_clean.parquet | shape=(19500, 54)
[TARGET] Modo: regression | Columna y creada.
[SAVE] Features: ptcg_data/ptcg_features.parquet | shape=(19500, 151)
[SAVE] Train: ptcg_data/ptcg_train.parquet | (15600, 151)
[SAVE] Test : ptcg_data/ptcg_test.parquet | (3900, 151)

[Preprocesamiento completo.]
[TARGET] Modo: regression | Columna y creada.
[SAVE] Features: ptcg_data/ptcg_features.parquet | shape=(19500, 151)
[SAVE] Train: ptcg_data/ptcg_train.parquet | (15600, 151)
[SAVE] Test : ptcg_data/ptcg_test.parquet | (3900, 151)

[Preprocesamiento completo.]
[SAVE] Train: ptcg_data/ptcg_train.parquet | (15600, 151)
[SAVE] Test : ptcg_data/ptcg_test.parquet | (3900, 151)

[Preprocesamiento completo.]
[SAVE] Test : ptcg_data/ptcg_test.parquet | (3900, 151)

[Preprocesamiento completo.]
(.venv) PS C:\Users\alons\OneDrive\Documents\universidad\proyecto final matching> []
```

Referencias

Géron, A. (2022). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (3rd ed.). O'Reilly Media.

Pokémon TCG Developers. (2025). *Pokémon TCG API*. Recuperado de <https://dev.pokemontcg.io/>

Raschka, S. (2020). *Machine Learning with Python*. Packt Publishing.

Glosario de términos

Aprendizaje supervisado: Paradigma de *Machine Learning* donde el modelo aprende a partir de ejemplos con etiqueta conocida (X, y) para predecir en nuevos datos.

Clasificación binaria: Tarea predictiva con dos clases posibles (p. ej., carta ‘cara’ = 1 vs. ‘no cara’ = 0).

Variable objetivo (*target*): Columna que se busca predecir; en este trabajo, la etiqueta de ‘carta cara’ definida por el percentil 95 del precio de mercado.

Percentil (p95): Valor por encima del cual se ubica el 5% superior de la distribución de precios.

EDA (Análisis Exploratorio de Datos): Proceso inicial para comprender la estructura, la calidad y los patrones de los datos antes del modelado.

Desbalance de clases: Situación en la que una clase (p. ej., ‘cara’) es mucho menos frecuente que la otra, afectando el entrenamiento y las métricas del modelo.

ROC-AUC: Área bajo la curva ROC; mide la capacidad de un modelo para distinguir entre clases a diferentes umbrales.

PR-AUC: Área bajo la curva Precisión–*Recall*; útil cuando la clase positiva es rara.

F1-score: Media armónica de precisión y exhaustividad; balancea la importancia de falsos positivos y falsos negativos.

Outlier: Observación extrema que se desvía mucho del resto de los datos; puede sesgar medidas estadísticas y modelos predictivos.

Imputación de valores: Estrategias para completar valores faltantes (nulos) con reglas o estimaciones.

Ingeniería de características (*feature engineering*): Creación y transformación de variables para mejorar el rendimiento de un modelo (p. ej., número de ataques, daño promedio).

Rareza (*rarity*): Clasificación editorial que indica cuán común o escasa es una carta; se correlaciona con su valor potencial.

Jugabilidad (*playability*): Utilidad competitiva de una carta en el *metajuego*; influye en su demanda y precio.

Legalidades (*standard/expanded/unlimited*): Formato(s) en los que la carta es jugable; afectan su relevancia en torneos y en el mercado.

Precio de mercado (*TCGplayer Market*): Referencia de precio agregada por la plataforma TCGplayer; se usa como una aproximación al valor económico real.

Agregados de ataque/habilidad: Resúmenes numéricos derivados de los atributos de la carta para el modelado, como el conteo y promedios de daño.