



*\_MongoDB Personal Project:\_*

## ***PokeMongoDB***

# A Biological/Environmental Deep Dive into the World of Pokemon

## Table of Contents

- [PokeMongoDB](#)
  - [Introduction](#)
  - [MongoDB Setup](#)
- [Species Endpoint](#)
  - [PokeAPI Requests](#)
  - [Data Landing](#)
  - [PokeAPI Requests - Bulk](#)
  - [Data Transformation](#)
- [Pokemon Endpoint](#)
  - [Data Landing](#)
  - [PokeAPI Requests - Bulk](#)
- [PokeDex Collection](#)
  - [PokeDex Sample Output](#)
- [Class Diagram](#)
- [Visualizations & Aggregations](#)
- [Sources](#)

This project can also be found on github [here](#)

## Introduction

For the purposes of this project, I chose to use the PokeApi API endpoints to download data into a local MongoDB instance. With this data, the plan is to dive in and find different features about the world of pokemon, different features in different regions, and take a look at an overview of all the Pokemon monsters that have been released.

First, we need to set up the database to have it ready to accept the data to be requested from the API.

## MongoDB Setup

```
In [44]: import os, json, time, requests, datetime as dt
from dotenv import load_dotenv, find_dotenv
from pymongo import MongoClient, ASCENDING

# 1 - config
load_dotenv(find_dotenv(usecwd=True))
MONGO_URI = os.getenv("MONGO_URI")
DB_NAME = os.getenv("DB_NAME")

# 2 - connect to MongoDB and ping
client = MongoClient(MONGO_URI, serverSelectionTimeoutMS=5000)
client.admin.command('ping')

# 3 - db handle
db = client[DB_NAME]

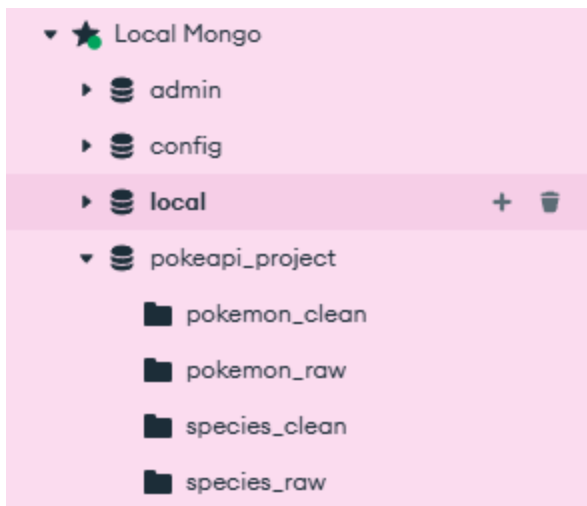
# 4 - create collections with indexes
species_raw = db["species_raw"]
species_clean = db["species_clean"]
pokemon_raw = db["pokemon_raw"]
pokemon_clean = db["pokemon_clean"]

#5 - create collections by creating minimal indexes
species_raw.create_index([("external_id", ASCENDING)], unique=True)
species_clean.create_index([("name", ASCENDING)], unique=True)
pokemon_raw.create_index([("external_id", ASCENDING)], unique=True)
pokemon_clean.create_index([("name", ASCENDING)], unique=True)

print("Connected to DB: ", DB_NAME)
print("Collections initialized: ", db.list_collection_names())
```

Connected to DB: pokeapi\_project

Collections initialized: ['pokemon\_clean', 'species\_clean', 'pokedex\_entries', 'pokemon\_raw', 'species\_raw']



As we can see now in `Mongo DB Compass`, the database has been created as well as the different collections we will use.

Before bulk-loading anything, we will fetch one species to understand the JSON shape and decide what fields we'll keep later in the clean collection. The endpoint we will target first for this is the `Pokemon Species` endpoint at `https://pokeapi.co/api/v2/pokemon-species/`. We'll use species number 413, called `wormadam`, as that is the example given in the API documentation, so that we may quickly recognize any error. Once we retrieve that information, we will print a human-readable summary of key attributes we are interested in.

## Species Endpoint

### PokeAPI Request (Species)

```
In [45]: import requests, json

BASE = "https://pokeapi.co/api/v2/"

def get_species (id_or_name):
    """Fetches a Pokemon species by ID or name from the PokeAPI."""
    r = requests.get(f"{BASE}pokemon-species/{id_or_name}", timeout=30)
    r.raise_for_status()
    return r.json()

def pick_en(entries, value_key):
    """Picks the English entry from a list of localized entries."""
    for entry in entries:
        if (entry.get("language") or {}).get("name") == "en":
            return entry.get(value_key)
    return None

# Fetch species with ID 413 (wormadam)
sp = get_species(413)
```

```

# summarize key fields we might normalize later
summary = {
    "id": sp.get("id"),
    "name": sp.get("name"),
    "order": sp.get("order"),
    "is_baby": sp.get("is_baby"),
    "is_legendary": sp.get("is_legendary"),
    "is_mythical": sp.get("is_mythical"),
    "gender_rate": sp.get("gender_rate"),
    "capture_rate": sp.get("capture_rate"),
    "base_happiness": sp.get("base_happiness"),
    "hatch_counter": sp.get("hatch_counter"),
    "forms_switchable": sp.get("forms_switchable"),
    "growth_rate": (sp.get("growth_rate") or {}).get("name"),
    "color": (sp.get("color") or {}).get("name"),
    "shape": (sp.get("shape") or {}).get("name"),
    "habitat": (sp.get("habitat") or {}).get("name"),
    "generation": (sp.get("generation") or {}).get("name"),
    "evolves_from_species": (sp.get("evolves_from_species") or {}).get("name"),
    "evolution_chain_id": (lambda url: int(url.rstrip('/').split('/')[-1]) if url else
        (sp.get("evolution_chain") or {}).get("url"))
    ),
    "egg_groups": [eg.get("name") for eg in (sp.get("egg_groups") or [])],
    "genus_en": pick_en(sp.get("genera"), "genus"),
    "flavor_text_en": (pick_en(sp.get("flavor_text_entries"), "flavor_text") or "")
        .replace("\n", " ").replace("\f", " ").strip(),
}

print(f"=== pokemon-species summary (id = {summary["id"]}) ===")
print(json.dumps(summary, indent=2))

print("\n === pokemon-species raw JSON preview ===")
print(json.dumps(sp, indent=2)[:2000] + "\n...")

```

=== pokemon-species summary (id = 413) ===

```
{
  "id": 413,
  "name": "wormadam",
  "order": 441,
  "is_baby": false,
  "is_legendary": false,
  "is_mythical": false,
  "gender_rate": 8,
  "capture_rate": 45,
  "base_happiness": 70,
  "hatch_counter": 15,
  "forms_switchable": false,
  "growth_rate": "medium",
  "color": "green",
  "shape": "blob",
  "habitat": null,
  "generation": "generation-iv",
  "evolves_from_species": "burmy",
  "evolution_chain_id": 213,
  "egg_groups": [
    "bug"
  ],
  "genus_en": "Bagworm Pok\u00e9mon",
  "flavor_text_en": "When BURMY evolved, its cloak became a part of this Pok\u00e9mon's body. The cloak is never shed."
}
```

=== pokemon-species raw JSON preview ===

```
{
  "base_happiness": 70,
  "capture_rate": 45,
  "color": {
    "name": "green",
    "url": "https://pokeapi.co/api/v2/pokemon-color/5/"
  },
  "egg_groups": [
    {
      "name": "bug",
      "url": "https://pokeapi.co/api/v2/egg-group/3/"
    }
  ],
  "evolution_chain": {
    "url": "https://pokeapi.co/api/v2/evolution-chain/213/"
  },
  "evolves_from_species": {
    "name": "burmy",
    "url": "https://pokeapi.co/api/v2/pokemon-species/412/"
  },
  "flavor_text_entries": [
    {
      "flavor_text": "When BURMY evolved, its cloak\nbecame a part of this Pok\u00e9mon's body. The cloak is never shed.",
      "language": {
        "name": "en",
        "url": "https://pokeapi.co/api/v2/language/9/"
      }
    }
  ]
}
```

```

    },
    "version": {
      "name": "diamond",
      "url": "https://pokeapi.co/api/v2/version/12/"
    }
  },
  {
    "flavor_text": "Its appearance changes depending\non where it evolved. The mat
erials\non hand become a part of its body.",
    "language": {
      "name": "en",
      "url": "https://pokeapi.co/api/v2/language/9/"
    },
    "version": {
      "name": "pearl",
      "url": "https://pokeapi.co/api/v2/version/13/"
    }
  },
  {
    "flavor_text": "When evolving, its body takes in\nsurrounding materials. As a
result,\nthere are many body variations.",
    "language": {
      "name": "en",
      "url": "https://pokeapi.co/api/v2/language/9/"
    },
    "version": {
      "name": "platinum",
      "url": "https://pokeapi.co/api/v2/version/14/"
    }
  },
  {
    "flavor_text": "It is said that a WORMADAM that\nevolves on a cold day will ha
ve a\nthicker cloak.",
    "language": {
      "name": "en",
      "url": "https://pokeapi.co/api/v2/language/9/"
    },
    "version": {
      "name": "heartgold",
      "url": "https://pokeapi.co/api/v2/version/15/"
    }
  },
  {
    "flavor_text": "It is said that a WORMADAM that\nevolves on a cold day will ha
ve a\nthicker cloak.",
    ...

```

Output looks good, so next we'll make sure that the raw JSON output lands into MongoDB as the first record of the staging collection.

## Data Landing (Species)

We'll create a raw landing collection where we store each API response as it is returned. This will let us re-run transformations without recalling the API, freeing up traffic for other users.

In the following cell we will:

- Add two "housekeeping" fields: `external_id` which will be the species' numeric id (413 in this case), and `_fetchedAt`, a timestamp.
- Upsert by `external_id` so reruns don't duplicate
- Confirm the insert by showing the collection count and a tiny projection of the stored doc

```
In [46]: import datetime as dt

assert "db" in globals(), "Connect to MongoDB first"
assert "sp" in globals(), "Run the previous cell to fetch species first"

species_raw = db["species_raw"]

doc = dict(sp) # make a copy
doc["external_id"] = int(sp["id"])
doc["_fetchedAt"] = dt.datetime.now()

# upsert by external_id
res = species_raw.update_one(
    {"external_id": doc["external_id"]},
    {"$set": doc},
    upsert=True
)

print("Upsert acknowledged:", res.acknowledged)
print("species_raw count:", species_raw.count_documents({}))

# show a projection to confirm
print(species_raw.find_one(
    {"external_id": doc["external_id"]},
    {"_id": 0, "external_id": 1, "name": 1, "growth_rate.name": 1}
))
```

```
Upsert acknowledged: True
species_raw count: 1025
{'external_id': 413, 'growth_rate': {'name': 'medium'}, 'name': 'wormadam'}
```

## PokeAPI Request (Species - Bulk)

Next up we want to bulk load all pokemon species into `species_raw` in order to populate our staging collection with every species document so we can transform locally without contacting the API again. Our approach will be to use the `paged-list` endpoint on `pokeapi.co` to discover all species, then follow each item's url to fetch the full document for each species. Since we're implementing an upsert safety measure, we'll be able to skip already present docs (wormadam) so we can resume or re-run this block safely. We will also allow a small sleep between requests to be nice to the public API.

```

In [47]: import time, requests, datetime as dt

assert "db" in globals(), "Connect to MongoDB first"

species_raw = db["species_raw"]
BASE = "https://pokeapi.co/api/v2/"

def list_species (limit=200, offset=0):
    """Lists Pokemon species with pagination."""
    r = requests.get(f"{BASE}pokemon-species", params={"limit": limit, "offset": of
    r.raise_for_status()
    return r.json()

# discover total species count and ids for resuming
meta = list_species(limit=1, offset=0)
total = meta["count"]
existing = {d["external_id"] for d in species_raw.find({}, {"external_id": 1})}

print(f"Total species supported by API: {total}")
print(f"Already iun species_raw: {len(existing)}")

inserted = 0
page = 100
for offset in range(0, total, page):
    page_data = list_species(limit=page, offset=offset)
    for item in page_data["results"]:
        url = item["url"]
        # fetch full species count
        doc = requests.get(url, timeout=30).json()
        sid = int(doc["id"])
        if sid in existing:
            continue
        doc["external_id"] = sid
        doc["_fetchedAt"] = dt.datetime.now()
        species_raw.update_one({"external_id": sid}, {"$set": doc}, upsert=True)
        inserted += 1
        time.sleep(0.05) # be nice to the public API
    print(f"Progress: {min(offset + page, total)}/{total} species, {inserted} new i

print("species_raw total documents:", species_raw.count_documents({}))

```

```

Total species supported by API: 1025
Already iun species_raw: 1025
Progress: 100/1025 species, 0 new inserted
Progress: 200/1025 species, 0 new inserted
Progress: 300/1025 species, 0 new inserted
Progress: 400/1025 species, 0 new inserted
Progress: 500/1025 species, 0 new inserted
Progress: 600/1025 species, 0 new inserted
Progress: 700/1025 species, 0 new inserted
Progress: 800/1025 species, 0 new inserted
Progress: 900/1025 species, 0 new inserted
Progress: 1000/1025 species, 0 new inserted
Progress: 1025/1025 species, 0 new inserted
species_raw total documents: 1025

```



All the existing species available on `pokeapi.co` have been stored locally in `species_raw`, which means we are ready to create a clean, analysis-friendly collection (`species_clean`) by extracting the fields we care about.

## Transformation (Species)

The next step, as previously stated, is transforming the data from `species_raw` to `species_clean`. We want flat, queryable documents, and we only want to keep the fields we mean to analyze, such as taxonomy, flags, rates, lineage, and only the selected English texts for each. We want one document per species, with simple scalar fields and small arrays, and we want them to be flat for easier querying and plotting, as well as migration later on or enriching the data with lookups from other endpoints. So next we will:

- Read all docs from `species_raw`
- Extract fields (`growth_rate.name`, `habitat.name`, `genus`, `flavor_text`)
- Write to `species_clean` with `_id = id` for stability
- Add helpful indexes for common analysis

```
In [48]: import datetime as dt

assert "db" in globals(), "Connect to MongoDB first"

species_raw = db["species_raw"]
species_clean = db["species_clean"]

def get_en(items, value_key):
    """Gets the English entry from a list of localized entries."""
    for it in items or []:
        if (it.get("language") or {}).get("name") == "en":
            return it.get(value_key)
    return None

def parse_chain_id(url):
    """Extracts the evolution chain ID from a URL. Some Pokemon don't have an evolu
    if not url:
        return None
    try:
        return int(url.rstrip('/').split('/')[-1])
    except:
        return None

species_clean.drop()

bulk = []
cursor = species_raw.find({}, {"_id":0})
for s in cursor:
    doc = {
        "_id": int(s["id"]),
        "id": int(s["id"]),
```

```

"name": s.get("name"),
"genus_en": get_en(s.get("genera"), "genus"),
"flavor_text_en": (get_en(s.get("flavor_text_entries"), "flavor_text") or "
                    .replace("\n", " ").replace("\f", " ").strip(),

# flags
"is_baby": bool(s.get("is_baby")),
"isLegendary": bool(s.get("is_legendary")),
"is_mythical": bool(s.get("is_mythical")),
"is_default": bool(s.get("is_default")),

# numeric attributes
"gender_rate": s.get("gender_rate"),
"capture_rate": s.get("capture_rate"),
"base_happiness": s.get("base_happiness"),
"hatch_counter": s.get("hatch_counter"),
"forms_switchable": bool(s.get("forms_switchable")),

# taxonomy (flattened as strings)
"growth_rate": (s.get("growth_rate") or {}).get("name"),
"color": (s.get("color") or {}).get("name"),
"habitat": (s.get("habitat") or {}).get("name"),
"shape": (s.get("shape") or {}).get("name"),
"generation": (s.get("generation") or {}).get("name"),

# lineage
"evolves_from_species": (s.get("evolves_from_species") or {}).get("name"),
"evolution_chain_id": parse_chain_id((s.get("evolution_chain") or {}).get("

# small arrays
"egg_groups": [eg.get("name") for eg in (s.get("egg_groups") or [])],

    "_updatedAt": dt.datetime.now(),
}
bulk.append(doc)

if bulk:
    species_clean.insert_many(bulk, ordered=False)

# helpful indexes
species_clean.create_index([("name", 1)], unique=True)
species_clean.create_index([("habitat", 1)])
species_clean.create_index([("generation", 1)])
species_clean.create_index([("color", 1)])
species_clean.create_index([("growth_rate", 1)])

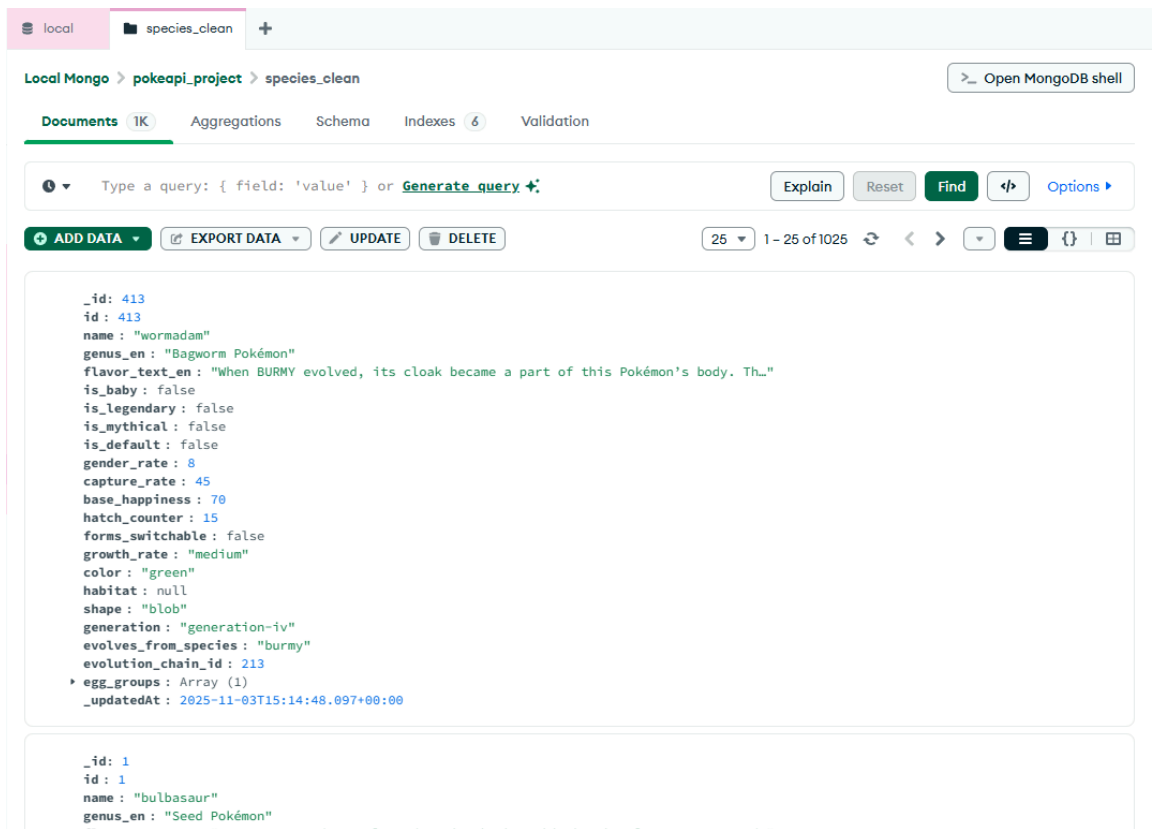
print("species_clean count: ", species_clean.count_documents({}))
print("example:\n", species_clean.find_one({}, {"_id":0, "id":1, "name":1, "habitat

```

species\_clean count: 1025

example:

```
{'id': 413, 'name': 'wormadam', 'flavor_text_en': 'When BURMY evolved, its cloak be
came a part of this Pokémon's body. The cloak is never shed.', 'habitat': None, 'gen
eration': 'generation-iv', 'egg_groups': ['bug']}
```



As we can see above from Compass, `species_clean` has been populated.

We are now ready to gather data from a second endpoint, `pokemon`, to enrich our study. This endpoint will give us height, weight, base\_experience, types, abilities, and sprite data. We'll stage a raw JSON into `pokemon_raw`, just we did before, before building a clean `pokemon_clean` collection and link it to `species_clean`.

We will be following the exact same process as before for this endpoint.

## Pokemon Endpoint

### Data Landing (Pokemon)

```
In [49]: import datetime as dt

pokemon_raw = db["pokemon_raw"]

def list_pokemon (limit=200, offset=0):
    """Return a page of pokemon metadata from /pokemon endpoint."""
    return requests.get(f"{BASE}pokemon", params={"limit": limit, "offset": offset})

meta = list_pokemon(limit=1)
total = meta["count"]
existing = {d["external_id"] for d in pokemon_raw.find({}, {"external_id": 1})}
```

```

print(f"Total pokemon supported by API: {total}")
print(f"Already in pokemon_raw: {len(existing)}")

inserted = 0
page = 100
for offset in range(0, total, page):
    page_data = list_pokemon(limit=page, offset=offset)
    for item in page_data["results"]:
        detail = requests.get(item["url"], timeout=30).json()
        pid = int(detail["id"])
        if pid in existing:
            continue
        detail["external_id"] = pid
        detail["_fetchedAt"] = dt.datetime.now()
        pokemon_raw.update_one({"external_id": pid}, {"$set": detail}, upsert=True)
        inserted += 1
        time.sleep(0.05)
    print(f"Progress: {min(offset + page, total)}/{total} pokemon, {inserted} new i

print("pokemon_raw total documents:", pokemon_raw.count_documents({}))

```

```

Total pokemon supported by API: 1328
Already in pokemon_raw: 1328
Progress: 100/1328 pokemon, 0 new inserted
Progress: 200/1328 pokemon, 0 new inserted
Progress: 300/1328 pokemon, 0 new inserted
Progress: 400/1328 pokemon, 0 new inserted
Progress: 500/1328 pokemon, 0 new inserted
Progress: 600/1328 pokemon, 0 new inserted
Progress: 700/1328 pokemon, 0 new inserted
Progress: 800/1328 pokemon, 0 new inserted
Progress: 900/1328 pokemon, 0 new inserted
Progress: 1000/1328 pokemon, 0 new inserted
Progress: 1100/1328 pokemon, 0 new inserted
Progress: 1200/1328 pokemon, 0 new inserted
Progress: 1300/1328 pokemon, 0 new inserted
Progress: 1328/1328 pokemon, 0 new inserted
pokemon_raw total documents: 1328

```

After downloading the data, once again we aim to transform the raw output and store it, migrating from `pokemon_raw` to `pokemon_clean`. Each pokemon will contain the following:

- Basic info: `id`, `name`, `height`, `weight`, `base_experience`
- Categorical info: `primary_type`, list of all `types`
- Stats (stored as key-value map): `{"speed": 35, "attack": 45, ...}`
- Sprite URLs (front and official artwork)
- Species name (used to link with `species_clean`) This flattened form is what we'll use for joins, aggregations, and visualizations.

## PokeAPI Request (Pokemon - Bulk)

```

In [50]: from pymongo import ASCENDING
import datetime as dt

pokemon_clean = db["pokemon_clean"]
pokemon_raw = db["pokemon_raw"]

pokemon_clean.drop()

bulk = []

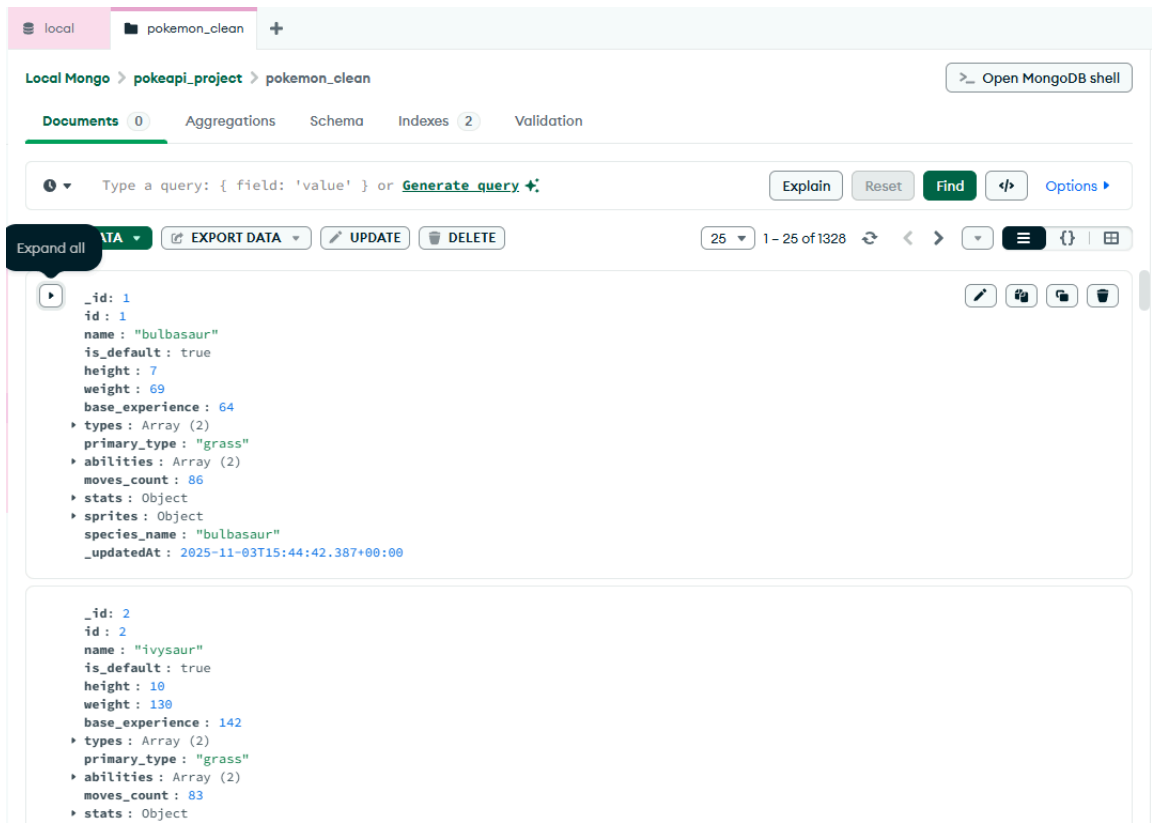
for p in pokemon_raw.find({}, {"_id":0}):
    types = [t["type"]["name"] for t in (p.get("types") or [])]
    stats_map = { s["stat"]["name"]: s["base_stat"] for s in (p.get("stats") or [])}
    sprites = {
        "front_default": p.get("sprites", {}).get("front_default"),
        "official_artwork": p.get("sprites", {}).get("other", {}).get("official-art
    }
    out = {
        "_id": int(p["id"]),
        "id": int(p["id"]),
        "name": p.get("name"),
        "is_default": bool(p.get("is_default")),
        "height": p.get("height"),
        "weight": p.get("weight"),
        "base_experience": p.get("base_experience"),
        "types": types,
        "primary_type": types[0] if types else None,
        "abilities": [a["ability"]["name"] for a in (p.get("abilities") or [])],
        "moves_count": len(p.get("moves") or []),
        "stats": stats_map,
        "sprites": sprites,
        "species_name": (p.get("species") or {}).get("name"),
        "_updatedAt": dt.datetime.now(),
    }
    bulk.append(out)

if bulk:
    pokemon_clean.insert_many(bulk, ordered=False)
    pokemon_clean.create_index([("name", ASCENDING)], unique=True)
    pokemon_clean.create_index([("primary_type", ASCENDING)])
    pokemon_clean.create_index([("species_name", ASCENDING)])

print("pokemon_clean count: ", pokemon_clean.count_documents({}))
print("example:\n", pokemon_clean.find_one({}, {"_id":0, "id":1, "name":1, "primary

pokemon_clean count: 1328
example:
{'id': 1, 'name': 'bulbasaur', 'height': 7, 'weight': 69, 'primary_type': 'grass'}

```



pokemon\_clean is now populated as well, which concludes the data collection part of this project.

We now have four collections, two of which are raw outputs and can still be used to transform into other collections, and two which we have cleaned for analysis. Next up, we will run a MongoDB aggregation that joins Pokemon with their corresponding species info (using `$lookup`) in order to explore trends, like average weight by habitat or generation.

Before moving on to visualization, we will create one more collection within MongoDB which will try to simulate the data stored in a Pokédex. After creating this collection, and viewing our Class Diagram, we will enter into more visual exploration of the data.

## Pokédex Collection

```
In [51]: # create/refresh a merged "pokedex_entries" collection
from pymongo import ASCENDING
import datetime as dt

assert 'db' in globals(), "Run the MongoDB setup cell first."

species_clean = db["species_clean"]
pokemon_clean = db["pokemon_clean"]
pokedex_entries = db["pokedex_entries"]

# Start fresh (safe to re-run)
pokedex_entries.drop()
```

```

best_by_species = {} # dex_number -> {"entry": doc, "prio": (score...)}

def pick_priority(p_doc):
    """
    Higher tuple wins.
    Preference order:
    1) is_default True
    2) name without hyphen (usually base form)
    3) lower numeric id (base forms < 10000; many alts are 10000+)
    """
    name = (p_doc.get("name") or "")
    pid = int(p_doc.get("id") or 999999)
    is_default = bool(p_doc.get("is_default", False))
    no_hyphen = 1 if "-" not in name else 0
    base_id = 1 if pid < 10000 else 0
    return (1 if is_default else 0, no_hyphen, base_id)

bulk_docs = 0
for p in pokemon_clean.find({}, {"_id":0}):
    sp = species_clean.find_one({"name": p.get("species_name")}, {"_id":0})
    if not sp:
        # fallback by id equality if needed
        sp = species_clean.find_one({"id": p.get("id")}, {"_id":0})
    if not sp:
        continue

    dex_number = int(sp["id"]) # National Dex #
    types = p.get("types") or []
    stats_map = p.get("stats") or {}
    sprites = p.get("sprites") or {}

    entry = {
        "_id": dex_number, # one doc per species
        "dex_number": dex_number,
        "name": sp.get("name"),
        "genus": sp.get("genus_en"),
        "flavor_text": sp.get("flavor_text_en"),
        "primary_type": types[0] if types else None,
        "types": types,
        "height_m": (p.get("height") or 0) / 10.0,
        "weight_kg": (p.get("weight") or 0) / 10.0,
        "abilities": p.get("abilities") or [],
        "stats": stats_map,
        "sprite_url": sprites.get("front_default"),
        "artwork_url": sprites.get("official_artwork"),
        # species context:
        "habitat": sp.get("habitat"),
        "egg_groups": sp.get("egg_groups") or [],
        "_updatedAt": dt.datetime.now(),
    }

    prio = pick_priority(p)
    slot = best_by_species.get(dex_number)
    if not slot or prio > slot["prio"]:
        best_by_species[dex_number] = {"entry": entry, "prio": prio}

```

```

# Insert ONE document per species
docs = [v["entry"] for v in best_by_species.values()]
if docs:
    pokedex_entries.insert_many(docs, ordered=False)

# Helpful indexes
pokedex_entries.create_index([("dex_number", ASCENDING)], unique=True)
pokedex_entries.create_index([("name", ASCENDING)], unique=True)

print("pokedex_entries inserted:", len(docs))
print("example:", pokedex_entries.find_one({}, {"_id":0, "dex_number":1, "name":1,

```

```

pokedex_entries inserted: 1025
example: {'dex_number': 1, 'name': 'bulbasaur', 'primary_type': 'grass'}

```

We now have a full-fledged PokéDex within MongoDB, complete with links to sprites and artwork, and displaying all the information that is usually displayed in the PokéDex in the show. Below is an example of what we can see when visualizing a PokéDex entry for a Pokemon using only the data we've stored in MongoDB.

## PokeDex Sample Output

```

In [52]: # Pokédex entry renderer
from IPython.display import HTML, display
import html

def render_pokedex(name_or_id):
    """Renders a Pokédex entry card for the given Pokemon name or dex number.
    Args:
        name_or_id (str|int): The name or dex number of the Pokemon to render
    """
    col = db["pokedex_entries"]

    if isinstance(name_or_id, int) or (isinstance(name_or_id, str) and name_or_id.isdigit()):
        doc = col.find_one({"dex_number": int(name_or_id)})
    else:
        doc = col.find_one({"name": str(name_or_id).lower()})

    if not doc:
        display(HTML(f"<b style='color:white;'>No entry found for:</b> {html.escape(name_or_id)}"))
        return

    dex = doc.get("dex_number")
    name = doc.get("name", "").title()
    genus = doc.get("genus", "")
    flavor = (doc.get("flavor_text") or "").replace("\n", " ").replace("\f", " ").strip()
    types = doc.get("types") or []
    height_m = doc.get("height_m", 0)
    weight_kg = doc.get("weight_kg", 0)
    sprite = doc.get("sprite_url")

```



```

art = doc.get("artwork_url")

types_html = " ".join(f"<span class='type-badge'>{t.title()}</span>" for t in t
h = f"{height_m:.1f} m"
w = f"{weight_kg:.1f} kg"

img_html = ""
if art:
    img_html += f''
if sprite:
    img_html += f'No image available</div>'

card = f"""
<style>
.dex-card {{
    font-family: 'Segoe UI', Roboto, Helvetica, Arial, sans-serif;
    background: linear-gradient(180deg, #d8eefe 0%, #b8e0ff 100%);
    color: #0f172a;
    border-radius: 12px;
    padding: 20px;
    max-width: 860px;
    border: 1px solid #60a5fa;
    box-shadow: 0 4px 18px rgba(96,165,250,0.3);
}}
.dex-header {{
    display:flex; justify-content:space-between; align-items:center;
    margin-bottom:10px;
}}
.dex-title {{ font-size:1.6rem; font-weight:700; }}
.dex-number {{ color:#1e3a8a; font-weight:600; }}
.row {{ display:flex; gap:16px; flex-wrap:wrap; margin-bottom:6px; }}
.label {{ font-weight:600; color:#1e3a8a; min-width:80px; }}
.type-badge {{
    display:inline-block; padding:3px 8px; border-radius:999px;
    background:#1e40af; color:white; font-size:0.85rem;
    margin-right:6px;
}}
.media {{
    display:flex; align-items:flex-end; gap:16px; margin:12px 0;
}}
.media img {{
    border-radius:10px; border:1px solid #60a5fa;
    background:#f0f9ff;
}}
.media .art {{ max-width:240px; }}
.media .sprite {{ max-width:120px; image-rendering:pixelated; }}
.flavor {{
    background:#e0f2fe; border:1px solid #93c5fd;
    border-radius:8px; padding:10px; color:#0c4a6e;
    line-height:1.4;
}}
.meta {{ margin-top:8px; font-size:0.85rem; color:#1e3a8a; }}
</style>

```

```
<div class="dex-card">
  <div class="dex-header">
    <div class="dex-title">{html.escape(name)}</div>
    <div class="dex-number">No. {dex:03d}</div>
  </div>

  <div class="row"><div class="label">Species</div><div>{html.escape(genus)}</d
  <div class="row"><div class="label">Type</div><div>{types_html or "-"}</div><
  <div class="row"><div class="label">Height</div><div>{h}</div>
    <div class="label">Weight</div><div>{w}</div></div>

  <div class="media">{img_html}</div>

  <div class="flavor">{html.escape(flavor or "No Pokédex description available.
  <div class="meta">Last updated: {html.escape(str(doc.get("_updatedAt") or ""))
</div>
  ""
  display(HTML(card))
```

# Examples

```
render_pokedex("pikachu")
```

```
render_pokedex(6)
```

# Pikachu

No. 025

Species Mouse Pokémon

Type Electric

Height 0.4 m Weight 6.0 kg



When several of these POKÉMON gather, their electricity could build and cause lightning storms.

Last updated: 2025-11-04 17:14:04.768000

# Charizard

No. 006

Species Flame Pokémon

Type **Fire** **Flying**

Height 1.7 m Weight 90.5 kg

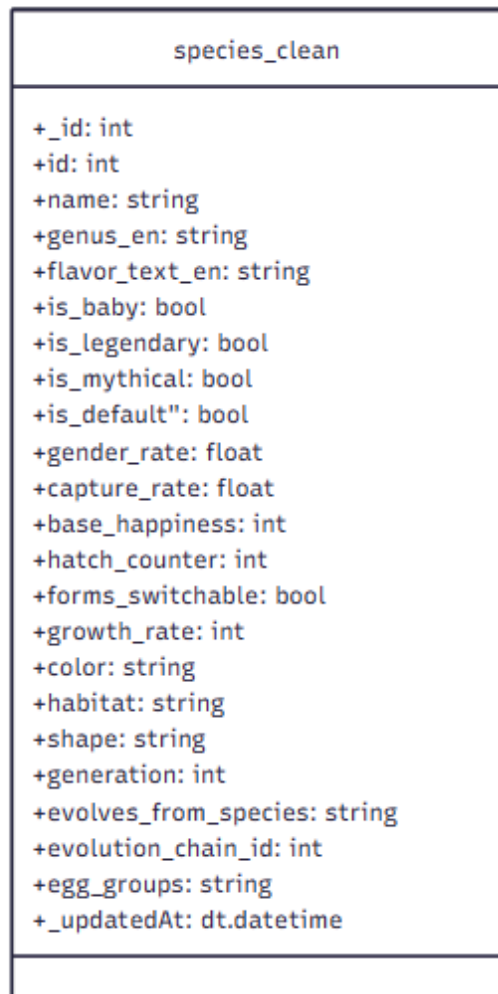
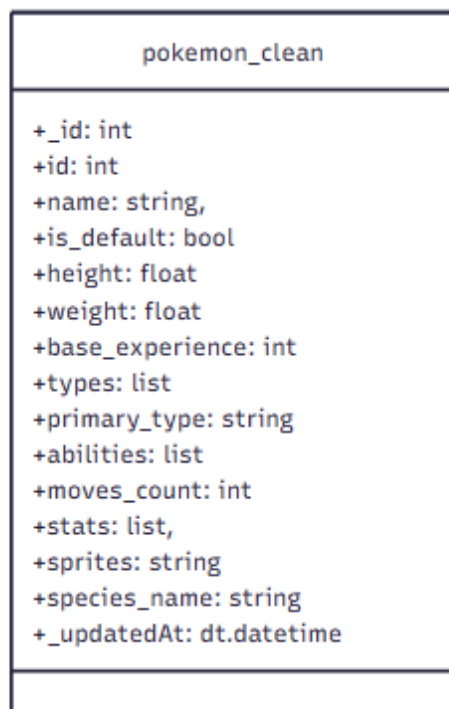


Spits fire that is hot enough to melt boulders. Known to cause forest fires unintentionally.

Last updated: 2025-11-04 17:14:04.763000

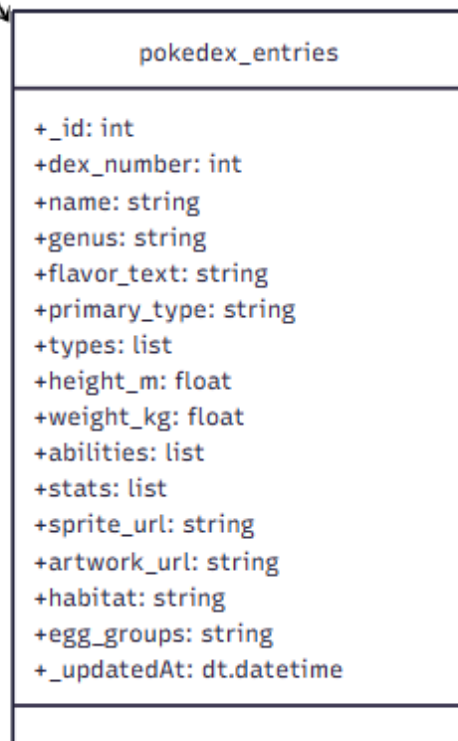
## Class Diagram

Below, we will use the [mermaidchart.com](https://mermaidchart.com) to visualize a Class Diagram of our database and see how the objects have been stored in MongoDB.



name → name

name ↔ name



# Visualizations & Aggregations

Below we will go through some aggregations like average weight by habitat, habitat distributions by legendary/mythical/regular status, type co-occurrence, average base stats by primary type, evolution chain sizes, and egg group x habitat cross tabulation.

```
In [53]: # Average Weight by Habitat
import pandas as pd

pipeline = [
    {"$lookup": {
        "from": "species_clean",
        "localField": "species_name",
        "foreignField": "name",
        "as": "species_info"
    }},
    {"$unwind": "$species_info"},
    {"$group": {
        "_id": "$species_info.habitat",
        "avg_weight": {"$avg": "$weight"},
        "avg_height": {"$avg": "$height"},
        "count": {"$sum": 1}
    }},
    {"$sort": {"avg_weight": -1}}
]

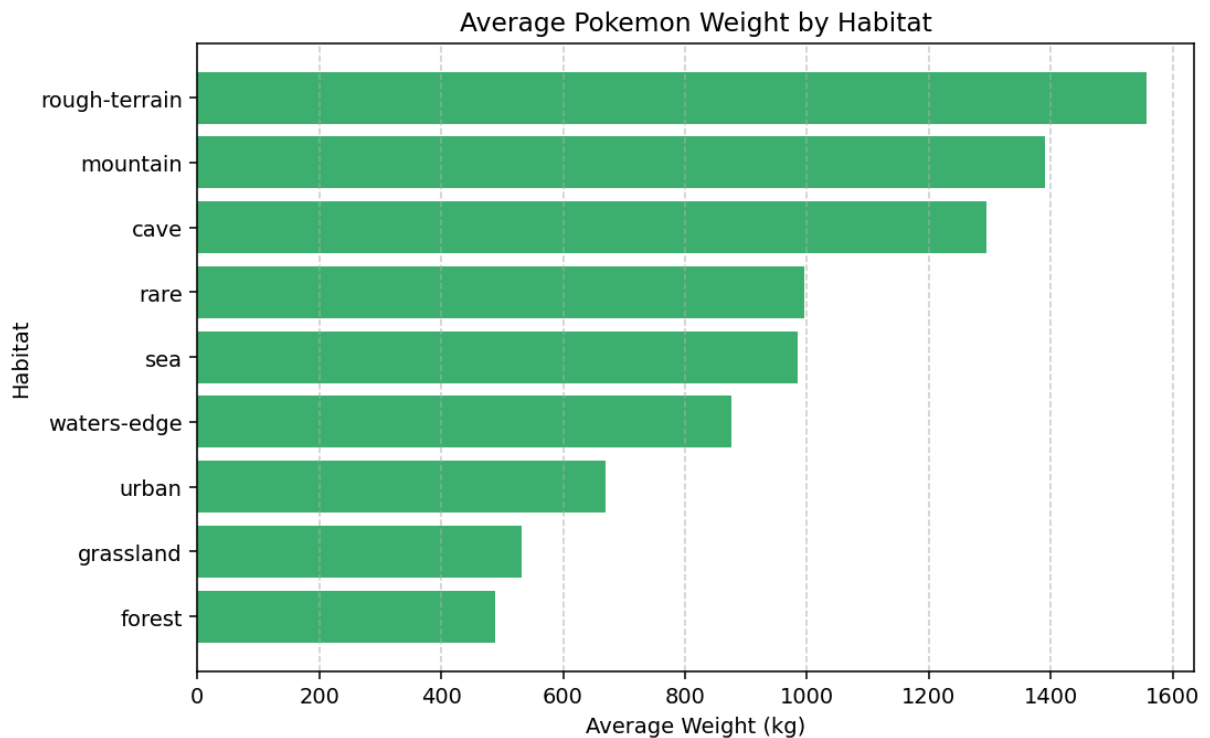
result = list(db["pokemon_clean"].aggregate(pipeline))
df = pd.DataFrame(result).rename(columns={"_id": "habitat"})
print(df.head())

import matplotlib.pyplot as plt

plot_df = df.dropna(subset=["habitat"]).sort_values("avg_weight", ascending=True)

plt.figure(figsize=(8,5))
plt.barh(plot_df["habitat"], plot_df["avg_weight"], color="mediumseagreen")
plt.title("Average Pokemon Weight by Habitat")
plt.xlabel("Average Weight (kg)")
plt.ylabel("Habitat")
plt.grid(axis="x", linestyle="--", alpha=0.6)
plt.tight_layout()
plt.show()
```

	habitat	avg_weight	avg_height	count
0	rough-terrain	1557.588235	13.235294	34
1	mountain	1391.274194	25.483871	62
2	cave	1294.729730	22.837838	37
3	None	1065.825123	21.831281	812
4	rare	995.157895	26.789474	19



With this first glance we have a little bit of insight. It seems rocky terrain like `rough-terrain` and `mountain` habitats tend to create heavier pokemon. Let's visualize this further for a clearer picture. We can see the average weight by habitat of all pokemon available on the API. We can see `rough-terrain`, `mountain`, and `cave` are the only habitats with an average weight of pokemon of over 1000kg, which is surprising to me personally, since whenever I think of cave pokemon, I usually think of `Zubat`, which is not a very heavy pokemon.

```
In [54]: # imports
from pymongo import MongoClient
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

client = MongoClient("mongodb://localhost:27017")
db = client["pokeapi_project"]

plt.rcParams["figure.dpi"] = 140

# Habitat distribution by status
# A1 - Species by Habitat and Status (robust version)

pipe = [
    {"$project": {
        "habitat": {"$ifNull": ["$habitat", "unknown"]},
        "isLegend": {"$ifNull": ["$is_legendary", False]},
        "isMyth": {"$ifNull": ["$is_mythical", False]}
    }},

```

```

    {"$group": {
      "_id": "$habitat",
      "regular": {"$sum": {"$cond": [{"$and": [{"$eq": ["$isLegend", False]}, {"
      "legendary": {"$sum": {"$cond": [{"$eq": ["$isLegend", True]}, 1, 0]}]},
      "mythical": {"$sum": {"$cond": [{"$eq": ["$isMyth", True]}, 1, 0]}]}
    }},
    {"$sort": {"_id": 1}}
  ]
}

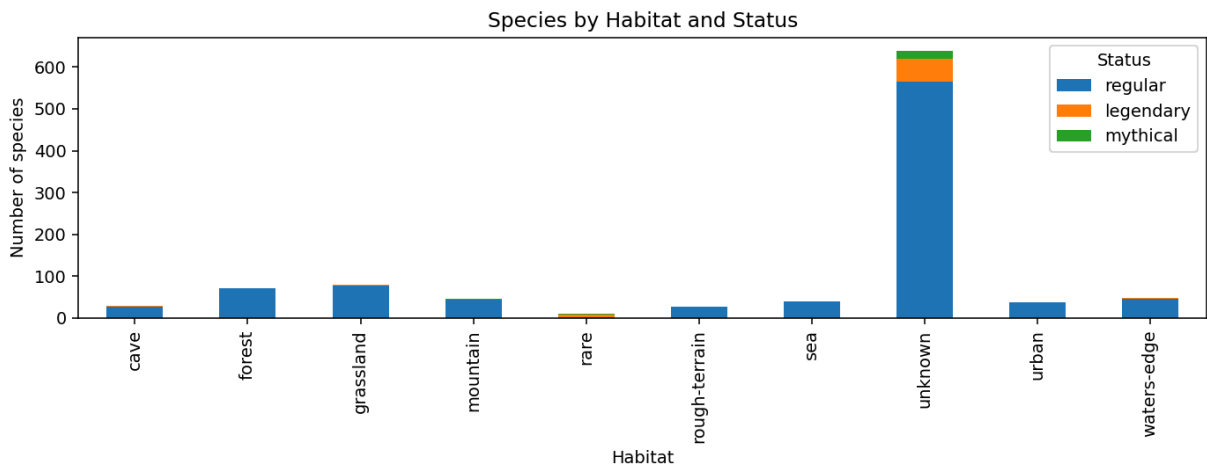
rows = list(db.species_clean.aggregate(pipe))

import pandas as pd, matplotlib.pyplot as plt
df = pd.DataFrame(rows).rename(columns={"_id": "habitat"})
df = df.sort_values("habitat")

ax = df.plot(kind="bar", x="habitat", y=["regular", "legendary", "mythical"], stacked)
ax.set_xlabel("Habitat")
ax.set_ylabel("Number of species")
ax.set_title("Species by Habitat and Status")
ax.legend(title="Status", loc="upper right")
plt.tight_layout()
plt.show()

print(df) # quick sanity table

```



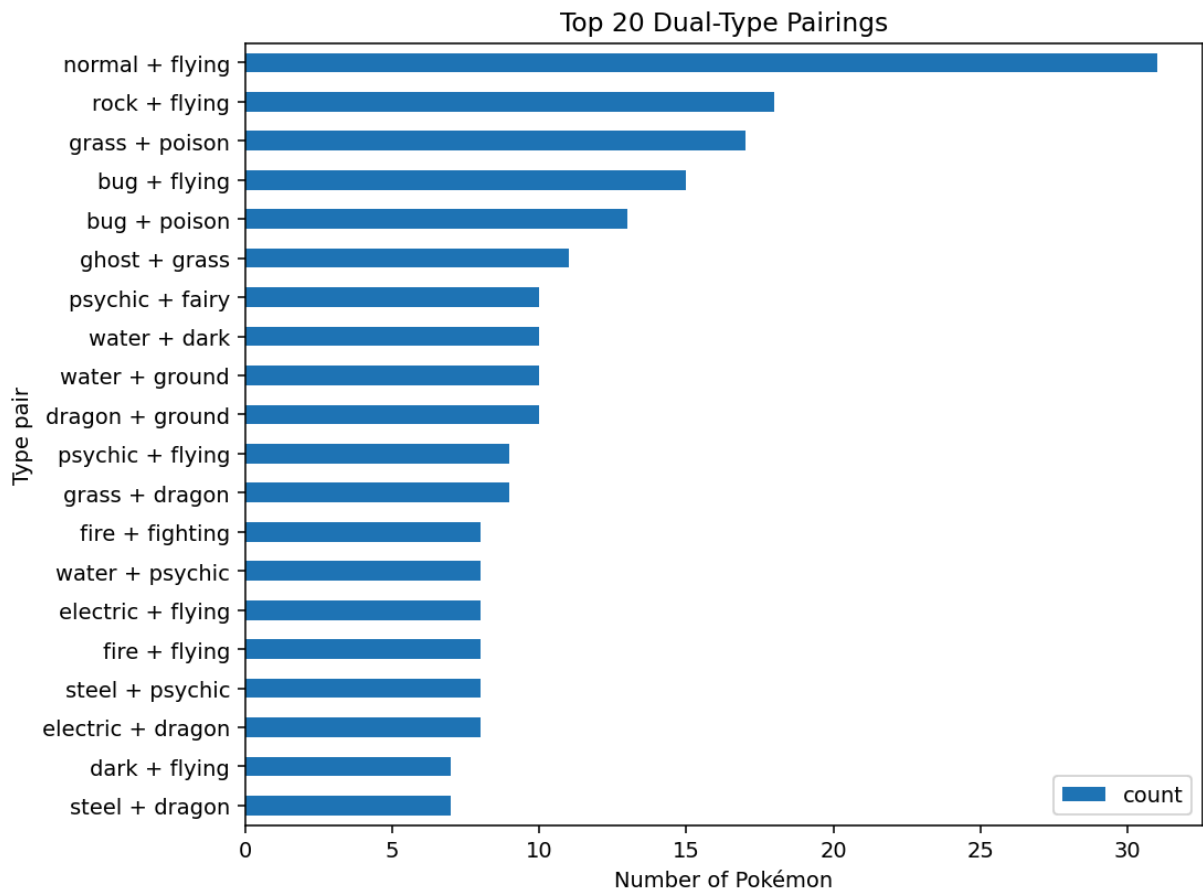
	habitat	regular	legendary	mythical
0	cave	26	3	0
1	forest	70	0	1
2	grassland	77	3	0
3	mountain	44	0	1
4	rare	1	7	2
5	rough-terrain	26	1	0
6	sea	39	1	0
7	unknown	566	54	19
8	urban	37	0	0
9	waters-edge	45	2	0

It seems most Pokemon's habitat is unknown. This could be due to missing data in the API, which is probable, as the new game just came out, and with it came about changes to different Pokemon. However looking at the `legendary` and `mythical` pokemon within the `unknown` habitat makes

sense, since there have been pokemon introduced who have been stated to come from outside the planet where Pokemon takes place.

```
In [55]: # Type Co-Occurrence (top 20)
pipe = [
    {"$match": {"types": {"$exists": True, "$ne": []}}},
    {"$project": {"types": 1, "types2": "$types"}},
    {"$unwind": {"path": "$types", "includeArrayIndex": "i"}},
    {"$unwind": {"path": "$types2", "includeArrayIndex": "j"}},
    {"$match": {"$expr": {"$lt": [{"i}, {"j}]}},
    {"$group": {"_id": {"a": "$types", "b": "$types2"}, "count": {"$sum": 1}}},
    {"$sort": {"count": -1}},
    {"$limit": 20}
]
pairs = list(db.pokemon_clean.aggregate(pipe))
df = pd.DataFrame(pairs)
df["pair"] = df["_id"].apply(lambda d: f"{d['a']} + {d['b']}")
df = df[["pair", "count"]].sort_values("count")

ax = df.plot(kind="barh", x="pair", y="count", figsize=(8,6))
ax.set_xlabel("Number of Pokémon")
ax.set_ylabel("Type pair")
ax.set_title("Top 20 Dual-Type Pairings")
plt.tight_layout()
plt.show()
```



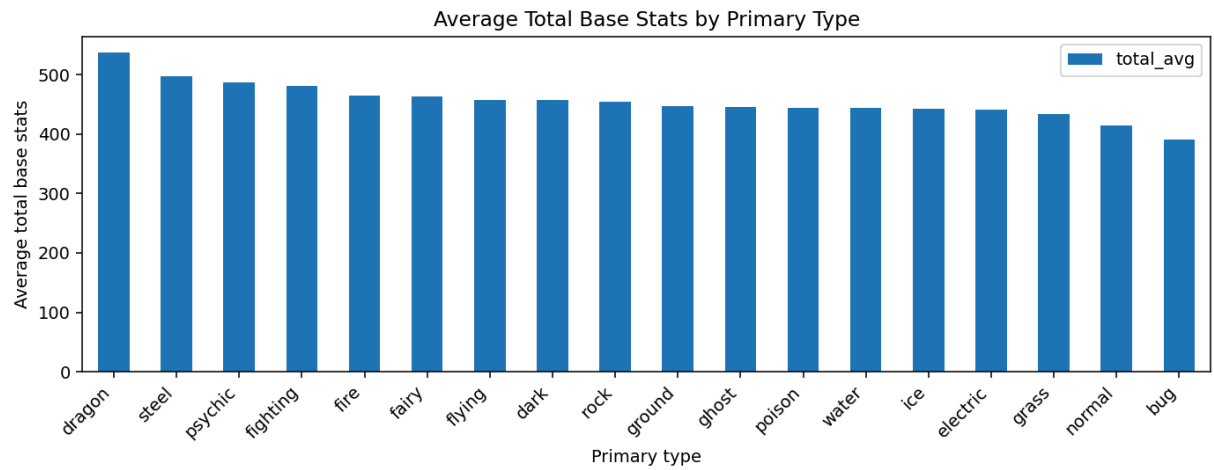


In the bar chart above, we can see the most common pairing of types is `normal + flying` by a large margin, followed by `rock+flying`. Bug type pokemon also seem to be common in the top pairings, which to the average Pokemon enjoyer, may be unsurprising. Personally, having played many of the pokemon games, the top 5 pairings seem rather familiar, since these are usually the types we encounter in the beginning areas of the games. Some notable species that come to mind are Pidgey, Wurmple, and Venonat, to name a few.

```
In [56]: # Average Base Stats by Primary Type
pipe = [
    {"$project": {
        "primary_type": {"$arrayElemAt": ["$types", 0]},
        "hp": "$stats.hp",
        "atk": "$stats.attack",
        "def": "$stats.defense",
        "spa": "$stats.special-attack",
        "spd": "$stats.special-defense",
        "spe": "$stats.speed"
    }},
    {"$group": {
        "_id": "$primary_type",
        "n": {"$sum": 1},
        "hp": {"$avg": "$hp"},
        "atk": {"$avg": "$atk"},
        "def": {"$avg": "$def"},
        "spa": {"$avg": "$spa"},
        "spd": {"$avg": "$spd"},
        "spe": {"$avg": "$spe"}
    }},
    {"$sort": {"_id": 1}}
]
avg_stats = list(db.pokemon_clean.aggregate(pipe))
df = pd.DataFrame(avg_stats).rename(columns={"_id": "type"})
df["total_avg"] = df[["hp", "atk", "def", "spa", "spd", "spe"]].sum(axis=1)
df = df.sort_values("total_avg", ascending=False)

ax = df.plot(kind="bar", x="type", y="total_avg", figsize=(10,4))
ax.set_xlabel("Primary type")
ax.set_ylabel("Average total base stats")
ax.set_title("Average Total Base Stats by Primary Type")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()

# (Optional extra view: table of the six stat means)
display(df[["type", "n", "hp", "atk", "def", "spa", "spd", "spe"]].round(1))
```

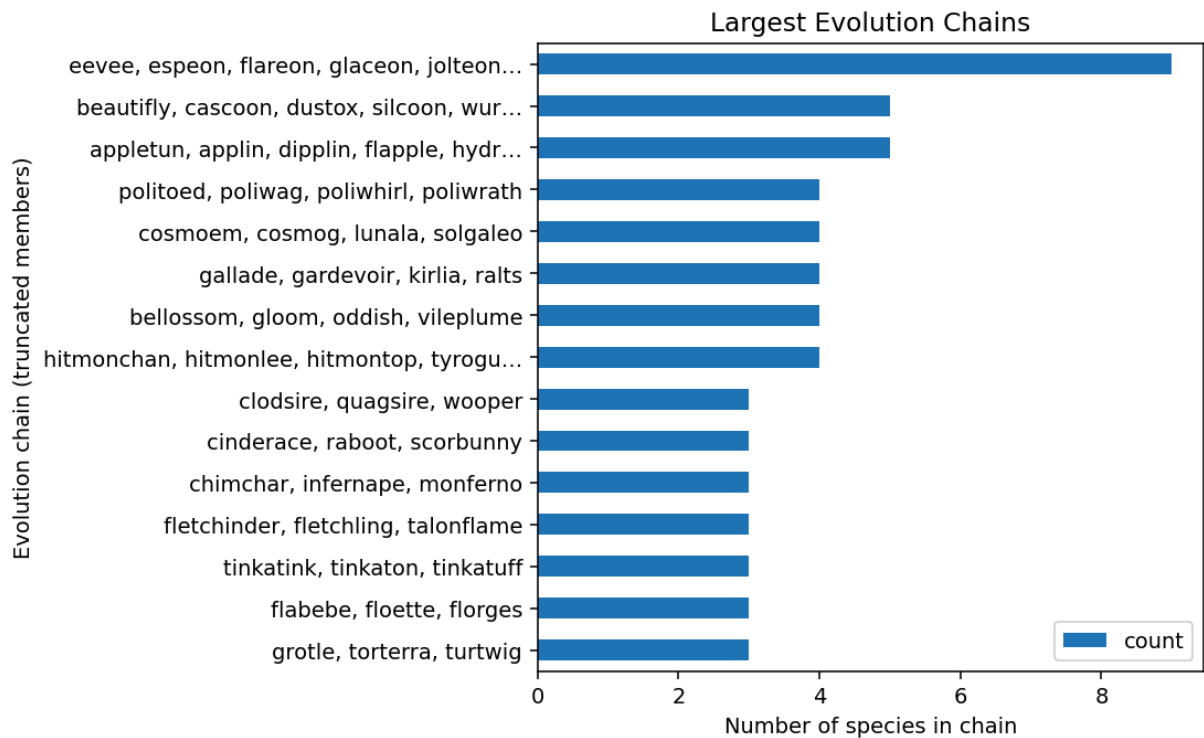


	type	n	hp	atk	def	spa	spd	spe
2	dragon	54	86.8	103.5	83.0	92.9	85.0	86.0
16	steel	50	73.5	95.3	113.9	76.6	79.3	58.3
14	psychic	83	73.6	75.8	71.8	99.2	86.5	79.7
5	fighting	59	78.3	109.3	81.3	58.2	72.0	81.6
6	fire	83	71.9	85.8	70.3	88.1	73.4	75.7
4	fairy	35	72.7	71.2	74.4	84.2	91.7	68.8
7	flying	13	72.8	82.8	68.4	73.0	75.7	85.0
1	dark	61	73.9	87.4	72.8	72.2	73.0	77.7
15	rock	82	67.5	89.6	95.5	64.7	74.4	63.0
10	ground	49	74.6	95.8	87.9	57.2	66.5	64.8
8	ghost	52	63.0	73.0	78.4	83.2	81.0	66.6
13	poison	52	75.5	77.2	77.3	70.8	75.2	67.9
17	water	160	72.7	78.9	74.0	76.4	71.9	69.8
11	ice	44	75.1	80.0	75.2	70.7	72.1	68.8
3	electric	97	61.3	72.3	63.5	85.1	69.3	89.6
9	grass	121	68.5	79.7	74.1	76.9	72.6	62.5
12	normal	139	79.1	77.0	62.1	60.0	65.6	70.1
0	bug	94	57.5	71.8	72.9	58.3	66.5	63.3

`dragon` types, unsurprisingly, have the highest cumulative stats of all pokemon. They are also one of the newer types to be added to the games, which may or may not be an influence on this stat, as `fairy` types were also introduced at the same time. Canonically, `dragon` type pokemon are known to be very resilient to most forms of damage, so it would make sense for them to have high cumulative base stats because of this.

```
In [57]: # Evolution chain sizes (top 15)
pipe = [
    {"$project": {
        "name": 1,
        "chain_id": {
            "$ifNull": [
                "$evolution_chain_id",
                {"$toInt": {
                    "$getField": {
                        "field": "0",
                        "input": {"$regexFind": {
                            "input": "$evolution_chain.url",
                            "regex": r"/evolution-chain/(\d+)/"
                        }}
                    }}
                ]
            }
        }
    }},
    {"$group": {"_id": "$chain_id", "count": {"$sum": 1}, "members": {"$addToSet":
    {"$sort": {"count": -1}}},
    {"$limit": 15}
]
chains = list(db.species_clean.aggregate(pipe))
df = pd.DataFrame(chains).rename(columns={"_id": "chain_id"})
df["label"] = df["members"].apply(lambda m: ", ".join(sorted(m))[:40] + ("..." if len
df = df.sort_values("count", ascending=True)

ax = df.plot(kind="barh", x="label", y="count", figsize=(8,5))
ax.set_xlabel("Number of species in chain")
ax.set_ylabel("Evolution chain (truncated members)")
ax.set_title("Largest Evolution Chains")
plt.tight_layout()
plt.show()
```



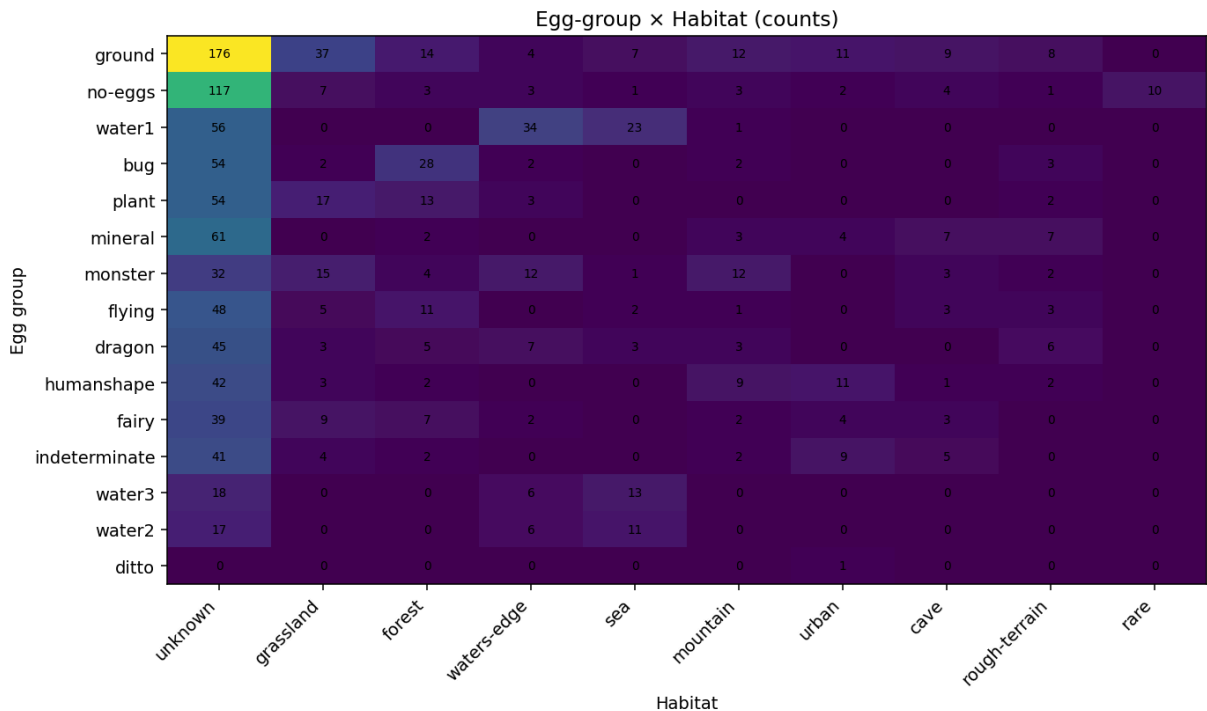
To no Pokemon fan's surprise, the `eevee` evolution chain is the largest in the games. This is because an Eevee can be evolved into 8 different pokemon as of the latest update from its base form. Many fans, however, may not have realized that some pokemon species split evolution chains, as we can see in this bar chart, with the `wurmple` and `applin` chains extending to different final forms, for example.

In [58]: # Egg Group vs Habitat Heatmap

```
pipe = [
    {"$project": {"habitat": {"$ifNull": ["$habitat", "unknown"]}, "egg_groups": 1},
    {"$unwind": "$egg_groups"},
    {"$group": {"_id": {"habitat": "$habitat", "egg": "$egg_groups"}, "count": {"$s
]
cells = list(db.species_clean.aggregate(pipe))
df = pd.DataFrame(cells)
df["habitat"] = df["_id"].apply(lambda d: d["habitat"])
df["egg"] = df["_id"].apply(lambda d: d["egg"])
pt = df.pivot_table(index="egg", columns="habitat", values="count", fill_value=0)
pt = pt.loc[pt.sum(axis=1).sort_values(ascending=False).index] # popular eggs first
pt = pt[pt.sum(axis=0).sort_values(ascending=False).index] # popular habitats

fig, ax = plt.subplots(figsize=(10,6))
im = ax.imshow(pt.values, aspect="auto")
ax.set_xticks(np.arange(pt.shape[1]), labels=pt.columns, rotation=45, ha="right")
ax.set_yticks(np.arange(pt.shape[0]), labels=pt.index)
ax.set_xlabel("Habitat")
ax.set_ylabel("Egg group")
ax.set_title("Egg-group x Habitat (counts)")
```

```
# annotate cells lightly (optional; comment out if slow)
for i in range(pt.shape[0]):
    for j in range(pt.shape[1]):
        ax.text(j, i, int(pt.values[i, j]), ha="center", va="center", fontsize=7)
plt.tight_layout()
plt.show()
```



If we remove the `unknown` habitat from our database, we could see that there is a more or less even distribution of `egg groups` among `habitats`. The massive amount of pokemon within the `unknown` habitat makes it difficult to appreciate this, but we can see a large fraction of egg groups belonging to the `ground` and `no-eggs` groups.

## Sources

- API: [PokeApi.co](https://pokeapi.co)

## Media

## Data

- Pokemon Logo: Nintendo/The Pokemon Company
- Class Diagram: Generated on [mermaidchart.com](https://mermaidchart.com)