# Failed Endpoint Resource Analysis: Error Categories and Suggested Solutions

This analysis categorises failed active endpoint resources into distinct groups based on the failure patterns observed in their URLs or content. Below is a breakdown of the main issue types, including how they are detected and recommended next steps for resolution.

## 1. **Active Document Links**

- **Detection Method:** File extensions in the `endpoint_url` ( `.pdf` , `.docx` , etc), URL slugs like `-pdf` , or content-type headers via a HEAD request.
- **Details:** These links point directly to static files such as PDFs. These formats are not compatible with the expected data pipeline, which likely expects structured machine-readable data (e.g. CSV, GeoJSON, JSON).
- **Suggested Action:** Marked as `recommend_retirement = yes` . These should be retired unless there's a plan to manually process or extract data from the documents.

---

## 2. **Auth Error / Token Required**

- **Detection Method:**
  - JSON body with `{"error": {"code": 499, "message": "Token Required"}}`
  - Response text includes `token required` , `unauthorized` , or HTTP 403-like exceptions.
- **Details:** These endpoints are secured and require a valid authentication token or API key to access data.
- **Suggested Action:**
  - Contact the data provider to obtain access credentials or an API key.
  - If this is not feasible, mark as `non-actionable` and retain without retrying.

---

## 3. **WFS Errors (OGC Web Feature Service)**

- **Detection Method:**
  - URL contains `service=WFS` or `GetFeature`
  - Response includes `<ServiceExceptionReport>` or `<ServiceException>` XML tags.
- **Details:** These errors occur when the `typeName` is invalid, misspelled, or deprecated.
- **Suggested Action:**
  - Visit the WFS endpoint in a browser.
  - Remove the `request=GetFeature` part and replace it with `request=GetCapabilities` to list valid layer names ( `typeName` s).
  - Update the dataset config to reflect a valid `typeName` .

---

## 4. Zipped File

- **Detection Method:** `.zip` found in `endpoint_url`.
- **Details:** These resources are downloadable zip archives. The pipeline likely cannot automatically extract and process these.
- **Suggested Action:**
  - Either retire the source or manually download and extract contents.
  - If viable, publish extracted files to a supported format and location.

---

## 5. Protected XLS File

- **Detection Method:** `.xls` found in `endpoint_url`.
- **Details:** These resources are excel files. Possible issue might occur from files being protected or read only.
- **Suggested Action:**
  - Download and convert to CSV file type.
  - Discuss with source to fix issues before being made available.

---

## 6. Unclassified / Unknown

- **Detection Method:** None of the above heuristics matched.
- **Details:** These rows should be investigated manually. The error could be intermittent, network-related, or require special parsing logic.
- **Suggested Action:**
  - Review the `exception` and `endpoint_url`.
  - Attempt manual access or check historical working status.

---

This categorisation ensures better triage of failing resources, allowing structured decisions around retirement, manual intervention, or reconfiguration.

In [1]:
```python
"""
Script to classify failed resources from Digital Land endpoints,
grouping them by failure type (e.g. document links, WFS errors, auth issues),
and providing suggested resolution details and retirement recommendations.

"""

import pandas as pd
import requests
from io import StringIO

# Utility Functions

def is_pdf_url(url):
    """Check if URL points to a PDF by sending a HEAD request and inspecting Conter
    try:
        response = requests.head(url, allow_redirects=True, timeout=5)
        content_type = response.headers.get("Content-Type", "").lower()
        return "application/pdf" in content_type
    except:
        return False
```

```
def fetch_text_content(url):
    """Fetch and return lowercased plain text content of a URL (used for WFS error
    try:
        r = requests.get(url, timeout=8)
        if r.status_code == 200:
            return r.text.lower()
        return ""
    except:
        return ""

# Load Failed Resources

converted_resource_url_csv = (
    "https://datasette.planning.data.gov.uk/digital-land.csv?"
    "sql=select+dataset%2C+elapsed%2C+r.end_date%2C+r.start_date%2C+exception%2C+r.
    "from+converted_resource+cr+inner+join+resource+r+on+cr.resource%3Dr.resource+"
    "where+status%3D'failed'+and+(r.end_date+is+null+or+r.end_date%3D'')+"
    "order+by+r.start_date+desc+limit+1000"
)
df_failed = pd.read_csv(StringIO(requests.get(converted_resource_url_csv).text))

# Load Supporting Tables

df_endpoint = pd.read_csv("https://datasette.planning.data.gov.uk/digital-land/endp
df_resource_endpoint = pd.read_csv("https://datasette.planning.data.gov.uk/digital-
df_source_raw = pd.read_csv("https://datasette.planning.data.gov.uk/digital-land/so
df_source_raw["organisation_ref"] = df_source_raw["organisation"].str.replace(r"^.*
df_source = df_source_raw[["endpoint", "source", "collection", "organisation_ref"]]

# Merge All Metadata to Failed Records

df_resource_endpoint = df_resource_endpoint.drop_duplicates(subset="resource", keep
df_source = df_source.drop_duplicates(subset="endpoint", keep="last")

df = df_failed.merge(df_resource_endpoint, on="resource", how="left")
df = df.merge(df_endpoint, on="endpoint", how="left")
df = df.merge(df_source, on="endpoint", how="left")

# Classification Logic

def classify_issue(row):
    """
    Classify each row by examining the endpoint URL and exception content.
    Returns a tuple (group, details) with the error category and advice.
    """
    url = str(row.get("endpoint_url") or "").strip()
    exc = str(row.get("exception") or "").lower()

    ext_map = {
        ".pdf": "pdf", ".doc": "doc", ".docx": "docx",
        ".xls": "xls", ".xlsx": "xlsx", ".ppt": "ppt", ".pptx": "pptx"
    }

    # Detect zipped files
    if ".zip" in url.lower():
        return ("zipped file", "file needs to be unzipped first")

    # Classify known document extensions
    for ext, label in ext_map.items():
        if ext in url.lower():
            if label == "xls":
                return ("XLS files", "Possible issues with opening xls file")
            return ("active document links", f"{label} file in URL")
    # Detect slug-style document references
    for label in ext_map.values():
        if f"-{label}" in url.lower():
            if label == "xls":
                return ("XLS files", "Possible issues with opening xls file")
            return ("active document links", f"{label} inferred from slug")

    # Fallback: test if it's actually a PDF by header
    if is_pdf_url(url):
        return ("active document links", "confirmed via Content-Type check")

    # Detect known ArcGIS JSON auth error
    try:
        r = requests.get(url, timeout=8)
        if r.headers.get("Content-Type", "").lower().startswith("application/json")
            json_body = r.json()
            error = json_body.get("error", {})
            if str(error.get("code", "")).strip() == "499" or "token" in error.get(
                return ("auth error", "Token or authentication required (JSON error
        elif "token required" in r.text.lower():
            return ("auth error", "Token or authentication required (text body)")
    except:
        pass

    # WFS ServiceException detection
    if "getfeature" in url.lower() or "wfs" in url.lower():
        text = fetch_text_content(url)
        if "serviceexception" in text and "feature" in text:
            return ("wfs error", "Likely invalid typeName - check WFS GetCapabiliti

    return (None, "")

# Apply classifier
df[["group", "details"]] = df.apply(lambda row: pd.Series(classify_issue(row)), axi

# Manual Patch for Known PDF URLs with No Hints

force_pdf_urls = [
    "https://www.bolton.gov.uk/downloads/file/3212/infrastructure-funding-statement
    "https://www.bolton.gov.uk/downloads/file/4045/infrastructure-funding-statement
]
df.loc[df["endpoint_url"].isin(force_pdf_urls), "group"] = "active document links"
df.loc[df["endpoint_url"].isin(force_pdf_urls), "details"] = "manually flagged as p

# Retirement Recommendation Logic

df["recommend_retirement"] = df["group"].apply(lambda g: "yes" if g == "active docu

# Final Output

df_out = df[[
    "resource", "source", "collection", "endpoint_url", "group", "details", "recomm
]]
df_out.to_csv("flagged_failed_resources.csv", index=False)
```