data_analysis.py

```
from .fetch_CVE import get_all_cves
from .group_CVE import group_by_severity, group_by_cwe, timeline_group,
top_n_cwes
from .cwe_map import cwe_title, CWE_TITLES
from functools import lru_cache
from datetime import datetime
from collections import defaultdict

@lru_cache(maxsize=1)
def _cached_cves():
    #Pull all CVEs and cache the whole batch (so repeated calls are instant)
    #Tuple used for lru_cache (hashable).
    return tuple(get_all_cves())

def get_dashboard_metrics():
    #Just group CVEs by severity, count them up for dashboard pie chart
    cves = _cached_cves()
    return group_by_severity(cves)

def get_timeline_data():
    #Monthly CVE counts for the timeline (for the chart up top)
    cves = _cached_cves()
    timeline = defaultdict(int)
    for cve in cves:
        date = cve.get("Published", "")
        if date:
            #Only "YYYY-MM" (easier for charting)
            try:
                month = date[:7]  # "YYYY-MM"
                timeline[month] += 1
            except:
                continue
    #Sort months chronologically for the graph
    sorted_months = sorted(timeline.items())
    return {
        "labels": [item[0] for item in sorted_months],
        "values": [item[1] for item in sorted_months]
    }

def get_severity_stats():
    #Quick severity count for summary stats (CRITICAL/HIGH/MEDIUM/LOW)
```

```
    cves = _cached_cves()
    counts = {
        'CRITICAL': 0,
        'HIGH': 0,
        'MEDIUM': 0,
        'LOW': 0
    }
    for cve in cves:
        sev = cve.get("Severity", "").upper()
        if sev in counts:
            counts[sev] += 1
    return counts

def get_cwe_radar():
    #For the radar chart: count CVEs per CWE, then show top 7 visually
    cves = _cached_cves()
    cwe_counts = defaultdict(int)
    for cve in cves:
        cwe = cve.get("CWE")
        if cwe:
            cwe_counts[cwe] += 1
    #Get top 7 CWEs by frequency for visualization
    top_cwes = sorted(cwe_counts.items(), key=lambda x: x[1], reverse=True)
[:7]
    return {
        "labels": [cwe_title(cwe) for cwe, _ in top_cwes],
        "values": [count for _, count in top_cwes]
    }

def get_yearly_trends():
    #Count CVEs published each year for last 5 (for trend-over-time chart)
    cves = _cached_cves()
    current_year = datetime.now().year
    year_counts = {year: 0 for year in range(current_year - 4, current_year +
1)}
    for cve in cves:
        published = cve.get('Published', '')
        if published:
            try:
                year = int(published[:4])
                if year in year_counts:
                    year_counts[year] += 1
            except ValueError:
                continue
    return {
        "labels": list(year_counts.keys()),
```

```
        "values": list(year_counts.values())
    }

def get_top_cwes(n=7):
    #Return the top N most frequent CWEs (good for table or bar chart)
    cves = _cached_cves()
    cwe_counts = group_by_cwe(cves)
    return [cwe_title(cwe) for cwe in top_n_cwes(cwe_counts, n=n)]

def get_latest_cves(n=5):
    # Give back the N most recently published CVEs (lets you show latest
threats)
    cves = _cached_cves()
    sorted_cves = sorted(
        cves,
        key=lambda x: x.get('Published') or '',
        reverse=True
    )
    return [
        {
            "id": cve.get("ID"),
            "description": cve.get("Description"),
            "severity": cve.get("Severity", "UNKNOWN")
        }
        for cve in sorted_cves[:n]
    ]

def get_product_count():
    #How many unique products are affected in this whole CVE set?
    cves = _cached_cves()
    prods = set()
    for cve in cves:
        for prod in cve.get("Products", []):
            prods.add(prod)
    return len(prods)

def search_cves(q=None, severity=None):
    # Search CVEs by keyword and (optionally) severity.
    # Fast fuzzy search for dashboard
    cves = _cached_cves()
    results = []
    for cve in cves:
        _id = cve.get("ID", "")
        _desc = cve.get("Description", "")
        _sev = cve.get("Severity", "UNKNOWN")
        #If query is set, it must match ID at least somewhere
```

```
        if q and q.lower() not in _id.lower() and q.lower() not in
_desc.lower():
            continue
        #If filtering by severity, match (case-insensitive)
        if severity and _sev.upper() != severity.upper():
            continue
        results.append({
            "id": _id,
            "description": _desc,
            "severity": _sev
        })
    return results

def get_cve_by_id(cve_id):
    #Look up full details for one CVE by ID (hit NVD API)
    from .nvd_api import get_cve_detail
    return get_cve_detail(cve_id)

def get_cwe_severity_data():
    #For each CWE, count CVEs at each severity (even if zero).
    #Makes the severity bar chart stacked (all CWEs show, even with zero
counts).
    from collections import defaultdict
    cves = _cached_cves()
    cwe_severity = defaultdict(lambda: defaultdict(int))
    for cve in cves:
        cwe = cve.get('CWE')
        if cwe:
            severity = cve.get('Severity', 'UNKNOWN').upper()
            if severity in ['CRITICAL', 'HIGH', 'MEDIUM', 'LOW']:
                cwe_severity[cwe][severity] += 1
    # Always include every CWE from CWE_TITLES, even if it's all zeros
    cwe_keys = list(CWE_TITLES.keys())
    return {
        'labels': [cwe_title(cwe) for cwe in cwe_keys],
        'data': {
            'CRITICAL': [cwe_severity[cwe].get('CRITICAL', 0) for cwe in
cwe_keys],
            'HIGH':     [cwe_severity[cwe].get('HIGH', 0) for cwe in
cwe_keys],
            'MEDIUM':   [cwe_severity[cwe].get('MEDIUM', 0) for cwe in
cwe_keys],
            'LOW':      [cwe_severity[cwe].get('LOW', 0) for cwe in cwe_keys]
        }
    }
```