03_Exploratory_Analysis (Python)

Import notebook

GOLD LAYER: Exploratory Data Analysis

```
from pyspark.sql.functions import col, count, avg, sum as spark_sum, min as spark_min, max as spark_max, year,
month, datediff

print("=" * 60)
print("GOLD LAYER - EXPLORATORY ANALYSIS")
print("=" * 60)
print("\nAnalyzing CVE data from Silver layer...")
print(f"  Source: cve_silver.core")
print(f"  Source: cve_silver.affected_products")
print("=" * 60 + "\n")
```

```
============================================================
GOLD LAYER - EXPLORATORY ANALYSIS
============================================================

Analyzing CVE data from Silver layer...
  Source: cve_silver.core
  Source: cve_silver.affected_products
============================================================
```

Analysis 1: Temporal Analysis - Yearly CVE Counts

```
print("=" * 60)
print("1. TEMPORAL ANALYSIS - Yearly CVE Counts")
print("=" * 60)

df_yearly = spark.sql("""
    SELECT
        YEAR(date_published) as year,
        COUNT(*) as cve_count
    FROM cve_silver.core
    WHERE date_published IS NOT NULL
    GROUP BY YEAR(date_published)
    ORDER BY year
""")

display(df_yearly)

print("\n✅ Temporal analysis complete\n")
```

▸ df_yearly: pyspark.sql.connect.dataframe.DataFrame = [year: integer, cve_count: long]

```
============================================================
1. TEMPORAL ANALYSIS - Yearly CVE Counts
============================================================
```

Table

;

✅ Temporal analysis complete

Analysis 2: Publication Latency (Reserved vs Published)

```
print("=" * 60)
print("2. PUBLICATION LATENCY ANALYSIS")
print("=" * 60)

df_latency = spark.sql("""
    SELECT
        ROUND(AVG(DATEDIFF(date_published, date_reserved)), 2) as avg_days_to_publish,
        MIN(DATEDIFF(date_published, date_reserved)) as min_days,
        MAX(DATEDIFF(date_published, date_reserved)) as max_days,
        COUNT(*) as records_with_both_dates
    FROM cve_silver.core
    WHERE date_reserved IS NOT NULL
      AND date_published IS NOT NULL
      AND date_published >= date_reserved
""")

display(df_latency)

print("\n✅ Publication latency analysis complete\n")
```

▸ df_latency: pyspark.sql.connect.dataframe.DataFrame = [avg_days_to_publish: double, min_days: integer ... 2 more fields]

```
============================================================
2. PUBLICATION LATENCY ANALYSIS
============================================================
```

Table

✅ Publication latency analysis complete

Analysis 3: Monthly Publication Patterns (Seasonality)

```
print("=" * 60)
print("3. MONTHLY PUBLICATION PATTERNS")
print("=" * 60)

df_monthly_patterns = spark.sql("""
    SELECT
        CASE MONTH(date_published)
            WHEN 1 THEN 'Jan'
            WHEN 2 THEN 'Feb'
            WHEN 3 THEN 'Mar'
            WHEN 4 THEN 'Apr'
            WHEN 5 THEN 'May'
            WHEN 6 THEN 'Jun'
            WHEN 7 THEN 'Jul'
            WHEN 8 THEN 'Aug'
            WHEN 9 THEN 'Sep'
            WHEN 10 THEN 'Oct'
            WHEN 11 THEN 'Nov'
            WHEN 12 THEN 'Dec'
        END as month,
        COUNT(*) as cve_count
    FROM cve_silver.core
    WHERE date_published IS NOT NULL
    GROUP BY MONTH(date_published)
    ORDER BY MONTH(date_published)
""")

display(df_monthly_patterns)

print("\n✅ Monthly patterns analysis complete\n")
```

▸ df_monthly_patterns: pyspark.sql.connect.dataframe.DataFrame = [month: string, cve_count: long]

```
============================================================
3. MONTHLY PUBLICATION PATTERNS
============================================================
```

Table Visualization 1

✅ Monthly patterns analysis complete

Analysis 4: CVSS Risk Score Distribution

```
# ============================================================
# Analysis 4: CVSS Risk Distribution
# ============================================================

print("=" * 60)
print("4. CVSS RISK DISTRIBUTION")
print("=" * 60)

df_risk = spark.sql("""
    SELECT
        cvss_severity as risk_category,
        COUNT(*) as cve_count,
        ROUND(COUNT(*) * 100.0 / SUM(COUNT(*)) OVER(), 2) as percentage
    FROM cve_silver.core
    WHERE cvss_severity IS NOT NULL
    GROUP BY cvss_severity
    ORDER BY
        CASE cvss_severity
            WHEN 'CRITICAL' THEN 1
            WHEN 'HIGH' THEN 2
            WHEN 'MEDIUM' THEN 3
            WHEN 'LOW' THEN 4
        END
""")

display(df_risk)

print("\n✅ CVSS distribution analysis complete\n")
```

▸ df_risk: pyspark.sql.connect.dataframe.DataFrame = [risk_category: string, cve_count: long ... 1 more field]

```
============================================================
4. CVSS RISK DISTRIBUTION
============================================================
```

Table Visualization 1

✅ CVSS distribution analysis complete

Analysis 5: Top 25 Vendors by Vulnerability Count

```
print("=" * 60)
print("5. VENDOR INTELLIGENCE - Top 25 Vendors")
print("=" * 60)

df_top_vendors = spark.sql("""
    SELECT
        vendor,
        COUNT(DISTINCT cve_id) as vulnerability_count,
        COUNT(*) as affected_products_count
    FROM cve_silver.affected_products
    WHERE vendor IS NOT NULL
    GROUP BY vendor
    ORDER BY vulnerability_count DESC
    LIMIT 25
""")

display(df_top_vendors)

print("\n✅ Vendor intelligence analysis complete\n")
```

▸ df_top_vendors: pyspark.sql.connect.dataframe.DataFrame = [vendor: string, vulnerability_count: long ... 1 more field]

```
============================================================
5. VENDOR INTELLIGENCE - Top 25 Vendors
============================================================
```

Table Visualization 1

✅ Vendor intelligence analysis complete

Analysis 6: CVE State Distribution

```
print("=" * 60)
print("6. CVE STATE DISTRIBUTION")
print("=" * 60)

df_state = spark.sql("""
    SELECT
        state,
        COUNT(*) as count,
        ROUND(COUNT(*) * 100.0 / SUM(COUNT(*)) OVER(), 2) as percentage
    FROM cve_silver.core
    GROUP BY state
    ORDER BY count DESC
""")

display(df_state)

print("\n✅ State distribution analysis complete\n")
```

▸ df_state: pyspark.sql.connect.dataframe.DataFrame = [state: string, count: long ... 1 more field]

```
============================================================
6. CVE STATE DISTRIBUTION
============================================================
```

Table Visualization 1

✅ State distribution analysis complete

Analysis 7: Market Concentration Index

```
print("=" * 60)
print("7. MARKET CONCENTRATION - Vendor Analysis")
print("=" * 60)

df_concentration = spark.sql("""
    WITH vendor_counts AS (
        SELECT
            vendor,
            COUNT(DISTINCT cve_id) as cve_count
        FROM cve_silver.affected_products
        WHERE vendor IS NOT NULL
        GROUP BY vendor
    ),
    totals AS (
        SELECT SUM(cve_count) as total_cves FROM vendor_counts
    ),
    top10 AS (
        SELECT SUM(cve_count) as top10_cves
        FROM (SELECT cve_count FROM vendor_counts ORDER BY cve_count DESC LIMIT 10)
    ),
    top25 AS (
        SELECT SUM(cve_count) as top25_cves
        FROM (SELECT cve_count FROM vendor_counts ORDER BY cve_count DESC LIMIT 25)
    )
    SELECT
        'Top 10 vendors' as segment,
        top10_cves as cves,
        ROUND(top10_cves * 100.0 / total_cves, 2) as percentage
    FROM top10, totals
    UNION ALL
    SELECT
        'Top 25 vendors' as segment,
        top25_cves as cves,
        ROUND(top25_cves * 100.0 / total_cves, 2) as percentage
    FROM top25, totals
    UNION ALL
    SELECT
        'All vendors' as segment,
        total_cves as cves,
        100.0 as percentage
    FROM totals
""")

display(df_concentration)

print("\n✅ Market concentration analysis complete\n")
```

▸ df_concentration: pyspark.sql.connect.dataframe.DataFrame = [segment: string, cves: long ... 1 more field]

```
============================================================
7. MARKET CONCENTRATION - Vendor Analysis
============================================================
```

Table Visualization 1

```
✅ Market concentration analysis complete
```

Analysis 8: Monthly Publication Trends

```
print("=" * 60)
print("8. MONTHLY TRENDS - 2024 CVE Count & Average CVSS")
print("=" * 60)

df_monthly = spark.sql("""
    SELECT
        CASE MONTH(date_published)
            WHEN 1 THEN 'Jan'
            WHEN 2 THEN 'Feb'
            WHEN 3 THEN 'Mar'
            WHEN 4 THEN 'Apr'
            WHEN 5 THEN 'May'
            WHEN 6 THEN 'Jun'
            WHEN 7 THEN 'Jul'
            WHEN 8 THEN 'Aug'
            WHEN 9 THEN 'Sep'
            WHEN 10 THEN 'Oct'
            WHEN 11 THEN 'Nov'
            WHEN 12 THEN 'Dec'
        END as month,
        COUNT(*) as cve_count,
        ROUND(AVG(cvss_base_score), 2) as avg_cvss_score
    FROM cve_silver.core
    WHERE YEAR(date_published) = 2024
      AND cvss_base_score IS NOT NULL
    GROUP BY MONTH(date_published)
    ORDER BY MONTH(date_published)
""")

display(df_monthly)
```

```
▸ df_monthly:  pyspark.sql.connect.dataframe.DataFrame = [month: string, cve_count: long ... 1 more field]
============================================================
8. MONTHLY TRENDS - 2024 CVE Count & Average CVSS
============================================================
```

Table Visualization 1

Analysis 9: Seasonal Vulnerability Patterns

▸ df_seasonal: pyspark.sql.connect.dataframe.DataFrame = [season: string, cve_count: long ... 1 more field]

```
============================================================
9. SEASONAL ANALYSIS - CVE Patterns by Season
============================================================
```

Table Visualization 1

✅ Seasonal analysis complete