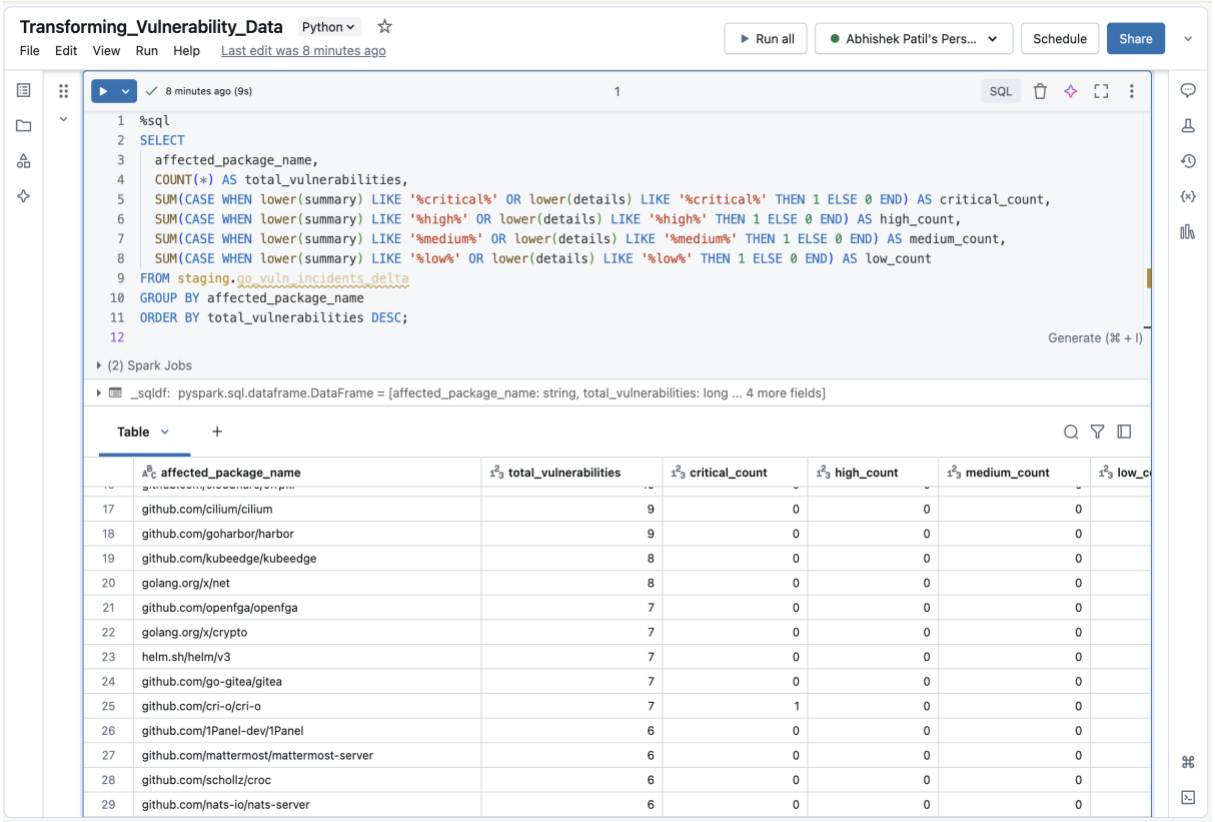
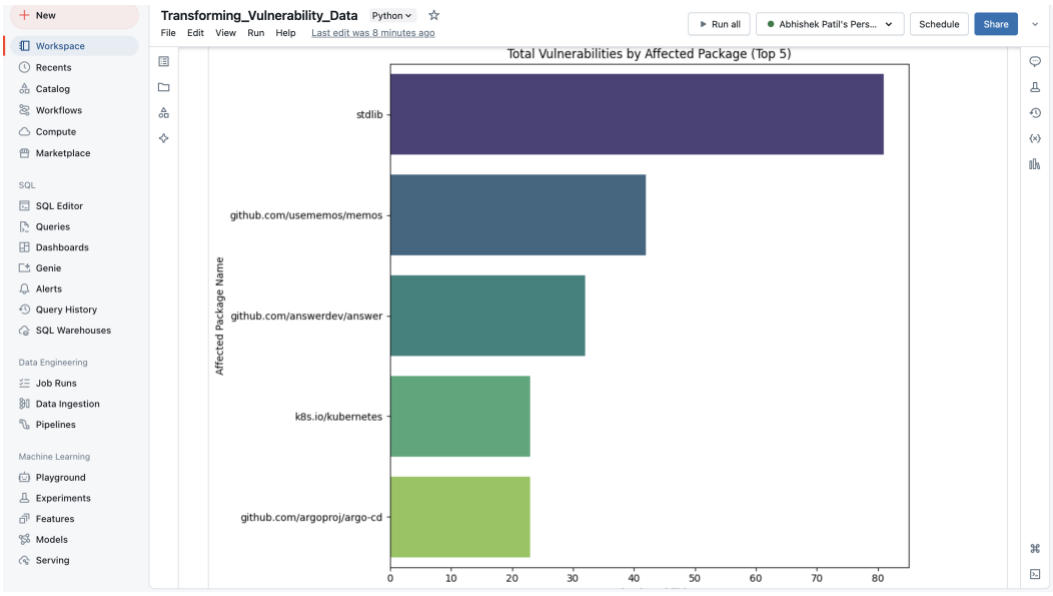
Spark Jobs for Transformations

1. Severity Analysis

```
SELECT
  affected_package_name,
  COUNT(*) AS total_vulnerabilities,
  SUM(CASE WHEN lower(summary) LIKE '%critical%' OR lower(details) LIKE '%critical%' THEN 1 ELSE 0 END) AS critical_count,
  SUM(CASE WHEN lower(summary) LIKE '%high%' OR lower(details) LIKE '%high%' THEN 1 ELSE 0 END) AS high_count,
  SUM(CASE WHEN lower(summary) LIKE '%medium%' OR lower(details) LIKE '%medium%' THEN 1 ELSE 0 END) AS medium_count,
  SUM(CASE WHEN lower(summary) LIKE '%low%' OR lower(details) LIKE '%low%' THEN 1 ELSE 0 END) AS low_count
FROM staging.go_vuln_incidents_delta
GROUP BY affected_package_name
ORDER BY total_vulnerabilities DESC;
```


Transforming_Vulnerability_Data
Python
File Edit View Run Help
Last edit was 8 minutes ago
Run all
Abhishek Patil's Pers...
Schedule
Share
8 minutes ago (9s)
SQL
%sql
SELECT
affected_package_name,
COUNT(*) AS total_vulnerabilities,
SUM(CASE WHEN lower(summary) LIKE '%critical%' OR lower(details) LIKE '%critical%' THEN 1 ELSE 0 END) AS critical_count,
SUM(CASE WHEN lower(summary) LIKE '%high%' OR lower(details) LIKE '%high%' THEN 1 ELSE 0 END) AS high_count,
SUM(CASE WHEN lower(summary) LIKE '%medium%' OR lower(details) LIKE '%medium%' THEN 1 ELSE 0 END) AS medium_count,
SUM(CASE WHEN lower(summary) LIKE '%low%' OR lower(details) LIKE '%low%' THEN 1 ELSE 0 END) AS low_count
FROM staging.go_vuln_incidents_delta
GROUP BY affected_package_name
ORDER BY total_vulnerabilities DESC;
Generate (⌘ + I)
(2) Spark Jobs
_sqldf: pyspark.sql.dataframe.DataFrame = [affected_package_name: string, total_vulnerabilities: long ... 4 more fields]
Table
affected_package_name
total_vulnerabilities
critical_count
high_count
medium_count
17 github.com/cilium/cilium 9 0 0 0
18 github.com/goharbor/harbor 9 0 0 0
19 github.com/kubeedge/kubeedge 8 0 0 0
20 golang.org/x/net 8 0 0 0
21 github.com/openfga/openfga 7 0 0 0
22 golang.org/x/crypto 7 0 0 0
23 helm.sh/helm/v3 7 0 0 0
24 github.com/go-gitea/gitea 7 0 0 0
25 github.com/cri-o/cri-o 7 1 0 0
26 github.com/1Panel-dev/1Panel 6 0 0 0
27 github.com/mattermost/mattermost-server 6 0 0 0
28 github.com/schollz/croc 6 0 0 0
29 github.com/nats-io/nats-server 6 0 0 0


New
Workspace
Recents
Catalog
Workflows
Compute
Marketplace
SQL
SQL Editor
Queries
Dashboards
Genie
Alerts
Query History
SQL Warehouses
Data Engineering
Job Runs
Data Ingestion
Pipelines
Machine Learning
Playground
Experiments
Features
Models
Serving
Transforming_Vulnerability_Data
Python
File Edit View Run Help
Last edit was 8 minutes ago
Run all
Abhishek Patil's Pers...
Schedule
Share
Total Vulnerabilities by Affected Package (Top 5)
Affected Package Name
stdlib
github.com/usememos/memos
github.com/answerdev/answer
k8s.io/kubernetes
github.com/argoproj/argo-cd
0
10
20
30
40
50
60
70
80

2. Vulnerability Reporting Trends

This query aggregates the number of vulnerability records per month (and year) for each ecosystem. It ignores records with default dates.

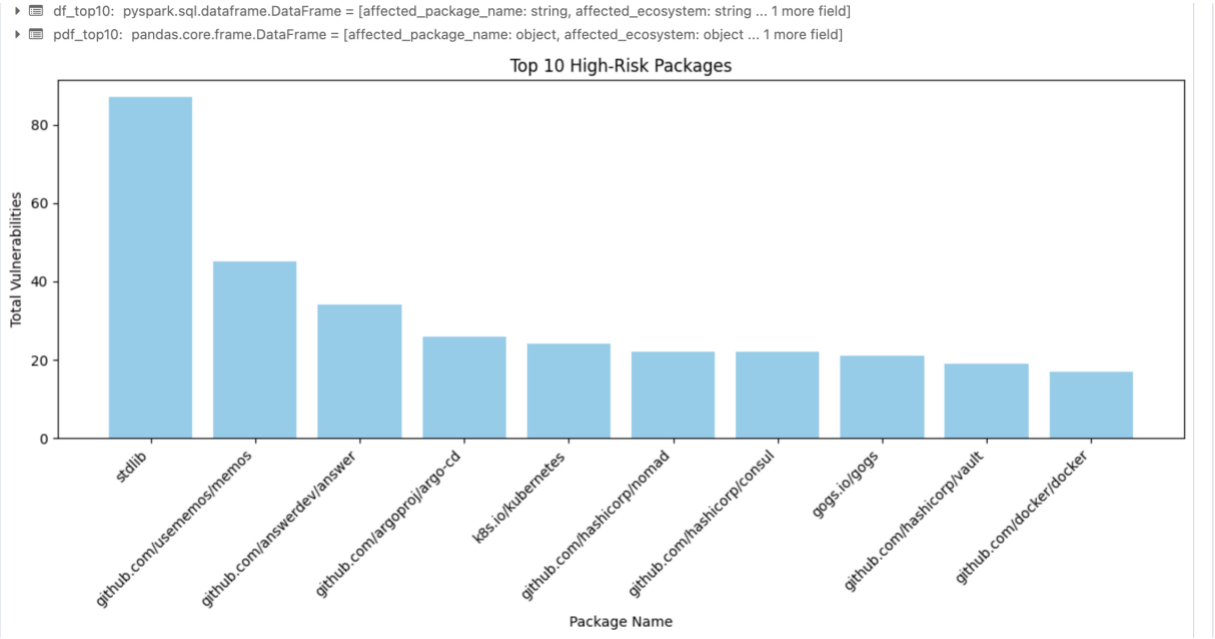
```
import matplotlib.pyplot as plt


# Run the query to extract top 10 high-risk packages
query = """
SELECT
  affected_package_name,
  affected_ecosystem,
  COUNT(*) AS total_vulnerabilities
FROM staging.go_vuln_incidents_delta
GROUP BY affected_package_name, affected_ecosystem
ORDER BY total_vulnerabilities DESC
LIMIT 10
"""


# Execute the query in Spark
df_top10 = spark.sql(query)


# Convert the Spark DataFrame to Pandas DataFrame for plotting
pdf_top10 = df_top10.toPandas()


# Plotting the bar chart
plt.figure(figsize=(12,6))
plt.bar(pdf_top10['affected_package_name'], pdf_top10['total_vulnerabilities'], color='skyblue')
plt.xlabel("Package Name")
plt.ylabel("Total Vulnerabilities")
plt.title("Top 10 High-Risk Packages")
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



3. Textual Insights from Summaries and Details

a. Common Patterns (Keyword Frequency)

You can tokenize the combined text from `summary` and `details` to identify the most frequent keywords. Here's an example using a common table expression (CTE):

```
WITH tokens AS (
  SELECT explode(
          split(
            lower(regexp_replace(concat(summary, ' ', details), '[^a-zA-Z0-9 ]', '')),
            ' '
          )
        ) AS token
  FROM staging.go_vuln_incidents_delta
),
filtered_tokens AS (
  SELECT token, COUNT(*) AS frequency
```

```
  FROM tokens
  WHERE token <> ''
    -- Exclude common English stop words
    AND token NOT IN (
        'the', 'and', 'for', 'are', 'but', 'not', 'you', 'all', 'any', 'can',
        'her', 'was', 'one', 'our', 'out', 'day', 'get', 'has', 'had', 'him',
        'his', 'how', 'its', 'let', 'say', 'she', 'too', 'use', 'in', 'of', 'to'
    )
    -- Optionally exclude very short tokens
    AND length(token) > 3
  GROUP BY token
)
SELECT token, frequency
FROM filtered_tokens
WHERE frequency > 10  -- Filter to tokens with high count (adjust threshold as needed)
ORDER BY frequency DESC
LIMIT 20;
```

4. Patching Timelines

Assuming you want to compute the duration between the introduction and fix events, if those fields are stored as dates, you can compute the difference (if not, you might consider using `published_dt` vs. `modified_dt` as a proxy). Here are two options:

```
from pyspark.sql import functions as F
import matplotlib.pyplot as plt


# Step 1: Query the data from the staging table ensuring valid dates for patch duration calculation
df = spark.sql("""
    SELECT
  affected_package_name,
```

```
  published_dt,
  modified_dt,
  DATEDIFF(modified_dt, published_dt) AS patch_duration_days
FROM staging.go_vuln_incidents_delta
WHERE published_dt <> '9999-99-9' OR published_dt  !='0001-01-01'
  AND modified_dt <> '9999-99-9'OR modified_dt  !='0001-01-01';

""")

# Step 2: Aggregate by package to calculate average patch duration per package
agg_df = (
    df.groupBy("affected_package_name")
      .agg(F.avg("patch_duration_days").alias("avg_patch_duration"))
      .orderBy(F.desc("avg_patch_duration"))
)

# Step 3: Select top 10 packages with the highest average patch duration
top10_df = agg_df.limit(10)

# Convert to Pandas DataFrame for visualization
top10_pd = top10_df.toPandas()

# Step 4: Plotting the data using Matplotlib
plt.figure(figsize=(12, 6))
plt.bar(top10_pd['affected_package_name'], top10_pd['avg_patch_duration'], color='tomato')
plt.xlabel("Affected Package Name")
plt.ylabel("Average Patch Duration (Days)")
plt.title("Top 10 Packages Taking Longest to Fix Vulnerabilities")
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```