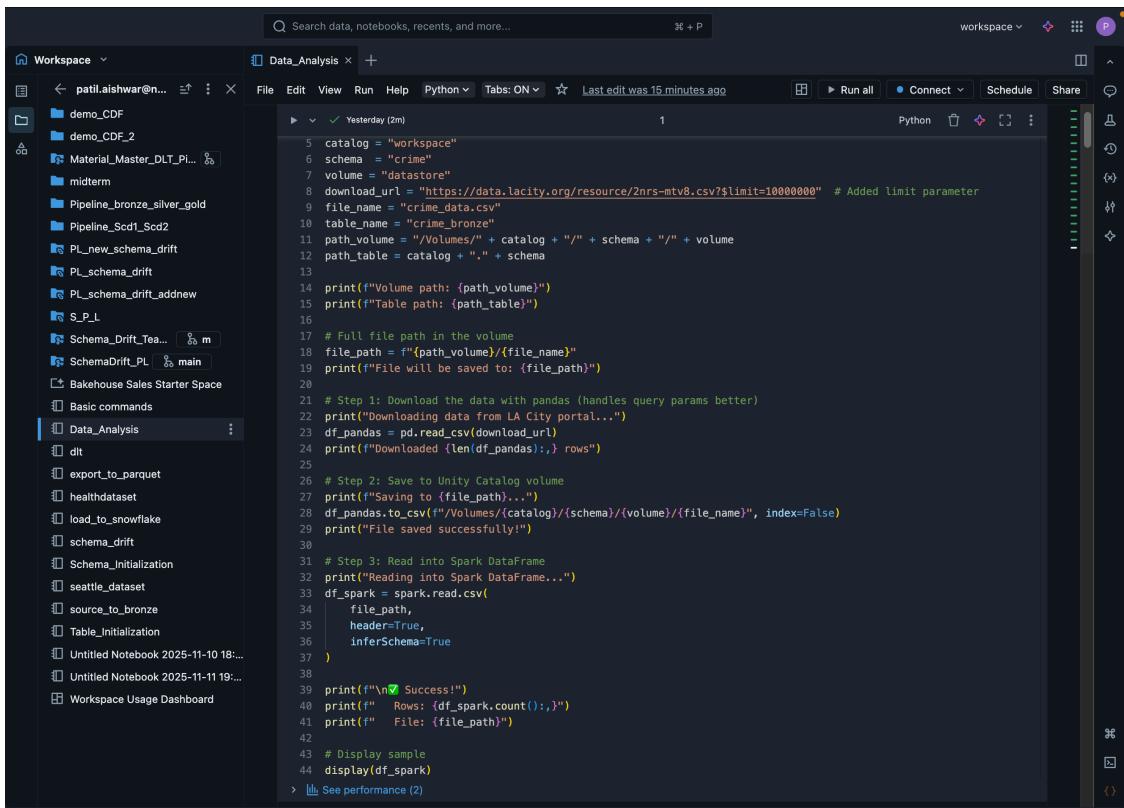


LA Crime Analysis Pipeline

Section 1: Data Ingestion from LA City Portal

Downloaded crime incident data from LA City's open data API and store it in Databricks Unity Catalog Volume
catalog = "workspace" schema = "crime" volume = "datastore"



The screenshot shows a Databricks workspace interface. On the left, there is a sidebar with a tree view of notebooks and workspace contents. The main area is a notebook titled 'Data_Analysis'. The code in the notebook is as follows:

```
5 catalog = "workspace"
6 schema = "crime"
7 volume = "datastore"
8 download_url = "https://data.lacity.org/resource/2nrs-mtv8.csv?$limit=10000000" # Added limit parameter
9 file_name = "crime_data.csv"
10 table_name = "crime_bronze"
11 path_volume = "/Volumes/" + catalog + "/" + schema + "/" + volume
12 path_table = catalog + "." + schema
13
14 print(f"Volume path: {path_volume}")
15 print(f"Table path: {path_table}")
16
17 # Full file path in the volume
18 file_path = f'{path_volume}/{file_name}'
19 print(f"File will be saved to: {file_path}")
20
21 # Step 1: Download the data with pandas (handles query params better)
22 print("Downloading data from LA City portal...")
23 df_pandas = pd.read_csv(download_url)
24 print(f"Downloaded {len(df_pandas)} rows")
25
26 # Step 2: Save to Unity Catalog volume
27 print(f"Saving to {file_path}...")
28 df_pandas.to_csv(f"/Volumes/{catalog}/{schema}/{volume}/{file_name}", index=False)
29 print("File saved successfully!")
30
31 # Step 3: Read into Spark DataFrame
32 print("Reading into Spark DataFrame...")
33 df_spark = spark.read.csv(
34     file_path,
35     header=True,
36     inferSchema=True
37 )
38
39 print(f"\n✓ Success!")
40 print(f" Rows: {df_spark.count()}")
41 print(f" File: {file_path}")
42
43 # Display sample
44 display(df_spark)
> See performance (2)
```

Downloaded via pandas to handle API query parameters and download full dataset
Saved to Unity Catalog Volume - Persists CSV file to /Volumes/workspace/crime/datastore/
crime_data.csv
Load into Spark DataFrame - Reads CSV into Spark with schema inference for distributed
processing;
And then created Bronze Table - Registers data as workspace.crime.crime_bronze table

Expected Output:

- Dataset: ~1 million crime incident records
- Columns: 28 attributes covering incident details, location, victim demographics, and case status

Dataset Overview

Checks Performed:

- Total row count and column count
- Primary key integrity (duplicate DR_NO detection)
- Schema validation (column names and data types)

Screenshot of Databricks workspace showing the LA_Crime_Gold_Pipeline.

The sidebar navigation includes:

- New
- Workspace
- Recents
- Catalog
- Jobs & Pipelines
- Compute
- Marketplace
- SQL
- SQL Editor
- Queries
- Dashboards
- Genie
- Alerts
- Query History
- SQL Warehouses
- Data Engineering
- Job Runs
- Data Ingestion
- AI/ML
- Playground
- Experiments
- Features
- Models
- Serving

The main area displays the Pipeline configuration for the LA_Crime_Gold_Pipeline. The pipeline assets listed include:

- demo_CDF
- demo_CDF_2
- Material_Master_DLT_Pl...
- midterm
- Pipeline_bronze_silver_gold
- crime_bronze_to_silver
- gold_dims_table
- gold_fact_table
- silver_to_stage_Scd2
- Pipeline_Scd1_Scd2
- PL_new_schema_drift
- PL_schema_drift
- PL_schema_drift_addnew
- S_P_L
- Schema_Drift_Tea...
- SchemaDrift_PL
- Bakehouse Sales Start...
- Basic commands
- Data_Analysis
- dt
- export_to_parquet
- healthdataset
- load_to_snowflake
- schema_drift
- Schema_Initialization
- seattle_dataset

The Pipeline graph shows the data flow between these assets. A Python notebook titled "silver_to_stage_Scd2" is open, showing code for dataset overview and null value analysis. The notebook output shows the following Python code:

```
20 total_cols = len(df.columns)
21
22 print("\n1.1 DATASET OVERVIEW")
23 print("(*=*80*")
24 print(f"Total Records: {total_rows},")
25 print(f"Total Columns: {total_cols}")
26
27 # Check duplicates
28 duplicate_cases = df.groupby('DR_NO').count().filter(col ('count') > 1).count()
29 print(f"Duplicate Case Numbers: {duplicate_cases};")
30
31 # Display schema
32 print("\n1.2 DATA TYPES")
33 print("(*=*80*")
34 df.printSchema()
> [See performance (2)]
```

The notebook also contains a table titled "Tables 12 Performance 12" with the following data:

vict_sex	144644	14.39
vict_descent	144656	14.39
premis_desc	588	0.06
dr_no	0	0.00
lat	0	0.00
location	0	0.00
crm_cd_1	11	0.00
status_desc	0	0.00
premis_cd	16	0.00
date_rptd	0	0.00
vict_age	0	0.00
crm_cd_desc	0	0.00

Screenshot of Databricks workspace showing the LA_Crime_Gold_Pipeline.

The sidebar navigation is identical to the first screenshot.

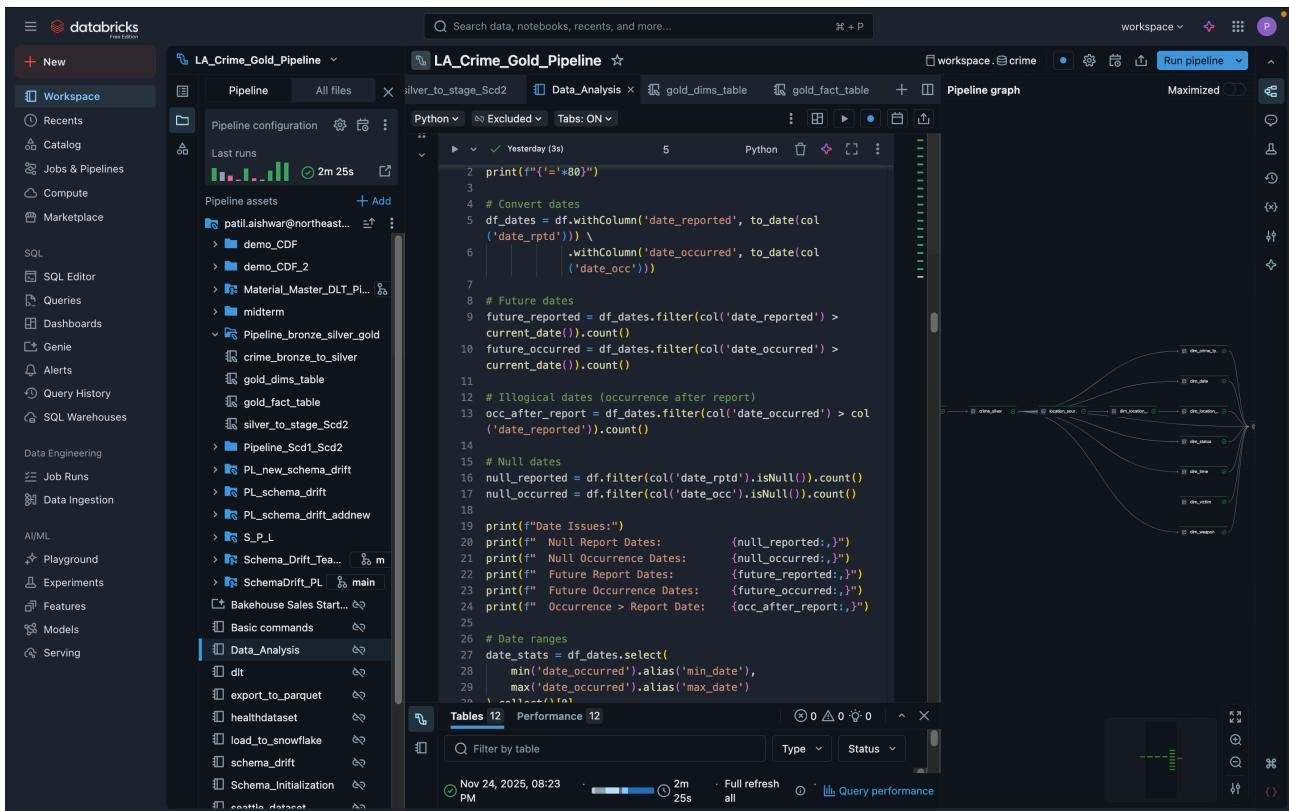
The main area displays the Pipeline configuration for the LA_Crime_Gold_Pipeline. The pipeline assets listed are the same as the first screenshot.

The Pipeline graph shows the data flow between these assets. A Python notebook titled "silver_to_stage_Scd2" is open, showing code for dataset overview and null value analysis. The notebook output shows the following Python code:

```
1 print("\n1.3 NULL VALUE ANALYSIS")
2 print("(*=*80*")
3 # Calculate nulls for all columns
4 null_analysis = []
5 for column_name in df.columns:
6     null_count = df.filter(col(column_name).isNull()).count()
7     null_pct = (null_count / total_rows) * 100
8     null_analysis.append({
9         'Column': column_name,
10        'Null_Count': null_count,
11        'Null_Percentage': float(f'{null_pct:.2f}') # Format instead of round
12    })
13 null_df = pd.DataFrame(null_analysis).sort_values
14 print(null_df.to_string(index=False))
> [See performance (28)]
```

The notebook also contains a table titled "Tables 12 Performance 12" with the following data:

vict_sex	144644	14.39
vict_descent	144656	14.39
premis_desc	588	0.06
dr_no	0	0.00
lat	0	0.00
location	0	0.00
crm_cd_1	11	0.00
status_desc	0	0.00
premis_cd	16	0.00
date_rptd	0	0.00
vict_age	0	0.00
crm_cd_desc	0	0.00



Data Types Analysis

Used Spark's printSchema() to display column names, data types, and nullable status.

Key Observations:

- Date fields (date_rptd, date_occ) - String format requiring conversion
 - Time field (time_occ) - Integer in military time format (0-2359)
 - Geographic fields (lat, lon) - Double precision required for mapping
 - Demographic fields (vict_age, vict_sex, vict_descent) - Mixed types requiring standardization

NULL Value Analysis

- Iterates through all 28 columns
 - Counts NULL values per column
 - Calculates NULL percentage relative to total rows
 - Sorts by NULL percentage (descending)

Date Quality Issues

Validation Checks:

1. NULL Date Detection: Missing date_rptd or date_occ values
 2. Future Date Detection: Crimes reported/occurred in the future (impossible - data error)
 3. Logical Timeline Validation: date_occurred \leq date_reported (crime can't be reported before it happened)

```

1 print(f"\n1.6 TIME QUALITY ISSUES")
2 print(f"{'='*80}")
3
4 # Time statistics
5 null_time = df.filter(col('time_occ').isNull()).count()
6 invalid_time = df.filter((col('time_occ') < 0) | (col('time_occ') > 2359)).count()
7
8 print(f"Time Issues:")
9 print(f" Null Times: {null_time:,} ({null_time:,})")
10 print(f" Invalid Times (<0 or >2359): {invalid_time:,}")
11
12 # Time range
13 time_stats = df.select(min('time_occ'), max('time_occ')).collect()[0]
14 print(f"\nTime Range: {time_stats[0]} to {time_stats[1]}")
> See performance (3)

1.6 TIME QUALITY ISSUES
=====
Time Issues:
Null Times: 0
Invalid Times (<0 or >2359): 0

Time Range: 1 to 2359

```

Tables 12 Performance 12

Nov 24, 2024, 08:23 2m 25s Full refresh all Query performance

Time Quality Issues

Validation Rules:

- Format: Military time (24-hour format without colon)
- Valid Range: 0000 to 2359
- NULL Times: Missing time_occ values
- Invalid Times: Values < 0 or > 2359 (impossible times)

Geographic Data Quality Issues

LA Geographic Boundaries:

- Latitude Range: 33.7°N to 34.8°N
- Longitude Range: -118.7°W to -118.0°W

Quality Issues Detected:

1. NULL Coordinates: Missing latitude/longitude values
2. Zero Coordinates (0, 0): Placeholder values indicating missing location data (points to Gulf of Guinea, Africa)

```

silver_to_stage_Scd2
# Coordinate quality issues
1 print("\n1.7 COORDINATE QUALITY ISSUES")
2 print("\n1.80")
3
4 # Coordinate issues
5 null_lat = df.filter(col('lat').isNull()).count()
6 null_lon = df.filter(col('lon').isNull()).count()
7 zero_lat = df.filter(col('lat') == 0).count()
8 zero_lon = df.filter(col('lon') == 0).count()
9 both_zero = df.filter((col('lat') == 0) & (col('lon') == 0)).count()
10 both_null = df.filter(col('lat').isNull() & col('lon').isNull()).count()
11
12 # Valid coordinates
13 valid_coords = df.filter(
14     (col('lat').isNotNull() & (col('lon').isNotNull()) &
15      (col('lat') != 0) & (col('lon') != 0))
16 ).count()
17
18 # LA bounds check (Lat: 33.7-34.8, Lon: -118.7 to -118.0)
19 outside_la = df.filter(
20     (col('lat').isNotNull() & (col('lon').isNotNull()) &
21      (col('lat') != 0) & (col('lon') != 0) &
22      ((col('lat') < 33.7) | (col('lat') > 34.8) | 
23       (col('lon') < -118.7) | (col('lon') > -118.0))
24 ).count()
25
26 print("Coordinate Issues:")
27 print(f" Null Latitude: {null_lat}/{total_rows=100:.2f}%")
28 print(f" Null Longitude: {null_lon}/{total_rows=100:.2f}%")

```

Victim Age Quality Issues

Valid Age Range: 1 to 120 years

Invalid Patterns Detected:

1. Negative Ages: Data entry errors (impossible values)
2. Zero Ages: Placeholder for unknown victim age
3. NULL Ages: Missing demographic data
4. Ages > 120: Unrealistic values (data errors or placeholder codes)

```

silver_to_stage_Scd2
# Age quality issues
1 print("\n1.8 AGE QUALITY ISSUES")
2 print("\n1.80")
3
4 # Age issues
5 null_age = df.filter(col('vict_age').isNull()).count()
6 negative_age = df.filter(col('vict_age') < 0).count()
7 zero_age = df.filter(col('vict_age') == 0).count()
8 over_120 = df.filter(col('vict_age') > 120).count()
9 valid_age = df.filter((col('vict_age') >= 1) & (col('vict_age') <= 120)).count()
10
11 print("Age Issues:")
12 print(f" Null Ages: {null_age}/{total_rows=100:.2f}%")
13 print(f" Negative Ages: {negative_age}/{total_rows=100:.2f}%")
14 print(f" Zero Ages: {zero_age}/{total_rows=100:.2f}%")
15 print(f" Ages > 120: {over_120}/{total_rows=100:.2f}%")
16 print(f" Valid Ages (1-120): {valid_age}/{total_rows=100:.2f}%")
17
18 # Age statistics
19 age_stats = df.filter((col('vict_age') >= 0) & (col('vict_age') <= 120)).select(
20     min('vict_age').alias('min'),
21     max('vict_age').alias('max'),
22     avg('vict_age').alias('mean')
23 ).collect()[0]

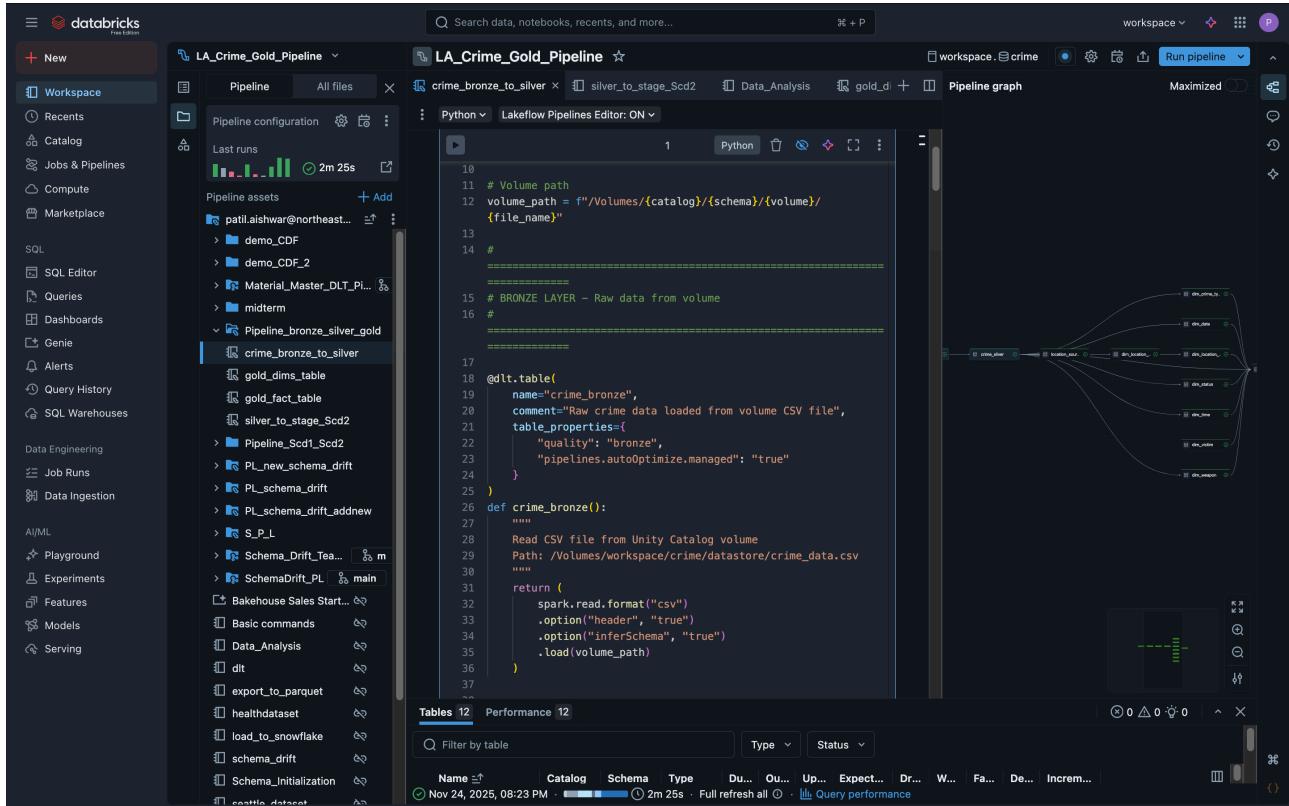
```

Bronze Layer: Ingesting raw CSV data from Unity Catalog Volume into DLT-managed table
Silver Layer: Data cleaning and transforming

Create a DLT-managed table that reads raw CSV data from Unity Catalog Volume with no transformations.

Source: /Volumes/workspace/crime/datastore/crime_data.csv

Target: workspace.crime.crime_bronze (DLT-managed table)



```
10 # Volume path
11 volume_path = f"/Volumes/{catalog}/{schema}/{volume}/
12 {file_name}"
13 #
14 =====
15 # BRONZE LAYER - Raw data from volume
16 #
17 =====
18 @dlt.table(
19     name="crime_bronze",
20     comment="Raw crime data loaded from volume CSV file",
21     table_properties={
22         "quality": "bronze",
23         "pipelines.autoOptimize.managed": "true"
24     }
25 )
26 def crime_bronze():
27     """
28     Read CSV file from Unity Catalog volume
29     Path: /Volumes/workspace/crime/datastore/crime_data.csv
30     """
31     return (
32         spark.read.format("csv")
33             .option("header", "true")
34             .option("inferSchema", "true")
35             .load(volume_path)
36     )
37 
```

Expected Output

- Records: 1,004,991 crime incidents (all records from CSV)
- Columns: 28 original attributes
- Quality: Raw data, no transformations applied
- Table Type: DLT-managed Delta table

Silver Layer - Data Cleaning & Transformation

Transform raw Bronze data into clean, analysis-ready Silver layer with:

- 100% record retention - No data loss
- Standardized values - Consistent encoding for unknown/missing data
- Geographic imputation - Area centroid assignment for missing coordinates
- Quality flags - Enable flexible filtering in Gold layer

Result: Clean, unified dataset ready for dimensional modeling with NO data loss.

The screenshot shows the Databricks Lakeflow Pipelines Editor interface. On the left, the sidebar lists various workspace items like Workspace, Catalog, and Compute. The main area has a title bar "LA_Crime_Gold_Pipeline" and a sub-bar "crime_bronze_to_silver". The central pane contains a Python code editor with the following script:

```
from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark.sql.window import Window

@dlt.table(
    name="crime_silver",
    comment="Cleanned LA crime data - ALL 1,004,991 records preserved with area centroid imputation"
)
@dlt.expect_all_or_drop(
    "valid_dr_no": "dr_no IS NOT NULL",
    "valid_date_rptd": "date_rptd IS NOT NULL",
    "valid_date_occ": "date_occ IS NOT NULL",
    "valid_time_occ": "time_occ IS NOT NULL",
    "valid_area": "area IS NOT NULL AND area_name IS NOT NULL",
    "valid_crime": "crm_cd IS NOT NULL AND crm_cd_desc IS NOT NULL"
)
def crime_silver():
    """
    Silver Layer: Cleanned data with 100% record retention

    Data Quality Rules:
    - Drop only if critical fields are NULL (should be 0 based on profiling)
    - Keep ALL records with cleaned values
    - Use -1 for unknown ages (not NULL)
    - Use 'X' for unknown sex/descent (not NULL)
    - Use area centroid for missing coordinates
    - Add helper flags for optional fields

    Expected Output: 1,004,991 records (100% retention)
    """
    """

```

To the right of the code editor is a "Pipeline graph" window showing the data flow between various tables and dimensions. Below the code editor is a "Tables" section with a table showing the status of different tables, including their last run time (Nov 24, 2025, 08:23 PM) and refresh frequency (2m 25s).

SCD Type 2 for Location

We felt that Location data can change over time in the LA crime dataset, so that's why we thought sad 2 was needed here

SCD Type 2 preserves this history by:

- Creating a new record when location attributes change (new lat/lon or area_name)
- Marking old records as is_current = False with an end_date

Gold Layer Dimensions: Loaded all the remaining six dimension apart from location

The screenshot shows the Databricks workspace interface. The left sidebar contains a navigation menu with sections like Workspace, Recents, Catalog, Jobs & Pipelines, Compute, Marketplace, SQL, SQL Editor, Queries, Dashboards, Genie, Alerts, Query History, and SQL Warehouses. Below these are Data Engineering, Job Runs, and Data Ingestion. Further down are A/ML, Playground, Experiments, Features, Models, and Serving.

The main workspace is titled "LA_Crime_Gold_Pipeline". It features a "Pipeline configuration" tab, which includes a "Last runs" section showing a recent run from Nov 24, 2028, at 08:23 PM. The "Pipeline assets" section lists several notebooks and pipelines, including "demo_CDF", "demo_CDF_2", "Material_Master_DLT_Pi...", "midterm", "Pipeline_bronze_silver_gold", "crime_bronze_to_silver", "gold_dims_table", "gold_fact_table", "silver_to_stage_Scd2", "Pipeline_Scd1_Scd2", "PL_new_schema_drift", "PL_schema_drift", "PL_schema_drift_addnew", "S_P_L", "Schema_Drift_Tea...", "SchemaDrift_PL", and "main".

The central area displays the Python code for the "dim_date" dimension:

```
# DIMENSION 1: dim_date
#
=====
# Extract all unique dates from silver layer
# Add year, quarter, month, day attributes
#
df = dlt.read("crime_silver")

# Get unique dates from date_occ
dates_df = df.select(
    to_date(col("date_occ")).alias("full_date")
).distinct()

return (
    dates_df
    .withColumn("date_key",
        regexp_replace(date_format(col
            ("full_date"), "yyyyMMdd"), "-", "").cast
            ("int"))
    .withColumn("year", year(col("full_date")))
    .withColumn("quarter", quarter(col("full_date")))
    .withColumn("month", month(col("full_date")))
    .withColumn("month_name", date_format(col
        ("full_date"), "MMMM"))
    .withColumn("day_of_week", dayofweek(col
```

To the right of the code editor is a "Pipeline graph" visualization, which is a directed acyclic graph (DAG) showing the flow of data between various stages and dimensions. The graph consists of nodes representing different parts of the pipeline, such as "crime_silver", "dlt.table", "dim_date", "gold_dims_table", "gold_fact_table", "midterm", "silver_to_stage_Scd2", and "Pipeline_bronze_silver_gold". Arrows indicate the dependencies and data flow between these components.

Dimension Tables

1. dim_date - Date Dimension

Purpose: Enable time-series analysis for crime trends (yearly, quarterly, monthly, daily patterns)

Source: Unique date_occ values from Silver layer

Grain: One row per unique date

Key Attributes:

- date_key (PK) - Integer format: YYYYMMDD (e.g., 20230615)
- full_date - Full date value
- year, quarter, month, month_name - Hierarchical attributes
- day_of_week (1=Sunday, 7=Saturday), day_name - Day analysis
- created_by, created_dt - Metadata

2. dim_time - Time of Day Dimension

Purpose: Analyze crime patterns by time of day and time periods

Source: Unique time_occ values from Silver layer

Grain: One row per unique military time value (0000-2359)

Key Attributes:

- time_key (PK) - Integer: military time (e.g., 1430 = 2:30 PM)
- time_occ - Original time value
- time_period - Classification: Morning (6:00-11:59), Afternoon (12:00-16:59), Evening (17:00-20:59), Night (21:00-5:59)
- created_by, created_dt - Metadata

3. dim_crime_type - Crime Classification Dimension

Purpose: Categorize crimes by type code and FBI classification

Source: Unique crm_cd and crm_cd_desc from Silver layer

Grain: One row per unique crime code

Key Attributes:

- crime_type_key (PK) - Same as crm_cd (crime code)
- crm_cd - Crime code (e.g., 624 = Battery - Simple Assault)
- crm_cd_desc - Crime description
- part_1_2 - FBI Uniform Crime Reporting classification:
 - 1 = Part I crimes (serious: murder, rape, robbery, burglary, etc.)
 - 2 = Part II crimes (less serious: simple assault, vandalism, etc.)
- created_by, created_dt - Metadata

4. dim_weapon - Weapon Classification Dimension

Purpose: Track weapons used in crimes

Source: Unique weapon_used_cd_clean and weapon_desc_clean from Silver layer

Grain: One row per unique weapon code

Key Attributes:

- weapon_key (PK) - Same as weapon_used_cd
- weapon_used_cd - Weapon code (0 = no weapon, 400 = firearm, etc.)
- weapon_desc - Weapon description (cleaned with "NO WEAPON/UNKNOWN" for nulls)
- created_by, created_dt - Metadata

5. dim_victim - Victim Demographics Dimension

Purpose: Analyze crime patterns by victim age groups and sex

Source: Pre-generated combinations of age groups and sex categories

Grain: One row per unique age_group + sex combination

Key Attributes:

- victim_key (PK) - Sequential integer (1-18)

- age_group - Categorical: Unknown, 0-17 (Juvenile), 18-25 (Young Adult), 26-35 (Adult), 36-50 (Middle Age), 51+ (Senior)
- sex - M (Male), F (Female), X (Unknown)
- created_by, created_dt - Metadata

6. dim_status - Case Status Dimension

Purpose: Track investigation and arrest status

Source: Unique status_clean and status_desc from Silver layer

Grain: One row per unique status code

Key Attributes:

- status_key (PK) - Sequential integer (auto-generated)
- status_code - Status code (IC, AA, JA, JO, etc.)
- status_desc - Status description
 - IC = Investigation Continues (active case)
 - AA = Adult Arrest
 - JA = Juvenile Arrest
 - JO = Juvenile Other
- created_by, created_dt - Metadata

Gold Layer Fact Table

```

LA_Crime_Gold_Pipeline
Pipeline configuration
Last runs
Pipeline assets
gold_fact_table
Python
# =====#
# STEP 6: Select final fact table columns
# =====#
return (
    fact_with_status
    .withColumn("created_by", lit("SYSTEM"))
    .withColumn("created_dt", current_timestamp())
    .select(
        col("f.crime_incident_key"),
        col("f.date_key"),
        col("t.time_key"),
        col("l.location_key"),
        col("c.crime_type_key"),
        col("w.weapon_key"),
        col("v.victim_key"),
        col("s.status_key"),
        col("f.descent"),
        col("f.rpt_dist_no"),
        col("f.is_arrest_flag"),
        col("f.created_by"),
        col("f.created_dt")
    )
    .dropDuplicates(["crime_incident_key"]) # CRITICAL:
    Force one row per DR_NO
)
  
```

Tables 12 Performance 12

Name	Catalog	Schema	Type	Du...	Ou...	Up...	Expect...	Dr...	W...	Fa...	De...	Incre...
dim...	worksp...	crime	Material...	6s	1.9K	-	Not define	-	-	-	Full ref...	...
dim...	worksp...	crime	Material...	4s	21	-	Not define	-	-	-	Full ref...	...
Nov 24, 2025, 10:01 PM				2m 28s		Full refresh all		Query performance				

Fact Table Design

Grain: One row per crime incident (uniquely identified by DR_NO)

Fact Table Type

Transaction Fact Table - Records individual crime events at the most granular level

Fact Table Schema

Dimension Foreign Keys (7)

Column	Type	References
date_key	Integer	dim_date.date_key
time_key	Integer	dim_time.time_key
location_key	String	dim_location.location_key
crime_type_key	Integer	dim_crime_type.crime_type_key
weapon_key	Integer	dim_weapon.weapon_key
victim_key	Integer	dim_victim.victim_key
status_key	Integer	dim_status.status_key

Degenerate Dimensions (2)

Column	Type
crime_incident_key	String
descent	String
rpt_dist_no	Integer

Fact Measures (1)

Column	Type	Purpose
is_arrest_flag	Boolean	Whether crime resulted in arrest (AA or JA status)

Metadata (2)

Column	Type	Purpose
created_by	String	"SYSTEM" - audit trail
created_dt	Timestamp	When record was loaded

LA_Crime_Gold_Pipeline

Pipeline graph

```

graph LR
    crime_bronze[crime_bronze] --> crime_silver[crime_silver]
    crime_silver --> location_sour[location_sour]
    location_sour --> dim_crime_ty[dim_crime_ty]
    location_sour --> dim_date[dim_date]
    location_sour --> dim_location[dim_location_...]
    location_sour --> dim_status[dim_status]
    location_sour --> dim_time[dim_time]
    location_sour --> dim_victim[dim_victim]
    location_sour --> dim_weapon[dim_weapon]
    fact_crime[fact_crime] --> dim_crime_ty
    fact_crime --> dim_date
    fact_crime --> dim_location_
    fact_crime --> dim_status
    fact_crime --> dim_time
    fact_crime --> dim_victim
    fact_crime --> dim_weapon
  
```

Tables 12 Performance 12

Name	Catalog	Schema	Type	Du...	Ou...	Up...	Expect...	Dr...	W...	Fai...	De...	Incre...
dim_crime_ty	workspace	crime	Material...	6s	1.9K	-	Not define	-	-	-	-	Full ref
dim_date	workspace	crime	Material...	7s	140	-	Not define	-	-	-	-	Full ref
dim_location_...	workspace	crime	Material...	6s	1.9K	-	Not define	-	-	-	-	Full ref
dim_status	workspace	crime	Material...	4s	21	-	Not define	-	-	-	-	Full ref
dim_time	workspace	crime	Streamli...	22s	-	21	Not define	-	-	-	0	N/A
dim_victim	workspace	crime	Material...	8s	7	-	Not define	-	-	-	-	Full ref
dim_weapon	workspace	crime	Material...	6s	1.4K	-	Not define	-	-	-	-	Full ref
fact_crime	workspace	crime	Material...	4s	18	-	Not define	-	-	-	-	Full ref
location_sour	workspace	crime	Material...	7s	80	-	Not define	-	-	-	-	Full ref
gold_fact_table	workspace	crime	Material...	6s	1M	-	Not define	-	-	-	-	Full ref
gold_dims_table	workspace	crime	Material...	7s	78K	-	Not define	-	-	-	-	Full ref

Nov 24, 2025, 10:01PM · 2m 28s · Full refresh all · Query performance

LA_Crime_Gold_Pipeline

Pipeline graph

```

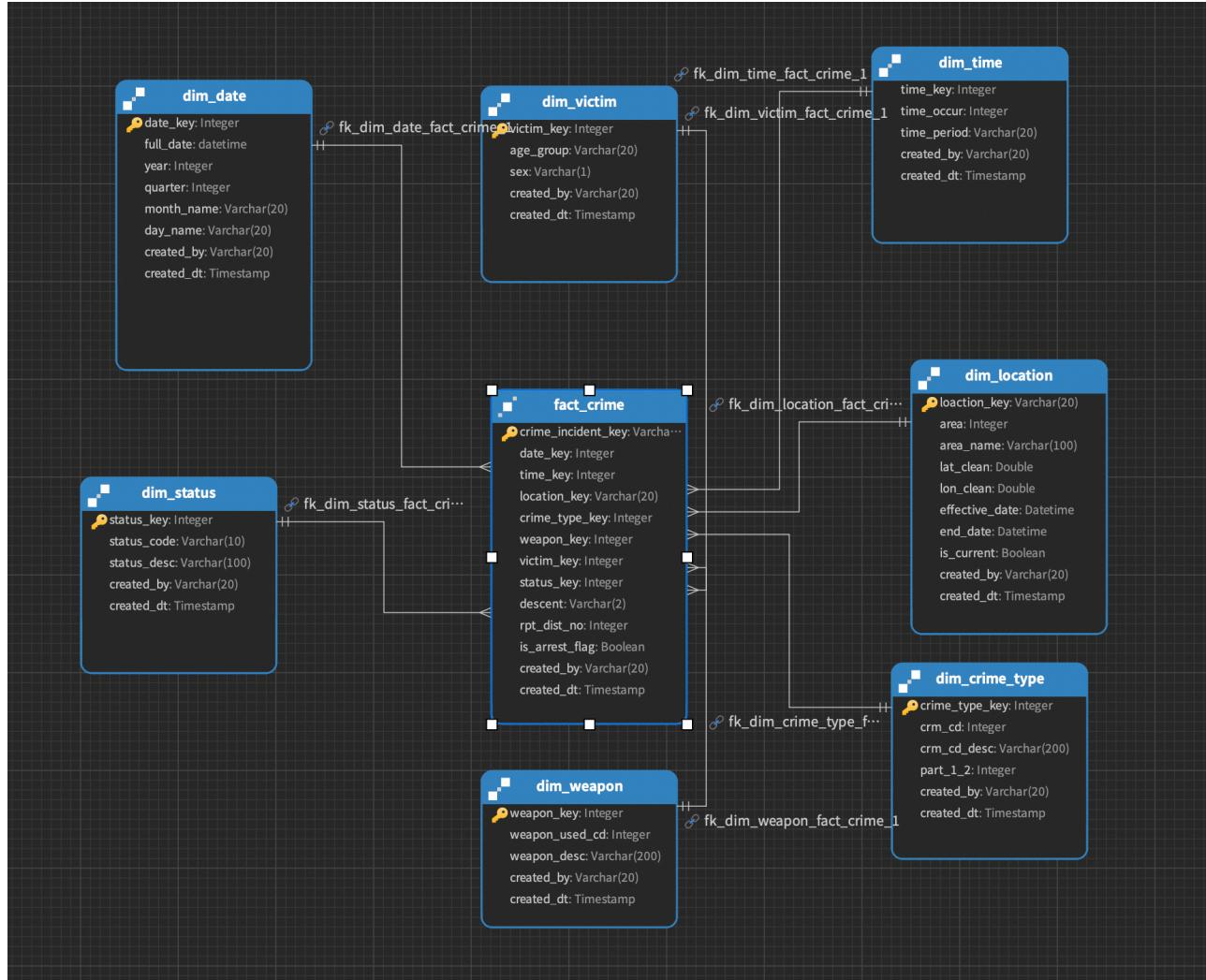
graph LR
    crime_bronze[crime_bronze] --> crime_silver[crime_silver]
    crime_silver --> location_sour[location_sour]
    location_sour --> dim_crime_ty[dim_crime_ty]
    location_sour --> dim_date[dim_date]
    location_sour --> dim_location[dim_location_...]
    location_sour --> dim_status[dim_status]
    location_sour --> dim_time[dim_time]
    location_sour --> dim_victim[dim_victim]
    location_sour --> dim_weapon[dim_weapon]
    fact_crime[fact_crime] --> dim_crime_ty
    fact_crime --> dim_date
    fact_crime --> dim_location_
    fact_crime --> dim_status
    fact_crime --> dim_time
    fact_crime --> dim_victim
    fact_crime --> dim_weapon
  
```

Tables 12 Performance 12

Name	Catalog	Schema	Type	Du...	Ou...	Up...	Expect...	Dr...	W...	Fai...	De...	Incre...
dim_crime_ty	workspace	crime	Material...	8s	1M	-	Not define	-	-	-	-	Full ref
dim_date	workspace	crime	Material...	10s	1M	-	6 met	0	0	0	-	Full ref
dim_location_...	workspace	crime	Material...	7s	140	-	Not define	-	-	-	-	Full ref
dim_status	workspace	crime	Material...	6s	1.9K	-	Not define	-	-	-	-	Full ref
dim_time	workspace	crime	Material...	4s	21	-	Not define	-	-	-	-	Full ref
dim_victim	workspace	crime	Streamli...	22s	-	21	Not define	-	-	-	0	N/A
dim_weapon	workspace	crime	Material...	8s	7	-	Not define	-	-	-	-	Full ref
fact_crime	workspace	crime	Material...	6s	1.4K	-	Not define	-	-	-	-	Full ref
location_sour	workspace	crime	Material...	4s	18	-	Not define	-	-	-	-	Full ref
gold_fact_table	workspace	crime	Material...	7s	80	-	Not define	-	-	-	-	Full ref
gold_dims_table	workspace	crime	Material...	6s	1M	-	Not define	-	-	-	-	Full ref

Nov 24, 2025, 10:01PM · 2m 28s · Full refresh all · Query performance

This is the revised dimension model that we created

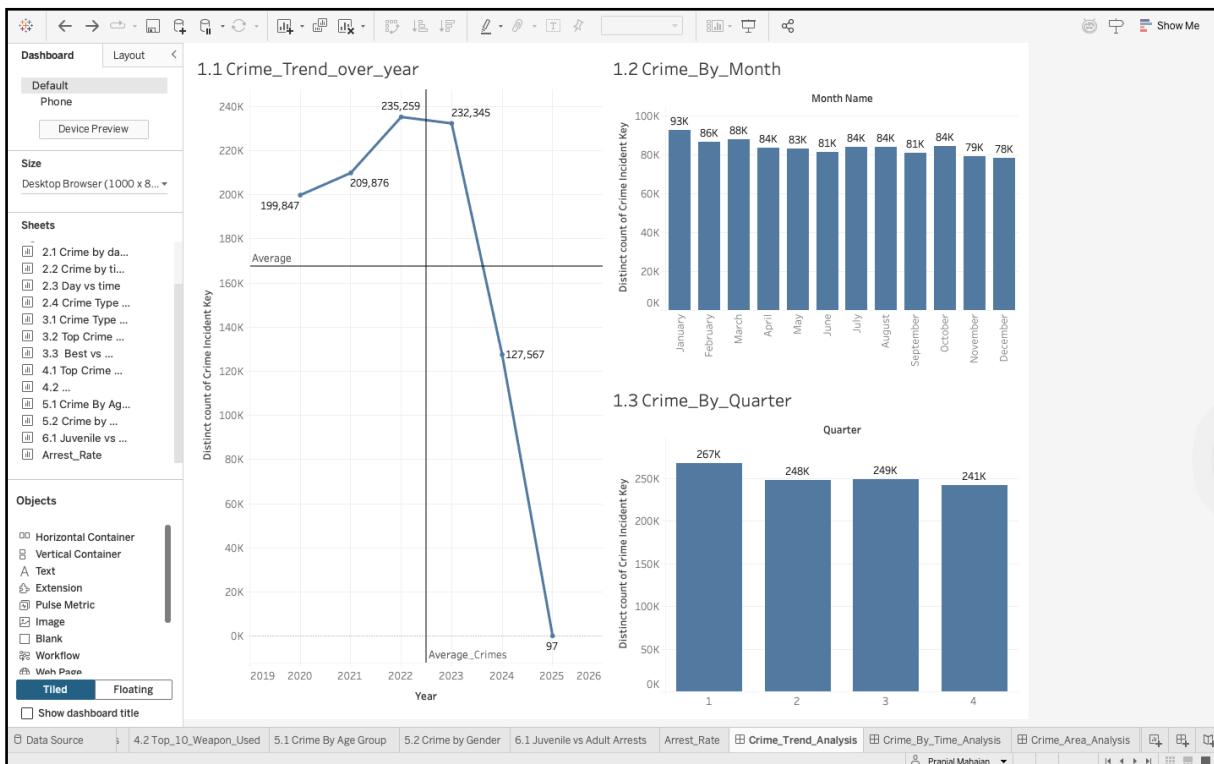


We established a connection between Tableau and Databricks, imported all dimension and fact tables, configured the dimensional relationships, and developed visualizations to address the following business requirements:

Business Questions:

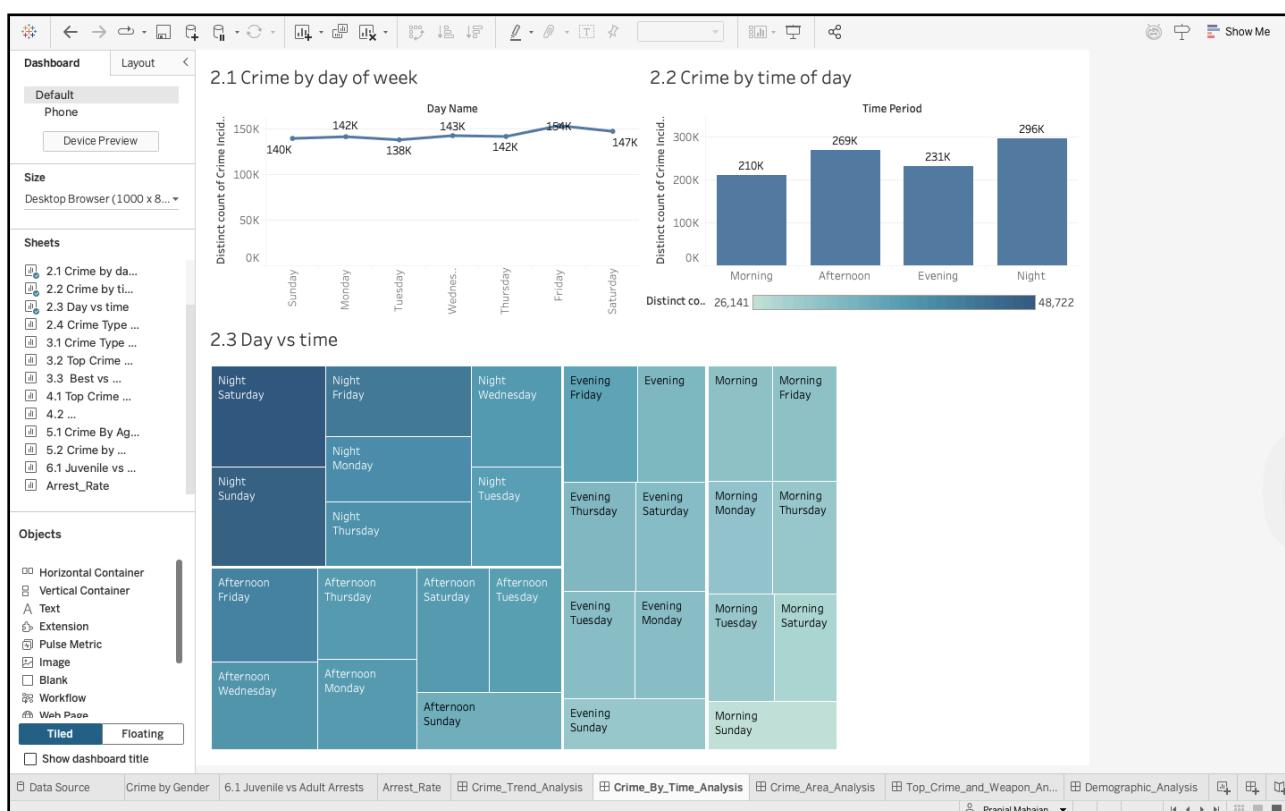
Crime Rates Over Time:

- What is the overall trend in crime rates over the years?
- How have crime rates changed on a monthly basis?
- How have crime rates changed on a quarterly basis?



Day Time and Week Factors:

- Is there a correlation between the day of the week and the number of reported crimes?
 - Do certain types of crimes tend to occur at specific times of the day?

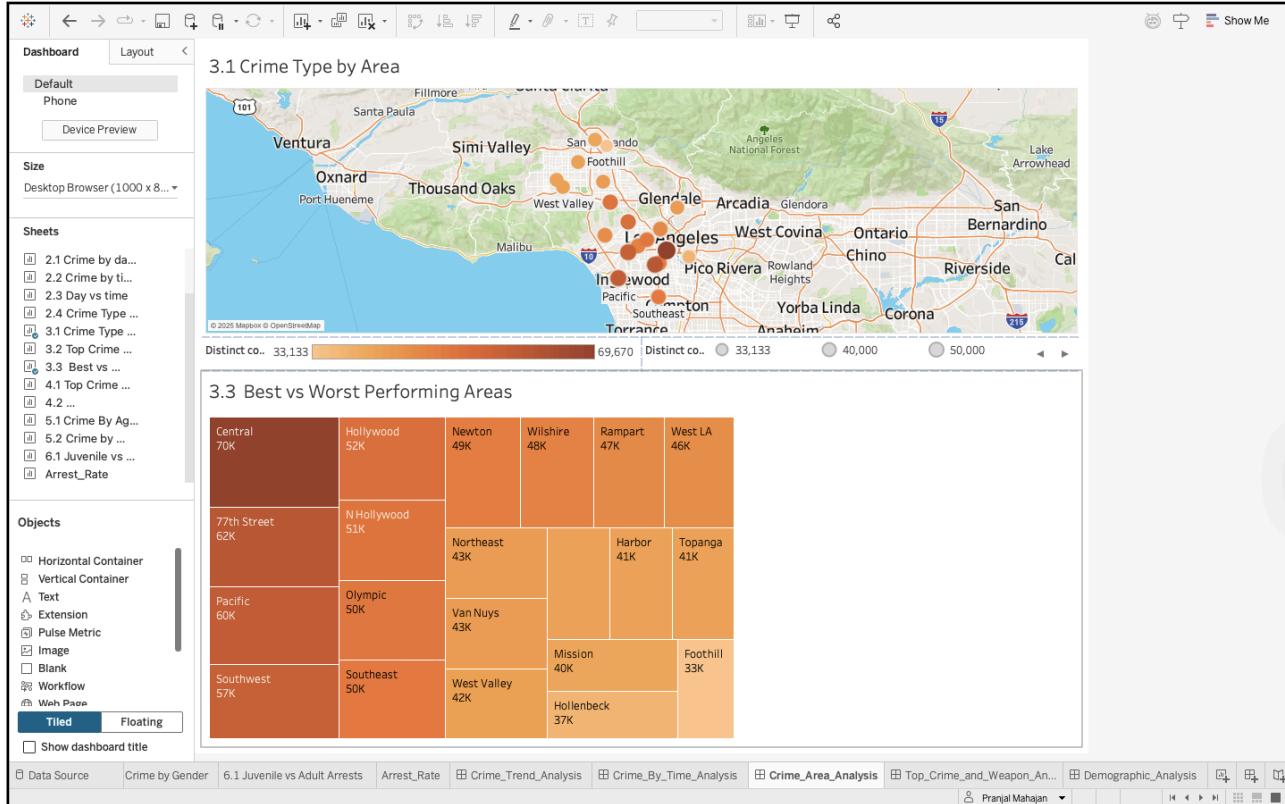


Crime by Location:

- Where are the high-crime areas in Los Angeles?

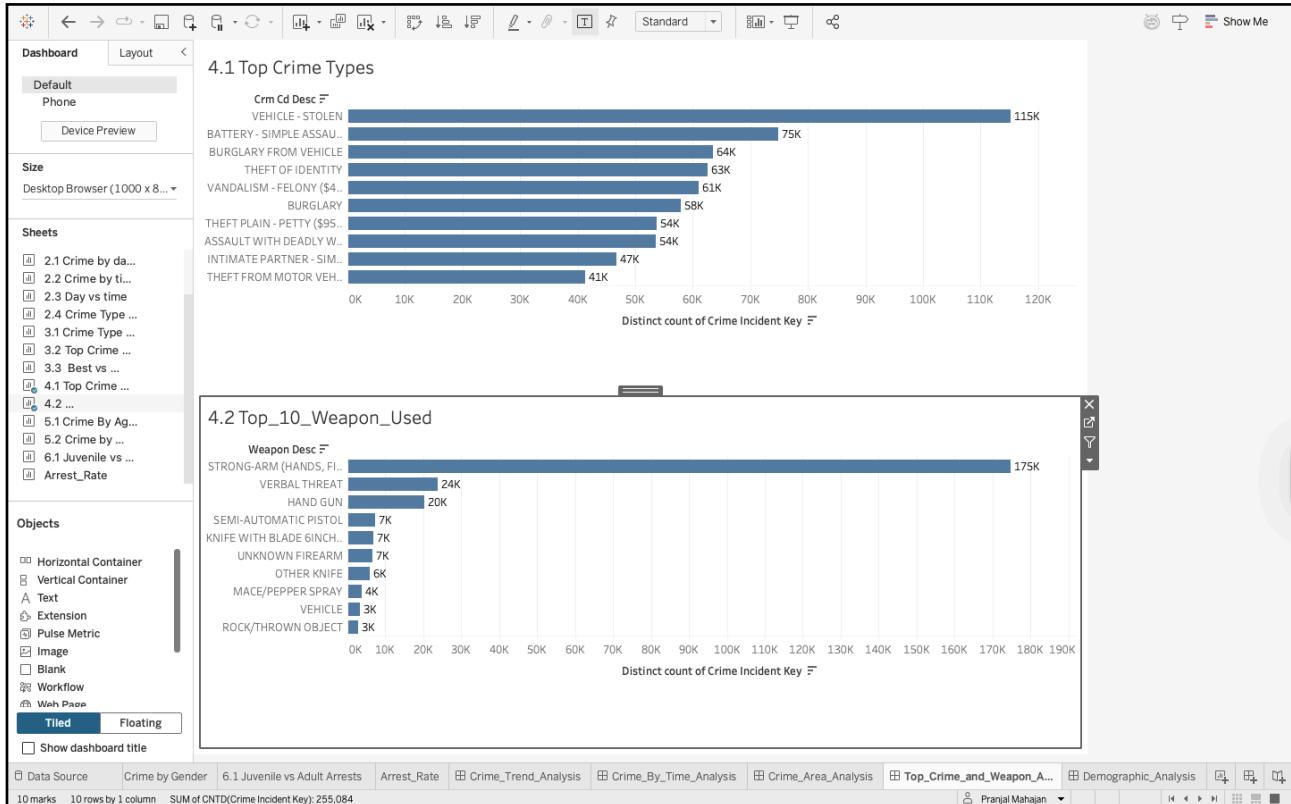
We want to know (your inferences on):

- By seeing the geo map, what are the crime hotspots?
- Can you identify areas where Los Angeles performs better or worse in terms of crime rates?



Types of Crime:

- What are the most commonly used weapons in reported crimes?



Demographic Analysis:

- Show in the visualization the age patterns in crime.
- Show in the visualization the gender-related patterns in crime.

Arrests Ratio:

- What percentage of reported crimes result in Juvenile, Adult arrests?

