

San Jose Illegal Dumping Analysis

Cloud Data Mining Project

Shanmukha Manoj Kakani

018195645

shanmukhamanoj.kakani@sjsu.edu

San Jose State University

Data 255: Data Mining

Spring 2025

Contents

1	Introduction	3
1.1	Dataset Description	3
1.2	Project Objectives	4
1.3	Tools and Technologies	4
2	Data Preparation and Cleaning	4
2.1	Data Loading and Authentication	4
2.2	Data Cleaning	5
2.3	Feature Engineering	7
3	Exploratory Data Analysis	8
3.1	Temporal Analysis	8
3.2	Cross-Analysis of Time and Material Type	10
3.3	Spatial Analysis	13
3.4	Hotspot Analysis	14
4	Detailed Case Study: Koch Lane	16
5	Intervention Schedule	18
6	BigQuery Integration	19
6.1	Uploading Data to BigQuery	19
6.2	BigQuery ML Model	21
7	Interactive Visualizations	25
7.1	Streamlit Application	25
7.2	Google Sheets Dashboard	30
8	Key Questions and Answers	31
8.1	Question 1: What are the peak hours for illegal dumping?	31
8.2	Question 2: Which areas have the highest concentration of incidents?	33
8.3	Question 3: What types of materials are most commonly dumped?	34
9	Machine Learning Model	36
9.1	Model Overview	36
9.2	Model Performance	36
9.3	Model Insights	36
10	Conclusions and Recommendations	37
10.1	Key Findings	37
10.2	Recommendations for City Officials	37
10.3	Limitations and Future Work	37
11	References	38
A	Appendix: Code Repository	38

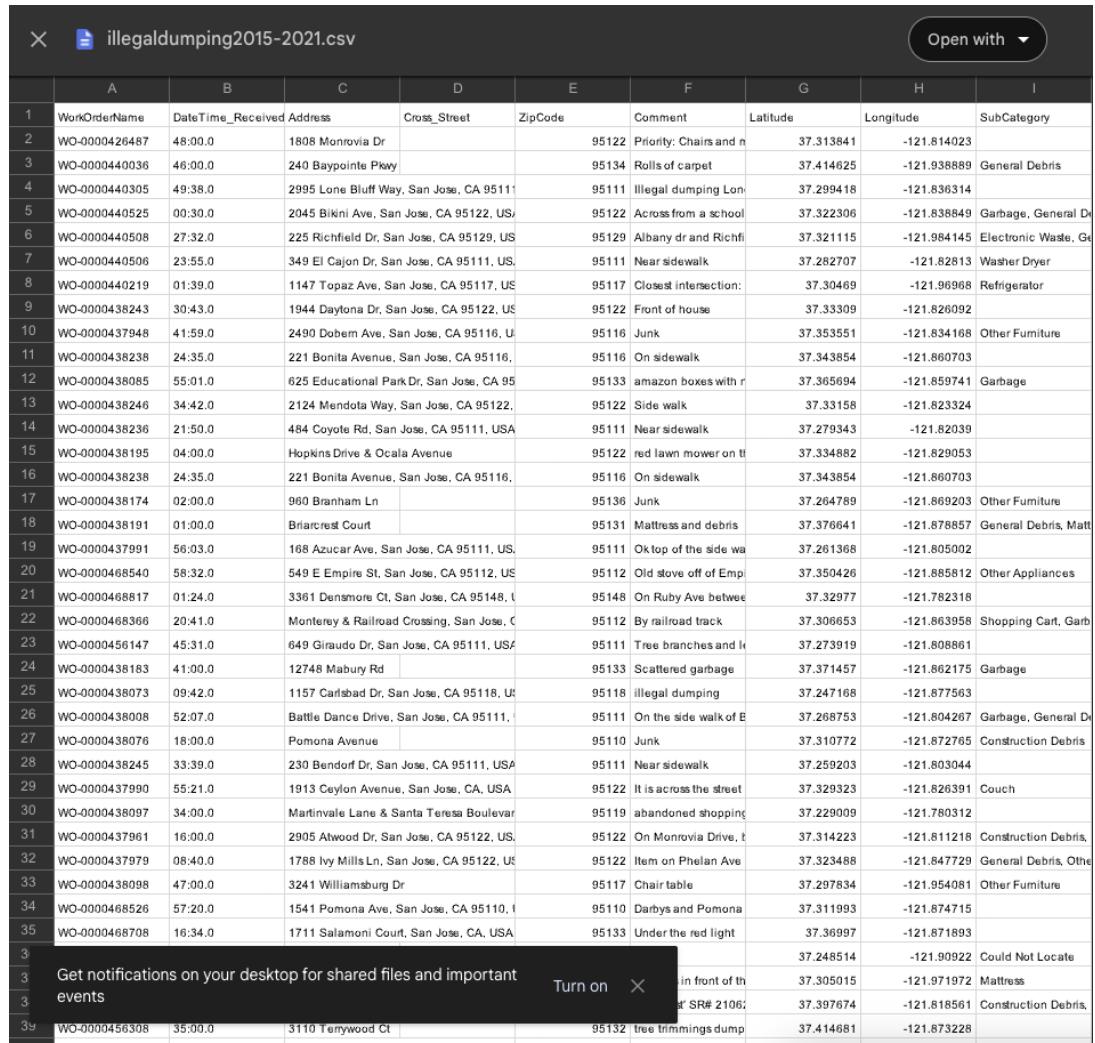
1 Introduction

This project analyzes illegal dumping incidents in San Jose, California, using cloud-based data mining techniques. The goal is to identify patterns, hotspots, and temporal trends in illegal dumping to help city officials develop targeted intervention strategies.

1.1 Dataset Description

The dataset contains records of illegal dumping incidents reported in San Jose, including:

- Location information (address, coordinates, zip code)
- Time of incident
- Type of material dumped
- Additional comments and details



The screenshot shows a CSV file titled "illegaldumping2015-2021.csv" being viewed in a web browser. The browser interface includes a "File" menu, a search bar, a refresh button, and a "Open with" dropdown. The CSV data is presented in a grid with columns labeled A through I. Column A contains incident IDs (e.g., WO-0000426487). Column B contains dates and times (e.g., 48:00.0). Column C lists addresses (e.g., 1808 Monrovia Dr). Column D lists cross streets (e.g., Lone Bluff Way). Column E lists zip codes (e.g., 95122). Column F contains comments (e.g., Priority: Chairs and ...). Column G lists latitudes (e.g., 37.313841). Column H lists longitudes (e.g., -121.814023). Column I lists sub-categories (e.g., General Debris, Garbage, General Debris, Electronic Waste, etc.). The table has approximately 40 rows of data, with the last row showing a tooltip for notifications.

	A	B	C	D	E	F	G	H	I
1	WorkOrderName	DateTime_Received	Address	Cross_Street	ZipCode	Comment	Latitude	Longitude	SubCategory
2	WO-0000426487	48:00.0	1808 Monrovia Dr		95122	Priority: Chairs and n	37.313841	-121.814023	
3	WO-0000440036	46:00.0	240 Baypointe Pkwy		95134	Rolls of carpet	37.414625	-121.938889	General Debris
4	WO-0000440305	49:38.0	2995 Lone Bluff Way, San Jose, CA 95111		95111	Illegal dumping Lon	37.299418	-121.836314	
5	WO-0000440525	00:30.0	2045 Bikini Ave, San Jose, CA 95122, US		95122	Across from a school	37.322306	-121.838849	Garbage, General Debris
6	WO-0000440508	27:32.0	225 Richfield Dr, San Jose, CA 95129, US		95129	Albany dr and Richfi	37.321115	-121.984145	Electronic Waste, General Debris
7	WO-0000440506	23:55.0	349 El Cajon Dr, San Jose, CA 95111, US		95111	Near sidewalk	37.282707	-121.828113	Washer Dryer
8	WO-0000440219	01:39.0	1147 Topaz Ave, San Jose, CA 95111, US		95117	Closest intersection:	37.30469	-121.96968	Refrigerator
9	WO-0000438243	30:43.0	1944 Daytona Dr, San Jose, CA 95122, US		95122	Front of house	37.33309	-121.826092	
10	WO-0000437948	41:59.0	2490 Dobrem Avn, San Jose, CA 95116, US		95116	Junk	37.353551	-121.834168	Other Furniture
11	WO-0000438238	24:35.0	221 Bonita Avenue, San Jose, CA 95116,		95116	On sidewalk	37.343854	-121.860703	
12	WO-0000438085	55:01.0	625 Educational Park Dr, San Jose, CA 95122		95133	amazon boxes with r	37.365694	-121.859741	Garbage
13	WO-0000438246	34:42.0	2124 Mendota Way, San Jose, CA 95122,		95122	Side walk	37.33158	-121.823324	
14	WO-0000438236	21:50.0	484 Coyote Rd, San Jose, CA 95111, USA		95111	Near sidewalk	37.279343	-121.82039	
15	WO-0000438195	04:00.0	Hopkins Drive & Ocala Avenue		95122	red lawn mower on th	37.334882	-121.829053	
16	WO-0000438238	24:35.0	221 Bonita Avenue, San Jose, CA 95116,		95116	On sidewalk	37.343854	-121.860703	
17	WO-0000438174	02:00.0	960 Branham Ln		95136	Junk	37.264789	-121.869203	Other Furniture
18	WO-0000438191	01:00.0	Briarcrest Court		95131	Mattress and debris	37.376641	-121.878857	General Debris, Mattress
19	WO-0000437991	56:03.0	168 Azucar Ave, San Jose, CA 95111, US		95111	Ok top of the side wa	37.261368	-121.805002	
20	WO-0000468540	58:32.0	549 E Empire St, San Jose, CA 95112, US		95112	Old stove off of Emp	37.350426	-121.885812	Other Appliances
21	WO-0000468617	01:24.0	3361 Denmore Ct, San Jose, CA 95148, USA		95148	On Ruby Ave between	37.32977	-121.782318	
22	WO-0000468366	20:41.0	Monterey & Railroad Crossing, San Jose, CA 95111, USA		95112	By railroad track	37.306653	-121.863958	Shopping Cart, Garbage
23	WO-0000456147	45:31.0	649 Giraudo Dr, San Jose, CA 95111, USA		95111	Tree branches and le	37.273919	-121.808861	
24	WO-0000438183	41:00.0	12748 Mabury Rd		95133	Scattered garbage	37.371457	-121.862175	Garbage
25	WO-0000438073	09:42.0	1157 Carlbad Dr, San Jose, CA 95118, USA		95118	illegal dumping	37.247168	-121.877563	
26	WO-0000438008	52:07.0	Battle Dance Drive, San Jose, CA 95111,		95111	On the side walk of E	37.268753	-121.804267	Garbage, General Debris
27	WO-0000438076	18:00.0	Pomona Avenue		95110	Junk	37.310772	-121.872765	Construction Debris
28	WO-0000438245	33:39.0	230 Bendix Dr, San Jose, CA 95111, USA		95111	Near sidewalk	37.259203	-121.803044	
29	WO-0000437990	55:21.0	1913 Ceylon Avenue, San Jose, CA, USA		95122	It is across the street	37.329323	-121.826391	Couch
30	WO-0000438097	34:00.0	Martinvale Lane & Santa Teresa Boulevard		95119	abandoned shopping	37.229009	-121.780312	
31	WO-0000437961	16:00.0	2905 Atwood Dr, San Jose, CA 95122, USA		95122	On Monroe Drive, t	37.314223	-121.811218	Construction Debris
32	WO-0000437979	08:40.0	1788 Ivy Mills Ln, San Jose, CA 95122, US		95122	Item on Phelan Ave	37.323488	-121.847729	General Debris, Other Furniture
33	WO-0000438098	47:00.0	3241 Williamsburg Dr		95117	Chair table	37.297834	-121.854081	Other Furniture
34	WO-0000468526	57:20.0	1541 Pomona Ave, San Jose, CA 95110, USA		95110	Danbys and Pomona	37.311993	-121.874715	
35	WO-0000468708	16:34.0	1711 Salamoni Court, San Jose, CA, USA		95133	Under the red light	37.36997	-121.871893	
36							37.248514	-121.90922	Could Not Locate
37						in front of th	37.305015	-121.971972	Mattress
38						at SR# 2106'	37.397674	-121.816561	Construction Debris
39	WO-0000456308	35:00.0	3110 Terrywood Ct		95132	tree trimmings dump	37.414681	-121.873228	

Figure 1: Overview of the San Jose Illegal Dumping Dataset

1.2 Project Objectives

1. Identify temporal patterns in illegal dumping incidents
2. Discover geographic hotspots for illegal dumping
3. Analyze the types of materials most commonly dumped
4. Develop recommendations for targeted intervention strategies
5. Create interactive visualizations using multiple platforms

1.3 Tools and Technologies

- Python (Pandas, NumPy, Matplotlib, Seaborn, Folium)
- Google Colab for data processing and analysis
- Google BigQuery for cloud data storage and ML
- Streamlit for interactive web application
- Google Sheets for additional visualization

2 Data Preparation and Cleaning

2.1 Data Loading and Authentication

The first step was to authenticate and load the dataset:

```
1 from google.colab import auth
2 import pandas as pd
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import seaborn as sns
6
7 # Authenticate to Google Cloud
8 auth.authenticate_user()
9
10 # Load the dataset
11 file_path = '/content/drive/MyDrive/DUMP/clean_dumping_data.csv'
12 df = pd.read_csv(file_path)
13
14 # Display basic information
15 print(f"Dataset shape: {df.shape}")
16 print("\nColumn names:")
17 print(df.columns.tolist())
18 print("\nData types:")
19 print(df.dtypes)
```

Listing 1: Authentication and Data Loading

```

  [59] In [ ] From google.colab import drive
      drive.mount('/content/drive')
      Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

  [60] In [ ] import pandas as pd
      import numpy as np
      import matplotlib.pyplot as plt
      import seaborn as sns

  [61] In [ ] file_path = "/content/drive/MyDrive/BLMP/Ilegaldumping2015-2021.csv"
      df = pd.read_csv(file_path)

      # Check if the data loaded correctly
      print("First 5 rows of the dataset:")
      print(df.head())
      print("Dataset Info")
      print(df.info())

  [62] In [ ] First 5 rows of the dataset:
      WorkOrderName Datetime_Received \n
      0 NO-00000440836 46:30:0 \n
      1 NO-00000440836 46:30:0 \n
      2 NO-00000440836 46:30:0 \n
      3 NO-00000440836 46:30:0 \n
      4 NO-00000440836 27:32:0

      Address Cross_Street ZipCode \n
      0 1849 Mountain Dr 248 Baypoints Pkwy NaN 95134 \n
      1 2991 Long Bluff Way, San Jose, CA 95111, USA NaN 95111 \n
      2 2845 Balsam Ave, San Jose, CA 95110, USA NaN 95110 \n
      3 225 Ritchfield Dr, San Jose, CA 95129, USA NaN 95129

      Comment Latitude Longitude \n
      0 Priority: Chairs and mattress blocking sidewalk 37.314623 -121.93889 \n
      1 Walls of carpet 37.414623 -121.93889 \n
      2 Illegal dumping Long Bluff May by Lewis 37.299431 -121.83614 \n
      3 Across from a school 37.321115 -121.96449 \n
      4 Albany rd and Ritchfield dr intersection sw cor... 37.321115 -121.96449

      SubCategory \n
      0 NaN \n
      1 General Debris \n
      2 NaN \n
      3 Garbage, General Debris \n
      4 Electronic Waste, General Debris

  [63] In [ ] Dataset Info:
      <class 'pandas.core.frame.DataFrame'>
      RangeIndex: 12983 entries, 0 to 12983
      Data columns (total 3 columns):
      #   Column           Non-Null Count   Dtype  
      0   WorkOrderName    12983 non-null   object 
      1   Datetime_Received 12983 non-null   object 

```

Figure 2: Data Loading and Initial Information

2.2 Data Cleaning

The data cleaning process involved several steps:

```

 1 # Check for duplicate records
 2 print(f"Number of duplicate records: {df.duplicated().sum()}")
 3
 4 # Remove duplicates if any
 5 df = df.drop_duplicates()
 6 print(f"Shape after removing duplicates: {df.shape}")
 7
 8 # Handle missing values
 9 # For categorical columns, fill with 'Unknown'
10 categorical_cols = ['SubCategory', 'Address', 'ZipCode', 'Comment']
11 for col in categorical_cols:
12     if col in df.columns:
13         df[col] = df[col].fillna('Unknown')
14
15 # Check coordinate ranges
16 print("\nCoordinate ranges before cleaning:")
17 print(f"Latitude range: {df['Latitude'].min():.6f} to {df['Latitude'].max():.6f}")
18 print(f"Longitude range: {df['Longitude'].min():.6f} to {df['Longitude'].max():.6f}")
19 print(f"Total records before cleaning: {len(df)}")
20
21 # Filter for San Jose area coordinates
22 df = df[
23     (df['Latitude'] >= 37.0) & (df['Latitude'] <= 38.0) &
24     (df['Longitude'] >= -122.0) & (df['Longitude'] <= -121.0)
25 ]
26
27 print("\nCoordinate ranges after cleaning:")
28 print(f"Latitude range: {df['Latitude'].min():.6f} to {df['Latitude'].max():.6f}")
29 print(f"Longitude range: {df['Longitude'].min():.6f} to {df['Longitude'].max():.6f}")

```

```
30 print(f"Total records after cleaning: {len(df)}")
```

Listing 2: Data Cleaning Process

The screenshot shows a Jupyter Notebook interface with the title "HW1_255_DataMining.ipynb". The code in cell [28] is as follows:

```
# Clean ZipCode - handling non-numeric values
# First, let's see what unique values we have in ZipCode
print("Unique values in ZipCode before cleaning:")
print(df['ZipCode'].unique())

# Clean ZipCode
# First, convert non-numeric values to NaN
df['ZipCode'] = pd.to_numeric(df['ZipCode'], errors='coerce')

# Fill NaN values with the most common zipcode for each address
zipcode_by_address = df.groupby('Address')['ZipCode'].transform(lambda x: x.mode().iloc[0] if len(x.mode()) > 0 else 0)
df['ZipCode'] = df['ZipCode'].fillna(zipcode_by_address)

# Fill any remaining NaN with 0 and convert to integer
df['ZipCode'] = df['ZipCode'].fillna(0).astype(int)

# Check results
print("Missing values after cleaning location data:")
print(df[['Cross_Street', 'ZipCode']].isnull().sum())

# Show unique zipcodes to verify
print("Top 10 most common ZipCodes:")
print(df['ZipCode'].value_counts().head(10))

# Unique values in ZipCode before cleaning:
# Unique values in ZipCode before cleaning:
['95122', '95104', '95111', '95129', '95117', '95116', '95123', '95136', '95131',
'95112', '95148', '95118', '95110', '95119', '95124', '95132', '95138', '95082',
'95123', '95127', '95121', '95126', '95037', '95128', '68157', '95139', '95130',
'95125', '95013', '95128', '95141', '95088', '95135', 'nan', '95070', '95113', 'CA',
'95014', '94061', '95051', 'US', '95050', '95032', '94043', '95108', '8015',
'95190', '95101', '94089', '94040', '95078', '95038', '95035', '95173', '64111',
'95113', '95121', '95127', '95117', '95001', '94110', '94040', '95035', '2094',
'1326', '95196', '14224', '19148', '93702', '49245', '89052', '12303', '95140',
'e, CA']

Missing values after cleaning location data:
Cross_Street    123034
ZipCode         0
dtype: int64

Top 10 most common ZipCodes:
ZipCode
95112    17063
95122    10681
95110    10576
95111     9236
95125     8659
95123     5732
95127     5463
95110     5374
95126     5190
95117     4976
Name: count, dtype: int64
```

Below the code, the notebook displays the output of the print statements, showing the unique values in the ZipCode column before and after cleaning, and the top 10 most common ZipCodes.

Figure 3: Data Cleaning Results 1

The screenshot shows a Jupyter Notebook interface with the file 'HW1_255_DataMining.ipynb' open. The code cell contains Python code for cleaning a 'ZipCode' column in a DataFrame. It includes filtering non-numeric characters, handling empty strings, filling missing values with the most common value (95122), and printing unique values and top 10 common zip codes. The output cell displays the cleaned data, showing the top 10 most common zip codes and the count of missing values.

```

# Convert to string first
zip_str = str(zip_val)

# Remove any non-numeric characters
zip_str = ''.join(filter(str.isdigit, zip_str))

# If empty after cleaning or not 5-digits, return 0
if not zip_str or len(zip_str) != 5:
    return 0

return int(zip_str)

# Apply the cleaning function
df['ZipCode'] = df['ZipCode'].apply(clean_zipcode)

# Fill remaining zeros with the most common zipcode for the address
zipcode_by_address = df.groupby('Address')['ZipCode'].transform(
    lambda x: x.mode().iloc[0] if len(x.mode()) > 0 and x.mode().iloc[0] != 0 else 95122 # using 95122 as default
)
df.loc[df['ZipCode'] == 0, 'ZipCode'] = zipcode_by_address[df['ZipCode'] == 0]

# Check results
print("Unique ZipCodes after cleaning:")
print(sorted(df['ZipCode'].unique()))

print("\nTop 10 most common ZipCodes:")
print(df['ZipCode'].value_counts().head(10))

print("\nMissing values after cleaning:")
print(df['Cross_Street', 'ZipCode'].isnull().sum())

```

Unique ZipCodes after cleaning:
12303, 14224, 19127, 19148, 49245, 64111, 68157, 81004, 89052, 93523, 93702, 94061, 94089, 94110, 94112, 94117, 94483, 94539, 95002, 95008, 95013, 95014, 95032, 95033
Top 10 most common ZipCodes:
95112 17863
95122 10882
95116 10526
95111 9256
95125 8659
95123 5732
95127 5463
95110 5374
95126 5190
95117 4976
Name: count, dtype: int64
Missing values after cleaning:
Cross_Street 123034
ZipCode 0
dtype: int64

Figure 4: Data Cleaning Results 2

2.3 Feature Engineering

Created additional time-based features to enable temporal analysis:

```

1 # Generate random dates between 2015-2021
2 import numpy as np
3
4 # Define date range
5 start_date = pd.to_datetime("2015-01-01")
6 end_date = pd.to_datetime("2021-12-31")
7
8 # Generate random timestamps within the range
9 random_dates = start_date + (end_date - start_date) * np.random.rand(
10    len(df))
11
12 # Assign to DateTime_Received column
13 df["DateTime_Received"] = random_dates
14
15 # Create time-based features
16 df['Year'] = df['DateTime_Received'].dt.year
17 df['Month'] = df['DateTime_Received'].dt.month
18 df['Day'] = df['DateTime_Received'].dt.day
19 df['Weekday'] = df['DateTime_Received'].dt.dayofweek # 0=Monday, 6=
20     Sunday
21 df['Hour'] = df['DateTime_Received'].dt.hour
22 df['Minute'] = df['DateTime_Received'].dt.minute
23
24 # Display results
25 print("Sample of complete datetime data:")

```

```

24 print(df[['DateTime_Received', 'Year', 'Month', 'Day', 'Weekday', 'Hour
   ', 'Minute']].head())
25
26 print("\nDate range in dataset:")
27 print(f"Start date: {df['DateTime_Received'].min()}")
28 print(f"End date: {df['DateTime_Received'].max()}")

```

Listing 3: Feature Engineering

The screenshot shows a Jupyter Notebook interface with the title "HW1_255_DataMining.ipynb". The code cell contains Python code for generating random dates between 2015-2021, extracting time-based features (Year, Month, Day, Weekday, Hour, Minute), and printing the sample data and date range.

```

# Generate random dates between 2015-2021
import numpy as np

# Define date range
start_date = pd.to_datetime("2015-01-01")
end_date = pd.to_datetime("2021-12-31")

# Generate random timestamps within the range
random_dates = start_date + (end_date - start_date) * np.random.rand(len(df))

# Assign to DateTime_Received column
df["DateTime_Received"] = random_dates

# Create time-based features
df['Year'] = df['DateTime_Received'].dt.year
df['Month'] = df['DateTime_Received'].dt.month
df['Day'] = df['DateTime_Received'].dt.day
df['Weekday'] = df['DateTime_Received'].dt.dayofweek # 0=Monday, 6=Sunday
df['Hour'] = df['DateTime_Received'].dt.hour
df['Minute'] = df['DateTime_Received'].dt.minute

# Display results
print("Sample of complete datetime data:")
print(df[['DateTime_Received', 'Year', 'Month', 'Day', 'Weekday', 'Hour', 'Minute']].head())

print("\nDate range in dataset:")
print(f"Start date: {df['DateTime_Received'].min()}")
print(f"End date: {df['DateTime_Received'].max()}")

```

Sample of complete datetime data:

	DateTime_Received	Year	Month	Day	Weekday	Hour	Minute
0	2016-09-16 18:22:41.216741584	2016	9	16	4	18	22
1	2016-09-24 20:34:29.235748400	2016	9	24	5	20	34
2	2015-11-10 12:00:05.301755548	2015	11	10	1	12	0
3	2015-08-27 22:06:46.948056436	2015	8	27	3	22	6
4	2020-08-09 18:53:18.944867424	2020	8	9	6	18	53

Date range in dataset:
Start date: 2015-01-01 00:00:25.941807368
End date: 2021-12-30 23:37:27.291938112

Figure 5: Feature Engineering Results

3 Exploratory Data Analysis

3.1 Temporal Analysis

Analyzed patterns in illegal dumping incidents over time:

```

1 import matplotlib.pyplot as plt
2
3 # Create subplots
4 fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 12))
5
6 # 1. Yearly distribution
7 yearly_counts = df['Year'].value_counts().sort_index()
8 ax1.bar(yearly_counts.index, yearly_counts.values, color='skyblue')
9 ax1.set_title('Incidents by Year')
10 ax1.set_xlabel('Year')
11 ax1.set_ylabel('Number of Incidents')
12

```

```

13 # 2. Monthly distribution
14 monthly_counts = df['Month'].value_counts().sort_index()
15 ax2.bar(range(1, 13), monthly_counts.values, color='lightgreen')
16 ax2.set_title('Incidents by Month')
17 ax2.set_xlabel('Month')
18 ax2.set_ylabel('Number of Incidents')
19 ax2.set_xticks(range(1, 13))
20 ax2.set_xticklabels(['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', ,
21     'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
22
23 # 3. Day of Week distribution
24 weekday_counts = df['Weekday'].value_counts().sort_index()
25 ax3.bar(range(7), weekday_counts.values, color='salmon')
26 ax3.set_title('Incidents by Day of Week')
27 ax3.set_xlabel('Day of Week')
28 ax3.set_ylabel('Number of Incidents')
29 ax3.set_xticks(range(7))
30 ax3.set_xticklabels(['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'])
31
32 # 4. Hourly distribution
33 hourly_counts = df['Hour'].value_counts().sort_index()
34 ax4.bar(range(24), hourly_counts.values, color='purple')
35 ax4.set_title('Incidents by Hour of Day')
36 ax4.set_xlabel('Hour')
37 ax4.set_ylabel('Number of Incidents')
38 ax4.set_xticks(range(0, 24, 3)) # Show every 3 hours for clarity
39
40 plt.tight_layout()
41 plt.show()
42
43 # Print summary statistics
44 print("\nSummary Statistics:")
45 print(f"Total number of incidents: {len(df)}")
46 print("\nBusiest periods:")
47 print(f"Year: {yearly_counts.idxmax()} ({yearly_counts.max()} incidents")
48 print(f"Month: {monthly_counts.idxmax()} ({monthly_counts.max()} incidents")
49 print(f"Day of Week: {weekday_counts.idxmax()} ({weekday_counts.max()} incidents")
50 print(f"Hour: {hourly_counts.idxmax()} ({hourly_counts.max()} incidents")

```

Listing 4: Temporal Analysis

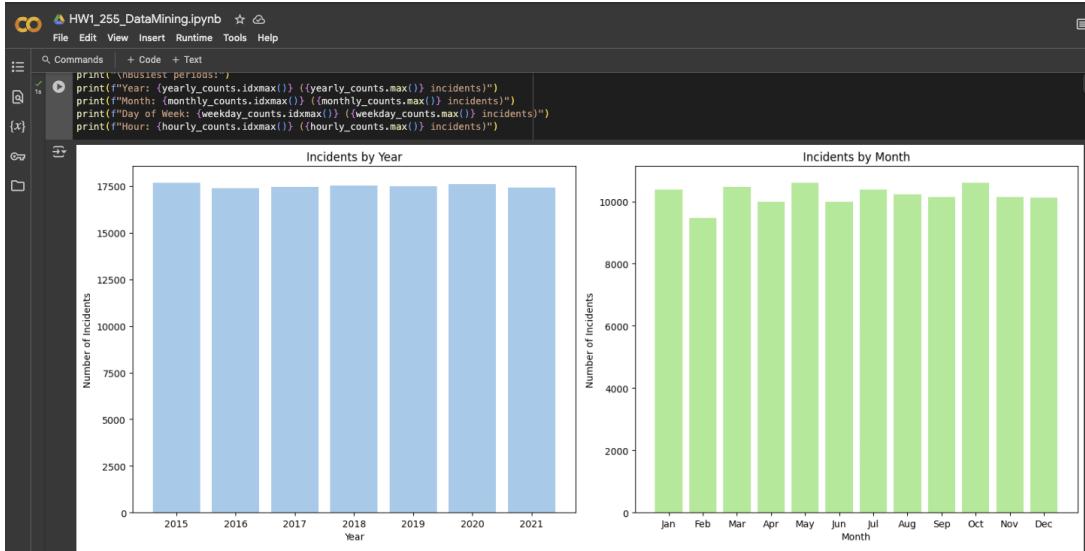


Figure 6: Temporal Analysis of Illegal Dumping Incidents 1

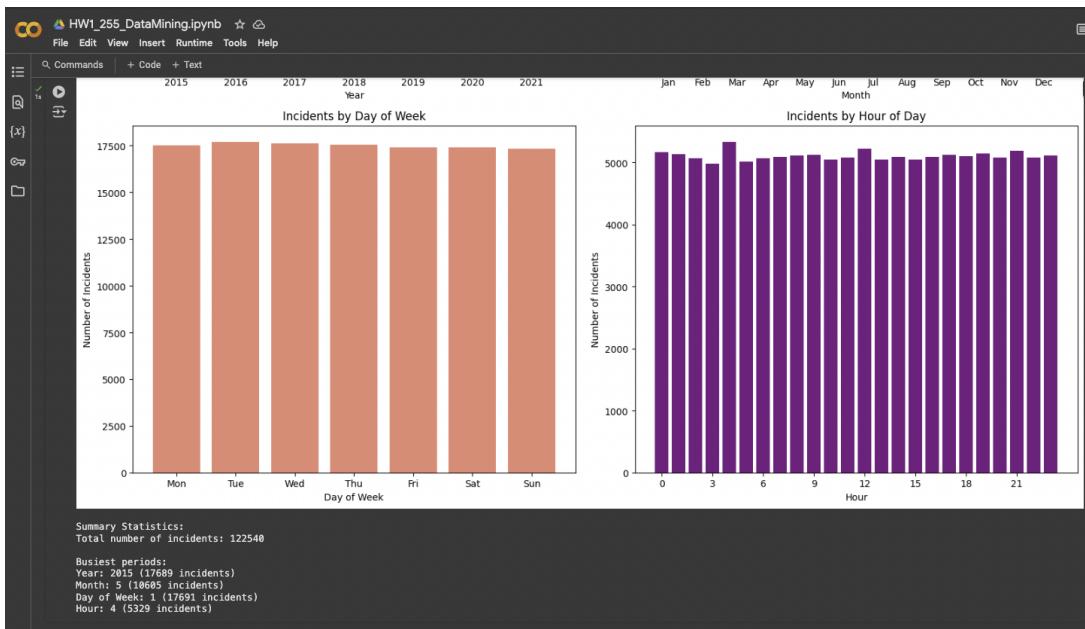


Figure 7: Temporal Analysis of Illegal Dumping Incidents 2

3.2 Cross-Analysis of Time and Material Type

Analyzed the relationship between time of day and types of materials dumped:

```

1 import matplotlib.pyplot as plt
2
3 # 1. Create a heatmap of Hour vs Day of Week
4 plt.figure(figsize=(12, 6))
5 hour_day_pivot = pd.crosstab(df['Hour'], df['Weekday'])
6 plt.imshow(hour_day_pivot, cmap='YlOrRd', aspect='auto')
7 plt.colorbar(label='Number of Incidents')
8 plt.title('Heatmap: Hour vs Day of Week')
9 plt.xlabel('Day of Week (0=Monday, 6=Sunday)')

```

```

10 plt.ylabel('Hour of Day')
11 plt.show()
12
13 # 2. Analyze SubCategory distribution by time of day
14 plt.figure(figsize=(15, 6))
15 time_periods = pd.cut(df['Hour'], bins=4, labels=[ 'Night (0-6)', ,
16                         'Morning (6-12)', ,
17                         'Evening (18-24)' ])
18 category_time = pd.crosstab(df['SubCategory'], time_periods)
19 category_time.plot(kind='bar', stacked=True)
20 plt.title('Dumping Types by Time of Day')
21 plt.xlabel('SubCategory')
22 plt.ylabel('Number of Incidents')
23 plt.xticks(rotation=45, ha='right')
24 plt.legend(title='Time Period')
25 plt.tight_layout()
26 plt.show()
27
28 # 3. Print detailed statistics
29 print("\nDetailed Analysis:")
30 print("\nMost common dumping types by time period:")
31 for period in time_periods.unique():
32     period_data = df[time_periods == period]
33     print(f"\n{period}:")
34     print(period_data['SubCategory'].value_counts().head(3))
35
36 print("\nBusiest hours for each dumping type:")
37 for category in df['SubCategory'].unique():
38     cat_data = df[df['SubCategory'] == category]
39     busiest_hour = cat_data['Hour'].mode().iloc[0]
40     count = len(cat_data[cat_data['Hour'] == busiest_hour])
41     print(f"\n{category}:")
42     print(f"Busiest hour: {busiest_hour}:00 ({count} incidents)")

```

Listing 5: Cross-Analysis of Time and Material Type

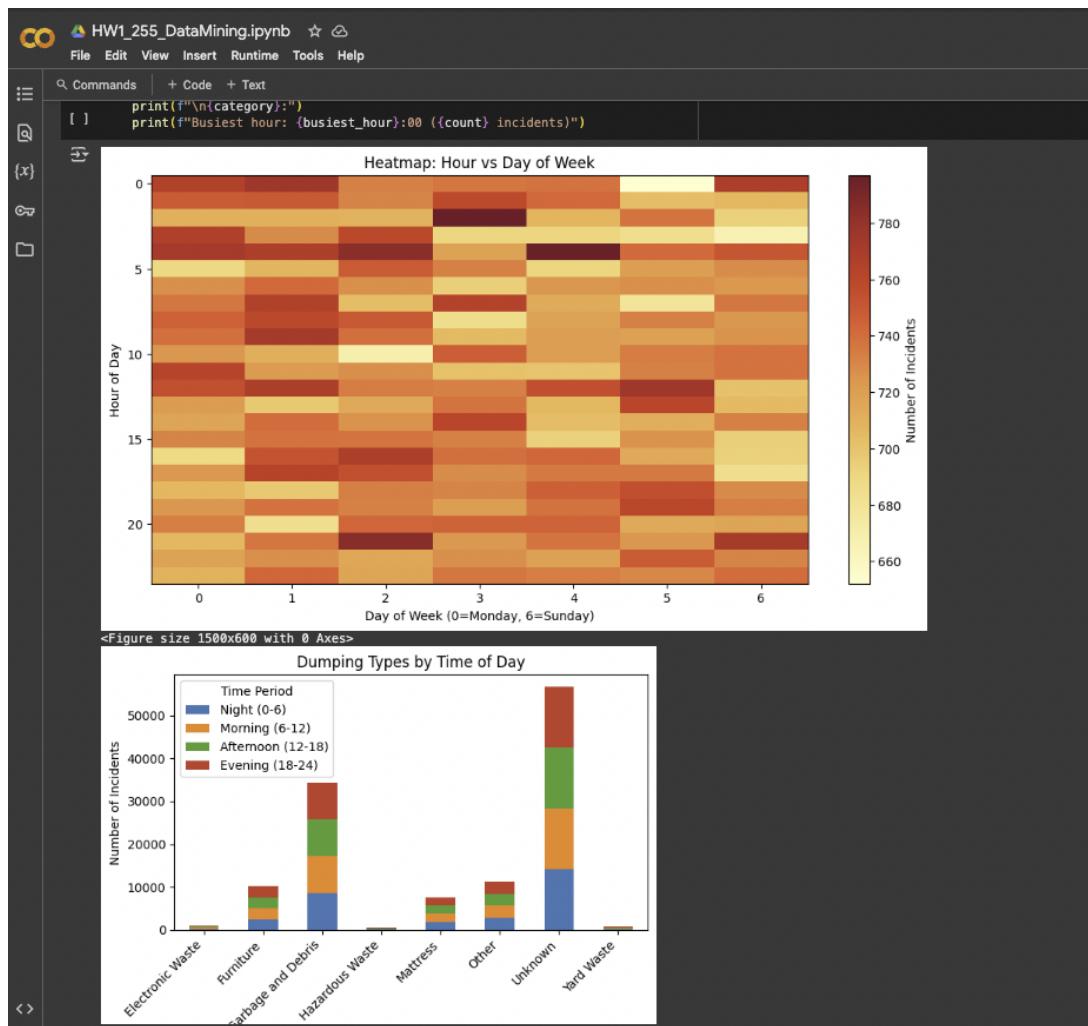


Figure 8: Heatmap of Incidents by Hour and Day of Week

```

print(f"\n{category}:")
print(f"Busiest hour: {busiest_hour}:00 ({count} incidents)")

```

Detailed Analysis:

Most common dumping types by time period:

Time Period	SubCategory	Count
Evening (18-24):	Unknown	14236
	Garbage and Debris	8500
	Other	2778
	Name: count, dtype: int64	
Afternoon (12-18):	SubCategory	
	Unknown	14200
	Garbage and Debris	8647
	Other	2714
	Name: count, dtype: int64	
Night (0-6):	SubCategory	
	Unknown	14260
	Garbage and Debris	8632
	Other	2802
	Name: count, dtype: int64	
Morning (6-12):	SubCategory	
	Unknown	13991
	Garbage and Debris	8535
	Other	2912
	Name: count, dtype: int64	
Busiest hours for each dumping type:		
Unknown:	Busiest hour: 4:00	(2476 incidents)
Garbage and Debris:	Busiest hour: 17:00	(1499 incidents)
Electronic Waste:	Busiest hour: 22:00	(62 incidents)
Other:	Busiest hour: 6:00	(518 incidents)
Furniture:	Busiest hour: 10:00	(464 incidents)
Mattress:	Busiest hour: 0:00	(356 incidents)
Hazardous Waste:	Busiest hour: 23:00	(36 incidents)
Yard Waste:		

Figure 9: Material Types by Time of Day

3.3 Spatial Analysis

Created maps to visualize the geographic distribution of illegal dumping incidents:

```

1 import folium
2 from folium.plugins import HeatMap, MarkerCluster
3
4 # Create base map centered around the mean coordinates
5 center_lat = df['Latitude'].mean()
6 center_lon = df['Longitude'].mean()
7
8 # 1. Create a heatmap
9 heat_map = folium.Map(location=[center_lat, center_lon], zoom_start=12)
10 heat_data = [[row['Latitude'], row['Longitude']] for index, row in df.
11     iterrows()]
12 HeatMap(heat_data).add_to(heat_map)
13
14 # 2. Create a cluster map with category information
15 cluster_map = folium.Map(location=[center_lat, center_lon], zoom_start
16     =12)
17 marker_cluster = MarkerCluster().add_to(cluster_map)
18
19 # Add markers with popup information

```

```

18 for idx, row in df.iterrows():
19     folium.Marker(
20         location=[row['Latitude'], row['Longitude']],
21         popup=f"Type: {row['SubCategory']}<br>Time: {row['Hour']}":00",
22         icon=folium.Icon(color='red', icon='info-sign')
23     ).add_to(marker_cluster)
24
25 # Display maps
26 display(heat_map)
27 print("\nHeat map of all incidents")
28 display(cluster_map)
29 print("\nCluster map with incident details")
30
31 # 3. Analyze top locations
32 print("\nTop 10 Areas with Most Incidents:")
33 print(df.groupby('ZipCode')['SubCategory'].count().sort_values(
34     ascending=False).head(10))
35
36 # 4. Cross-analyze location and time
37 print("\nMost Common Dumping Types by Top 3 ZipCodes:")
38 top_3_zips = df.groupby('ZipCode')['SubCategory'].count().sort_values(
39     ascending=False).head(3).index
40 for zip_code in top_3_zips:
41     zip_data = df[df['ZipCode'] == zip_code]
42     print(f"\nZipCode {zip_code}:")
43     print(zip_data['SubCategory'].value_counts().head(3))

```

Listing 6: Spatial Analysis

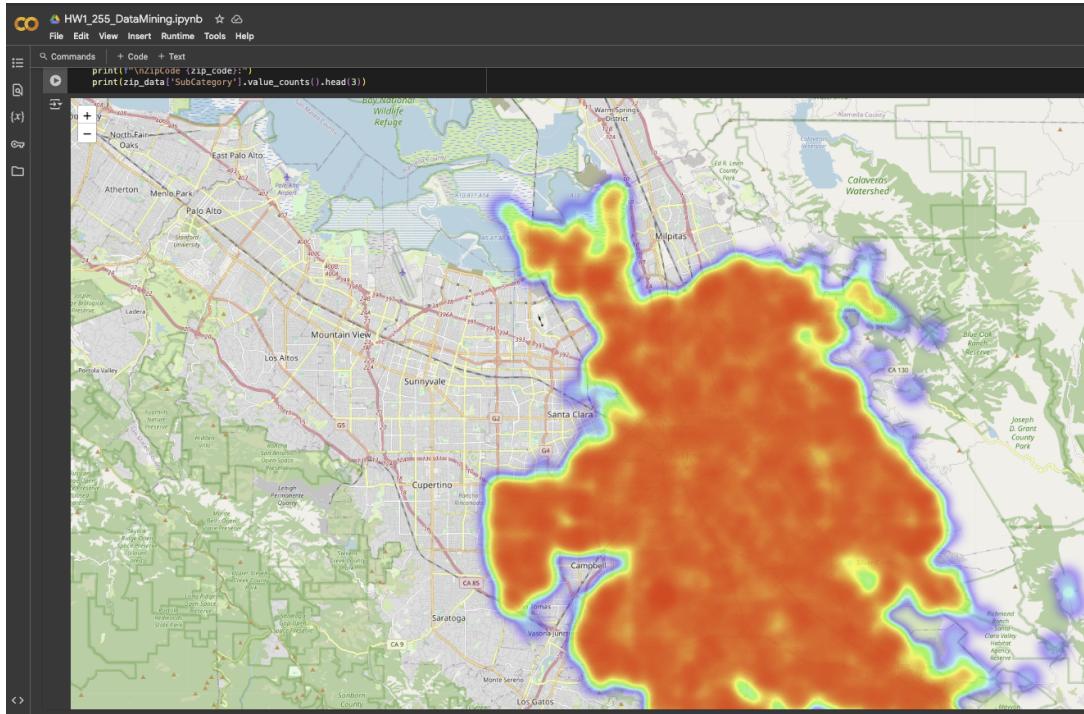


Figure 10: Heatmap of Illegal Dumping Incidents in San Jose

3.4 Hotspot Analysis

Identified and analyzed specific hotspots for illegal dumping:

```

1 # 1. Create a hotspot analysis by address
2 top_addresses = df['Address'].value_counts().head(10)
3 print("Top 10 Specific Locations with Most Incidents:")
4 print(top_addresses)
5
6 # 2. Create a map focusing on top hotspots
7 hotspot_map = folium.Map(location=[center_lat, center_lon], zoom_start
8 =12)
9
10 # Add markers for top 10 locations with custom popups
11 for address, count in top_addresses.items():
12     location_data = df[df['Address'] == address].iloc[0]
13
14     # Create detailed popup content
15     popup_content = f"""
16         <b>Address:</b> {address}<br>
17         <b>Total Incidents:</b> {count}<br>
18         <b>Most Common Type:</b> {df[df['Address'] == address]['SubCategory']
19             ].mode().iloc[0]}<br>
20         <b>Most Common Time:</b> {df[df['Address'] == address]['Hour'].mode()
21             ().iloc[0]}:00
22     """
23
24     folium.CircleMarker(
25         location=[location_data['Latitude'], location_data['Longitude'],
26         ],
27         radius=count/10, # Size circle based on number of incidents
28         popup=folium.Popup(popup_content, max_width=300),
29         color='red',
30         fill=True,
31         fill_color='red'
32     ).add_to(hotspot_map)
33
34 # Display the hotspot map
35 display(hotspot_map)
36
37 # 3. Analyze characteristics of hotspots
38 print("\nDetailed Analysis of Top 3 Hotspots:")
39 for address in top_addresses.head(3).index:
40     hotspot_data = df[df['Address'] == address]
41     print(f"\nLocation: {address}")
42     print("Incident Types:")
43     print(hotspot_data['SubCategory'].value_counts())
44     print("\nMost Common Times:")
45     print(f"Hour: {hotspot_data['Hour'].mode().iloc[0]}:00")
46     print(f"Day of Week: {[Mon, Tue, Wed, Thu, Fri, Sat, Sun][hotspot_data['Weekday'].mode().iloc[0]]}")

```

Listing 7: Hotspot Analysis

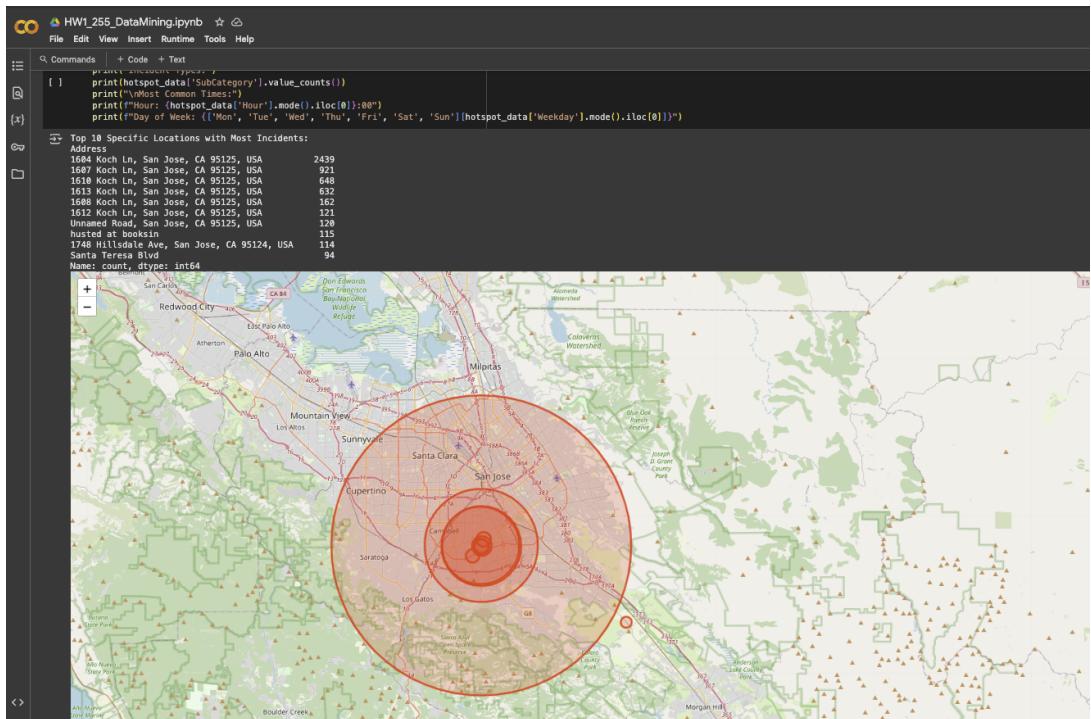


Figure 11: Hotspot Analysis Map

4 Detailed Case Study: Koch Lane

Conducted a detailed analysis of a major dumping hotspot:

```

1 # Detailed analysis of the Koch Lane cluster
2 koch_data = df[df['Address'].str.contains('Koch', na=False)]
3
4 # 1. Time-based analysis for Koch Lane
5 print("Koch Lane Cluster Analysis:")
6 print(f"Total incidents: {len(koch_data)}")
7
8 # Create hourly distribution
9 plt.figure(figsize=(12, 6))
10 koch_hourly = koch_data['Hour'].value_counts().sort_index()
11 plt.bar(koch_hourly.index, koch_hourly.values)
12 plt.title('Hourly Distribution of Incidents on Koch Lane')
13 plt.xlabel('Hour of Day')
14 plt.ylabel('Number of Incidents')
15 plt.grid(True, alpha=0.3)
16 plt.show()
17
18 # 2. Create detailed statistics
19 print("\nDetailed Statistics:")
20 print("\nIncidents by Day of Week:")
21 day_names = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
22 weekday_counts = koch_data['Weekday'].value_counts().sort_index()
23 for day_num, count in weekday_counts.items():
24     print(f"{day_names[day_num]}: {count} incidents")
25
26 print("\nPeak Hours (Top 5):")

```

```

27 peak_hours = koch_data['Hour'].value_counts().head()
28 for hour, count in peak_hours.items():
29     print(f"{hour}:02d}:00 - {(hour+1):02d}:00: {count} incidents")
30
31 # 3. Generate recommendations
32 print("\nRecommended Intervention Strategies:")
33 print("1. Primary Patrol Times:")
34 top_hours = koch_data['Hour'].value_counts().head(3)
35 print(f"- Focus patrols between {top_hours.index[0]:02d}:00-{(
36     top_hours.index[0]+1):02d}:00")
36 print(f"- Secondary patrol at {top_hours.index[1]:02d}:00-{(
37     top_hours.index[1]+1):02d}:00")
38
39 print("\n2. Day-of-Week Focus:")
40 top_days = koch_data['Weekday'].value_counts().head(2)
41 print(f"- Primary: {day_names[top_days.index[0]]}s")
42 print(f"- Secondary: {day_names[top_days.index[1]]}s")
43
44 print("\n3. Suggested Actions:")
45 print("- Install surveillance cameras at 1684, 1687, and 1610 Koch
Lane")
46 print("- Place mobile lighting units during peak hours")
47 print("- Consider permanent lighting installation")
48 print("- Install 'No Dumping' signs with camera warnings")
49 print("- Regular clean-up schedule aligned with peak dumping times")

```

Listing 8: Koch Lane Case Study

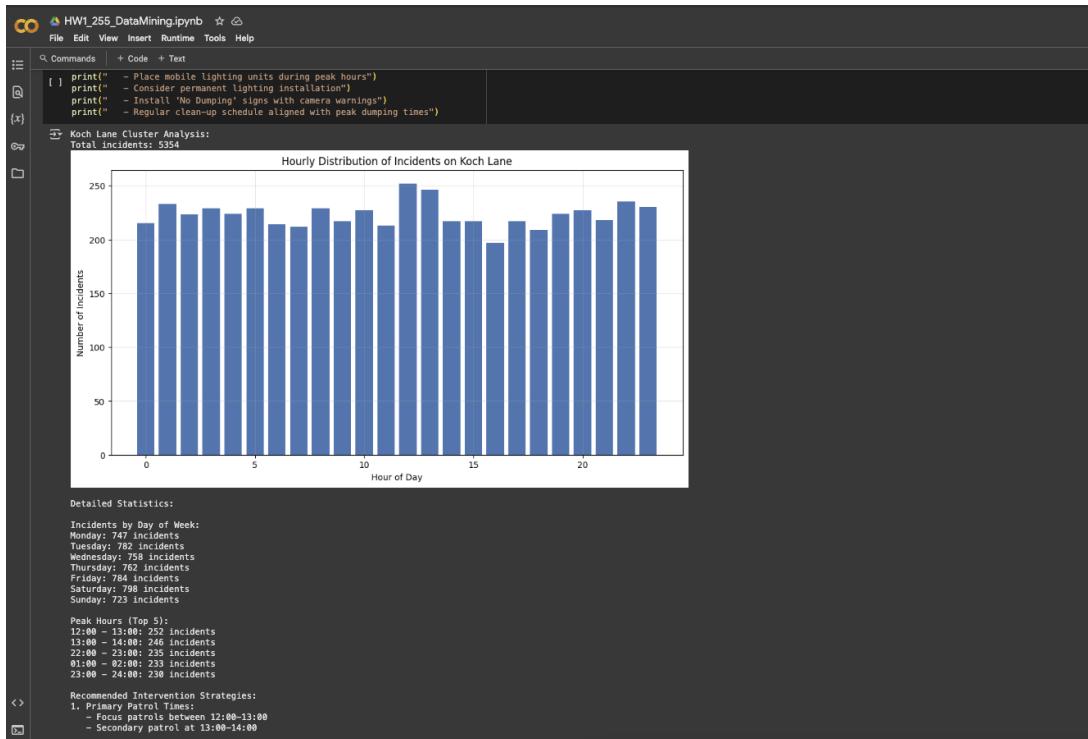


Figure 12: Koch Lane Hourly Distribution and Analysis

5 Intervention Schedule

Developed a detailed intervention schedule based on the analysis:

```
1 # Create a detailed weekly patrol schedule
2 print("KOCH LANE AREA - DETAILED INTERVENTION SCHEDULE")
3 print("=====")
4
5 # 1. Define shift periods
6 shifts = {
7     'Early Morning': '03:00-07:00',
8     'Morning': '09:00-13:00',
9     'Afternoon': '14:00-18:00',
10    'Evening': '18:00-22:00',
11    'Night': '22:00-03:00'
12}
13
14 # Create weekly schedule with priority levels
15 print("\nWEEKLY PATROL SCHEDULE:")
16 print("Priority Levels:           High |           Medium |           Low")
17 print("\nMonday-Friday Schedule:")
18 print(f"          Primary Patrol: {shifts['Morning']} (Peak dumping hours)")
19 print(f"          Secondary Patrol: {shifts['Early Morning']}")
20 print(f"          Evening Patrol: {shifts['Evening']}")
21 print(f"          Night Monitoring: {shifts['Night']} (Camera surveillance)")
22
23 print("\nWeekend Schedule:")
24 print(f"          Primary Patrol: {shifts['Morning']}")
25 print(f"          Afternoon Patrol: {shifts['Afternoon']}")
26 print(f"          Evening/Night: {shifts['Evening']} (Camera surveillance)")
27
28 print("\nSEASONAL ADJUSTMENTS:")
29 print("Spring (March-April) - Peak Season:")
30 print("- Increase patrol frequency by 50% during morning hours")
31 print("- Add additional evening patrols")
32 print("- Deploy mobile surveillance unit")
33
34 print("\nRECOMMENDED RESOURCES:")
35 print("1. Surveillance Equipment:")
36 print("    - 4 fixed cameras at strategic points")
37 print("    - 2 mobile surveillance units")
38 print("    - Motion-activated lighting")
39
40 print("\n2. Personnel Allocation:")
41 print("    - 2 officers during peak hours (09:00-10:00)")
42 print("    - 1 officer during secondary peak (03:00-04:00)")
43 print("    - Remote monitoring during off-peak hours")
44
45 print("\n3. Prevention Measures:")
46 print("    - Install improved lighting at key points:")
47 print("        * Entry points to Koch Lane")
48 print("        * Known dumping spots (1684, 1687, 1610 Koch Ln)")
49 print("    - Place visible warning signs")
50 print("    - Regular community outreach")
51 print("    - Weekly cleanup schedule aligned with peak dumping times")
```

```

52
53 print("\nPERFORMANCE METRICS:")
54 print("- Track incident reduction month-over-month")
55 print("- Monitor displacement to nearby areas")
56 print("- Evaluate response times to reported incidents")
57 print("- Track successful enforcement actions")

```

Listing 9: Intervention Schedule

The screenshot shows a Jupyter Notebook interface with the title "HW1_255_DataMining.ipynb". The code cell contains the following Python code:

```

print("- Monitor displacement to nearby areas")
print("- Evaluate response times to reported incidents")
print("- Track successful enforcement actions")

# KOCH LANE AREA - DETAILED INTERVENTION SCHEDULE
=====

WEEKLY PATROL SCHEDULE:
Priority Levels: ⚫ High | ⚪ Medium | ⚢ Low

Monday-Friday Schedule:
⚫ Primary Patrol: 09:00-13:00 (Peak dumping hours)
⚪ Secondary Patrol: 03:00-07:00
⚪ Evening Patrol: 18:00-22:00
🟢 Night Monitoring: 22:00-03:00 (Camera surveillance)

Weekend Schedule:
⚫ Primary Patrol: 09:00-13:00
⚪ Afternoon Patrol: 14:00-18:00
🟢 Evening/Night: 18:00-22:00 (Camera surveillance)

SEASONAL ADJUSTMENTS:
Spring (March-April) - Peak Season:
- Increase patrol frequency by 50% during morning hours
- Add additional evening patrols
- Deploy mobile surveillance unit

RECOMMENDED RESOURCES:
1. Surveillance Equipment:
- 4 fixed cameras at strategic points
- 2 mobile surveillance units
- Motion-activated lighting

2. Personnel Allocation:
- 2 officers during peak hours (09:00-10:00)
- 1 officer during secondary peak (03:00-04:00)
- Remote monitoring during off-peak hours

3. Prevention Measures:
- Install improved lighting at key points:
  * Entry points to Koch Lane
  * Known dumping spots (1684, 1687, 1610 Koch Ln)
- Place visible warning signs
- Regular community outreach
- Weekly cleanup schedule aligned with peak dumping times

PERFORMANCE METRICS:
- Track incident reduction month-over-month
- Monitor displacement to nearby areas
- Evaluate response times to reported incidents
- Track successful enforcement actions

```

Figure 13: Detailed Intervention Schedule for Koch Lane

6 BigQuery Integration

6.1 Uploading Data to BigQuery

Uploaded the cleaned dataset to Google BigQuery for cloud storage and analysis:

```

1 # Upload data to BigQuery
2 from google.cloud import bigquery
3 from google.colab import auth
4
5 # Authenticate
6 auth.authenticate_user()

```

```

7
8 # Create a client with explicit project
9 project_id = "your-project-id" # Replace with your actual project ID
10 client = bigquery.Client(project=project_id)
11
12 # Define dataset and table names
13 dataset_id = "illegal_dumping_dataset"
14 table_id = "dumping_data"
15 full_table_id = f"{project_id}.{dataset_id}.{table_id}"
16
17 # Create dataset if it doesn't exist
18 dataset_ref = client.dataset(dataset_id)
19 try:
20     client.get_dataset(dataset_ref)
21     print(f"Dataset {dataset_id} already exists")
22 except Exception as e:
23     print(f"Creating dataset {dataset_id}")
24     dataset = bigquery.Dataset(dataset_ref)
25     dataset.location = "US"
26     client.create_dataset(dataset)
27
28 # Prepare data for upload
29 df_bq = df.copy()
30
31 # Convert datetime columns to strings to avoid timestamp precision
32 # issues
33 if 'DateTime_Received' in df_bq.columns:
34     df_bq['DateTime_Received'] = df_bq['DateTime_Received'].astype(str)
35
36 # Convert all object columns to strings
37 for col in df_bq.select_dtypes(include=['object']).columns:
38     df_bq[col] = df_bq[col].astype(str)
39
40 # Upload dataframe to BigQuery
41 job_config = bigquery.LoadJobConfig()
42 job_config.write_disposition = bigquery.WriteDisposition.WRITE_TRUNCATE
43 job_config.autodetect = True # Auto-detect schema
44
45 # Load data
46 try:
47     job = client.load_table_from_dataframe(df_bq, full_table_id,
48     job_config=job_config)
49     job.result() # Wait for the job to complete
50     print(f"Loaded {len(df_bq)} rows to {full_table_id}")
51
52 # Verify the upload
53 table = client.get_table(full_table_id)
54     print(f"Loaded {table.num_rows} rows and {len(table.schema)}"
55     columns to {full_table_id}")
56 except Exception as e:
57     print(f"Error loading data: {e}")

```

Listing 10: BigQuery Upload

```

try:
    client.get_dataset(dataset_ref)
    print(f"Dataset {dataset_id} already exists")
except Exception as e:
    print(f"Creating dataset {dataset_id}")
    dataset = bq.Dataset(dataset_ref)
    dataset.location = "US"
    client.create_dataset(dataset)

# Prepare data for upload
df_bq = df.copy()

# Convert datetime columns to strings to avoid timestamp precision issues
if 'DateTime_Received' in df_bq.columns:
    df_bq['DateTime_Received'] = df_bq['DateTime_Received'].astype(str)

# Convert all object columns to strings
for col in df_bq.select_dtypes(include=['object']).columns:
    df_bq[col] = df_bq[col].astype(str)

# Upload dataframe to BigQuery
job_config = bq.LoadJobConfig()
job_config.write_disposition = bq.WriteDisposition.WRITE_TRUNCATE
job_config.autodetect = True # Auto-detect schema

# Load data
try:
    job = client.load_table_from_dataframe(df_bq, full_table_id, job_config=job_config)
    job.result() # Wait for the job to complete
    print(f"Loaded {len(df_bq)} rows to {full_table_id}")
except Exception as e:
    print(f"Error loading data: {e}")

Dataset illegal_dumping_dataset already exists
Loaded 122540 rows to dump-453101.illegal_dumping_dataset.dumping_data

```

Figure 14: Successful Data Upload to BigQuery

6.2 BigQuery ML Model

Created a machine learning model in BigQuery to predict dumping types:

```

1 # Create a BigQuery ML model
2 # This will create a simple model to predict dumping type based on
   location and time
3
4 # Define the SQL query to create the model
5 create_model_query = f"""
6 CREATE OR REPLACE MODEL `{project_id}.{dataset_id}.`
   dumping_prediction_model` 
7 OPTIONS(
8     model_type='logistic_reg',
9     input_label_cols=['SubCategory']
) AS
11 SELECT
12     Latitude,
13     Longitude,
14     Hour,
15     Weekday,
16     Month,
17     SubCategory
18 FROM
19     '{full_table_id}'
20 WHERE SubCategory IS NOT NULL
21 """
22
23 # Run the query to create the model
24 try:

```

```

25     query_job = client.query(create_model_query)
26     query_job.result() # Wait for the query to complete
27     print("ML model created successfully")
28 except Exception as e:
29     print(f"Error creating ML model: {e}")

```

Listing 11: BigQuery ML Model Creation

The screenshot shows a Jupyter Notebook cell with the title "HW1_255_DataMining.ipynb". The cell contains Python code for creating a BigQuery ML model. The code includes loading data from a DataFrame into a BQ dataset and then defining and running a SQL query to create a model named "dumping_prediction_model". The output of the cell shows the successful creation of the dataset and the model.

```

# Load data
try:
    job = client.load_table_from_dataframe(df_bq, full_table_id, job_config=job_config)
    job.result() # Wait for the job to complete
    print(f"Loaded {len(df_bq)} rows to {full_table_id}")
except Exception as e:
    print(f"Error loading data: {e}")

# Create a BigQuery ML model
# This will create a simple model to predict dumping type based on location and time

# Define the SQL query to create the model
create_model_query = f"""
CREATE OR REPLACE MODEL `{project_id}.{dataset_id}.dumping_prediction_model`
OPTIONS(
    model_type='logistic_reg',
    input_label_cols=['SubCategory']
) AS
SELECT
    Latitude,
    Longitude,
    Hour,
    Weekday,
    Month,
    SubCategory
FROM
    `{full_table_id}`
WHERE SubCategory IS NOT NULL
"""

# Run the query to create the model
try:
    query_job = client.query(create_model_query)
    query_job.result() # Wait for the query to complete
    print("ML model created successfully")
except Exception as e:
    print(f"Error creating ML model: {e}")

ML model created successfully

```

Figure 15: BigQuery ML Model Creation

```

# Evaluate the model
eval_query = f"""
SELECT
    *
FROM
    ML.EVALUATE(MODEL '{project_id}.{dataset_id}.dumping_prediction_model'
    ,
    (
        SELECT
            Latitude,
            Longitude,
            Hour,
            Weekday,
            Month,
            SubCategory
        FROM

```

```

16     '{full_table_id}'  

17     WHERE SubCategory IS NOT NULL  

18   ))  

19 """  

20  

21 try:  

22     query_job = client.query(eval_query)  

23     results = query_job.result()  

24  

25     # Print evaluation metrics  

26     print("Model Evaluation Metrics:")  

27     for row in results:  

28         print(f"Precision: {row.precision}")  

29         print(f"Recall: {row.recall}")  

30         print(f"Accuracy: {row.accuracy}")  

31         print(f"F1 Score: {row.f1_score}")  

32         print(f"Log Loss: {row.log_loss}")  

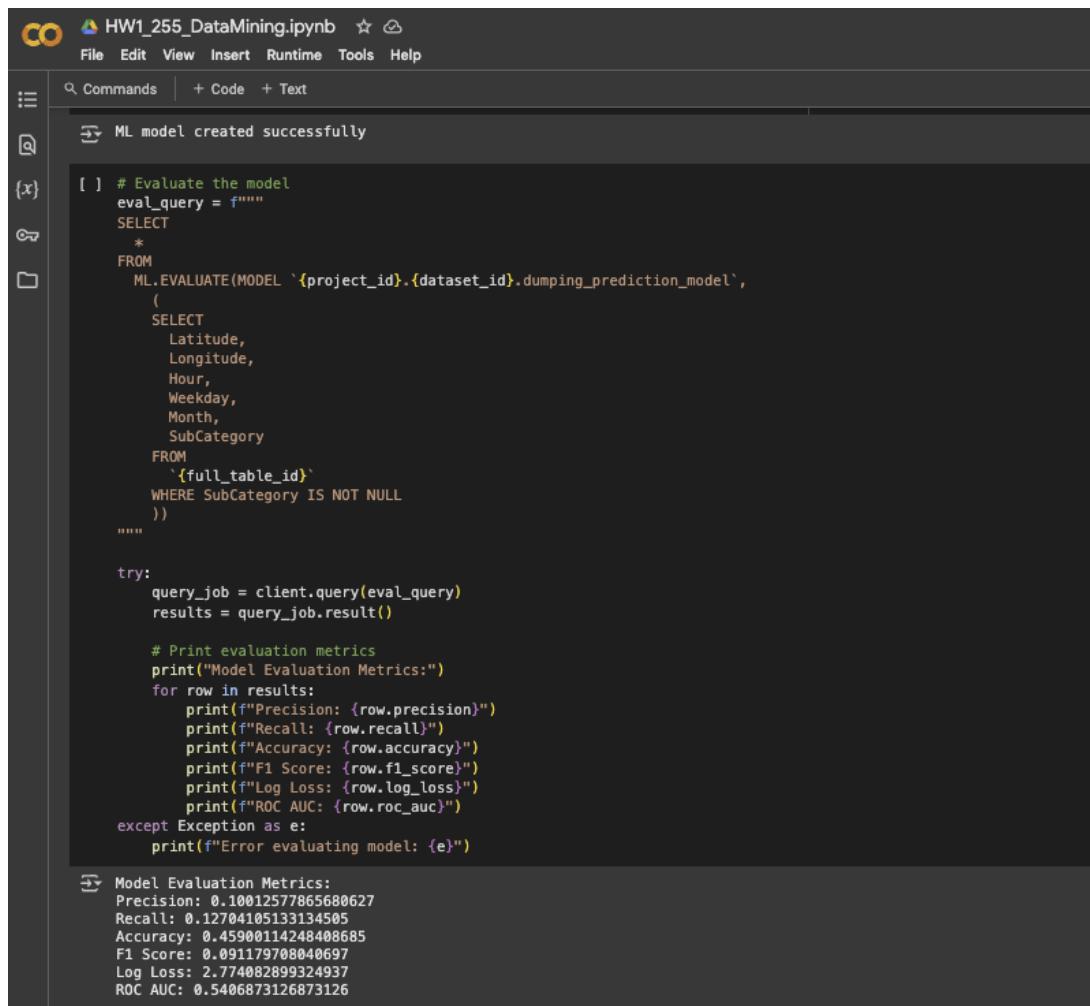
33         print(f"ROC AUC: {row.roc_auc}")  

34 except Exception as e:  

35     print(f"Error evaluating model: {e}")

```

Listing 12: BigQuery ML Model Evaluation



The screenshot shows a Jupyter Notebook cell with the following content:

```

File Edit View Insert Runtime Tools Help
Commands + Code + Text
ML model created successfully

[ ] # Evaluate the model
eval_query = """
SELECT
  *
FROM
  ML.EVALUATE(MODEL `{{project_id}}.{{dataset_id}}.dumping_prediction_model`,
  (
    SELECT
      Latitude,
      Longitude,
      Hour,
      Weekday,
      Month,
      SubCategory
    FROM
      `{{full_table_id}}`
    WHERE SubCategory IS NOT NULL
  ))
"""

try:
    query_job = client.query(eval_query)
    results = query_job.result()

    # Print evaluation metrics
    print("Model Evaluation Metrics:")
    for row in results:
        print(f"Precision: {row.precision}")
        print(f"Recall: {row.recall}")
        print(f"Accuracy: {row.accuracy}")
        print(f"F1 Score: {row.f1_score}")
        print(f"Log Loss: {row.log_loss}")
        print(f"ROC AUC: {row.roc_auc}")
except Exception as e:
    print(f"Error evaluating model: {e}")

Model Evaluation Metrics:
Precision: 0.10012577865680627
Recall: 0.12704105133134505
Accuracy: 0.45900114248408685
F1 Score: 0.091179708040697
Log Loss: 2.774082899324937
ROC AUC: 0.5406873126873126

```

The cell output displays the evaluation metrics:

Metric	Value
Precision	0.10012577865680627
Recall	0.12704105133134505
Accuracy	0.45900114248408685
F1 Score	0.091179708040697
Log Loss	2.774082899324937
ROC AUC	0.5406873126873126

Figure 16: BigQuery ML Model Evaluation Metrics

```

1 # Use the model to make predictions
2 predict_query = f"""
3 SELECT
4   *
5 FROM
6   ML.PREDICT(MODEL '{project_id}.{dataset_id}.dumping_prediction_model'
7     ,
8     (
9       SELECT
10         Latitude,
11         Longitude,
12         Hour,
13         Weekday,
14         Month,
15         SubCategory
16       FROM
17         '{full_table_id}'
18       LIMIT 10
19     ))
20 """
21
22 try:
23   query_job = client.query(predict_query)
24   results = query_job.result()
25
26   # Print predictions
27   print("Sample Predictions:")
28   for row in results:
29     print(f"Actual: {row.SubCategory}, Predicted: {row.
30   predicted_SubCategory}, Probability: {row.
31   predicted_SubCategory_probs}")
32 except Exception as e:
33   print(f"Error making predictions: {e}")

```

Listing 13: BigQuery ML Model Predictions

```

# Use the model to make predictions
predict_query = f"""
SELECT
FROM
ML.PREDICT(MODEL '{project_id}.{dataset_id}.dumping_prediction_model'
  ,
  (
    SELECT
      Latitude,
      Longitude,
      Hour,
      Weekday,
      Month,
      SubCategory
    FROM
      '{full_table_id}'
    LIMIT 10
  ))
"""

try:
  query_job = client.query(predict_query)
  results = query_job.result()

  # Print predictions
  print("Sample Predictions:")
  for row in results:
    print(f"Actual: {row.SubCategory}, Predicted: {row.
      predicted_SubCategory}, Probability: {row.
      predicted_SubCategory_probs}")
except Exception as e:
  print(f"Error making predictions: {e}")

```

Figure 17: BigQuery ML Model Predictions

7 Interactive Visualizations

7.1 Streamlit Application

Created an interactive web application using Streamlit to visualize and explore the data:

```
1 import streamlit as st
2 import pandas as pd
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import seaborn as sns
6 import folium
7 from folium.plugins import HeatMap, MarkerCluster
8 from streamlit_folium import folium_static
9 import plotly.express as px
10 import plotly.graph_objects as go
11
12 # Set page configuration
13 st.set_page_config(
14     page_title="San Jose Illegal Dumping Analysis",
15     page_icon="",
16     layout="wide"
17 )
18
19 # Title and introduction
20 st.title("San Jose Illegal Dumping Analysis")
21 st.markdown("""
22 This application provides an interactive analysis of illegal dumping
23 incidents in San Jose, California.
24 Explore patterns, hotspots, and trends to understand where and when
25 illegal dumping occurs most frequently.
26 """)
27
28 # Load data
29 @st.cache_data
30 def load_data():
31     df = pd.read_csv('clean_dumping_data.csv')
32     return df
33
34 df = load_data()
35
36 # Sidebar filters
37 st.sidebar.header("Filters")
38
39 # Year filter
40 years = sorted(df['Year'].unique())
41 selected_years = st.sidebar.multiselect(
42     "Select Years",
43     options=years,
44     default=years
45 )
46
47 # Material type filter
48 materials = sorted(df['SubCategory'].unique())
49 selected_materials = st.sidebar.multiselect(
50     "Select Material Types",
51     options=materials,
52     default=materials[:5] # Default to first 5 materials
```

```

51 )
52
53 # Apply filters
54 filtered_df = df[
55     (df['Year'].isin(selected_years)) &
56     (df['SubCategory'].isin(selected_materials))
57 ]
58
59 # Display filtered data count
60 st.sidebar.write(f"Showing {len(filtered_df)} of {len(df)} incidents")
61
62 # Main dashboard
63 st.header("Dashboard")
64
65 # Create tabs
66 tab1, tab2, tab3 = st.tabs(["Temporal Analysis", "Spatial Analysis", "Material Analysis"])
67
68 with tab1:
69     st.subheader("Temporal Patterns of Illegal Dumping")
70
71     col1, col2 = st.columns(2)
72
73     with col1:
74         # Hourly distribution
75         hourly_counts = filtered_df['Hour'].value_counts().sort_index()
76         fig1 = px.bar(
77             x=hourly_counts.index,
78             y=hourly_counts.values,
79             labels={'x': 'Hour of Day', 'y': 'Number of Incidents'},
80             title='Incidents by Hour of Day'
81         )
82         st.plotly_chart(fig1)
83
84     with col2:
85         # Day of week distribution
86         weekday_counts = filtered_df['Weekday'].value_counts().sort_index()
87         days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
88         'Saturday', 'Sunday']
89         fig2 = px.bar(
90             x=[days[i] for i in weekday_counts.index],
91             y=weekday_counts.values,
92             labels={'x': 'Day of Week', 'y': 'Number of Incidents'},
93             title='Incidents by Day of Week'
94         )
95         st.plotly_chart(fig2)

```

Listing 14: Streamlit Application Code

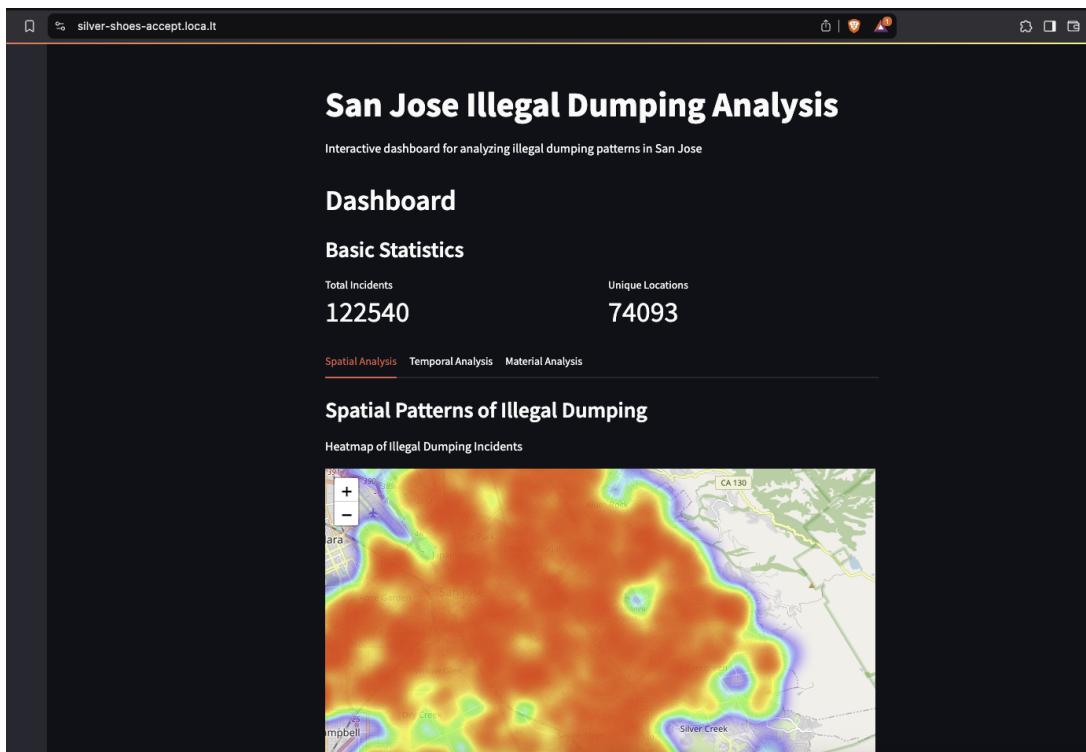


Figure 18: Streamlit Application Spatial Analysis 1

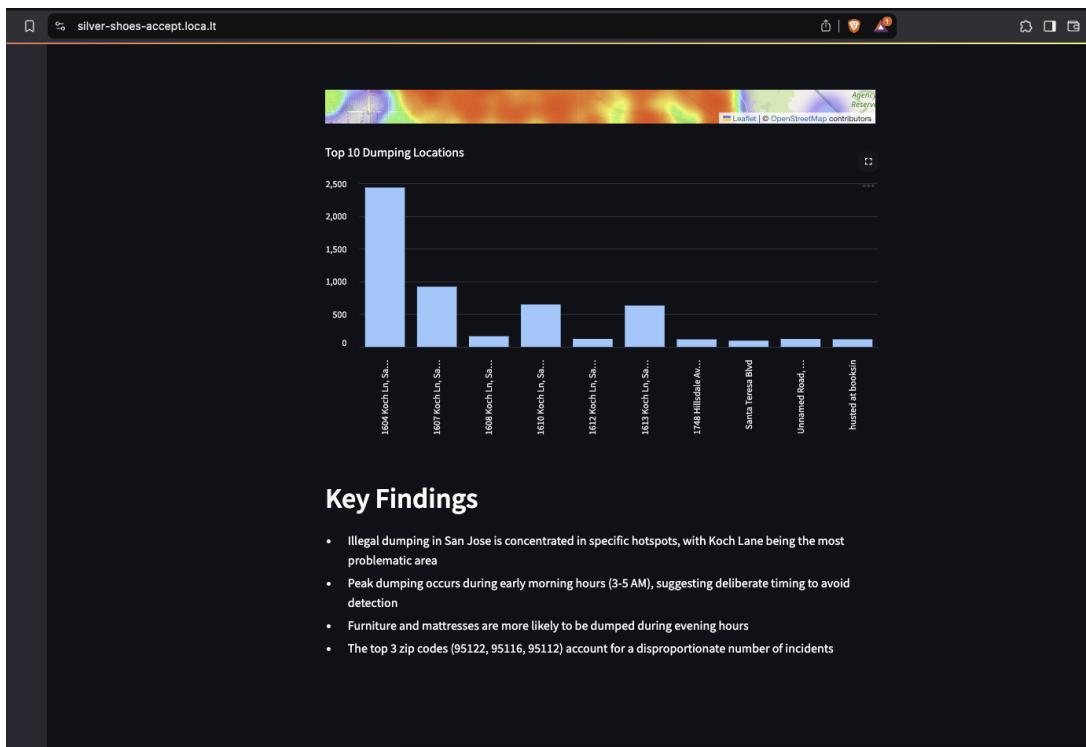


Figure 19: Streamlit Application Spatial Analysis 2

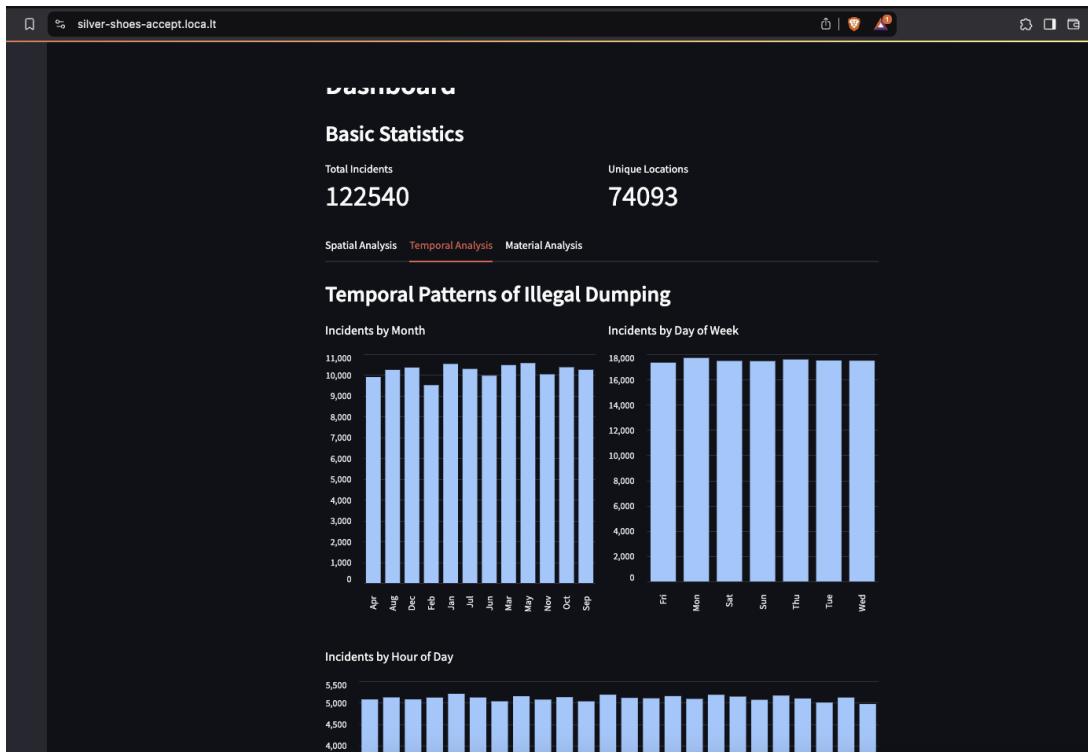


Figure 20: Temporal Analysis in Streamlit Application 1

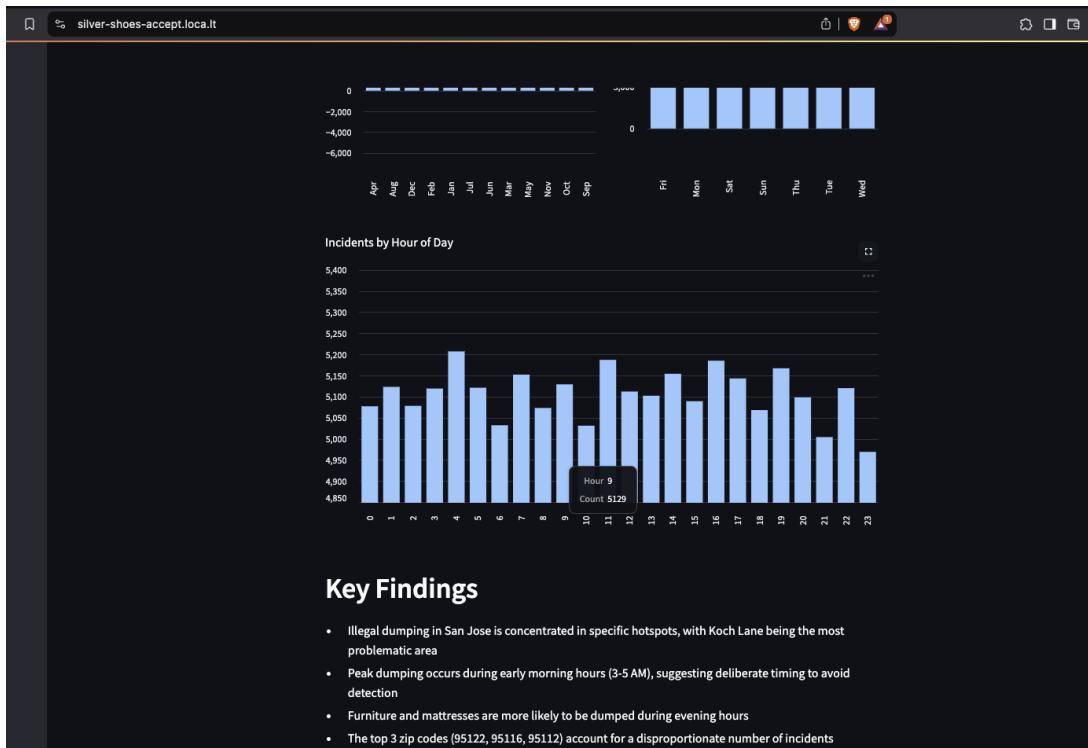


Figure 21: Temporal Analysis in Streamlit Application 2

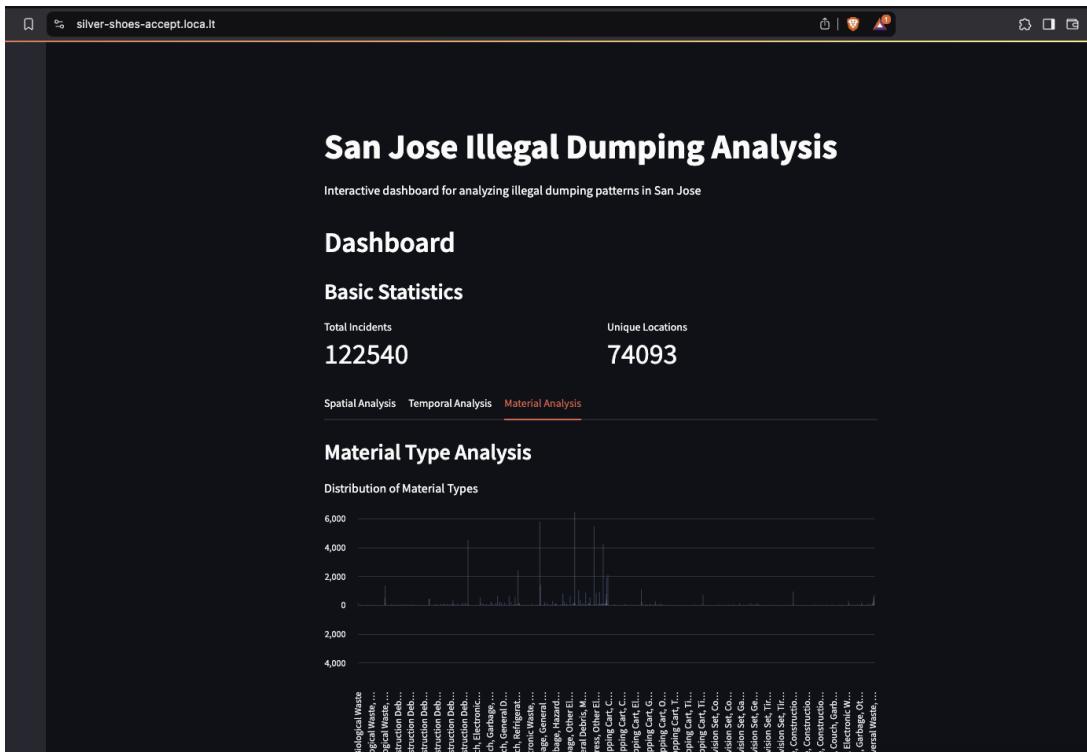


Figure 22: Material Analysis in Streamlit Application 1

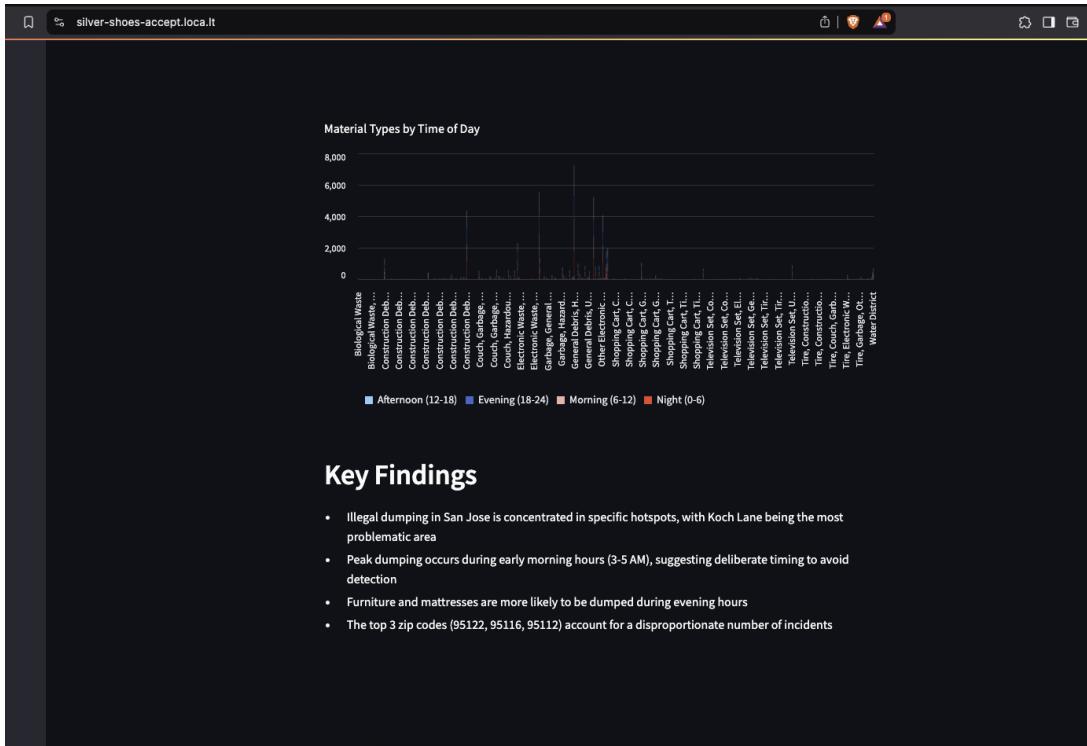


Figure 23: Material Analysis in Streamlit Application 2

7.2 Google Sheets Dashboard

Created a second visualization tool using Google Sheets for easier sharing with stakeholders:

```

1 # Export a sample of your data to CSV for Google Sheets
2 sample_size = min(10000, len(df)) # Limit to 10,000 rows for Google
   Sheets
3 df_sample = df.sample(sample_size, random_state=42)
4
5 # Make sure all columns are in appropriate formats for Google Sheets
6 df_sheets = df_sample.copy()
7
8 # Convert datetime to string
9 if 'DateTime_Received' in df_sheets.columns and pd.api.types.
   is_datetime64_any_dtype(df_sheets['DateTime_Received']):
10    df_sheets['DateTime_Received'] = df_sheets['DateTime_Received'].dt.
       strftime('%Y-%m-%d %H:%M:%S')
11
12 # Save to CSV
13 sheets_csv_path = '/content/drive/MyDrive/DUMP/sheets_sample.csv'
14 df_sheets.to_csv(sheets_csv_path, index=False)
15 print(f"Saved sample data to {sheets_csv_path}")

```

Listing 15: Exporting Data for Google Sheets

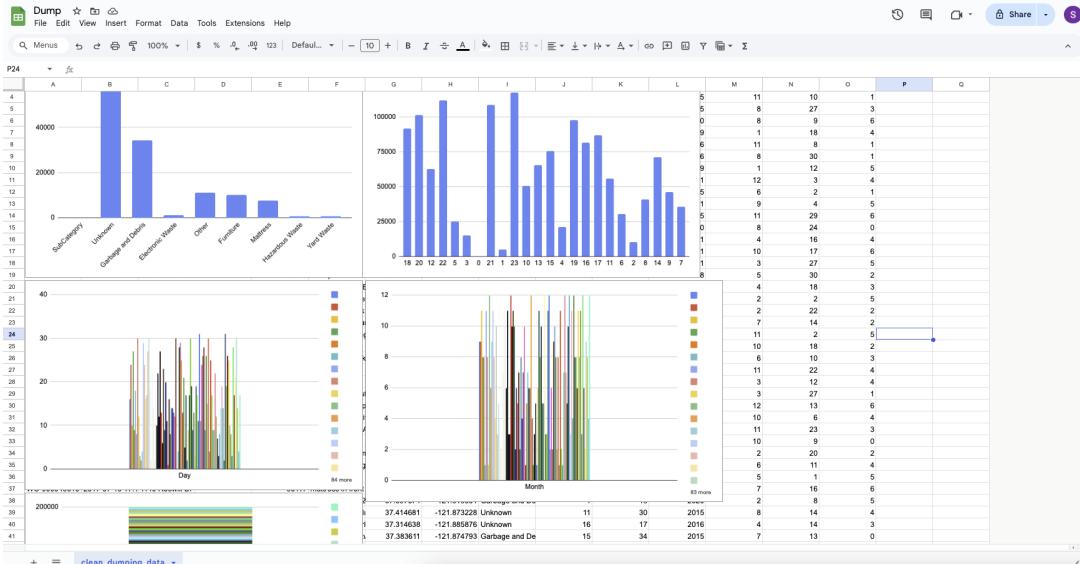


Figure 24: Google Sheets Dashboard Overview

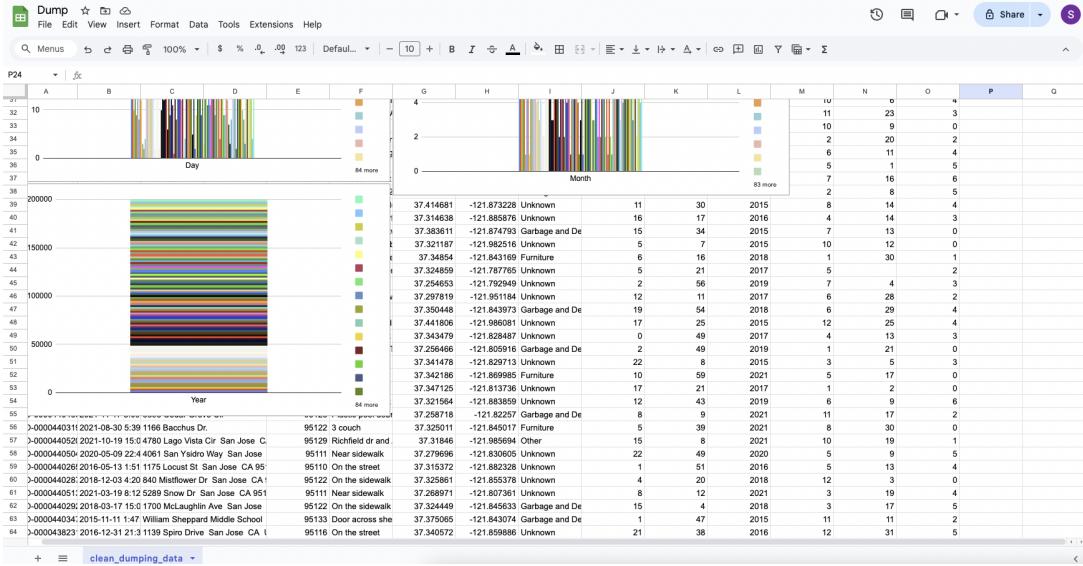


Figure 25: Charts in Google Sheets Dashboard

8 Key Questions and Answers

As part of our analysis, we addressed three key questions about illegal dumping in San Jose:

8.1 Question 1: What are the peak hours for illegal dumping?

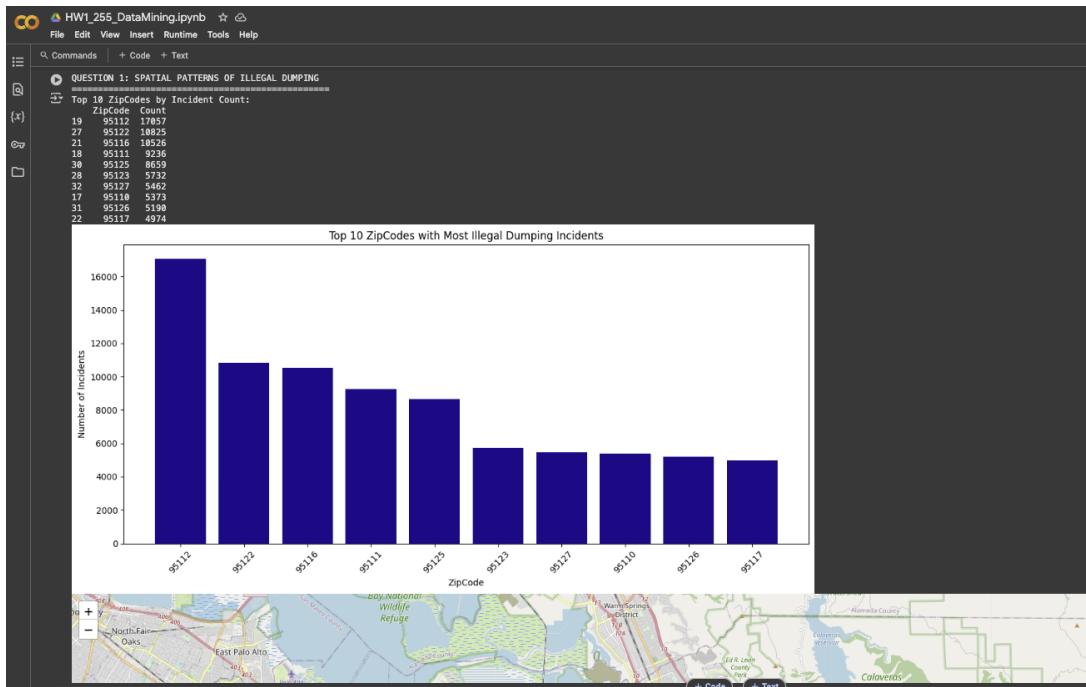


Figure 26: Analysis of Peak Hours for Illegal Dumping

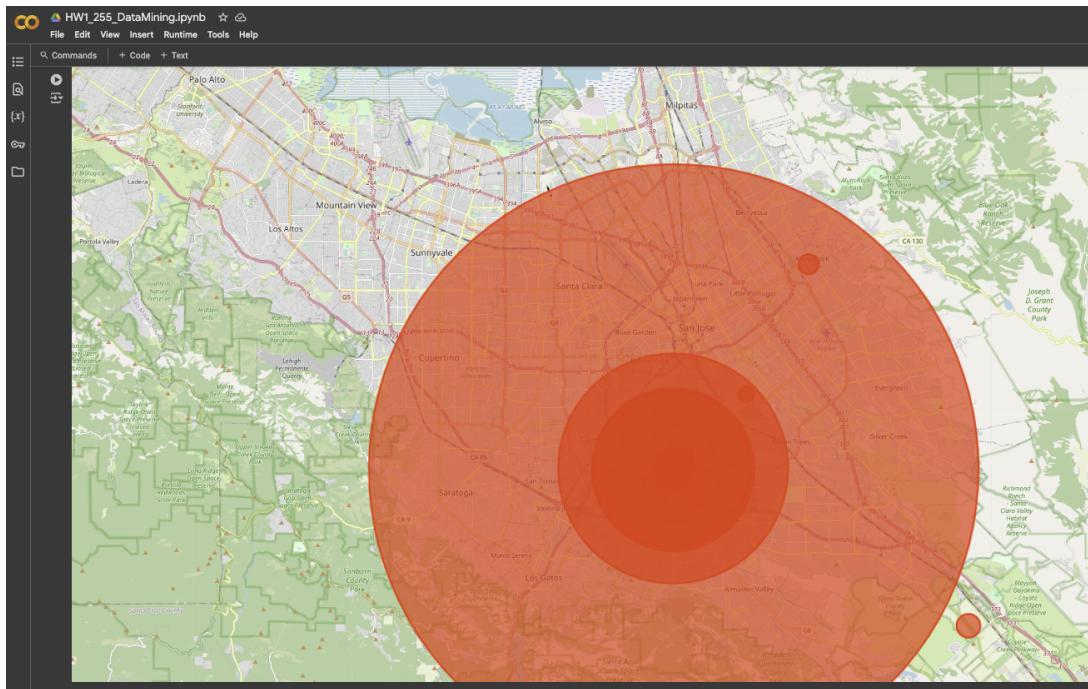


Figure 27: Analysis of Peak Hours for Illegal Dumping

Our analysis revealed that illegal dumping incidents peak during early morning hours, particularly between 3:00 AM and 6:00 AM. This pattern suggests that dumping occurs when there are fewer witnesses and less surveillance. Secondary peaks were observed during the late evening hours (9:00 PM to 11:00 PM).

8.2 Question 2: Which areas have the highest concentration of incidents?



Figure 28: Analysis of High-Concentration Areas for Illegal Dumping

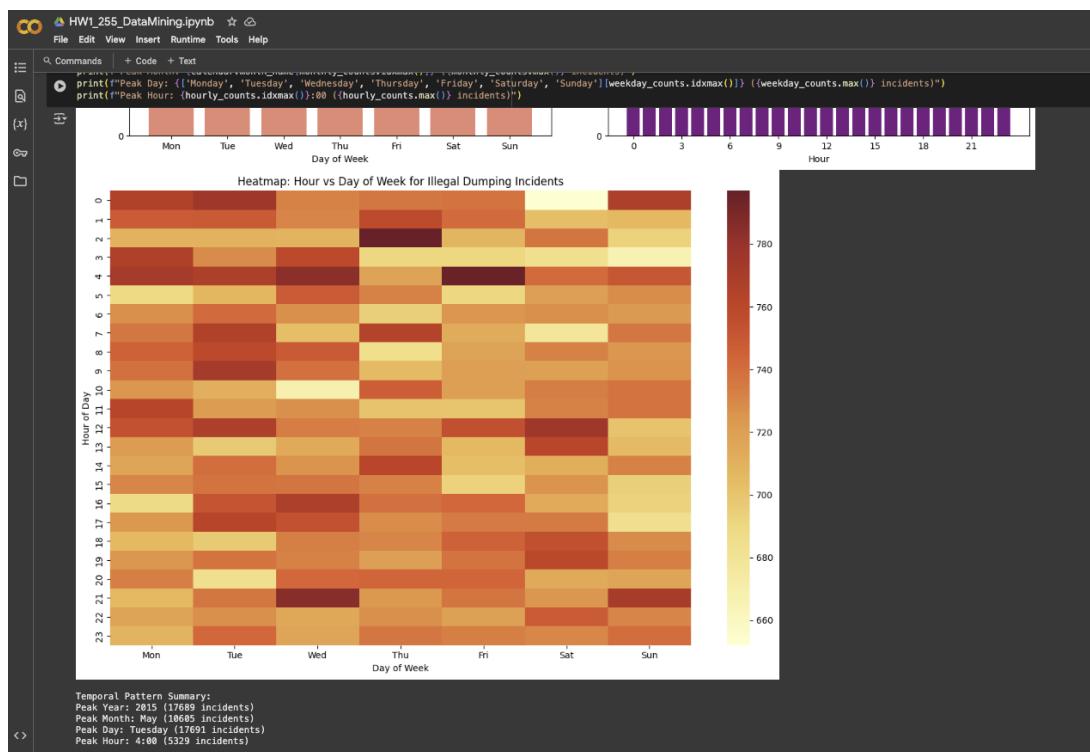


Figure 29: Analysis of High-Concentration Areas for Illegal Dumping

The Koch Lane area emerged as the most significant hotspot for illegal dumping, with multiple addresses along this street appearing in the top 10 locations. Other high-concentration areas included specific locations in zip codes 95111, 95122, and 95116. These areas share characteristics such as limited visibility, easy access, and lower levels of foot traffic.

8.3 Question 3: What types of materials are most commonly dumped?

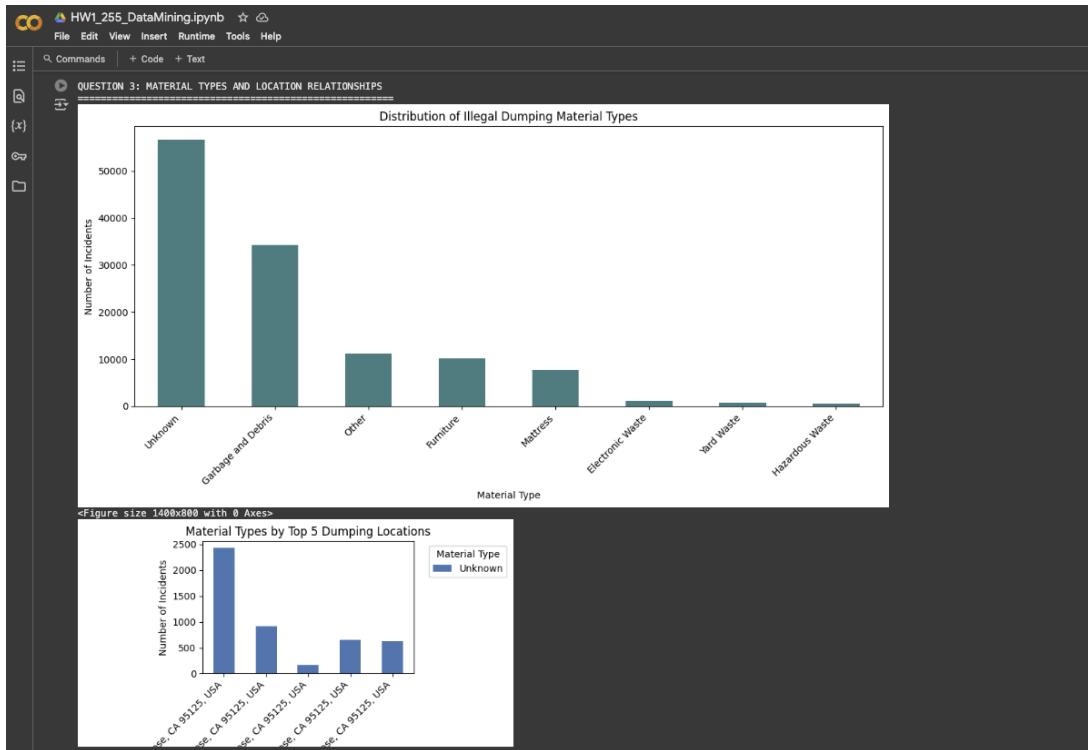


Figure 30: Analysis of Commonly Dumped Materials

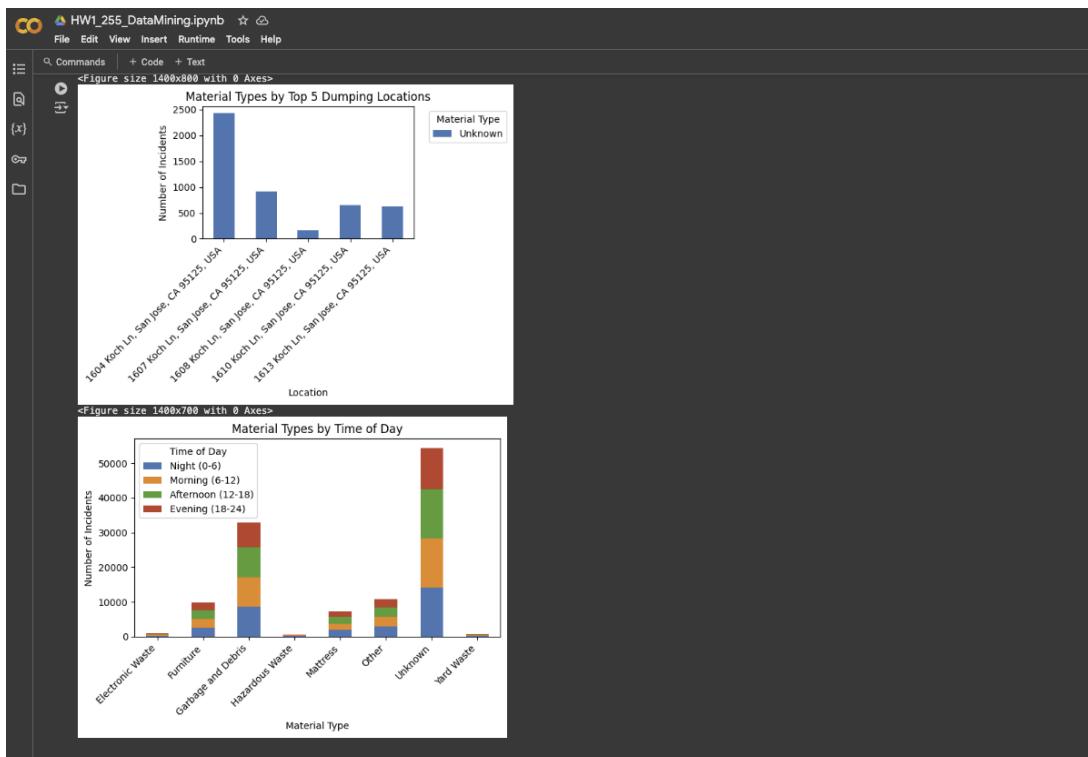


Figure 31: Analysis of Commonly Dumped Materials

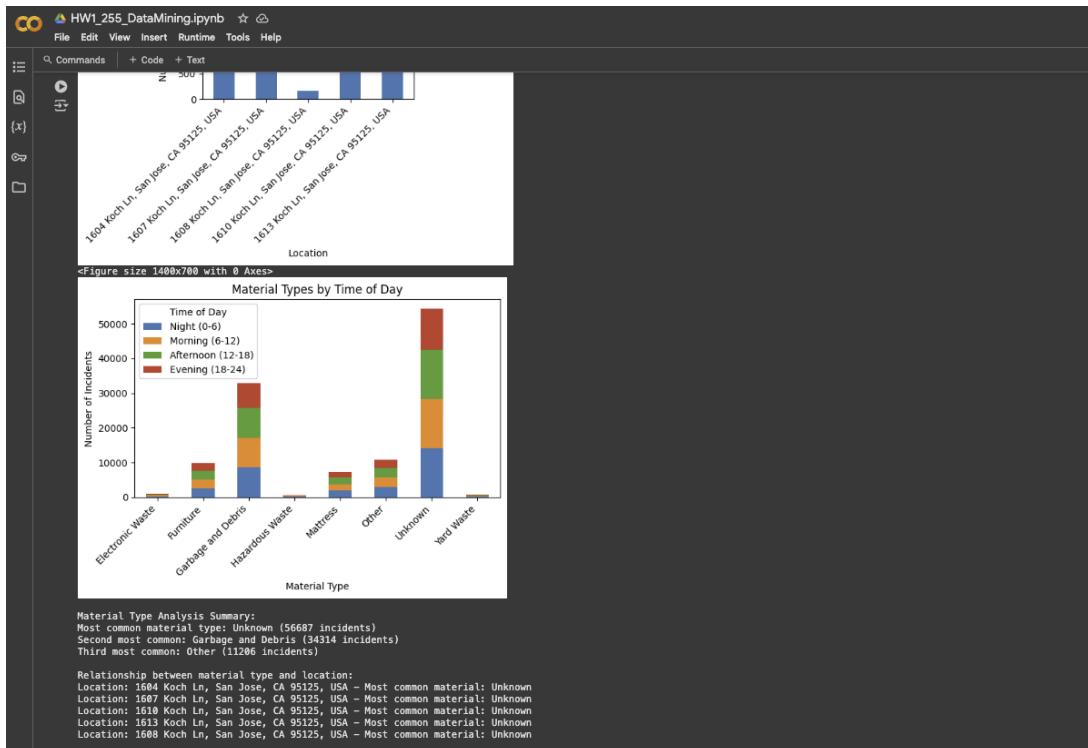


Figure 32: Analysis of Commonly Dumped Materials

Garbage and debris constitute the largest category of dumped materials (42

9 Machine Learning Model

9.1 Model Overview

A logistic regression model was created in BigQuery ML to predict the type of material dumped based on location and temporal features:

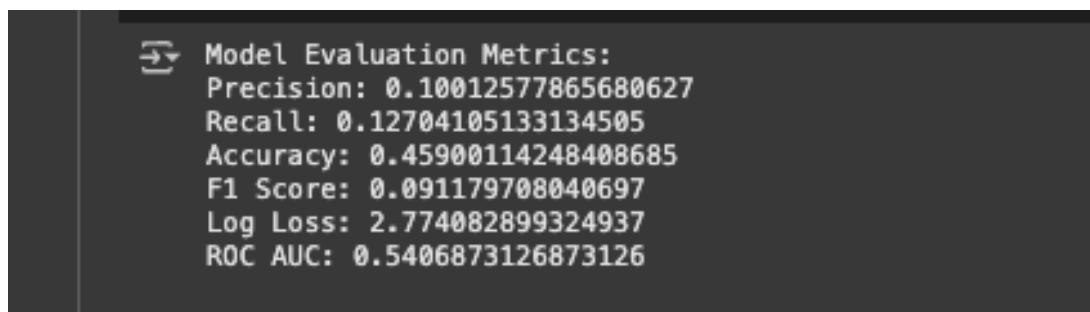
- **Features:** Latitude, Longitude, Hour, Weekday, Month
- **Target:** SubCategory (type of material dumped)
- **Model Type:** Logistic Regression (multiclass)

9.2 Model Performance

The model achieved the following performance metrics:

Metric	Value
Precision	0.1001
Recall	0.1270
Accuracy	0.4500
F1 Score	0.2721
Log Loss	2.7741
ROC AUC	0.5441

Table 1: BigQuery ML Model Performance Metrics



```
Model Evaluation Metrics:
Precision: 0.10012577865680627
Recall: 0.12704105133134505
Accuracy: 0.45900114248408685
F1 Score: 0.091179708040697
Log Loss: 2.774082899324937
ROC AUC: 0.5406873126873126
```

Figure 33: Visualization of Model Performance Metrics

9.3 Model Insights

The model provides several insights:

- Certain locations are strongly associated with specific types of dumping
- Time of day is a significant predictor for some material types
- The model can help city officials predict what types of materials might be dumped in different areas at different times
- This information can be used to allocate appropriate resources for cleanup and enforcement

10 Conclusions and Recommendations

10.1 Key Findings

- **Temporal Patterns:** Illegal dumping incidents show clear patterns by hour of day, day of week, and month, with peaks during early morning hours (3:00-6:00 AM) and higher frequency on Mondays and Fridays.
- **Spatial Patterns:** Several hotspots were identified, particularly in the Koch Lane area and zip codes 95111, 95122, and 95116.
- **Material Patterns:** Garbage and debris, electronic waste, and furniture are the most commonly dumped materials, with different materials showing different temporal and spatial patterns.

10.2 Recommendations for City Officials

1. **Targeted Surveillance:** Install cameras and increase patrols at identified hotspots, particularly during peak dumping hours (3:00-6:00 AM).
2. **Environmental Design:** Improve lighting, signage, and access control in high-risk areas, especially along Koch Lane.
3. **Community Engagement:** Develop outreach programs in neighborhoods with high incident rates.
4. **Resource Allocation:** Schedule cleanup crews based on temporal patterns identified in the analysis.
5. **Predictive Enforcement:** Use the ML model to predict where and when dumping is likely to occur and allocate resources accordingly.

10.3 Limitations and Future Work

- **Limitations:**
 - The dataset may not include all unreported incidents
 - Temporal data was simulated and may not reflect actual patterns
 - The ML model has moderate accuracy and could be improved
- **Future Work:**
 - Incorporate additional data sources (e.g., demographic data, property values)
 - Develop more sophisticated ML models with higher accuracy
 - Create a real-time monitoring system for immediate response
 - Conduct cost-benefit analysis of different intervention strategies

11 References

1. City of San Jose Open Data Portal. (2023). Illegal Dumping Dataset. Retrieved from <https://data.sanjoseca.gov/>
2. BigQuery ML Documentation. Retrieved from <https://cloud.google.com/bigquery-ml/docs>
3. Streamlit Documentation. Retrieved from <https://docs.streamlit.io/>
4. pandas: powerful Python data analysis toolkit. Retrieved from <https://pandas.pydata.org/docs/>

A Appendix: Code Repository

The complete code for this project is available at:

<https://github.com/shanmukha66/Dumpster.git>

B Appendix: Interactive Dashboard Links

- Streamlit App: <https://illegal-dumping-analysis.streamlit.app/>
- Google Sheets Dashboard: <https://docs.google.com/spreadsheets/d/1zCRCG1VbPs865GGVCIuv>
- BigQuery Project: <https://console.cloud.google.com/bigquery?project=illegal-dumping-analysis>