Final Project csc 380

May 8, 2025

---

1 CSC 380 Final Project

Contributor: Stephin Tomson

1.1 Analyzing the Correlation Between Streetlights and Property-Related Crimes in Tucson

---

1.1.1 Introduction

Problem Statement
Crime prevention is a critical aspect of urban planning. Adequate street lighting is commonly assumed to deter criminal activity and enhance public safety. However, its actual impact—particularly on property-related offenses—remains underexplored in Tucson.

Hypothesis
Areas with fewer streetlights are more prone to property-related crimes, both during the day and at night.

Objective
This project aims to:
- Determine if a measurable correlation exists between streetlight density and property-related crime rates in Tucson.
- Use machine learning models to predict crime risk in underlit areas, helping to inform future urban infrastructure planning.

1.1.2 Data Gathering and Cleaning

Overview
Data for this project was sourced from public Tucson datasets in CSV format and includes: - Arrest records (Tucson Police Department) - Streetlight infrastructure - Neighborhood income data - Parcel-level land/property information

Each dataset underwent the following cleaning and preprocessing steps:

- **Date & Time Parsing**
  Transformed date and time strings into datetime objects to enable accurate filtering and grouping (e.g., by hour or time of day).
- **Essential Column Selection**
  Unnecessary or irrelevant columns were dropped to focus on key attributes such as location, demographic info, and charge descriptions.
- **Column Renaming**
  Renamed columns for clarity and consistency (e.g., `X` → `*_location_x`), ensuring uniform naming conventions across datasets.
- **Time Period Classification**
  Derived a `time_period` column (Morning, Afternoon, Evening, Night) using the hour extracted from arrest time.
- **Filtering Active Records**
  Kept only "Active" entries from the streetlight dataset, and removed rows with critical null values in other datasets.
- **Grid-Based Spatial Indexing**
  Mapped coordinates to spatial grid cells (approx. 1km²) to facilitate neighborhood-level aggregation and merging across all datasets using `grid_cell_id`.

These steps ensured that each dataset was well-structured, consistent, and ready for integration and analysis.

---

[2]:
```
!pip install s3fs
```

```
Requirement already satisfied: s3fs in /usr/local/lib/python3.11/dist-packages
(2025.3.2)
Requirement already satisfied: aiobotocore<3.0.0,>=2.5.4 in
/usr/local/lib/python3.11/dist-packages (from s3fs) (2.22.0)
Requirement already satisfied: fsspec==2025.3.2.* in
/usr/local/lib/python3.11/dist-packages (from s3fs) (2025.3.2)
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in
/usr/local/lib/python3.11/dist-packages (from s3fs) (3.11.15)
Requirement already satisfied: aioitertools<1.0.0,>=0.5.1 in
/usr/local/lib/python3.11/dist-packages (from aiobotocore<3.0.0,>=2.5.4->s3fs)
(0.12.0)
Requirement already satisfied: botocore<1.37.4,>=1.37.2 in
/usr/local/lib/python3.11/dist-packages (from aiobotocore<3.0.0,>=2.5.4->s3fs)
(1.37.3)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in
/usr/local/lib/python3.11/dist-packages (from aiobotocore<3.0.0,>=2.5.4->s3fs)
(2.9.0.post0)
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in
/usr/local/lib/python3.11/dist-packages (from aiobotocore<3.0.0,>=2.5.4->s3fs)
(1.0.1)
```

```
Requirement already satisfied: multidict<7.0.0,>=6.0.0 in
/usr/local/lib/python3.11/dist-packages (from aiobotocore<3.0.0,>=2.5.4->s3fs)
(6.4.3)
Requirement already satisfied: wrapt<2.0.0,>=1.10.10 in
/usr/local/lib/python3.11/dist-packages (from aiobotocore<3.0.0,>=2.5.4->s3fs)
(1.17.2)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in
/usr/local/lib/python3.11/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->s3fs)
(2.6.1)
Requirement already satisfied: aiosignal>=1.1.2 in
/usr/local/lib/python3.11/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->s3fs)
(1.3.2)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-
packages (from aiohttp!=4.0.0a0,!=4.0.0a1->s3fs) (25.3.0)
Requirement already satisfied: frozenlist>=1.1.1 in
/usr/local/lib/python3.11/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->s3fs)
(1.6.0)
Requirement already satisfied: propcache>=0.2.0 in
/usr/local/lib/python3.11/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->s3fs)
(0.3.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in
/usr/local/lib/python3.11/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->s3fs)
(1.20.0)
Requirement already satisfied: urllib3!=2.2.0,<3,>=1.25.4 in
/usr/local/lib/python3.11/dist-packages (from
botocore<1.37.4,>=1.37.2->aiobotocore<3.0.0,>=2.5.4->s3fs) (2.4.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-
packages (from python-dateutil<3.0.0,>=2.1->aiobotocore<3.0.0,>=2.5.4->s3fs)
(1.17.0)
Requirement already satisfied: idna>=2.0 in /usr/local/lib/python3.11/dist-
packages (from yarl<2.0,>=1.17.0->aiohttp!=4.0.0a0,!=4.0.0a1->s3fs) (3.10)
```

[3]:

```python
# Importing all of the necessary packages that are going to be utilized␣
↪throughout the Notebook.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_squared_error, classification_report
import folium
from datetime import datetime
import geopandas as gpd
from scipy import stats
```

```python
import s3fs
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).
```

###Importing data

[4]:
```python
tuc_arrest_data = pd.read_csv('/content/drive/MyDrive/csc380_data/
↪Tucson_Police_Arrests_-_2021_-_Open_Data.csv', low_memory=False,␣
↪encoding=None)
tuc_light_data = pd.read_csv('/content/drive/MyDrive/csc380_data/
↪Streetlights_-_City_of_Tucson_-_Open_Data.csv', low_memory=False,␣
↪encoding=None)
tuc_neighbourhood_income = pd.read_csv('/content/drive/MyDrive/csc380_data/
↪Neighborhood_Income.csv', low_memory=False, encoding=None)
tuc_parcel_data = pd.read_csv('/content/drive/MyDrive/csc380_data/
↪Parcels_-_Regional.csv', low_memory=False, encoding=None)
```

1.1.3 Printing Data Types

[5]:
```python
print("Arrests Data Info:")  # Print a header for the arrests data information
print(tuc_arrest_data.info())  # Print the information summary of the arrests␣
↪DataFrame
print("\nArrests Data Head:")  # Print a header for the first few rows of the␣
↪arrests data
print(tuc_arrest_data.head())  # Print the first 5 rows of the arrests DataFrame
```

```
Arrests Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45421 entries, 0 to 45420
Data columns (total 39 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   OBJECTID        45421 non-null  int64
 1   X               45421 non-null  float64
 2   Y               45421 non-null  float64
 3   arre_id         45421 non-null  int64
 4   case_id         45421 non-null  int64
 5   agency          45421 non-null  object
 6   date_arr        45421 non-null  object
 7   time_arr        45421 non-null  int64
 8   datetime_arr    45421 non-null  object
 9   MONTH_ARR       45421 non-null  object
 10  YEAR_ARR        45421 non-null  int64
 11  DOW_ARR         45421 non-null  object
```

```
 12  TIME_ARRST      45421 non-null  int64
 13  age             45421 non-null  object
 14  race            45421 non-null  object
 15  sex             45421 non-null  object
 16  ethnicity       45421 non-null  object
 17  arr_type        45421 non-null  object
 18  neighborhd      45421 non-null  object
 19  ADDRESS_PUBLIC  45416 non-null  object
 20  city            43901 non-null  object
 21  state           43899 non-null  object
 22  zip             43901 non-null  object
 23  arr_chrg        45421 non-null  object
 24  chrgdesc        45421 non-null  object
 25  chrg_cnt        45421 non-null  int64
 26  fel_misd        45421 non-null  object
 27  chrg_seq        45421 non-null  int64
 28  APPSTATE        45421 non-null  int64
 29  LOC_STATUS      45421 non-null  object
 30  WARD            44591 non-null  float64
 31  NHA_NAME        32086 non-null  object
 32  TMSECT          0 non-null      float64
 33  DIVISION        45065 non-null  object
 34  DIVISION_NO     44576 non-null  object
 35  DIVSECT         44576 non-null  object
 36  TRSQ            45049 non-null  object
 37  City_geo        44995 non-null  object
 38  ADDRESS_100BLK  45416 non-null  object
dtypes: float64(4), int64(9), object(26)
memory usage: 13.5+ MB
None

Arrests Data Head:
   OBJECTID             X              Y     arre_id     case_id agency  \
0         1  9.900089e+05  470751.276735  2021000107  2101020104    TPD
1         2  9.900089e+05  470751.276735  2021000107  2101020104    TPD
2         3  9.900089e+05  470751.276735  2021000107  2101020104    TPD
3         4  1.053154e+06  443419.380064  2021000110  2101020138    TPD
4         5  1.053154e+06  443419.380064  2021000110  2101020138    TPD

                 date_arr  time_arr            datetime_arr MONTH_ARR  …  \
0  2021/01/02 00:00:00+00      1731  2021/01/02 17:31:00+00    01-Jan  …
1  2021/01/02 00:00:00+00      1731  2021/01/02 17:31:00+00    01-Jan  …
2  2021/01/02 00:00:00+00      1731  2021/01/02 17:31:00+00    01-Jan  …
3  2021/01/02 00:00:00+00      1844  2021/01/02 18:44:00+00    01-Jan  …
4  2021/01/02 00:00:00+00      1844  2021/01/02 18:44:00+00    01-Jan  …

   LOC_STATUS WARD  NHA_NAME TMSECT                  DIVISION DIVISION_NO  \
0    GEOCODED  3.0       NaN    NaN  Operations Division West          T2
```

```
1    GEOCODED  3.0       NaN    NaN  Operations Division West          T2
2    GEOCODED  3.0       NaN    NaN  Operations Division West          T2
3    GEOCODED  2.0  Eastside    NaN  Operations Division East          T4
4    GEOCODED  2.0  Eastside    NaN  Operations Division East          T4

  DIVSECT         TRSQ City_geo            ADDRESS_100BLK
0    T203  13S13E24NW   TUCSON          4598 N ORACLE RD
1    T203  13S13E24NW   TUCSON          4598 N ORACLE RD
2    T203  13S13E24NW   TUCSON          4598 N ORACLE RD
3    T406  14S15E14NE   TUCSON  10198 E ESSEX VILLAGE DR
4    T406  14S15E14NE   TUCSON  10198 E ESSEX VILLAGE DR

[5 rows x 39 columns]
```

[6]:

```python
print("Lights Data Info:")  # Print a header for the streetlights data␣
 ↪information
print(tuc_light_data.info())  # Print the information summary of the␣
 ↪streetlights DataFrame
print("\nLights Data Head:")  # Print a header for the first few rows of the␣
 ↪streetlights data
print(tuc_light_data.head())  # Print the first 5 rows of the streetlights␣
 ↪DataFrame
```

```
Lights Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22781 entries, 0 to 22780
Data columns (total 21 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   X                    22768 non-null  float64
 1   Y                    22768 non-null  float64
 2   OBJECTID             22781 non-null  int64
 3   Model                22423 non-null  object
 4   Type                 22446 non-null  object
 5   Bulb_Type            22455 non-null  object
 6   Wattage              22338 non-null  float64
 7   Voltage              22235 non-null  float64
 8   Address_Number       18468 non-null  float64
 9   Street               19451 non-null  object
 10  City                 22542 non-null  object
 11  Light_Fixture_Theme  22781 non-null  object
 12  DATASOURCE           22781 non-null  object
 13  Permit_Number        562 non-null    object
 14  CartegraphID         22768 non-null  object
 15  SHAPE                0 non-null      float64
 16  Retired              8 non-null      object
 17  Status               22768 non-null  object
```

```
 18  MacID                 19638 non-null  object
 19  TEP_Account_Number    22183 non-null  float64
 20  Power_Pedestal_ID     22191 non-null  float64
dtypes: float64(8), int64(1), object(12)
memory usage: 3.7+ MB
None

Lights Data Head:
              X              Y  OBJECTID      Model      Type Bulb_Type  \
0  1.001233e+06  421018.579396         1  ATBM D R3  Autobahn       LED
1  1.001142e+06  420902.066601         2  ATBM D R3  Autobahn       LED
2  1.001234e+06  420785.376969         3  ATBM D R3  Autobahn       LED
3  1.001143e+06  420667.403543         4  ATBM D R3  Autobahn       LED
4  1.001237e+06  420582.028543         5  ATBM D R3  Autobahn       LED

   Wattage  Voltage  Address_Number         Street  … Light_Fixture_Theme  \
0     95.0    480.0          5425.0  S Campbell Av  …               Other
1     95.0    480.0          5434.0  S Campbell Av  …               Other
2     95.0    480.0          5441.0  S Campbell Av  …               Other
3     95.0    480.0          5454.0  S Campbell Av  …               Other
4     95.0    480.0          5457.0  S Campbell Av  …               Other

          DATASOURCE Permit_Number CartegraphID SHAPE  Retired  Status  \
0  TDOT_STREETLIGHTS           NaN           51   NaN      NaN  Active
1  TDOT_STREETLIGHTS           NaN           52   NaN      NaN  Active
2  TDOT_STREETLIGHTS           NaN           53   NaN      NaN  Active
3  TDOT_STREETLIGHTS           NaN           54   NaN      NaN  Active
4  TDOT_STREETLIGHTS           NaN           55   NaN      NaN  Active

      MacID TEP_Account_Number  Power_Pedestal_ID
0  00F14C41       3.710003e+09              514.0
1  00F10A57       3.710003e+09              514.0
2  00F16079       3.710003e+09              514.0
3  00F10902       3.710003e+09              514.0
4  00F15E6B       3.710003e+09              514.0

[5 rows x 21 columns]
```

[7]:

```python
print("Income Data Info:")  # Print a header for the neighborhood income data information
print(tuc_neighbourhood_income.info())  # Print the information summary of the neighborhood income DataFrame
print("\nIncome Data Head:")  # Print a header for the first few rows of the neighborhood income data
print(tuc_neighbourhood_income.head())  # Print the first 5 rows of the neighborhood income DataFrame
```

```
Income Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 159 entries, 0 to 158
Columns: 168 entries, OBJECTID to WLTHINDXCY
dtypes: float64(5), int64(157), object(6)
memory usage: 208.8+ KB
None

Income Data Head:
   OBJECTID             NAME  WARD    DATASOURCE  ID sourceCountry  \
0         1       A Mountain     1  NEIGHBORHOODS   0            US
1         2         Adelanto     3  NEIGHBORHOODS   1            US
2         3  Alvernon Heights     5  NEIGHBORHOODS   2            US
3         4            Amphi     3  NEIGHBORHOODS   3            US
4         5      Armory Park     6  NEIGHBORHOODS   4            US

   ENRICH_FID                 aggregationMethod  \
0           1  BlockApportionment:US.BlockGroups
1           2  BlockApportionment:US.BlockGroups
2           3  BlockApportionment:US.BlockGroups
3           4  BlockApportionment:US.BlockGroups
4           5  BlockApportionment:US.BlockGroups

   populationToPolygonSizeRating  apportionmentConfidence  …  AGGDIA75CY  \
0                          2.191                    2.576  …     1590160
1                          2.191                    2.576  …      154598
2                          2.191                    2.576  …      172634
3                          2.191                    2.576  …     2760918
4                          2.191                    2.576  …     3785750

   ID_1  sourceCountry_1  ENRICH_FID_1               aggregationMethod_1  \
0     0               US             1  BlockApportionment:US.BlockGroups
1     1               US             2  BlockApportionment:US.BlockGroups
2     2               US             3  BlockApportionment:US.BlockGroups
3     3               US             4  BlockApportionment:US.BlockGroups
4     4               US             5  BlockApportionment:US.BlockGroups

   populationToPolygonSizeRating_1  apportionmentConfidence_1  HasData_1  \
0                            2.191                      2.576          1
1                            2.191                      2.576          1
2                            2.191                      2.576          1
3                            2.191                      2.576          1
4                            2.191                      2.576          1

   TOTHH_CY  WLTHINDXCY
0      1103          32
1       117          28
2        99          26
```

```
3      3105          20
4      1223          48

[5 rows x 168 columns]
```

[8]:

```
print("Parcel Data Info:")  # Print a header for the parcel data information
print(tuc_parcel_data.info())  # Print the information summary of the parcel␣
↪DataFrame
print("\nParcel Data Head:")  # Print a header for the first few rows of the␣
↪parcel data
tuc_parcel_data.head()  # Display the first 5 rows of the parcel DataFrame
```

```
Parcel Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 443275 entries, 0 to 443274
Data columns (total 50 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   OBJECTID          443275 non-null  int64
 1   PARCEL            443275 non-null  object
 2   GISAREA           443275 non-null  float64
 3   GISACRES          443275 non-null  float64
 4   X_HPGN            443275 non-null  float64
 5   Y_HPGN            443275 non-null  float64
 6   LON               443275 non-null  float64
 7   LAT               443275 non-null  float64
 8   LOT_R             383138 non-null  object
 9   LINK              443275 non-null  object
 10  TRS_OL            443275 non-null  object
 11  MP_OL             353970 non-null  float64
 12  SEQ_NUM_S         388490 non-null  float64
 13  JURIS_OL          443275 non-null  object
 14  CURZONE_OL        443258 non-null  object
 15  ADDRESS_OL        386148 non-null  object
 16  ADR_STATUS        443275 non-null  object
 17  SEQ_NUM_D         439054 non-null  float64
 18  PARCEL_USE        441936 non-null  float64
 19  LANDMEAS          441936 non-null  float64
 20  LANDUNIT          441936 non-null  object
 21  LASTCHANGE        441936 non-null  object
 22  LEGAL1            441936 non-null  object
 23  LEGAL2            141539 non-null  object
 24  LEGAL3            21788 non-null   object
 25  LEGAL4            6727 non-null    object
 26  LEGAL5            11649 non-null   object
 27  LOT               383411 non-null  object
 28  MAIL1             441936 non-null  object
 29  MAIL2             439185 non-null  object
```

```
 30  MAIL3             439092 non-null  object
 31  MAIL4             88858 non-null   object
 32  MAIL5             13060 non-null   object
 33  MP                388545 non-null  object
 34  PAGE              438282 non-null  float64
 35  RECORDDATE        434041 non-null  float64
 36  DOCKET            438282 non-null  float64
 37  RECTRACT          427636 non-null  object
 38  SECTMODIF         236605 non-null  object
 39  TAXAREA           441936 non-null  float64
 40  ZIP               441827 non-null  object
 41  ZIP4              441480 non-null  object
 42  TAXYR             439546 non-null  float64
 43  LIMNET            439546 non-null  float64
 44  FCV               439546 non-null  float64
 45  last_edited_user  443275 non-null  object
 46  last_edited_date  443275 non-null  object
 47  PC_RESTRIC        983 non-null     object
 48  ShapeSTArea       443275 non-null  float64
 49  ShapeSTLength     443275 non-null  float64
dtypes: float64(20), int64(1), object(29)
memory usage: 169.1+ MB
None

Parcel Data Head:
```

[8]:

```
   OBJECTID     PARCEL        GISAREA  GISACRES          X_HPGN          Y_HPGN  \
0         1  10101001D  110172.233186  2.529114  979285.912649  487059.242668
1         2  10101002A  114140.158529  2.620201  979275.589076  486317.677724
2         3  10101003E   94614.544440  2.171971  979098.778243  482696.304361
3         4  10101003L   80326.675549  1.843979  978758.273758  482660.756604
4         5  10101003M   79901.424565  1.834217  978437.135645  482651.278309

          LON        LAT LOT_R                                   LINK  …  \
0 -111.012409  32.335766   NaN  HTTPS://GIS.PIMA.GOV/D.HTM?P=10101001D  …
1 -111.012462  32.333728   NaN  HTTPS://GIS.PIMA.GOV/D.HTM?P=10101002A  …
2 -111.013134  32.323779   NaN  HTTPS://GIS.PIMA.GOV/D.HTM?P=10101003E  …
3 -111.014237  32.323689   NaN  HTTPS://GIS.PIMA.GOV/D.HTM?P=10101003L  …
4 -111.015276  32.323670   NaN  HTTPS://GIS.PIMA.GOV/D.HTM?P=10101003M  …

     ZIP  ZIP4   TAXYR    LIMNET        FCV last_edited_user  \
0  00000  0000  2024.0       0.0     3710.0  u142832@CENTRAL
1  85705  1547  2024.0       0.0  1918109.0  u142832@CENTRAL
2  60015  6002  2024.0  436783.0  2964029.0  u142832@CENTRAL
3  85741  3117  2024.0  330439.0  3220744.0        GISPARFAB
4  85715  3808  2024.0  236550.0  1690713.0        GISPARFAB
```

```
         last_edited_date  PC_RESTRIC    ShapeSTArea  ShapeSTLength
0  2021/01/06 15:20:56+00         NaN  110169.778019    1911.315304
1  2021/01/06 15:20:56+00         NaN  114138.031173    1676.392288
2  2021/01/06 15:20:56+00         NaN   94612.521449    1235.311573
3  2020/10/20 00:14:03+00         NaN   80326.675549    1142.977487
4  2020/10/20 00:49:07+00         NaN   79901.424565    1166.740103

[5 rows x 50 columns]
```

1.2 Data Cleaning and Pre-Processing

1.2.1 Data Cleaning & Preprocessing Overview

This section cleans and prepares four Tucson datasets for spatial analysis:

- **Arrests:** Parsed dates/times, converted age, extracted arrest hour & time period, renamed key fields.
- **Streetlights:** Filtered for active lights, retained location and device info.
- **Neighborhoods:** Selected income & household data, renamed columns for clarity.
- **Parcels:** Cleaned location and address info, converted record dates.

Each dataset was gridded into 1km^2 cells.
Aggregated stats (e.g., arrests, lights, parcels) per grid cell.
Merged with ward-level socioeconomic data.
Calculated densities and added cell center coordinates for spatial mapping.

Final dataset is ready for analysis of crime patterns, infrastructure, and inequality.

1.2.2 Extracting data from arrests

This step prepares the arrests dataset by converting types, extracting time features, and selecting relevant columns for analysis.

[9]:
```python
# Make a copy of the original dataset so we don't change it directly
clean_arrests = tuc_arrest_data.copy()

# Convert the 'age' column to numeric values (invalid ones become NaN)
clean_arrests['age'] = pd.to_numeric(clean_arrests['age'], errors='coerce')

# Convert arrest date to proper datetime format
clean_arrests['arrest_date'] = pd.to_datetime(clean_arrests['date_arr'])

# Copy time and year of arrest into clearer columns
clean_arrests['arrest_time'] = clean_arrests['time_arr']
clean_arrests['arrest_year'] = clean_arrests['YEAR_ARR']

# Extract just the hour from the arrest time (e.g., 1345 → 13)
clean_arrests['arrest_hour'] = clean_arrests['time_arr'].apply(
```

```python
    lambda x: int(str(x).zfill(4)[:2]) if pd.notnull(x) else None
)

# Categorize the arrest time into parts of the day
def categorize_time(hour):
    if pd.isna(hour):
        return 'Unknown'
    if 6 <= hour < 12:
        return 'Morning'
    elif 12 <= hour < 17:
        return 'Afternoon'
    elif 17 <= hour < 22:
        return 'Evening'
    return 'Night'

# Add the new time period column
clean_arrests['time_period'] = clean_arrests['arrest_hour'].
 ↪apply(categorize_time)

# Keep only the columns we care about and rename some for clarity
essential_columns = [
    'OBJECTID', 'X', 'Y', 'date_arr', 'time_arr', 'YEAR_ARR',
    'age', 'race', 'zip', 'sex', 'ethnicity', 'arr_type',
    'arr_chrg', 'chrgdesc', 'WARD', 'NHA_NAME', 'ADDRESS_100BLK',
    'arrest_date', 'arrest_time', 'arrest_year', 'arrest_hour', 'time_period'
]

clean_arrests = clean_arrests[essential_columns].rename(columns={
    'X': 'arrest_x_cord',
    'Y': 'arrest_y_cord',
    'arr_chrg': 'arrest_charge_code',
    'chrgdesc': 'arrest_charge_description',
    'WARD': 'arrest_ward_number',
    'arr_type': 'arrest_type',
    'ADDRESS_100BLK': 'arrest_block_address',
    'NHA_NAME': 'arrest_neighborhood_name',
    'zip': 'arrest_zip'
})

# Show info and a few rows of the cleaned dataset
print("Arrests Data Info:")
print(clean_arrests.info())

print("\nArrests Data Head:")
print(clean_arrests.head())
```

```
Arrests Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45421 entries, 0 to 45420
Data columns (total 22 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   OBJECTID                   45421 non-null  int64
 1   arrest_x_cord              45421 non-null  float64
 2   arrest_y_cord              45421 non-null  float64
 3   date_arr                   45421 non-null  object
 4   time_arr                   45421 non-null  int64
 5   YEAR_ARR                   45421 non-null  int64
 6   age                        44536 non-null  float64
 7   race                       45421 non-null  object
 8   arrest_zip                 43901 non-null  object
 9   sex                        45421 non-null  object
 10  ethnicity                  45421 non-null  object
 11  arrest_type                45421 non-null  object
 12  arrest_charge_code         45421 non-null  object
 13  arrest_charge_description  45421 non-null  object
 14  arrest_ward_number         44591 non-null  float64
 15  arrest_neighborhood_name   32086 non-null  object
 16  arrest_block_address       45416 non-null  object
 17  arrest_date                45421 non-null  datetime64[ns, UTC]
 18  arrest_time                45421 non-null  int64
 19  arrest_year                45421 non-null  int64
 20  arrest_hour                45421 non-null  int64
 21  time_period                45421 non-null  object
dtypes: datetime64[ns, UTC](1), float64(4), int64(6), object(11)
memory usage: 7.6+ MB
None

Arrests Data Head:
   OBJECTID  arrest_x_cord  arrest_y_cord                date_arr  time_arr  \
0         1   9.900089e+05  470751.276735  2021/01/02 00:00:00+00      1731
1         2   9.900089e+05  470751.276735  2021/01/02 00:00:00+00      1731
2         3   9.900089e+05  470751.276735  2021/01/02 00:00:00+00      1731
3         4   1.053154e+06  443419.380064  2021/01/02 00:00:00+00      1844
4         5   1.053154e+06  443419.380064  2021/01/02 00:00:00+00      1844

   YEAR_ARR   age race  arrest_zip sex  …         arrest_charge_code  \
0      2021  28.0    H  85705        M  …  13-3613A
1      2021  28.0    H  85705        M  …  13-3613A
2      2021  28.0    H  85705        M  …  13-3613A
3      2021  15.0    B  85748        M  …  13-1203A2DV
4      2021  15.0    B  85748        M  …  13-2904A1DV

                           arrest_charge_description arrest_ward_number  \
```

```
0  CONTRIBUTE TO DELINQUENCY/DEPENDENCY OF MINOR …               3.0
1  CONTRIBUTE TO DELINQUENCY/DEPENDENCY OF MINOR …               3.0
2  CONTRIBUTE TO DELINQUENCY/DEPENDENCY OF MINOR …               3.0
3  ASSAULT-CAUSE FEAR OF PHYSICAL INJURY (DOMESTI…               2.0
4  DISORDERLY CONDUCT/DV                          …               2.0

  arrest_neighborhood_name      arrest_block_address  \
0                      NaN          4598 N ORACLE RD
1                      NaN          4598 N ORACLE RD
2                      NaN          4598 N ORACLE RD
3                 Eastside  10198 E ESSEX VILLAGE DR
4                 Eastside  10198 E ESSEX VILLAGE DR

                 arrest_date arrest_time arrest_year  arrest_hour  time_period
0 2021-01-02 00:00:00+00:00        1731        2021           17      Evening
1 2021-01-02 00:00:00+00:00        1731        2021           17      Evening
2 2021-01-02 00:00:00+00:00        1731        2021           17      Evening
3 2021-01-02 00:00:00+00:00        1844        2021           18      Evening
4 2021-01-02 00:00:00+00:00        1844        2021           18      Evening

[5 rows x 22 columns]
```

1.2.3 Streetlights Dataset Cleaning

This step filters only active streetlights and selects relevant location and device information for further analysis.

[10]:

```
# Clean and filter the streetlights dataset

try:
    # Make a copy of the original dataset
    clean_lights = tuc_light_data.copy()

    # Keep only the rows where the streetlight is active
    clean_lights = clean_lights[clean_lights['Status'] == 'Active']

    # Choose the important columns we want to keep
    essential_columns = [
        'X', 'Y',
        'Address_Number',
        'Street',
        'Status',
        'MacID'
    ]

    # Rename columns to more descriptive names
    clean_lights = clean_lights[essential_columns].rename(columns={
        'X': 'light_location_x',
```

```python
        'Y': 'light_location_y',
        'Address_Number': 'light_address_number',
        'Street': 'light_street_name',
        'Status': 'light_operational_status',
        'MacID': 'light_device_id'
    })

except Exception as e:
    # Show the error if something goes wrong
    print(f"An error occurred: {str(e)}")
    import traceback
    traceback.print_exc()
# Show info and a few rows of the cleaned dataset
print("Arrests Data Info:")
print(clean_lights.info())

print("\nArrests Data Head:")
print(clean_lights.head())
```

```
Arrests Data Info:
<class 'pandas.core.frame.DataFrame'>
Index: 22452 entries, 0 to 22780
Data columns (total 6 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   light_location_x          22452 non-null  float64
 1   light_location_y          22452 non-null  float64
 2   light_address_number      18265 non-null  float64
 3   light_street_name         19221 non-null  object
 4   light_operational_status  22452 non-null  object
 5   light_device_id           19510 non-null  object
dtypes: float64(3), object(3)
memory usage: 1.2+ MB
None

Arrests Data Head:
   light_location_x  light_location_y  light_address_number light_street_name  \
0      1.001233e+06     421018.579396                5425.0      S Campbell Av
1      1.001142e+06     420902.066601                5434.0      S Campbell Av
2      1.001234e+06     420785.376969                5441.0      S Campbell Av
3      1.001143e+06     420667.403543                5454.0      S Campbell Av
4      1.001237e+06     420582.028543                5457.0      S Campbell Av

  light_operational_status light_device_id
0                   Active        00F14C41
1                   Active        00F10A57
2                   Active        00F16079
```

```
3                    Active        00F10902
4                    Active        00F15E6B
```

1.2.4 Neighborhood Income Data Cleaning

This step extracts key socioeconomic indicators from the neighborhood dataset and renames columns for clarity.

[11]:
```
# Clean and simplify the neighborhood income dataset

try:
    # Make a copy of the original data to keep it unchanged
    clean_neighborhoods = tuc_neighbourhood_income.copy()

    # Keep only the important columns with economic and location data
    essential_columns = [
        'NAME',
        'WARD',
        'MEDHINC_CY',
        'AVGHINC_CY',
        'PCI_CY',
        'TOTHH_CY',
        'WLTHINDXCY'
    ]

    # Rename the columns to more readable names
    clean_neighborhoods = clean_neighborhoods[essential_columns].
 ↪rename(columns={
        'NAME': 'neigh_full_name',
        'WARD': 'neigh_ward_number',
        'MEDHINC_CY': 'neigh_median_household_income',
        'AVGHINC_CY': 'neigh_average_household_income',
        'PCI_CY': 'neigh_per_capita_income',
        'TOTHH_CY': 'neigh_total_households',
        'WLTHINDXCY': 'neigh_wealth_index'
    })

except Exception as e:
    # Show error message if anything fails
    print(f"An error occurred: {str(e)}")
    import traceback
    traceback.print_exc()

# Show info and a few rows of the cleaned dataset
print("Arrests Data Info:")
print(clean_neighborhoods.info())

print("\nArrests Data Head:")
```

```python
print(clean_neighborhoods.head())
```

```
Arrests Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 159 entries, 0 to 158
Data columns (total 7 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   neigh_full_name                159 non-null    object
 1   neigh_ward_number              159 non-null    int64
 2   neigh_median_household_income  159 non-null    int64
 3   neigh_average_household_income 159 non-null    int64
 4   neigh_per_capita_income        159 non-null    int64
 5   neigh_total_households         159 non-null    int64
 6   neigh_wealth_index             159 non-null    int64
dtypes: int64(6), object(1)
memory usage: 8.8+ KB
None

Arrests Data Head:
    neigh_full_name  neigh_ward_number  neigh_median_household_income  \
0        A Mountain                  1                          39293
1          Adelanto                  3                          33635
2  Alvernon Heights                  5                          29762
3             Amphi                  3                          20213
4       Armory Park                  6                          36870

   neigh_average_household_income  neigh_per_capita_income  \
0                           47471                    15189
1                           41101                    11776
2                           38963                    12634
3                           31338                    13040
4                           62259                    37424

   neigh_total_households  neigh_wealth_index
0                    1103                  32
1                     117                  28
2                      99                  26
3                    3105                  20
4                    1223                  48
```

1.2.5 Parcel Data Cleaning

This step extracts relevant location and address information from the parcel dataset and ensures date values are properly formatted.

[12]:
```python
# Clean and prepare the parcel dataset

try:
    # Make a copy of the original data to avoid changing it
    clean_parcels = tuc_parcel_data.copy()

    # Keep only the important columns with location, address, and date info
    essential_columns = [
        'X_HPGN',
        'Y_HPGN',
        'ADDRESS_OL',
        'MAIL2',
        'RECORDDATE',
        'ZIP'
    ]

    # Rename columns to be easier to understand
    clean_parcels = clean_parcels[essential_columns].rename(columns={
        'X_HPGN': 'parcel_location_x',
        'Y_HPGN': 'parcel_location_y',
        'ADDRESS_OL': 'parcel_official_address',
        'MAIL2': 'parcel_mailing_address',
        'RECORDDATE': 'parcel_record_date',
        'ZIP': 'parcel_zipcode'
    })

    # Convert record date to datetime format
    clean_parcels['parcel_record_date'] = pd.
 ↪to_datetime(clean_parcels['parcel_record_date'], errors='coerce')

except Exception as e:
    # Show an error message if something goes wrong
    print(f"An error occurred: {str(e)}")
    import traceback
    traceback.print_exc()
```

1.2.6 Grid System, Merging, and Validation

This section performs spatial aggregation and merging of all cleaned datasets:

- **create_grid(df, x_col, y_col, cell_size):**
  Splits the spatial data into uniform square grid cells (default 1km x 1km). Assigns each record a grid cell ID.
- **calc_cell(merged_df, cell_size):**
  Calculates the geographic center (X, Y) of each grid cell based on its indices.
- **merge_datasets(arrests_df, lights_df, neighborhoods_df, parcels_df, cell_size):**
  Combines all gridded datasets by:

  - Aggregating totals (arrests, lights, parcels) per cell
  - Merging neighborhood info by ward
  - Calculating density per square km
  - Adding spatial centroids for mapping and visualization
- **validate(df):**
  Returns summary statistics like total grid cells, ward counts, and mean density values for quick data validation.

[13]:
```python
# Create a grid system by dividing spatial coordinates into uniform square cells
def create_grid(df, x_col, y_col, cell_size=1000):
    """Divide the area into grid cells based on X and Y coordinates."""
    df_grid = df.copy()

    # Get the full range of coordinates
    x_min, x_max = df_grid[x_col].min(), df_grid[x_col].max()
    y_min, y_max = df_grid[y_col].min(), df_grid[y_col].max()

    # Define the edges of each grid cell
    x_edges = np.arange(x_min, x_max + cell_size, cell_size)
    y_edges = np.arange(y_min, y_max + cell_size, cell_size)

    # Assign each point to a grid cell based on its X and Y position
    df_grid['grid_x_index'] = pd.cut(df_grid[x_col], bins=x_edges,␣
↪labels=range(len(x_edges) - 1))
    df_grid['grid_y_index'] = pd.cut(df_grid[y_col], bins=y_edges,␣
↪labels=range(len(y_edges) - 1))

    # Create a unique ID for each grid cell
    df_grid['grid_cell_id'] = df_grid['grid_x_index'].astype(str) + '_' +␣
↪df_grid['grid_y_index'].astype(str)

    return df_grid

# Calculate the center (X, Y) coordinates for each grid cell
def calc_cell(merged_df, cell_size=1000):
    def extract_coordinates(row):
        try:
            x_idx, y_idx = map(int, row['grid_cell_id'].split('_'))
            return pd.Series({
                'cell_center_x': x_idx * cell_size + cell_size / 2,
                'cell_center_y': y_idx * cell_size + cell_size / 2
            })
        except (ValueError, AttributeError):
            return pd.Series({'cell_center_x': np.nan, 'cell_center_y': np.nan})

    coordinates = merged_df.apply(extract_coordinates, axis=1)
    merged_df['cell_center_x'] = coordinates['cell_center_x']
```

```
    merged_df['cell_center_y'] = coordinates['cell_center_y']

    return merged_df

# Merge all datasets together and calculate grid-level statistics
def merge_datasets(arrests_df, lights_df, neighborhoods_df, parcels_df,␣
 ↪cell_size=1000):

    print("Creating grid systems for each dataset...")
    processed_arrests = create_grid(arrests_df, 'arrest_x_cord',␣
 ↪'arrest_y_cord')
    gridded_lights = create_grid(lights_df, 'light_location_x',␣
 ↪'light_location_y')
    gridded_parcels = create_grid(parcels_df, 'parcel_location_x',␣
 ↪'parcel_location_y')

    print("Calculating arrest statistics...")
    arrest_agg = processed_arrests.groupby('grid_cell_id').agg({
        'OBJECTID': 'count',
        'age': 'mean',
        'arrest_type': lambda x: x.mode().iloc[0] if not x.empty else None
    }).reset_index().rename(columns={
        'OBJECTID': 'total_arrests',
        'age': 'average_arrestee_age',
        'arrest_type': 'common_arrest_type'
    })

    print("Calculating light statistics...")
    light_agg = gridded_lights.groupby('grid_cell_id').agg({
        'light_device_id': 'count'
    }).reset_index().rename(columns={'light_device_id': 'total_streetlights'})

    print("Calculating parcel statistics...")
    parcel_agg = gridded_parcels.groupby('grid_cell_id').agg({
        'parcel_official_address': 'count'
    }).reset_index().rename(columns={'parcel_official_address':␣
 ↪'total_parcels'})

    print("Merging aggregated statistics...")
    merged = pd.merge(arrest_agg, light_agg, on='grid_cell_id', how='outer')
    merged = pd.merge(merged, parcel_agg, on='grid_cell_id', how='outer')

    # Add ward info and neighborhood data
    distinct_ward = processed_arrests[['grid_cell_id', 'arrest_ward_number']].
 ↪drop_duplicates()
    merged = pd.merge(merged, distinct_ward, on='grid_cell_id', how='left')
```

```
    merged = pd.merge(merged, neighborhoods_df, left_on='arrest_ward_number',␣
↪right_on='neigh_ward_number', how='left')

    # Calculate density values for each cell
    cell_area_sqkm = (cell_size * cell_size) / 1_000_000
    merged['arrests_per_sqkm'] = merged['total_arrests'] / cell_area_sqkm
    merged['streetlights_per_sqkm'] = merged['total_streetlights'] /␣
↪cell_area_sqkm
    merged['parcels_per_sqkm'] = merged['total_parcels'] / cell_area_sqkm

    # Fill any missing data with 0
    merged = merged.fillna({
        'total_arrests': 0,
        'total_streetlights': 0,
        'total_parcels': 0,
        'arrests_per_sqkm': 0,
        'streetlights_per_sqkm': 0,
        'parcels_per_sqkm': 0
    })

    # Add X/Y center coordinates for each grid cell
    merged = calc_cell(merged, cell_size)

    # Merge back the full original datasets for spatial context
    final_merged = pd.merge(merged, processed_arrests, on='grid_cell_id',␣
↪how='left', suffixes=('', '_arrestorig'))
    final_merged = pd.merge(final_merged, gridded_lights, on='grid_cell_id',␣
↪how='left', suffixes=('', '_lightorig'))
    final_merged = pd.merge(final_merged, gridded_parcels, on='grid_cell_id',␣
↪how='left', suffixes=('', '_parcelorig'))

    return final_merged

# Validate merged dataset by summarizing key metrics
def validate(df):
    """Check key metrics from the final dataset to ensure data is valid."""
    validation_results = {
        'total_grid_cells': len(df['grid_cell_id'].unique()),
        'unique_wards': df['arrest_ward_number'].nunique(),
        'total_arrests_sum': df['total_arrests'].sum(),
        'total_streetlights_sum': df['total_streetlights'].sum(),
        'total_parcels_sum': df['total_parcels'].sum(),
        'ward_statistics': df.groupby('arrest_ward_number').agg({
            'arrests_per_sqkm': 'mean',
            'streetlights_per_sqkm': 'mean',
            'parcels_per_sqkm': 'mean',
            'neigh_median_household_income': 'first',
```

```
            'neigh_wealth_index': 'first'
        }).head()
    }
    return validation_results
```

[14]:
```
try:
    # Merge all datasets into one final dataset based on grid cells
    final_dataset = merge_datasets(
        clean_arrests, clean_lights, clean_neighborhoods, clean_parcels,␣
 ↪cell_size=1000
    )

    # If merge was successful, display dataset structure and preview rows
    if final_dataset is not None:
        print("\nFinal Dataset Columns:")
        print(final_dataset.columns.tolist())  # List all column names

        print("\nData Sample:")
        print(final_dataset.head())  # Show the first few rows

except NameError:

    print("Please ensure all required datasets are loaded properly.")
```

```
Creating grid systems for each dataset…
Calculating arrest statistics…
Calculating light statistics…
Calculating parcel statistics…
Merging aggregated statistics…

Final Dataset Columns:
['grid_cell_id', 'total_arrests', 'average_arrestee_age', 'common_arrest_type',
'total_streetlights', 'total_parcels', 'arrest_ward_number', 'neigh_full_name',
'neigh_ward_number', 'neigh_median_household_income',
'neigh_average_household_income', 'neigh_per_capita_income',
'neigh_total_households', 'neigh_wealth_index', 'arrests_per_sqkm',
'streetlights_per_sqkm', 'parcels_per_sqkm', 'cell_center_x', 'cell_center_y',
'OBJECTID', 'arrest_x_cord', 'arrest_y_cord', 'date_arr', 'time_arr',
'YEAR_ARR', 'age', 'race', 'arrest_zip', 'sex', 'ethnicity', 'arrest_type',
'arrest_charge_code', 'arrest_charge_description',
'arrest_ward_number_arrestorig', 'arrest_neighborhood_name',
'arrest_block_address', 'arrest_date', 'arrest_time', 'arrest_year',
'arrest_hour', 'time_period', 'grid_x_index', 'grid_y_index',
'light_location_x', 'light_location_y', 'light_address_number',
'light_street_name', 'light_operational_status', 'light_device_id',
'grid_x_index_lightorig', 'grid_y_index_lightorig', 'parcel_location_x',
'parcel_location_y', 'parcel_official_address', 'parcel_mailing_address',
```

```
'parcel_record_date', 'parcel_zipcode', 'grid_x_index_parcelorig',
'grid_y_index_parcelorig']

Data Sample:
  grid_cell_id  total_arrests  average_arrestee_age common_arrest_type  \
0        0_275            0.0                   NaN                NaN
1        0_307            0.0                   NaN                NaN
2         0_83            0.0                   NaN                NaN
3         0_83            0.0                   NaN                NaN
4         0_83            0.0                   NaN                NaN

   total_streetlights  total_parcels  arrest_ward_number neigh_full_name  \
0                 0.0            0.0                 NaN             NaN
1                 0.0            0.0                 NaN             NaN
2                 0.0            0.0                 NaN             NaN
3                 0.0            0.0                 NaN             NaN
4                 0.0            0.0                 NaN             NaN

   neigh_ward_number  neigh_median_household_income  …  \
0                NaN                            NaN  …
1                NaN                            NaN  …
2                NaN                            NaN  …
3                NaN                            NaN  …
4                NaN                            NaN  …

   grid_x_index_lightorig  grid_y_index_lightorig  parcel_location_x  \
0                     NaN                     NaN      276311.616198
1                     NaN                     NaN      276734.228534
2                       0                      83                NaN
3                       0                      83                NaN
4                       0                      83                NaN

   parcel_location_y  parcel_official_address  \
0      439392.248369                      NaN
1      471041.213893                      NaN
2                NaN                      NaN
3                NaN                      NaN
4                NaN                      NaN

                    parcel_mailing_address  parcel_record_date  \
0  (CABEZA PRIETA NATIONAL WILDLIFE REFUGE)                 NaT
1  (CABEZA PRIETA NATIONAL WILDLIFE REFUGE)                 NaT
2                                      NaN                 NaT
3                                      NaN                 NaT
4                                      NaN                 NaT

   parcel_zipcode  grid_x_index_parcelorig  grid_y_index_parcelorig
0           00000                        0                      275
```

```
1          00000                    0                  307
2            NaN                  NaN                  NaN
3            NaN                  NaN                  NaN
4            NaN                  NaN                  NaN

[5 rows x 59 columns]
```

---

2 Data Analysis and Visualization

2.0.1 Property Crime Analysis: Night vs Day

This section analyzes property crimes based on time of day and income levels:

- Identifies property crime records using keyword matching.
- Separates crimes into night and day.
- Aggregates crime counts per grid cell.
- Calculates log-transformed rates for better analysis.
- Bins neighborhoods into income quartiles.
- Computes correlations and summary stats.
- Visualizes crime patterns using KDE plots, violin plots, regression plots, and heatmaps.

[15]:
```
# Create a copy of the final dataset to work with
df = final_dataset.copy()

# Define keywords that indicate property crimes
property_crime_keywords = [
    "burglary", "theft", "larceny", "vandalism", "breaking", "robbery",
    "shoplifting", "trespass", "criminal damage", "property", "tamper",
    "organized retail", "purse snatch", "trafficking stolen property"
]

# Mark rows as property crimes if the description contains any of the keywords
df['is_property_crime'] = df['arrest_charge_description'].str.lower().str.
↪contains('|'.join(property_crime_keywords), na=False)

# Filter only the rows that are property crimes
df_prop = df[df['is_property_crime']].copy()

# Separate property crimes into night and day categories
df_prop_night = df_prop[df_prop['time_period'] == 'Night']
df_prop_day = df_prop[df_prop['time_period'] != 'Night']

# Count the number of property crimes per grid cell for night and day
night_crime_counts = df_prop_night.groupby('grid_cell_id').size().
↪reset_index(name='night_property_crime_count')
```

```
day_crime_counts = df_prop_day.groupby('grid_cell_id').size().
 ↪reset_index(name='day_property_crime_count')

# Merge these counts back into the main dataframe
df = df.merge(night_crime_counts, on='grid_cell_id', how='left')
df = df.merge(day_crime_counts, on='grid_cell_id', how='left')

# Replace any missing values with 0 to avoid NaNs in analysis
df['night_property_crime_count'] = df['night_property_crime_count'].fillna(0)
df['day_property_crime_count'] = df['day_property_crime_count'].fillna(0)

# Create crime-per-sqkm fields for normalization
df['night_property_crimes_per_sqkm'] = df['night_property_crime_count']
df['day_property_crimes_per_sqkm'] = df['day_property_crime_count']

# Log-transform the values to reduce skewness and handle zero values
df['log_night_property_crimes_per_sqkm'] = np.
 ↪log1p(df['night_property_crimes_per_sqkm'])
df['log_day_property_crimes_per_sqkm'] = np.
 ↪log1p(df['day_property_crimes_per_sqkm'])
df['log_streetlights_per_sqkm'] = np.log1p(df['streetlights_per_sqkm'])

# Filter dataset to only include rows with valid income data
df_income = df.dropna(subset=['neigh_median_household_income']).copy()

# Create income groups based on quartiles
df_income['income_bin'] = pd.qcut(df_income['neigh_median_household_income'],␣
 ↪q=4,
                                  labels=['Low', 'Medium-Low', 'Medium-High',␣
 ↪'High'])

# Compute correlation matrices for day and night
corr_vars_night = ['night_property_crimes_per_sqkm', 'streetlights_per_sqkm',␣
 ↪'neigh_median_household_income']
corr_matrix_night = df[corr_vars_night].corr()

corr_vars_day = ['day_property_crimes_per_sqkm', 'streetlights_per_sqkm',␣
 ↪'neigh_median_household_income']
corr_matrix_day = df[corr_vars_day].corr()

# Display correlation matrices
print("Correlation Matrix (Day):\n", corr_matrix_day)
print("Correlation Matrix (Night):\n", corr_matrix_night)

# Get summary statistics of property crimes by income level (night and day)
```

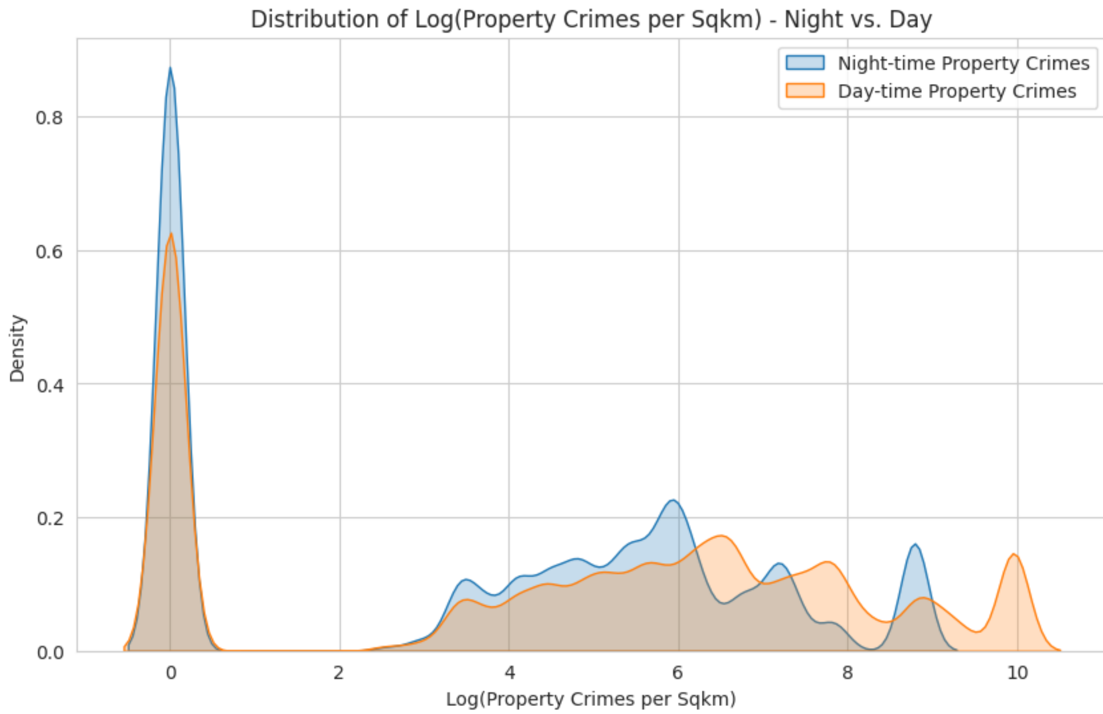
```
income_group_stats_night = df_income.groupby('income_bin',␣
↪observed=True)['night_property_crimes_per_sqkm'].agg(['mean', 'median',␣
↪'std', 'count'])
print("Property Crime Stats by Income Level (Night):\n",␣
↪income_group_stats_night)

income_group_stats_day = df_income.groupby('income_bin',␣
↪observed=True)['day_property_crimes_per_sqkm'].agg(['mean', 'median', 'std',␣
↪'count'])
print("Property Crime Stats by Income Level (Day):\n", income_group_stats_day)

# Plotting the Data


sns.set_style('whitegrid')

# 1. KDE Plot: Compare distributions of day vs night property crimes
print("\n1. KDE distributions comparing nighttime and daytime property␣
↪crimes\n")
plt.figure(figsize=(10,6))
sns.kdeplot(df['log_night_property_crimes_per_sqkm'], fill=True,␣
↪label='Night-time Property Crimes')
sns.kdeplot(df['log_day_property_crimes_per_sqkm'], fill=True, label='Day-time␣
↪Property Crimes')
plt.title("Distribution of Log(Property Crimes per Sqkm) - Night vs. Day")
plt.xlabel("Log(Property Crimes per Sqkm)")
plt.ylabel("Density")
plt.legend()
plt.show()

# 2. Violin Plots: Show crime variation by income bin
print("\n2. Violin plots by income group for night and day property crimes␣
↪separately\n")
plt.figure(figsize=(10,6))
sns.violinplot(x='income_bin', y='log_night_property_crimes_per_sqkm',␣
↪data=df_income, inner='quartile')
plt.title("Log(Night-time Property Crimes per Sqkm) by Income Level")
plt.xlabel("Income Level")
plt.ylabel("Log(Night-time Property Crimes per Sqkm)")
plt.show()

plt.figure(figsize=(10,6))
sns.violinplot(x='income_bin', y='log_day_property_crimes_per_sqkm',␣
↪data=df_income, inner='quartile')
plt.title("Log(Day-time Property Crimes per Sqkm) by Income Level")
plt.xlabel("Income Level")
```

```
plt.ylabel("Log(Day-time Property Crimes per Sqkm)")
plt.show()

# 3. Regression Plots: Visualize streetlight-crime relationship
print("\n3. Regression plots: Streetlights vs. Property Crimes (Night vs␣
↪Day)\n")
fig, axes = plt.subplots(1, 2, figsize=(14,6))
sns.regplot(x='log_streetlights_per_sqkm',␣
↪y='log_night_property_crimes_per_sqkm',
            data=df, scatter_kws={'alpha':0.3}, line_kws={"color":"red"},␣
↪ax=axes[0])
axes[0].set_title("Night-time Property Crimes vs. Streetlights")
axes[0].set_xlabel("Log(Streetlights per Sqkm)")
axes[0].set_ylabel("Log(Night-time Property Crimes per Sqkm)")

sns.regplot(x='log_streetlights_per_sqkm', y='log_day_property_crimes_per_sqkm',
            data=df, scatter_kws={'alpha':0.3}, line_kws={"color":"red"},␣
↪ax=axes[1])
axes[1].set_title("Day-time Property Crimes vs. Streetlights")
axes[1].set_xlabel("Log(Streetlights per Sqkm)")
axes[1].set_ylabel("Log(Day-time Property Crimes per Sqkm)")
plt.tight_layout()
plt.show()

# 4. Correlation Heatmaps: Show how variables relate for day and night
print("\n4. Heatmaps for nighttime and daytime correlations\n")
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
sns.heatmap(corr_matrix_night, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title("Night-time Correlation")

plt.subplot(1,2,2)
sns.heatmap(corr_matrix_day, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title("Day-time Correlation")
plt.tight_layout()
plt.show()
```

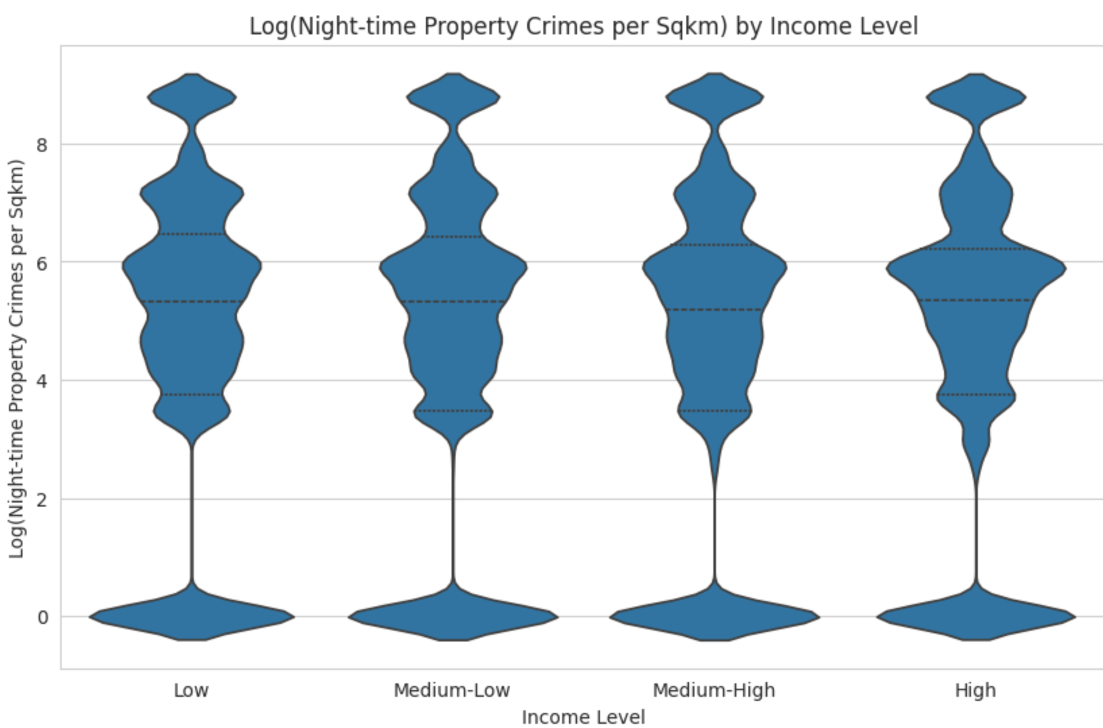
```
Correlation Matrix (Day):
                                day_property_crimes_per_sqkm  \
day_property_crimes_per_sqkm                        1.000000
streetlights_per_sqkm                              -0.035513
neigh_median_household_income                      -0.004157

                               streetlights_per_sqkm  \
day_property_crimes_per_sqkm               -0.035513
streetlights_per_sqkm                       1.000000
neigh_median_household_income                    NaN
```

```
                               neigh_median_household_income
day_property_crimes_per_sqkm                        -0.004157
streetlights_per_sqkm                                     NaN
neigh_median_household_income                        1.000000
Correlation Matrix (Night):
                                 night_property_crimes_per_sqkm  \
night_property_crimes_per_sqkm                         1.000000
streetlights_per_sqkm                                 -0.034470
neigh_median_household_income                          0.004735

                                streetlights_per_sqkm  \
night_property_crimes_per_sqkm               -0.03447
streetlights_per_sqkm                         1.00000
neigh_median_household_income                     NaN

                                neigh_median_household_income
night_property_crimes_per_sqkm                       0.004735
streetlights_per_sqkm                                     NaN
neigh_median_household_income                        1.000000
Property Crime Stats by Income Level (Night):
                    mean  median          std   count
income_bin
Low          866.835781   205.0  1740.061168  420774
Medium-Low   900.570956   205.0  1796.785657  425378
Medium-High  900.719244   180.0  1834.084424  412971
High         907.952683   210.0  1828.191778  414015
Property Crime Stats by Income Level (Day):
                     mean  median          std   count
income_bin
Low          2906.879643   527.0  5738.318298  420774
Medium-Low   3208.164477   513.0  5992.092039  425378
Medium-High  3062.583428   472.0  6050.925476  412971
High         3020.792964   539.0  6014.906221  414015
```

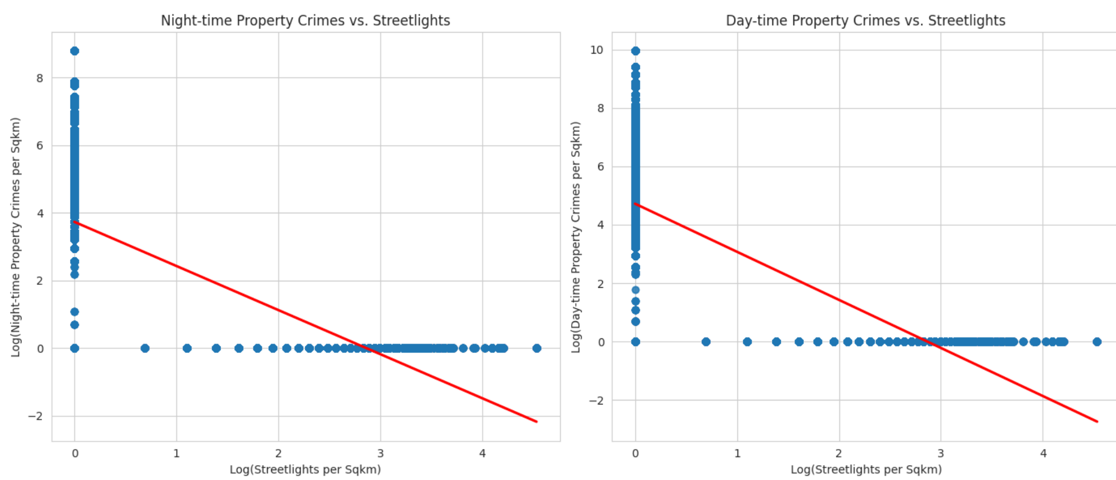
1. KDE distributions comparing nighttime and daytime property crimes



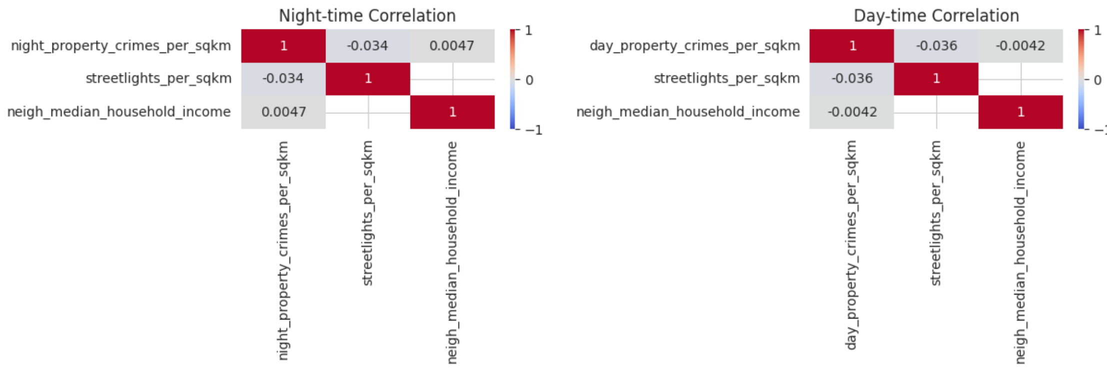
2. Violin plots by income group for night and day property crimes separately



3. Regression plots: Streetlights vs. Property Crimes (Night vs Day)



4. Heatmaps for nighttime and daytime correlations



2.1 Predictive Modeling for Property Crime Analysis

This section uses machine learning to test whether streetlight density and neighborhood income are predictive of property crime rates and risk levels. We build two types of models for both daytime and nighttime:

2.1.1 1 Regression Models (Linear Regression)

Purpose:
To **predict the continuous value** of property crime rates (log-transformed per square kilometer) based on environmental and socioeconomic features.

Target Variables: - `log_night_property_crimes_per_sqkm` - `log_day_property_crimes_per_sqkm`

Input Features: - `streetlights_per_sqkm` – Number of streetlights in a grid cell. - `neigh_median_household_income` – Median income for the neighborhood. - `parcels_per_sqkm` – A proxy for parcel/building density.

Model Used:
`LinearRegression()` from scikit-learn – a simple, interpretable model that assumes a linear relationship between features and the target variable.

Evaluation Metric:
Mean Squared Error (MSE) – Lower values indicate better model performance by measuring the average squared difference between predicted and actual crime rates.

2.1.2 2 Classification Models (Random Forest Classifier)

Purpose:
To **classify each grid cell** as either **High Crime Risk** or **Low Crime Risk** based on whether crime rates are above or below the median value. This helps in identifying hotspots.

Target Variables: - `crime_risk_night` (1 = above median, 0 = below median) - `crime_risk_day` (1 = above median, 0 = below median)

Input Features: Same as regression: - `streetlights_per_sqkm` - `neigh_median_household_income` - `parcels_per_sqkm`

Model Used:
`RandomForestClassifier()` – an ensemble-based classifier that combines multiple decision trees to improve accuracy and robustness. It also allows us to extract **feature importance**.

Evaluation Metric:
`classification_report()` – Shows precision, recall, f1-score, and support for both classes. Helps assess how well the model distinguishes between high and low crime areas.

2.1.3 Feature Importance Analysis

After training the Random Forest Classifiers, we compute and visualize feature importance to understand: - Which variables most strongly influence crime risk predictions. - Whether streetlight density or income level has more predictive power.

Separate plots are generated for nighttime and daytime models, making it easier to compare their behavior under different conditions.

2.1.4 Summary of Steps

1. Prepare input features and target labels for both regression and classification tasks.
2. Split data into training (80%) and testing (20%) sets.
3. Train and evaluate Linear Regression for predicting crime rates.
4. Train and evaluate Random Forest Classifier for predicting crime risk levels.
5. Visualize feature importance to interpret model behavior.

This approach gives us both **quantitative predictions** and **categorical classifications**, which are useful for planning interventions, resource allocation, and urban safety improvements.

3 Model Implementation

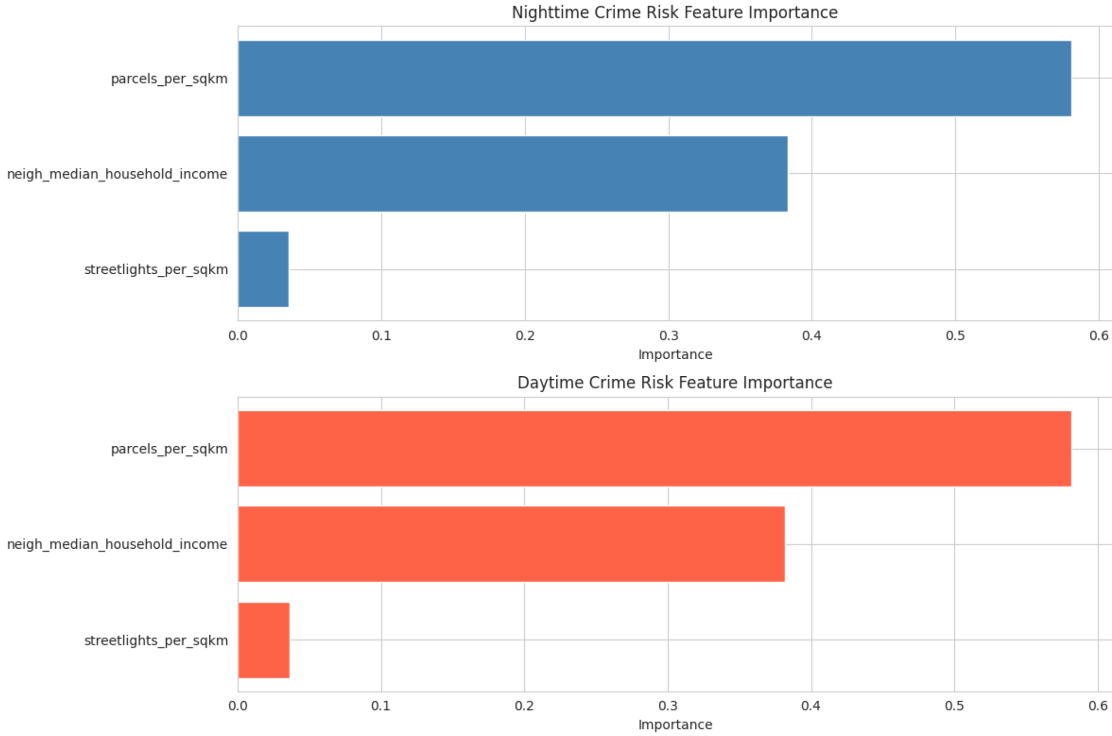
[16]:
```
from sklearn.impute import SimpleImputer

# Copy original data
df_clean = df.copy()

# Create binary labels for high crime risk
df_clean['risk_night'] = np.
 ↪where(df_clean['log_night_property_crimes_per_sqkm'] >␣
 ↪df_clean['log_night_property_crimes_per_sqkm'].median(), 1, 0)
df_clean['risk_day'] = np.where(df_clean['log_day_property_crimes_per_sqkm'] >␣
 ↪df_clean['log_day_property_crimes_per_sqkm'].median(), 1, 0)
```

```
# Define features and targets
features = ['streetlights_per_sqkm', 'neigh_median_household_income', 'parcels_per_sqkm']
target_log_night = 'log_night_property_crimes_per_sqkm'
target_log_day = 'log_day_property_crimes_per_sqkm'
target_risk_night = 'risk_night'
target_risk_day = 'risk_day'

# Handle missing values
imputer = SimpleImputer(strategy='median')
X = imputer.fit_transform(df_clean[features])

# Targets
y_log_night = df_clean[target_log_night]
y_log_day = df_clean[target_log_day]
y_cls_night = df_clean[target_risk_night]
y_cls_day = df_clean[target_risk_day]

# Train-test splits
X_train_night, X_test_night, y_train_log_night, y_test_log_night = train_test_split(X, y_log_night, test_size=0.2, random_state=42)
X_train_day, X_test_day, y_train_log_day, y_test_log_day = train_test_split(X, y_log_day, test_size=0.2, random_state=42)
X_train_cls_night, X_test_cls_night, y_train_cls_night, y_test_cls_night = train_test_split(X, y_cls_night, test_size=0.2, random_state=42)
X_train_cls_day, X_test_cls_day, y_train_cls_day, y_test_cls_day = train_test_split(X, y_cls_day, test_size=0.2, random_state=42)

# --- Nighttime Models ---
print("### NIGHTTIME MODELS ###")

# Linear Regression
lr_night = LinearRegression()
lr_night.fit(X_train_night, y_train_log_night)
y_pred_night = lr_night.predict(X_test_night)
mse_night = mean_squared_error(y_test_log_night, y_pred_night)
print(f"Nighttime MSE (Linear Regression): {mse_night:.4f}")

# Random Forest Classifier
rf_night = RandomForestClassifier(n_estimators=100, random_state=42)
rf_night.fit(X_train_cls_night, y_train_cls_night)
y_pred_cls_night = rf_night.predict(X_test_cls_night)
print("Nighttime Crime Risk Classification Report:")
print(classification_report(y_test_cls_night, y_pred_cls_night))

# Feature importance
```

```python
importance_night = pd.DataFrame({
    'Feature': features,
    'Importance': rf_night.feature_importances_
}).sort_values(by='Importance', ascending=False)

# --- Daytime Models ---
print("\n### DAYTIME MODELS ###")

# Linear Regression
lr_day = LinearRegression()
lr_day.fit(X_train_day, y_train_log_day)
y_pred_day = lr_day.predict(X_test_day)
mse_day = mean_squared_error(y_test_log_day, y_pred_day)
print(f"Daytime MSE (Linear Regression): {mse_day:.4f}")

# Random Forest Classifier
rf_day = RandomForestClassifier(n_estimators=100, random_state=42)
rf_day.fit(X_train_cls_day, y_train_cls_day)
y_pred_cls_day = rf_day.predict(X_test_cls_day)
print("Daytime Crime Risk Classification Report:")
print(classification_report(y_test_cls_day, y_pred_cls_day))

# Feature importance
importance_day = pd.DataFrame({
    'Feature': features,
    'Importance': rf_day.feature_importances_
}).sort_values(by='Importance', ascending=False)

# --- Visualization ---
plt.figure(figsize=(12, 8))

plt.subplot(2, 1, 1)
plt.barh(importance_night['Feature'], importance_night['Importance'],␣
 ↪color='steelblue')
plt.title("Nighttime Crime Risk Feature Importance")
plt.xlabel("Importance")
plt.gca().invert_yaxis()

plt.subplot(2, 1, 2)
plt.barh(importance_day['Feature'], importance_day['Importance'],␣
 ↪color='tomato')
plt.title("Daytime Crime Risk Feature Importance")
plt.xlabel("Importance")
plt.gca().invert_yaxis()

plt.tight_layout()
plt.show()
```

```python
# Print feature importances
print("\nFeature Importances - Night:")
print(importance_night)

print("\nFeature Importances - Day:")
print(importance_day)
```

```
### NIGHTTIME MODELS ###
Nighttime MSE (Linear Regression): 6.9393
Nighttime Crime Risk Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.56      0.67    214838
           1       0.67      0.88      0.76    213301

    accuracy                           0.72    428139
   macro avg       0.75      0.72      0.71    428139
weighted avg       0.75      0.72      0.71    428139


### DAYTIME MODELS ###
Daytime MSE (Linear Regression): 7.4918
Daytime Crime Risk Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.62      0.69    214654
           1       0.68      0.82      0.75    213485

    accuracy                           0.72    428139
   macro avg       0.73      0.72      0.72    428139
weighted avg       0.73      0.72      0.72    428139
```




```
Feature Importances - Night:
                        Feature  Importance
2               parcels_per_sqkm    0.580722
1  neigh_median_household_income    0.383557
0          streetlights_per_sqkm    0.035722

Feature Importances - Day:
                        Feature  Importance
2               parcels_per_sqkm    0.581613
1  neigh_median_household_income    0.382048
0          streetlights_per_sqkm    0.036339
```

4 Appropriate evaluation of the models

4.1 Appropriate Evaluation of the Models

To evaluate both regression and classification models used for predicting property crime, we use metrics that are standard and interpretable.

4.1.1 Regression Models (Linear Regression)

Metrics Used: - **Mean Squared Error (MSE)**:
Measures the average squared difference between predicted and actual values.
→ Lower values = better model performance.

- **R² Score (Coefficient of Determination)**:
  Indicates how much variance in the outcome variable is explained by the model.
  → Values closer to 1 = better fit.

4.1.2 Classification Models (Random Forest Classifier)

Metrics Used: - **Accuracy**:
Percentage of correctly predicted labels.

- **Precision**:
  Out of all predicted positives, how many were actually positive.
- **Recall**:
  Out of all actual positives, how many were correctly predicted.
- **F1-Score**:
  Harmonic mean of precision and recall.
  → Useful when dealing with imbalanced classes.
- **Confusion Matrix**:
  Visualizes how many instances were correctly and incorrectly classified as high or low risk.

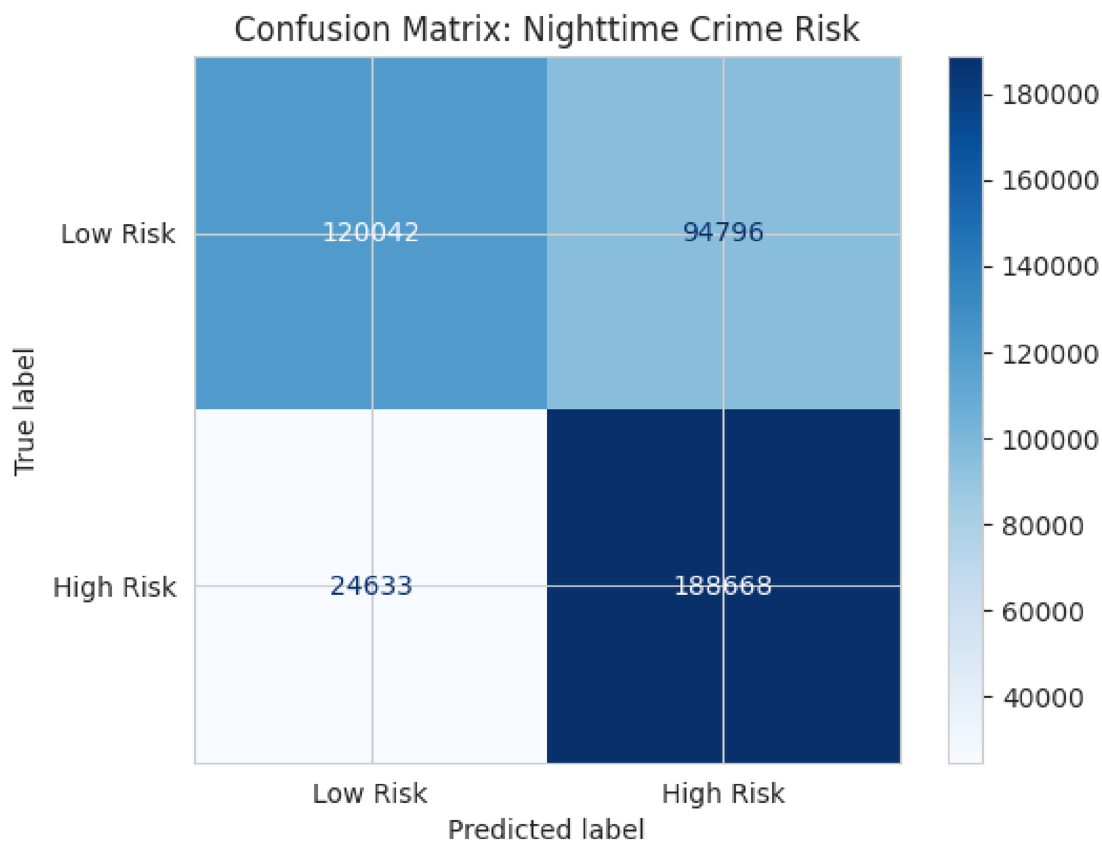
4.1.3 Outcome of Evaluation

After this step, we can: - Understand the strengths and weaknesses of each model. - Compare day vs. night model effectiveness. - Decide if improvements (like hyperparameter tuning or more features) are needed.

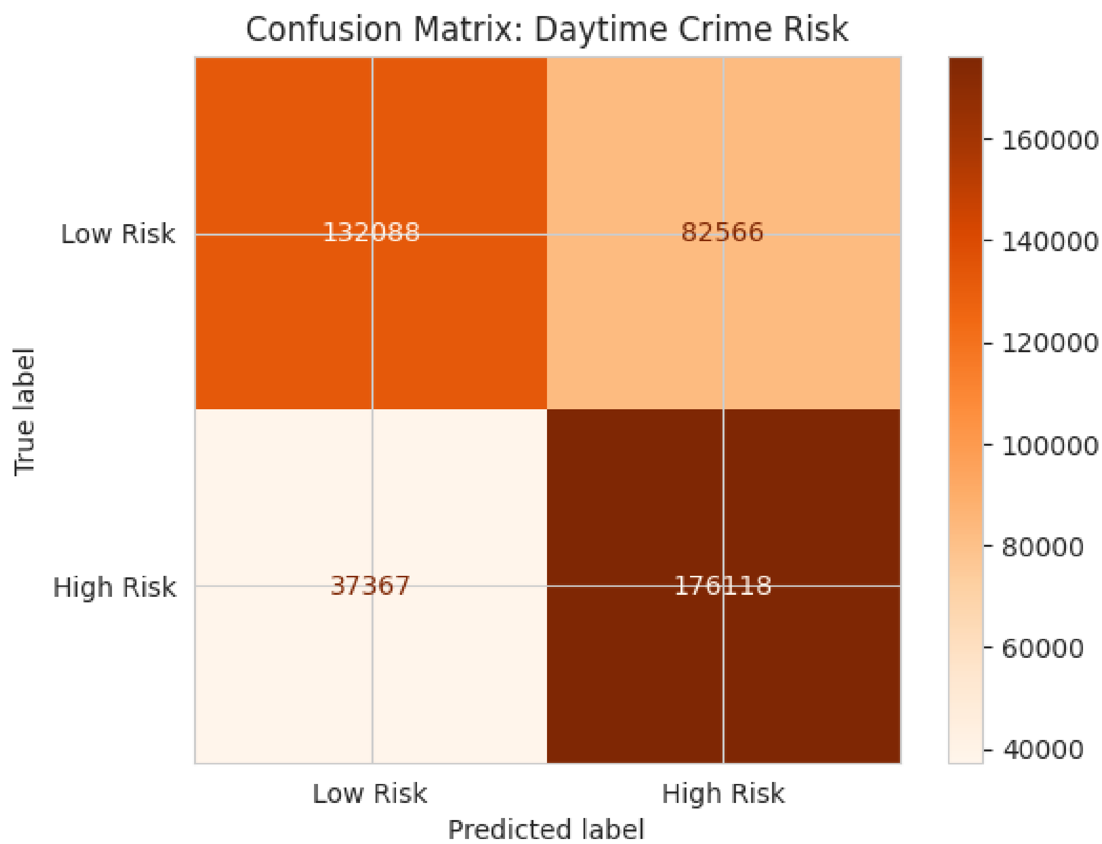
[17]:
```
from sklearn.metrics import r2_score, confusion_matrix, ConfusionMatrixDisplay

print("\n### NIGHTTIME MODEL EVALUATION ###\n")

# Linear Regression (Night)
r2_score_night = r2_score(y_test_log_night, y_pred_night)
print("Linear Regression (Night):")
print(f"  Mean Squared Error: {mse_night:.4f}")
print(f"  R² Score: {r2_score_night:.4f}")

# Random Forest Classifier (Night)
conf_matrix_night = confusion_matrix(y_test_cls_night, y_pred_cls_night)
ConfusionMatrixDisplay(confusion_matrix=conf_matrix_night, display_labels=['Low␣
↪Risk', 'High Risk']).plot(cmap='Blues')
```

```
plt.title("Confusion Matrix: Nighttime Crime Risk")
plt.show()

# ------------------------------
# DAYTIME MODEL EVALUATION
# ------------------------------
print("\n### DAYTIME MODEL EVALUATION ###\n")

# Linear Regression (Day)
r2_score_day = r2_score(y_test_log_day, y_pred_day)
print("Linear Regression (Day):")
print(f"  Mean Squared Error: {mse_day:.4f}")
print(f"  R² Score: {r2_score_day:.4f}")

# Random Forest Classifier (Day)
conf_matrix_day = confusion_matrix(y_test_cls_day, y_pred_cls_day)
ConfusionMatrixDisplay(confusion_matrix=conf_matrix_day, display_labels=['Low␣
↪Risk', 'High Risk']).plot(cmap='Oranges')
plt.title("Confusion Matrix: Daytime Crime Risk")
plt.show()

# ------------------------------
# CLASSIFICATION REPORTS
# ------------------------------
print("\n### CLASSIFICATION REPORTS ###\n")

print("Random Forest Classifier - Nighttime:")
print(classification_report(y_test_cls_night, y_pred_cls_night))

print("Random Forest Classifier - Daytime:")
print(classification_report(y_test_cls_day, y_pred_cls_day))
```

```
### NIGHTTIME MODEL EVALUATION ###

Linear Regression (Night):
  Mean Squared Error: 6.9393
  R² Score: 0.2543
```




```
### DAYTIME MODEL EVALUATION ###

Linear Regression (Day):
  Mean Squared Error: 7.4918
  R² Score: 0.3356
```




```
### CLASSIFICATION REPORTS ###

Random Forest Classifier - Nighttime:
              precision    recall  f1-score   support

           0       0.83      0.56      0.67    214838
           1       0.67      0.88      0.76    213301

    accuracy                           0.72    428139
   macro avg       0.75      0.72      0.71    428139
weighted avg       0.75      0.72      0.71    428139

Random Forest Classifier - Daytime:
              precision    recall  f1-score   support

           0       0.78      0.62      0.69    214654
           1       0.68      0.82      0.75    213485

    accuracy                           0.72    428139
```

```
   macro avg       0.73      0.72      0.72    428139
weighted avg       0.73      0.72      0.72    428139
```

4.2 Enhanced Evaluation of Crime Prediction Models

This section improves upon the basic model evaluation by incorporating advanced metrics and visual insights, allowing a deeper understanding of how the models perform during both nighttime and daytime.

4.2.1 Regression Evaluation (Night & Day)

Metrics Used: - **Mean Squared Error (MSE)**: Measures average prediction error. - **R^2 Score**: Indicates how well the model explains variance in crime rates.

4.2.2 Classification Evaluation (Random Forest Classifier)

Standard Metrics:

- **Confusion Matrix**: Shows true positives/negatives and false positives/negatives.
- **Classification Report**: Includes precision, recall, F1-score, and support for each class.

Advanced Metrics:

- **ROC-AUC Score**: Measures model's ability to distinguish between classes.
  - A higher ROC-AUC (closer to 1) means better separation between high-risk and low-risk zones.
- **Precision-Recall Curve**:
  - Useful when data is imbalanced.
  - Helps understand how precision and recall change at different thresholds.

4.2.3 Feature Importance

- Bar charts show how much each feature contributes to crime risk prediction.
- Separate plots for nighttime and daytime improve interpretability.

4.2.4 Key Takeaways

- Combines multiple views (metrics + visuals) to assess model reliability.
- Helps compare day vs night predictions and identify which features are most influential.

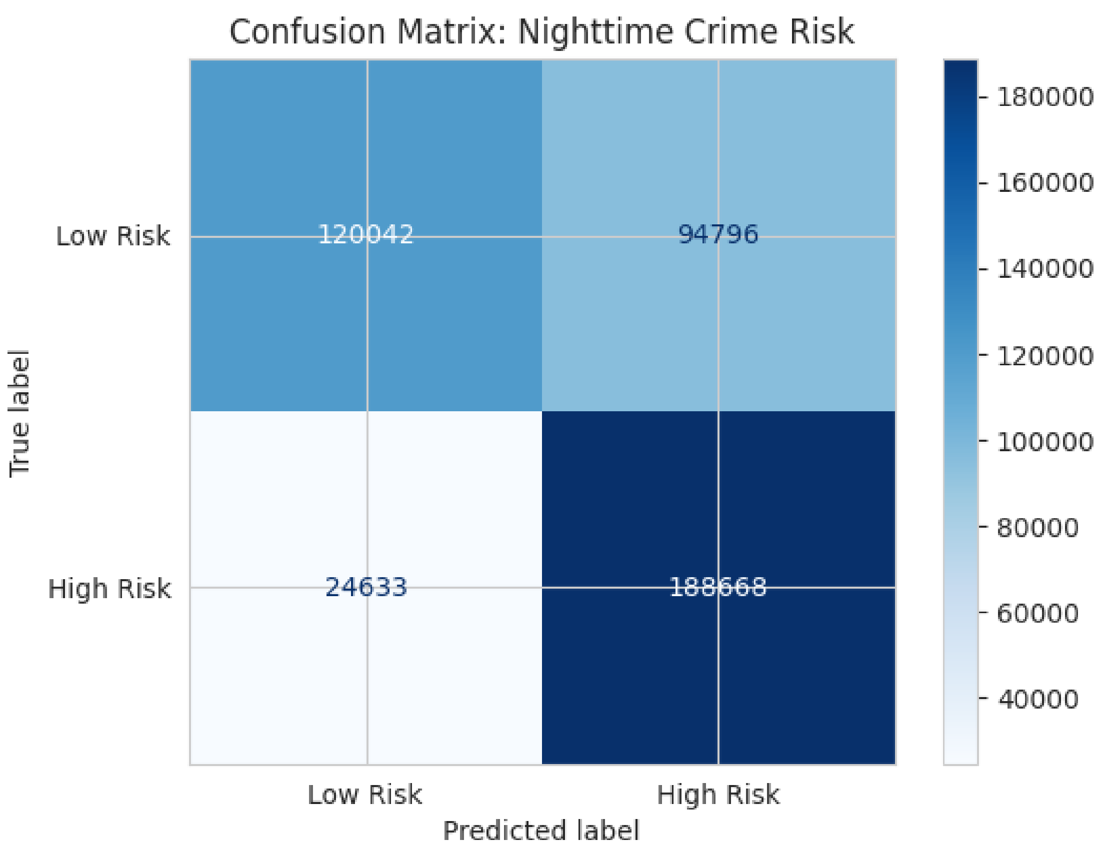
[18]:
```python
from sklearn.metrics import r2_score, confusion_matrix, ConfusionMatrixDisplay,␣
↪roc_auc_score, precision_recall_curve
```

```
print("\n### NIGHTTIME MODEL EVALUATION ###\n")

# Linear Regression (Night)
r2_night = r2_score(y_test_log_night, y_pred_night)
print("Linear Regression (Night):")
print(f"  Mean Squared Error: {mse_night:.4f}")
print(f"  R² Score: {r2_night:.4f}")

# Random Forest Classifier (Night) - Confusion Matrix
conf_matrix_night = confusion_matrix(y_test_cls_night, y_pred_cls_night)
ConfusionMatrixDisplay(confusion_matrix=conf_matrix_night, display_labels=['Low␣
↪Risk', 'High Risk']).plot(cmap='Blues')
plt.title("Confusion Matrix: Nighttime Crime Risk")
plt.show()

# ROC-AUC (Night)
roc_auc_night = roc_auc_score(y_test_cls_night, rf_night.
↪predict_proba(X_test_cls_night)[:, 1])
print(f"\nRandom Forest ROC-AUC (Night): {roc_auc_night:.4f}")

# Precision-Recall Curve (Night)
precision_night, recall_night, _ = precision_recall_curve(y_test_cls_night,␣
↪rf_night.predict_proba(X_test_cls_night)[:, 1])
plt.figure(figsize=(8, 6))
plt.plot(recall_night, precision_night, label='Precision-Recall Curve',␣
↪color='blue')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title("Precision-Recall Curve: Nighttime")
plt.legend()
plt.show()

# ------------------------------
# DAYTIME MODEL EVALUATION
# ------------------------------
print("\n### DAYTIME MODEL EVALUATION ###\n")

# Linear Regression (Day)
r2_day = r2_score(y_test_log_day, y_pred_day)
print("Linear Regression (Day):")
print(f"  Mean Squared Error: {mse_day:.4f}")
print(f"  R² Score: {r2_day:.4f}")

# Random Forest Classifier (Day) - Confusion Matrix
conf_matrix_day = confusion_matrix(y_test_cls_day, y_pred_cls_day)
ConfusionMatrixDisplay(confusion_matrix=conf_matrix_day, display_labels=['Low␣
↪Risk', 'High Risk']).plot(cmap='Oranges')
```

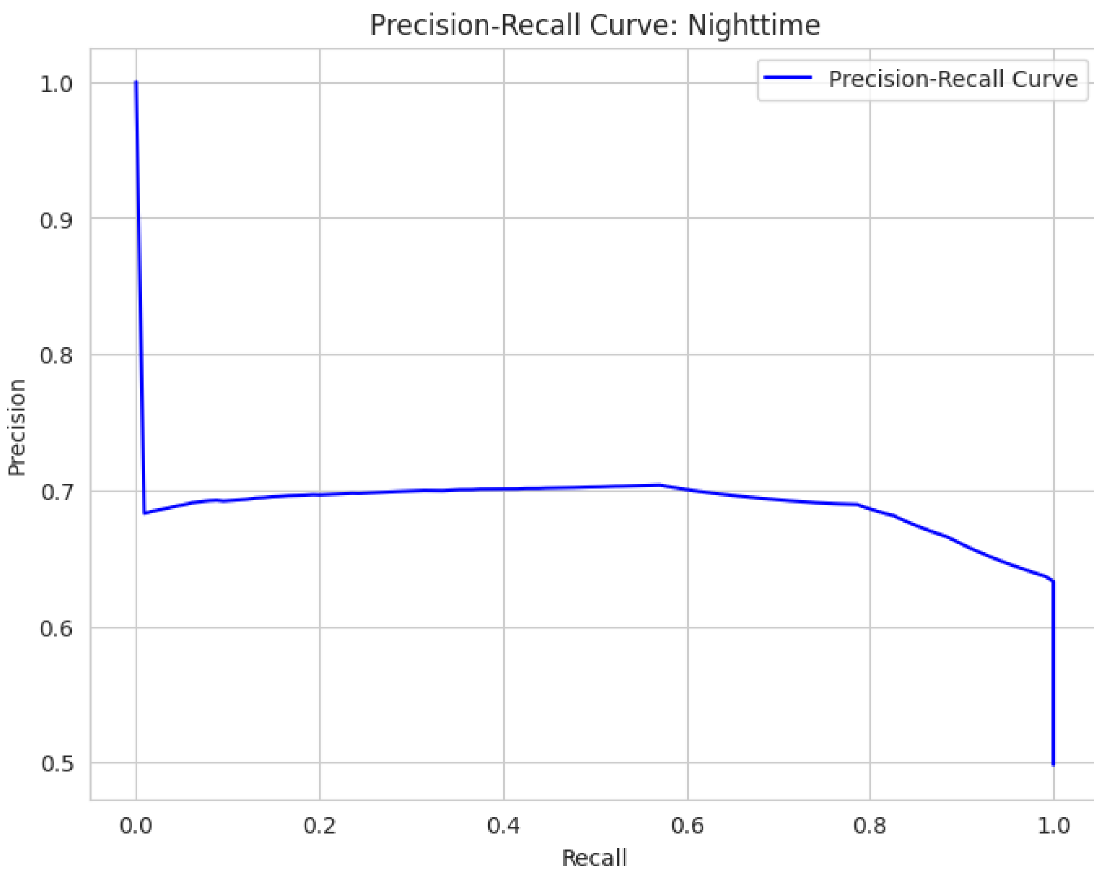
```
plt.title("Confusion Matrix: Daytime Crime Risk")
plt.show()

# ROC-AUC (Day)
roc_auc_day = roc_auc_score(y_test_cls_day, rf_day.
 ↪predict_proba(X_test_cls_day)[:, 1])
print(f"\nRandom Forest ROC-AUC (Day): {roc_auc_day:.4f}")

# Precision-Recall Curve (Day)
precision_day, recall_day, _ = precision_recall_curve(y_test_cls_day, rf_day.
 ↪predict_proba(X_test_cls_day)[:, 1])
plt.figure(figsize=(8, 6))
plt.plot(recall_day, precision_day, label='Precision-Recall Curve',␣
 ↪color='darkorange')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title("Precision-Recall Curve: Daytime")
plt.legend()
plt.show()

# ------------------------------
# CLASSIFICATION REPORTS
# ------------------------------
print("\n### CLASSIFICATION REPORTS ###\n")

print("Random Forest Classifier - Nighttime:")
print(classification_report(y_test_cls_night, y_pred_cls_night))

print("Random Forest Classifier - Daytime:")
print(classification_report(y_test_cls_day, y_pred_cls_day))

# ------------------------------
# FEATURE IMPORTANCE VISUALIZATION
# ------------------------------
plt.figure(figsize=(14, 10))

# Nighttime Feature Importance
plt.subplot(2, 1, 1)
plt.barh(importance_night['Feature'], importance_night['Importance'],␣
 ↪color='steelblue')
plt.xlabel("Importance")
plt.title("Feature Importance - Nighttime Crime Risk")
plt.gca().invert_yaxis()

# Daytime Feature Importance
plt.subplot(2, 1, 2)
```

```
plt.barh(importance_day['Feature'], importance_day['Importance'],␣
  ↪color='tomato')
plt.xlabel("Importance")
plt.title("Feature Importance - Daytime Crime Risk")
plt.gca().invert_yaxis()

plt.tight_layout()
plt.show()

# Print feature importance tables
print("\nFeature Importance - Nighttime:")
print(importance_night)

print("\nFeature Importance - Daytime:")
print(importance_day)
```

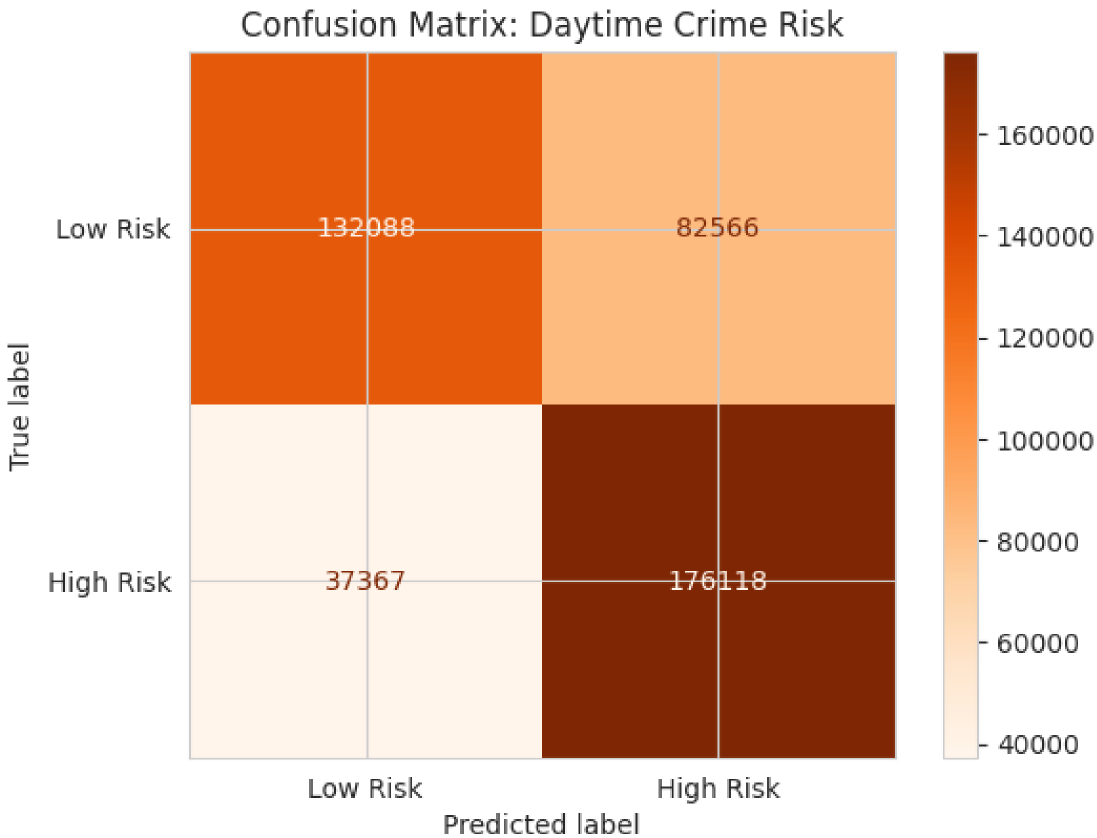
```
### NIGHTTIME MODEL EVALUATION ###

Linear Regression (Night):
  Mean Squared Error: 6.9393
  R² Score: 0.2543
```



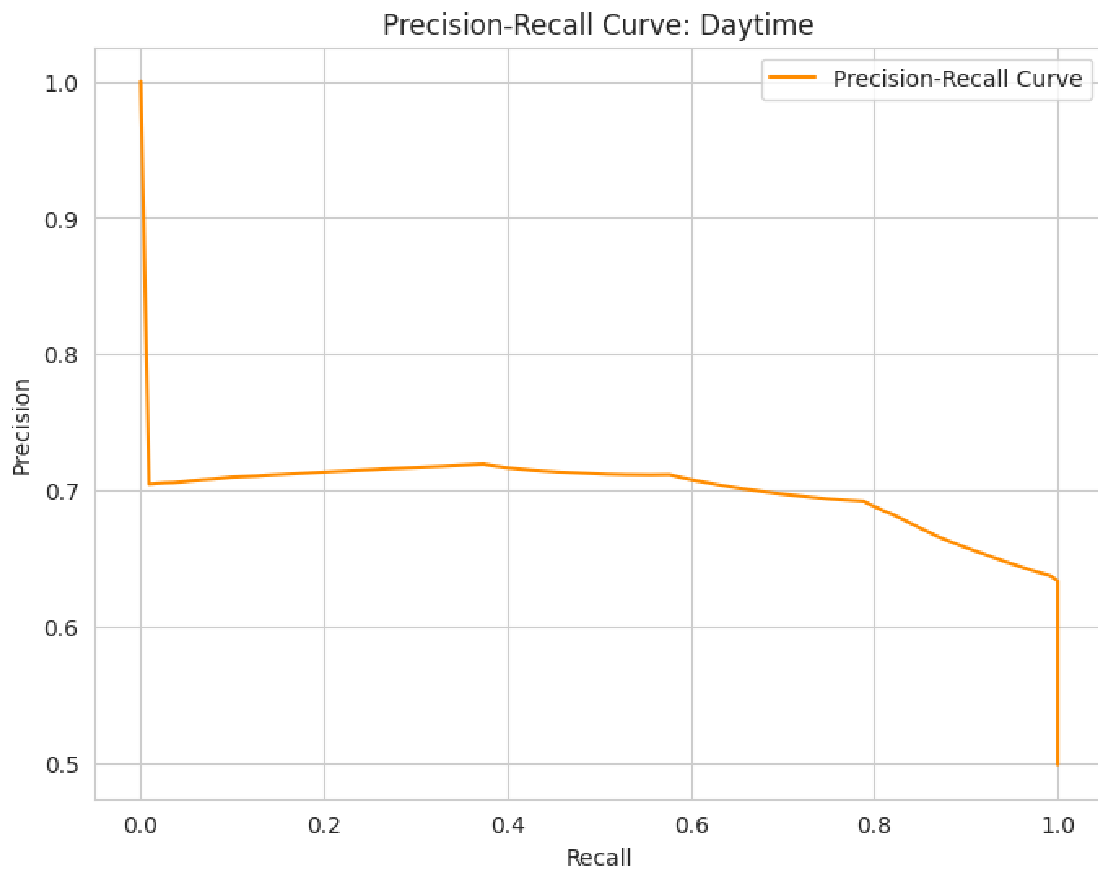
```
Random Forest ROC-AUC (Night): 0.7694
```




```
### DAYTIME MODEL EVALUATION ###

Linear Regression (Day):
  Mean Squared Error: 7.4918
  R² Score: 0.3356
```

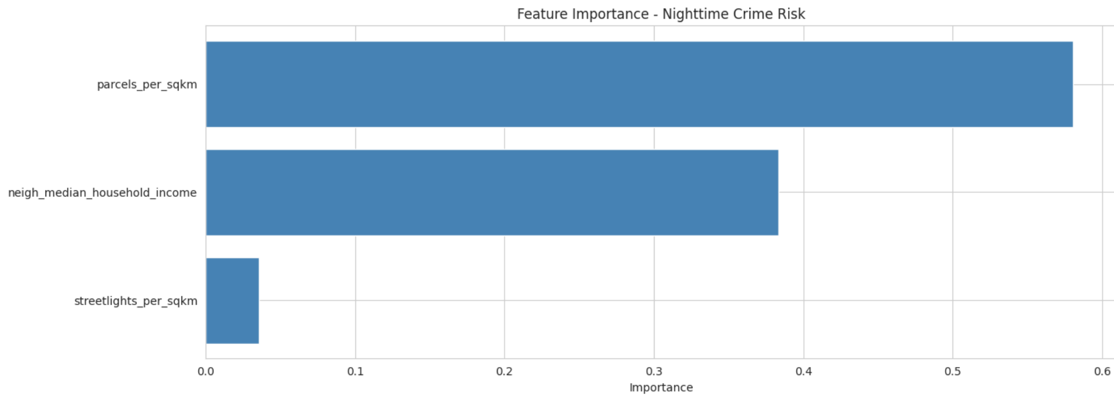


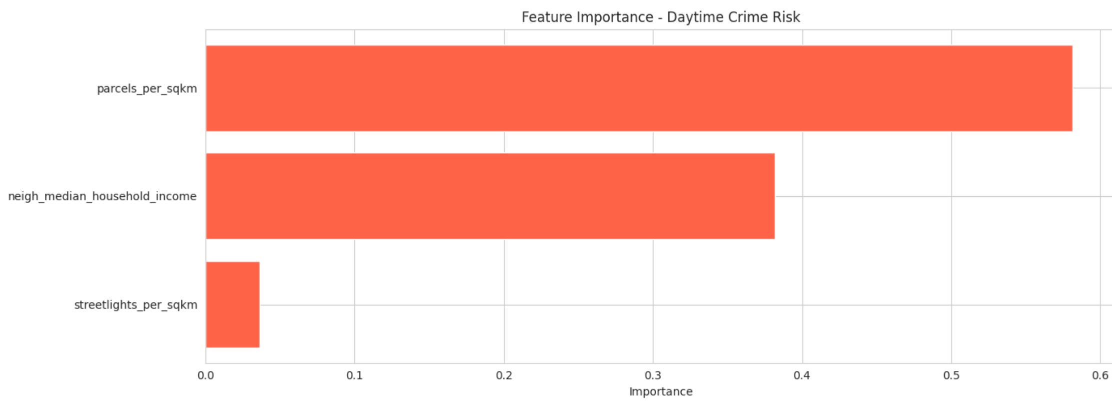

```
Random Forest ROC-AUC (Day): 0.7746
```




```
### CLASSIFICATION REPORTS ###

Random Forest Classifier - Nighttime:
              precision    recall  f1-score   support

           0       0.83      0.56      0.67    214838
           1       0.67      0.88      0.76    213301

    accuracy                           0.72    428139
   macro avg       0.75      0.72      0.71    428139
weighted avg       0.75      0.72      0.71    428139

Random Forest Classifier - Daytime:
              precision    recall  f1-score   support

           0       0.78      0.62      0.69    214654
           1       0.68      0.82      0.75    213485

    accuracy                           0.72    428139
```

```
   macro avg       0.73      0.72      0.72    428139
weighted avg       0.73      0.72      0.72    428139
```






```
Feature Importance - Nighttime:
                         Feature  Importance
2               parcels_per_sqkm    0.580722
1  neigh_median_household_income    0.383557
0          streetlights_per_sqkm    0.035722

Feature Importance - Daytime:
                         Feature  Importance
2               parcels_per_sqkm    0.581613
1  neigh_median_household_income    0.382048
0          streetlights_per_sqkm    0.036339
```

4.3 Analyzing the Correlation Between Median Household Income and Neighborhood Crimerate in Tucson

---

Problem Statement
Poverty, homelessness, and the divide between the top and bottom earners is a pressing issue in some parts of Tuscon. There are many country wide statistics examining the correlation between income and crime, but not specifically in Tucson.

Hypothesis
Neighborhoods with lower median household income will have a higher number of crimes than higher income areas.

Objective
This section aims to:
- Determine if a measurable correlation exists between median household income and crime rates in Tucson.
- Use machine learning models to predict crime risk in low income areas to help adress areas that need the most attention.

4.3.1 Data Gathering

Overview
Data for this project was sourced from public governmental Tucson datasets in CSV format and includes: - Arrest records (Tucson Police Department) in 2021 - Neighborhood income data

[19]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from IPython.display import display

#load data frames from csv
# neighborhoodPath = "Neighborhood_Income.csv"
df_neighborhood_income = tuc_neighbourhood_income
df_neighborhood_income =␣
↪df_neighborhood_income[df_neighborhood_income["HasData_1"] != 0]

# crimePath = "Tucson_Police_Arrests_-_2021_-_Open_Data.csv"
df_crime = tuc_arrest_data

print("crime head:")
display(df_crime.head())

print("neighborhood head:")
display(df_neighborhood_income.head())
```

```
crime head:

   OBJECTID             X              Y     arre_id     case_id agency  \
0         1  9.900089e+05  470751.276735  2021000107  2101020104    TPD
1         2  9.900089e+05  470751.276735  2021000107  2101020104    TPD
2         3  9.900089e+05  470751.276735  2021000107  2101020104    TPD
```

```
3         4  1.053154e+06  443419.380064  2021000110  2101020138   TPD
4         5  1.053154e+06  443419.380064  2021000110  2101020138   TPD

                 date_arr  time_arr            datetime_arr MONTH_ARR  …  \
0  2021/01/02 00:00:00+00      1731  2021/01/02 17:31:00+00    01-Jan  …
1  2021/01/02 00:00:00+00      1731  2021/01/02 17:31:00+00    01-Jan  …
2  2021/01/02 00:00:00+00      1731  2021/01/02 17:31:00+00    01-Jan  …
3  2021/01/02 00:00:00+00      1844  2021/01/02 18:44:00+00    01-Jan  …
4  2021/01/02 00:00:00+00      1844  2021/01/02 18:44:00+00    01-Jan  …

   LOC_STATUS WARD  NHA_NAME TMSECT                  DIVISION DIVISION_NO  \
0    GEOCODED  3.0       NaN    NaN  Operations Division West          T2
1    GEOCODED  3.0       NaN    NaN  Operations Division West          T2
2    GEOCODED  3.0       NaN    NaN  Operations Division West          T2
3    GEOCODED  2.0  Eastside    NaN  Operations Division East          T4
4    GEOCODED  2.0  Eastside    NaN  Operations Division East          T4

  DIVSECT        TRSQ City_geo            ADDRESS_100BLK
0    T203  13S13E24NW   TUCSON          4598 N ORACLE RD
1    T203  13S13E24NW   TUCSON          4598 N ORACLE RD
2    T203  13S13E24NW   TUCSON          4598 N ORACLE RD
3    T406  14S15E14NE   TUCSON  10198 E ESSEX VILLAGE DR
4    T406  14S15E14NE   TUCSON  10198 E ESSEX VILLAGE DR

[5 rows x 39 columns]

neighborhood head:

   OBJECTID             NAME  WARD     DATASOURCE  ID sourceCountry  \
0         1       A Mountain     1  NEIGHBORHOODS   0            US
1         2         Adelanto     3  NEIGHBORHOODS   1            US
2         3  Alvernon Heights     5  NEIGHBORHOODS   2            US
3         4            Amphi     3  NEIGHBORHOODS   3            US
4         5      Armory Park     6  NEIGHBORHOODS   4            US

   ENRICH_FID                 aggregationMethod  \
0           1  BlockApportionment:US.BlockGroups
1           2  BlockApportionment:US.BlockGroups
2           3  BlockApportionment:US.BlockGroups
3           4  BlockApportionment:US.BlockGroups
4           5  BlockApportionment:US.BlockGroups

   populationToPolygonSizeRating  apportionmentConfidence  …  AGGDIA75CY  \
0                          2.191                    2.576  …     1590160
1                          2.191                    2.576  …      154598
2                          2.191                    2.576  …      172634
3                          2.191                    2.576  …     2760918
4                          2.191                    2.576  …     3785750
```

```
   ID_1  sourceCountry_1  ENRICH_FID_1                aggregationMethod_1  \
0     0               US             1  BlockApportionment:US.BlockGroups
1     1               US             2  BlockApportionment:US.BlockGroups
2     2               US             3  BlockApportionment:US.BlockGroups
3     3               US             4  BlockApportionment:US.BlockGroups
4     4               US             5  BlockApportionment:US.BlockGroups

   populationToPolygonSizeRating_1  apportionmentConfidence_1  HasData_1  \
0                            2.191                      2.576          1
1                            2.191                      2.576          1
2                            2.191                      2.576          1
3                            2.191                      2.576          1
4                            2.191                      2.576          1

   TOTHH_CY  WLTHINDXCY
0      1103          32
1       117          28
2        99          26
3      3105          20
4      1223          48

[5 rows x 168 columns]
```

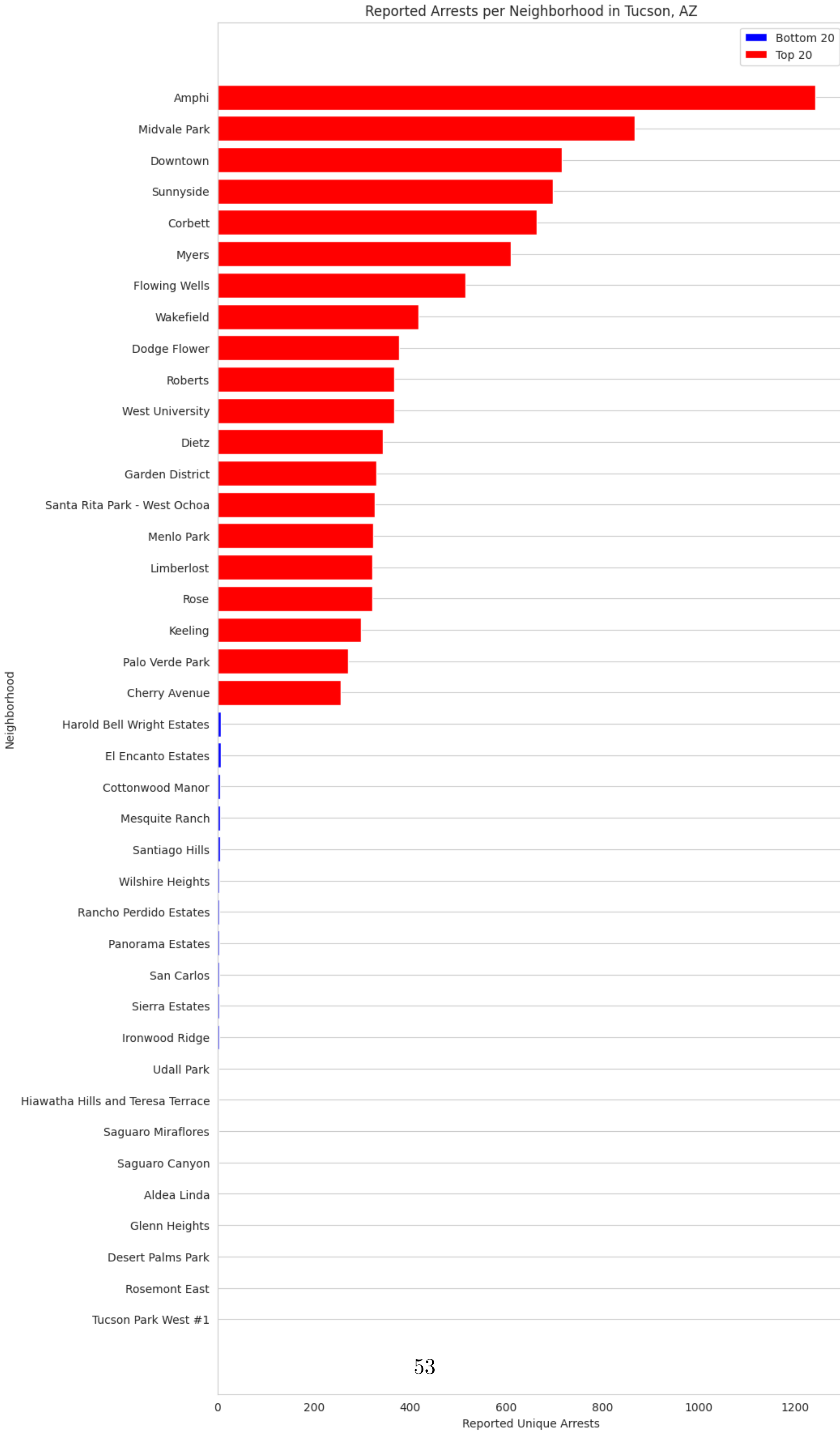
4.3.2 Visualisations

- **Plot the number of crimes per neighborhood:**
  First drop any duplicates in the data and plot unique arrests per neighborhood.

  Only show the first and last 20 neighborhoods.

[20]:
```
# Bar chart of crimes per neighborhood
df_crime.drop_duplicates("arre_id", inplace=True)
crimes_per_neighborhood = df_crime[["NHA_NAME"]].value_counts()


topcrimes_per_neighborhood = crimes_per_neighborhood.head(20)[::-1]
botcrimes_per_neighborhood = crimes_per_neighborhood.tail(20)[::-1]


neighborhoodCrimes = [n[0] for n in crimes_per_neighborhood.index]

topneighborhoodCrimes = neighborhoodCrimes[:20][::-1]
botneighborhoodCrimes = neighborhoodCrimes[-20:][::-1]

plt.figure(figsize=(10, 22))
plt.barh(botneighborhoodCrimes, botcrimes_per_neighborhood, color = "b")
plt.barh(topneighborhoodCrimes, topcrimes_per_neighborhood, color = "r")
```

```python
plt.legend(["Bottom 20", "Top 20"])
plt.ylabel("Neighborhood")
plt.xlabel("Reported Unique Arrests")
plt.title("Reported Arrests per Neighborhood in Tucson, AZ")
plt.grid(axis="x")
plt.show()
```



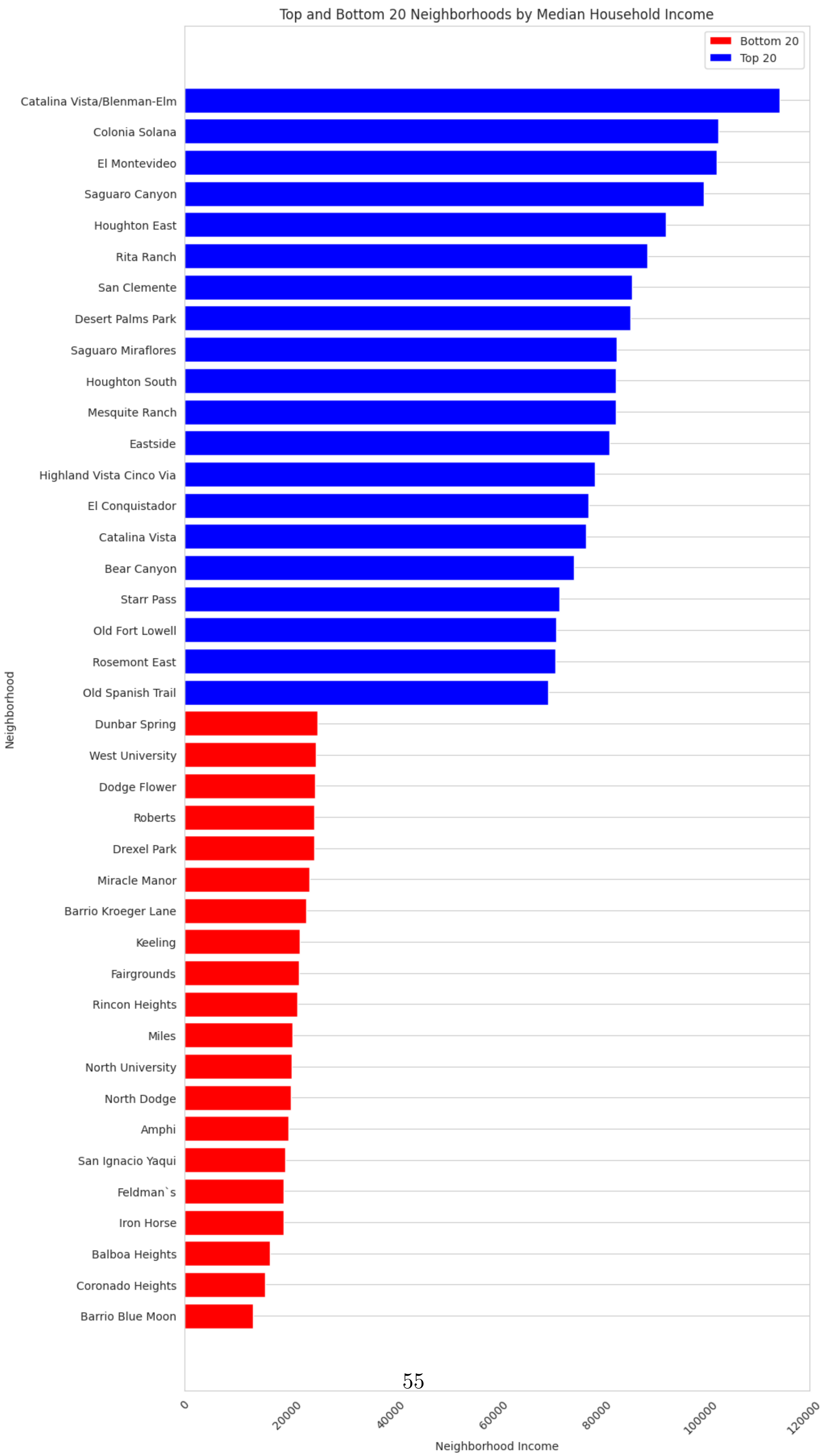
Reported Arrests per Neighborhood in Tucson, AZ
Bottom 20
Top 20
Amphi
Midvale Park
Downtown
Sunnyside
Corbett
Myers
Flowing Wells
Wakefield
Dodge Flower
Roberts
West University
Dietz
Garden District
Santa Rita Park - West Ochoa
Menlo Park
Limberlost
Rose
Keeling
Palo Verde Park
Cherry Avenue
Harold Bell Wright Estates
El Encanto Estates
Cottonwood Manor
Mesquite Ranch
Santiago Hills
Wilshire Heights
Rancho Perdido Estates
Panorama Estates
San Carlos
Sierra Estates
Ironwood Ridge
Udall Park
Hiawatha Hills and Teresa Terrace
Saguaro Miraflores
Saguaro Canyon
Aldea Linda
Glenn Heights
Desert Palms Park
Rosemont East
Tucson Park West #1
Neighborhood
0
200
400
600
800
1000
1200
Reported Unique Arrests

- **Plot the median income per neighborhood:**
  For comparison this also shows the top and bottom 20 neighborhoods.

  You may notice some overlap with the top and bottom and vice versa.

[21]:
```
#Get median household income from dataframe and set the index to the name from␣
↪2010 - 2019
df_neighborhood_income_name = df_neighborhood_income[["NAME", "MEDHINC_CY"]]
df_neighborhood_income_name.set_index("NAME", inplace=True)
sorted_df_top_20 = df_neighborhood_income_name.sort_values("MEDHINC_CY",␣
↪ascending=True).head(20)
sorted_df_bottom_20 = df_neighborhood_income_name.sort_values("MEDHINC_CY",␣
↪ascending=True).tail(20)

#source -> https://www.census.gov/quickfacts/fact/table/tucsoncityarizona/
↪INC110223 => 2019 - 2023
print(f"Median Household Income of all of Tuscon's neighborhood: $54,546")

plt.figure(figsize=(10, 22))
plt.xticks(rotation=45)
plt.barh(sorted_df_top_20.index, sorted_df_top_20["MEDHINC_CY"], color="r")
plt.barh(sorted_df_bottom_20.index, sorted_df_bottom_20["MEDHINC_CY"],␣
↪color="b")
plt.xlabel("Neighborhood Income")
plt.ylabel("Neighborhood")
plt.title("Top and Bottom 20 Neighborhoods by Median Household Income")
plt.grid(axis="x")
plt.legend(["Bottom 20", "Top 20"])
plt.show()
```

```
Median Household Income of all of Tuscon's neighborhood: $54,546
```


Top and Bottom 20 Neighborhoods by Median Household Income
Bottom 20
Top 20
Catalina Vista/Blenman-Elm
Colonia Solana
El Montevideo
Saguaro Canyon
Houghton East
Rita Ranch
San Clemente
Desert Palms Park
Saguaro Miraflores
Houghton South
Mesquite Ranch
Eastside
Highland Vista Cinco Via
El Conquistador
Catalina Vista
Bear Canyon
Starr Pass
Old Fort Lowell
Rosemont East
Old Spanish Trail
Dunbar Spring
West University
Dodge Flower
Roberts
Drexel Park
Miracle Manor
Barrio Kroeger Lane
Keeling
Fairgrounds
Rincon Heights
Miles
North University
North Dodge
Amphi
San Ignacio Yaqui
Feldman`s
Iron Horse
Balboa Heights
Coronado Heights
Barrio Blue Moon
Neighborhood
0
20000
40000
60000
80000
100000
120000
Neighborhood Income

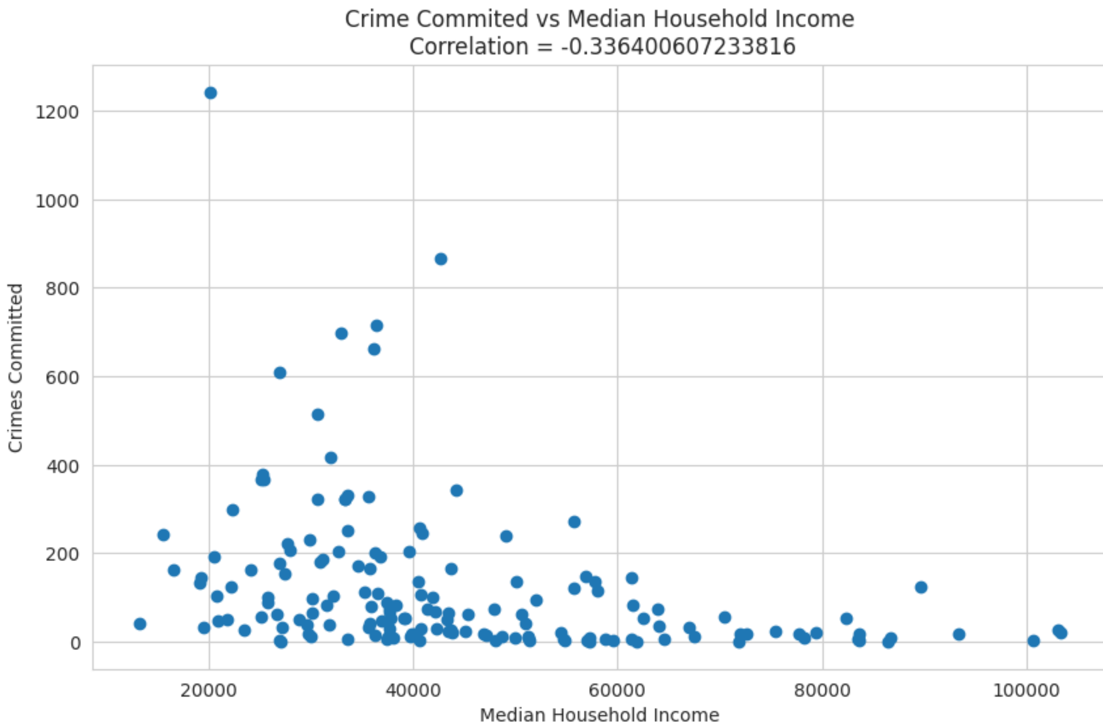
- **Data cleaning step:**
  Make dataframes based on neccessary information and beisdes dropping duplicates, remove empty or irrelivant data.

[22]:
```
#Data cleaning for correlation between neighborhood household income and␣
↪neighbood crime rate
crimes_per_neighborhood_df = df_crime[["NHA_NAME"]].value_counts().
↪to_frame(name="CRIMES")
crimes_per_neighborhood_df.index.rename("NAME", inplace=True)

crime_and_income_df = crimes_per_neighborhood_df.
↪merge(df_neighborhood_income_name, on="NAME", how="outer")
crime_and_income_df.sort_index(axis=0, ascending=True, inplace=True)
#droped 3 neighborhoods becauase there was NaN data for a column, ie either␣
↪crime/ income wasn't present
crime_and_income_df = crime_and_income_df.dropna()
```

4.3.3 Crime Analysis: Income vs Crime

This section provides visualisations to show the coorelation between the number of crimes and median household income of Tucson Neighborhoods:

[23]:
```
# Scatter plot for correlation between income and crime rate
correlation = crime_and_income_df["MEDHINC_CY"].
↪corr(crime_and_income_df["CRIMES"])


plt.figure(figsize=(10, 6))
plt.scatter(crime_and_income_df["MEDHINC_CY"], crime_and_income_df["CRIMES"] )
plt.xlabel("Median Household Income")
plt.ylabel("Crimes Committed")
plt.title(f"Crime Commited vs Median Household Income\n Correlation =␣
↪{correlation}")
plt.show()
```



4.3.4 Regression Model (Linear Regression)

We will try a linear regression model as it seems to follow a decreasing trend.

20% of the data will be used as testing data.

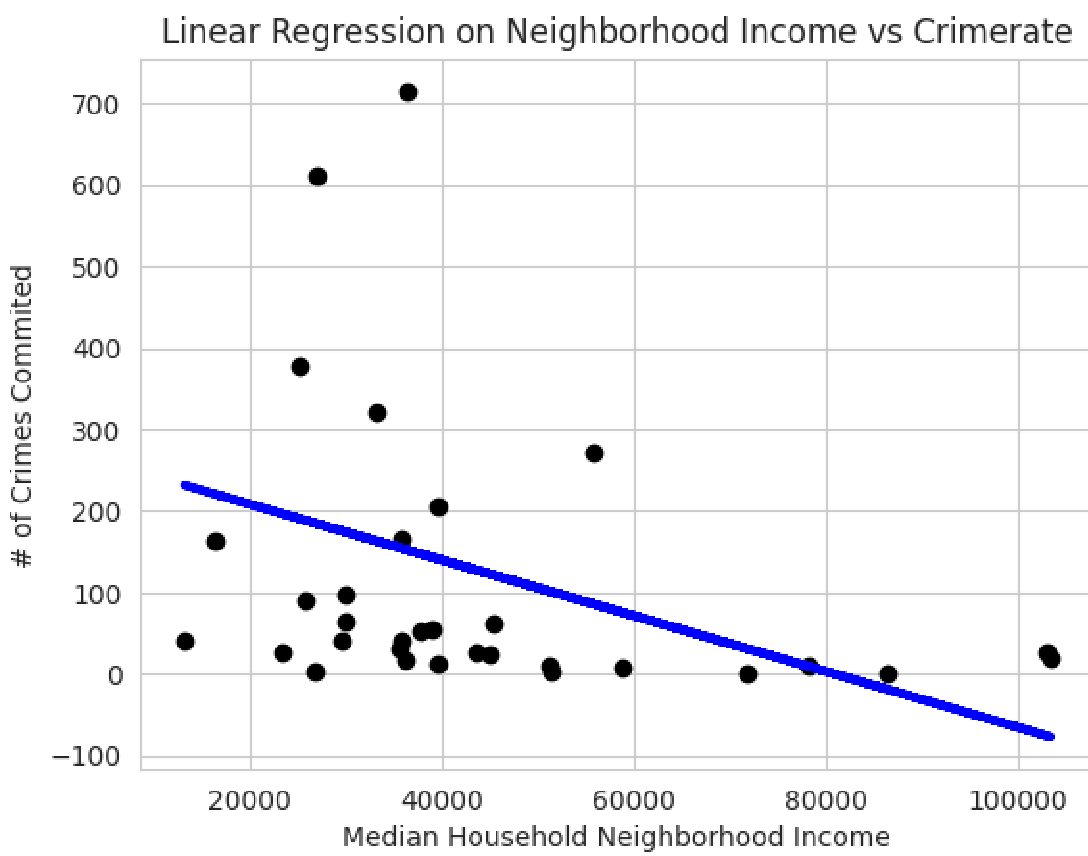
Metrics Used: - **Root Mean Squared Error (RMSE)**:
Measures the average difference between predicted and actual values.
→ Lower values = better model performance.

- **R^2 Score (Coefficient of Determination)**:
  Indicates how much variance in the outcome variable is explained by the model.
  → Values closer to 1 = better fit.

---

[24]:
```
# first model will be a linear regression model
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn import metrics
from sklearn.metrics import mean_squared_error, r2_score

x = crime_and_income_df[["MEDHINC_CY"]]
y = crime_and_income_df["CRIMES"]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

```
regr = linear_model.LinearRegression()

regr.fit(x_train, y_train)

y_pred = regr.predict(x_test)

linear_rmse = np.sqrt(mean_squared_error(y_test, y_pred))
linear_r2 = r2_score(y_test, y_pred)

# plot the data
plt.scatter(x_test, y_test, color='black')
plt.plot(x_test, y_pred, color = 'b', linewidth=3)

plt.title("Linear Regression on Neighborhood Income vs Crimerate")
plt.xlabel("Median Household Neighborhood Income")
plt.ylabel("# of Crimes Commited")

plt.xticks()
plt.yticks()
plt.show()
```



4.3.5 Regression Model (Logarithmic Regression)

It appears we can do better than a linear regression so lets try logarithmic regression to fit the logarithmic curve the data has.

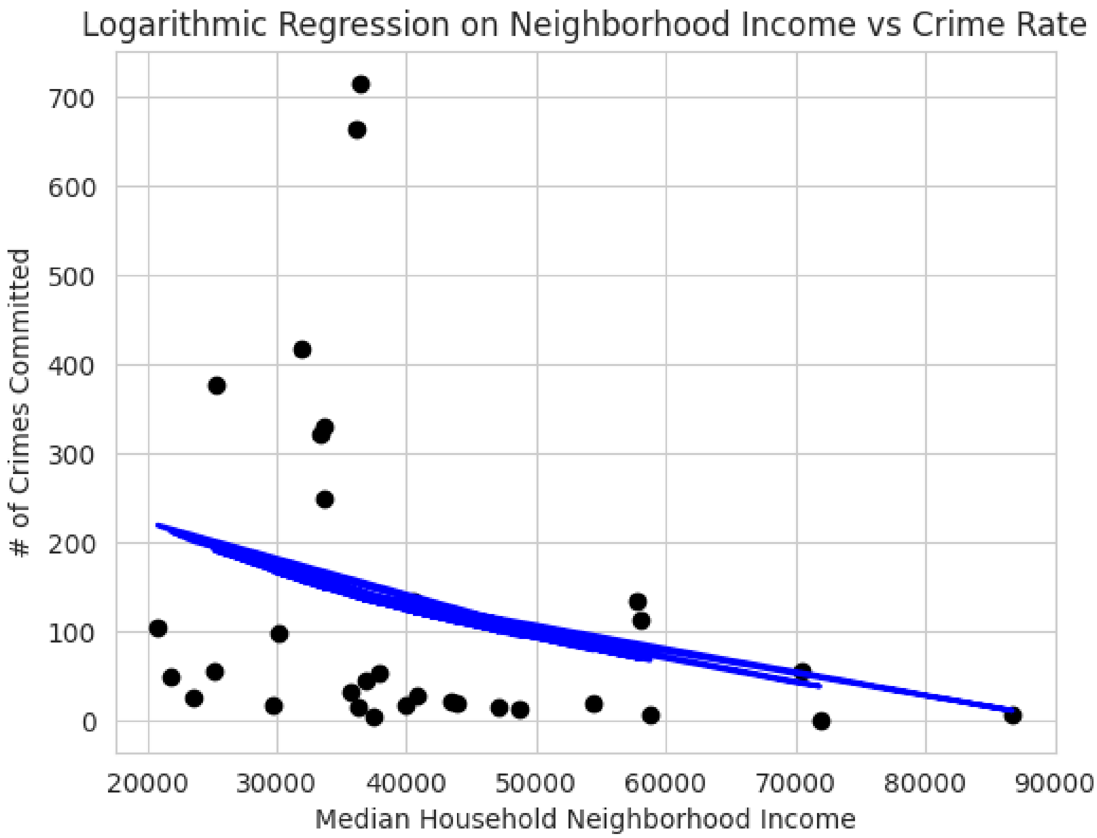
20% of the data will be used as testing data.

Same Metrics Used to Compare: - **Root Mean Squared Error (RMSE)**:
Measures the average difference between predicted and actual values.
→ Lower values = better model performance.

- **R^2 Score (Coefficient of Determination)**:
  Indicates how much variance in the outcome variable is explained by the model.
  → Values closer to 1 = better fit.

---

[25]:
```
# This seems to not be the best fit because of outliers in the lower income␣
 ↪range
# We can try a logarithmic regression model to better suit it.

x_train, x_test, y_train, y_test = train_test_split(np.log(x), y, test_size=0.2)

regr.fit(x_train, y_train)

y_pred = regr.predict(x_test)

log_rmse = np.sqrt(mean_squared_error(y_test, y_pred))
log_r2 = r2_score(y_test, y_pred)

plt.title("Logarithmic Regression on Neighborhood Income vs Crime Rate")
plt.scatter(np.exp(x_test), y_test, color='black')
plt.plot(np.exp(x_test), y_pred, color='b', linewidth=2)
plt.xlabel("Median Household Neighborhood Income")
plt.ylabel("# of Crimes Committed")
plt.xticks()
plt.yticks()
plt.show()
```



Compare metrics for the linear vs logarithmic regression

Metrics Used:

- Root Mean Squared Error (RMSE)
- R^2 Score

These results show that logarithmic regression is much better for this data.

- As stated earlier, a lower MSE (and RMSE which is just taking the square root of that to be more accurate to the data) signifies that it is a better model for the data.
- Also, the R^2 is higher showing that it fits the variance in data better. It is still not very high because of the outliers in the lower income neighborhoods.

[26]:
```
print("linear regression:")
print(f"RMSE = {linear_rmse}")
print(f"R^2 = {linear_r2}")
print("\nlogarithmic regression:")
print(f"RMSE = {log_rmse}")
print(f"R^2 = {log_r2}")
```

```
linear regression:
RMSE = 166.81496237791615
R^2 = 0.049218622096143916

logarithmic regression:
RMSE = 178.54820078463678
R^2 = 0.06085131171673752
```

[30]:

```
from google.colab import drive
drive.mount('/content/drive')
!ls "/content/drive/MyDrive/Colab_Notebook/"
!apt-get install pandoc

!apt-get install -y texlive-xetex texlive-fonts-recommended texlive-latex-extra
!pip install nbconvert
!jupyter nbconvert --to pdf "/content/drive/MyDrive/Colab Notebooks/Final␣
↪Project csc 380.ipynb" --output "/content/Final_Project_csc_380.pdf"


from google.colab import files
files.download("/content/Final_Project_csc_380.pdf")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
ls: cannot access '/content/drive/MyDrive/Colab_Notebook/': No such file or
directory
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
texlive-fonts-recommended is already the newest version (2021.20220204-1).
texlive-latex-extra is already the newest version (2021.20220204-1).
texlive-xetex is already the newest version (2021.20220204-1).
0 upgraded, 0 newly installed, 0 to remove and 34 not upgraded.
Requirement already satisfied: nbconvert in /usr/local/lib/python3.11/dist-
packages (7.16.6)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.11/dist-
packages (from nbconvert) (4.13.4)
Requirement already satisfied: bleach!=5.0.0 in /usr/local/lib/python3.11/dist-
packages (from bleach[css]!=5.0.0->nbconvert) (6.2.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.11/dist-
packages (from nbconvert) (0.7.1)
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.11/dist-
packages (from nbconvert) (3.1.6)
Requirement already satisfied: jupyter-core>=4.7 in
/usr/local/lib/python3.11/dist-packages (from nbconvert) (5.7.2)
Requirement already satisfied: jupyterlab-pygments in
/usr/local/lib/python3.11/dist-packages (from nbconvert) (0.3.0)
Requirement already satisfied: markupsafe>=2.0 in
```

```
/usr/local/lib/python3.11/dist-packages (from nbconvert) (3.0.2)
Requirement already satisfied: mistune<4,>=2.0.3 in
/usr/local/lib/python3.11/dist-packages (from nbconvert) (3.1.3)
Requirement already satisfied: nbclient>=0.5.0 in
/usr/local/lib/python3.11/dist-packages (from nbconvert) (0.10.2)
Requirement already satisfied: nbformat>=5.7 in /usr/local/lib/python3.11/dist-
packages (from nbconvert) (5.10.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-
packages (from nbconvert) (24.2)
Requirement already satisfied: pandocfilters>=1.4.1 in
/usr/local/lib/python3.11/dist-packages (from nbconvert) (1.5.1)
Requirement already satisfied: pygments>=2.4.1 in
/usr/local/lib/python3.11/dist-packages (from nbconvert) (2.19.1)
Requirement already satisfied: traitlets>=5.1 in /usr/local/lib/python3.11/dist-
packages (from nbconvert) (5.7.1)
Requirement already satisfied: webencodings in /usr/local/lib/python3.11/dist-
packages (from bleach!=5.0.0->bleach[css]!=5.0.0->nbconvert) (0.5.1)
Requirement already satisfied: tinycss2<1.5,>=1.1.0 in
/usr/local/lib/python3.11/dist-packages (from bleach[css]!=5.0.0->nbconvert)
(1.4.0)
Requirement already satisfied: platformdirs>=2.5 in
/usr/local/lib/python3.11/dist-packages (from jupyter-core>=4.7->nbconvert)
(4.3.7)
Requirement already satisfied: jupyter-client>=6.1.12 in
/usr/local/lib/python3.11/dist-packages (from nbclient>=0.5.0->nbconvert)
(6.1.12)
Requirement already satisfied: fastjsonschema>=2.15 in
/usr/local/lib/python3.11/dist-packages (from nbformat>=5.7->nbconvert) (2.21.1)
Requirement already satisfied: jsonschema>=2.6 in
/usr/local/lib/python3.11/dist-packages (from nbformat>=5.7->nbconvert) (4.23.0)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.11/dist-
packages (from beautifulsoup4->nbconvert) (2.7)
Requirement already satisfied: typing-extensions>=4.0.0 in
/usr/local/lib/python3.11/dist-packages (from beautifulsoup4->nbconvert)
(4.13.2)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.11/dist-
packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert) (25.3.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in
/usr/local/lib/python3.11/dist-packages (from
jsonschema>=2.6->nbformat>=5.7->nbconvert) (2025.4.1)
Requirement already satisfied: referencing>=0.28.4 in
/usr/local/lib/python3.11/dist-packages (from
jsonschema>=2.6->nbformat>=5.7->nbconvert) (0.36.2)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.11/dist-
packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert) (0.24.0)
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.11/dist-
packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (24.0.1)
Requirement already satisfied: python-dateutil>=2.1 in
```

```
/usr/local/lib/python3.11/dist-packages (from jupyter-
client>=6.1.12->nbclient>=0.5.0->nbconvert) (2.9.0.post0)
Requirement already satisfied: tornado>=4.1 in /usr/local/lib/python3.11/dist-
packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (6.4.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-
packages (from python-dateutil>=2.1->jupyter-
client>=6.1.12->nbclient>=0.5.0->nbconvert) (1.17.0)
[NbConvertApp] Converting notebook /content/drive/MyDrive/Colab Notebooks/Final
Project csc 380.ipynb to pdf
[NbConvertApp] ERROR | Error while converting '/content/drive/MyDrive/Colab
Notebooks/Final Project csc 380.ipynb'
Traceback (most recent call last):
  File "/usr/local/lib/python3.11/dist-packages/nbconvert/nbconvertapp.py", line
487, in export_single_notebook
    output, resources = self.exporter.from_filename(
                        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-
packages/nbconvert/exporters/templateexporter.py", line 390, in from_filename
    return super().from_filename(filename, resources, **kw)  #
type:ignore[return-value]
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-
packages/nbconvert/exporters/exporter.py", line 201, in from_filename
    return self.from_file(f, resources=resources, **kw)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-
packages/nbconvert/exporters/templateexporter.py", line 396, in from_file
    return super().from_file(file_stream, resources, **kw)  #
type:ignore[return-value]
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-
packages/nbconvert/exporters/exporter.py", line 220, in from_file
    return self.from_notebook_node(
           ^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-packages/nbconvert/exporters/pdf.py",
line 184, in from_notebook_node
    latex, resources = super().from_notebook_node(nb, resources=resources, **kw)
                       ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-packages/nbconvert/exporters/latex.py",
line 92, in from_notebook_node
    return super().from_notebook_node(nb, resources, **kw)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-
packages/nbconvert/exporters/templateexporter.py", line 429, in
from_notebook_node
    output = self.template.render(nb=nb_copy, resources=resources)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-packages/jinja2/environment.py", line
```

```
1295, in render
    self.environment.handle_exception()
  File "/usr/local/lib/python3.11/dist-packages/jinja2/environment.py", line
942, in handle_exception
    raise rewrite_traceback_stack(source=source)
  File "/usr/local/share/jupyter/nbconvert/templates/latex/index.tex.j2", line
8, in top-level template code
    ((* extends cell_style *))
    ^^^^^^^^^^^^^^^^^^^^^^^^^
  File
"/usr/local/share/jupyter/nbconvert/templates/latex/style_jupyter.tex.j2", line
176, in top-level template code
    \prompt{(((prompt)))}{(((prompt_color)))}{(((execution_count)))}{(((extra_sp
ace)))}
    ^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/share/jupyter/nbconvert/templates/latex/base.tex.j2", line 7,
in top-level template code
    ((*- extends 'document_contents.tex.j2' -*))
    ^^^^^^^^^^^^^^^^^^^^^^^^^
  File
"/usr/local/share/jupyter/nbconvert/templates/latex/document_contents.tex.j2",
line 51, in top-level template code
    ((*- block figure scoped -*))
    ^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/share/jupyter/nbconvert/templates/latex/display_priority.j2",
line 5, in top-level template code
    ((*- extends 'null.j2' -*))
    ^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/share/jupyter/nbconvert/templates/latex/null.j2", line 30, in
top-level template code
    ((*- block body -*))
  File "/usr/local/share/jupyter/nbconvert/templates/latex/base.tex.j2", line
241, in block 'body'
    ((( super() )))
  File "/usr/local/share/jupyter/nbconvert/templates/latex/null.j2", line 32, in
block 'body'
    ((*- block any_cell scoped -*))
    ^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/share/jupyter/nbconvert/templates/latex/null.j2", line 85, in
block 'any_cell'
    ((*- block markdowncell scoped-*)) ((*- endblock markdowncell -*))
    ^^^^^^^^^^^^^^^^^^^^^^^^^
  File
"/usr/local/share/jupyter/nbconvert/templates/latex/document_contents.tex.j2",
line 68, in block 'markdowncell'
    ((( cell.source | citation2latex | strip_files_prefix |
convert_pandoc('markdown+tex_math_double_backslash', 'json',extra_args=[]) |
resolve_references | convert_explicitly_relative_paths |
```

```
convert_pandoc('json','latex'))))
    ^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-packages/nbconvert/filters/pandoc.py",
line 36, in convert_pandoc
    return pandoc(source, from_format, to_format, extra_args=extra_args)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-packages/nbconvert/utils/pandoc.py", line
50, in pandoc
    check_pandoc_version()
  File "/usr/local/lib/python3.11/dist-packages/nbconvert/utils/pandoc.py", line
98, in check_pandoc_version
    v = get_pandoc_version()
        ^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-packages/nbconvert/utils/pandoc.py", line
75, in get_pandoc_version
    raise PandocMissing()
nbconvert.utils.pandoc.PandocMissing: Pandoc wasn't found.
Please check that pandoc is installed:
https://pandoc.org/installing.html
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-30-271b39cf7e26> in <cell line: 0>()
      9
     10 from google.colab import files
---> 11 files.download("/content/Final_Project_csc_380.pdf")
     12

/usr/local/lib/python3.11/dist-packages/google/colab/files.py in
 ↪download(filename)
    231   if not _os.path.exists(filename):
    232     msg = 'Cannot find file: {}'.format(filename)
--> 233     raise FileNotFoundError(msg)  # pylint: disable=undefined-variable
    234
    235   comm_manager = _IPython.get_ipython().kernel.comm_manager

FileNotFoundError: Cannot find file: /content/Final_Project_csc_380.pdf
```