IE6400 Foundations Data Analytics Engineering Fall Semester 2024

Project Group 2 : Rohan Verma and Bhavesh Kulkarni

Topic: Cleaning and Analyzing Crime Data

Objective:

This project aims to clean, inspect, and analyze a real-world crime dataset to identify trends, seasonal patterns, and other factors influencing crime rates. The dataset includes crime data from 2020 to the present, and the analysis follows a structured approach outlined in the project guidelines.

Table of Contents :

Task 1 : Data Acquisition

1. Data Acquisition:

```
In [7]: # Loading Crime Dataset
        crime_data = pd.read_csv('Crime_Data_from_2020_to_Present.csv')

In [9]: # Checking the shape of the dataframe
        crime_data.shape

Out[9]: (978628, 28)
```

Fig.1.1

Data acquisition is a critical step in any data analysis project. It involves gathering data from various sources, preparing it for analysis, and ensuring its integrity and relevance. In this project, we focus on acquiring a comprehensive crime dataset to analyze crime trends over time and assess various factors influencing crime rates.

Loading the Crime Dataset : For this analysis, we utilize a dataset titled "Crime Data from 2020 to Present," which is sourced from reliable public records. This dataset provides detailed information about reported crimes, including attributes such as crime type, date, time, location, and demographic data about victims. The dataset is loaded into a Pandas DataFrame shown in Fig.1.1.

Data Structure : After loading the dataset, it is crucial to understand its structure and dimensions to assess the volume of data we will be working with. This can be achieved by checking the shape of the DataFrame. The shape provides the number of rows and columns present in the dataset.

Output : The shape of the DataFrame is as follows: (978628, 28) :

- 978,628: This indicates that there are 978,628 individual records (rows) in the dataset. Each record represents a unique crime incident reported to the authorities.
- 28: This indicates that there are 28 columns, each corresponding to different attributes related to the crime incidents.

Task 2 : Data Inspection

Data Inspection : Data inspection is a crucial step in the data analysis process. It involves examining the dataset to understand its structure, identify any inconsistencies, and assess the quality of the data. In this project, we focus on inspecting a crime dataset to gain insights into its characteristics, including the number of records, the types of attributes present, and the overall data quality.

Displaying the First Few Rows: To get a glimpse of the dataset, we display the first five rows of the DataFrame. This allows us to understand the content and format of the data.

Input :

```
# Displaying first 5 rows of the dataframe
crime_data.head()
```

Output :

```
crime_data.head()
      DR_NO               Date Rptd                DATE OCC  TIME OCC
AREA  \
0  190326475  03/01/2020 12:00:00 AM  03/01/2020 12:00:00 AM      2130
7
1  200106753  02/09/2020 12:00:00 AM  02/08/2020 12:00:00 AM      1800
1
2  200320258  11/11/2020 12:00:00 AM  11/04/2020 12:00:00 AM      1700
3
3  200907217  05/10/2023 12:00:00 AM  03/10/2020 12:00:00 AM      2037
9
4  220614831  08/18/2022 12:00:00 AM  08/17/2020 12:00:00 AM      1200
6

   AREA NAME  Rpt Dist No  Part 1-2  Crm Cd  \
0   Wilshire          784         1     510
1    Central          182         1     330
2  Southwest          356         1     480
3   Van Nuys          964         1     343
4  Hollywood          666         2     354

                              Crm Cd Desc  ... Status   Status Desc
\
0                          VEHICLE - STOLEN  ...     AA  Adult Arrest

1                    BURGLARY FROM VEHICLE  ...     IC   Invest Cont

2                              BIKE - STOLEN  ...     IC   Invest Cont

3  SHOPLIFTING-GRAND THEFT ($950.01 & OVER)  ...     IC   Invest Cont

4                          THEFT OF IDENTITY  ...     IC   Invest Cont


  Crm Cd 1 Crm Cd 2  Crm Cd 3 Crm Cd 4  \
0    510.0    998.0       NaN      NaN
1    330.0    998.0       NaN      NaN
2    480.0      NaN       NaN      NaN
3    343.0      NaN       NaN      NaN
4    354.0      NaN       NaN      NaN

                                  LOCATION Cross Street      LAT
LON
0   1900 S  LONGWOOD                     AV          NaN  34.0375 -
118.3506
1   1000 S  FLOWER                      ST          NaN  34.0444 -
118.2628
2   1400 W  37TH                       ST          NaN  34.0210 -
118.3002
3  14000    RIVERSIDE                  DR          NaN  34.1576 -
118.4387
4                          1900    TRANSIENT          NaN  34.0944 -
118.3277

[5 rows x 28 columns]
```

Creating a Copy of the DataFrame: To prevent accidental modifications to the original dataset during the inspection and cleaning process, we create a copy of the DataFrame.

```
# Creating Copy of the Dataframe
data_crime = crime_data.copy()

# Creating Copy of the Dataframe
data_crime = crime_data.copy()

# Checking datatypes
crime_data.dtypes
```

Checking Data Types : Understanding the data types of each column is essential for subsequent data manipulation and analysis. We can use the {dtypes} attribute of the DataFrame to check the data types of all columns.

input:

```
# Checking datatypes
crime_data.dtypes
```

Output:

```
DR_NO               int64
Date Rptd          object
DATE OCC           object
TIME OCC            int64
AREA                int64
AREA NAME          object
Rpt Dist No         int64
Part 1-2           object
Crm Cd              int64
Crm Cd Desc        object
Mocodes            object
Vict Age            int64
Vict Sex           object
Vict Descent       object
Premis Cd         float64
Premis Desc        object
Weapon Used Cd    float64
Weapon Desc        object
Status             object
Status Desc        object
Crm Cd 1          float64
Crm Cd 2          float64
Crm Cd 3          float64
Crm Cd 4          float64
LOCATION           object
Cross Street       object
LAT               float64
LON               float64
dtype: object
```

Printing Columns : We can list all column names in the DataFrame for a clearer overview of the available attributes.

```
# Printing Columns
crime_data.columns

Index(['DR_NO', 'Date Rptd', 'DATE OCC', 'TIME OCC', 'AREA', 'AREA
NAME',
       'Rpt Dist No', 'Part 1-2', 'Crm Cd', 'Crm Cd Desc', 'Mocodes',
       'Vict Age', 'Vict Sex', 'Vict Descent', 'Premis Cd', 'Premis
Desc',
       'Weapon Used Cd', 'Weapon Desc', 'Status', 'Status Desc', 'Crm
Cd 1',
       'Crm Cd 2', 'Crm Cd 3', 'Crm Cd 4', 'LOCATION', 'Cross Street',
'LAT',
       'LON'],
      dtype='object')
```

Getting Information about the DataFrame : To gain a more comprehensive overview of the DataFrame,

including non-null counts, data types, and memory usage, we can use the info() method.

Input:

```
# Information about the Dataframe including the index dtype and
columns.
crime_data.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 978628 entries, 0 to 978627
Data columns (total 28 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   DR_NO           978628 non-null  int64
 1   Date Rptd       978628 non-null  object
 2   DATE OCC        978628 non-null  object
 3   TIME OCC        978628 non-null  int64
 4   AREA            978628 non-null  int64
 5   AREA NAME       978628 non-null  object
 7   Part 1-2        978628 non-null  int64
 8   Crm Cd          978628 non-null  int64
 9   Crm Cd Desc     978628 non-null  object
 10  Mocodes         834648 non-null  object
 11  Vict Age        978628 non-null  int64
 12  Vict Sex        841430 non-null  object
 13  Vict Descent    841419 non-null  object
 14  Premis Cd       978613 non-null  float64
 15  Premis Desc     978043 non-null  object
 16  Weapon Used Cd  325959 non-null  float64
 17  Weapon Desc     325959 non-null  object
 18  Status          978627 non-null  object
 19  Status Desc     978628 non-null  object
 20  Crm Cd 1        978617 non-null  float64
 21  Crm Cd 2        68816 non-null   float64
 22  Crm Cd 3        2309 non-null    float64
 23  Crm Cd 4        64 non-null      float64
 24  LOCATION        978628 non-null  object
 25  Cross Street    151427 non-null  object
 26  LAT             978628 non-null  float64
 27  LON             978628 non-null  float64
dtypes: float64(8), int64(7), object(13)
memory usage: 209.1+ MB
```

Task 3 : Data Cleaning

Identifying Missing Data:

- df.isnull() returns a DataFrame of the same shape as df, with True for cells that have missing values.
- df.isnull().sum() provides the sum of missing values for each column, giving you a count of how many entries are missing per feature.

Input **:**

```
# Task 3: Data Cleaning
# Loading the missing values

import pandas as pd

# Load the dataset
file_path = 'Crime_Data_from_2020_to_Present.csv'
df = pd.read_csv(file_path)

# Display the first few rows to understand the data structure
#print(df.head())

# Identifying missing values
missing_values = df.isnull().sum()
print(missing_values)
```

Output: The output represents the count of missing values in each column of your dataset. Let me break down what this means and what insights it provides:

- Each column name is listed along with the number of missing values in that column.
- For example: "Mocodes" has 145262 missing values. "Vict Sex" and "Vict Descent" have around 138,445 and 138,456 missing values, respectively.
- Columns like "Weapon Used Cd" and "Weapon Desc" have significant missing values, 656,471.

```
DR_NO                0
Date Rptd            0
DATE OCC             0
TIME OCC             0
AREA                 0
AREA NAME            0
Rpt Dist No          0
Part 1-2             0
Crm Cd               0
Crm Cd Desc          0
Mocodes         145262
Vict Age             0
Vict Sex        138445
Vict Descent    138456
Premis Cd           14
Premis Desc        585
Weapon Used Cd  656471
Weapon Desc     656471
Status               1
Status Desc          0
Crm Cd 1            11
Crm Cd 2        913763
Crm Cd 3        980327
Crm Cd 4        982574
LOCATION             0
Cross Street    830789
LAT                  0
LON                  0
dtype: int64
```

Handling missing data :

```
: # Task 3 : Handling data in rows named Premis Cd, Status and Crm Cd 1.

import pandas as pd

# Load the dataset
file_path = 'Crime_Data_from_2020_to_Present.csv'
df = pd.read_csv(file_path)

# Drop rows with missing values in columns that have only a few missing entries
df_cleaned = df.dropna(subset=['Premis Cd', 'Status', 'Crm Cd 1'])

# Verify if there are any missing values left in these columns
missing_values_after_dropping = df_cleaned[['Premis Cd', 'Status', 'Crm Cd 1']].isnull().sum()
print(missing_values_after_dropping)
```

```
Premis Cd    0
Status       0
Crm Cd 1     0
dtype: int64
```

Handling missing data is essential to ensure accurate results in data analysis. In this case, we dealt with missing values in key columns of the crime dataset, including **"Weapon Used Cd," "Weapon Desc," and "Cross Street."**

Data Imputation: Using Python's pandas library, we loaded the dataset and identified columns with missing data. To handle this:

- Weapon Used Cd: Missing values were replaced with 'Unknown' to indicate the weapon was not reported.
- Weapon Desc: Filled with 'Unknown' for unspecified weapon descriptions.
- Cross Street: Missing values were also replaced with 'Unknown' for unrecorded cross streets.

This method helps preserve the dataset's completeness for better analysis.

Dropping Columns with Excessive Missing Data: The column **"Crm Cd 4"** had too many missing values and was removed using the df.drop() method.

After the changes, we confirmed that no missing values remained in the imputed columns and that **"Crm Cd 4"** was successfully removed.

Remove the Check for and remove duplicate rows: There are no duplicate rows.

```
In [8]: # Task 3 : Check for and remove duplicate rows.

        # Check for duplicate rows
        duplicate_rows = df[df.duplicated()]
        print(f"Number of duplicate rows: {len(duplicate_rows)}")

        # Remove duplicate rows
        df_no_duplicates = df.drop_duplicates()

        # Verify if duplicates are removed
        print(f"Number of rows after removing duplicates: {len(df_no_duplicates)}")
```

```
Number of duplicate rows: 0
Number of rows after removing duplicates: 982638
```

Converting data types :

In this task, the goal was to ensure that the data types of various columns in the crime dataset were appropriate for analysis.

Date Conversion: - The 'Date Rptd' and 'DATE OCC' columns were originally in string format. To ensure proper handling of date values, both columns were converted to datetime format using pd.to_datetime(). This allows for easier manipulation and analysis of dates.

Numerical Conversion: - The 'Vict Age' column, which contains ages, was converted to a numeric data type using pd.to_numeric() to handle any non-numeric values.

Categorical Conversion: - Columns such as 'AREA' and 'Rpt Dist No' represent categorical data but were stored as numbers. These were converted to string (`object` type) to better reflect their nature as categories.

Final Data Types: The conversions ensure that columns are in the correct format, optimizing the dataset for accurate analysis. For example, dates are now in 'datetime64[ns]' format, ages are integers ('int64'), and categorical data is stored as strings ('object').

INPUT and OUTPUT:

```python
# Task 3 : Converting data types if needed.
import pandas as pd

# Load the dataset
file_path = 'Crime_Data_from_2020_to_Present.csv'
df = pd.read_csv(file_path)

# Print a few rows of the date columns to inspect the format
print("Inspecting date columns:")
print(df[['Date Rptd', 'DATE OCC']].head())

# Convert date columns to datetime format with explicit format
# Adjust the format string based on your inspection results
df['Date Rptd'] = pd.to_datetime(df['Date Rptd'], format='%m/%d/%Y', errors='coerce')
df['DATE OCC'] = pd.to_datetime(df['DATE OCC'], format='%m/%d/%Y', errors='coerce')

# Convert numerical columns to appropriate numeric types (if needed)
df['Vict Age'] = pd.to_numeric(df['Vict Age'], errors='coerce')

# Convert categorical codes to string (if applicable)
df['AREA'] = df['AREA'].astype(str)
df['Rpt Dist No'] = df['Rpt Dist No'].astype(str)

# Verify the data types after conversion
print("\nData types after conversion:")
print(df.dtypes)
```

```
Inspecting date columns:
              Date Rptd                 DATE OCC
0  03/01/2020 12:00:00 AM  03/01/2020 12:00:00 AM
1  02/09/2020 12:00:00 AM  02/08/2020 12:00:00 AM
2  11/11/2020 12:00:00 AM  11/04/2020 12:00:00 AM
3  05/10/2023 12:00:00 AM  03/10/2020 12:00:00 AM
4  08/18/2022 12:00:00 AM  08/17/2020 12:00:00 AM

Data types after conversion:
DR_NO                    int64
Date Rptd       datetime64[ns]
DATE OCC        datetime64[ns]
TIME OCC                 int64
AREA                    object
AREA NAME               object
Rpt Dist No             object
Part 1-2                 int64
Crm Cd                   int64
Crm Cd Desc             object
Mocodes                 object
Vict Age                 int64
Vict Sex                object
Vict Descent            object
Premis Cd              float64
Premis Desc             object
Weapon Used Cd         float64
Weapon Desc             object
Status                  object
Status Desc             object
Crm Cd 1               float64
Crm Cd 2               float64
Crm Cd 3               float64
Crm Cd 4               float64
LOCATION                object
Cross Street            object
LAT                    float64
LON                    float64
dtype: object
```

Outliers :

In this task, we focused on identifying and handling outliers in the Vict Age column of the crime dataset.

Outlier Detection:

1. Interquartile Range (IQR):
    - We calculated the first quartile (Q1) and third quartile (Q3), which represent the 25th and 75th percentiles of the Vict Age values.
    - The Interquartile Range (IQR) is the difference between Q3 and Q1, and it helps measure the spread of the data.

2. Outlier Boundaries:
    - Using the IQR, we defined the lower bound as Q1 - 1.5 * IQR and the upper bound as Q3 + 1.5 * IQR. Values falling outside this range are considered outliers.

3. Outlier Identification: Rows where Vict Age is below the lower bound or above the upper bound were identified as outliers.

Outlier Handling:

1. Option 1: Remove Outliers: We created a new DataFrame (df_no_outliers) where outliers in Vict Age were removed, ensuring the data remains clean for analysis.

2. Option 2: Impute Outliers: Instead of removing outliers, another approach is to replace them with the median age using the apply() function. This allows us to retain all data while mitigating the influence of extreme values.

Finally, after applying these steps, we verified that there were no remaining outliers in the Vict Age column.

INPUT and OUTPUT :

```python
# Task 3: Outliers
import pandas as pd

# Load the dataset
file_path = 'Crime_Data_from_2020_to_Present.csv'
df = pd.read_csv(file_path)

# Example: Dealing with outliers in the 'Vict Age' column
# Calculate Q1 (25th percentile) and Q3 (75th percentile)
Q1 = df['Vict Age'].quantile(0.25)
Q3 = df['Vict Age'].quantile(0.75)
IQR = Q3 - Q1

# Define the bounds for acceptable values
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify outliers
outliers = df[(df['Vict Age'] < lower_bound) | (df['Vict Age'] > upper_bound)]
print(f"Number of outliers detected in 'Vict Age': {len(outliers)}")

# Option 1: Remove outliers
df_no_outliers = df[(df['Vict Age'] >= lower_bound) & (df['Vict Age'] <= upper_bound)]

# Option 2: Impute outliers with median
median_age = df['Vict Age'].median()
df['Vict Age'] = df['Vict Age'].apply(lambda x: median_age if x < lower_bound or x > upper_bound else

# Verify the number of outliers remaining
remaining_outliers = df[(df['Vict Age'] < lower_bound) | (df['Vict Age'] > upper_bound)]
print(f"Number of remaining outliers in 'Vict Age': {len(remaining_outliers)}")
```

```
Number of outliers detected in 'Vict Age': 1
Number of remaining outliers in 'Vict Age': 0
```

Standardization and Normalization:

In this task, we standardized and normalized the Vict Age column to prepare the data for analysis.

Standardization: Standardization transforms the data to have a mean of 0 and a standard deviation of 1. We used the StandardScaler() to standardize Vict Age. The resulting values show how far each age is from the mean in terms of standard deviations. Normalization:

Normalization scales the data to a range of 0 to 1, using MinMaxScaler(). This ensures that all values fall within the same range, making comparisons between different features easier. Both techniques help to improve model performance by putting the data on a common scale.

INPUT and OUTPUT :

In [18]:
```
#Task 3 : Standardize or Normalize Numerical Data

import pandas as pd

from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Load the dataset
file_path = 'Crime_Data_from_2020_to_Present.csv'
df = pd.read_csv(file_path)

# Select the column(s) to standardize or normalize
numerical_columns = ['Vict Age']  # Add other numerical columns if needed

# Standardization
scaler = StandardScaler()
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])

# Check the standardized values
print("Standardized 'Vict Age':")
print(df[numerical_columns].head())

# Alternatively, for normalization:
# Initialize the MinMaxScaler
normalizer = MinMaxScaler()
df[numerical_columns] = normalizer.fit_transform(df[numerical_columns])

# Check the normalized values
print("Normalized 'Vict Age':")
print(df[numerical_columns].head())
```

```
Standardized 'Vict Age':
   Vict Age
0 -1.323610
1  0.815663
2 -0.458797
3 -0.458797
4 -0.049149
Normalized 'Vict Age':
   Vict Age
0  0.032258
1  0.411290
2  0.185484
3  0.185484
4  0.258065
```

Encoding:

In this task we have to analyze crime data by encoding categorical variables to prepare the dataset for machine learning models. Given the high cardinality in some columns, special techniques are used to manage and reduce complexity without losing valuable information.

Methodology:

Label Encoding: Converts categorical variables into numerical form, allowing them to be processed by machine learning algorithms. This is particularly useful for categorical data with fewer unique values.

Handling High Cardinality: Columns with a large number of unique values can complicate model training. To mitigate this, categories with low frequency (below a defined threshold) are grouped into an 'Other' category, reducing noise and improving the model's focus on significant patterns.

Data Cleaning: Preprocessing steps like grouping low-frequency categories and encoding categorical variables are crucial for ensuring that the dataset is ready for predictive modeling while maintaining computational efficiency.

Outcome: These techniques optimize the dataset, making it more manageable and suitable for machine learning models, allowing for meaningful insights into crime patterns and trends.

```
l]: # ENCODING:
    import pandas as pd
    from sklearn.preprocessing import LabelEncoder

    # Load the dataset
    file_path = 'Crime_Data_from_2020_to_Present.csv'
    df = pd.read_csv(file_path)

    # Inspect categorical columns
    categorical_columns = df.select_dtypes(include=['object']).columns.tolist()
    print("Categorical columns:", categorical_columns)

    # Initializing the Label Encoder
    label_encoder = LabelEncoder()

    #Label encoding for all categorical columns
    for column in categorical_columns:
        # Check unique values count
        unique_count = df[column].nunique()
        print(f"Unique values in '{column}': {unique_count}")



        # Encode the categorical variable
        df[column] = label_encoder.fit_transform(df[column])

    # printing the result
    print("\nDataFrame after Label Encoding:")
    print(df[categorical_columns].head())
```

```
Categorical columns: ['Date Rptd', 'DATE OCC', 'AREA NAME', 'Crm Cd Desc', 'Mocodes', 'Vict Sex', 'Vict Descent', 'Premis Des
c', 'Weapon Desc', 'Status', 'Status Desc', 'LOCATION', 'Cross Street']
Unique values in 'Date Rptd': 1735
Unique values in 'DATE OCC': 1735
Unique values in 'AREA NAME': 21
Unique values in 'Crm Cd Desc': 140
Unique values in 'Mocodes': 309364
Unique values in 'Vict Sex': 5
Unique values in 'Vict Descent': 20
Unique values in 'Premis Desc': 306
Unique values in 'Weapon Desc': 79
Unique values in 'Status': 6
Unique values in 'Status Desc': 6
Unique values in 'LOCATION': 66265
Unique values in 'Cross Street': 10326

DataFrame after Label Encoding:
   Date Rptd  DATE OCC  AREA NAME  Crm Cd Desc  Mocodes  Vict Sex  \
0        297       297         20          134   309364         3
1        195       190          1           21   268374         3
2       1531      1503         15           11    48902         4
3        650       342         17          106    15487         3
4       1149      1142          6          116   269901         3

   Vict Descent  Premis Desc  Weapon Desc  Status  Status Desc  LOCATION  \
0            12          266           79       0            0     21599
1            12           29           79       3            2      1856
2            18          207           79       3            2     13315
3            12           40           79       3            2     13512
4             7          254           79       3            2     21439

   Cross Street
0         10326
1         10326
2         10326
3         10326
4         10326
```

Task 3 : Exploratory Data Analysis (EDA)

EDA: For Exploratory data analysis we have to clean the dataset first.

INPUT and OUTPUT :

In [3]:
```
import pandas as pd

# Step 1: Load the dataset

crime_data = pd.read_csv('Crime_Data_from_2020_to_Present.csv')

# Step 2: Specify the date columns to clean and their formats
date_columns = {
    'DATE OCC': '%m/%d/%Y %I:%M:%S %p',  # Format for DATE OCC
    'Date Rptd': '%m/%d/%Y',  # Adjusting this format
}

# Step 3: Convert the date columns to datetime format with specified format
for col, date_format in date_columns.items():
    crime_data[col] = pd.to_datetime(crime_data[col], format=date_format, errors='coerce')

# Step 4: Drop rows with NaT in the specified date columns
crime_data.dropna(subset=date_columns.keys(), inplace=True)


print("Dataset cleaned successfully. NaT values removed.")
```

```
Dataset cleaned successfully. NaT values removed.
```

Visualizing the crime trends from 2020 to present:

This Python code visualizes crime trends from 2020 to the present by creating a bar chart using data from a CSV file (`Crime_Data_from_2020_to_Present.csv`).

The steps are as follows:

1. **Data Loading and Cleaning:** The crime dataset is loaded into a DataFrame, and the 'DATE OCC' column, which contains the date of occurrence, is converted to a datetime format for easier extraction of year information.

2. **Extracting Year and Grouping Data:** The code extracts the year from each occurrence date and then groups the data by year, counting the number of crimes per year.

3. **Bar Chart Creation:** A bar chart is plotted, showing the number of crimes for each year. The bars are black, and each bar has a label at the top showing the exact count of crimes.

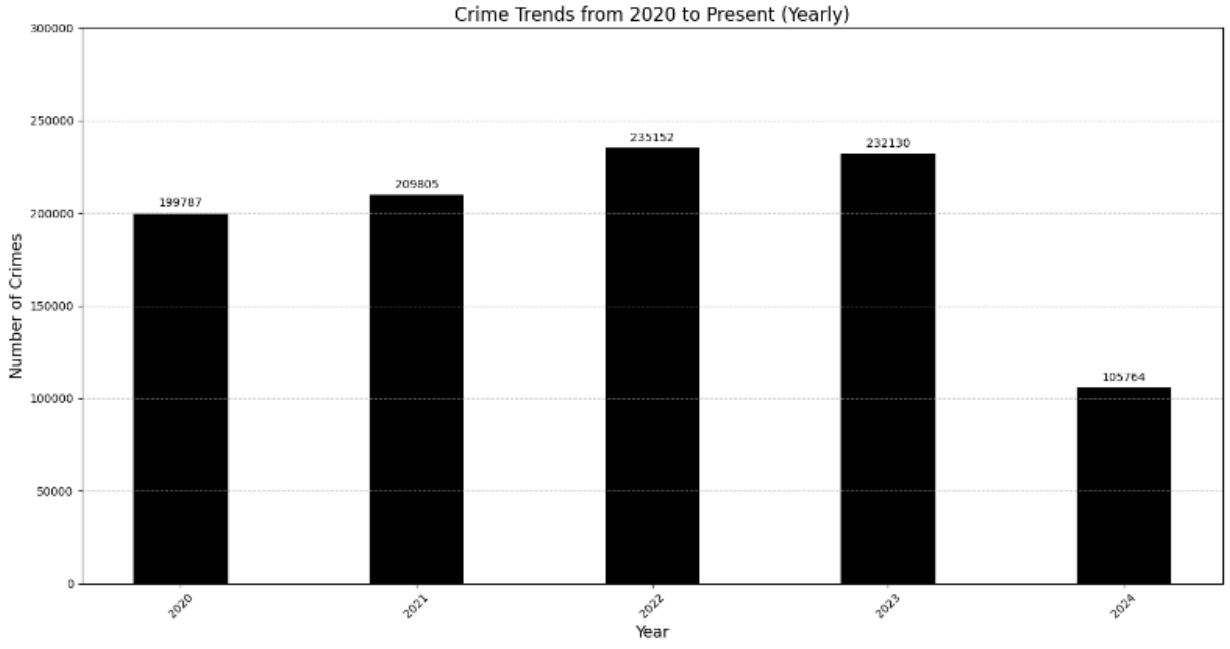
4. **Customization and Display:** The chart is customized with titles, axis labels, rotated x-axis ticks to prevent overlap, and a y-axis limit of 300,000 to ensure the chart fits well. A grid is also added for better readability.

Finally, the chart is displayed using plt.show().

Result: The results indicate that crime trends were highest in 2022, with a total of 235,152 reported crimes, and lowest in 2024, with only 105,764 reported crimes. This suggests a significant decline in crime from 2022 to 2024, which could be due to various factors such as changes in law enforcement, social policies, or public behavior. Analyzing these trends further might provide insights into the effectiveness of measures taken during this period.

INPUT and OUTPUT :

In [7]:
```
# Visualizing the crime trends from 2020 to present:

import pandas as pd
import matplotlib.pyplot as plt

# Loading the cleaned dataset

crime_data = pd.read_csv('Crime_Data_from_2020_to_Present.csv')

# Ensure 'DATE OCC' column is in datetime format
crime_data['DATE OCC'] = pd.to_datetime(crime_data['DATE OCC'], format='%m/%d/%Y %I:%M:%S %p')

# Extract the year from 'DATE OCC'
crime_data['Year'] = crime_data['DATE OCC'].dt.year

# Grouping by year and count the number of crimes
yearly_crime_counts = crime_data.groupby('Year').size().reset_index(name='Crime Counts')

# Creating the bar chart
plt.figure(figsize=(15, 8))
bars = plt.bar(yearly_crime_counts['Year'], yearly_crime_counts['Crime Counts'], color='black', width=0.4)

# Adding data labels on top of the bars
for x, y in zip(yearly_crime_counts['Year'], yearly_crime_counts['Crime Counts']):
    plt.text(x, y + 0.01 * max(yearly_crime_counts['Crime Counts']), f'{y}', ha='center', va='bottom')

plt.title('Crime Trends from 2020 to Present (Yearly)', fontsize=16)
plt.xlabel('Year', fontsize=14)
plt.ylabel('Number of Crimes', fontsize=14)
plt.xticks(yearly_crime_counts['Year'], rotation=45)  # Rotate x-ticks to avoid overlap
plt.ylim(0, 300000)  # Set y-axis limit to 300,000 as it was not fitting properly
plt.grid(axis='y', linestyle='--', alpha=0.6)

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



Analyze and visualize seasonal patterns in crime data:

This code visualizes seasonal patterns in crime data from 2020 to the present.

1. **Loading and Cleaning the Dataset:** The crime dataset is loaded from a CSV file, and the 'DATE OCC' column (which contains the date of occurrence) is converted into a datetime format.

2. **Extracting Year and Month:** The code extracts both the year and the full month name from the 'DATE OCC' column to enable grouping by both year and month.

3. **Grouping by Month and Year:** Crimes are grouped by each combination of year and month, counting the number of crimes for each period. The month names are set in chronological order (January to December).

4. **Pivot Table for Visualization:** A pivot table is created where each row represents a month and each column represents a year. The values in the table represent the number of crimes for the respective month and year. Missing values are filled with zeros.

5. **Plotting Seasonal Patterns:** A line plot is created to show the trend of crimes for each year, with markers for each month. Each line represents the crime trend across the months for a specific year, allowing easy comparison of seasonal crime patterns over multiple years.

6. **Customization:** The chart is customized with titles, axis labels, rotated x-ticks for better readability, and a grid for clarity. The legend identifies each line by year.

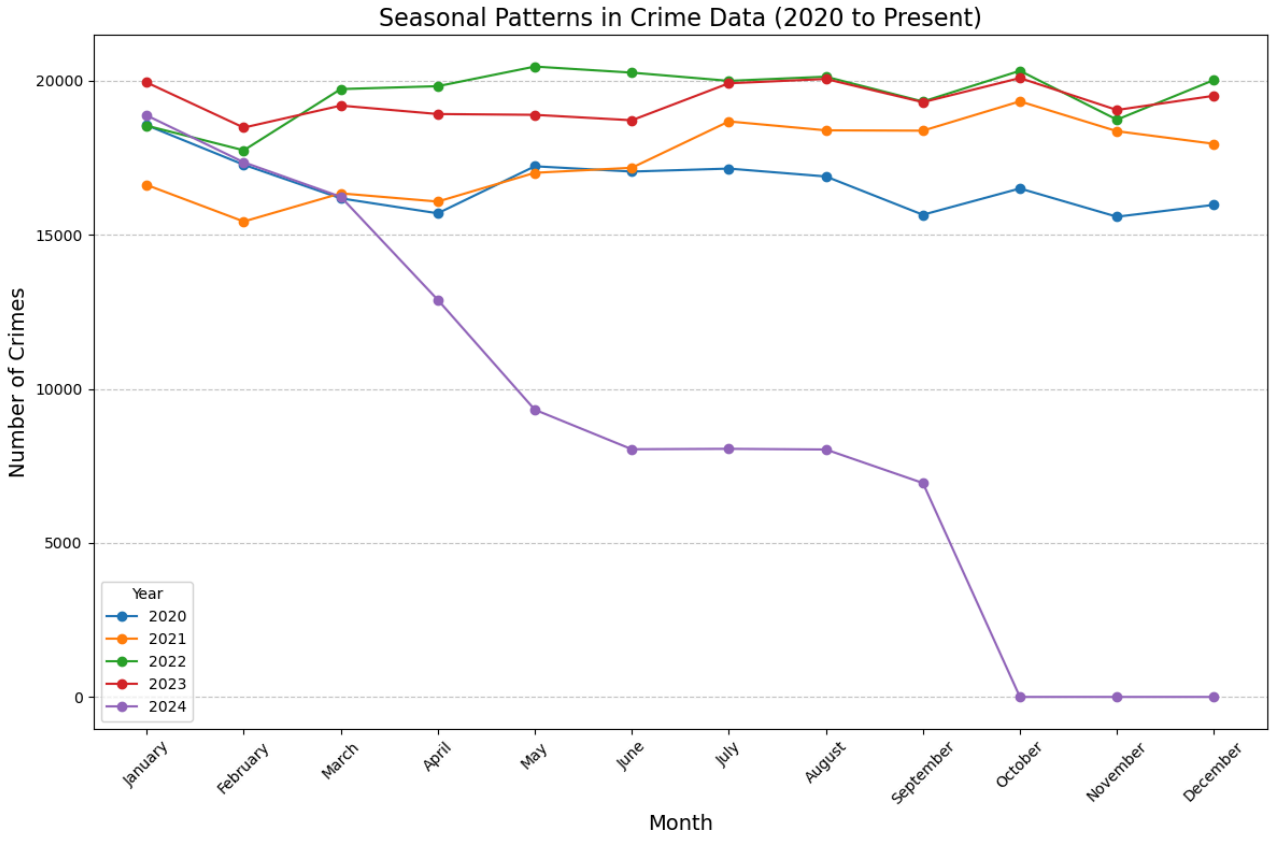
Finally, the chart is displayed using plt.show(). This plot helps visualize how crime rates fluctuate throughout the year and how these seasonal patterns have changed over time.

Result:

- The plot clearly shows that 2024 has a significant drop in crime numbers after May, reaching almost zero by October and staying that way for the rest of the year. This sharp decline could indicate incomplete data for 2024 or a substantial reduction in crime rates due to external factors (e.g., policy changes, data collection issues).
- For the years 2020 through 2023, the crime rates appear more consistent, with fluctuations between months but no significant drop-off.

INPUT and OUTPUT :

In [8]:
```
# Task 4 : Analyze and visualize seasonal patterns in crime data.

import pandas as pd
import matplotlib.pyplot as plt

# Load the cleaned dataset

crime_data = pd.read_csv('Crime_Data_from_2020_to_Present.csv')

# Ensuring 'DATE OCC' column is in datetime format
crime_data['DATE OCC'] = pd.to_datetime(crime_data['DATE OCC'], format='%m/%d/%Y %I:%M:%S %p')

# Extracting year and month from 'DATE OCC'
crime_data['Year'] = crime_data['DATE OCC'].dt.year
crime_data['Month'] = crime_data['DATE OCC'].dt.month_name()  # Get month name

# Grouping by month and year to count the number of crimes
monthly_crime_counts = crime_data.groupby(['Year', 'Month']).size().reset_index(name='Crime Counts')

# Sorting the data by year and month
months_order = ['January', 'February', 'March', 'April', 'May', 'June',
                'July', 'August', 'September', 'October', 'November', 'December']
monthly_crime_counts['Month'] = pd.Categorical(monthly_crime_counts['Month'], categories=months_order, ordered=True)
monthly_crime_counts = monthly_crime_counts.sort_values(['Year', 'Month'])

# Create a pivot table for visualization
pivot_table = monthly_crime_counts.pivot(index='Month', columns='Year', values='Crime Counts').fillna(0)

# Plot the seasonal patterns
plt.figure(figsize=(12, 8))
for year in pivot_table.columns:
    plt.plot(pivot_table.index, pivot_table[year], marker='o', label=str(year))

plt.title('Seasonal Patterns in Crime Data (2020 to Present)', fontsize=16)
plt.xlabel('Month', fontsize=14)
plt.ylabel('Number of Crimes', fontsize=14)
plt.xticks(rotation=45)
plt.legend(title='Year')
plt.grid(axis='y', linestyle='--', alpha=0.7)

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



Most common crime type:

Process:

1. Data Loading and Date Processing: The dataset is loaded and the 'DATE OCC' column is converted to a datetime format to extract the year of occurrence.

2. Crime Type Count: The code counts the occurrences of each crime type by using the value_counts() function. Most Common Crime: The most frequent crime type is identified as "Vehicle Stolen".

3. Filtering and Grouping Data: The dataset is filtered to include only records where the crime type is "Vehicle Stolen." The data is grouped by year to count how many vehicle thefts occurred each year.

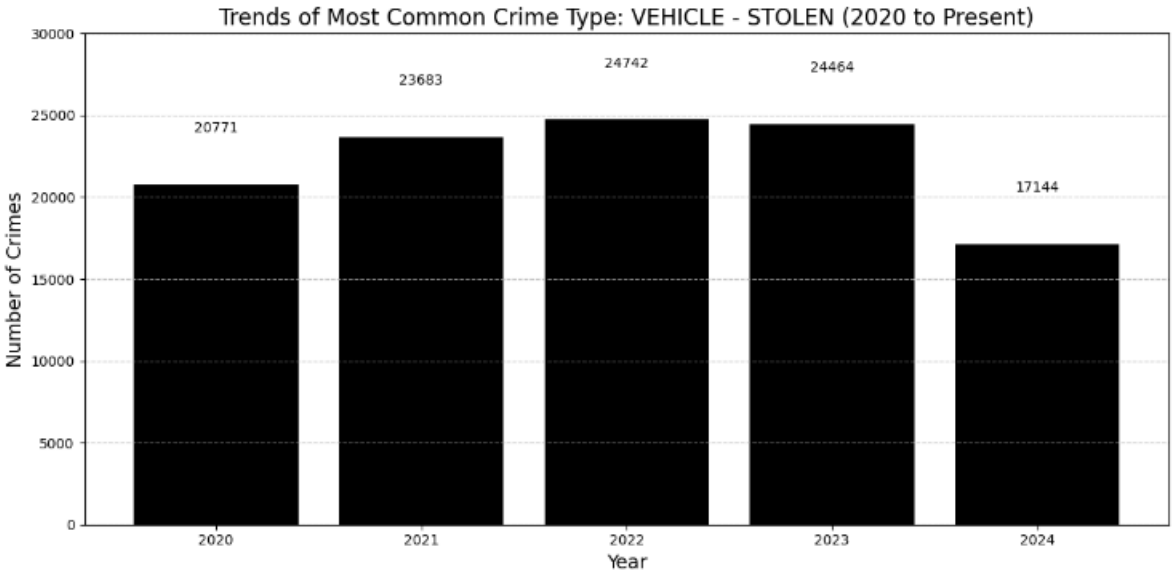
4. Visualization: A bar chart is created to visualize the trend of vehicle thefts over the years from 2020 to present.

Result: The code confirms that the most common crime type in your dataset is "Vehicle Stolen." The chart visualizes the yearly trends for this crime from 2020 onwards.

INPUT and OUTPUT :

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the cleaned dataset

crime_data = pd.read_csv('Crime_Data_from_2020_to_Present.csv')

# Ensure 'DATE OCC' column is in datetime format
crime_data['DATE OCC'] = pd.to_datetime(crime_data['DATE OCC'], format='%m/%d/%Y %I:%M:%S %p')

# Extract the year from 'DATE OCC'
crime_data['Year'] = crime_data['DATE OCC'].dt.year

# Step 1: Counting occurrences of each crime type
crime_type_counts = crime_data['Crm Cd Desc'].value_counts().reset_index()
crime_type_counts.columns = ['Crime Type', 'Count']

# Step 2: Identifying the most common crime type
most_common_crime_type = crime_type_counts.iloc[0]

print(f"Most Common Crime Type: {most_common_crime_type['Crime Type']} with {most_common_crime_type['Count']} occurrences.")

# Step 3: Filter the data for the most common crime type
most_common_crime_data = crime_data[crime_data['Crm Cd Desc'] == most_common_crime_type['Crime Type']]

# Group by year to count occurrences of the most common crime type
yearly_common_crime_counts = most_common_crime_data.groupby('Year').size().reset_index(name='Crime Counts')

# Step 4: Visualize the trends of the most common crime type over time
plt.figure(figsize=(12, 6))
plt.bar(yearly_common_crime_counts['Year'], yearly_common_crime_counts['Crime Counts'], color='black')

# Add data labels on top of the bars
for x, y in zip(yearly_common_crime_counts['Year'], yearly_common_crime_counts['Crime Counts']):
    plt.text(x, y + 0.01 * 300000, f'{y}', ha='center', va='bottom')

plt.title(f'Trends of Most Common Crime Type: {most_common_crime_type["Crime Type"]} (2020 to Present)', fontsize=16)
plt.xlabel('Year', fontsize=14)
plt.ylabel('Number of Crimes', fontsize=14)
plt.ylim(0, 30000)  # Set y-axis limit to 300,000
plt.grid(axis='y', linestyle='--', alpha=0.7)

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

```
Most Common Crime Type: VEHICLE - STOLEN with 110804 occurrences.
```



Investigating if there are any notable differences in crime rates between different regions:

1. **Loading Data:** The crime data is loaded, and the 'DATE OCC' column is converted into a datetime format.

2. **Crime Counts by Area:** The crimes are grouped by the 'AREA NAME' column to count how many crimes occurred in each area.

3. **Filtering Top Areas:** The data is filtered to show the top 11 areas with the highest crime counts for better visualization.

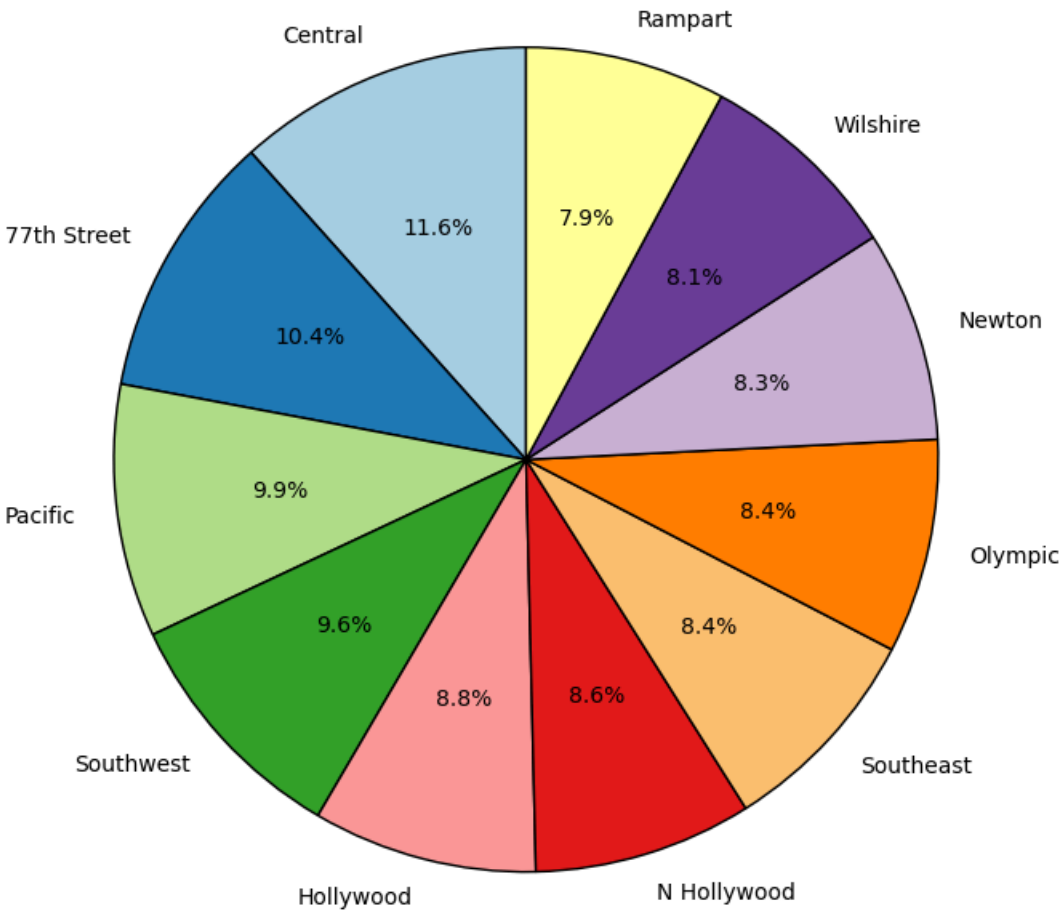
4. **Pie Chart:** The crime counts for the top areas are displayed as a pie chart with percentage labels. Used plt.pie() to create the pie chart. Setted labels, percentages, colors, and other visual properties. Also, Use plt.tight_layout() to adjust the layout and then plt.show() to display the pie chart.

Result : Investigation of Crime Differences Between Areas Based on the chart -

- Central area has the highest crime percentage (11.6%).
- Other areas, like 77th Street, Pacific, Southwest, and Hollywood, also have high crime counts. T
- The differences in crime percentages across these areas may reflect different population sizes, urban densities, or law enforcement practices.

INPUT and OUTPUT :

```
: # Task 4: Investigate if there are any notable differences in crime rates between regions or cities.

import pandas as pd
import matplotlib.pyplot as plt

# Load the cleaned dataset

crime_data = pd.read_csv('Crime_Data_from_2020_to_Present.csv')

# Ensure 'DATE OCC' column is in datetime format
crime_data['DATE OCC'] = pd.to_datetime(crime_data['DATE OCC'], format='%m/%d/%Y %I:%M:%S %p')

# Step 1: Count occurrences of crimes by area
crime_counts_by_area = crime_data['AREA NAME'].value_counts().reset_index()
crime_counts_by_area.columns = ['Area Name', 'Crime Counts']

# Step 2: Filter for the top N areas for better visualization (optional)
top_n = 11  # Adjust this value as needed
top_crime_counts = crime_counts_by_area.nlargest(top_n, 'Crime Counts')

# Step 3: Visualize the crime counts by area as a pie chart
plt.figure(figsize=(10, 7))
plt.pie(top_crime_counts['Crime Counts'], labels=top_crime_counts['Area Name'],
        autopct='%1.1f%%', startangle=90, colors=plt.cm.Paired.colors, wedgeprops={'edgecolor': 'black'})

# Equal aspect ratio ensures the pie is drawn as a circle.
plt.title('Top Areas by Crime Counts (2020 to Present)', fontsize=16)

# Display the pie chart
plt.tight_layout()
plt.show()
```


Top Areas by Crime Counts (2020 to Present)
Central
Rampart
Wilshire
77th Street
Newton
Pacific
Olympic
Southwest
Southeast
Hollywood
N Hollywood
11.6%
7.9%
8.1%
10.4%
8.3%
9.9%
8.4%
9.6%
8.4%
8.8%
8.6%

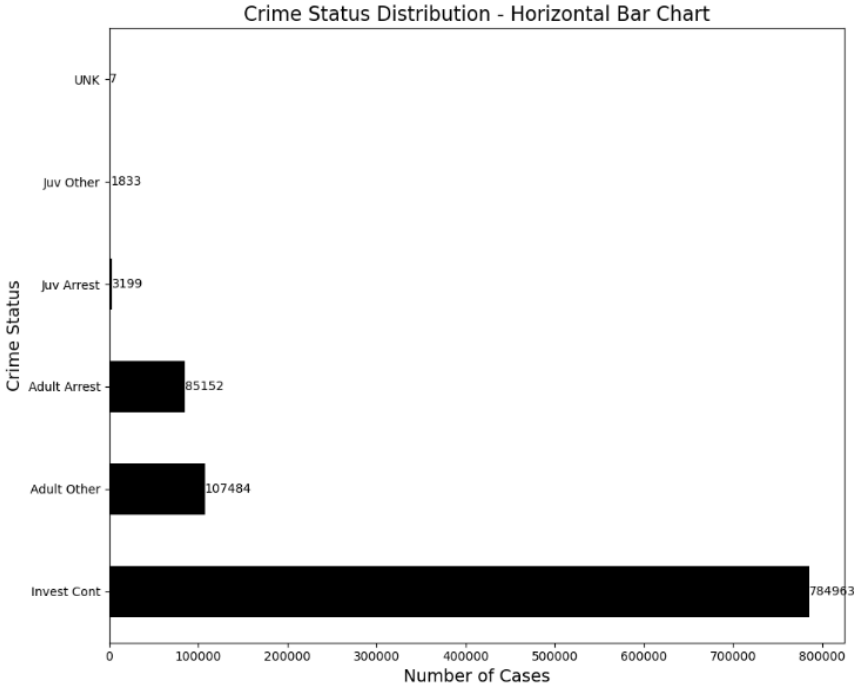
Exploring correlations between economic factors ie Area and crime rates :

- Crime Frequency Analysis : Crime Status Analysis
- Objective: Determining how often crimes occur over time to identify patterns or trends.
- We used the date columns (Date Rpdt, Date Occ) to analyze crime frequencies on a monthly, quarterly, or yearly basis.
- Outcome: You can create a time-series graph to visualize crime trends over time.

We used a Horizontal Bar Chart as it allows for easy comparison of different categories. It displays the frequency or count of items in each category horizontally.

INPUT and OUTPUT :

```
import pandas as pd
import matplotlib.pyplot as plt

import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset

df = pd.read_csv('Crime_Data_from_2020_to_Present.csv')

# Check if 'Status Desc' column has any missing values and fill them with 'Unknown' if needed
df['Status Desc'].fillna('Unknown', inplace=True)

# Get the distribution of crime statuses
crime_status = df['Status Desc'].value_counts()

# Plot a horizontal bar chart
plt.figure(figsize=(10, 8))
crime_status.plot(kind='barh', color='black')

# Add labels and title
plt.title('Crime Status Distribution - Horizontal Bar Chart', fontsize=16)
plt.xlabel('Number of Cases', fontsize=14)
plt.ylabel('Crime Status', fontsize=14)

# Show percentage on the bars
for index, value in enumerate(crime_status):
    plt.text(value, index, f'{value}', va='center', fontsize=10)

# Show the chart
plt.tight_layout()
plt.show()
```



Analyzing the relationship between the day of the week and the frequency of certain types of crimes:

We analyzed how crime frequency varies by day of the week, allowing for insights into crime patterns and trends.

1. **Data Loading:** The dataset containing crime records is loaded into a Pandas DataFrame. This dataset includes details about each crime incident, such as the date it occurred and the type of crime.

2. **Date Formatting:** The 'DATE OCC' column, which represents the occurrence date of each crime, is converted to a datetime format. This is essential for extracting day-related information accurately.

3. **Extracting Day of the Week:** A new column named 'Day of Week' is created by extracting the day name from the 'DATE OCC' column. This categorizes each crime by the day it occurred (e.g., Monday, Tuesday).

4. **Grouping Data:** The data is grouped by both 'Day of Week' and 'Crm Cd Desc' . The .size() method counts the number of occurrences for each combination, resulting in a DataFrame that shows how many crimes of each type occurred on each day of the week.

5. **Pivoting Data for Visualization:** The grouped data is reshaped into a pivot table format, where 'Day of Week' is the index and 'Crm Cd Desc' is the column. The values represent the counts of crimes, allowing for easier plotting.

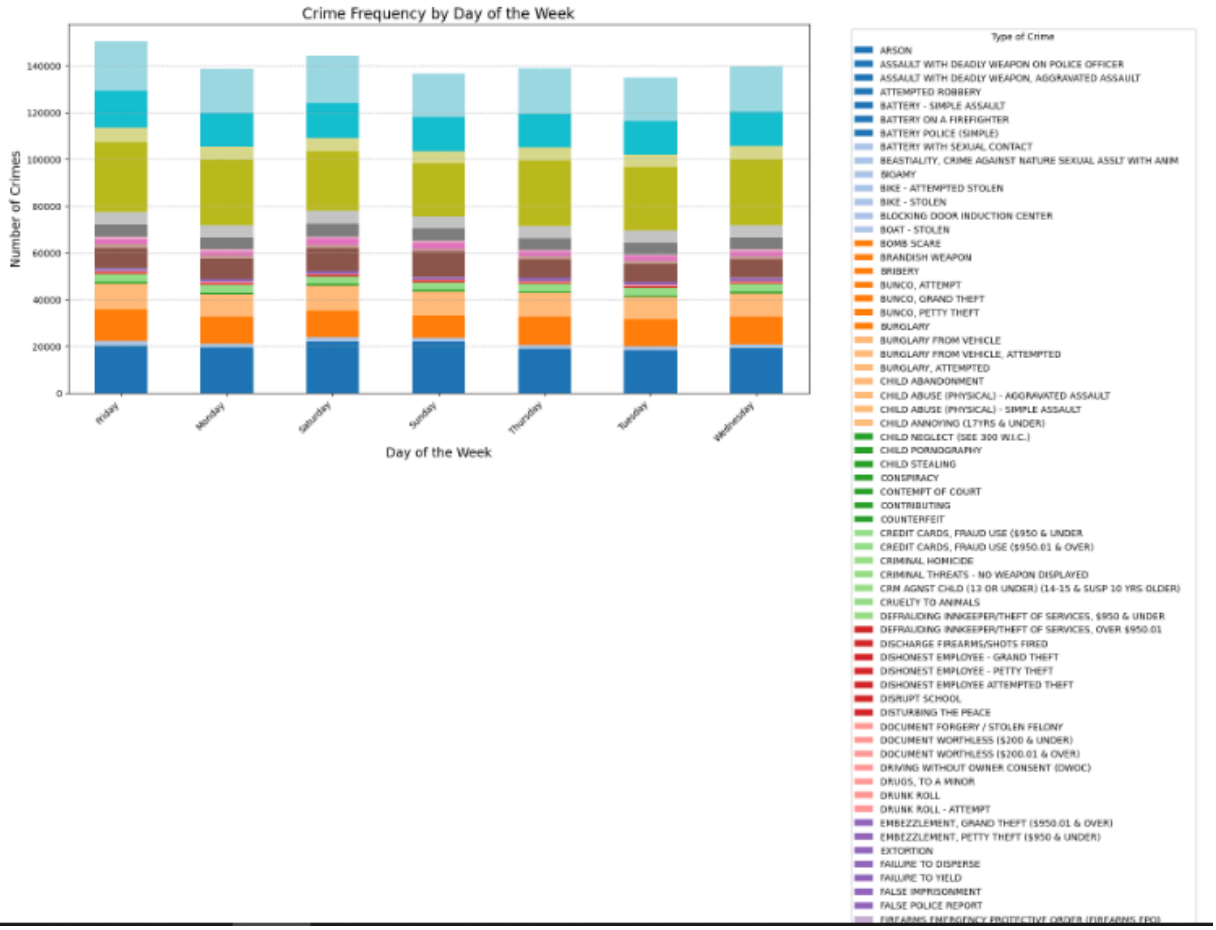
6. **Plotting:** A stacked bar chart is generated to visualize the frequency of different types of crimes for each day of the week. This allows for immediate visual comparison of crime types across days. The plot is customized with a title, axis labels, and a legend for clarity. The x-axis labels are rotated for better readability, and grid lines are added for easier interpretation of crime counts.

7. **Layout Adjustment:** The layout is adjusted to ensure that all elements of the plot are visible and well-organized.

INPUT and OUTPUT :

In [41]:
```
# Task 4: Analyze the relationship between the day of the week and the frequency of certain types of crimes.

import pandas as pd
import matplotlib.pyplot as plt

import pandas as pd
import matplotlib.pyplot as plt

# Load the cleaned dataset
file_path = 'Crime_Data_from_2020_to_Present.csv'  # Update with your cleaned file path
crime_data = pd.read_csv(file_path)

# Ensure 'DATE OCC' column is in datetime format
crime_data['DATE OCC'] = pd.to_datetime(crime_data['DATE OCC'], format='%m/%d/%Y %I:%M:%S %p')

# Step 1: Extract the day of the week from the 'DATE OCC' column
crime_data['Day of Week'] = crime_data['DATE OCC'].dt.day_name()

# Step 2: Group by day of the week and crime type, and count occurrences
crime_frequency_by_day = crime_data.groupby(['Day of Week', 'Crm Cd Desc']).size().reset_index(name='Crime Counts')

# Step 3: Pivot the table for better visualization
crime_pivot = crime_frequency_by_day.pivot(index='Day of Week', columns='Crm Cd Desc', values='Crime Counts').fillna(0)

# Step 4: Plotting
plt.figure(figsize=(18, 8))  # Increase figure size
crime_pivot.plot(kind='bar', stacked=True, figsize=(18, 8), colormap='tab20')

plt.title('Crime Frequency by Day of the Week', fontsize=16)
plt.xlabel('Day of the Week', fontsize=14)
plt.ylabel('Number of Crimes', fontsize=14)
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels
plt.legend(title='Type of Crime', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Adjusting layout
plt.subplots_adjust(bottom=0.2, right=0.75, top=0.9)  # Adjust margins

plt.show()
```

```
<Figure size 1800x800 with 0 Axes>
```



The types of crimes did not appear in the picture; please refer to the code in the PDF.

Time Series Analysis :

We performed a time series analysis to examine trends in gun-related crimes over time using a dataset containing crime reports.

The analysis involves several key steps:

1. **Data Loading and Inspection:** The dataset is loaded from a CSV file, and a preliminary inspection of the 'Date Rptd' column is conducted to understand its format.

2. **Date Conversion:** The 'Date Rptd' column is converted to a datetime format to facilitate time-based operations. This conversion ensures that dates are correctly recognized and can be used for indexing.

3. **Data Cleaning:** Any rows with invalid or missing dates (NaT) are dropped to maintain the integrity of the analysis.

4. **Indexing:** The 'Date' column is set as the index of the DataFrame, which allows for easy resampling based on time periods.

5. **Filtering:** The dataset is filtered to include only gun-related crimes by checking if the 'Weapon Desc' column contains the word "gun". This step is crucial for focusing the analysis on the specific type of crime.

6. **Resampling:** The data is resampled by month to count the number of occurrences of gun-related crimes. This aggregation helps to identify trends over time, smoothing out daily fluctuations and allowing for clearer visualizations.

7. **Visualization:** A line plot is created to visualize the trend of gun-related crimes over time. The x-axis represents the date, while the y-axis indicates the number of crimes. The plot includes a title, labels, and a grid for better readability.

By analyzing the resulting time series plot, stakeholders can observe trends, spikes, or declines in gun-related crime incidents, potentially correlating these trends with significant events or policy changes.

INPUT and OUTPUT :

In [15]:
```
# Task-4: Time Series Analysis:
#Use time series analysis to examine crime trends over time.
#This involves plotting crime rates against time and marking significant events on the same graph.

import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('Crime_Data_from_2020_to_Present.csv')

# Check the first few rows to inspect the format
print(df['Date Rptd'].head())

# Convert 'Date Rpdt' column to datetime format including time
df['Date'] = pd.to_datetime(df['Date Rptd'], format='%m/%d/%Y %I:%M:%S %p', errors='coerce')

# Drop rows where 'Date' is NaT (Not a Time)
df.dropna(subset=['Date'], inplace=True)

# Set the 'Date' column as the index for resampling
df.set_index('Date', inplace=True)

# Filter data for gun-related crimes
gun_related_crimes = df[df['Weapon Desc'].str.contains('gun', case=False, na=False)]

# Resample data by month and count occurrences of gun-related crimes
monthly_gun_crimes = gun_related_crimes.resample('M').size()

# Plot the time series of gun-related crimes over time
plt.figure(figsize=(10, 6))
plt.plot(monthly_gun_crimes, label='Gun-Related Crimes')
plt.title('Gun-Related Crimes Over Time')
plt.xlabel('Date')
plt.ylabel('Number of Crimes')
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```

```
0    03/01/2020 12:00:00 AM
1    02/09/2020 12:00:00 AM
2    11/11/2020 12:00:00 AM
3    05/10/2023 12:00:00 AM
4    08/18/2022 12:00:00 AM
Name: Date Rptd, dtype: object
```