About Dataset

This dataset contains official crime records reported in Los Angeles City from January 2020 to December 2023.

The data provides valuable information about reported crimes, including the date, area, crime details, victim information, premises, weapons used, and status.

Reading Data Set:

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import pickle
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
df=pd.read_csv('Crime Dataset.csv')
```

In [3]:
```
df
```

Out[3]:

d	date_occurred	area	area_name	reporting_district	part	crime_code	crime_description	modus_operandi	...	status	status_description	crime_cod
08	2020-01-08 22:30:00	3	Southwest	377	2	624	BATTERY - SIMPLE ASSAULT	0444 0913	...	AO	Adult Other	62
02	2020-01-01 03:30:00	1	Central	163	2	624	BATTERY - SIMPLE ASSAULT	0416 1822 1414	...	IC	Invest Cont	62
14	2020-02-13 12:00:00	1	Central	155	2	845	SEX OFFENDER REGISTRANT OUT OF COMPLIANCE	1501	...	AA	Adult Arrest	84
01	2020-01-01 17:30:00	15	N Hollywood	1543	2	745	VANDALISM - MISDEAMEANOR ($399 OR UNDER)	0329 1402	...	IC	Invest Cont	74
01	2020-01-01 04:15:00	19	Mission	1998	2	740	VANDALISM - FELONY ($400 & OVER, ALL CHURCH VA...	0329	...	IC	Invest Cont	74
...	...	...	...	...	...	...	...	...	...	...	...	
22	2023-03-22 10:00:00	16	Foothill	1602	1	230	ASSAULT WITH DEADLY WEAPON, AGGRAVATED ASSAULT	0416 0411 1822	...	IC	Invest Cont	23
12	2023-04-12 16:30:00	12	77th Street	1239	1	230	ASSAULT WITH DEADLY WEAPON, AGGRAVATED ASSAULT	0601 0445 0416 0359	...	IC	Invest Cont	23
02	2023-07-01 00:01:00	1	Central	154	1	352	PICKPOCKET	1822 0344	...	IC	Invest Cont	35
05	2023-03-05 09:00:00	9	Van Nuys	914	2	745	VANDALISM - MISDEAMEANOR ($399 OR UNDER)	0329 1822	...	IC	Invest Cont	74
10	2023-11-09 23:00:00	3	Southwest	395	1	331	THEFT FROM MOTOR VEHICLE - GRAND ($950.01 AND ...	1822 1606 0344 0385 1300	...	IC	Invest Cont	33

About Colums:

In [4]:

```
#division_number:  Numeric code representing the division
#date_reported:   Date when the crime was reported
#date_occurred  Actual date and time when the crime occurre
#area   Numeric code representing the area
#area_name  Name of the area where the crime occurred
#reporting_district Numeric code of the reporting district
#part   Part number of the crime
#crime_description  Detailed description of the crime
#crime_code Numeric code representing the type of crime
#modus_operandi Methods or patterns in the crime execution
#victim_age Age of the victim
#victim_sex Gender of the victim
#victim_descent Ethnic descent of the victim
  #The "victim_descent" attribute may contain categorical values or codes that represent various ethnic or racial backgr

     #W: White
     #B: Black or African American
     #H: Hispanic or Latino
     #A: Asian
     #O: Other
     # Unknown or Not Specified



#premise_code   Code for the type of location of the crime
#premise_description    Description of the premise where crime occurred
#weapon_code    Code for the weapon used (if any)
#weapon_description Description of the weapon used
#status Status of the crime report
#status_description Detailed status of the crime
#crime_code_1   Additional code related to the crime
#crime_code_2   Additional code related to the crime
#crime_code_3   Additional code related to the crime
#crime_code_4   Additional code related to the crime
#location   General location description of the crime
#cross_street   Nearby cross street (if applicable)
#latitude   Latitude coordinate of the crime location
#longitude  Longitude coordinate of the crime location
```

Checking for null values

In [5]:

```
df.isnull().sum()
```

Out[5]:

```
division_number             0
date_reported               0
date_occurred               0
area                        0
area_name                   0
reporting_district          0
part                        0
crime_code                  0
crime_description           0
modus_operandi         118311
victim_age                  0
victim_sex             112606
victim_descent         112614
premise_code               10
premise_description       518
weapon_code            556202
weapon_description     556202
status                      0
status_description          0
crime_code_1               11
crime_code_2           790429
crime_code_3           850837
crime_code_4           852888
location                    0
cross_street           717289
latitude                    0
longitude                   0
dtype: int64
```

Getting Datatypes

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 852950 entries, 0 to 852949
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   division_number       852950 non-null  int64  
 1   date_reported         852950 non-null  object 
 2   date_occurred         852950 non-null  object 
 3   area                  852950 non-null  int64  
 4   area_name             852950 non-null  object 
 5   reporting_district    852950 non-null  int64  
 6   part                  852950 non-null  int64  
 7   crime_code            852950 non-null  int64  
 8   crime_description     852950 non-null  object 
 9   modus_operandi        734639 non-null  object 
 10  victim_age            852950 non-null  int64  
 11  victim_sex            740344 non-null  object 
 12  victim_descent        740336 non-null  object 
 13  premise_code          852940 non-null  float64
 14  premise_description   852432 non-null  object 
 15  weapon_code           296748 non-null  float64
 16  weapon_description    296748 non-null  object 
 17  status                852950 non-null  object 
 18  status_description    852950 non-null  object 
 19  crime_code_1          852939 non-null  float64
 20  crime_code_2          62521 non-null   float64
 21  crime_code_3          2113 non-null    float64
 22  crime_code_4          62 non-null      float64
 23  location              852950 non-null  object 
 24  cross_street          135661 non-null  object 
 25  latitude              852950 non-null  float64
 26  longitude             852950 non-null  float64
dtypes: float64(8), int64(6), object(13)
memory usage: 175.7+ MB
```

Replace Missing Values

In [7]:

```
df.fillna("Unknown",inplace=True)
df.isnull().sum()
```

Out[7]:

```
division_number        0
date_reported          0
date_occurred          0
area                   0
area_name              0
reporting_district     0
part                   0
crime_code             0
crime_description      0
modus_operandi         0
victim_age             0
victim_sex             0
victim_descent         0
premise_code           0
premise_description    0
weapon_code            0
weapon_description     0
status                 0
status_description     0
crime_code_1           0
crime_code_2           0
crime_code_3           0
crime_code_4           0
location               0
cross_street           0
latitude               0
longitude              0
dtype: int64
```

Converting date column to date time

In [8]:
```
df['date_reported']=pd.to_datetime(df['date_reported'])
#here date reported is in object and by using pandas we are converting it into date time object
```

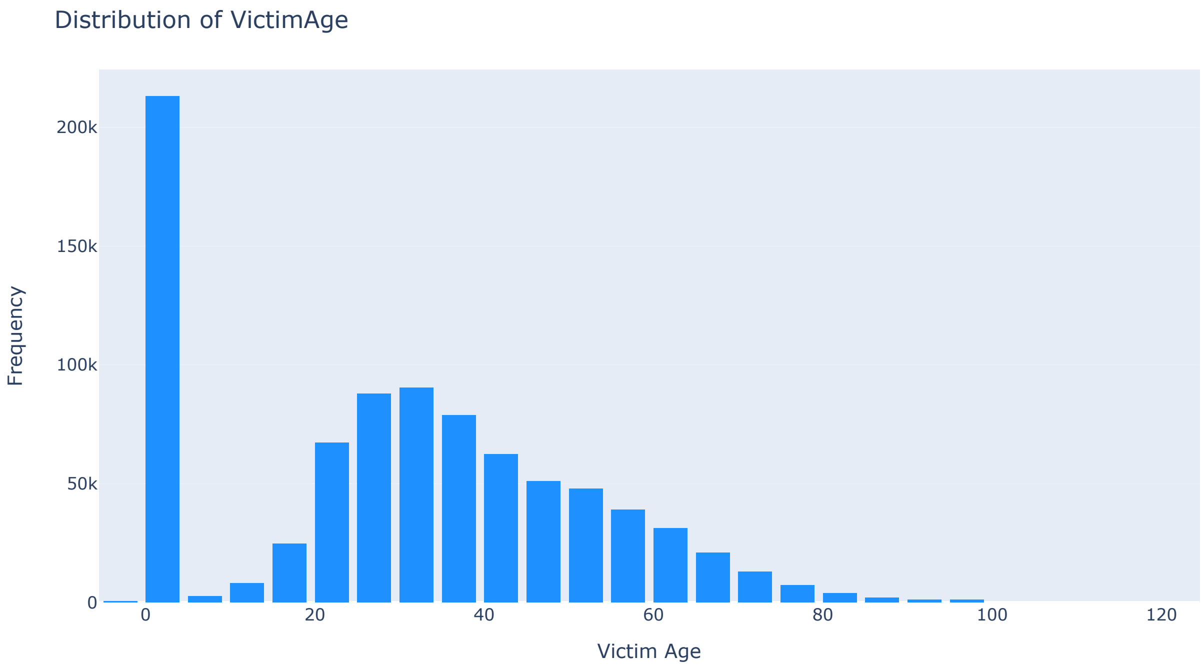
In [9]:
```
df['date_occurred']=pd.to_datetime(df['date_occurred'])
```

In [10]:
```
#plotly library is used for visualization
#Plotly Express is a high-level interface for creating a variety of interactive plots quickly and easily.
#Plotly Graph Objects is a lower-level interface that provides more control and customization options for creating compl
#make_subplots from plotly.subplots:
#The make_subplots function allows you to create complex subplots, i.e., arranging multiple plots in a single figure wit
```

In [11]:
```
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

In [12]:
```
# nbins parameter specifies how many intervals or bins the range of data values will be divided into along the x-axis.
#bar gap leads to gaps between bars
```

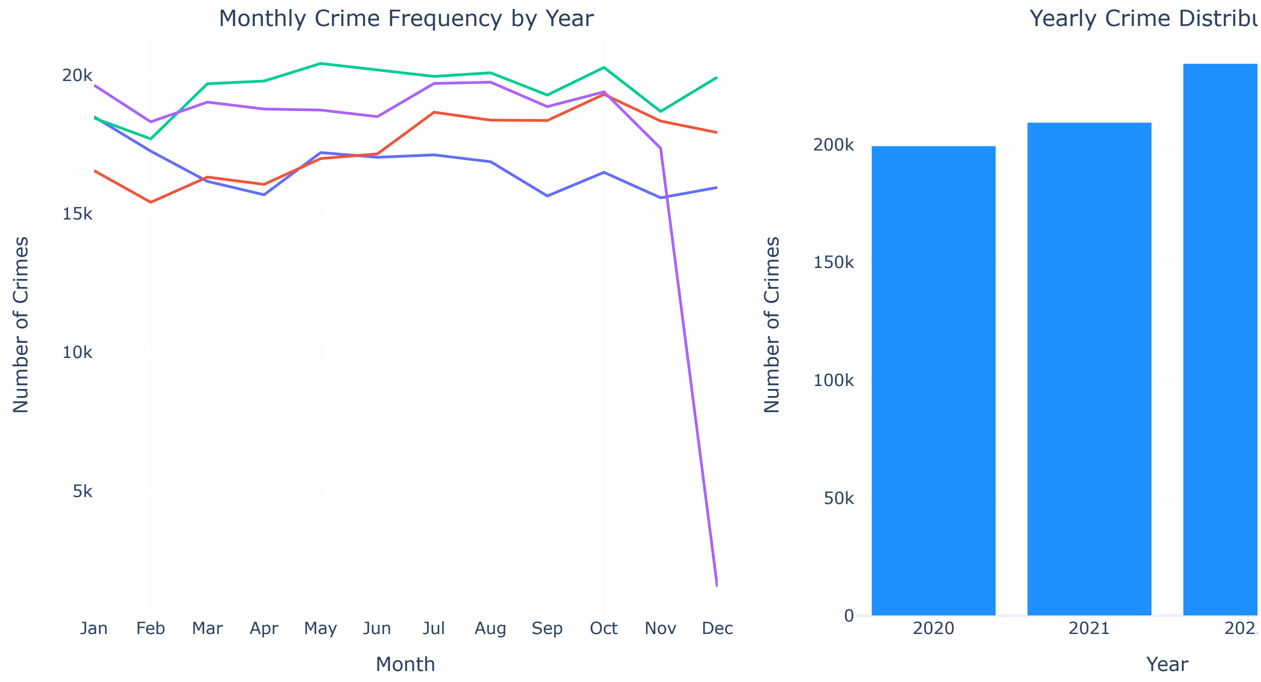
In [13]:
```
fig=px.histogram(df,x='victim_age',nbins=30,color_discrete_sequence=['dodgerblue'])
fig.update_layout(title='Distribution of VictimAge',xaxis_title='Victim Age',yaxis_title='Frequency',bargap=0.2)
fig.show()
#Replacing invalid ages(0 and negative values) with NAN
df['victim_age']=df['victim_age'].apply(lambda x:x if x>0 else None )
df['victim_age'].replace(0,pd.NA,inplace=True)
fig=px.histogram(df,x='victim_age',nbins=30,color_discrete_sequence=['dodgerblue'])
```



In [14]:
```
#pivot_table=monthly_crime_counts.pivot(index='month',columns='year',values='crime_count'):
#index='month': The 'month' column from monthly_crime_counts will be used as the index (rows) in the pivot table.
#columns='year': The 'year' column from monthly_crime_counts will be used to create separate columns in the pivot table.
#values='crime_count': The values in the pivot table will be populated from the 'crime_count' column in monthly_crime_co
#'''fig.add_trace(): This method is used to add a trace (plot) to a specific subplot within the figure (fig).

#go.Scatter(): This is the Plotly Graph Objects function used to create a scatter plot (or line plot, in this case).

#x=pivot_table.index: Specifies the x-axis data for the scatter plot. It uses the index of the pivot_table DataFrame, wh

#y=pivot_table[year]: Indicates the y-axis data for the scatter plot. name=str(year)) converts value to a string #'''
```

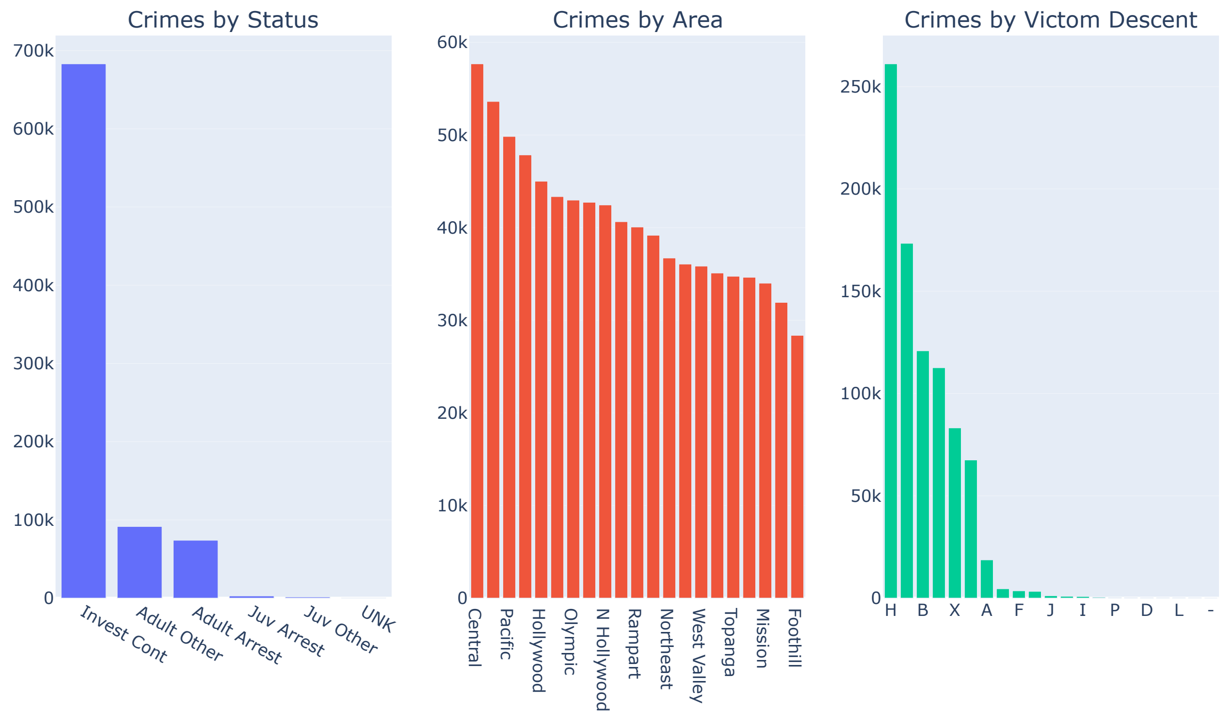
In [15]:

```
#Creating Columns for year and month
df['year']=df['date_occurred'].dt.year
df['month']=df['date_occurred'].dt.month
#craeting a pivot table for monthly crime counts
monthly_crime_counts=df.groupby(['year','month']).size().reset_index(name='crime_count')
pivot_table=monthly_crime_counts.pivot(index='month',columns='year',values='crime_count')
#creating yearly crime counts and sorting out based on ascending order
yearly_crime_counts=df['year'].value_counts().sort_index()
#creating subplots (make_subplots(): This function from Plotly is used to create subplots within a single figure. )with
fig=make_subplots(rows=1,cols=2,subplot_titles=("Monthly Crime Frequency by Year","Yearly Crime Distribution"))
#plotting Monthly Crime Frequency by Year
for year in pivot_table.columns:
    fig.add_trace(go.Scatter(x=pivot_table.index, y=pivot_table[year], mode='lines', name=str(year)), row=1, col=1)

# Plotting Yearly Crime Distribution
fig.add_trace(go.Bar(x=yearly_crime_counts.index, y=yearly_crime_counts.values, marker_color='dodgerblue'), row=1, col=2
#updating layout for plots
fig.update_layout(height=600,width=1200,template='plotly_white',showlegend=True)
fig.update_xaxes(title_text='Month', row=1, col=1, tickmode='array', tickvals=list(range(1, 13)), ticktext=['Jan', 'Feb'
fig.update_xaxes(title_text='Year', row=1, col=2)
fig.update_yaxes(title_text='Number of Crimes', row=1, col=1)
fig.update_yaxes(title_text='Number of Crimes', row=1, col=2)
fig.show()
```



#Count Plots

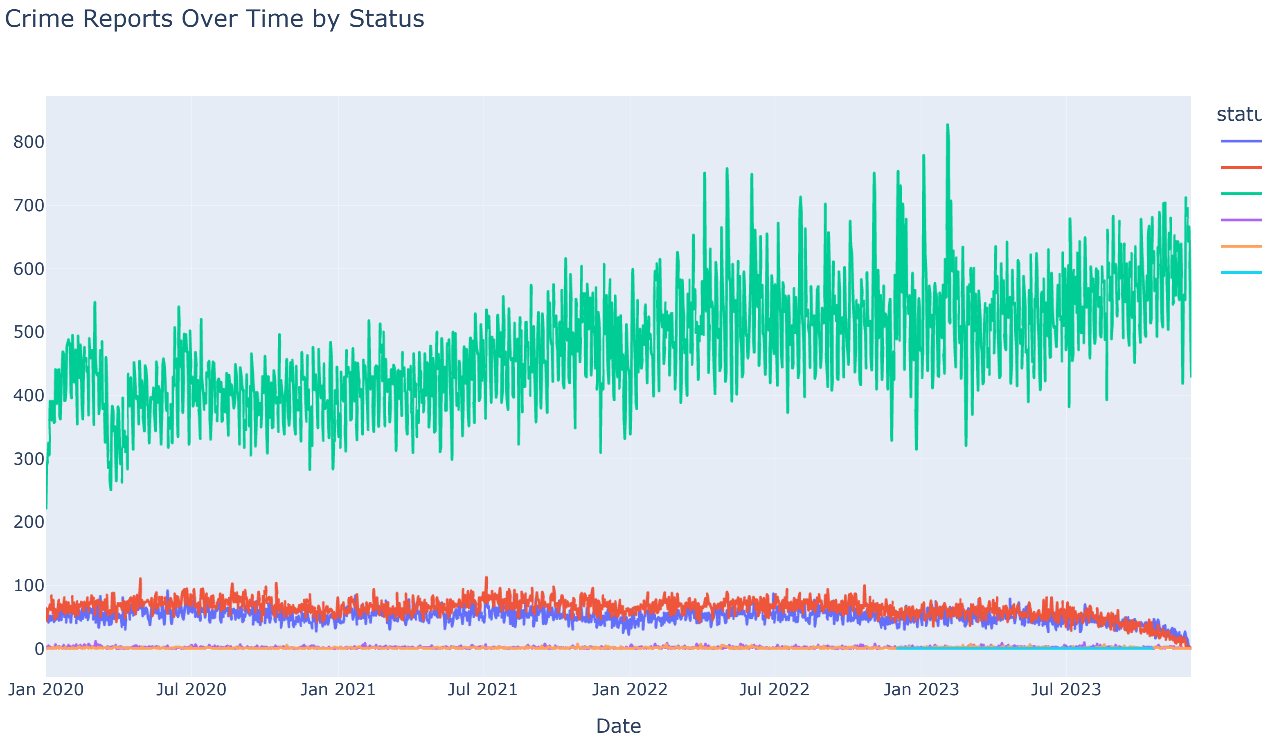
```
In [16]:  # Preparing data for bar plots
          status_counts = df['status_description'].value_counts()
          area_counts = df['area_name'].value_counts()
          victim_descent_counts = df['victim_descent'].value_counts()

          # Creating subplots
          fig = make_subplots(rows=1, cols=3, subplot_titles=("Crimes by Status", "Crimes by Area", 'Crimes by Victom Descent'))

          fig.add_trace(go.Bar(x=status_counts.index, y=status_counts.values, name="Status"), row=1, col=1)
          fig.add_trace(go.Bar(x=area_counts.index, y=area_counts.values, name="Area"), row=1, col=2)
          fig.add_trace(go.Bar(x=victim_descent_counts.index, y=victim_descent_counts.values, name="Victim Descent"), row=1, col=3
          fig.update_layout(height=600, width=1000, showlegend=False)
          fig.show()
```



Time Series Plots

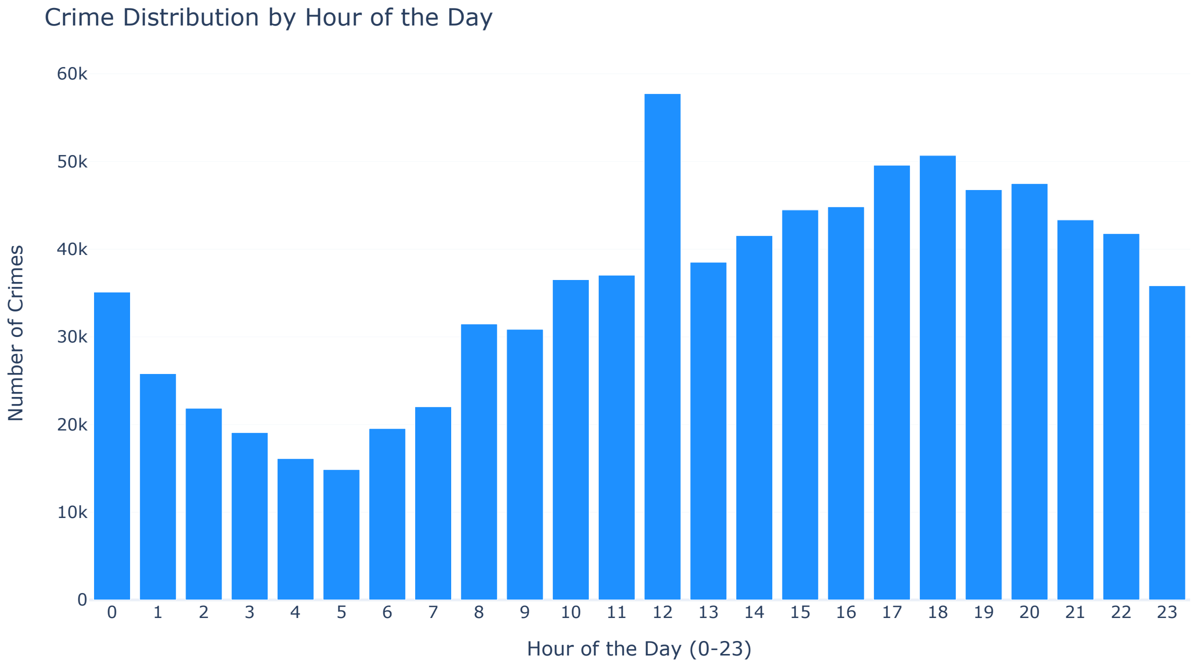
In [17]:

```
# Grouping data by date and status
time_series_status = df.groupby([df['date_reported'].dt.date, 'status']).size().reset_index(name='counts')

# Line plot
fig = px.line(time_series_status, x='date_reported', y='counts', color='status', title='Crime Reports Over Time by Statu
fig.update_xaxes(title_text='Date')
fig.update_yaxes(title_text='Number of Crimes Reported')
fig.update_layout(height=600, width=1000)
fig.show()
```



In [18]:

```
# Extracting hour from the 'date_occurred' column
df['hour'] = df['date_occurred'].dt.hour

# Counting crimes by each hour of the day
hourly_crime_counts = df['hour'].value_counts().sort_index()

# Plotting the distribution of crimes by hour
fig = px.bar(x=hourly_crime_counts.index, y=hourly_crime_counts.values, labels={'x': 'Hour of the Day (0-23)', 'y': 'Num

# Updating layout for the plot
fig.update_layout(title='Crime Distribution by Hour of the Day',
    template='plotly_white',
    showlegend=False
)

fig.update_xaxes(
    tickmode='array',
    tickvals=list(range(24)),
    ticktext=[str(hour) for hour in range(24)]
)

# Display the plot
fig.show()
```



In [19]:

```
unique_sex = df['victim_sex'].unique()
unique_descent = df['victim_descent'].unique()

unique_sex, unique_descent
```

Out[19]:

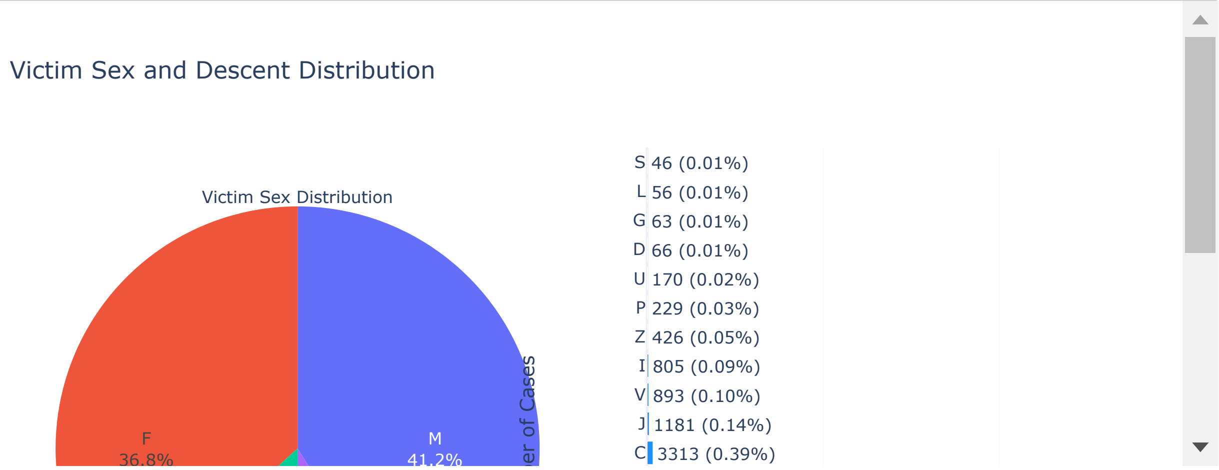
```
(array(['F', 'M', 'X', 'Unknown', 'H', '-'], dtype=object),
 array(['B', 'H', 'X', 'W', 'A', 'O', 'Unknown', 'C', 'F', 'K', 'I', 'V',
        'Z', 'J', 'P', 'G', 'U', 'D', 'S', 'L', '-'], dtype=object))
```

In [20]:

```
df['victim_sex'] = df['victim_sex'].replace(['H', '-'], 'Unknown')
df['victim_descent'] = df['victim_descent'].replace('-', 'Unknown')
```

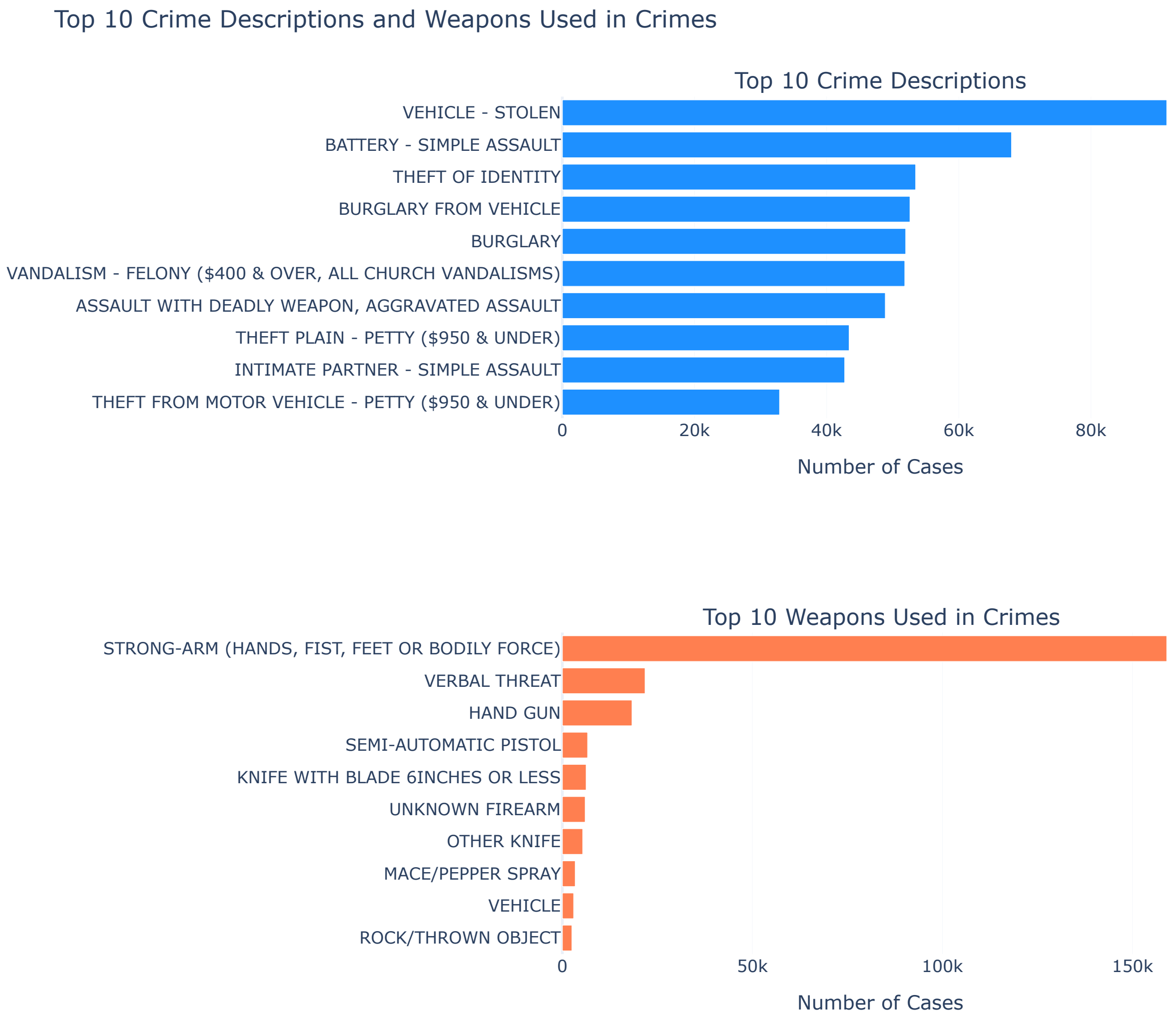
In [21]:

```
# Data for ploting
import matplotlib.pyplot as plt
victim_sex_data = df['victim_sex'].value_counts()
victim_descent_data = df['victim_descent'].value_counts()
total_cases = victim_descent_data.sum()

# Create subplots: 1 row, 2 columns
fig = make_subplots(rows=1, cols=2, specs=[[{"type": "pie"}, {"type": "bar"}]])

# Pie plot for victim_sex
fig.add_trace(
    go.Pie(
        labels=victim_sex_data.index,
        values=victim_sex_data,
        title="Victim Sex Distribution",
        textinfo='label+percent',
        insidetextorientation='radial'
    ),
    row=1, col=1
)

#fig = make_subplots(rows=1, cols=2, subplot_titles=('Pie Chart 1', 'Pie Chart 2'))
#plt.pie(df['victim_sex'], labels=df['victim_sex], autopct='%1.1f%%', startangle=140)
#plt.pie(df['victim_sex'].value_counts(), labels=df['victim_sex'].unique(), autopct='%1.1f%%', startangle=140)
#plt.title('Victim Sex Distribution')
#plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
#plt.show()

# Horizontal bar chart for victim_descent
fig.add_trace(
    go.Bar(
        x=victim_descent_data.values,
        y=victim_descent_data.index,
        orientation='h',
        marker_color='dodgerblue',
        text=[f"{count} ({count/total_cases:.2%})" for count in victim_descent_data.values],
        textposition='outside'
    ),
    row=1, col=2
)



# Update layout for the bar chart
fig.update_layout(
    title_text="Victim Sex and Descent Distribution",
    template='plotly_white',
    showlegend=False,
    height=600
)

fig.update_yaxes(title_text="Number of Cases", row=1, col=2)
fig.update_xaxes(title_text="Victim Descent", row=1, col=2)

# Display the plot
fig.show()
```



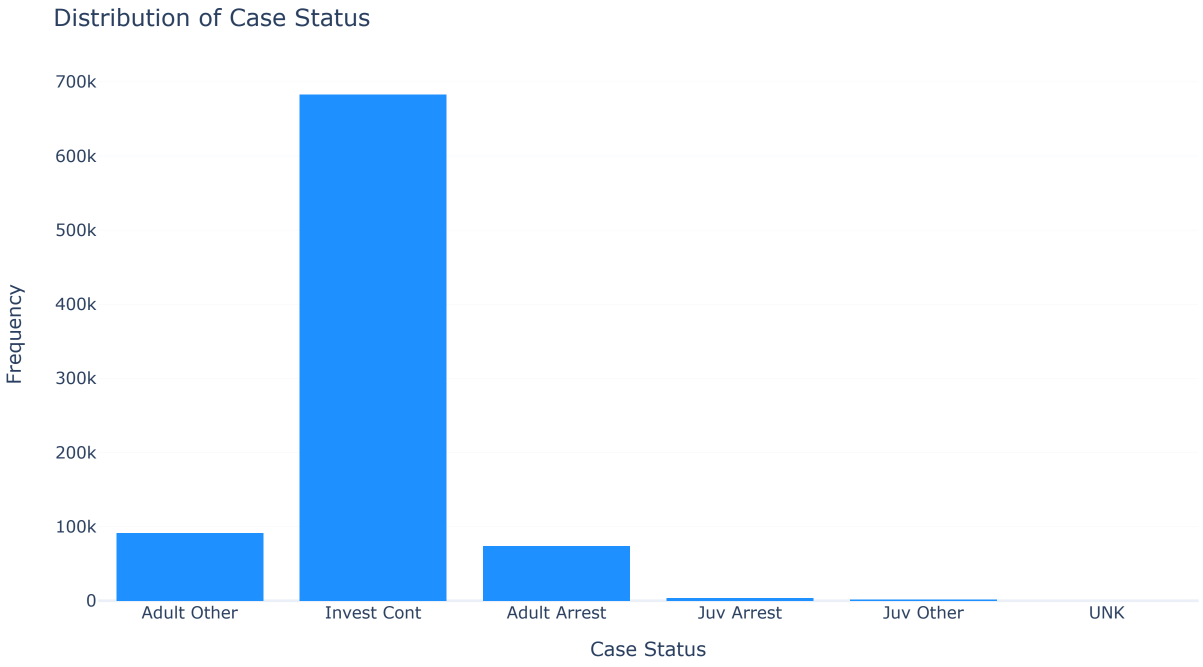
In [22]:

```python
# Top 10 most common crime descriptions (excluding 'Unknown')
top_crimes = df[df['crime_description'] != 'Unknown']['crime_description'].value_counts().head(10)

# Top 10 most common weapons (excluding 'Unknown', 'UNKNOWN WEAPON/OTHER WEAPON')
top_weapons = df[~df['weapon_description'].isin(['Unknown', 'UNKNOWN WEAPON/OTHER WEAPON'])]['weapon_description'].value

# Setting up the figure with two subplots
fig = make_subplots(rows=2, cols=1, subplot_titles=('Top 10 Crime Descriptions', 'Top 10 Weapons Used in Crimes'))
# Horizontal bar chart for top 10 crime descriptions
fig.add_trace(
    go.Bar(x=top_crimes.values, y=top_crimes.index, orientation='h', marker_color='dodgerblue'),
    row=1, col=1
)

# Horizontal bar chart for top 10 weapons used
fig.add_trace(
    go.Bar(x=top_weapons.values, y=top_weapons.index, orientation='h', marker_color='coral'),
    row=2, col=1
)
# Update layout for the charts
fig.update_layout(
    height=800,
    showlegend=False,
    template='plotly_white',
    title_text="Top 10 Crime Descriptions and Weapons Used in Crimes"
)

# Inverting y-axis for both plots to display the highest value at the top
fig.update_yaxes(autorange="reversed", row=1, col=1)
fig.update_yaxes(autorange="reversed", row=2, col=1)
# Update x-axis titles
fig.update_xaxes(title_text="Number of Cases", row=1, col=1)
fig.update_xaxes(title_text="Number of Cases", row=2, col=1)

# Display the plot
fig.show()
```


Top 10 Crime Descriptions and Weapons Used in Crimes
Top 10 Crime Descriptions
VEHICLE - STOLEN
BATTERY - SIMPLE ASSAULT
THEFT OF IDENTITY
BURGLARY FROM VEHICLE
BURGLARY
VANDALISM - FELONY ($400 & OVER, ALL CHURCH VANDALISMS)
ASSAULT WITH DEADLY WEAPON, AGGRAVATED ASSAULT
THEFT PLAIN - PETTY ($950 & UNDER)
INTIMATE PARTNER - SIMPLE ASSAULT
THEFT FROM MOTOR VEHICLE - PETTY ($950 & UNDER)
0
20k
40k
60k
80k
Number of Cases
Top 10 Weapons Used in Crimes
STRONG-ARM (HANDS, FIST, FEET OR BODILY FORCE)
VERBAL THREAT
HAND GUN
SEMI-AUTOMATIC PISTOL
KNIFE WITH BLADE 6INCHES OR LESS
UNKNOWN FIREARM
OTHER KNIFE
MACE/PEPPER SPRAY
VEHICLE
ROCK/THROWN OBJECT
0
50k
100k
150k
Number of Cases

In [23]:

```
# Create a histogram for the distribution of status
fig = px.histogram(df, x='status_description', color_discrete_sequence=['dodgerblue'])

# Updating layout for the plot
fig.update_layout(
    title='Distribution of Case Status',
    xaxis_title='Case Status',
    yaxis_title='Frequency',
    bargap=0.2,
    template='plotly_white'
)
fig.show()
```



In [24]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 852950 entries, 0 to 852949
Data columns (total 30 columns):
 #   Column                Non-Null Count   Dtype         
---  ------                --------------   -----         
 0   division_number       852950 non-null  int64         
 1   date_reported         852950 non-null  datetime64[ns]
 2   date_occurred         852950 non-null  datetime64[ns]
 3   area                  852950 non-null  int64         
 4   area_name             852950 non-null  object        
 5   reporting_district    852950 non-null  int64         
 6   part                  852950 non-null  int64         
 7   crime_code            852950 non-null  int64         
 8   crime_description     852950 non-null  object        
 9   modus_operandi        852950 non-null  object        
 10  victim_age            641034 non-null  float64       
 11  victim_sex            852950 non-null  object        
 12  victim_descent        852950 non-null  object        
 13  premise_code          852950 non-null  object        
 14  premise_description   852950 non-null  object        
 15  weapon_code           852950 non-null  object        
 16  weapon_description    852950 non-null  object        
 17  status                852950 non-null  object        
 18  status_description    852950 non-null  object        
 19  crime_code_1          852950 non-null  object        
 20  crime_code_2          852950 non-null  object        
 21  crime_code_3          852950 non-null  object        
 22  crime_code_4          852950 non-null  object        
 23  location              852950 non-null  object        
 24  cross_street          852950 non-null  object        
 25  latitude              852950 non-null  float64       
 26  longitude             852950 non-null  float64       
 27  year                  852950 non-null  int32         
 28  month                 852950 non-null  int32         
 29  hour                  852950 non-null  int32         
dtypes: datetime64[ns](2), float64(3), int32(3), int64(5), object(17)
memory usage: 185.5+ MB
```

To streamline our analysis for predictive modeling, we'll categorize these cases into two distinct groups: solved and unsolved.

In [25]:

```
#Unsolved Cases: We will classify a case as unsolved if its status is marked as "Invest Cont" (Investigation Continuing)
#Solved Cases: Conversely, cases with any other status will be considered as solved. This includes statuses that imply t
```

In [26]:

```
#add case_solved column
df['case_solved']=df['status_description'].apply(lambda x: 'Not solved' if x=='Invest Cont' else 'Solved')
```

In [27]:

```
df
```

Out[27]:

	division_number	date_reported	date_occurred	area	area_name	reporting_district	part	crime_code	crime_description	modus_operandi	..
0	10304468	2020-01-08	2020-01-08 22:30:00	3	Southwest	377	2	624	BATTERY - SIMPLE ASSAULT	0444 0913	..
1	190101086	2020-01-02	2020-01-01 03:30:00	1	Central	163	2	624	BATTERY - SIMPLE ASSAULT	0416 1822 1414	..
2	200110444	2020-04-14	2020-02-13 12:00:00	1	Central	155	2	845	SEX OFFENDER REGISTRANT OUT OF COMPLIANCE	1501	..
3	191501505	2020-01-01	2020-01-01 17:30:00	15	N Hollywood	1543	2	745	VANDALISM - MISDEAMEANOR ($399 OR UNDER)	0329 1402	..
4	191921269	2020-01-01	2020-01-01 04:15:00	19	Mission	1998	2	740	VANDALISM - FELONY ($400 & OVER, ALL CHURCH VA...	0329	..
...	...	...	...	...	...	...	...	...	...	...	..
852945	231606525	2023-03-22	2023-03-22 10:00:00	16	Foothill	1602	1	230	ASSAULT WITH DEADLY WEAPON, AGGRAVATED ASSAULT	0416 0411 1822	..
852946	231210064	2023-04-12	2023-04-12 16:30:00	12	77th Street	1239	1	230	ASSAULT WITH DEADLY WEAPON, AGGRAVATED ASSAULT	0601 0445 0416 0359	..
852947	230115220	2023-07-02	2023-07-01 00:01:00	1	Central	154	1	352	PICKPOCKET	1822 0344	..
852948	230906458	2023-03-05	2023-03-05 09:00:00	9	Van Nuys	914	2	745	VANDALISM - MISDEAMEANOR ($399 OR UNDER)	0329 1822	..
852949	230319786	2023-11-10	2023-11-09 23:00:00	3	Southwest	395	1	331	THEFT FROM MOTOR VEHICLE - GRAND ($950.01 AND ...	1822 1606 0344 0385 1300	..

852950 rows × 31 columns

In [28]:

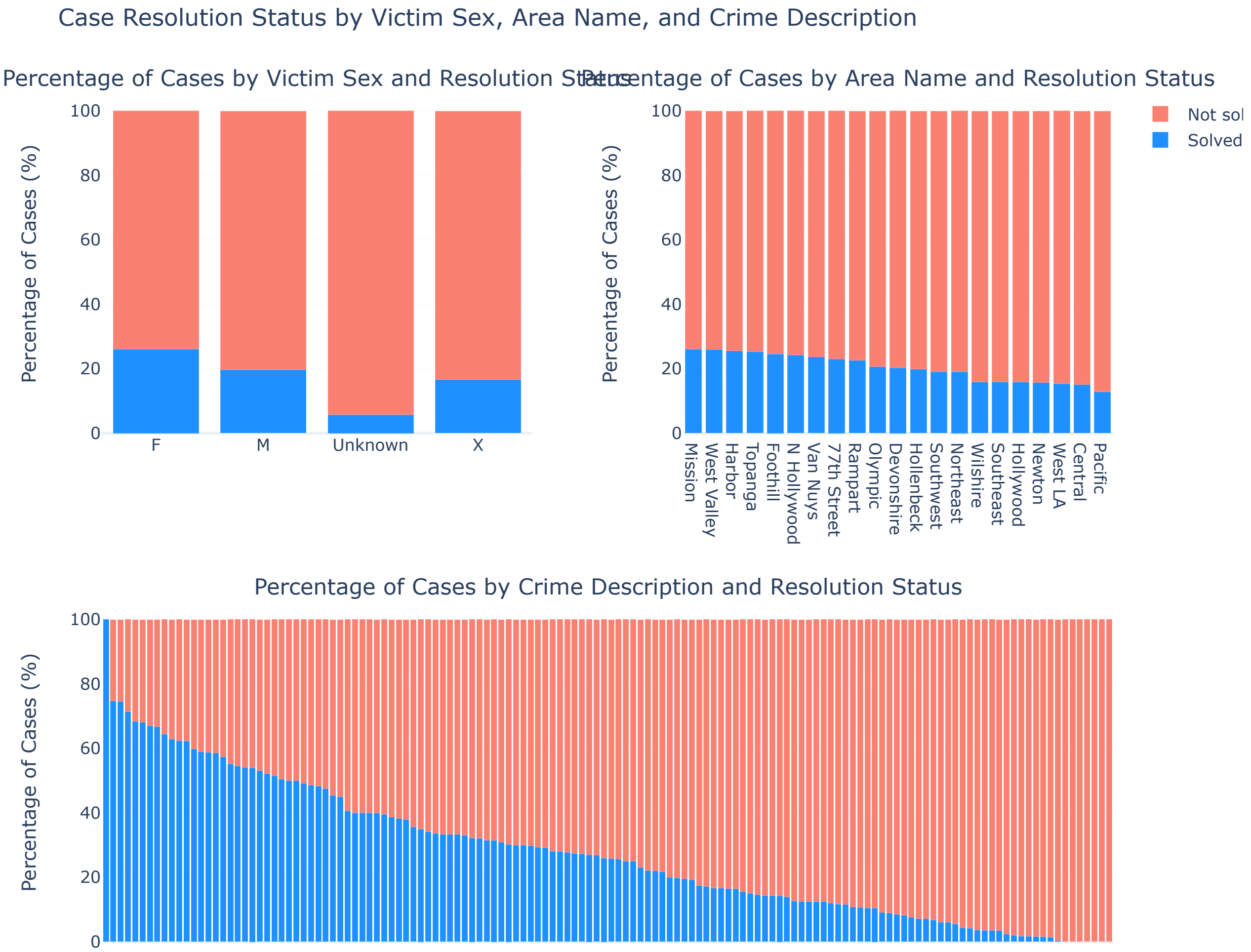
```
#df=pd.get_dummies(df,dtype=int)
#df1 = pd.get_dummies(df['crime_description'], sparse=True,dtype=int)
```

In [29]:

```
#df1
```

In [30]:

```
# Data preparation for victim_sex plot
sex_solved_counts = df.groupby(['victim_sex', 'case_solved']).size().unstack()
sex_solved_percent = sex_solved_counts.div(sex_solved_counts.sum(axis=1), axis=0) * 100



# Data preparation for crime_description plot
crime_solved_counts = df.groupby(['crime_description', 'case_solved']).size().unstack()
crime_solved_percent = crime_solved_counts.div(crime_solved_counts.sum(axis=1), axis=0) * 100
crime_solved_percent_sorted = crime_solved_percent.sort_values(by='Solved', ascending=False)


# Data preparation for area_name plot
area_solved_counts = df.groupby(['area_name', 'case_solved']).size().unstack()
area_solved_percent = area_solved_counts.div(area_solved_counts.sum(axis=1), axis=0) * 100
area_solved_percent_sorted = area_solved_percent.sort_values(by='Solved', ascending=False)
fig = make_subplots(
    rows=2, cols=2,
    subplot_titles=('Percentage of Cases by Victim Sex and Resolution Status',
                    'Percentage of Cases by Area Name and Resolution Status',
                    'Percentage of Cases by Crime Description and Resolution Status'),
    specs=[[{"type": "bar"}, {"type": "bar"}],
           [{"type": "bar", "colspan": 2}, None]],
    horizontal_spacing=0.15, vertical_spacing=0.2
)


# Plot for Victim Sex
fig.add_trace(
    go.Bar(x=sex_solved_percent.index, y=sex_solved_percent['Solved'], name='Solved', marker_color='dodgerblue'),
    row=1, col=1
)
fig.add_trace(
    go.Bar(x=sex_solved_percent.index, y=sex_solved_percent['Not solved'], name='Not solved', marker_color='salmon'),
    row=1, col=1
)


# Plot for Area Name
fig.add_trace(
    go.Bar(x=area_solved_percent_sorted.index, y=area_solved_percent_sorted['Solved'], name='Solved', marker_color='dodg
    row=1, col=2
)
fig.add_trace(
    go.Bar(x=area_solved_percent_sorted.index, y=area_solved_percent_sorted['Not solved'], name='Not solved', marker_col
    row=1, col=2
)


# Plot for Crime Description
fig.add_trace(
    go.Bar(x=crime_solved_percent_sorted.index, y=crime_solved_percent_sorted['Solved'], name='Solved', marker_color='do
    row=2, col=1
)
fig.add_trace(
    go.Bar(x=crime_solved_percent_sorted.index, y=crime_solved_percent_sorted['Not solved'], name='Not solved', marker_c
    row=2, col=1
)


# Update layout for the charts
fig.update_layout(
    height=800,
    barmode='stack',
    title_text="Case Resolution Status by Victim Sex, Area Name, and Crime Description",
    template='plotly_white'
)



# Update y-axis titles
fig.update_yaxes(title_text="Percentage of Cases (%)", row=1, col=1)
fig.update_yaxes(title_text="Percentage of Cases (%)", row=1, col=2)
fig.update_yaxes(title_text="Percentage of Cases (%)", row=2, col=1)

fig.update_xaxes(showticklabels=False, row=2, col=1)
fig.show()
```



In [ ]:

```
```

In [31]:

```
# Data preparation for area_name plot
#area_solved_counts = df.groupby(['area_name', 'case_solved']).size().unstack()
#area_solved_percent = area_solved_counts.div(area_solved_counts.sum(axis=1), axis=0) * 100
#area_solved_percent_sorted = area_solved_percent.sort_values(by='Solved', ascending=False)
```

In [32]:

```
#fig = make_subplots(
    #rows=2, cols=2,
    #subplot_titles=('Percentage of Cases by Victim Sex and Resolution Status',
                    #'Percentage of Cases by Area Name and Resolution Status',
                    #'Percentage of Cases by Crime Description and Resolution Status'),
    #specs=[[{"type": "bar"}, {"type": "bar"}],
          # [{"type": "bar", "colspan": 2}, None]],
    #horizontal_spacing=0.15, vertical_spacing=0.2
#)
```

In [33]:

```
# Plot for Victim Sex
#fig.add_trace(
    #go.Bar(x=sex_solved_percent.index, y=sex_solved_percent['Solved'], name='Solved', marker_color='dodgerblue'),
    #row=1, col=1
#)
#fig.add_trace(
    #go.Bar(x=sex_solved_percent.index, y=sex_solved_percent['Not solved'], name='Not solved', marker_color='salmon'),
    #row=1, col=1
#)
```

Features

In [34]:
```
#area,crime_code,victim_sex,victim_descent,weapon_code,hour,reported_delay,days_after_reported
```

In [35]:
```
# Calculate reported_delay and days_after_reported
from datetime import datetime
today = datetime.now()
df['reported_delay'] = (df['date_reported'] - df['date_occurred']).dt.days
df['reported_delay'] = df['reported_delay'].apply(lambda x: x if x >= 0 else 0)
df['days_after_reported'] = (today - df['date_reported']).dt.days
```

In [36]:
```
# Preparing the target variable and converting it to binary
from sklearn.preprocessing import LabelEncoder
target = 'case_solved'
le = LabelEncoder()
df[target] = le.fit_transform(df[target])

# Features for the model
features = ['area', 'crime_code', 'victim_sex', 'victim_descent', 'weapon_code', 'hour', 'reported_delay', 'days_after_r
```

In [37]:
```
# Explicitly convert all categorical features to strings
for feature in ['victim_sex', 'victim_descent', 'weapon_code']:
    df[feature] = df[feature].astype(str)
```

In [38]:
```
# Encoding categorical features
label_encoders = {}
for feature in ['victim_sex', 'victim_descent', 'weapon_code']:
    le = LabelEncoder()
    df[feature] = le.fit_transform(df[feature])
    label_encoders[feature] = le
```

In [39]:
```
# Display the first few rows of the data to verify the encoding
df[features].head()
```

Out[39]:

	area	crime_code	victim_sex	victim_descent	weapon_code	hour	reported_delay	days_after_reported
0	3	624	0	1	61	22	0	1450
1	1	624	1	6	62	3	0	1456
2	1	845	3	18	79	12	60	1353
3	15	745	0	17	79	17	0	1457
4	19	740	3	18	79	4	0	1457

In [40]:
```
# Preparing the data
X = df[features]  # Features
y = df[target]    # Target
```

In [41]:
```
# Splitting the dataset into training and testing sets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [42]:
```
# Standardizing the features (important for logistic regression)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [43]:

```
# Logistic Regression Model
log_reg = LogisticRegression()
log_reg.fit(X_train_scaled, y_train)
```

Out[43]:

```
▾ LogisticRegression
LogisticRegression()
```

In [44]:

```
# Random Forest Classifier Model
rf_clf = RandomForestClassifier()
rf_clf.fit(X_train, y_train)
```

Out[44]:

```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [45]:

```
# Making predictions and evaluating the models
log_reg_pred = log_reg.predict(X_test_scaled)
rf_clf_pred = rf_clf.predict(X_test)
```

In [46]:

```
# Making predictions and evaluating the models
log_reg_pred = log_reg.predict(X_test_scaled)
rf_clf_pred = rf_clf.predict(X_test)
```

In [48]:

```
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
print("Logistic Regression Accuracy:", accuracy_score(y_test, log_reg_pred))
print("Random Forest Classifier Accuracy:", accuracy_score(y_test, rf_clf_pred))
# You can also print out classification reports for more detailed performance analysis
print("\nLogistic Regression Classification Report:\n", classification_report(y_test, log_reg_pred))
print("\nRandom Forest Classifier Classification Report:\n", classification_report(y_test, rf_clf_pred))
```

```
Logistic Regression Accuracy: 0.7895460851554409
Random Forest Classifier Accuracy: 0.8292318815092717

Logistic Regression Classification Report:
               precision    recall  f1-score   support

           0       0.80      0.97      0.88    204840
           1       0.32      0.05      0.08     51045

    accuracy                           0.79    255885
   macro avg       0.56      0.51      0.48    255885
weighted avg       0.71      0.79      0.72    255885


Random Forest Classifier Classification Report:
               precision    recall  f1-score   support

           0       0.87      0.93      0.90    204840
           1       0.60      0.42      0.50     51045

    accuracy                           0.83    255885
   macro avg       0.73      0.68      0.70    255885
weighted avg       0.81      0.83      0.82    255885
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: