Report: ETL Traffic Data Analysis

This report presents an Exploratory Data Analysis of California traffic collision data using PySpark and AWS services. Data analytics techniques to clean, transform, and explore crash data are employed to draw meaningful insights to support traffic safety and urban planning. Beyond understanding how big data tools optimize performance on a single machine and across clusters, employing structured approach to analyzing crash trends, identifying high-risk locations, and evaluating contributing factors to traffic incidents.

Business Value:

Traffic collisions pose significant risks to public safety, requiring continuous monitoring and analysis to enhance road safety measures. Government agencies, city planners, and policymakers must leverage data-driven insights to improve infrastructure, optimize traffic management, and implement preventive measures.

In this assignment, California traffic collision data is analyzed to uncover patterns related to accident severity, location-based risks, and key contributing factors. With Apache Spark's ability to handle large datasets efficiently and AWS S3's scalable storage, transportation authorities can process vast amounts of crash data in real time, enabling faster and more informed decision-making.

As an analyst examining traffic safety trends, the task is to analyze historical crash data to derive actionable insights that can drive policy improvements and safety interventions. This analysis will help identify high-risk areas, categorize accidents by severity and contributing factors, and store the processed data in an AWS S3 bucket for scalable and long-term storage.

Data Preparation

Loading the dataset

Sample the data and combine the files

The dataset used in this analysis consists of California traffic collision data obtained from the Statewide Integrated Traffic Records System (SWITRS). It includes detailed records of traffic incidents across California, covering various attributes such as location, severity, involved parties, and contributing factors. The dataset has been preprocessed and transformed using PySpark to facilitate large-scale analysis. By leveraging Apache Spark, we ensure efficient data handling, enabling deeper insights into traffic patterns, accident trends, and potential safety improvements.

The dataset is a .sqlite file contains detailed information about traffic collisions across California and is structured into four primary tables:

- collisions table - contains information about the collision, where it happened, what vehicles were involved.
- parties table - contains information about the groups people involved in the collision including age, sex, and sobriety.
- victims table - contains information about the injuries of specific people involved in the collision.
- locations table - contains information about the geographical location and details of road intersections.

Since we will be working only with the collisions dataset and the victims dataset, the datasets are loaded into 2 separate dataframes to be cleaned and prepared for the analysis.

Data Cleaning

Handling Missing Values

Identify the number of missing values in each column

Missing data may be present in the dataset due to data entry errors, mismatched data-types, etc. Analysis shows that a few columns are missing over 50% of the values.

Collisions dataset:

```
Total Rows :  935791
jurisdiction : 1158
officer_id : 2264
reporting_district : 552653
population : 165
special_condition : 51730
chp_beat_class : 274
beat_number : 80780
primary_road : 10
secondary_road : 2
direction : 232663
intersection : 9484
weather_1 : 4626
state_highway_indicator : 318
caltrans_county : 682075
caltrans_district : 648728
state_route : 648728
postmile : 648728
location_type : 682075
side_of_highway : 682077
tow_away : 6153
killed_victims : 200
injured_victims : 230
primary_collision_factor : 5660
pcf_violation_category : 15424
pcf_violation : 64175
pcf_violation_subsection : 601427
type_of_collision : 7655
motor_vehicle_involved_with : 4887
pedestrian_action : 615
road_surface : 8141
road_condition_1 : 7903
lighting : 5301
control_device : 5791
statewide_vehicle_type_at_fault : 186119
chp_vehicle_type_at_fault : 251662
latitude : 669049
longitude : 669049
collision_time : 8084
```

Victims dataset:

```
Total Rows :  963933
victim_role : 1
victim_sex : 24041
victim_age : 32145
victim_seating_position : 1913
victim_safety_equipment_1 : 55215
victim_safety_equipment_2 : 310220
victim_ejected : 4272
```

Columns with more than 15% missing values are sparse columns, and can be dropped entirely. Corresponding rows for columns will less than 15% missing values can be handled individually.

Handle missing values

Once the columns with missing data are identified and sparse columns dropped, a decision is made for each column whether to fill in the missing data with most common value, or to filter out the rows with missing values. For the columns `*jurisdiction*`, `officer_id`, `*beat_number*`, `*primary_road*`, `*secondary_road*` and `*collision_time*` in collisions data and `*victim_role*` in victims dataset, the rows with missing values can be removed as there is no ideal value that can be substituted. The other missing values may be replaced with mean or mode of the respective columns.

```
col_filter_cols = ['jurisdiction', 'officer_id', 'beat_number', 'primary_road', 'secondary_road', 'collision_time']
collisions_df = collisions_df.dropna(subset=col_filter_cols)
collisions_df = collisions_df.fillna({"population": "unincorporated", \
                                      "special_condition": "0.0", \
                                      "chp_beat_class": "not chp", \
                                      "intersection": "0.0", \
                                      "weather_1": "clear", \
                                      "state_highway_indicator": "0.0", \
                                      "tow_away": "1.0", \
                                      "killed_victims": "0.0", \
                                      "injured_victims": "0.0", \
                                      "primary_collision_factor": "unknown", \
                                      "pcf_violation_category": "unknown", \
                                      "pcf_violation": "0.0", \
                                      "type_of_collision": "rear end", \
                                      "motor_vehicle_involved_with": "other motor vehicle", \
                                      "pedestrian_action": "no pedestrian involved", \
                                      "road_surface": "dry", \
                                      "road_condition_1": "normal", \
                                      "lighting": "daylight", \
                                      "control_device": "none"})
```

```
from pyspark.sql.functions import mean
vic_filter_cols = ['victim_role']

# victim_age_mean = victims_df.select(round(mean("victim_age"), 0)).collect()[0][0]
victims_df = victims_df.dropna(subset=vic_filter_cols)
victims_df = victims_df.fillna({"victim_sex": "female", \
                                "victim_age": str(victims_df.select(mean("victim_age")).collect()[0][0]), \
                                "victim_seating_position": "position unknown", \
                                "victim_safety_equipment_1": "unknown", \
                                "victim_ejected": "unknown"})
```

Convert datatypes

The datatypes of the columns are estimated by observing the data. Similar columns are classified into lists for each datatype, and converted.

Major observable datatypes – string, integer, long, date, timestamp

```
# Convert Data Types
from pyspark.sql.functions import col, concat_ws, year, month, dayofmonth, hour, dayofweek, quarter

# Convert to Integer datatypes
int_cols = ['jurisdiction', 'distance', 'intersection', 'state_highway_indicator', 'tow_away', 'killed_victims', 'injured_victims', 'party_count', 'pedestrian_collision', \
            'bicycle_collision', 'motorcycle_collision', 'truck_collision', 'not_private_property', 'chp_road_type', 'severe_injury_count', 'other_visible_injury_count', 'complaint_of_pain_injury_count', \
            'pedestrian_killed_count', 'pedestrian_injured_count', 'bicyclist_killed_count', 'bicyclist_injured_count', 'motorcyclist_killed_count', 'motorcyclist_injured_count' \
            'id', 'party_number', 'victim_age']

double_cols = ['case_id']

# Convert to Date datatypes
date_cols = ['collision_date', 'process_date']

# Convert to Timestamp datatypes
time_cols = ['collision_time']

def convert_datatypes(df):
    for column in df.columns:
        if column in int_cols:
            df = df.withColumn(column, col(column).cast("int"))
        if column in double_cols:
            df = df.withColumn(column, col(column).cast("double"))
        if column in time_cols:
            df = df.withColumn("collision_timestamp", concat_ws(" ", col("collision_date"), col(column)))
            df = df.withColumn("collision_timestamp", to_timestamp("collision_timestamp", "yyyy-MM-dd HH:mm:ss"))
            df = df.withColumn("collision_day", dayofmonth("collision_timestamp"))
            df = df.withColumn("collision_month", month("collision_timestamp"))
            df = df.withColumn("collision_quarter", quarter("collision_timestamp"))
            df = df.withColumn("collision_year", year("collision_timestamp"))
            df = df.withColumn("collision_hour", hour("collision_timestamp"))
            df = df.withColumn("collision_weekday", dayofweek("collision_timestamp"))
        if column in date_cols:
            df = df.withColumn(column, to_date(column, "yyyy-MM-dd"))
    return df
```

Drop duplicates

Duplicate records are dropped using `*dropDuplicates()*` method.

```
#Remove Duplicates
collisions_df_dedup = collisions_df_convert.dropDuplicates()
print("Collisions after deduplication", collisions_df_dedup.count())

victims_df_dedup = victims_df_convert.dropDuplicates()
print("Victims after deduplication", victims_df_dedup.count())
```

```
Collisions after deduplication 845609
Victims after deduplication 963932
```

Handling Outliers and Standardising Values

Detecting outliers

Performing an outlier analysis by describing the dataframe for each column shows if there are any potential outliers present. In a general sense, there are potential outliers in columns where the mean and median are further apart.

```
#Detect Outliers using IQR

# List of numerical columns to check for outliers
from pyspark.sql.functions import col, sum, mean, median, min, max, countDistinct

for col in int_cols:
    if col in collisions_df_dedup.columns:
        collisions_df_dedup.select(col).summary().show()
```

Based on the above analysis, *`distance`*, *`injured_victims`*, *`complaint_of_pain_injury_count`*, *`party_number`*, *`victim_age`* are identified as columns that contain outliers.

```
#Remove Outliers
def remove_outliers(df):
    for col in outlier_cols:
        if col in df.columns:
            Q1 = df.approxQuantile(col,[0.25],relativeError=0)
            Q3 = df.approxQuantile(col,[0.75],relativeError=0)

            IQR = Q3[0] - Q1[0]

            lower =  Q1[0] - 1.5*IQR
            upper =  Q3[0] + 1.5*IQR

            df = df.filter(df[col]>=lower)
            df = df.filter(df[col]<=upper)

    return df

collisions_df_clean = remove_outliers(collisions_df_dedup)
victims_df_clean = remove_outliers(victims_df_dedup)

print("Collisions after cleaning: ", collisions_df_clean.count())
print("Victims after cleaning: ", victims_df_clean.count())
```

```
Collisions after cleaning:  715855
Victims after cleaning:  944131
```

Exploratory Data Analysis

Final Data Cleaning

Classify variables into categorical and numerical

Categorical Columns – *'chp_shift', 'population', 'beat_type', 'chp_beat_type', 'chp_beat_class', 'weather_1', 'collision_severity', 'hit_and_run', 'type_of_collision', 'motor_vehicle_involved_with', 'pedestrian_action', 'road_surface', 'road_condition_1', 'lighting', 'control_device', 'statewide_vehicle_type_at_fault', 'victim_role', 'victim_sex', 'victim_degree_of_injury', 'victim_seating_position', 'victim_safety_equipment_1', 'victim_safety_equipment_2', 'victim_ejected'*

Numerical Columns – *'distance', 'killed_victims', 'injured_victims', 'party_count', 'severe_injury_count', 'other_visible_injury_count', 'complaint_of_pain_injury_count', 'pedestrian_killed_count', 'pedestrian_injured_count', 'bicyclist_killed_count', 'bicyclist_injured_count', 'motorcyclist_killed_count', 'motorcyclist_injured_count', 'id', 'party_number', 'victim_age'*

Removing dirty values

On initial analysis, there some observed categorical columns with dirty values – values that don't fit into the columns, or are improperly entered. These incorrect categorical values are to be removed.

motor_vehicle_involved_with	columns
other motor vehicle	613134
fixed object	145192
parked motor vehicle	69532
pedestrian	25756
non-collision	25032
bicycle	22465
other object	19348
motor vehicle on ...	6110
NULL	4887
animal	4031
train	273
8	18
2	10
0	1
3	1
9	1

victim_sex	columns
female	471349
male	468519
NULL	24041
4	13
3	5
X	4
1	2

It is observed that these invalid values are 1 or 2 characters in length.

```
for column in cat_cols:
    if column in collisions_df_clean.columns:
        collisions_df_clean = collisions_df_clean.filter(length(col(column)) > 2)
    if column in victims_df_clean.columns:
        victims_df_clean = victims_df_clean.filter(length(col(column)) > 2)
```

Reordering and Renaming columns

The final collisions dataset is prepared by removing irrelevant columns and fixing column names. The columns are also reorders for coherence.

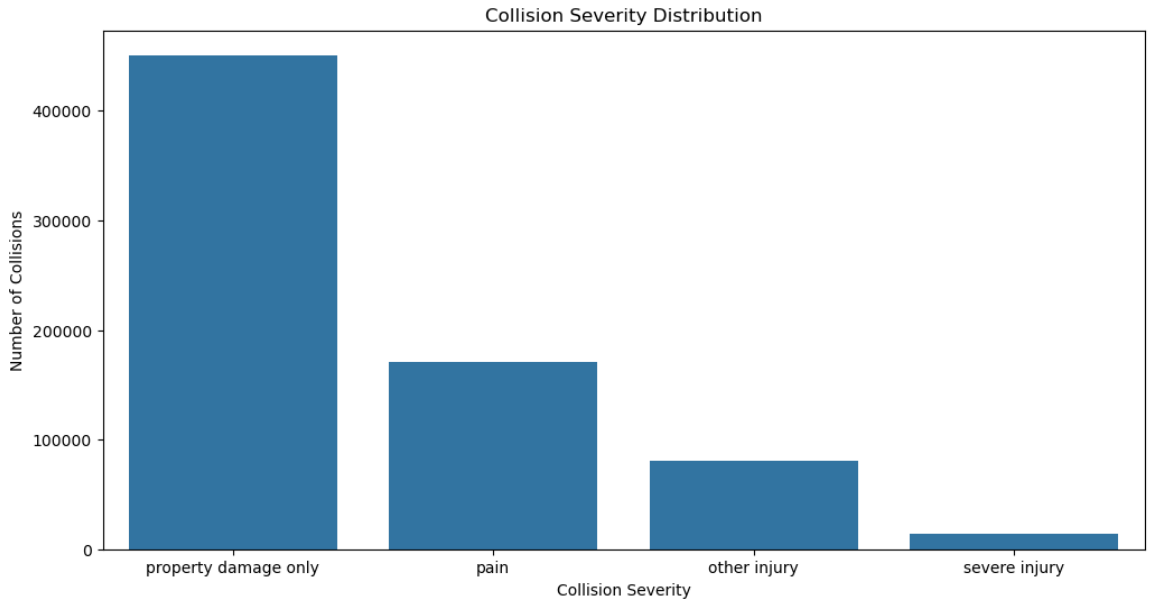
```python
# Reordering & Renaming Columns
final_collisions_df = spark.sql("""SELECT case_id,
                                population,
                                county_city_location as city,
                                county_location as county,
                                chp_beat_type as beat_type,
                                chp_beat_class as beat_class,
                                beat_number,
                                primary_road,
                                secondary_road,
                                distance,
                                intersection,
                                weather_1 as weather,
                                state_highway_indicator as state_highway,
                                tow_away,
                                collision_severity as severity,
                                primary_collision_factor,
                                pcf_violation_category,
                                hit_and_run,
                                type_of_collision,
                                motor_vehicle_involved_with,
                                road_surface,
                                road_condition_1 as road_condition,
                                chp_road_type as road_type,
                                lighting,
                                control_device,
                                pedestrian_action,
                                pedestrian_collision,
                                bicycle_collision,
                                motorcycle_collision,
                                truck_collision,
                                killed_victims as killed,
                                injured_victims as injured,
                                killed_victims+injured_victims as victims,
                                party_count,
                                severe_injury_count,
                                other_visible_injury_count as visible_injury_count,
                                complaint_of_pain_injury_count as pain_injury_count,
                                pedestrian_killed_count,
                                pedestrian_injured_count,
                                bicyclist_killed_count,
                                bicyclist_injured_count,
                                motorcyclist_killed_count,
                                motorcyclist_injured_count,
                                collision_date,
                                process_date,
                                collision_timestamp,
                                collision_day,
                                collision_month,
                                collision_quarter,
                                collision_year,
                                collision_hour,
                                collision_weekday
                                from collisions_df""")
```

Python

General EDA

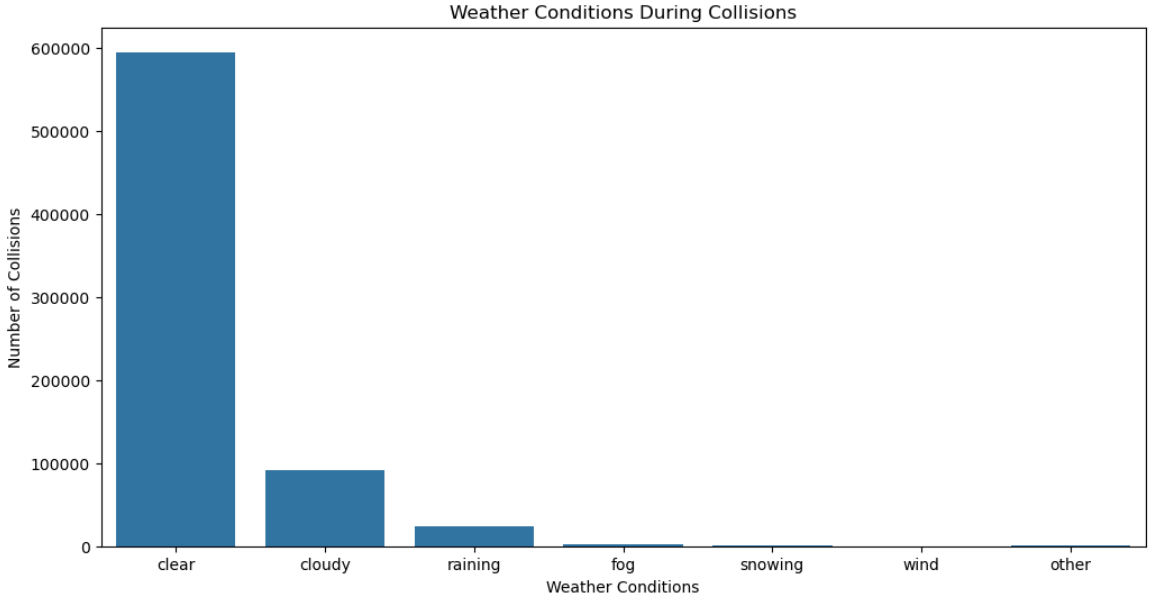
Distribution of Collision Severity

The severity of over 50% of the recorded collisions has resulted in only property damage.



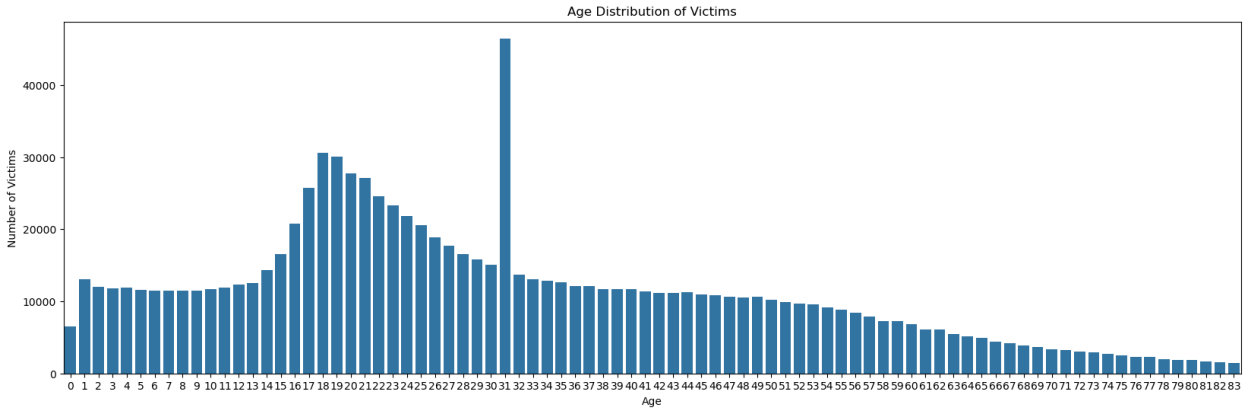
Weather Conditions during Collisions

A majority of the collisions have taken place in clear weather. There appears to be very little effect of the weather in the recorded crashes. It is possible that there could be a correlation between ill weather and the intensity of the crashes, but no effect in the number of collisions recorded.



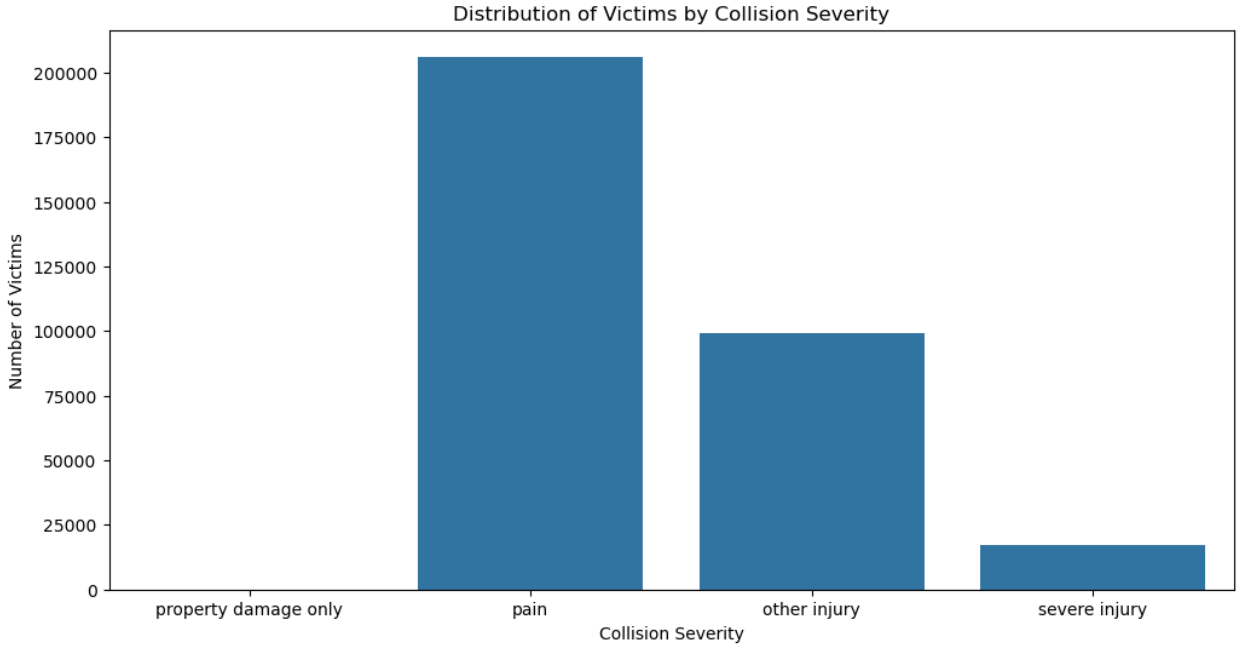
Distribution in Victim Ages

There is a curve observed in the ages of the victims. There are a higher number of victims in the age range of 16-25. However, there are a large number of victims aged 31.



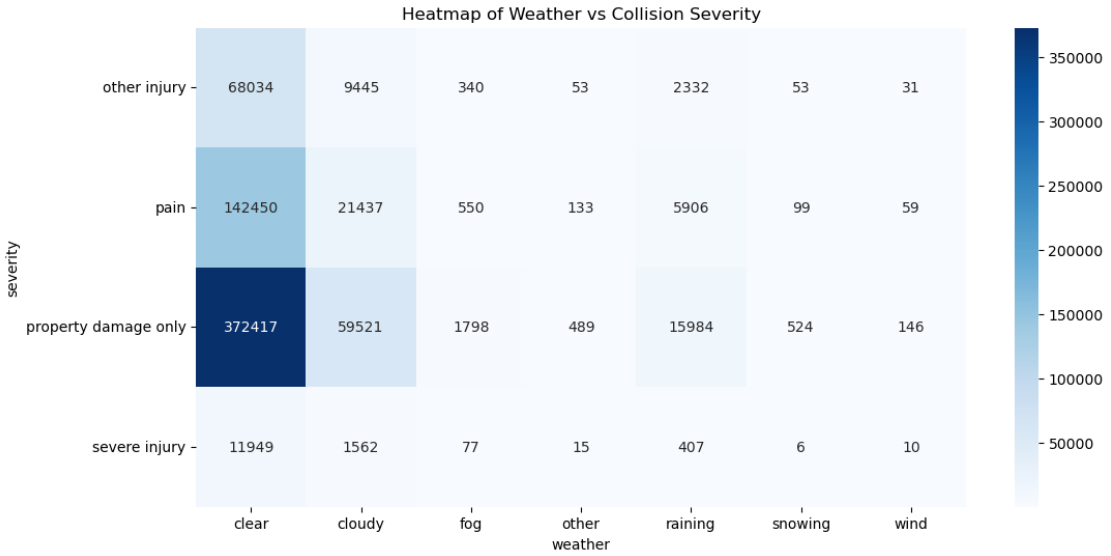
Collision Severity vs Number of Victims

Majority of the victims appear to have complaints of pain, with half that number with other minor injuries. Victims with severe injuries appear to be fewer in number, just under 20000.



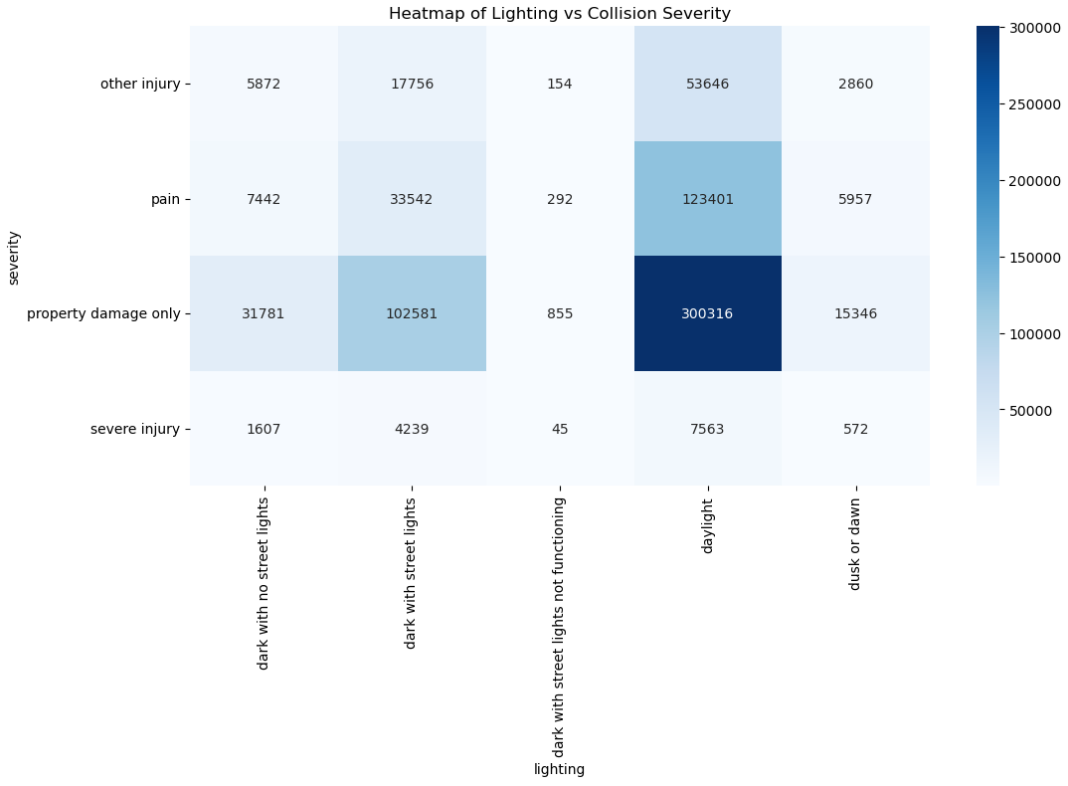
Correlation between Weather and Collision Severity

Clear days appear to have the highest number of collisions, followed by **cloudy** and **rainy days**. Property damage appears to be the major severity in collisions.



Lighting Conditions vs. Collision Severity

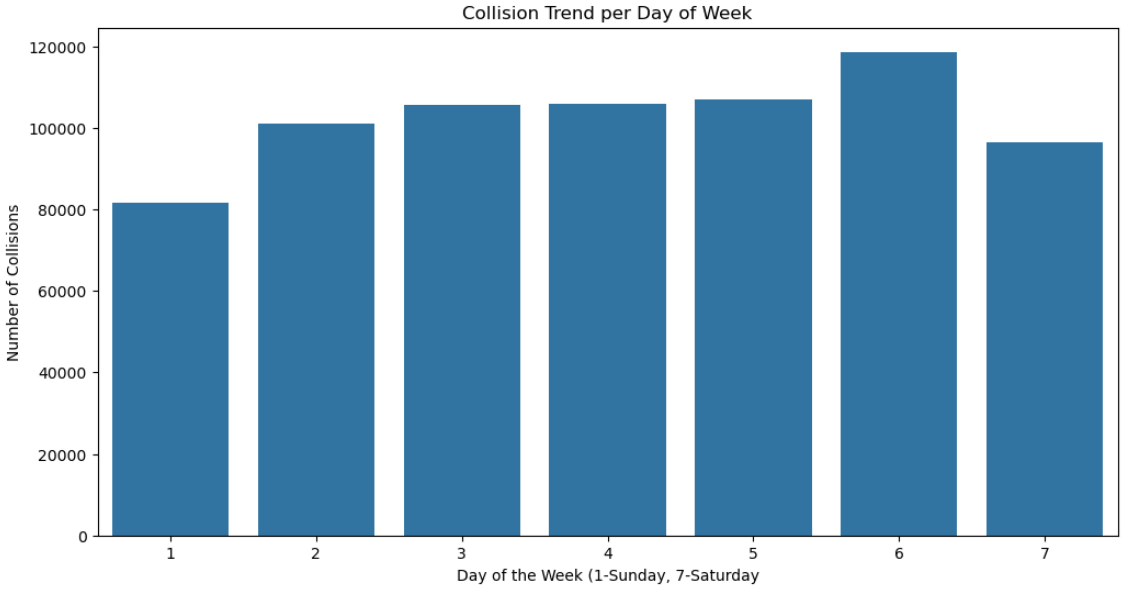
Analysis reveals a significant number of collisions in **daylight**, though a substantial portion also occurs in **dark with street lights**.



Time-series Analysis

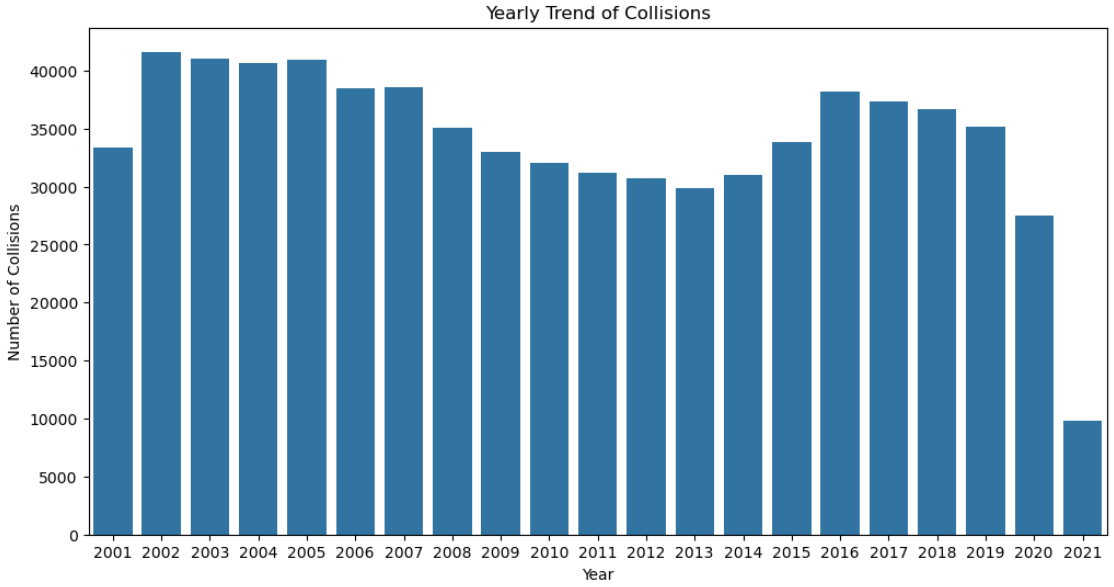
Weekday Collision Trends

Analysis shows that more collisions occur on Fridays, compared to other days of the week.



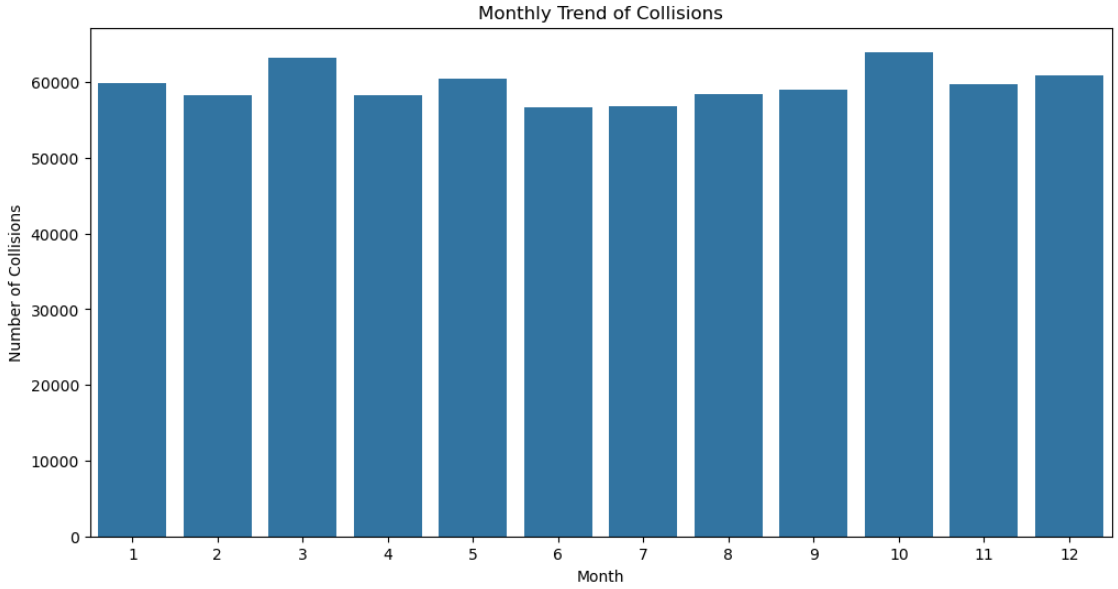
Yearly Collision Trend

The chart shows that number of collisions have dropped in the years between 2008 and 2014, before rising through the years 2015 to 2019 before falling sharply in 2020 and 2021, likely due to COVID lockdowns.



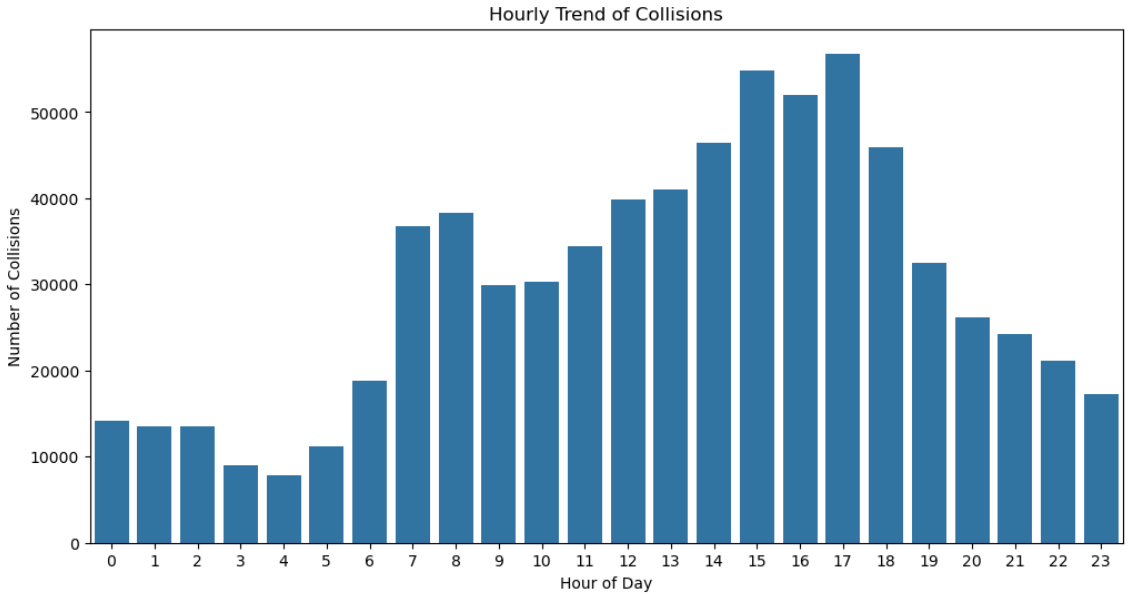
Monthly Collision Trends

Number of collisions are highest in the months of October and March, but not significantly.



Hourly Collision Trends

Late hours of the afternoon, between 3pm and 6pm shows a peak in collisions. This could be due to traffic in peak hours, resulting in sever rear-end collisions.



Extract-Transform-Load Querying

Load the processed dataset as CSV files in S3 bucket.

Detailed EDA: Insights and Strategies

Top 5 counties with most collisions

Analysis shows **Los Angeles** to be the county with the highest number of collisions – over 230000.

county	collision_count
los angeles	234157
orange	56144
san diego	42121
alameda	37387
san bernardino	37365

Month with highest number of collisions

October appears to be the month with highest number of collisions in California.

collision_month	collision_count
10	63990

Percentage of collisions that resulted in fatalities

fatality_percent
0.0

Most dangerous time of day for collisions

Highest number of collisions occur around the **5pm** mark, followed by **3pm** and **4pm**.

collision_hour	collision_count
17	56806
15	54770
16	51963
14	46466
18	45898

Top 5 road types with the highest collision frequency

Road type 0, followed by **road type 1**, contribute most towards traffic collisions.

road_type	collision_count
0	584715
1	115204
6	8843
5	5864
4	884

Top 3 lighting conditions that lead to the most collisions

Most collisions are recorded to occur in **daylight**, with close to 500000 collisions, followed by **dark with street lights**

lighting	collision_count
daylight	484926
dark with street ...	158118
dark with no stre...	46702

Conclusions

Final Insights and Recommendations

Infrastructure and Strategic Planning

The analysis identifies specific high-risk locations and temporal trends that warrant targeted infrastructure interventions:

High-Risk Counties: Strategic safety audits should be prioritized for the top 5 counties with the highest collision frequency.

Peak Collision Times: Infrastructure improvements, such as improved lighting and automated traffic monitoring, should be focused on the "most dangerous time of day" identified in the time-series analysis.

Road Design Optimization: By analyzing the correlation between road surface conditions and accident frequency, city planners can identify areas requiring resurfacing or specialized drainage to mitigate hazards during unfavourable weather.

Traffic Management and Operational Adjustments

Data-driven insights into environmental and operational factors suggest the following operational changes:

Lighting and Visibility: Given that a fair number of accidents occur in "dark with street lights" conditions, local authorities should evaluate the lighting conditions and placement of street lighting to improve nighttime visibility.

Signal Timing: Traffic signal cycles should be optimized based on the identified peak collision hours to manage traffic flow more effectively and reduce rear-end collisions, which were found to be a frequent collision type.

Variable Speed Limits: Implementing weather-responsive speed limits in areas identified as high-risk during "cloudy" or "raining" conditions can reduce the severity of accidents.

Pedestrian and Cyclist Safety: Policy changes should prioritize the creation of dedicated bike lanes and enhanced pedestrian crossings in zones where accidents involving these vulnerable groups are most prevalent.

Age-Specific Interventions: Based on the victim age distribution analysis, safety awareness campaigns or specialized driver training should be targeted toward the most vulnerable or high-risk age demographics.

Recommendations

High-Risk Zones: Utilizing collision density maps, law enforcement can establish "proactive intervention zones" for increased patrolling or the placement of safety signage.

Predictive Modeling: The project provides a foundation for developing predictive models that can anticipate collision hotspots based on historical data, enabling authorities to implement preventative measures before incidents occur.