Analyzing California Traffic Collision Data to Identify Factors that contributes to Collisions

Prediction Task

Dataset Collection

The dataset for this project if from the California Traffic Collision Data set, which contains information about every traffic collision from 2001 to 2020 in the state of California. The dataset is publicly available from The State of California, which maintains a database of traffic collisions called the Statewide Integrated Traffic Records System (SWITRS). The data is extensive, containing 9.46 million rows at 6.21 GB in total.

Source:

Dataset on Zenodo

Dataset Description

The database switrs.sqlite contains four tables as shown;

SWITRS.SQLITE

Case_ids(9.46M)	Collisions(9.46M)	Parties(18.7M)	Victims(9.6M)
2 columns	75 columns	31 Columns	11 Columns

Steps to Follow for the prediction task in this project

1. **Data Collection**: Downloading the California Traffic Collision Data set from the provided reference.
2. **Data Preprocessing**: Cleaning the dataset, handle missing values, and perform any necessary data transformations.
3. **Exploratory Data Analysis**: Exploring the dataset to gain insights into the distribution of variables and identify any patterns or correlations.
4. **Feature Selection**: Selecting relevant features that are likely to contribute to collisions based on domain knowledge and exploratory analysis.
5. **Data Split**: Spliting the dataset into training and testing sets for model development and evaluation.
6. **Model Development**: Choosing a suitable machine learning model for predicting collisions based on the selected features.
7. **Model Training**: Training the chosen model on the training data.
8. **Model Evaluation**: Evaluating the performance of the trained model using appropriate evaluation metrics.
9. **Results and Conclusion**: Summarizing the findings of the analysis and discuss the most important factors contributing to collisions in California.

Resources

Here are some resources that might be helpful for this project:

- Apache Spark-Python Documentation

Spark Context and its dependencies

In [19]:
```
from pyspark.sql import SparkSession
from pyspark import SparkConf
```

In [20]:
```
sparkConf = SparkConf().setAppName("Model_building")\
    .set('spark.executor.memory','4g')\
    .set("spark.driver.memory", "3g")

spark = SparkSession.builder.config(conf=sparkConf).getOrCreate()
spark.sparkContext.setLogLevel("ERROR")
```

In [3]:
```
spark
```

Out[3]: **SparkSession - in-memory**

SparkContext

Spark UI

Version	`v3.2.1`
Master	`local[*]`
AppName	`Model_building`

1.1 Loading the dataset from the three tables collsions, parties, victims

In [4]:
```
##reading the dataset
training = spark.read.csv('collisions2.csv',header=True,inferSchema=True)
```

In [5]:
```
training2 = spark.read.csv('parties2.csv',header=True,inferSchema=True)
```

In [6]:
```
training3 = spark.read.csv('victims2.csv',header=True,inferSchema=True)
```

Type casting the type of case_id of each table to a common data type called "Integer"

In [7]:
```
# cast function is used to type cast

from pyspark.sql.functions import col

training = training.withColumn("case_id", col("case_id").cast("integer"))


training2 = training2.withColumn("case_id", col("case_id").cast("integer"))

training3 = training3.withColumn("case_id", col("case_id").cast("integer"))
```

In [10]:
```
training3.printSchema()
```

```
root
 |-- id: integer (nullable = true)
 |-- case_id: integer (nullable = true)
 |-- party_number: integer (nullable = true)
 |-- victim_role: integer (nullable = true)
 |-- victim_sex: string (nullable = true)
 |-- victim_age: integer (nullable = true)
 |-- victim_degree_of_injury: string (nullable = true)
 |-- victim_seating_position: string (nullable = true)
 |-- victim_safety_equipment_1: string (nullable = true)
 |-- victim_safety_equipment_2: string (nullable = true)
 |-- victim_ejected: string (nullable = true)
```

Joining all three tables and the join condition is set to filter the records based on the severity_levels and at_fault = Yes or 1 class of table 2

In [11]:
```
# severity levels are the entries within the collision_severity columns.
severity_levels = ["property damage only", "pain", "other injury", "severe injury", "fatal"]

dfs = []
for i in severity_levels:

     # Apply join and filter operations and select specific columns
    df = training.join(training2, training2["case_id"] == training["case_id"]) \
        .join(training3, training3["case_id"] == training["case_id"]) \
        .filter((training["collision_severity"] == i) & (training2["at_fault"] == 1)) \
        .select(
            training["case_id"],
            training["collision_severity"],
            training["weather_1"],
            training["state_highway_indicator"],
            training["party_count"],
            training["type_of_collision"],
            training["lighting"],
            training["tow_away"],
            training2["party_age"],
            training2["party_sobriety"],
            training["road_surface"],
            training["hit_and_run"],
            training3["victim_sex"],
            training3["victim_age"],
            training3["victim_seating_position"]
        )
```

```
    dfs.append(df)

# Combine all DataFrames
df = dfs[0]
for i in range(1, len(dfs)):
    df = df.union(dfs[i])
```

Filtering, transforming and replacing the records

In [12]:
```
from pyspark.sql.functions import col,when

# Replacing  "dark with street lights not functioning"
# with "dark with no street lights" in 'lighting' column


df = df.withColumn('lighting', \
                   when(col('lighting') == 'dark with street lights not functioning', 'dark with no street ligh
                   .otherwise(col('lighting')))
```

In [13]:
```
#filtering rows with the records having 'G' and 'holes'
# since the entries for 'G' and 'holes' is neglibile in comparision

df = df.filter(
    ~(col('lighting').isin('G','holes')))
```

In [14]:
```
# Replace ['other injury', 'pain', 'severe injury'] in 'collision_severity' column with
# 'injury'. Also filtering out the negligible entries for "0" and "N"


df = df.withColumn('collision_severity', \
                   when(col('collision_severity').isin(['other injury', 'pain', 'severe injury']), 'injury') \
                   .otherwise(col('collision_severity')))
df =df.filter(
    ~(col('collision_severity').isin('0','N')))
```

In [15]:
```
from pyspark.sql.functions import col

# Filtering out the negligible entries


df = df.filter(~col('victim_seating_position').isin('B','C','A')) \
       .filter(~col('victim_sex').isin(['4', 'X', '2', '1', '3', 'W', 'N','5','U','G','H','B'])) \
       .filter(~col('type_of_collision').isin('N','C','I','D','O')) \
       .filter(~col('weather_1').isin('0')) \
       .filter(~col('hit_and_run').isin('N')) \
       .filter(~col('tow_away').isin('northbound','Y')) \
       .filter(col('party_count') <= 6.0)
```

In [16]:
```
# Filtering negligible entries.
df = df.filter(
    ~(col('road_surface').isin('H', 'J', 'I', 'no pedestrian involved', 'crossing in intersection crosswalk'))
)
```

Removing all the missing values

In [17]:
```
#dropping missing values from the dataframe

df = df.dropna()
```

In [18]:
```
# Finding  count for empty, None, Null, Nan with string literals.

from pyspark.sql.functions import col,isnan,when,count
df2 = df.select([count(when(col(c).contains('None') | \
                            col(c).contains('NULL') | \
                            (col(c) == '' ) | \
                            col(c).isNull() | \
                            isnan(c), c
                           )).alias(c)
                    for c in df.columns])
df2.show()
```

```
[Stage 20:====================================================>  (120 + 5) / 125]
```

```
+-------+------------------+---------+----------------------+-----------+-----------------+--------+--------+-
--------+--------------+------------+-----------+----------+----------+----------------------+
|case_id|collision_severity|weather_1|state_highway_indicator|party_count|type_of_collision|lighting|tow_away|p
arty_age|party_sobriety|road_surface|hit_and_run|victim_sex|victim_age|victim_seating_position|
+-------+------------------+---------+----------------------+-----------+-----------------+--------+--------+-
--------+--------------+------------+-----------+----------+----------+----------------------+
|      0|                 0|        0|                     0|          0|                0|       0|       0|
0|             0|           0|          0|         0|         0|                     0|

+-------+------------------+---------+----------------------+-----------+-----------------+--------+--------+-
--------+--------------+------------+-----------+----------+----------+----------------------+
```

Calculating the total count after removal of missing values and count for each features in the df dataframe

In [21]:
```python
total_rows = df.count()
print("Total number of rows: ", total_rows)
```

```
[Stage 44:====================================================>  (119 + 6) / 125]
Total number of rows:  7665399
```

In [22]:
```python
df.groupBy('collision_severity').count().show()
```

```
[Stage 68:====================================================>  (120 + 5) / 125]
+--------------------+-------+
|  collision_severity|  count|
+--------------------+-------+
|property damage only|2027541|
|              injury|5517776|
|               fatal| 120082|
+--------------------+-------+
```

Sampling the given dataframe with fraction for each target label set to 1% of the total records

This sample is used for building the models. The sampling is done by sampleBy which is stratified sampling . Stratified sampling ensures that the resulting sample has a similar class distribution as the original dataset.

In [23]:
```python
from pyspark.sql.functions import col

# Defining the fraction for sampling each class
fractions = {
    "property damage only": 0.01,
    "injury": 0.01,
    "fatal": 0.01
}

# Performing stratified sampling
sample_df = df.sampleBy("collision_severity", fractions, seed=42)

# Verifying the class distribution in the sampled dataset
sample_df.groupBy("collision_severity").count().show()
```

```
[Stage 92:===================================================>   (118 + 7) / 125]
+--------------------+-----+
|  collision_severity|count|
+--------------------+-----+
|property damage only|20501|
|              injury|55138|
|               fatal| 1218|
+--------------------+-----+
```

this sample will be used to create a new stratified sample for testing the model performance. The stratified sample is termed as sample_testing

In [25]:
```python
from pyspark.sql.functions import col

fractions = {
    "property damage only": 0.01,
    "injury": 0.01,
    "fatal": 0.01
}

sample_testing = test_sample_df.sampleBy("collision_severity", fractions, seed=42)
sample_testing.groupBy("collision_severity").count().show()
```

```
[Stage 215:=================================================>      (8 + 1) / 9]
```

```
+--------------------+-----+
|  collision_severity|count|
+--------------------+-----+
|property damage only|19842|
|              injury|54734|
|               fatal| 1218|
+--------------------+-----+
```

In [290…
```
#Save the stratified sample as a CSV file
sample_testing.coalesce(1).write.csv("stratified_sample2.csv", header=True)
```

In [141…
```
sample_df = sample_df.drop("case_id")
```

In [142…
```
sample_df.printSchema()
```

```
root
 |-- collision_severity: string (nullable = true)
 |-- weather_1: string (nullable = true)
 |-- state_highway_indicator: integer (nullable = true)
 |-- party_count: integer (nullable = true)
 |-- type_of_collision: string (nullable = true)
 |-- lighting: string (nullable = true)
 |-- tow_away: string (nullable = true)
 |-- party_age: integer (nullable = true)
 |-- party_sobriety: string (nullable = true)
 |-- road_surface: string (nullable = true)
 |-- hit_and_run: string (nullable = true)
 |-- victim_sex: string (nullable = true)
 |-- victim_age: integer (nullable = true)
 |-- victim_seating_position: string (nullable = true)
```

Retrieving the counts for each features in sample_df after stratified sampling

In [29]:
```
sample_df.groupBy("type_of_collision").count().show()
```

```
[Stage 323:=====================================================>  (120 + 5) / 125]
+-----------------+-----+
|type_of_collision|count|
+-----------------+-----+
|        sideswipe| 8827|
|         rear end|30084|
|          head-on| 4277|
|            other| 1183|
|       hit object| 7863|
|        broadside|20173|
|       overturned| 2324|
|       pedestrian| 2126|
+-----------------+-----+
```

In [30]:
```
sample_df.groupBy("weather_1").count().show()
```

```
[Stage 347:=====================================================>  (119 + 6) / 125]
+---------+-----+
|weather_1|count|
+---------+-----+
|      fog|  284|
|  raining| 2388|
|    other|   45|
|    clear|63863|
|   cloudy|10117|
|  snowing|  133|
|     wind|   27|
+---------+-----+
```

In [31]:
```
sample_df.groupBy("hit_and_run").count().show()
```

```
[Stage 371:=====================================================>  (120 + 5) / 125]
+---------------+-----+
|    hit_and_run|count|
+---------------+-----+
|not hit and run|73062|
|    misdemeanor| 2314|
|         felony| 1481|
+---------------+-----+
```

In [32]:
```
sample_df.groupBy("lighting").count().show()
```

```
In [32]: sample_df.groupBy("lighting").count().show()
```

```
[Stage 395:====================================================> (122 + 3) / 125]
+--------------------+-----+
|            lighting|count|
+--------------------+-----+
|dark with no stre...| 5968|
|dark with street ...|14074|
|            daylight|54259|
|        dusk or dawn| 2556|
+--------------------+-----+
```

```
In [33]: sample_df.groupBy("tow_away").count().show()
```

```
[Stage 419:====================================================> (121 + 4) / 125]
+--------+-----+
|tow_away|count|
+--------+-----+
|       0|27371|
|       1|49486|
+--------+-----+
```

```
In [34]: sample_df.groupBy("road_surface").count().show()
```

```
[Stage 443:===================================================>  (120 + 5) / 125]
+------------+-----+
|road_surface|count|
+------------+-----+
|         wet| 5845|
|       snowy|  412|
|         dry|70545|
|    slippery|   55|
+------------+-----+
```

```
In [35]: sample_df.groupBy("victim_sex").count().show()
```

```
[Stage 467:=====================================================>(124 + 1) / 125]
+----------+-----+
|victim_sex|count|
+----------+-----+
|    female|38786|
|      male|38071|
+----------+-----+
```

```
In [36]: sample_df.groupBy("victim_seating_position").count().show()
```

```
[Stage 491:===================================================>  (120 + 5) / 125]
+-----------------------+-----+
|victim_seating_position|count|
+-----------------------+-----+
|                      7|  825|
|                      3|23089|
|                      8|  616|
|                      0| 1170|
|                      5| 2808|
|                      6| 7982|
|                      9| 3285|
|                      1|29154|
|                      4| 6680|
|                      2| 1248|
+-----------------------+-----+
```

```
In [37]: sample_df.groupBy("state_highway_indicator").count().show()
```

```
[Stage 515:====================================================> (122 + 3) / 125]
+-----------------------+-----+
|state_highway_indicator|count|
+-----------------------+-----+
|                      1|31744|
|                      0|45113|
+-----------------------+-----+
```

```
In [38]: sample_df.groupBy("party_count").count().show()
```

```
[Stage 539:===================================================>  (120 + 5) / 125]
```

```
+-----------+-----+
|party_count|count|
+-----------+-----+
|          1| 9402|
|          6|  210|
|          3|12230|
|          5|  795|
|          4| 3261|
|          2|50959|
+-----------+-----+
```

In [39]:
```
sample_df.groupBy("party_sobriety").count().show()
```

```
[Stage 563:====================================================>  (120 + 5) / 125]
+--------------+-----+
|party_sobriety|count|
+--------------+-----+
|             B| 6058|
|             D|  446|
|             C|  838|
|             A|66956|
|             G| 2057|
|             H|  502|
+--------------+-----+
```

In [26]:
```
from pyspark.mllib.stat import Statistics
sample_df.stat.crosstab("collision_severity", "type_of_collision").show()
```

```
[Stage 246:=====================================================>(124 + 1) / 125]
+-------------------------------------+---------+-------+----------+-----+----------+----------+--------+-------
--+
|collision_severity_type_of_collision|broadside|head-on|hit object|other|overturned|pedestrian|rear end|sideswi
pe|
+-------------------------------------+---------+-------+----------+-----+----------+----------+--------+-------
--+
|                 property damage only|     3469|    465|      2077|  284|       215|        10|    9702|     42
79|
|                               injury|    16444|   3614|      5504|  873|      1995|      1978|   20249|     44
81|
|                                fatal|      260|    198|       282|   26|       114|       138|     133|       
67|
+-------------------------------------+---------+-------+----------+-----+----------+----------+--------+-------
--+
```

In [27]:
```
sample_df.stat.crosstab("collision_severity", "weather_1").show()
```

```
[Stage 270:====================================================>  (120 + 5) / 125]
+----------------------------+-----+------+---+-----+-------+-------+----+
|collision_severity_weather_1|clear|cloudy|fog|other|raining|snowing|wind|
+----------------------------+-----+------+---+-----+-------+-------+----+
|        property damage only|16863|  2889| 65|    5|    614|     57|   8|
|                      injury|45997|  7063|209|   37|   1738|     76|  18|
|                       fatal| 1003|   165| 10|    3|     36|      0|   1|
+----------------------------+-----+------+---+-----+-------+-------+----+
```

In [28]:
```
sample_df.stat.crosstab("collision_severity", "state_highway_indicator").show()
```

```
+-----------------------------------------+-----+-----+
|collision_severity_state_highway_indicator|    0|    1|
+-----------------------------------------+-----+-----+
|                     property damage only| 9515|10986|
|                                   injury|34998|20140|
|                                    fatal|  600|  618|
+-----------------------------------------+-----+-----+
```

In [29]:
```
sample_df.stat.crosstab("collision_severity", "lighting").show()
```

```
+---------------------------+--------------------------+----------------------+--------+-----------+
|collision_severity_lighting|dark with no street lights|dark with street lights|daylight|dusk or dawn|
+---------------------------+--------------------------+----------------------+--------+-----------+
|       property damage only|                      1650|                   3396|   14857|        598|
|                     injury|                      4001|                  10387|   38843|       1907|
|                      fatal|                       317|                    291|     559|         51|
+---------------------------+--------------------------+----------------------+--------+-----------+
```

In [31]:
```
sample_df.stat.crosstab("collision_severity", "tow_away").show()
```

```
[Stage 342:====================================================>(123 + 2) / 125]
+-------------------------+-----+-----+
|collision_severity_tow_away|    0|    1|
+-------------------------+-----+-----+
|       property damage only|11717| 8784|
|                     injury|15568|39570|
|                      fatal|   86| 1132|
+-------------------------+-----+-----+
```

In [32]:
```
sample_df.stat.crosstab("collision_severity", "hit_and_run").show()
```

```
[Stage 366:==================================================>  (120 + 5) / 125]
+----------------------------+------+----------+--------------+
|collision_severity_hit_and_run|felony|misdemeanor|not hit and run|
+----------------------------+------+----------+--------------+
|          property damage only|     7|       1501|          18993|
|                        injury|  1427|        811|          52900|
|                         fatal|    47|          2|           1169|
+----------------------------+------+----------+--------------+
```

In [33]:
```
sample_df.stat.crosstab("collision_severity", "victim_sex").show()
```

```
[Stage 390:==================================================>  (120 + 5) / 125]
+---------------------------+------+-----+
|collision_severity_victim_sex|female| male|
+---------------------------+------+-----+
|         property damage only| 10383|10118|
|                       injury| 27967|27171|
|                        fatal|   436|  782|
+---------------------------+------+-----+
```

In [34]:
```
sample_df.stat.crosstab("collision_severity", "party_count").show()
```

```
[Stage 414:==================================================>  (120 + 5) / 125]
+----------------------------+----+-----+----+----+---+---+
|collision_severity_party_count|   1|    2|   3|   4|  5|  6|
+----------------------------+----+-----+----+----+---+---+
|          property damage only|2045|14848|2845| 606|123| 34|
|                        injury|7000|35505|9233|2579|652|169|
|                         fatal| 357|  606| 152|  76| 20|  7|
+----------------------------+----+-----+----+----+---+---+
```

In [35]:
```
sample_df.stat.crosstab("collision_severity", "victim_seating_position").show()
```

```
[Stage 438:==================================================>  (119 + 6) / 125]
+----------------------------------------+---+-----+---+-----+----+----+----+---+---+----+
|collision_severity_victim_seating_position|  0|    1|  2|    3|   4|   5|   6|  7|  8|   9|
+----------------------------------------+---+-----+---+-----+----+----+----+---+---+----+
|                      property damage only|437|  137|426|10440|3044|1243|3600|378|303| 493|
|                                    injury|700|28404|802|12417|3567|1538|4300|435|306|2669|
|                                     fatal| 33|  613| 20|  232|  69|  27|  82| 12|  7| 123|
+----------------------------------------+---+-----+---+-----+----+----+----+---+---+----+
```

In [36]:
```
sample_df.stat.crosstab("collision_severity", "party_sobriety").show()
```

```
+-------------------------------+-----+----+---+---+----+---+
|collision_severity_party_sobriety|    A|   B|  C|  D|   G|  H|
+-------------------------------+-----+----+---+---+----+---+
|             property damage only|18221|1369|215| 86| 542| 68|
|                           injury|48110|4350|596|318|1331|433|
|                            fatal|  625| 339| 27| 42| 184|  1|
+-------------------------------+-----+----+---+---+----+---+
```

In [37]:
```
sample_df.stat.crosstab("collision_severity", "road_surface").show()
```

```
[Stage 486:==================================================>  (120 + 5) / 125]
```

```
+-----------------------------+-----+--------+-----+----+
|collision_severity_road_surface|  dry|slippery|snowy| wet|
+-----------------------------+-----+--------+-----+----+
|           property damage only|18727|      13|  171|1590|
|                         injury|50689|      42|  237|4170|
|                          fatal| 1129|       0|    4|  85|
+-----------------------------+-----+--------+-----+----+
```

plotting a graph to visualize the frequency of collision severity categories

In [ ]:
```
import plotly.graph_objs as go

# Calculating the frequency of collision severity categories
cs_freqs = sample_df.groupBy('collision_severity').count().orderBy('collision_severity')

# Converting the frequency DataFrame to Pandas for plotting
cs_freqs_pd = cs_freqs.toPandas()

# Creating a Plotly bar plot
fig = go.Figure(data=[go.Bar(x=cs_freqs_pd['collision_severity'], y=cs_freqs_pd['count'])])
fig.update_layout(title='Frequency of Collision Severity Categories', xaxis_title='Collision Severity', yaxis_t

# Shows the plot
fig.show()
# Save the plot as a PNG image
fig.write_image('bar_plot.png')
```

Data Visualization

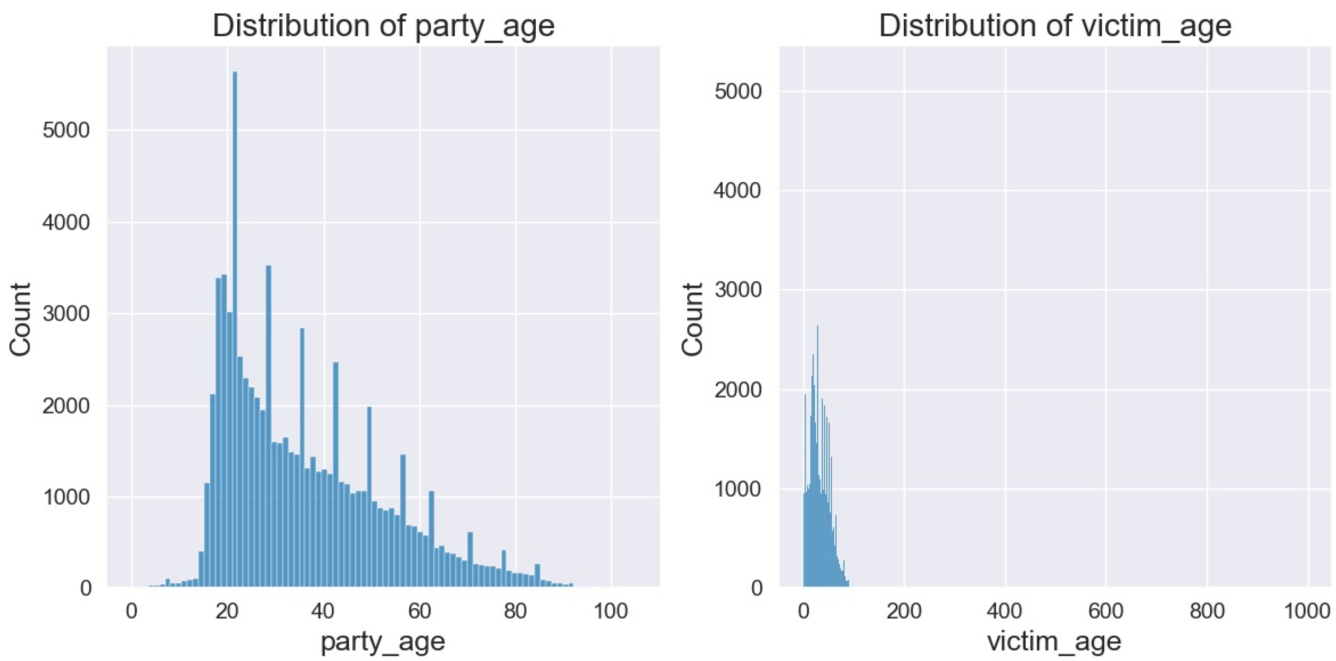
In [53]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [54]:
```
sns.set_style("darkgrid")
sns.set_context("notebook", rc={"lines.linewidth": 2,
                                "xtick.labelsize": 14,
                                "ytick.labelsize": 14,
                                "axes.labelsize": 18,
                                "axes.titlesize": 20})
```

In [55]:
```
numerical_df = sample_df.select(NUMERICAL_FEATURES)
```

In [56]:
```
numerical_pd = numerical_df.toPandas()
```

In [58]:
```
# Creating subplots for numerical variables
fig, axes = plt.subplots(nrows=1, ncols=len(NUMERICAL_FEATURES), figsize=(12, 6))

# Iterating over each numerical variable and plot the distribution
for i, col in enumerate(NUMERICAL_FEATURES):
    sns.histplot(data=numerical_pd, x=col, ax=axes[i])
    axes[i].set_title(f'Distribution of {col}')
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Count')

# Adjusting the spacing between subplots
plt.tight_layout()

# Shows the plots
plt.show()
```



In [218…

```
#assigning labels for the plot.

cs_labels = sample_df.select('collision_severity').distinct().toPandas()['collision_severity'].tolist()
print(cs_labels)

cs_counts = sample_df.groupBy('collision_severity').count().orderBy('count', ascending=False).select('count').t
cs_counts[0], cs_counts[1] = cs_counts[1], cs_counts[0]
```

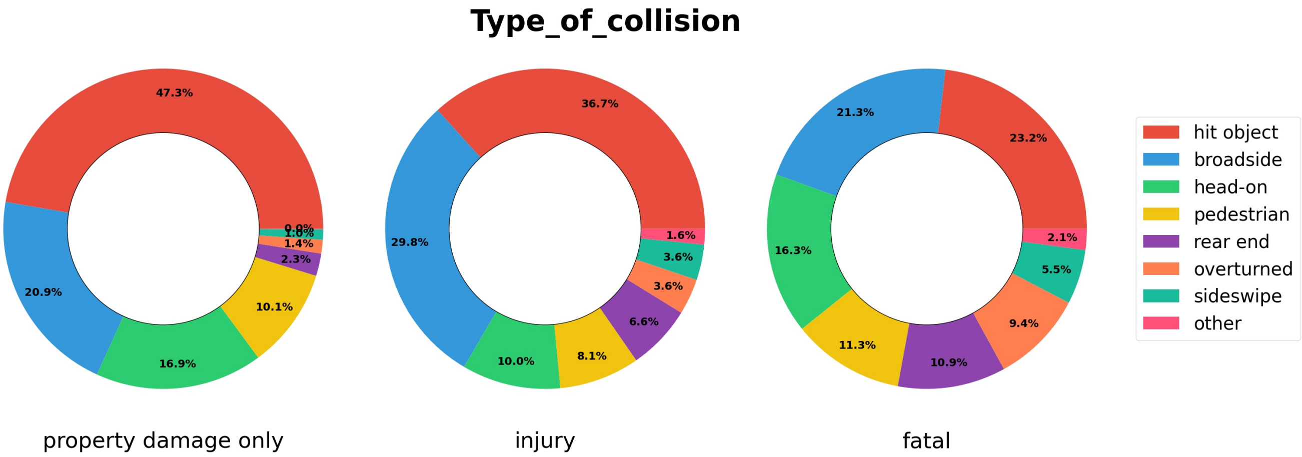
```
['property damage only', 'injury', 'fatal']
```

In [206…

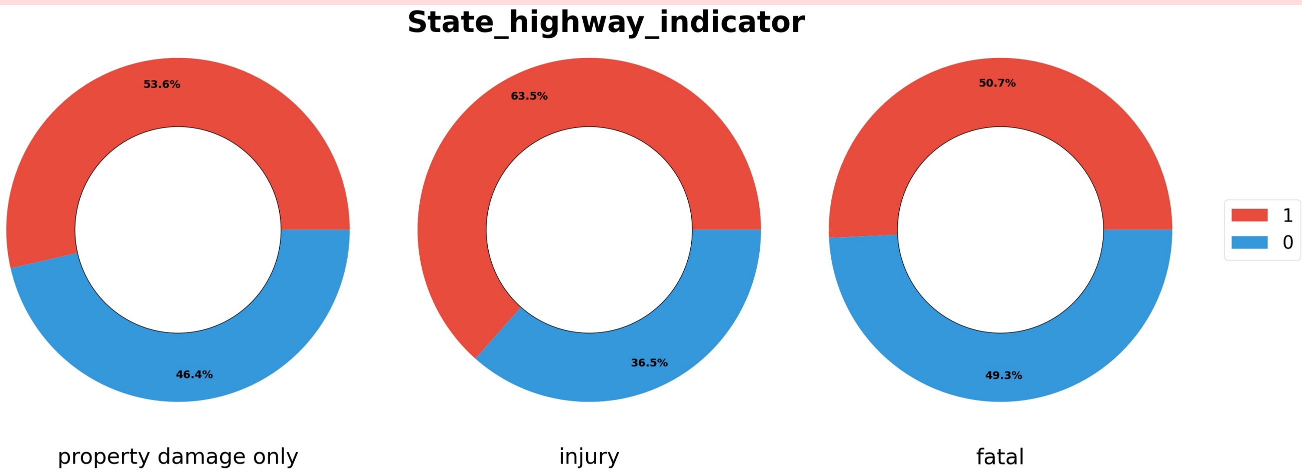
```
import matplotlib.pyplot as plt
from pyspark.sql.functions import desc

def generate_pie_chart(df, feat):
    colors = ['#E74C3C', '#3498DB', '#2ECC71', '#F1C40F', '#8E44AD', '#FF7F50', '#1ABC9C', '#FF5079','#A5A5A5',

    fig, ax = plt.subplots(1, 3, figsize=(30, 10))

    for i in range(3):
        spec_df = df.filter(df['collision_severity'] == cs_labels[i]).groupBy(feat).count().orderBy(desc('count
        labs = [row[feat] for row in spec_df.collect()]
        freqs = [row['count'] for row in spec_df.collect()]
        wedges, texts, autotexts = ax[i].pie(freqs, autopct='%1.1f%%', colors=colors, pctdistance=0.85,textprop
        centre_circle = plt.Circle((0, 0), 0.6, fc='white', ec='black')
        ax[i].add_artist(centre_circle)
        ax[i].set_xlabel(cs_labels[i], fontsize=36)

        # Make the percentage format bold
        for autotext in autotexts:
            autotext.set_fontweight('bold')

    lgd = ax[2].legend(wedges, labs,
                       loc="center left",
                       bbox_to_anchor=(1, 0, 0.5, 1),
                       prop={'size': 30})

    fig.suptitle(feat.capitalize(), fontsize=48, fontweight="bold", x=0.48, y=1)
    plt.tight_layout(pad=-3.0)

    plt.savefig('{0}.png'.format(feat), bbox_extra_artists=(lgd,), bbox_inches='tight')
    plt.show()
```

In [207…

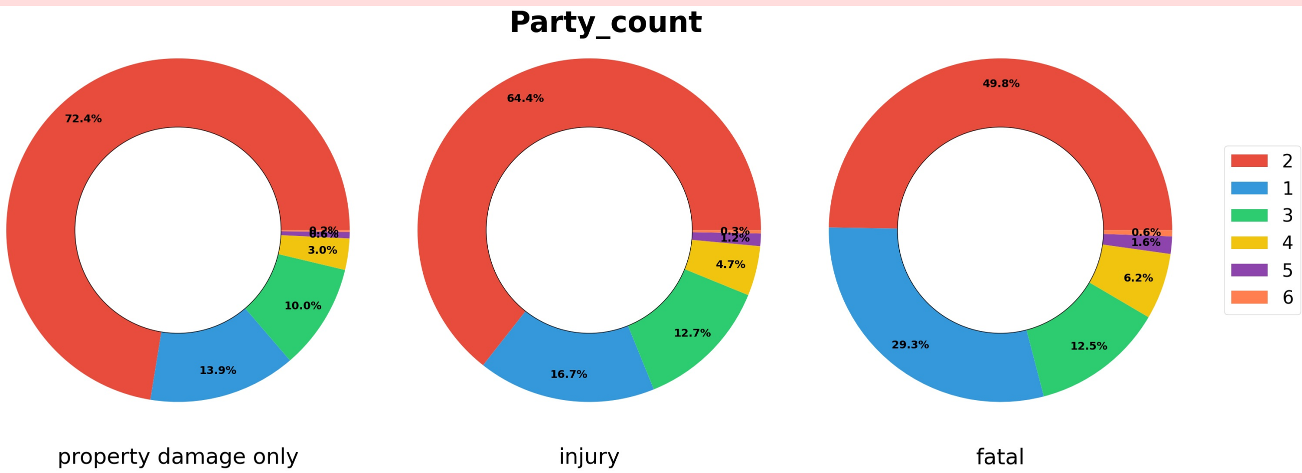
```
generate_pie_chart(sample_df, "type_of_collision")
```



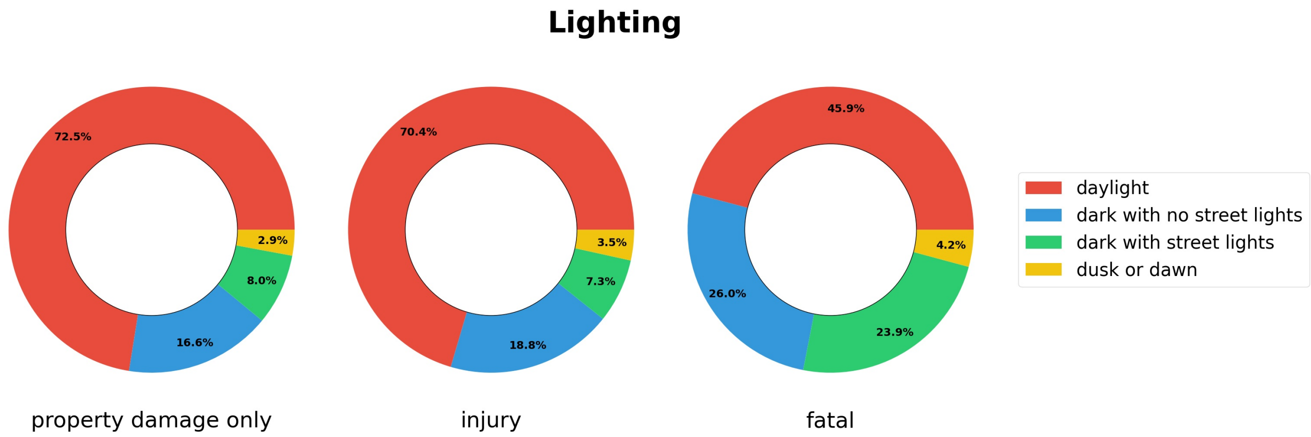
In [208…
```
generate_pie_chart(sample_df, "state_highway_indicator")
```



In [209…
```
generate_pie_chart(sample_df, "party_count")
```

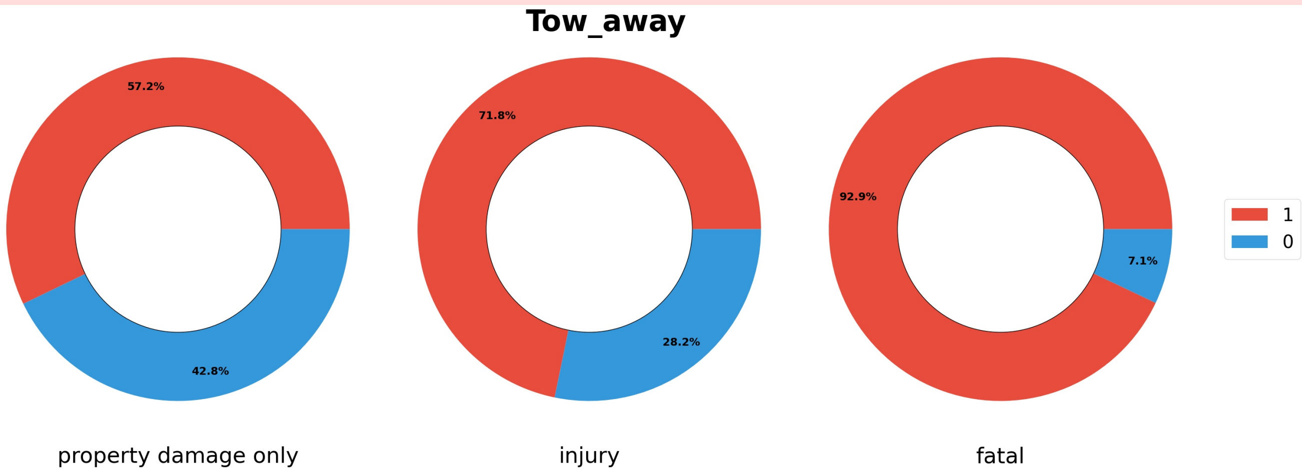


In [210…
```
generate_pie_chart(sample_df, "lighting")
```



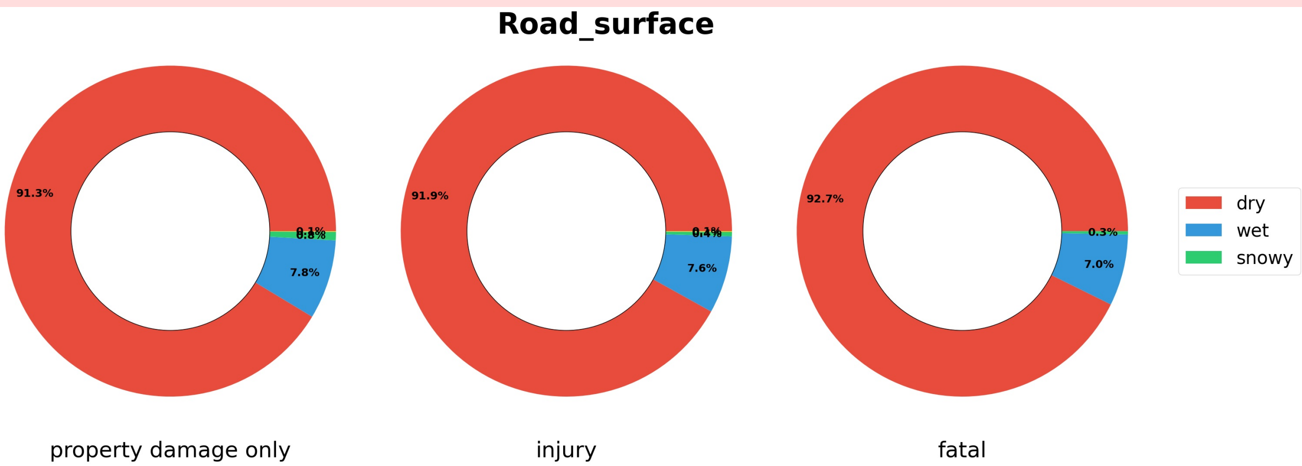
In [211…

```
generate_pie_chart(sample_df, "tow_away")
```



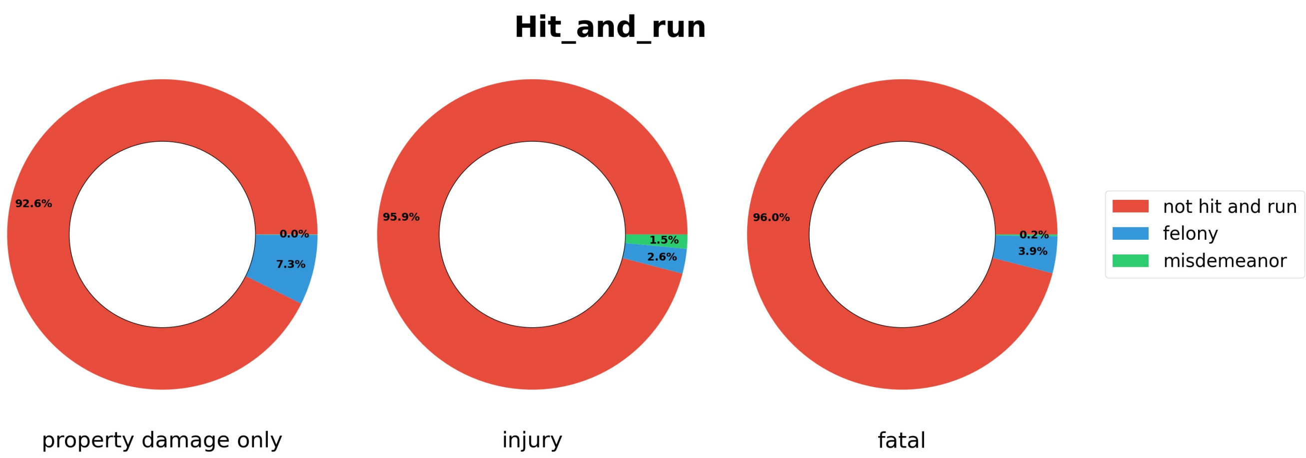
In [212…

```
generate_pie_chart(sample_df, "road_surface")
```



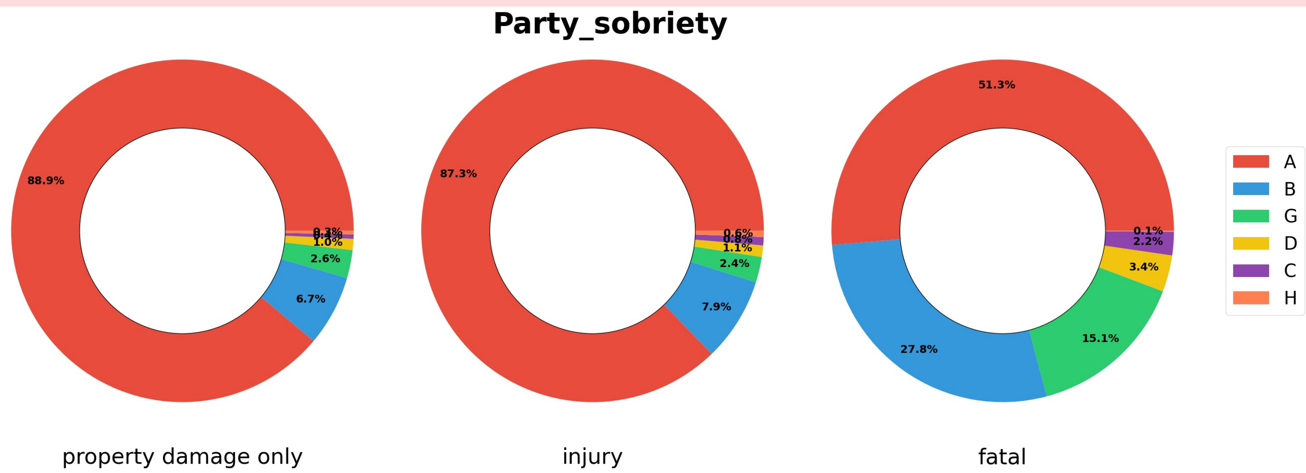
In [213…

```
generate_pie_chart(sample_df, "hit_and_run")
```



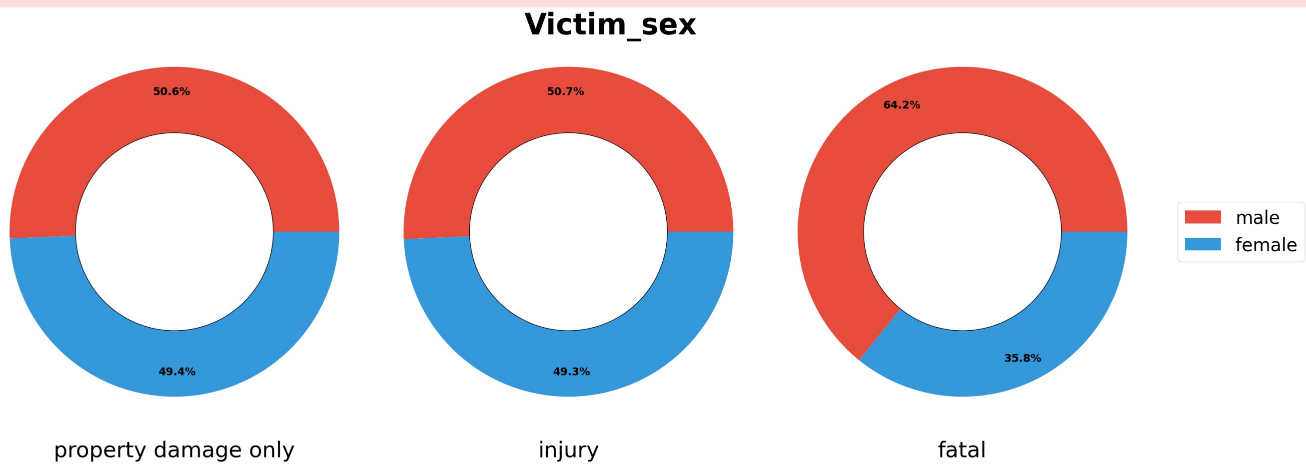
In [214…

```
generate_pie_chart(sample_df, "party_sobriety")
```



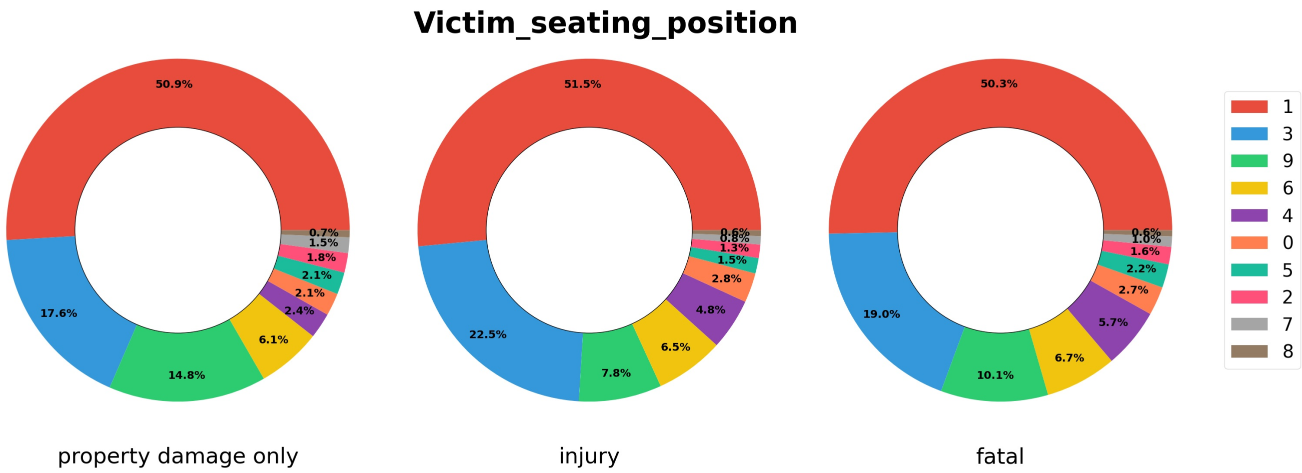
In [215…

```
generate_pie_chart(sample_df, "victim_sex")
```



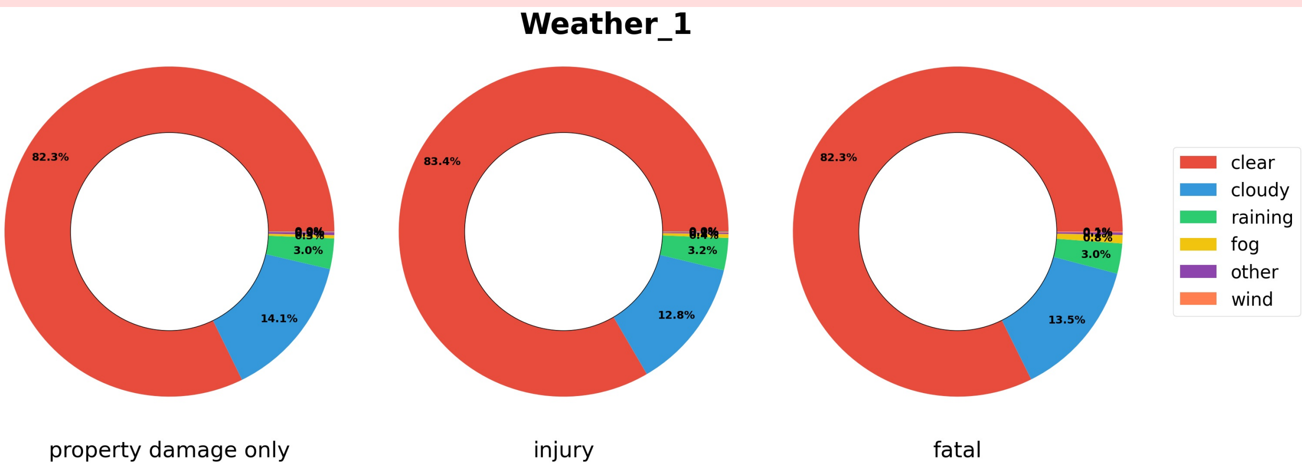
In [216…

```
generate_pie_chart(sample_df, "victim_seating_position")
```



In [217…

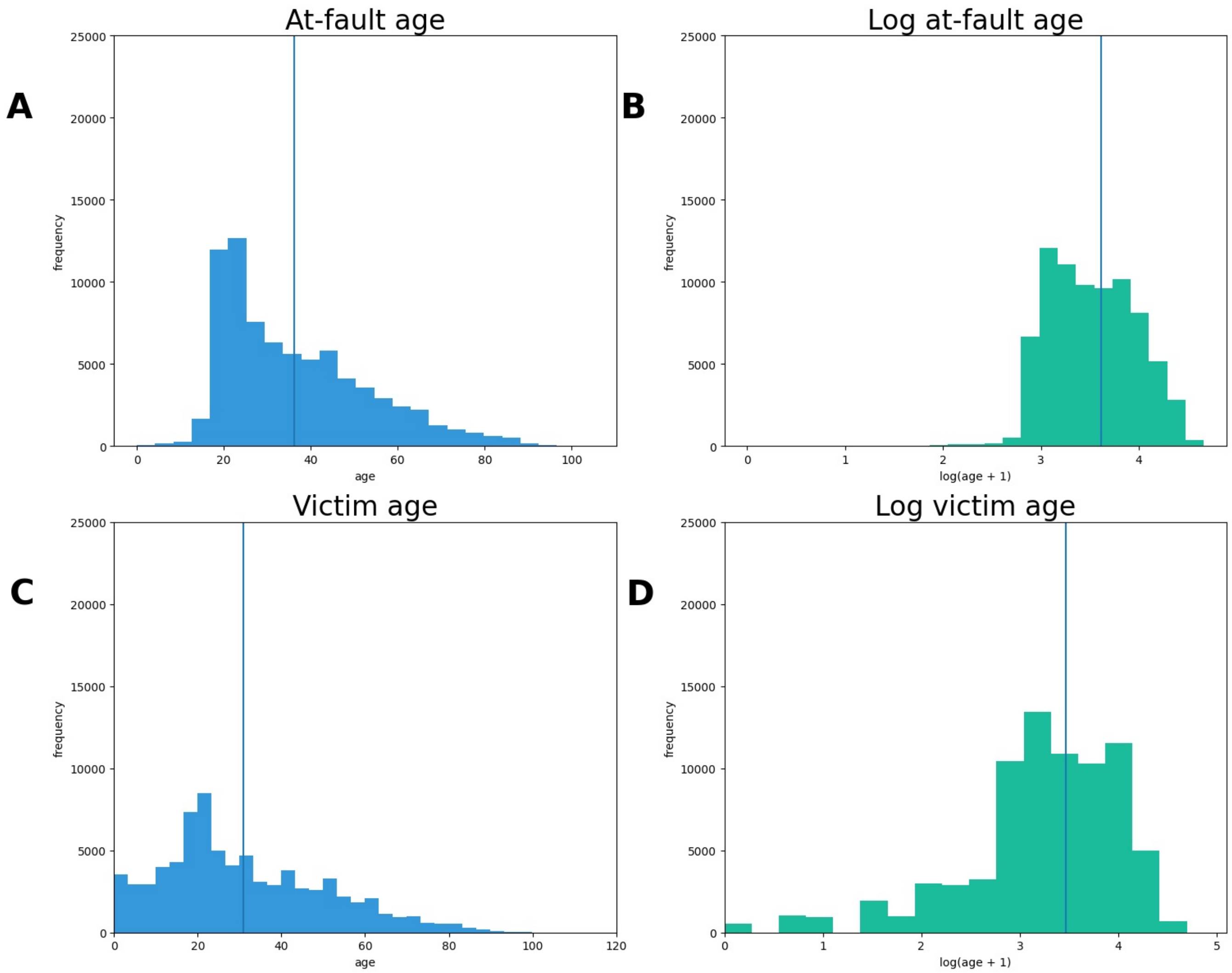
```
generate_pie_chart(sample_df, "weather_1")
```



In [25]:

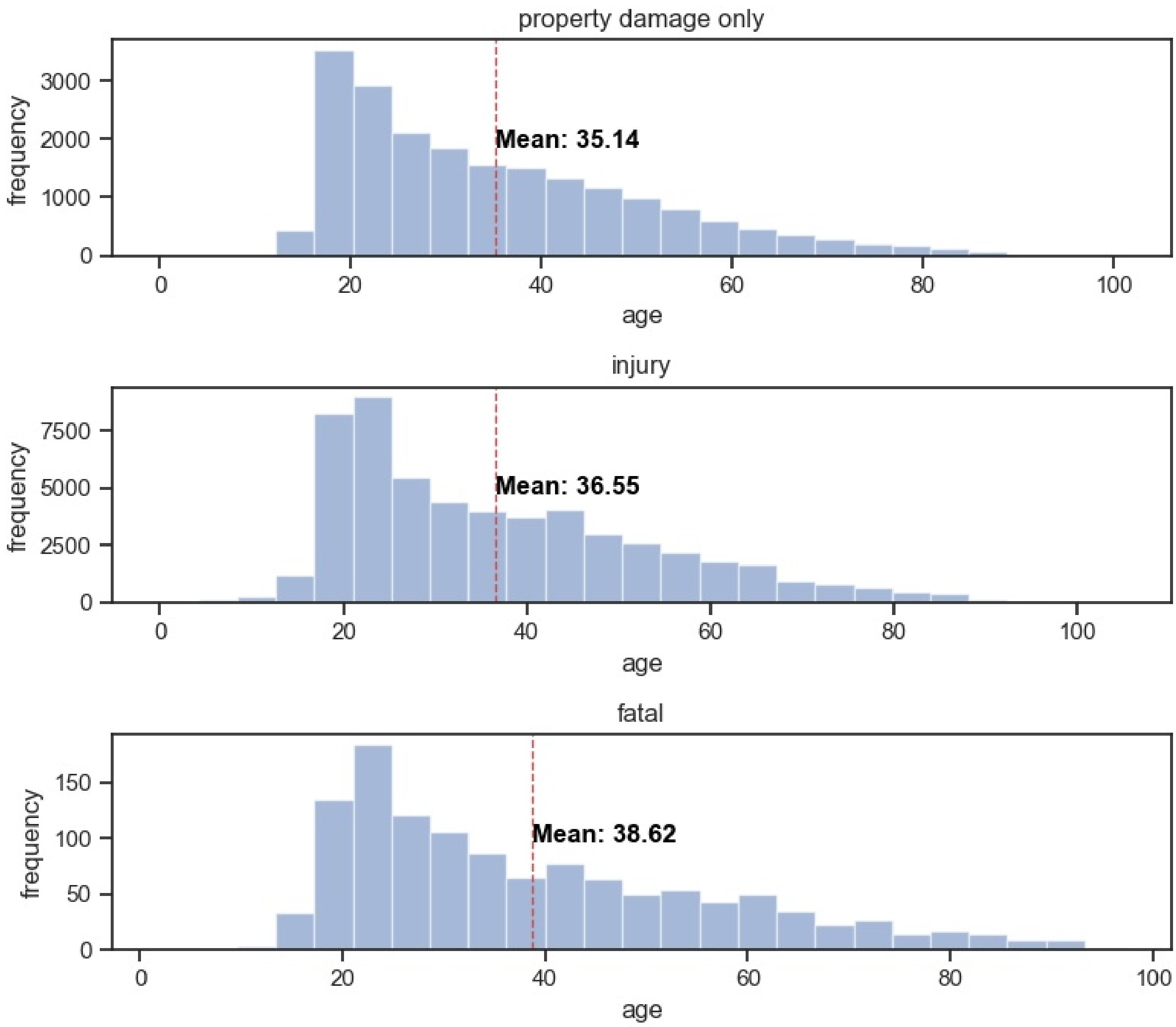
```
import matplotlib.pyplot as plt
from pyspark.sql.functions import mean
import numpy as np

fig, ax = plt.subplots(2, 2, figsize=(15, 12))

df_age = sample_df.select('party_age').toPandas()
df_victim_age = sample_df.select('victim_age').toPandas()

ax[0, 0].hist(df_age['party_age'], bins=25, color='#3498DB')
ax[0, 0].vlines(df_age['party_age'].mean(), 0, 25000)
ax[0, 0].set_ylim(0,25000)
ax[0, 0].set_title('At-fault age', fontsize=24)
ax[0, 0].set_ylabel('frequency')
ax[0, 0].set_xlabel('age')
ax[0, 0].text(-30, 20000, 'A', size=30, weight='bold')

ax[0, 1].hist(np.log(np.asarray(df_age['party_age']) + 1), bins=25,color='#1ABC9C')
ax[0, 1].vlines(np.log(df_age['party_age'].mean()+1), 0, 25000)
ax[0, 1].set_ylim(0, 25000)
ax[0, 1].set_title('Log at-fault age',fontsize=24)
ax[0, 1].set_ylabel('frequency')
ax[0, 1].set_xlabel('log(age + 1)')
ax[0, 1].text(-1.3, 20000, 'B', size=30, weight='bold')

ax[1, 0].hist(df_victim_age['victim_age'], bins=300, color='#3498DB')
ax[1, 0].set_xlim(0, 120)
ax[1, 0].vlines(df_victim_age['victim_age'].mean(), 0, 25000)
ax[1, 0].set_ylim(0, 25000)
ax[1, 0].set_title('Victim age',fontsize=24)
ax[1, 0].set_ylabel('frequency')
ax[1, 0].set_xlabel('age')
ax[1, 0].text(-25, 20000, 'C', size=30, weight='bold')

ax[1, 1].hist(np.log(np.asarray(df_victim_age['victim_age']) + 1), bins=25, color='#1ABC9C')
ax[1, 1].vlines(np.log(df_victim_age['victim_age'].mean()+1), 0, 25000)
ax[1, 1].set_xlim(0, 5.1)
ax[1, 1].set_ylim(0, 25000)
ax[1, 1].set_title('Log victim age',fontsize=24)
ax[1, 1].set_ylabel('frequency')
ax[1, 1].set_xlabel('log(age + 1)')
ax[1, 1].text(-1, 20000, 'D', size=30, weight='bold')
```

```
plt.savefig('ages.png')

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



In [108…

```
fig, ax = plt.subplots(3, 1, figsize=(8, 7))

for i in range(3):
    spec_df = sample_df.filter(sample_df['collision_severity'] == cs_labels[i]).select('party_age').toPandas()
    ax[i].hist(spec_df['party_age'], bins=25, alpha=0.5)
    ax[i].set_title(cs_labels[i])
    ax[i].set_xlabel('age')
    ax[i].set_ylabel('frequency')
    ax[i].axvline(spec_df['party_age'].mean(), color='r', linestyle='dashed', linewidth=1)
    ax[i].text(spec_df['party_age'].mean(), ax[i].get_ylim()[1]*0.5, f"Mean: {spec_df['party_age'].mean():.2f}"


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



Defining the Numerical , categorical and target variables

In [143…
```
NUMERICAL_FEATURES = ['party_age','victim_age']
CATEGORICAL_FEATURES = [
                        'type_of_collision',
                        'weather_1',
                        'state_highway_indicator',
                        'party_sobriety',
                        'lighting',
                        'tow_away',
                        'hit_and_run',
                        'victim_sex',
                        'victim_seating_position',
                        'party_count',
                        'road_surface'
                         ]
TARGET_VARIABLE = 'collision_severity'
```

In [144…
```
print("{:d} Numerical features = [{:s}]".format(len(NUMERICAL_FEATURES), ", ".join(["`{:s}`".format(nf) for nf 
print("{:d} Categorical features = [{:s}]".format(len(CATEGORICAL_FEATURES), ", ".join(["`{:s}`".format(nf) for
print("1 Target variable = `{:s}`".format(TARGET_VARIABLE))
```

```
2 Numerical features = [`party_age`, `victim_age`]
11 Categorical features = [`type_of_collision`, `weather_1`, `state_highway_indicator`, `party_sobriety`, `ligh
ting`, `tow_away`, `hit_and_run`, `victim_sex`, `victim_seating_position`, `party_count`, `road_surface`]
1 Target variable = `collision_severity`
```

Model Building

logarithmic transformation of numerical varibales

In [145…

```python
from pyspark.sql.functions import col, log
# Logarithmic transformation of numerical variables
df_transformed = sample_df.withColumn('party_age', log(col('party_age') + 0.01))
df_transformed = df_transformed.withColumn('victim_age', log(col('victim_age') + 0.01))
```

Logistic Regression

In [147…

```python
def get_index(df,categoricalCols,continuousCols,labelCol):

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
    from pyspark.sql.functions import col

    indexers = [ StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c))
                 for c in categoricalCols ]

    # default setting: dropLast=True
    encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),
                 outputCol="{0}_encoded".format(indexer.getOutputCol()))
                 for indexer in indexers ]

    assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in encoders]
                                + continuousCols, outputCol="features")

    pipeline = Pipeline(stages=indexers + encoders + [assembler])

    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(labelCol))

    return data.select('features','label')
```

In [148…

```python
data = get_index(df_transformed,CATEGORICAL_FEATURES,NUMERICAL_FEATURES,TARGET_VARIABLE)
data.show(5)
```

```
[Stage 4752:====================================================> (26 + 1) / 27]
+--------------------+--------------------+
|            features|               label|
+--------------------+--------------------+
|(45,[0,7,14,19,23...|property damage only|
|(45,[0,7,14,19,23...|property damage only|
|(45,[0,7,15,19,23...|property damage only|
|(45,[2,7,13,15,20...|property damage only|
|(45,[1,7,13,14,19...|property damage only|
+--------------------+--------------------+
only showing top 5 rows
```

In [149…

```python
from pyspark.ml.feature import StringIndexer
# Index labels, adding metadata to the label column
labelIndexer = StringIndexer(inputCol='label',
                             outputCol='indexedLabel').fit(data)
labelIndexer.transform(data).show(5, True)
```

```
[Stage 4791:====================================================> (26 + 1) / 27]
+--------------------+--------------------+------------+
|            features|               label|indexedLabel|
+--------------------+--------------------+------------+
|(45,[0,7,14,19,23...|property damage only|         1.0|
|(45,[0,7,14,19,23...|property damage only|         1.0|
|(45,[0,7,15,19,23...|property damage only|         1.0|
|(45,[2,7,13,15,20...|property damage only|         1.0|
|(45,[1,7,13,14,19...|property damage only|         1.0|
+--------------------+--------------------+------------+
only showing top 5 rows
```

In [150…

```python
from pyspark.ml.feature import VectorIndexer
# Automatically identify categorical features, and index them.
# Set maxCategories so features with > 4 distinct values are treated as continuous.
featureIndexer =VectorIndexer(inputCol="features", \
```

```
                                  outputCol="indexedFeatures", \
                                  maxCategories=4).fit(data)
featureIndexer.transform(data).show(5, True)
```

```
[Stage 4822:==================================================>   (25 + 2) / 27]
+--------------------+--------------------+--------------------+
|            features|               label|     indexedFeatures|
+--------------------+--------------------+--------------------+
|(45,[0,7,14,19,23...|property damage only|(45,[0,7,14,19,23...|
|(45,[0,7,14,19,23...|property damage only|(45,[0,7,14,19,23...|
|(45,[0,7,15,19,23...|property damage only|(45,[0,7,15,19,23...|
|(45,[2,7,13,15,20...|property damage only|(45,[2,7,13,15,20...|
|(45,[1,7,13,14,19...|property damage only|(45,[1,7,13,14,19...|
+--------------------+--------------------+--------------------+
only showing top 5 rows
```

In [151…
```
# Split the data into training and test sets (40% held out for testing)
(trainingData, testData) = data.randomSplit([0.6, 0.4])

trainingData.show(5,False)
testData.show(5,False)
```

```
+-----------------------------------------------------------------------------------------------------------------
-----------------+--------------------+
|features
|label               |
+-----------------------------------------------------------------------------------------------------------------
-----------------+--------------------+
|(45,[0,7,13,14,19,22,23,25,27,35,40,43,44],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,4.189806245700679,3.68
91294228691436]) |property damage only|
|(45,[0,7,13,14,19,22,23,25,27,35,40,43,44],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,4.442768896629268,4.31
7621437981545])  |property damage only|
|(45,[0,7,13,14,19,22,23,25,27,36,40,43,44],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,2.7732135270086236,3.9
891692146047997])|property damage only|
|(45,[0,7,13,14,19,22,23,25,27,38,40,43,44],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,3.1359289040472746,3.4
015306594522756])|property damage only|
|(45,[0,7,13,14,19,22,23,25,28,36,40,43,44],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,3.850360345036039,4.20
484186198508])   |property damage only|
+-----------------------------------------------------------------------------------------------------------------
-----------------+--------------------+
only showing top 5 rows
```

```
[Stage 4860:>                                                        (0 + 1) / 1]
+-----------------------------------------------------------------------------------------------------------------
-----------------+--------------------+
|features
|label               |
+-----------------------------------------------------------------------------------------------------------------
-----------------+--------------------+
|(45,[0,7,13,14,19,22,23,25,27,35,40,43,44],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,2.8909271591878647,2.7
732135270086236])|property damage only|
|(45,[0,7,13,14,19,22,23,25,27,36,40,43,44],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,3.332561589272017,4.36
9574426734642])  |property damage only|
|(45,[0,7,13,14,19,22,23,25,27,36,40,43,44],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,3.555633734966574,3.66
3818023518565])  |property damage only|
|(45,[0,7,13,14,19,22,23,25,27,36,40,43,44],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,3.583796677660784,3.55
5633734966574])  |property damage only|
|(45,[0,7,13,14,19,22,23,25,27,38,40,43,44],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,3.2962071678045244,3.2
962071678045244])|property damage only|
+-----------------------------------------------------------------------------------------------------------------
-----------------+--------------------+
only showing top 5 rows
```

In [152…
```
from pyspark.ml.classification import LogisticRegression
logr = LogisticRegression(featuresCol='indexedFeatures', labelCol='indexedLabel')
```

In [153…
```
# Convert indexed labels back to original labels.
from pyspark.ml.feature import IndexToString
labelConverter = IndexToString(inputCol="prediction", outputCol="predictedLabel",
                               labels=labelIndexer.labels)
```

In [154…
```
from pyspark.ml import Pipeline
# Chain indexers and tree in a Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, logr,labelConverter])
```

In [155…
```
# Train model.  This also runs the indexers.
model = pipeline.fit(trainingData)
```

```
[Stage 5844:====================================================> (26 + 1) / 27]
```

In [156…
```
# Make predictions.
```

```
predictions = model.transform(testData)
# Select example rows to display.
predictions.select("features","label","predictedLabel").show(5)
```

```
[Stage 5859:>                                                       (0 + 1) / 1]
+--------------------+--------------------+--------------+
|            features|               label|predictedLabel|
+--------------------+--------------------+--------------+
|(45,[0,7,13,14,19...|property damage only|        injury|
|(45,[0,7,13,14,19...|property damage only|        injury|
|(45,[0,7,13,14,19...|property damage only|        injury|
|(45,[0,7,13,14,19...|property damage only|        injury|
|(45,[0,7,13,14,19...|property damage only|        injury|
+--------------------+--------------------+--------------+
only showing top 5 rows
```

In [101…

```
predictions.show(1)
```

```
[Stage 4629:===================================================> (26 + 1) / 27]
+--------------------+--------------------+------------+--------------------+--------------------+-------------
-------+----------+--------------+
|            features|               label|indexedLabel|     indexedFeatures|       rawPrediction|         prob
ability|prediction|predictedLabel|
+--------------------+--------------------+------------+--------------------+--------------------+-------------
-------+----------+--------------+
|(45,[0,7,13,14,19...|property damage only|         1.0|(45,[0,7,13,14,19...|[2.25730424771962...|[0.6798779632
7185...|       0.0|        injury|
+--------------------+--------------------+------------+--------------------+--------------------+-------------
-------+----------+--------------+
only showing top 1 row
```

In [102…

```
predictions.printSchema()
```

```
root
 |-- features: vector (nullable = true)
 |-- label: string (nullable = true)
 |-- indexedLabel: double (nullable = false)
 |-- indexedFeatures: vector (nullable = true)
 |-- rawPrediction: vector (nullable = true)
 |-- probability: vector (nullable = true)
 |-- prediction: double (nullable = false)
 |-- predictedLabel: string (nullable = true)
```

Logistic Regression Classifier Performance Evaluation

In [157…

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("accuracy =",accuracy)
print("Test Error = %g" % (1.0 - accuracy))
```

```
[Stage 5874:===================================================> (121 + 4) / 125]
accuracy = 0.7915233335496852
Test Error = 0.208477
```

In [158…

```
# Calculate other metrics such as precision, recall, and F1-score
precision = evaluator.evaluate(predictions, {evaluator.metricName: "weightedPrecision"})
recall = evaluator.evaluate(predictions, {evaluator.metricName: "weightedRecall"})
f1_score = evaluator.evaluate(predictions, {evaluator.metricName: "f1"})

print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
```

```
[Stage 5922:====================================================>(123 + 2) / 125]
Precision: 0.7822998363517759
Recall: 0.7915233335496852
F1-score: 0.7818560638846146
```

Confusion Matrix for logistic Regression

In [159…

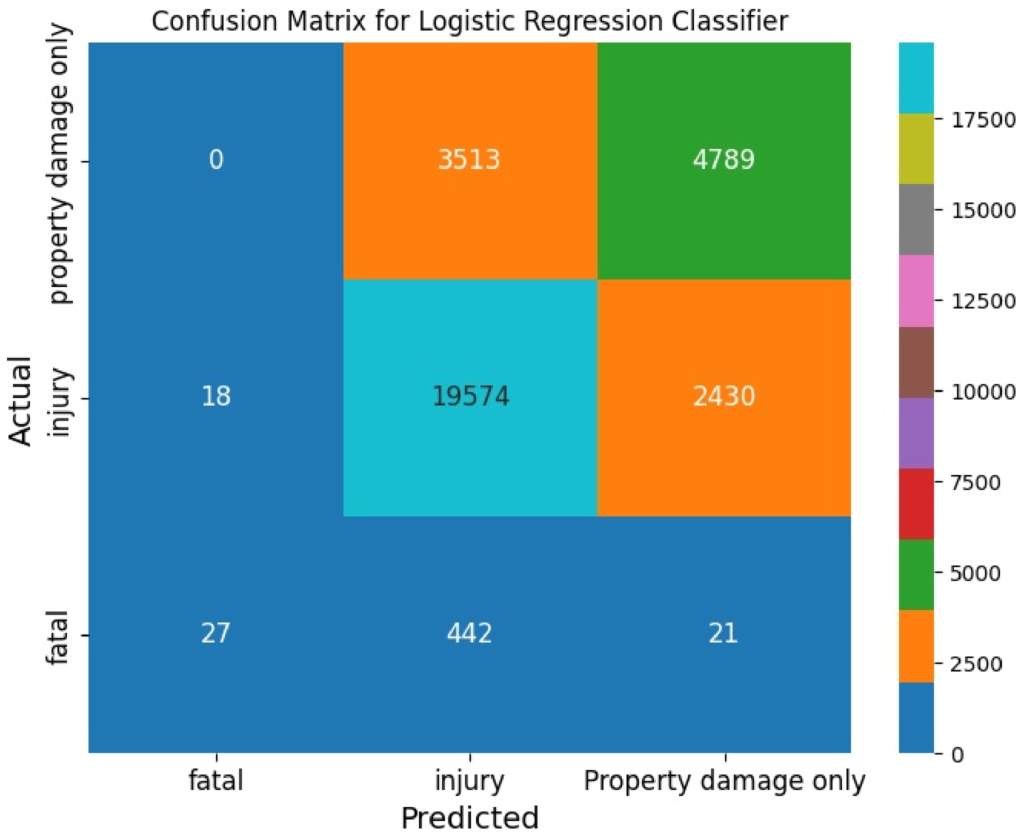
```
confusion_matrix_train = predictions.groupBy('label').pivot('predictedLabel').count().na.fill(0)
confusion_matrix_train.show()
```

```
[Stage 5972:====================================================>(124 + 1) / 125]
```

```
+--------------------+-----+------+--------------------+
|               label|fatal|injury|property damage only|
+--------------------+-----+------+--------------------+
|property damage only|    0|  3513|                4789|
|              injury|   18| 19574|                2430|
|               fatal|   27|   442|                  21|
+--------------------+-----+------+--------------------+
```

In [160…

```python
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

cs_labels = ["property damage only","injury", "fatal"]
cs_labels2 =['fatal','injury','Property damage only']

# Convert PySpark DataFrame to Pandas DataFrame
confusion_matrix_pd1 = confusion_matrix_train.toPandas()

# Set 'label' column as the index
confusion_matrix_pd1.set_index('label', inplace=True)

# Convert the DataFrame to a matrix
confusion_matrix_np1 = confusion_matrix_pd1.to_numpy()
plt.figure(figsize=(8, 6))
# Plot confusion matrix heatmap
ax = sns.heatmap(confusion_matrix_np1, annot=True, fmt=".0f", cmap="tab10",
                 annot_kws={"fontsize": 12})

plt.title('Confusion Matrix for Logistic Regression Classifier')
plt.xlabel('Predicted', fontsize =14)
plt.ylabel('Actual', fontsize =14)

# Rotate and align x-axis tick labels
ax.set_xticklabels(cs_labels2, rotation=0, ha='center', fontsize=12)  # Adjust the rotation angle and font size

# Rotate and align y-axis tick labels
ax.set_yticklabels(cs_labels, rotation=90, ha='right', fontsize=12)  # Adjust the rotation angle and font size

plt.show()
```



Decision Tree Classifier

In [164…

```python
# This function defines the general pipeline for logistic regression
def get_index2(df,categorical_features,numerical_features,target_variable):

    from pyspark.ml.feature import StringIndexer, VectorAssembler
    from pyspark.ml import Pipeline
```

```
    # Configure a decision tree pipeline, which consists of the following stages:

    indexers = [StringIndexer(inputCol=f, outputCol="{0}_indexed".format(f), handleInvalid="keep") for f in cat

    # Indexing the target column (i.e., transform it into 0/1) and rename it as "label"
    # Note that by default StringIndexer will assign the value `0` to the most frequent label, which in the cas
    # As such, this nicely resembles the idea of having `deposit = 0` if no deposit is subscribed, or `deposit
    label_indexer = StringIndexer(inputCol = target_variable, outputCol = "label")

    # Assemble all the features (both one-hot-encoded categorical and numerical) into a single vector
    assembler = VectorAssembler(inputCols=[indexer.getOutputCol() for indexer in indexers] + numerical_features

    # Populate the stages of the pipeline with all the preprocessing steps
    stages = indexers + [label_indexer] + [assembler] # + ...


    # 6. Set up the pipeline
    pipeline = Pipeline(stages=stages)
    model=pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label',col(target_variable))
    return data.select('features','label')
```

In [ ]:
```
data2 = get_index2(df_transformed,CATEGORICAL_FEATURES,NUMERICAL_FEATURES,TARGET_VARIABLE)
```

In [135…
```
data2.show()
```

```
[Stage 4421:=====================================================> (26 + 1) / 27]
+--------------------+-------------------+
|            features|              label|
+--------------------+-------------------+
|(13,[2,5,8,11,12]...|property damage only|
|(13,[2,5,8,9,11,1...|property damage only|
|(13,[2,3,5,8,11,1...|property damage only|
|[2.0,0.0,0.0,1.0,...|property damage only|
|(13,[0,5,8,11,12]...|property damage only|
|(13,[2,8,9,11,12]...|property damage only|
|(13,[0,1,2,8,10,1...|property damage only|
|(13,[1,2,5,8,10,1...|property damage only|
|(13,[5,8,11,12],[...|property damage only|
|(13,[2,5,8,11,12]...|property damage only|
|[4.0,0.0,0.0,1.0,...|property damage only|
|(13,[2,7,8,11,12]...|property damage only|
|(13,[5,7,8,11,12]...|property damage only|
|(13,[1,2,4,8,11,1...|property damage only|
|(13,[1,2,8,9,10,1...|property damage only|
|(13,[2,5,7,8,9,11...|property damage only|
|(13,[2,5,7,8,9,11...|property damage only|
|(13,[2,5,7,8,11,1...|property damage only|
|(13,[2,8,9,11,12]...|property damage only|
|[2.0,0.0,0.0,2.0,...|property damage only|
+--------------------+-------------------+
only showing top 20 rows
```

In [95]:
```
# Index labels, adding metadata to the label column
labelIndexer = StringIndexer(inputCol='label',
                             outputCol='indexedLabel').fit(data2)
labelIndexer.transform(data2).show(5, True)
```

```
[Stage 3465:=====================================================> (26 + 1) / 27]
+--------------------+-------------------+------------+
|            features|              label|indexedLabel|
+--------------------+-------------------+------------+
|(13,[2,5,8,11,12]...|property damage only|         1.0|
|(13,[2,5,8,9,11,1...|property damage only|         1.0|
|(13,[2,3,5,8,11,1...|property damage only|         1.0|
|[2.0,0.0,0.0,1.0,...|property damage only|         1.0|
|(13,[0,5,8,11,12]...|property damage only|         1.0|
+--------------------+-------------------+------------+
only showing top 5 rows
```

In [96]:
```
# Split the data into training and test sets (40% held out for testing)
(trainingData, testData) = data2.randomSplit([0.6, 0.4])

trainingData.show(5)
testData.show(5)
```

```
+--------------------+--------------------+
|            features|               label|
+--------------------+--------------------+
|(13,[0,1,2,8,9,11...|property damage only|
|(13,[0,1,2,8,11,1...|property damage only|
|(13,[0,1,5,7,8,11...|property damage only|
|(13,[0,1,5,7,8,11...|property damage only|
|(13,[0,1,5,7,8,11...|property damage only|
+--------------------+--------------------+
only showing top 5 rows

[Stage 3495:=====================================================> (26 + 1) / 27]
+--------------------+--------------------+
|            features|               label|
+--------------------+--------------------+
|(13,[0,1,2,5,8,11...|property damage only|
|(13,[0,1,2,5,8,11...|property damage only|
|(13,[0,1,2,7,8,11...|property damage only|
|(13,[0,1,2,8,9,11...|property damage only|
|(13,[0,1,2,8,10,1...|property damage only|
+--------------------+--------------------+
only showing top 5 rows
```

In [97]:

```python
from pyspark.ml.classification import DecisionTreeClassifier

# Train a DecisionTree model
dt = DecisionTreeClassifier(labelCol='indexedLabel', featuresCol='features')
```

In [98]:

```python
# Convert indexed labels back to original labels.
labelConverter = IndexToString(inputCol="prediction", outputCol="predictedLabel",
                               labels=labelIndexer.labels)
```

In [100…

```python
# Chain indexers and tree in a Pipeline
pipeline2 = Pipeline(stages=[labelIndexer, dt,labelConverter])
```

In [101…

```python
# Train model.  This also runs the indexers.
model2 = pipeline.fit(trainingData)
```

In [165…

```python
# Make predictions.
predictions2 = model2.transform(testData)
# Select example rows to display.
predictions2.select("features","label","predictedLabel").show(5)
```

```
[Stage 6054:=====================================================> (26 + 1) / 27]
+--------------------+--------------------+--------------+
|            features|               label|predictedLabel|
+--------------------+--------------------+--------------+
|(45,[0,7,13,14,19...|property damage only|        injury|
|(45,[0,7,13,14,19...|property damage only|        injury|
|(45,[0,7,13,14,19...|property damage only|        injury|
|(45,[0,7,13,14,19...|property damage only|        injury|
|(45,[0,7,13,14,19...|property damage only|        injury|
+--------------------+--------------------+--------------+
only showing top 5 rows
```

Performance Evaluation for decision tree

In [166…

```python
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions2)
print("Accuracy=", accuracy)
print("Test Error = %g" % (1.0 - accuracy))

dtModel = model2.stages[-2]
print(dtModel)  # summary only
```

```
[Stage 6077:=================================================>   (117 + 8) / 125]
Accuracy= 0.478061919906536
Test Error = 0.521938
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_8435f29fb31d, depth=5, numNodes=39, numClasses=3, n
umFeatures=13
```

In [167…

```python
# Calculate other metrics such as precision, recall, and F1-score
precision = evaluator.evaluate(predictions2, {evaluator.metricName: "weightedPrecision"})
recall = evaluator.evaluate(predictions2, {evaluator.metricName: "weightedRecall"})
```

```
f1_score = evaluator.evaluate(predictions2, {evaluator.metricName: "f1"})

print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
```

```
[Stage 6125:==================================================>  (120 + 5) / 125]
Precision: 0.5478422239283823
Recall: 0.478061919906536
F1-score: 0.5012920195880273
```

Confusion matrix for decision tree classiffier

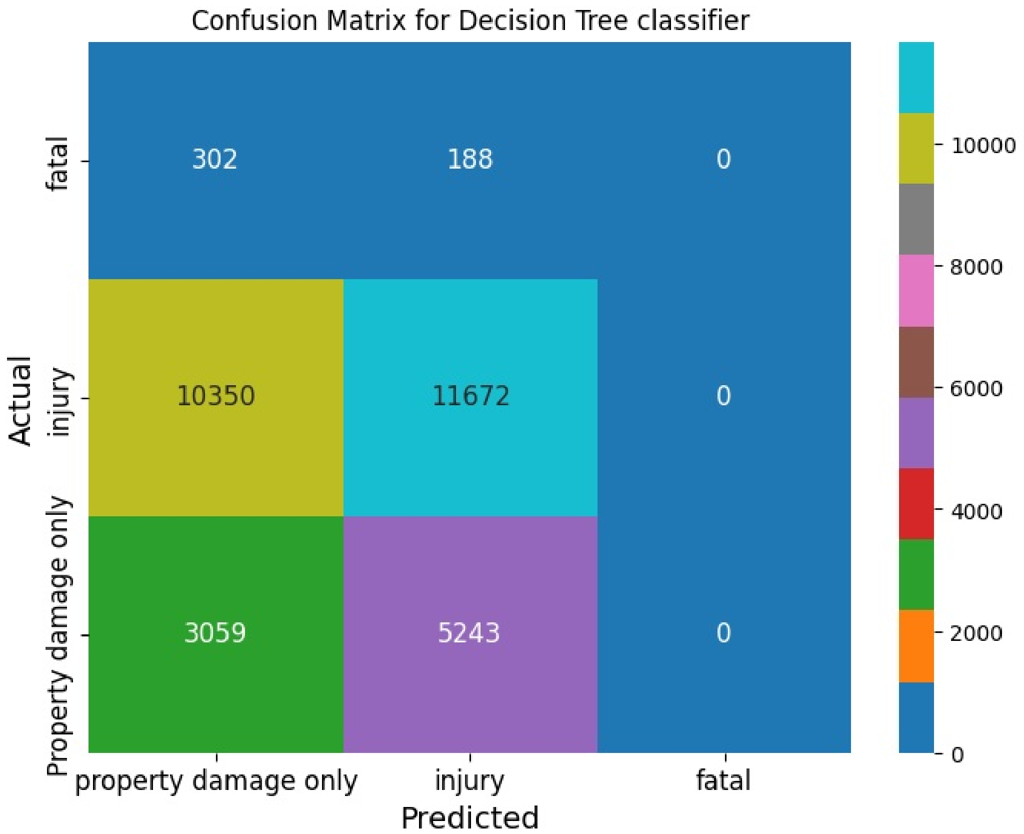
In [168…

```
all_labels = ["property damage only", "injury","fatal"]
confusion_matrix2 = predictions2.groupBy('label').pivot('predictedLabel',all_labels).count().na.fill(0).orderBy
confusion_matrix2.show()
```

```
[Stage 6141:==================================================>  (120 + 5) / 125]
+--------------------+--------------------+------+-----+
|               label|property damage only|injury|fatal|
+--------------------+--------------------+------+-----+
|               fatal|                 302|   188|    0|
|              injury|               10350| 11672|    0|
|property damage only|                3059|  5243|    0|
+--------------------+--------------------+------+-----+
```

In [169…

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

cs_labels = ["property damage only","injury", "fatal"]
cs_labels2 =['fatal','injury','Property damage only']

# Convert PySpark DataFrame to Pandas DataFrame
confusion_matrix_pd2 = confusion_matrix2.toPandas()

# Set 'label' column as the index
confusion_matrix_pd2.set_index('label', inplace=True)

# Convert the DataFrame to a matrix
confusion_matrix_np1 = confusion_matrix_pd2.to_numpy()
plt.figure(figsize=(8, 6))
# Plot confusion matrix heatmap
ax = sns.heatmap(confusion_matrix_np1, annot=True, fmt=".0f", cmap="tab10",
                 annot_kws={"fontsize": 12})

plt.title('Confusion Matrix for Decision Tree classifier')
plt.xlabel('Predicted',fontsize=14)
plt.ylabel('Actual',fontsize=14)

# Rotate and align x-axis tick labels
ax.set_xticklabels(cs_labels, rotation=0, ha='center', fontsize=12)  # Adjust the rotation angle and font size

# Rotate and align y-axis tick labels
ax.set_yticklabels(cs_labels2, rotation=90, ha='right', fontsize=12)  # Adjust the rotation angle and font size

plt.show()
```



In [109…

```
predictions2.groupBy('label').count().show()
```

```
[Stage 3832:===============================================>     (115 + 8) / 125]
+--------------------+-----+
|               label|count|
+--------------------+-----+
|property damage only| 8291|
|              injury|21871|
|               fatal|  479|
+--------------------+-----+
```

In [110…

```
predictions2.groupBy('predictedLabel').count().show()
```

```
[Stage 3856:=================================================>   (119 + 6) / 125]
+--------------------+-----+
|      predictedLabel|count|
+--------------------+-----+
|property damage only| 4584|
|              injury|26057|
+--------------------+-----+
```

Random Forest Classifier

In [170…
```
from pyspark.ml.classification import RandomForestClassifier

# Train a RandomForest model.
rf = RandomForestClassifier(labelCol="indexedLabel", featuresCol="features", numTrees=10)
```

In [171…
```
# Chain indexers and tree in a Pipeline
pipeline3 = Pipeline(stages=[labelIndexer, rf,labelConverter])
```

In [172…
```
# Train model.  This also runs the indexers.
model3 = pipeline3.fit(trainingData)
```

```
[Stage 6306:===================================================>   (25 + 2) / 27]
```

In [173…
```
# Make predictions.
predictions3 = model3.transform(testData)
# Select example rows to display.
predictions3.select("features","label","predictedLabel").show(5)
```

```
[Stage 6313:=====================================================> (26 + 1) / 27]
+--------------------+--------------------+--------------+
|            features|               label|predictedLabel|
+--------------------+--------------------+--------------+
|(45,[0,7,13,14,19...|property damage only|        injury|
|(45,[0,7,13,14,19...|property damage only|        injury|
|(45,[0,7,13,14,19...|property damage only|        injury|
|(45,[0,7,13,14,19...|property damage only|        injury|
|(45,[0,7,13,14,19...|property damage only|        injury|
+--------------------+--------------------+--------------+
only showing top 5 rows
```

Performance Evaluation of Random Forest Classifier

In [174…
```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions3)
print("Accuracy=", accuracy)
print("Test Error = %g" % (1.0 - accuracy))

rfModel = model3.stages[-2]
print(rfModel)  # summary only
```

```
[Stage 6336:==================================================>  (119 + 6) / 125]
Accuracy= 0.7368079444408385
Test Error = 0.263192
RandomForestClassificationModel: uid=RandomForestClassifier_ea7a405b18ea, numTrees=10, numClasses=3, numFeature
s=45
```

In [175…
```
# Calculate other metrics such as precision, recall, and F1-score
precision = evaluator.evaluate(predictions3, {evaluator.metricName: "weightedPrecision"})
recall = evaluator.evaluate(predictions3, {evaluator.metricName: "weightedRecall"})
f1_score = evaluator.evaluate(predictions3, {evaluator.metricName: "f1"})

print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
```

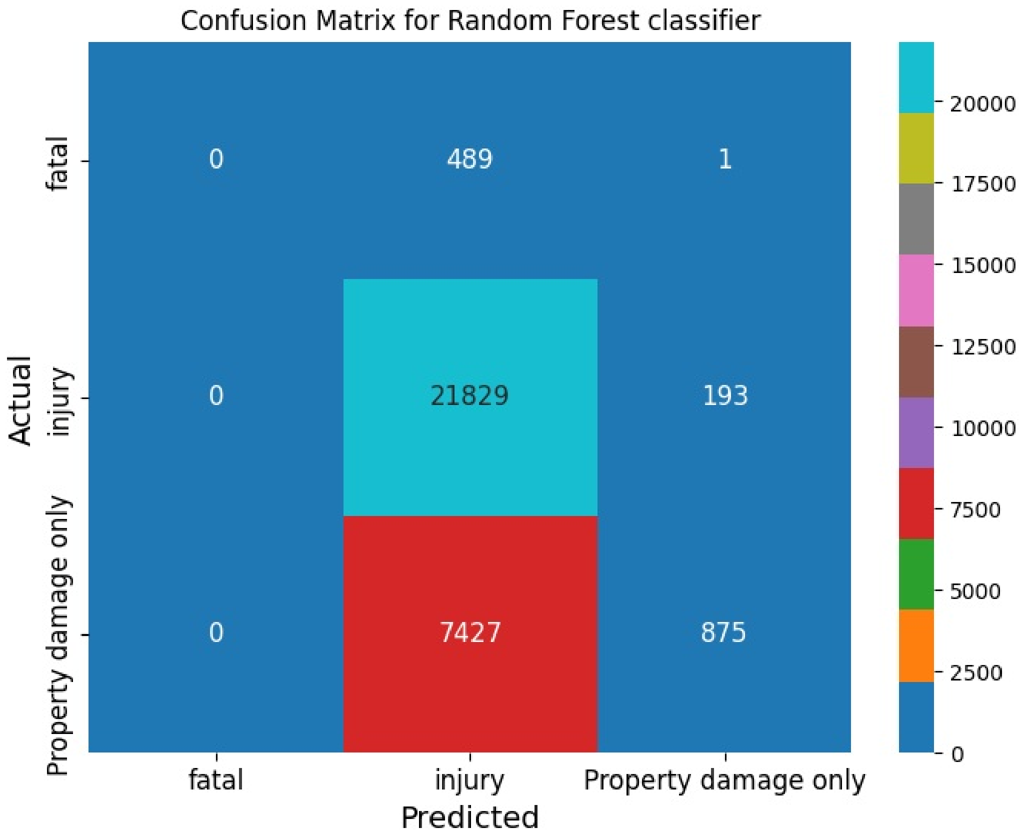
```
[Stage 6384:===================================================> (121 + 4) / 125]
Precision: 0.7450081757144604
Recall: 0.7368079444408386
F1-score: 0.6530391936424277
```

Confusion matrix for Random forest classifier

In [176…
```
all_labels = ["fatal", "injury", "property damage only"]
confusion_matrix3 = predictions3.groupBy('label').pivot('predictedLabel',all_labels).count().na.fill(0).orderBy
confusion_matrix3.show()
```

```
[Stage 6400:==================================================>  (120 + 5) / 125]
+--------------------+-----+------+--------------------+
|               label|fatal|injury|property damage only|
+--------------------+-----+------+--------------------+
|               fatal|    0|   489|                   1|
|              injury|    0| 21829|                 193|
|property damage only|    0|  7427|                 875|
+--------------------+-----+------+--------------------+
```

In [177…
```python
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

cs_labels2 =['fatal','injury','Property damage only']

# Convert PySpark DataFrame to Pandas DataFrame
confusion_matrix_pd3 = confusion_matrix3.toPandas()

# Set 'label' column as the index
confusion_matrix_pd3.set_index('label', inplace=True)

# Convert the DataFrame to a matrix
confusion_matrix_np2 = confusion_matrix_pd3.to_numpy()
plt.figure(figsize=(8, 6))
# Plot confusion matrix heatmap
ax = sns.heatmap(confusion_matrix_np2, annot=True, fmt=".0f", cmap="tab10",
                 annot_kws={"fontsize": 12})

plt.title('Confusion Matrix for Random Forest classifier')
plt.xlabel('Predicted',fontsize=14)
plt.ylabel('Actual',fontsize=14)

# Rotate and align x-axis tick labels
ax.set_xticklabels(cs_labels2, rotation=0, ha='center', fontsize=12)  # Adjust the rotation angle and font size

# Rotate and align y-axis tick labels
ax.set_yticklabels(cs_labels2, rotation=90, ha='right', fontsize=12)  # Adjust the rotation angle and font size

plt.show()
```



Naive Bayes (Gaussian) classifier

In [178…
```python
from pyspark.ml.classification import NaiveBayes
nb = NaiveBayes(labelCol="indexedLabel", featuresCol="features",smoothing=1.0, modelType="gaussian")
```

In [179…
```python
# Chain indexers and tree in a Pipeline
pipeline4 = Pipeline(stages=[labelIndexer, nb,labelConverter])
```

In [180…
```python
# Train model.  This also runs the indexers.
model4 = pipeline4.fit(trainingData)
```

In [181…
```python
# Make predictions.
predictions4 = model4.transform(testData)
# Select example rows to display.
predictions4.select("features","label","predictedLabel").show(5)
```

```
[Stage 6512:=====================================================> (26 + 1) / 27]
```

```
+--------------------+--------------------+-------------+
|            features|               label|predictedLabel|
+--------------------+--------------------+-------------+
|(45,[0,7,13,14,19...|property damage only|        fatal|
|(45,[0,7,13,14,19...|property damage only|        fatal|
|(45,[0,7,13,14,19...|property damage only|        fatal|
|(45,[0,7,13,14,19...|property damage only|        fatal|
|(45,[0,7,13,14,19...|property damage only|        fatal|
+--------------------+--------------------+-------------+
only showing top 5 rows
```

Performance Evaluation of Gaussian Naive Bayes

In [182…

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions4)
print("Accuracy=", accuracy)
print("Test Error = %g" % (1.0 - accuracy))
```

```
[Stage 6535:==================================================>  (120 + 5) / 125]
Accuracy= 0.13471149477510222
Test Error = 0.865289
```

In [183…

```
# Calculate other metrics such as precision, recall, and F1-score
precision = evaluator.evaluate(predictions4, {evaluator.metricName: "weightedPrecision"})
recall = evaluator.evaluate(predictions4, {evaluator.metricName: "weightedRecall"})
f1_score = evaluator.evaluate(predictions4, {evaluator.metricName: "f1"})

print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
```

```
[Stage 6583:==================================================>  (120 + 5) / 125]
Precision: 0.8243766712503708
Recall: 0.13471149477510222
F1-score: 0.15693879973702238
```

Confusion matrix of Gaussian Naive Bayes

In [186…

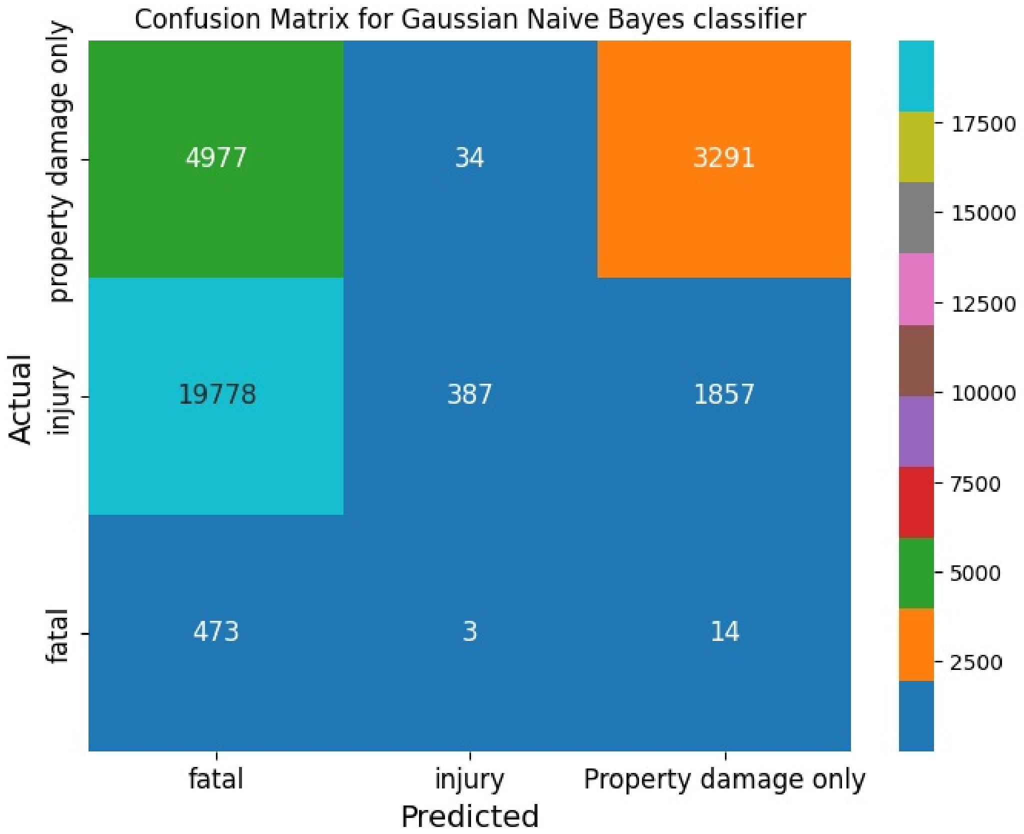
```
all_labels = ["fatal", "injury", "property damage only"]
confusion_matrix4 = predictions4.groupBy('label').pivot('predictedLabel').count().na.fill(0)
confusion_matrix4.show()
```

```
[Stage 6701:===================================================> (121 + 4) / 125]
+--------------------+-----+------+--------------------+
|               label|fatal|injury|property damage only|
+--------------------+-----+------+--------------------+
|property damage only| 4977|    34|                3291|
|              injury|19778|   387|                1857|
|               fatal|  473|     3|                  14|
+--------------------+-----+------+--------------------+
```

In [188…

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

cs_labels2 =['fatal','injury','Property damage only']
cs_labels = ["property damage only","injury", "fatal"]
# Convert PySpark DataFrame to Pandas DataFrame
confusion_matrix_pd4 = confusion_matrix4.toPandas()

# Set 'label' column as the index
confusion_matrix_pd4.set_index('label', inplace=True)

# Convert the DataFrame to a matrix
confusion_matrix_np3 = confusion_matrix_pd4.to_numpy()
plt.figure(figsize=(8, 6))
# Plot confusion matrix heatmap
ax = sns.heatmap(confusion_matrix_np3, annot=True, fmt=".0f", cmap="tab10",
                 annot_kws={"fontsize": 12})

plt.title('Confusion Matrix for Gaussian Naive Bayes classifier')
plt.xlabel('Predicted',fontsize=14)
plt.ylabel('Actual',fontsize=14)

# Rotate and align x-axis tick labels
ax.set_xticklabels(cs_labels2, rotation=0, ha='center', fontsize=12)  # Adjust the rotation angle and font size
```

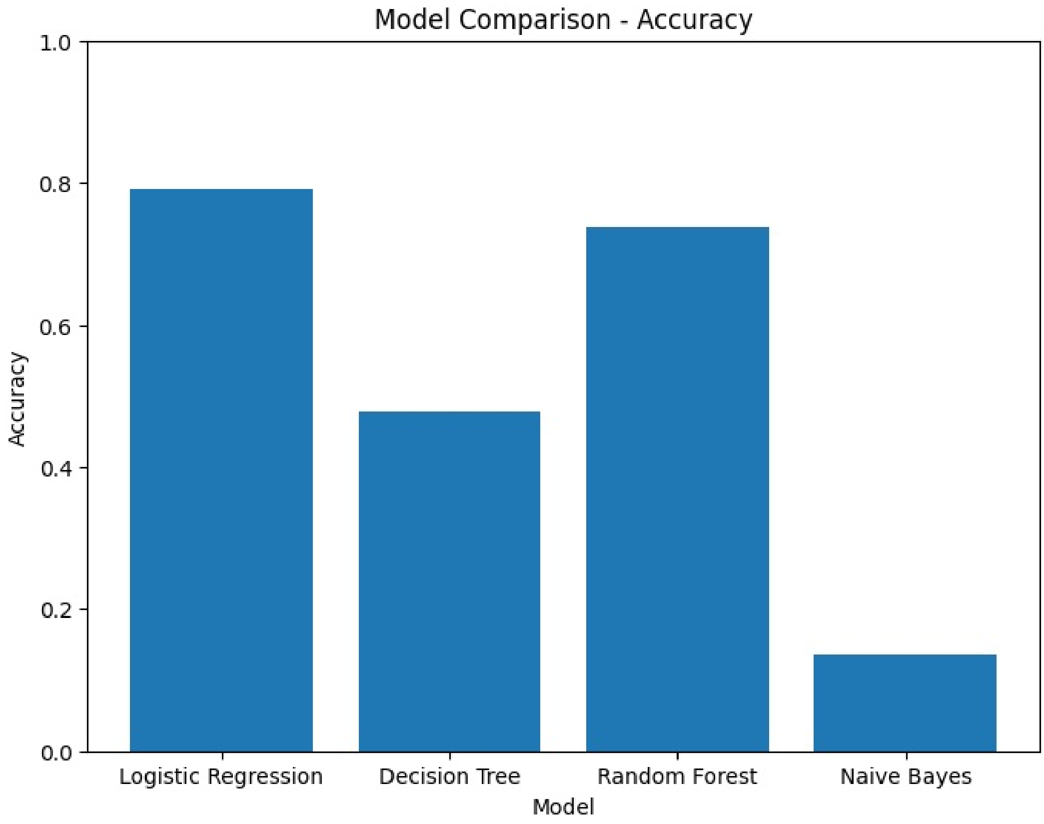
```
# Rotate and align y-axis tick labels
ax.set_yticklabels(cs_labels, rotation=90, ha='right', fontsize=12)  # Adjust the rotation angle and font size

plt.show()
```



Model Comparision and Model Selection

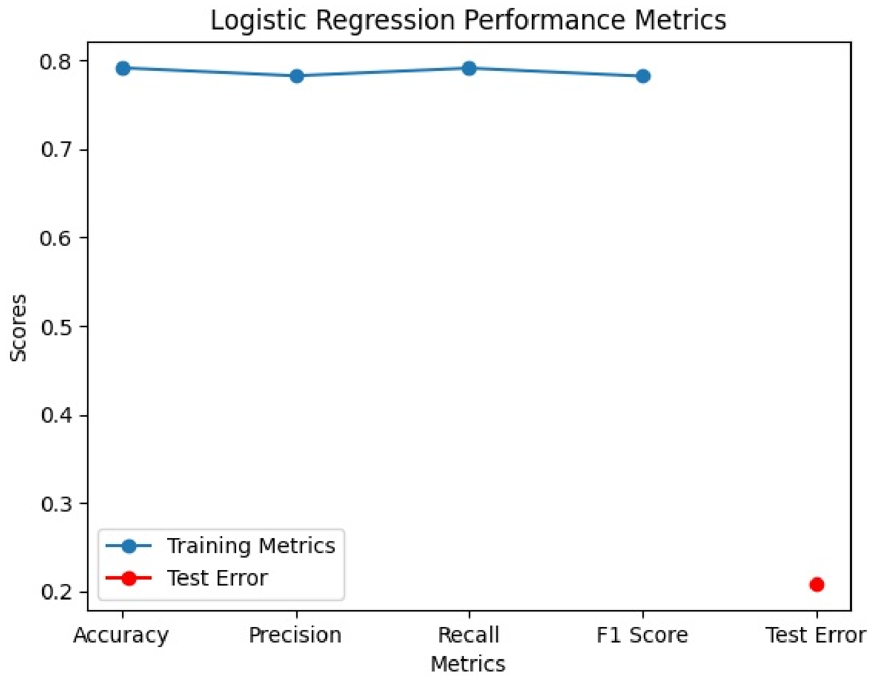
In [193…

```
import matplotlib.pyplot as plt

model_names = ["Logistic Regression","Decision Tree", "Random Forest", "Naive Bayes"]
accuracies = [0.7922,0.4780,0.73680,0.1347]

plt.figure(figsize=(8, 6))
plt.bar(model_names, accuracies)
plt.xlabel("Model")
plt.ylabel("Accuracy")
plt.title("Model Comparison - Accuracy")
plt.ylim([0, 1])
plt.show()
```



In [199…

```python
import matplotlib.pyplot as plt

# Example data
model_names = ['Accuracy', 'Precision', 'Recall', 'F1 Score', 'Test Error']
scores = [0.7915, 0.7823, 0.7912, 0.7819, 0.2084]

# Plotting the line chart
plt.plot(model_names[:-1], scores[:-1], marker='o', label='Training Metrics')
plt.plot(model_names[-1:], scores[-1:], marker='o', color='red', label='Test Error')

# Adding labels and title
plt.xlabel('Metrics')
plt.ylabel('Scores')
plt.title('Logistic Regression Performance Metrics')

# Adding a legend
plt.legend()

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



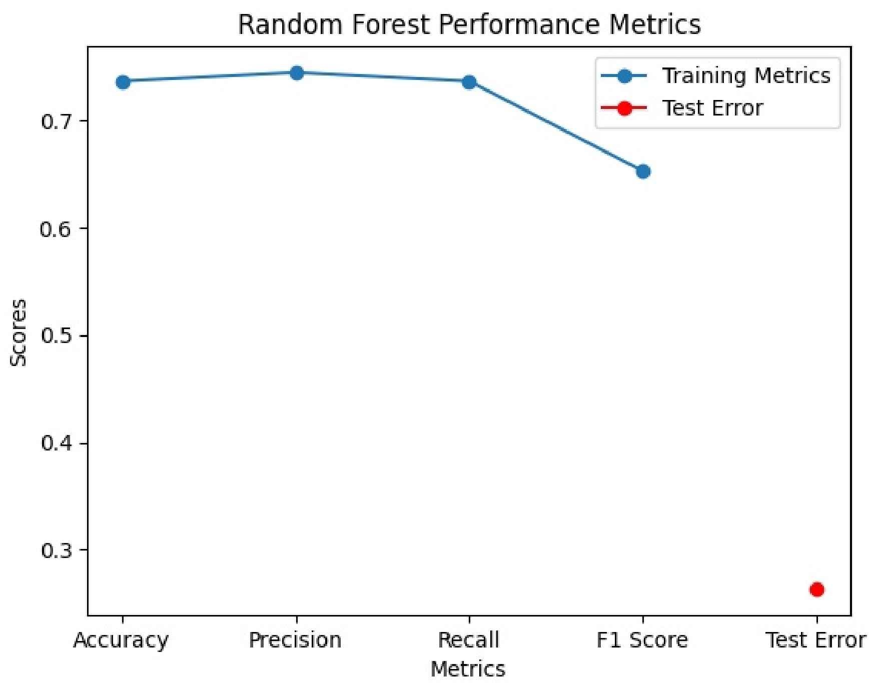
In [200…

```python
import matplotlib.pyplot as plt

# Example data
```

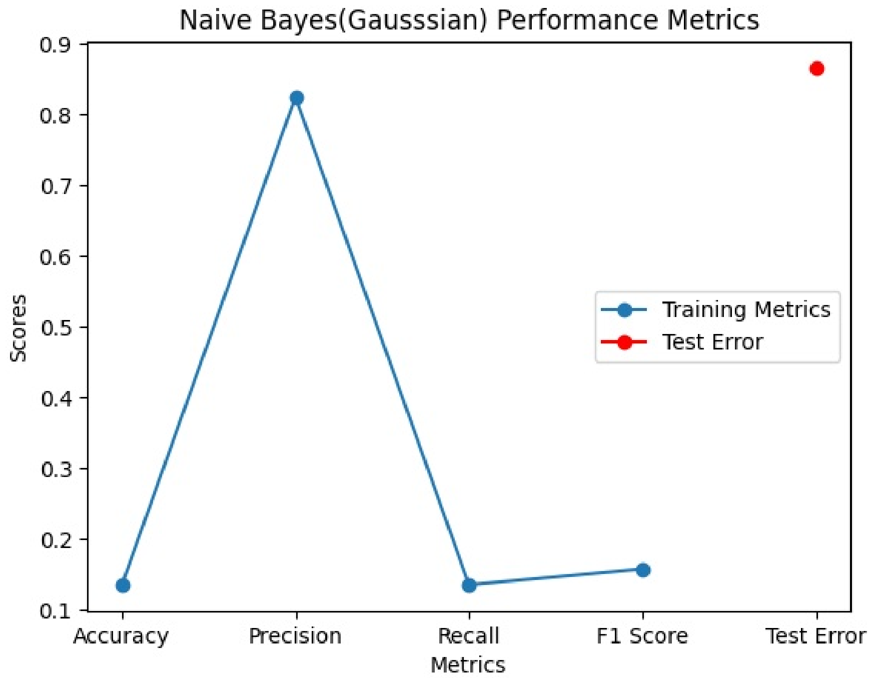
```
model_names = ['Accuracy', 'Precision', 'Recall', 'F1 Score', 'Test Error']
scores = [0.4781, 0.5478, 0.7806, 0.75012, 0.5219]

# Plotting the line chart
plt.plot(model_names[:-1], scores[:-1], marker='o', label='Training Metrics')
plt.plot(model_names[-1:], scores[-1:], marker='o', color='red', label='Test Error')

# Adding labels and title
plt.xlabel('Metrics')
plt.ylabel('Scores')
plt.title('Decision tree Performance Metrics')

# Adding a legend
plt.legend()

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

In [201…

```
import matplotlib.pyplot as plt

# Example data
model_names = ['Accuracy', 'Precision', 'Recall', 'F1 Score', 'Test Error']
scores = [0.7368,0.7450,0.7368,0.6530,0.2632]

# Plotting the line chart
plt.plot(model_names[:-1], scores[:-1], marker='o', label='Training Metrics')
plt.plot(model_names[-1:], scores[-1:], marker='o', color='red', label='Test Error')

# Adding labels and title
plt.xlabel('Metrics')
plt.ylabel('Scores')
plt.title('Random Forest Performance Metrics')

# Adding a legend
plt.legend()

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



In [202…
```python
import matplotlib.pyplot as plt

# Example data
model_names = ['Accuracy', 'Precision', 'Recall', 'F1 Score', 'Test Error']
scores = [0.1347, 0.8243, 0.1347, 0.1569, 0.8653]

# Plotting the line chart
plt.plot(model_names[:-1], scores[:-1], marker='o', label='Training Metrics')
plt.plot(model_names[-1:], scores[-1:], marker='o', color='red', label='Test Error')

# Adding labels and title
plt.xlabel('Metrics')
plt.ylabel('Scores')
plt.title('Naive Bayes(Gausssian) Performance Metrics')

# Adding a legend
plt.legend()

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



In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js