DATA SCIENCE REPORT

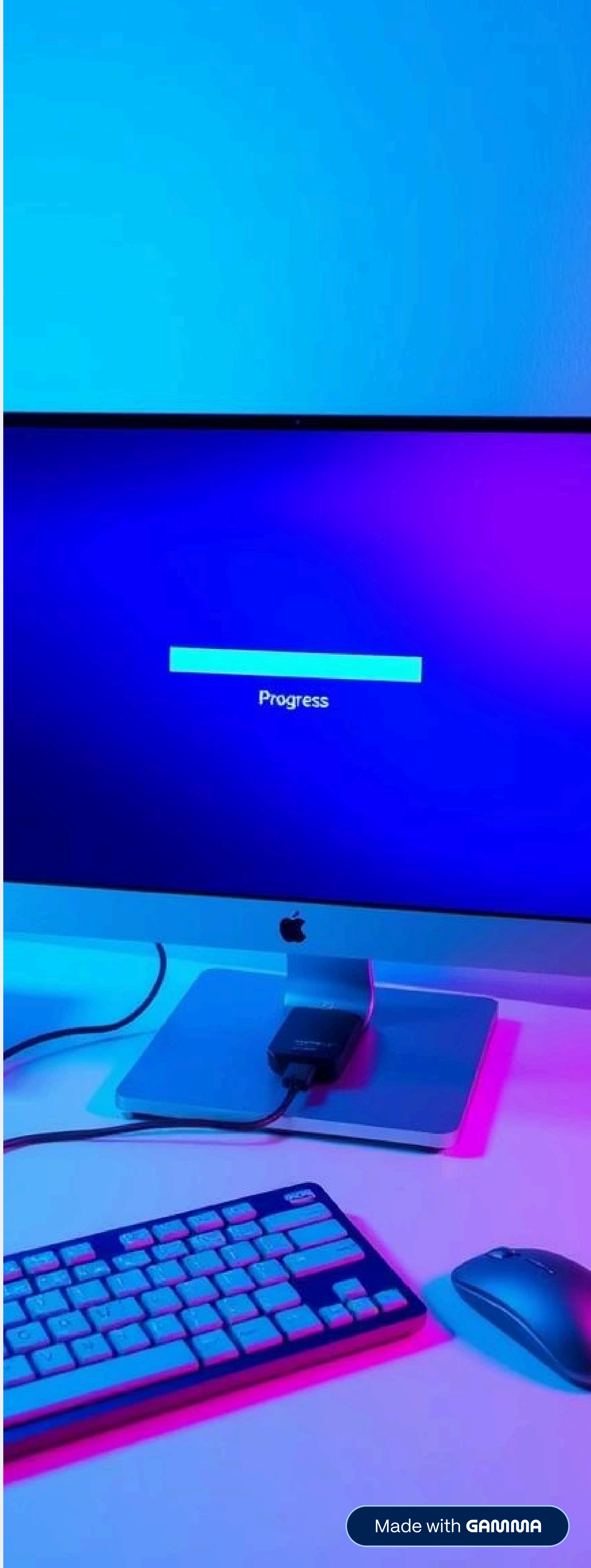
PRESENTED BY :

Yazan Aqtash

PRESENTED TO :

DR.MOHAMMAD AL-HAMMOURI

Hashemite University



Introduction

This project aims to prepare a road-accident dataset for machine learning by performing full data cleaning, preprocessing, and target selection.

Phase 1 focuses on transforming the raw CSV data into a structured, reliable dataset that can be used for predictive analytics in Phase 2.

Using Python (Pandas + Scikit-Learn), several structured steps were performed to ensure the dataset is:

- Accurate
- Free of duplicates
- Correctly typed
- Enhanced with engineered features
- Ready for machine learning in Phase 2

Dataset Overview

- **Total rows (before cleaning):** 73,095
- **Total rows (after cleaning):** 72,461
- **Total columns:** 20

Main Categories of Variables

- **Accident details:**
  `Accident Type`, `Date`, `Speed (Km)`, `Number of Cars`
- **Injury outcomes:**
  `Simple Injuries`, `Medium Injuries`, `Severe Injuries`, `Death`
- **Environmental & Road conditions:**
  `Weather`, `Light`, `Road Type`, `Road Surfuce Description`, `Road Proerties`, `Road Lanes`
- **Driver & Vehicle info:**
  `Driver Age`, `Driver Licensee Type`, `Driver Mistake`, `Sex`, `Vehicle type`, `Vehicle Country`

---

1. LOAD DATA

Purpose

The goal of this step is to load the dataset into memory and preserve a raw copy for comparison.

Code Used

```
df = pd.read_csv("Accident.csv")
df_raw = df.copy()
```

Explanation

- `df` contains the working dataset.
- `df_raw` is an untouched backup, used to examine data *before* cleaning

2. BEFORE-CLEANING SUMMARY

Purpose

Before cleaning, it is necessary to:

- Check dataset size
- Identify duplicate rows
- Check for missing values
- Understand whether certain columns (e.g., Driver Age) contain unrealistic values

Code Used

```
print("=== BEFORE CLEANING ===")
print(f"Rows: {len(df_raw)}")
print(f"Duplicate rows: {df_raw.duplicated().sum()}")

print("\nMissing values per column:")
print(df_raw.isna().sum())

if "Driver Age" in df_raw.columns:
    print("\n'Driver Age' stats (raw):")
    print(df_raw["Driver Age"].describe())
```

Explanation

- `df_raw.duplicated().sum()` identifies how many duplicate rows exist.
- `df_raw.isna().sum()` checks for missing values in each column.
- The statistical description of `"Driver Age"` helps detect anomalies such as ages below 10 or above 100.

3. DATA CLEANING

The cleaning workflow consists of several essential operations.

3.1 Removing Duplicate Rows

Code Used

```
df = df.drop_duplicates()
```

Explanation

Duplicate rows distort model learning by overrepresenting certain patterns.
This step ensures the dataset contains unique entries only.

3.2 Date Processing

Code Used

```
df["Date"] = pd.to_datetime(df["Date"], errors="coerce")
df["Year"] = df["Date"].dt.year
df["Month"] = df["Date"].dt.month
df["DayOfWeek"] = df["Date"].dt.dayofweek
df["Hour"] = df["Date"].dt.hour
```

Explanation

- Converts all date formats (e.g., `14-Jan-18`, `2018-01-14`) into a standardized `datetime` format.
- Invalid dates become `NaT` due to `errors="coerce"`.
- Extracted features (`Year`, `Month`, `DayOfWeek`, `Hour`) provide meaningful temporal information for modeling:
  - **Year**
  - **Month**: seasonal trends
  - **DayOfWeek**: weekday vs weekend accidents
  - **Hour**: rush hour patterns

3.3 Cleaning Driver Age

Step 1: Identify invalid ages

code used

```
df["Driver Age Clean"] = df["Driver Age"].where(
    (df["Driver Age"] >= 16) & (df["Driver Age"] <= 90)
)
```

- Ages **below 16** or **above 90** were marked invalid
- These values were replaced with `NaN`

Step 2: Impute missing ages

code used

```
median_age = df["Driver Age Clean"].median()
df["Driver Age Clean"] = df["Driver Age Clean"].fillna(median_age)
```

- Missing/invalid ages were filled using the **median age (34)**
- This preserves the natural distribution without distortion

3.4 Creating the Target Variable

Code Used

```
df["Injury_or_Death"] = (
    (df["Simple Injuries"] > 0) |
    (df["Medium Injuries"] > 0) |
    (df["Severe Injuries"] > 0) |
    (df["Death"] > 0)
).astype(int)
```

Explanation

The dataset originally contained **exact counts of injuries and deaths**.
This step converts those multiple outcome columns into a **single binary classification target**:

- **1 = Accident resulted in at least one injury or death**
- **0 = Accident resulted in no injuries and no deaths**

This simplifies the predictive modeling process.

3.5 Dropping Leakage and Replaced Columns

Code Used

```
drop_cols = [
    "Simple Injuries", "Medium Injuries", "Severe Injuries",
    "Death", "Date", "Driver Age"
]
df = df.drop(columns=drop_cols)
```

Explanation

Removed columns:

- Simple Injuries
- Medium Injuries
- Severe Injuries
- Death
- Date (replaced by Year/Month/etc.)
- Driver Age (replaced with Driver Age Clean)

Columns like `"Simple Injuries"` or `"Severe Injuries"` occur *after* the accident and would create **data leakage** if used for prediction.

4. AFTER-CLEANING SUMMARY

Code Used

```
print("\n=== AFTER CLEANING ===")
print(f"Rows: {len(df)}")

print("\nMissing values per column:")
print(df.isna().sum())

print("\n'Driver Age Clean' stats (cleaned):")
print(df["Driver Age Clean"].describe())

print("\nTarget 'Injury_or_Death' distribution:")
print(df["Injury_or_Death"].value_counts(normalize=True).rename("proportion"))
```

Explanation

This step verifies:

- Final dataset size
- Remaining missing data (should be none)
- Cleaned age distribution
- How imbalanced the `Injury_or_Death` target is (fatal accidents are very rare)

---

5. SAVE CLEANED DATA

Code Used

```
df.to_csv("Accident_cleaned.csv", index=False)
```

Explanation

The cleaned dataset is exported for future use in modeling.

6. FEATURE AND TARGET SEPARATION

Code Used

```
X = df.drop(columns=["Injury_or_Death"])
y = df["Injury_or_Death"]
```

Explanation

- **X** contains all predictor (feature) variables
- **y** contains the binary target variable (**Injury_or_Death**)
- The target equals **1** if an accident resulted in **any injury or death**, and **0** otherwise
- This formulation avoids extreme class imbalance caused by rare fatal accidents and enables meaningful model learning

Due to the very low number of fatal accidents, injury and death outcomes were merged into a single severity-based target variable.

6.1 Identifying Numerical and Categorical Columns

Code Used

```
num_cols = X.select_dtypes(include=["int64", "float64"]).columns.tolist()
cat_cols = X.select_dtypes(include=["object"]).columns.tolist()
```

Explanation

Scikit-learn preprocessing requires knowing which columns are numeric vs categorical:

- Numeric → StandardScaler
- Categorical → OneHotEncoder

7. PREPROCESSOR PIPELINE

Code Used

```
preprocess = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), num_cols),
        ("cat", OneHotEncoder(handle_unknown="ignore"), cat_cols),
    ]
)
```

Explanation

This pipeline does two things:

- **Scale** all numerical features (centers and normalizes values)
- **One-hot encode** all categorical features into numeric binary vectors

Using `handle_unknown='ignore'` ensures unseen categories at prediction time do not break the model.

---

8. TRAIN / TEST SPLIT

Code Used

```
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=42,
    stratify=y
)
```

Explanation

- Split is **80% training** / **20% testing**
- `stratify=y` ensures both sets keep the same proportion of fatal vs non-fatal accidents
- `random_state=42` ensures reproducibility

9. OPTIONAL PREPROCESSOR TRANSFORMATION

Code Used

```
X_train_trans = preprocess.fit_transform(X_train)

X_test_trans = preprocess.transform(X_test)
```

Explanation

If uncommented, this would transform training and testing data according to the preprocessing pipeline. This is typically done right before training machine learning models in Phase 2

2. BEFORE-CLEANING OUTPUT

```
=== BEFORE CLEANING ===
Rows: 73095
Duplicate rows: 634

Missing values per column:
Accident Type               0
Date                        0
Speed (Km)                  0
Simple Injuries             0
Severe Injuries             0
Death                       0
Medium Injuries             0
Road Lanes                  0
Road Surface Description    0
Vehicle Country             0
Driver Licensee Type        0
Road Type                   0
Light                       0
Weather                     0
Road Proerties              0
Driver Mistake              0
Vehicle type                0
Number of Cars              0
Driver Age                  0
Sex                         0
dtype: int64
```

```
'Driver Age' stats (raw):
count    73095.000000
mean        35.497353
std         23.094740
min          0.000000
25%         24.000000
50%         33.000000
75%         44.000000
max        118.000000
Name: Driver Age, dtype: float64
```

AFTER-CLEANING OUTPUT

```
=== AFTER CLEANING ===
Rows: 72461

Missing values per column:
Accident Type                0
Speed (Km)                   0
Road Lanes                   0
Road Surface Description     0
Vehicle Country              0
Driver Licensee Type         0
Road Type                    0
Light                        0
Weather                      0
Road Proerties               0
Driver Mistake               0
Vehicle type                 0
Number of Cars               0
Sex                          0
Year                        94
Month                       94
DayOfWeek                   94
Hour                        94
Driver Age Clean             0
Injury_or_Death              0
dtype: int64
```

```
'Driver Age Clean' stats (cleaned):
count    72461.000000
mean        36.132485
std         11.283246
min         18.000000
25%         28.000000
50%         34.000000
75%         42.000000
max         90.000000
Name: Driver Age Clean, dtype: float64
```

```
Target 'Injury_or_Death' distribution:
Injury_or_Death
0    0.946233
1    0.053767
Name: proportion, dtype: float64

Cleaned data saved to 'Accident_cleaned.csv'

Numeric columns: ['Speed (Km)', 'Number of Cars', 'Sex', 'Year', 'Month', 'DayOfWeek', 'Hour', 'Driver Age Clean']
Categorical columns: ['Accident Type', 'Road Lanes', 'Road Surfuce Description', 'Vehicle Country', 'Driver Licensee Type', 'Road Type', 'Light', 'Weather'

=== DATA SPLIT SHAPES ===
X_train: (57968, 19)
X_test:  (14493, 19)
y_train: (57968,)
y_test:  (14493,)

Process finished with exit code 0
```

10. Conclusion

- Dataset loaded and examined
- Removed duplicates
- Mixed-format dates standardized and decomposed
- Driver ages cleaned and invalid entries corrected
- Binary target variable created (`Has_Death`)
- Removed leakage columns
- Classified columns into numerical and categorical
- Scikit-learn preprocessing pipeline constructed
- Train/test split executed
- Cleaned dataset exported