Truck driver Crash Prevention

Full Name: Wanjiru Randolph

Report Submission Date: November 1, 2024

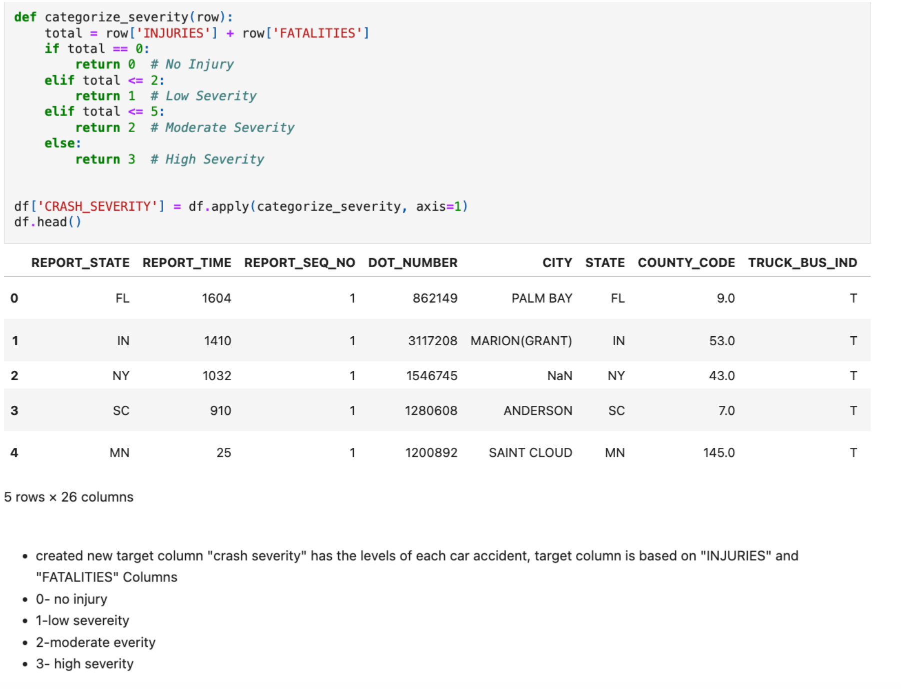
1. Introduction

In this project, I am analyzing crash data to uncover key factors that contribute to crash severity through a structured approach of data exploration, feature engineering, and machine learning. After thoroughly cleaning the data and addressing formatting inconsistencies, I derived new features, including a target variable "crash severity" based on existing fatality and injury metrics. Initial visualizations revealed an imbalance in the dataset's severe outcomes, which I plan to address in my machine learning models. To build robust predictive insights, I encoded categorical features, created targeted visualizations, and implemented a Random Forest model alongside SMOTE to manage class imbalances. As I move forward, I am exploring additional modeling techniques and incorporating class weights to further improve model accuracy and reliability. Through this comprehensive analysis, my goal is to better understand and predict crash severity factors for more effective data-driven insights.

2. Summary

Summary of Work Done: I have been finishing up my EDA and preparing my data for machine learning. I used feature engineering to create new columns such as my target column crash severity based off my fatal and injury columns. I also fixed formatting issues and separated the dates column with year, month and day. I fixed any formatting issues with other columns as well. After cleaning my data, I began visualizing it. I created a pie chart for my target variables and saw there was an imbalance between my more severe targets, I plan to fix this in machine learning. I created graphs for my numerical and categorical features against the target variable "crash severity". I also created heatmaps for mu numerical columns and found that the number of vehicles in the crash had the strongest correlation with I also encoded my categorical features, with label encoding. I ran a random forest model and use smote for oversampling I plan to use more models and weights to improve the outcomes

Graphs and Visualizations:

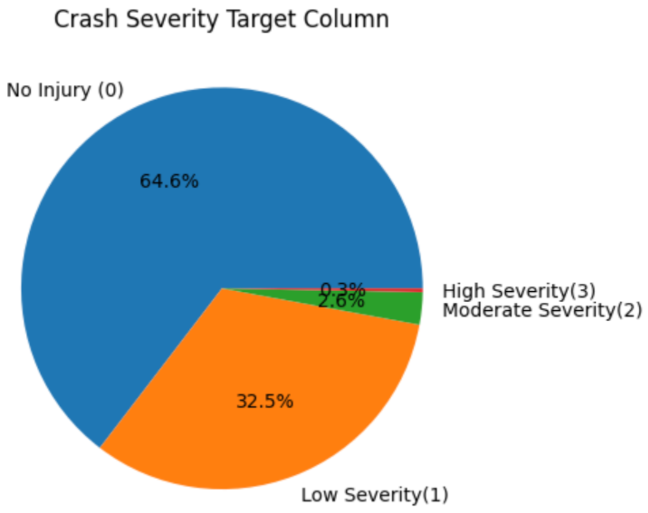


```python
def categorize_severity(row):
    total = row['INJURIES'] + row['FATALITIES']
    if total == 0:
        return 0  # No Injury
    elif total <= 2:
        return 1  # Low Severity
    elif total <= 5:
        return 2  # Moderate Severity
    else:
        return 3  # High Severity


df['CRASH_SEVERITY'] = df.apply(categorize_severity, axis=1)
df.head()
```

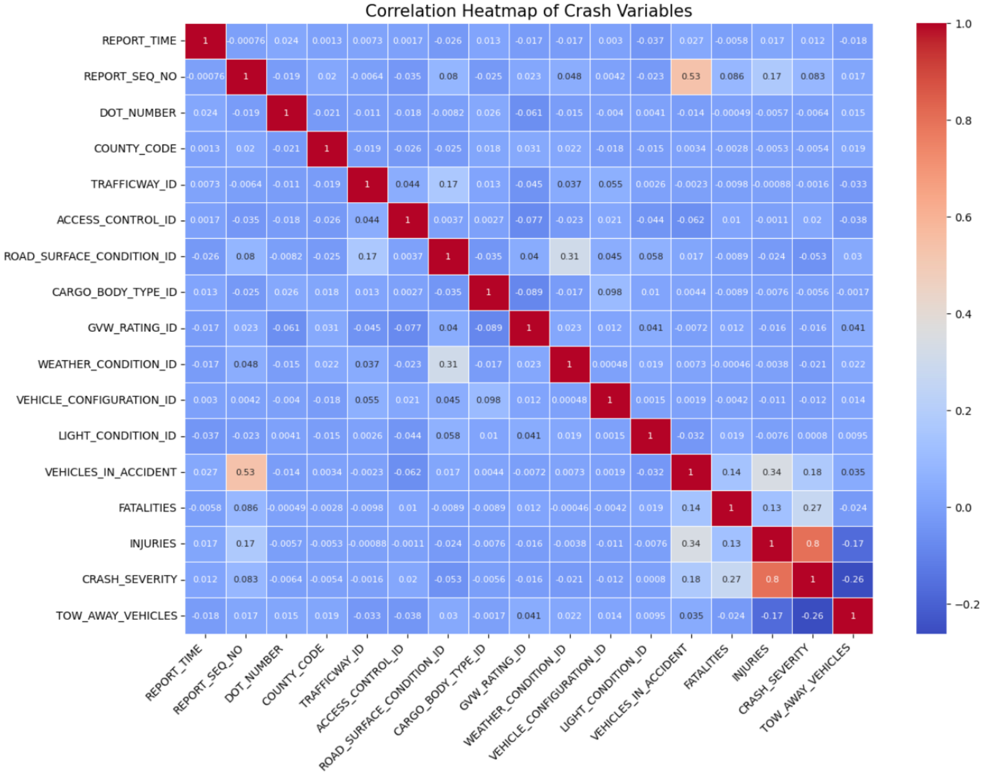
	REPORT_STATE	REPORT_TIME	REPORT_SEQ_NO	DOT_NUMBER	CITY	STATE	COUNTY_CODE	TRUCK_BUS_IND
0	FL	1604	1	862149	PALM BAY	FL	9.0	T
1	IN	1410	1	3117208	MARION(GRANT)	IN	53.0	T
2	NY	1032	1	1546745	NaN	NY	43.0	T
3	SC	910	1	1280608	ANDERSON	SC	7.0	T
4	MN	25	1	1200892	SAINT CLOUD	MN	145.0	T

5 rows × 26 columns

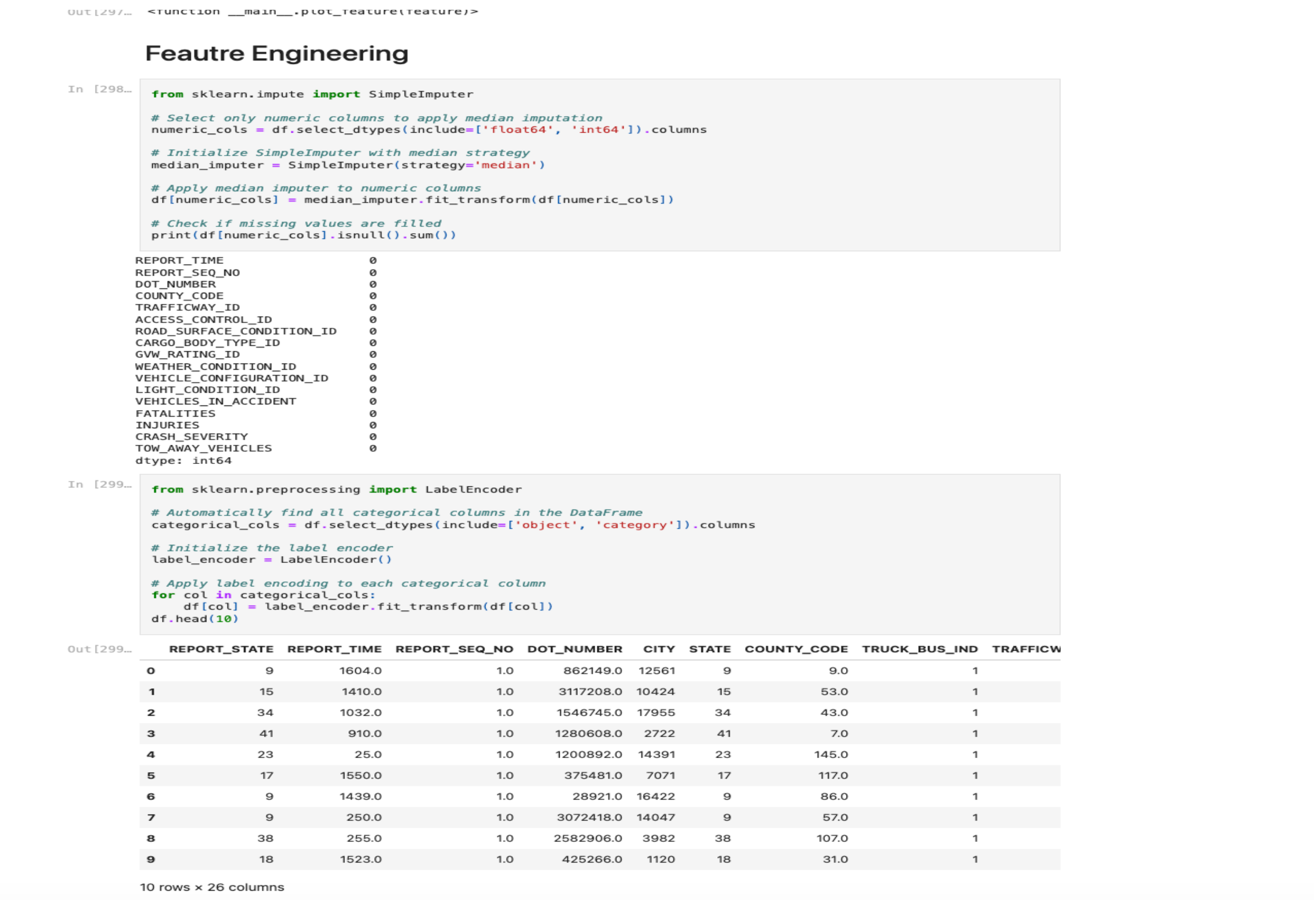
- created new target column "crash severity" has the levels of each car accident, target column is based on "INJURIES" and "FATALITIES" Columns
- 0- no injury
- 1-low severeity
- 2-moderate everity
- 3- high severity



this is a severely imbalanced target column. no injury dominates at 65 percent, low severity has about 33 %, while moderate severity is 2.6 % and high severity as a very small percentage of 2.3 percent



Correlation Heatmap of Crash Variables



```
Out[297…  <function __main__.plot_feature(feature)>
```

Feautre Engineering

In [298…

```
from sklearn.impute import SimpleImputer

# Select only numeric columns to apply median imputation
numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns

# Initialize SimpleImputer with median strategy
median_imputer = SimpleImputer(strategy='median')

# Apply median imputer to numeric columns
df[numeric_cols] = median_imputer.fit_transform(df[numeric_cols])

# Check if missing values are filled
print(df[numeric_cols].isnull().sum())
```

```
REPORT_TIME                  0
REPORT_SEQ_NO                0
DOT_NUMBER                   0
COUNTY_CODE                  0
TRAFFICWAY_ID                0
ACCESS_CONTROL_ID            0
ROAD_SURFACE_CONDITION_ID    0
CARGO_BODY_TYPE_ID           0
GVW_RATING_ID                0
WEATHER_CONDITION_ID         0
VEHICLE_CONFIGURATION_ID     0
LIGHT_CONDITION_ID           0
VEHICLES_IN_ACCIDENT         0
FATALITIES                   0
INJURIES                     0
CRASH_SEVERITY               0
TOW_AWAY_VEHICLES            0
dtype: int64
```

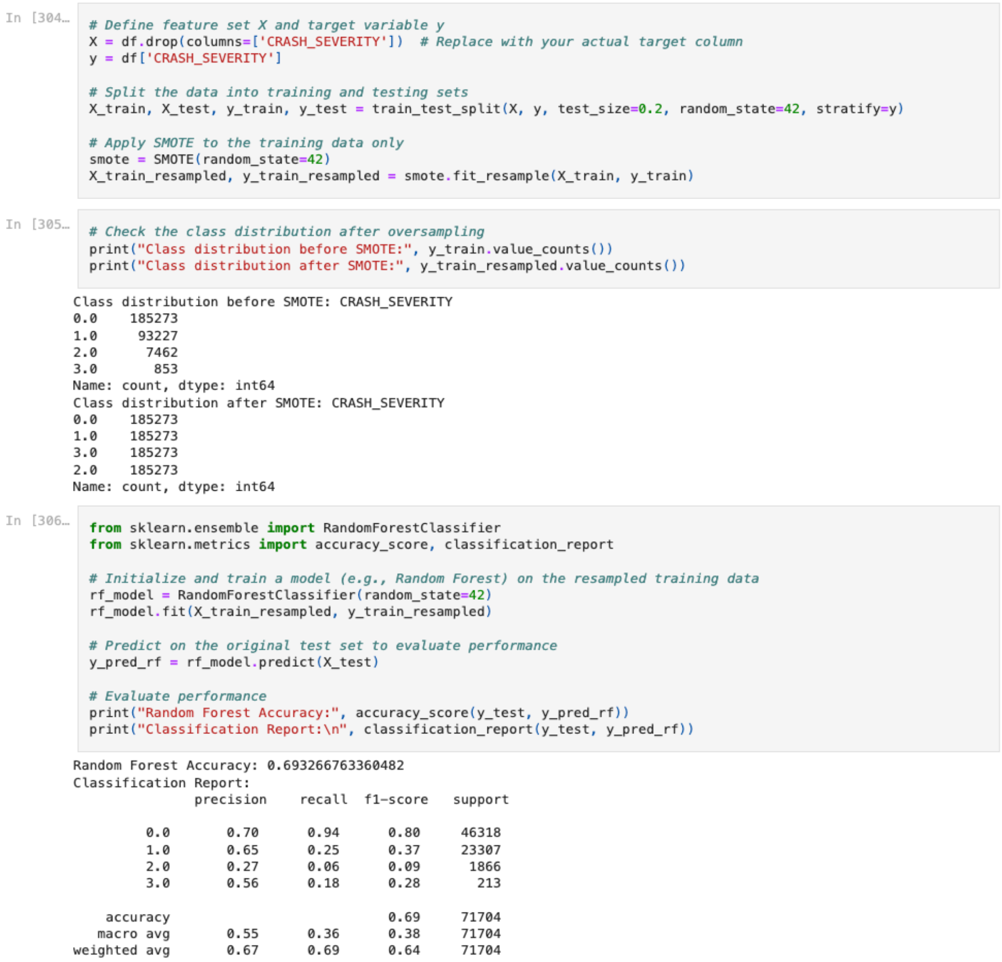
In [299…

```
from sklearn.preprocessing import LabelEncoder

# Automatically find all categorical columns in the DataFrame
categorical_cols = df.select_dtypes(include=['object', 'category']).columns

# Initialize the label encoder
label_encoder = LabelEncoder()

# Apply label encoding to each categorical column
for col in categorical_cols:
    df[col] = label_encoder.fit_transform(df[col])
df.head(10)
```

Out[299…

	REPORT_STATE	REPORT_TIME	REPORT_SEQ_NO	DOT_NUMBER	CITY	STATE	COUNTY_CODE	TRUCK_BUS_IND	TRAFFICW
0	9	1604.0	1.0	862149.0	12561	9	9.0	1	
1	15	1410.0	1.0	3117208.0	10424	15	53.0	1	
2	34	1032.0	1.0	1546745.0	17955	34	43.0	1	
3	41	910.0	1.0	1280608.0	2722	41	7.0	1	
4	23	25.0	1.0	1200892.0	14391	23	145.0	1	
5	17	1550.0	1.0	375481.0	7071	17	117.0	1	
6	9	1439.0	1.0	28921.0	16422	9	86.0	1	
7	9	250.0	1.0	3072418.0	14047	9	57.0	1	
8	38	255.0	1.0	2582906.0	3982	38	107.0	1	
9	18	1523.0	1.0	425266.0	1120	18	31.0	1	

10 rows × 26 columns



In [304…

```
# Define feature set X and target variable y
X = df.drop(columns=['CRASH_SEVERITY'])  # Replace with your actual target column
y = df['CRASH_SEVERITY']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Apply SMOTE to the training data only
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

In [305…

```
# Check the class distribution after oversampling
print("Class distribution before SMOTE:", y_train.value_counts())
print("Class distribution after SMOTE:", y_train_resampled.value_counts())
```

```
Class distribution before SMOTE: CRASH_SEVERITY
0.0    185273
1.0     93227
2.0      7462
3.0       853
Name: count, dtype: int64
Class distribution after SMOTE: CRASH_SEVERITY
0.0    185273
1.0    185273
3.0    185273
2.0    185273
Name: count, dtype: int64
```

In [306…

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Initialize and train a model (e.g., Random Forest) on the resampled training data
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train_resampled, y_train_resampled)

# Predict on the original test set to evaluate performance
y_pred_rf = rf_model.predict(X_test)

# Evaluate performance
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Classification Report:\n", classification_report(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 0.693266763360482
Classification Report:
               precision    recall  f1-score   support

         0.0       0.70      0.94      0.80     46318
         1.0       0.65      0.25      0.37     23307
         2.0       0.27      0.06      0.09      1866
         3.0       0.56      0.18      0.28       213

    accuracy                           0.69     71704
   macro avg       0.55      0.36      0.38     71704
weighted avg       0.67      0.69      0.64     71704
```

3. Progress and Milestones

- **Completed Tasks:** data cleaning, EDA, graphs and visuals, feature engineering, random forest model
- **Pending Tasks: XG Boost model, SV machine, balance w/ weights, try different sampling techniques to oversample the imbalanced data**

4. Problem-Solving and Challenges

- **Challenges Encountered:** Smote technique seemed to not work so well, since the model did not train well for the more severe targets in crash severity. Will try other sampling techniques and balanced the data with weights. If the various models I use don't train well still, I'll turn the target column from multi class to binary and see if it can just predict if a crash occurs or not rather than the severity level.
- **Approaches and Solutions: various oversampling techniques, weights for balancing, and various models**
- **Impact of Solutions:** comparing the differences in how the model train, will let me know which techniques to use.
- **Next Two Weeks Goals:** Run more machine learning models, use various over sampling techniques and weights.