Student Names:

**Sandeep Neupane 301400662**

**Diego Armando Sarmiento 301379640**

**Ujjwal Poudel 301284284**

### Pre-processing - data retrieval and preparation

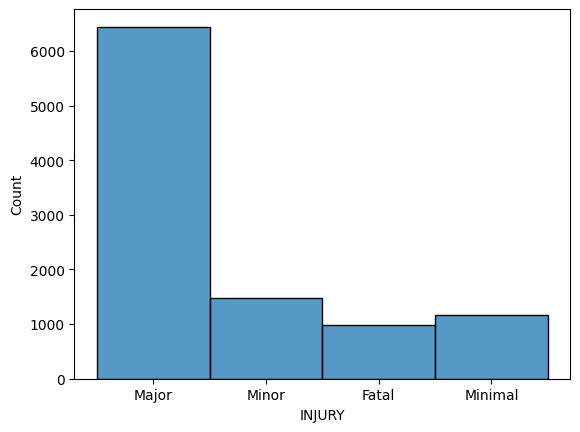
#### Loading the data

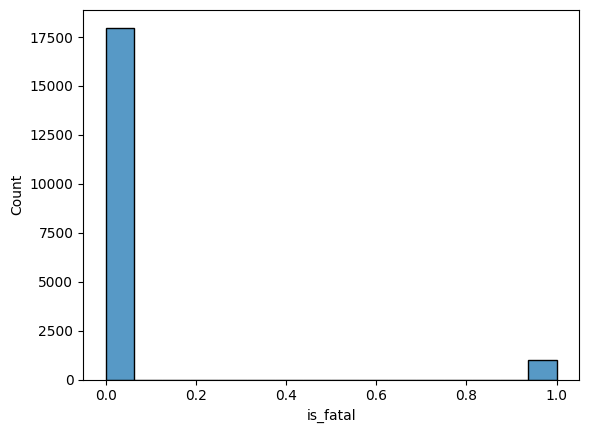
The dataset named as KSI dataset is downloaded from the [Public Safety Data Portal](https://data.torontopolice.on.ca/datasets/0a1ee9d9436546dcbdc0ee9301e45e83_0/explore) hosted by Toronto Police Service. [The documentation](https://data.torontopolice.on.ca/documents/a12c7438592a47ceb5f656d45a0fca70/explore) for this dataset is also downloaded in order to get a clear understanding of each of the columns.

After pulling the dataset, it was loaded using Pandas dataframe and stored as a data frame variable. All further exploration operations are carried out in Python using this dataframe.

Additionally, the dataset csv file is also loaded in Power BI in order to generate interactive visualization. These will help us understand the high level structure of data and drill down into specific patterns.

#### Exploring the columns

Our dataset consists of 18957 records with 54 columns. We used the documentation and decided to use the column INJURY as the target column as this is the closest field to our project goal. For this column, our data distribution is as follows:  
  


However, our project goal is only to detect whether a given set of observation would result in fatality or not. So, we decided to add another column IS\_FATAL by transforming this INJURY column into two values, 0 for not fatal and 1 for fatal. Thus, data distribution of this IS\_FATAL looks like this:  


We can see that there is a high imbalance of value in our target class. This will be fixed in our next step by using oversampling technique.

Below are the columns we have selected for training our model. The justification for selecting or not selecting a column is also provided.

##### Selected Columns

• **DATE**: The date of the accident can reveal temporal patterns, such as seasonality and annual trends.

• **TIME**: The time of the accident is important for analyzing whether fatal accidents occur more frequently at certain times of the day, such as at night or during peak hours.

• **STREET1** and **STREET2**: Street names allow for identifying specific locations where fatal accidents may occur.

• **ROAD\_CLASS**: The class of the road can influence the severity of accidents, as some classes of roads may be more dangerous than others.

• **DISTRICT**: The district helps in conducting geospatial analysis and planning specific safety measures for different areas.

• **LATITUDE** and **LONGITUDE**: Geographic coordinates enable precise mapping and geospatial analysis of accidents.

• **TRAFFCTL**: Traffic control can significantly impact the severity of accidents.

• **VISIBILITY**: Visibility can influence the likelihood of fatal accidents, especially in low visibility conditions.

• **LIGHT**: Light conditions are important for analyzing accident severity.

• **RDSFCOND**: Road conditions can affect the likelihood of fatal accidents.

• **IMPACTYPE**: The type of impact can determine the accident's severity.

• **INVTYPE**: The type of involved party (driver, pedestrian, cyclist, etc.) is crucial for understanding who is at higher risk.

• **INJURY**: Although this is a column we want to predict indirectly, it provides additional information about the severity of the accident.

• **FATAL\_NO**: This is the target column that indicates the number of fatalities in the accident.

##### Excluded Columns

• **INVAGE**: The age of the involved parties may be relevant but was excluded in this initial analysis to simplify the dataset.

• **OFFSET**: This column had many missing values.

• **INITDIR**: Similarly to the Offset column.

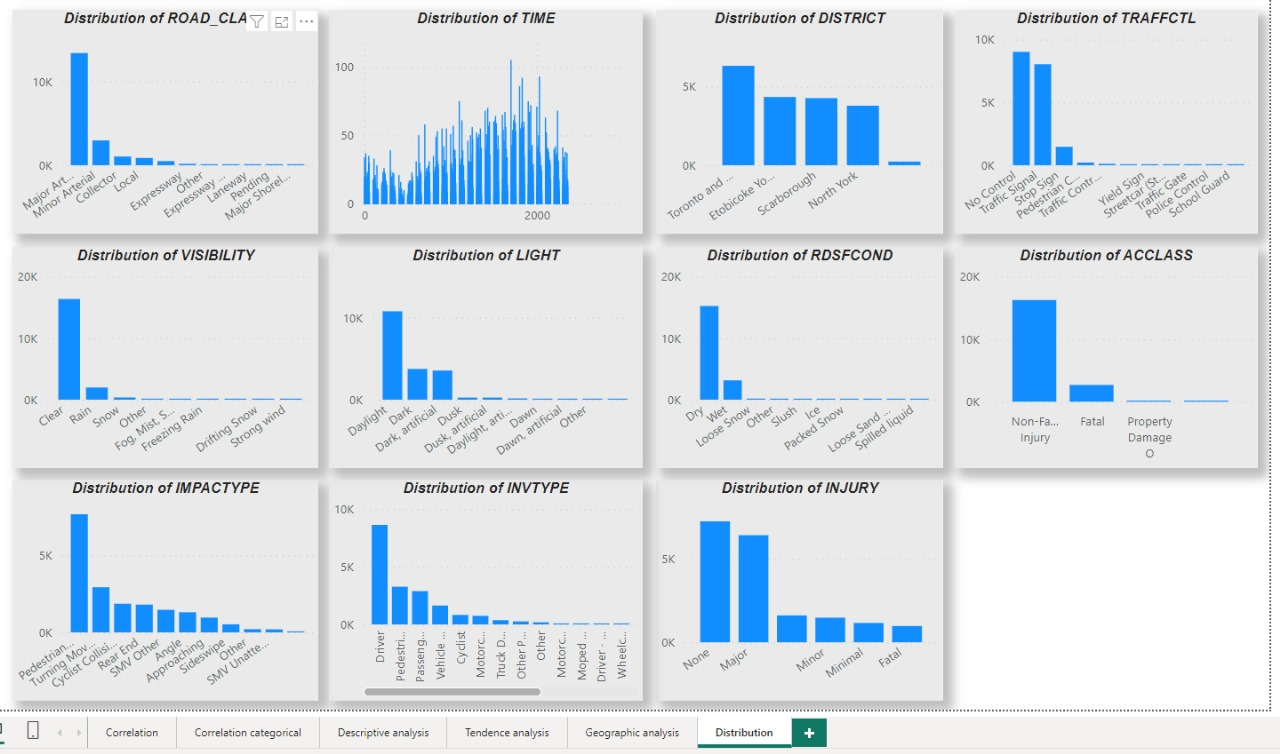
• **VEHTYPE**: Vehicle type has its importance, but was excluded in this initial analysis to simplify the dataset.

• **MANOEUVER, DRIVACT, DRIVCOND, PEDTYPE, PEDACT, PEDCOND, CYCLISTYPE, CYCACT, CYCCOND, PEDESTRIAN, CYCLIST**: These columns were worked like binary 1 or 0, but we identified that we could get information from TRAFFCTL, VISIBILITY, INVTYPE columns.

• **HOOD\_, NEIGHBOURHOOD\_, DIVISION**: Redundant geographic information, as we are already using DISTRICT, LATITUDE, and LONGITUDE.

• **ACCLASS**: The accident class provides information about the type of collision, but is redundant with INJURY and FATAL\_NO.

Here is a distribution of some of the columns of our dataset.

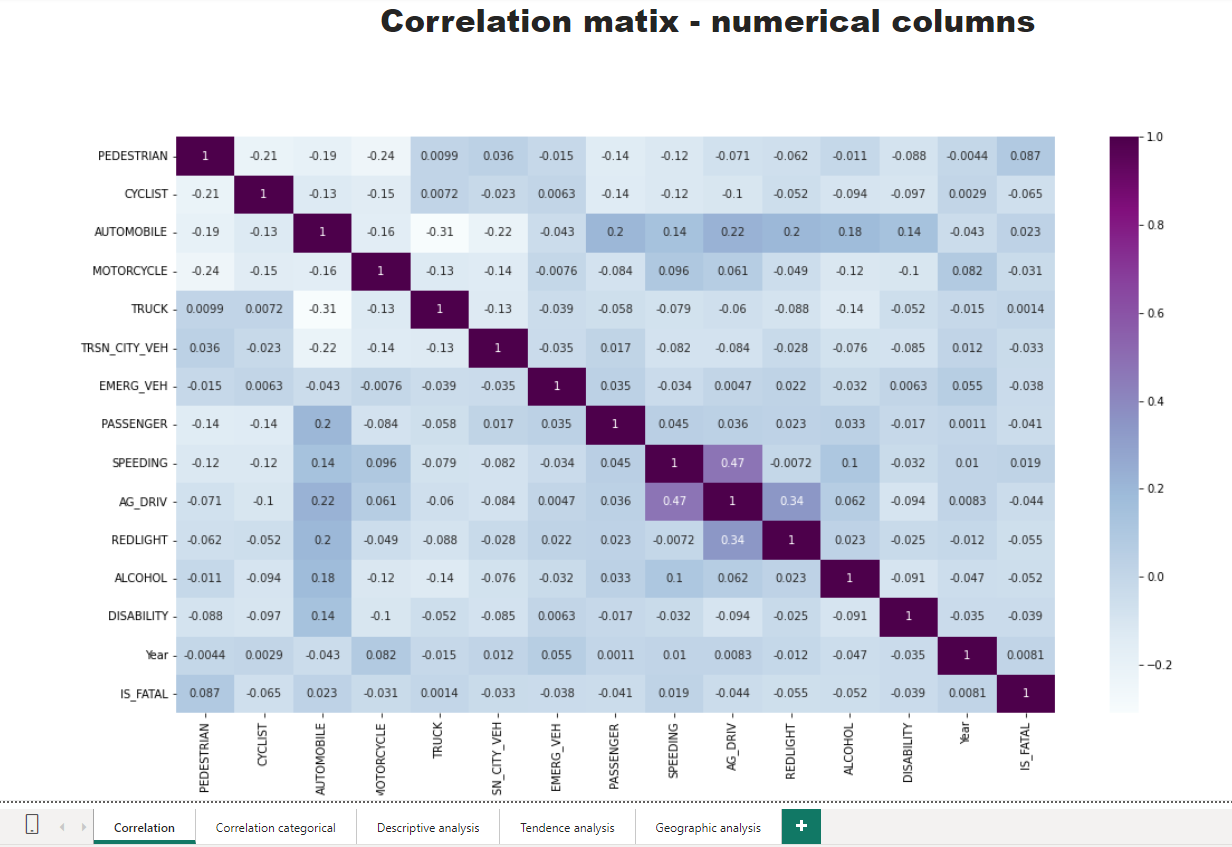


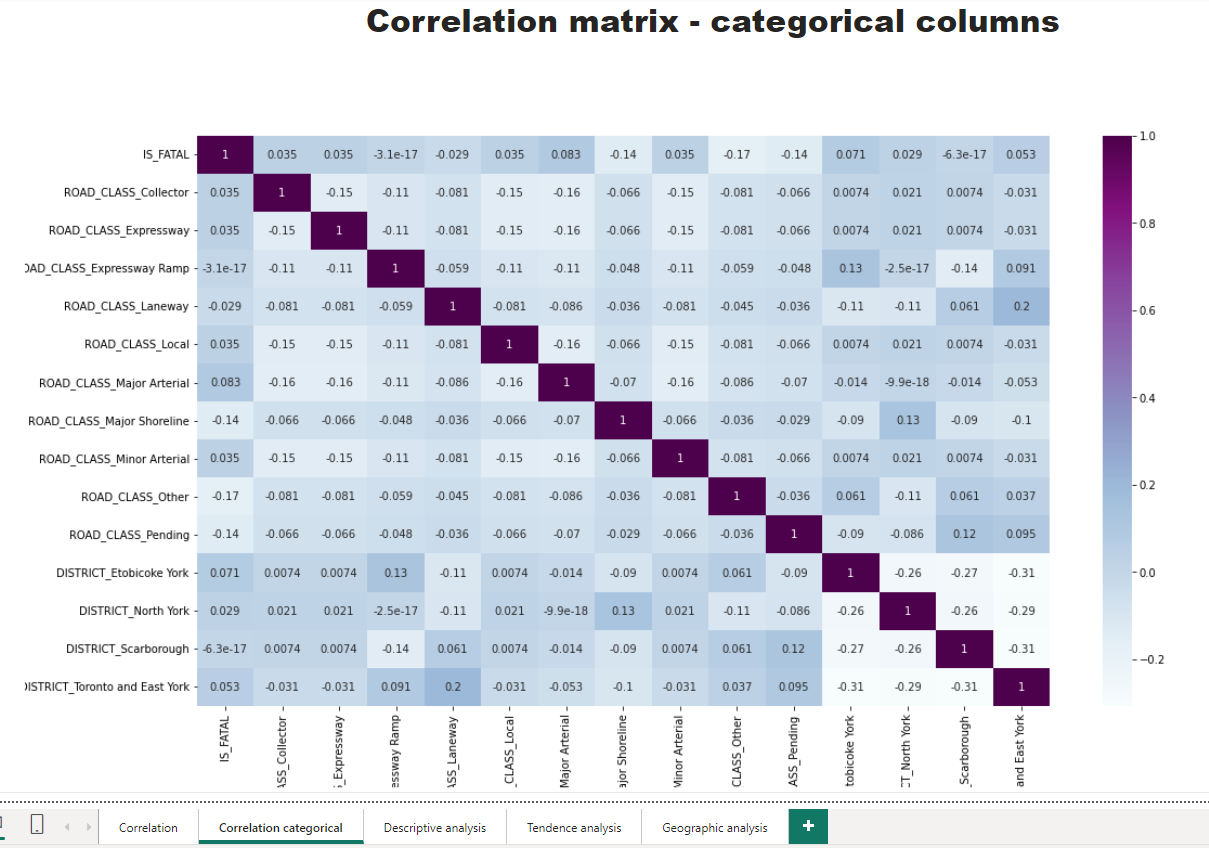
### Visualizing the dataset

For visually representing the data, a combination of PowerBI as well as seaborn+matplotlib library were used. Since the dataset has large columns, we separated the columns as numeric and non-numerical columns. This will also help us in preprocessing the data by handling each type separately.

#### Correlation between columns

Below is the correlation matrix observed for our dataset across numerical and categorical columns.

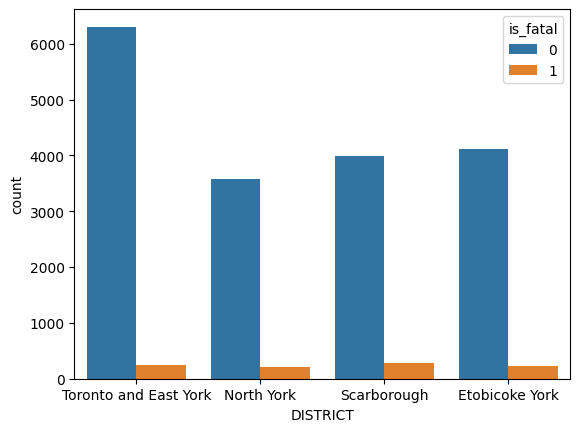




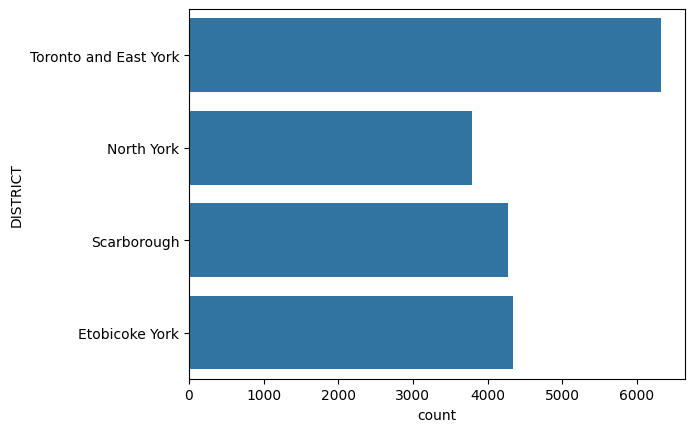
From both of our dataset, we can see very less correlation between the different columns that we selected. The most significant correlation coefficient among features was between AG\_DRIVING and SPEEDING which is 0.47. Our target column IS\_FATAL has the highest correlation with A55\_Pending and Major Shoreline at -0.14, which is not that significant.

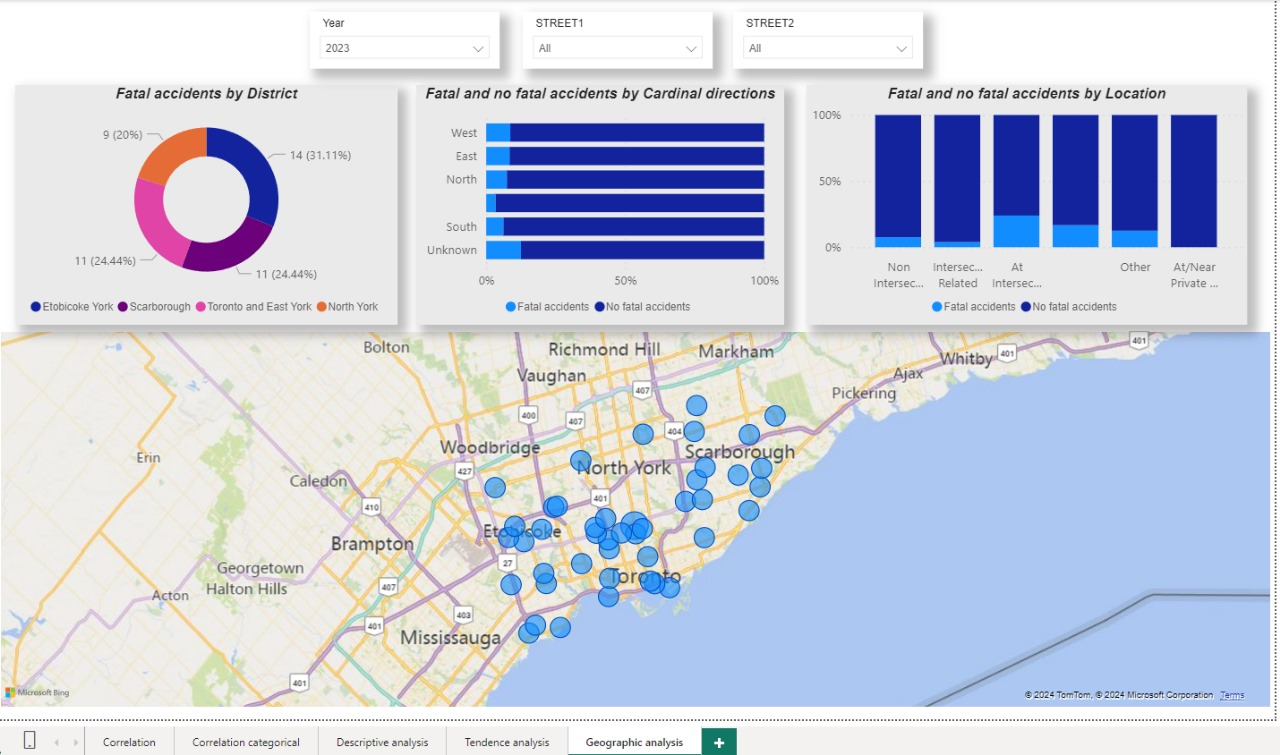
#### Analyzing fatal accidents data by location

It was interesting to note that across four of the available districts, Toronto and East York had the almost same number of fatal accidents compared to the other districts North York, Scarborough and Etobicoke+York; while the number of non-fatal observations is significantly higher for Toronto and East York.



We can also look at the data distribution of column DISTRICTS which shows more observations are recorded for Toronto and East York.

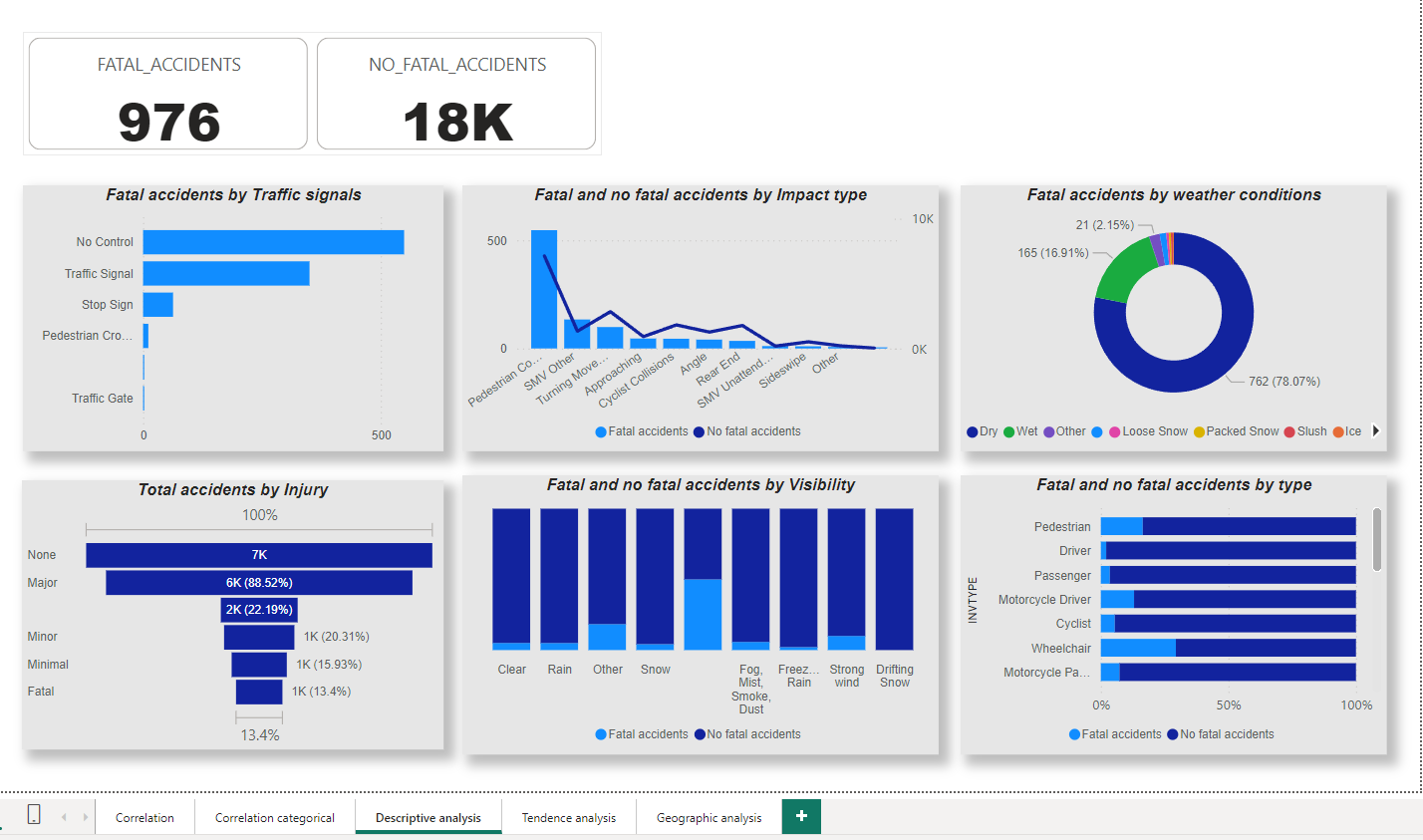




#### Analyzing fatal accidents data against feature columns

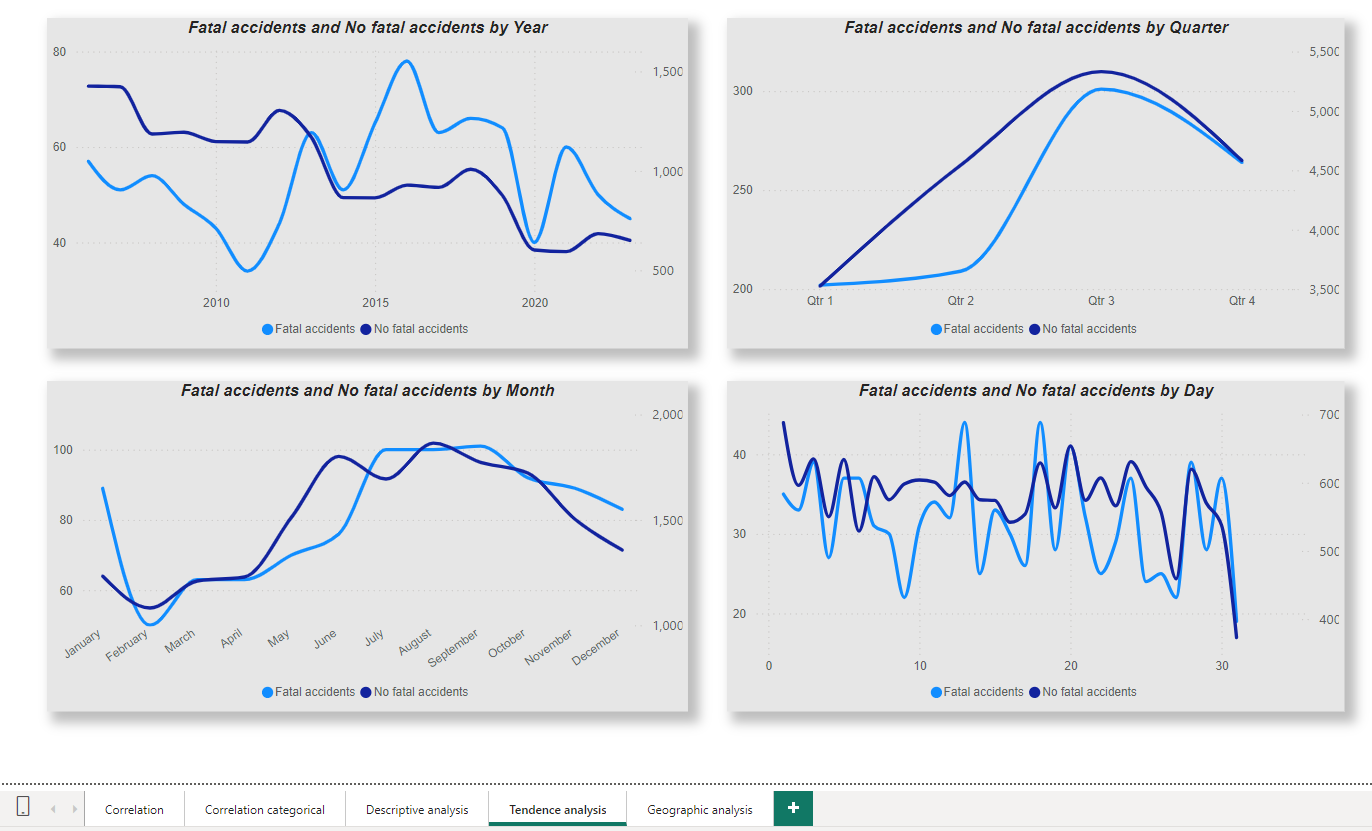
Below are some additional analyses of relation between different columns of our dataset. For 976 recorded fatal accidents, we can see some interesting patterns emerge in our dataset.

* More fatal accidents were caused by no control in traffic signal than by stop sign or pedestrian crossings.
* Pedestrians were more significantly impacted than any other.
* More fatal accidents were caused in dry weather conditions than any other.
* A high number of accidents resulted in a major injury, with 13 percent resulting in fatality.
* Most of the fatal accidents involved wheelchair users and pedestrians. Cyclists were surprisingly less victims of fatal accidents based on the data.



#### Analyzing fatal accidents by date

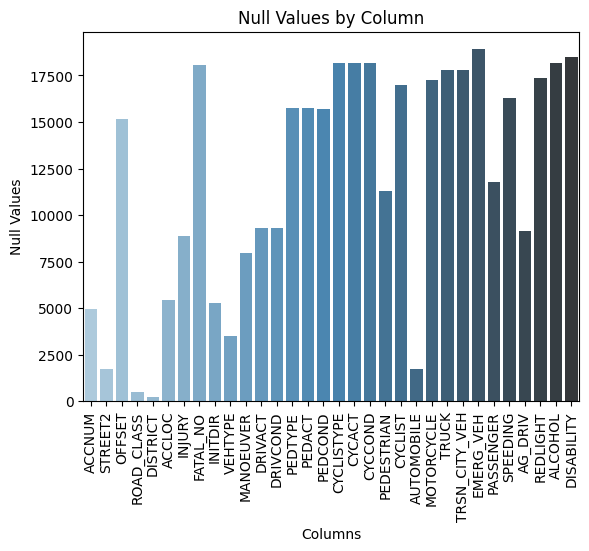
* According to below line graphs, the highest number of fatal accidents were observed in 2016 and lowest in 2011. Likewise, the trend of non-fatal accidents seems to roughly decrease as we progress through years 2010 to 2020.
* For any given year, the highest number of fatal and non-fatal accidents were observed in the third quarter of the year. It looks to decline as we approach the end of year. Interestingly, fatal accidents remain at a low number up until the second quarter.
* Also, the number of fatal accidents is generally low during the month of February and it climbs to a maximum number during the months of July, August and September. Non-fatal accidents also follow a somewhat similar pattern reaching the highest peak in the month of August and trending downwards till December.



### Transforming the dataset

#### Handling missing values

While our dataset shows excellent quality, it is also lacking in some areas as having high null values. Below graph shows distribution of null values.



For numerical values, we decided to use the mode of each column to fill missing values with highest frequency data

We decided to include columns that, despite having some missing values, are fundamental to the analysis. Missing values were handled as follows:

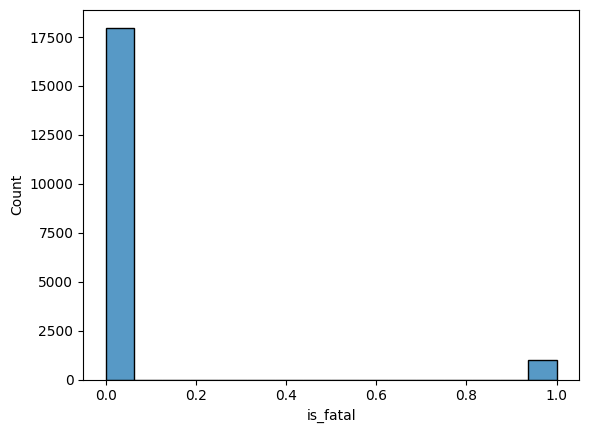
• For **INJURY**, missing values were replaced with 'None'.

• For **FATAL\_NO**, missing values were replaced with 0.

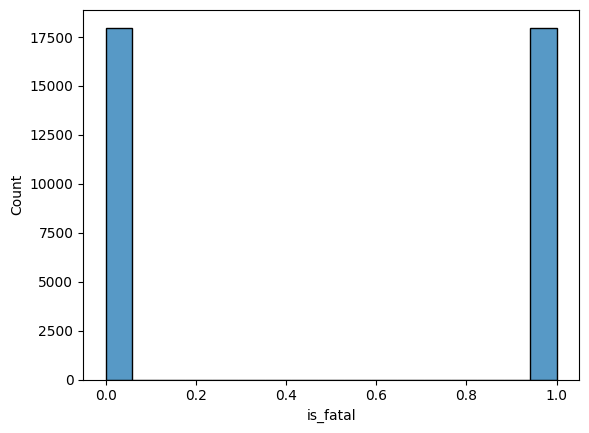
#### Handling imbalance target class

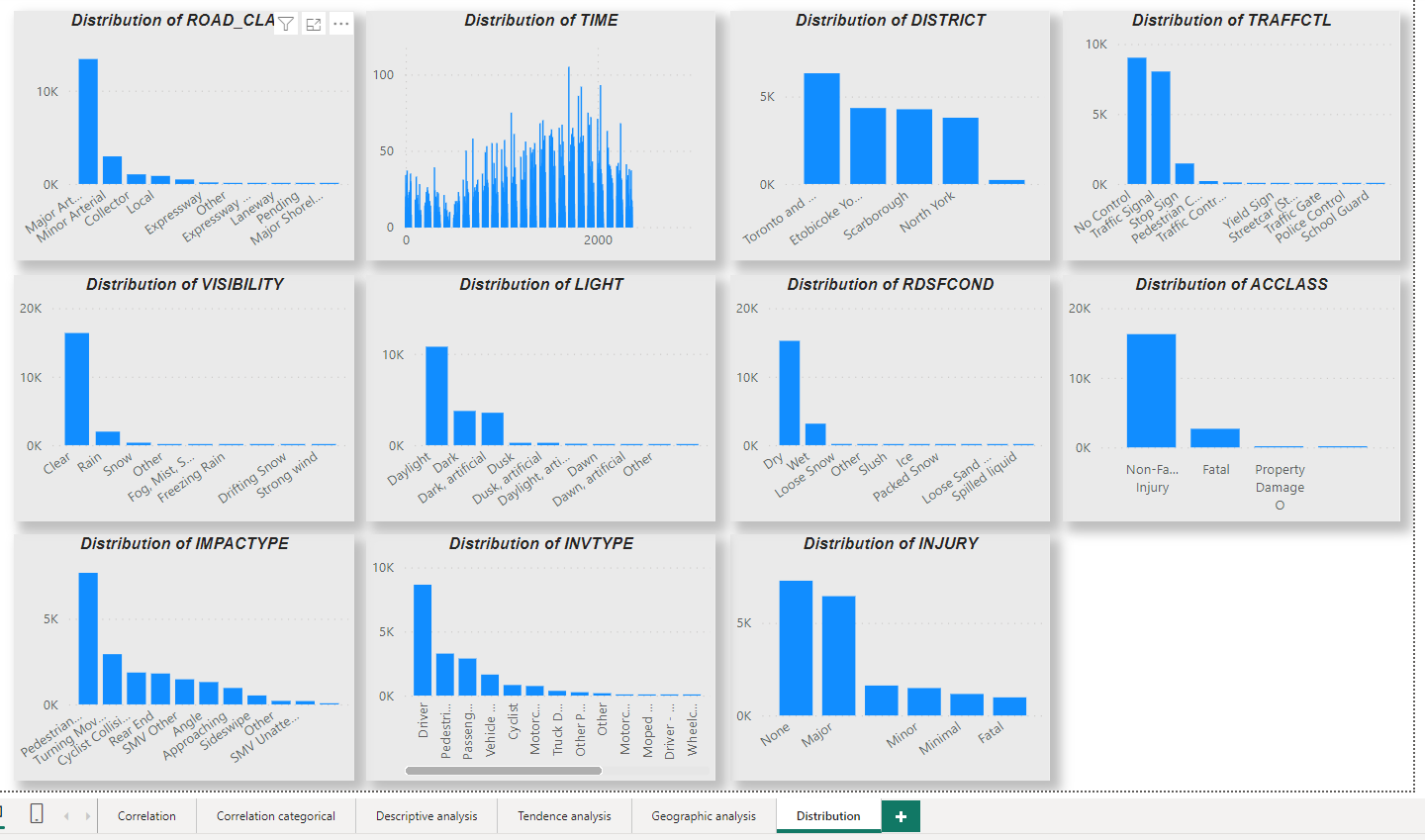
As observed previously, our target class IS\_FATAL has a high imbalance with most of the values as 0 (or not fatal). To handle this we will use oversampling using the SMOTE technique.

Before oversampling our target distribution looks like this:



Notice the high variance in count of 0 versus 1. After applying SMOTE, we were able to produce a balanced target column as follows:





### Model building & fine tuning

#### **Overview**

This report outlines the steps taken to evaluate and tune different machine learning models for predicting fatal accidents based on a dataset. The primary goal is to determine the most effective model for predicting fatal accidents using various classification algorithms and to optimize these models using Grid Search.

#### **Data Preparation**

1. **Feature and Target Variable Separation:**

y = df\_accidents['is\_fatal']

X = df\_accidents.drop('is\_fatal', axis=1)

* + The target variable y is set as 'is\_fatal', indicating whether an accident was fatal.
  + The feature matrix X contains all other columns from the dataset, excluding 'is\_fatal'.

1. **Data Splitting:**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=84, stratify=y)

* + The data is split into training and testing sets with 20% allocated to testing.
  + Stratification ensures that the proportion of the target classes is maintained in both training and testing sets.

1. **Feature Identification:**

numeric\_cols = X\_train.select\_dtypes(include=['number']).columns

non\_numeric\_cols = X\_train.select\_dtypes(exclude=['number']).columns

* + Numeric and non-numeric columns are identified separately to apply appropriate preprocessing techniques.

#### **Preprocessing**

1. **Numeric Feature Scaling:**

numeric\_transformer = Pipeline(steps=[

('scaler', StandardScaler())

])

* + A pipeline is created to scale numeric features using StandardScaler, which standardizes features by removing the mean and scaling to unit variance.

1. **Categorical Feature Encoding:**

categorical\_transformer = Pipeline(steps=[

('onehot', OneHotEncoder(handle\_unknown='ignore'))

])

* + Categorical features are encoded using OneHotEncoder, which converts categorical variables into a format that can be provided to machine learning algorithms.

1. **Combining Preprocessing Steps:**

preprocessor = ColumnTransformer(

transformers=[

('num', numeric\_transformer, numeric\_cols),

('cat', categorical\_transformer, non\_numeric\_cols)

])

* + ColumnTransformer combines both numeric and categorical preprocessing steps into a single transformation pipeline.

#### **Model Definition and Tuning**

1. **Model Selection:**

models = {

'Logistic Regression': LogisticRegression(max\_iter=1000),

'Decision Tree': DecisionTreeClassifier(),

'SVM': SVC()

}

* + Three classification models are defined: Logistic Regression, Decision Tree, and Support Vector Machine (SVM).

1. **Pipeline Creation with SMOTE:**

pipelines = {name: ImbPipeline(steps=[('preprocessor', preprocessor), ('smote', SMOTE()), ('classifier', model)])

for name, model in models.items()}

* + Each model is combined into a pipeline with SMOTE for handling class imbalance, which performs oversampling of the minority class.

1. **Parameter Grid Definition:**

param\_grid\_lr = {

'classifier\_\_C': [0.1, 1, 10, 100],

'classifier\_\_solver': ['lbfgs', 'liblinear']

}

param\_grid\_dt = {

'classifier\_\_max\_depth': [3, 5, 7, 10],

'classifier\_\_min\_samples\_split': [2, 5, 10],

'classifier\_\_min\_samples\_leaf': [1, 2, 4]

}

param\_grid\_svc = {

'classifier\_\_C': [0.1, 1, 6],

'classifier\_\_kernel': ['rbf'],

'classifier\_\_gamma': ['scale'],

'smote\_\_k\_neighbors': [3, 5]

}

* + Parameter grids are defined for Grid Search to tune hyperparameters of each model, including regularization strength for Logistic Regression, tree depth and splits for Decision Trees, and C-value and kernel parameters for SVM.

1. **Grid Search Initialization:**

grid\_searches = {

'Logistic Regression': GridSearchCV(pipelines['Logistic Regression'], param\_grid\_lr, cv=5, n\_jobs=-1),

'Decision Tree': GridSearchCV(pipelines['Decision Tree'], param\_grid\_dt, cv=5, n\_jobs=-1),

'SVM': GridSearchCV(pipelines['SVM'], param\_grid\_svc, cv=5, n\_jobs=-1)

}

* + GridSearchCV is used to perform an exhaustive search over the parameter grid for each model, utilizing cross-validation.

#### **Model Evaluation**

1. **Model Training and Evaluation:**

for name, grid\_search in grid\_searches.items():

grid\_search.fit(X\_train, y\_train)

print(f"Best parameters for {name}: {grid\_search.best\_params\_}")

y\_pred = grid\_search.predict(X\_test)

print(f"Classification report for {name} after tuning:")

print(classification\_report(y\_test, y\_pred))

* + Grid Search is performed for each model to find the best hyperparameters.
  + The best model is evaluated on the test set, and classification reports are printed.

1. **Best Model Selection:**

best\_models = {name: grid\_search.best\_estimator\_ for name, grid\_search in grid\_searches.items()}

best\_model = best\_models['Decision Tree']

best\_model.fit(X\_train, y\_train)

y\_pred = best\_model.predict(X\_test)

print("Classification report after tuning:")

print(classification\_report(y\_test, y\_pred))

* + The best-performing model (Decision Tree in this case) is selected and evaluated again to verify its performance.

#### **Results**

* **Logistic Regression:**
  + Best parameters: {'classifier\_\_C': 10, 'classifier\_\_solver': 'liblinear'}
  + The model achieved an accuracy of 96%, with a precision of 0.60 and recall of 0.60 for the fatal class.
* **Decision Tree:**
  + Best parameters: {'classifier\_\_max\_depth': 3, 'classifier\_\_min\_samples\_leaf': 1, 'classifier\_\_min\_samples\_split': 10}
  + This model performed exceptionally well with an accuracy of 99%, high precision (1.00) and recall (0.89) for the fatal class.
* **SVM:**
  + Best parameters: {'classifier\_\_C': 6, 'classifier\_\_gamma': 'scale', 'classifier\_\_kernel': 'rbf', 'smote\_\_k\_neighbors': 3}
  + The SVM model achieved an accuracy of 98%, with precision of 0.83 and recall of 0.66 for the fatal class.

#### **Conclusion**

The Decision Tree model emerged as the most effective for this dataset, achieving the highest accuracy and balanced performance across precision and recall metrics. Logistic Regression and SVM also performed well, but with some trade-offs between precision and recall for the fatal class. Further tuning and additional data exploration could provide more insights and potentially improve the performance of the models.