Analysis of Road Accident Severity in Rainy Weather

Image source: https://unsplash.com/photos/p3Ip8U0eNNM

Introduction/Business problem

Image source: https://unsplash.com/photos/PN-YnI5stdQ

The goal of this project is to predict the severity of road accidents in rainy weather due to various conditions.

Why is rainy weather selected for this analysis?
Road accidents are a common occurrence, and wet roads along with poor visibility increase the risk further during the rainy season. Lack of awareness is a major cause of accidents.

Benefits of this project:

- This will help the community to stay safe and avoid damage or loss due to accidents in rainy weather.
- This will also help the Road Safety team to take necessary precautions.

How can we achieve the goals?
As a first step, we need the latest data related to road accidents. The data should contain all possible information such as Weather when the accident occured, area, condition of the roads, time of accident (day or night), severity, injuries, damages etc.

Once we get the data, it will be processed, analysed, trained and tested using machine learning models and identify the right model to predict road accident possibilities.

Data Understanding

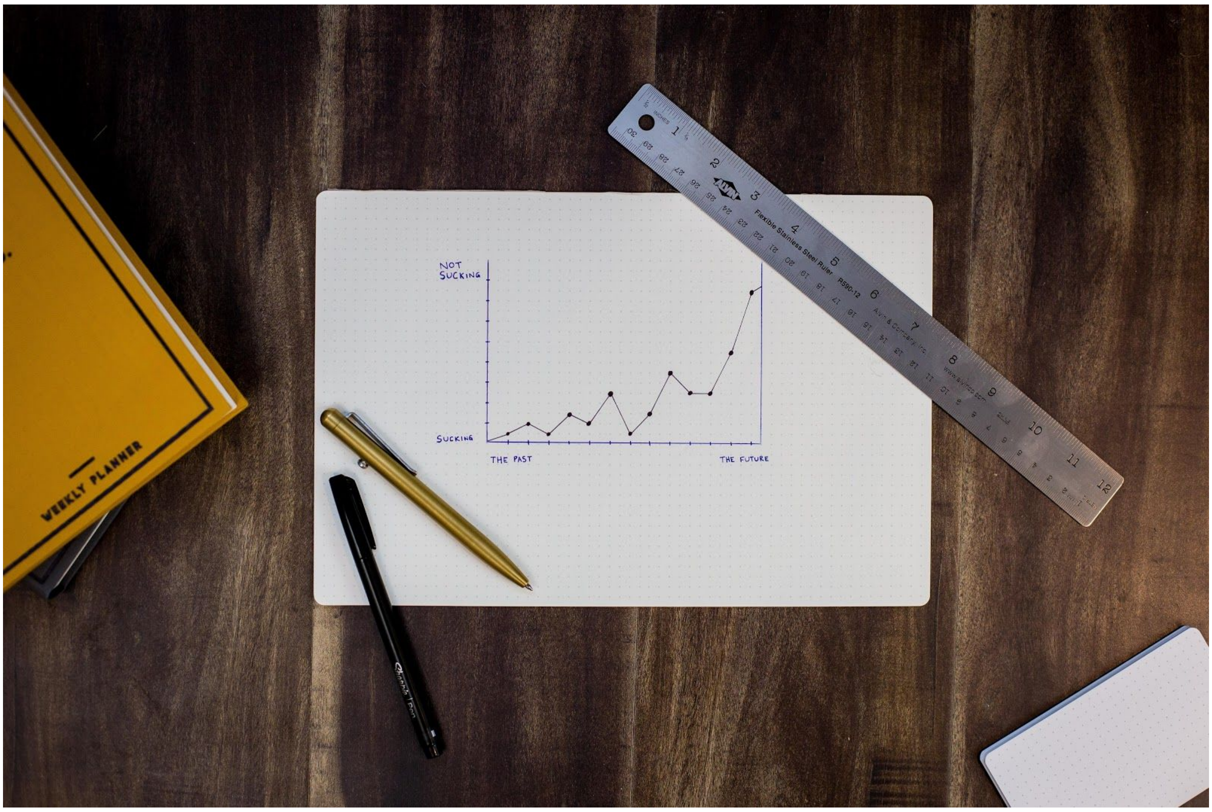
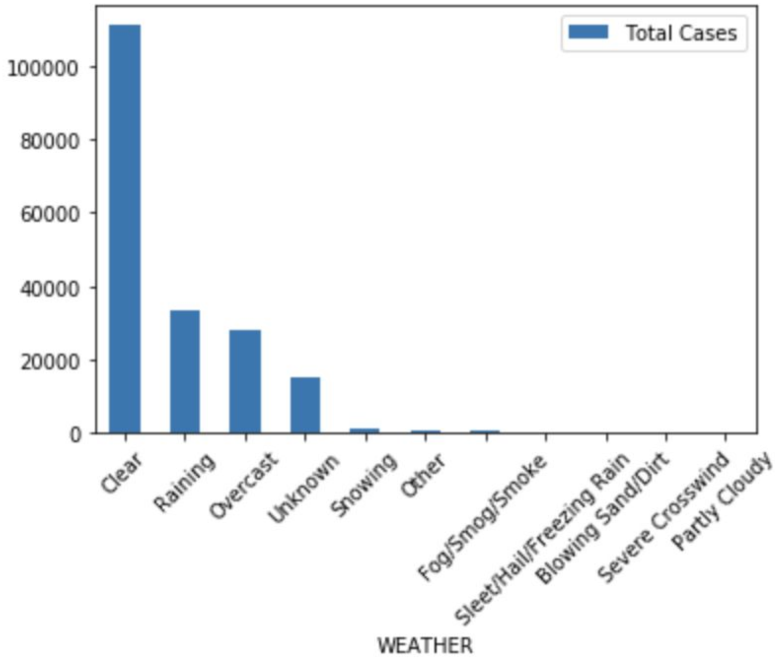


Image source:https://unsplash.com/photos/6EnTPvPPL6I

```
In [31]: plot_df.plot.bar(x='WEATHER', y='Total Cases', rot=45)
```

```
Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x7f0a79cbef60>
```



By analysing the raw data, the following conclusion can be done.

1. Severity code is the target parameter or predictor variable which as it shows the severity of the accidents.
2. Data clean up is required as few columns are not required for analysis.
3. Since the goal is to get predictions for rainy weather, only rows corresponding to 'Rainy' Weather can be used.
4. **ROADCOND** and **LIGHTCOND** are different categories that can be derived from the Weather column.
5. Convert raw unbalanced data to balanced dataset.

Severity code

For Rainy weather, the SEVERITY CODE is either 1 or 2, where 1 indicates it is Safe to travel and 2 indicates damage to life or property. This can be used as a Target variable to derive a solution.

```
In [6]: #check severity value counts of initial data
        df['SEVERITYCODE'].value_counts()

Out[6]: 1    136485
        2     58188
        Name: SEVERITYCODE, dtype: int64
```

Data Clean Up

Post extracting csv data to the data frame, a clean up is required to remove unwanted data. Columns excluding SEVERITY CODE, WEATHER, ROADCOND, LIGHTCOND can be removed. This creates a clean data set with only required columns.

In [7]:
```
# Clean up data by dropping unwanted columns
dataset = df.drop(columns = ['X', 'Y', 'OBJECTID', 'INCKEY', 'COLDETKEY',
         'REPORTNO', 'STATUS', 'ADDRTYPE', 'INTKEY', 'LOCATION',
         'EXCEPTRSNCODE', 'EXCEPTRSNDESC', 'SEVERITYCODE.1', 'SEVERITYDESC',
         'COLLISIONTYPE', 'PERSONCOUNT', 'PEDCOUNT', 'PEDCYLCOUNT',
         'VEHCOUNT', 'INCDATE', 'INCDTTM', 'JUNCTIONTYPE', 'SDOT_COLCODE',
         'SDOT_COLDESC', 'INATTENTIONIND', 'UNDERINFL', 'PEDROWNOTGRNT', 'SDOTCOLNUM', 'SPEEDING',
         'ST_COLCODE', 'ST_COLDESC', 'SEGLANEKEY', 'CROSSWALKKEY',
         'HITPARKEDCAR'])
dataset.shape
```

Out[7]: (194673, 4)

Extract Required Data

Since analysis is based on Rainy weather, rows including other weather conditions can be removed. This creates a clean data set with only required rows.

In [321]:
```
#Extract data corresponding to rainy weather
rain_data = dataset[(dataset['WEATHER'] == 'Raining')].copy()
rain_data.head()
```

Out[321]:

	SEVERITYCODE	WEATHER	ROADCOND	LIGHTCOND
1	1	Raining	Wet	Dark - Street Lights On
4	2	Raining	Wet	Daylight
6	1	Raining	Wet	Daylight
12	1	Raining	Wet	Dark - Street Lights On
13	1	Raining	Wet	Dark - No Street Lights

Categorise data

ROADCOND and LIGHTCOND are two columns which impact target variables along with Weather data. One of the major reasons why we convert categorical variables into factors i.e number because to make Analysis easy and effective.

	index	SEVERITYCODE	WEATHER	ROADCOND	LIGHTCOND	CATEGORY_WEATHER	CATEGORY_ROADCOND	CATEGORY_LIGHTCOND
0	1	1	Raining	Wet	Dark - Street Lights On	0	8	2
1	4	2	Raining	Wet	Daylight	0	8	5
2	6	1	Raining	Wet	Daylight	0	8	5
3	12	1	Raining	Wet	Dark - Street Lights On	0	8	2
4	13	1	Raining	Wet	Dark - No Street Lights	0	8	0

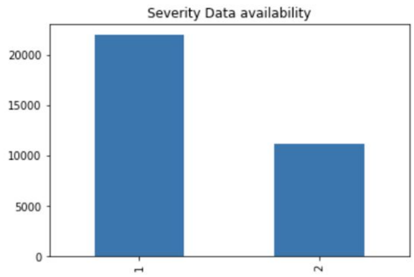
Convert Unbalanced data to balanced dataset

Unbalanced data refers to classification problems where we have unequal instances for different classes. Having unbalanced data is actually very common in general. Downsampling method is followed to achieve this.The main goal of downsampling (and upsampling) is to increase the discriminative power between the two classes. Here is a plot of Unbalanced data:

```
In [100]: rain_data['SEVERITYCODE'].value_counts()

Out[100]: 1    21969
          2    11176
          Name: SEVERITYCODE, dtype: int64
```

```
In [102]: #Raw unbalanced data
          severity_unbalanced = rain_data['SEVERITYCODE'].value_counts().plot(kind='bar',title="Severity Data availability")
```



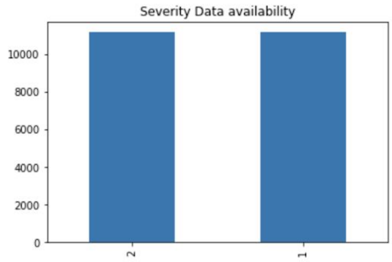
Downsampling step

```
In [107]: from sklearn.utils import resample
          #Majority class downsampling
          rain_data_adjusted = resample(rain_data[rain_data.SEVERITYCODE==1],replace=False,
                                        n_samples=11176,random_state=123)
          # combining downsampled majority class with minority class
          rain_data_balanced = pd.concat([rain_data_adjusted, rain_data[rain_data.SEVERITYCODE==2]])
          rain_data_balanced.SEVERITYCODE.value_counts()

Out[107]: 2    11176
          1    11176
          Name: SEVERITYCODE, dtype: int64
```

Data after downsampling

In [536]:
```
severity_balanced = rain_data_balanced['SEVERITYCODE'].value_counts().plot(kind='bar',title="Severity Data availability")
```



Methodology

Initial Steps before training and testing models.

1. Defining variables X and Y

In [66]:
```
import numpy as np
X=np.asarray(rain_data_balanced[['CATEGORY_ROADCOND','CATEGORY_LIGHTCOND']])
X[0:5]
```

Out[66]:
```
array([[8, 2],
       [8, 5],
       [8, 5],
       [8, 6],
       [8, 5]], dtype=int8)
```

In [67]:
```
y=np.asarray(rain_data_balanced[['SEVERITYCODE']])
y[0:5]
```

Out[67]:
```
array([[1],
       [1],
       [1],
       [1],
       [1]])
```

2. Standardising the data set using scikit-learn

In [68]:
```
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

Out[68]:
```
array([[ 0.15600865, -1.08480647],
       [ 0.15600865,  0.75780923],
       [ 0.15600865,  0.75780923],
       [ 0.15600865,  1.37201447],
       [ 0.15600865,  0.75780923]])
```

3. Split Data into train and test set

In [126]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2)
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (17881, 2) (17881, 1)
Test set: (4471, 2) (4471, 1)
```

4. Modelling

Following machine learning models are used:

1. K Nearest Neighbor (KNN)
2. Decision Tree
3. Logical Regression

Image Source: https://unsplash.com/photos/SyzQ5aByJnE

K Nearest neighbor (KNN)

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. To select the K that's right for your data, we run the KNN algorithm several times with different values of K and choose the K that reduces the number of errors we encounter while maintaining the algorithm's ability to accurately make predictions when it's given data it hasn't seen before.

KNN

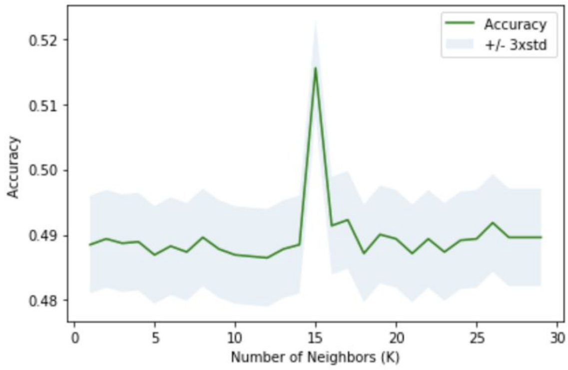
In [355]:
```
from sklearn.neighbors import KNeighborsClassifier
k=15
#Steps to train the model and predict
neigh = KNeighborsClassifier(n_neighbors=k).fit(X_train,y_train)
yhatKn = neigh.predict(X_test)
yhatKn[0:5]
```

Out[355]:
```
array([2, 2, 2, 2, 2])
```

In [356]:
```
from sklearn import metrics
print('Train set accuracy', metrics.accuracy_score(y_train, neigh.predict(X_train)))
print('Test set accuracy', metrics.accuracy_score(y_test, yhatKn))
```

```
Train set accuracy 0.5076897265253622
Test set accuracy 0.5155446208901812
```

In [338]:
```
#plotting the graph
import matplotlib.pyplot as plt
plt.plot(range(1,Ks), mean_acc, 'g')
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
plt.legend(('Accuracy ', '+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()
print( "The best accuracy was with", mean_acc.max(), "with k=", mean_acc.argmax()+1)
```




```
The best accuracy was with 0.5155446208901812 with k= 15
```

Decision Tree algorithm

Decision Tree algorithm belongs to the family of supervised learning algorithms. The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by learning simple decision rules inferred from prior data.

Decision tress

In [528]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
import matplotlib.pyplot as plt
```

In [529]:
```
depth = 12
dTree = DecisionTreeClassifier(criterion="entropy", max_depth=depth)
dTree.fit(X_train,y_train)
yhatD = dTree.predict(X_test)
print(yhatD[0:5])
print("Accuracy:",metrics.accuracy_score(y_test, yhatD))
```

```
[1 1 2 1 1]
Accuracy: 0.5157682845001118
```

In [530]:
```
dtree_f1 = f1_score(y_test, yhatD)
dtree_jaccard = jaccard_similarity_score(y_test, yhatD)
print('F1 score is ' + str(dtree_f1))
print('jaccard similarity score is ' + str(dtree_jaccard))
print('Most accurate max depth',depth)
```

```
F1 score is 0.48341684562157006
jaccard similarity score is 0.5157682845001118
Most accurate max depth 12
```

Linear Regression Model

Linear Regression is a supervised machine learning algorithm where the predicted output is continuous and has a constant slope.

Linear regression model

In [531]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import log_loss
from sklearn import datasets, linear_model, metrics
LR = LogisticRegression(C=7, solver='liblinear').fit(X_train,y_train)
LR
```

Out[531]:
```
LogisticRegression(C=7, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='liblinear',
          tol=0.0001, verbose=0, warm_start=False)
```

In [532]:
```
# Train Model & Predicr
LRyhat = LR.predict(X_test)
yhat_prob = LR.predict_proba(X_test)
# Linear Regression Jaccard Similarity Score
linreg_f1 = f1_score(y_test,LRyhat)
linreg_jaccard = jaccard_similarity_score(y_test, LRyhat)
logloss = log_loss(y_test, yhat_prob)
```

Results

Accuracy of models on test data set
Following metrics can be used to evaluate a model.

1. **Jaccard Index**: It's a measure of similarity for the two sets of data, with a range from 0% to 100%
2. **F1 score**: The F-score, also called the F1-score, is a measure of a model's accuracy on a dataset.
3. **Log loss**: Log Loss quantifies the accuracy of a classifier by penalising false classifications.

	Algorithms	F1 scores	Jaccard Scores	Log loss
0	KNN	0.486339	0.489600	NA
1	Decision Tree	0.483417	0.515768	NA
2	Linear Regression	0.480210	0.512413	0.69313

Discussion

i. Found input variables with inconsistent numbers of samples error is caused due to imbalance in data. Severity codes are not evenly distributed which caused error in training and test split data. This is resolved by downsampling. Details are provided in Data Understanding section.
ii. K Nearest Neighbor accuracy is verified with different ranges of K. The accuracy value is 15 in all the executions. In order to reduce time in multiple executions, calculations were executed in a loop to get K accuracy value.
iii. Decision tree gives best accuracy when depth is 12.
iv. Linear regression has minimum log loss when hyper parameter is 7.

Conclusion

Since the goal is to decide whether travelling by road is safe or not, this classifies under binary results. Logical regression model is more accurate to predict safety. Based on severity code, higher the severity code in rainy weather, higher level of risk such as injury is predicted. This is applicable for Severity 2. Damage is predicted for code 1.