Assignment on

Car Accident Severity – Prediction using Machine Learning Algorithms

Submitted in partial fulfillment of the course

Applied Data Science Capstone

by

Sanket Amdalli

Date of submission – 5th October 2020

Acknowledgements

I whole-heartedly thank Coursera and all the instructors of the IBM Data Science Professional Certificate for their mentorship and guidance through all the courses. All courses have been impeccably designed, and have a perfect mix of theoretical concepts learning and practical application. The evaluation system is also quite well-designed. It has been a thoroughly enriching experience getting to learn from the best at IBM, and I'm certain that the knowledge and skills gained will help me immensely in the future.

I would also like to thank my parents and family, and of course, special thanks to my dearest Sanjana, without whose constant support, motivation and inspiration, this would never have been possible.

Table of Contents

1. Introduction

1.1 Background

Road accidents cause upwards of 1.35 million deaths, and between 20-50 million injuries every year globally, as reported by the World Health Organization. The major factors that cause road accidents and increase the risk of injury from accidents, stated by the WHO, are - Speeding, Driving under the influence of alcohol and other psychoactive substances, Non-use of motorcycle helmets, seat-belts, and child restraints, Distracted driving (using mobile phones), Unsafe vehicles, Unsafe road infrastructure, and Inadequate post-crash care. While most of these factors are the responsibility of the vehicle driver, the last two (road infrastructure and post-crash service) are under control of the government, and road authorities can certainly take many steps to ensure that accidents are averted, and when they do happen, the casualties are attended to as quickly as possible.

1.2 Problem

We see warning signboards of "Accident Prone Area, Drive Carefully" in places where a lot of accidents have happened in the past. This is a rudimentary example of how a road authority has used a simple data point (number of accidents) to derive an insight (accident-prone area) and set-up a warning sign for drivers. The idea of this project is to use the same principle of deriving insights from different data points, which can be used by the authorities to improve the infrastructure and post-crash service. With various attributes about the accident – the time of day, location, road conditions, light conditions, weather conditions, etc. – available to us, a deeper analysis can be performed to determine the extent to which these factors impact road accidents, and consequently a model can be developed using the key factors to predict the severity of potential accidents.

1.3 Interest

The primary consumers of our model (target audience) will be the governments – the road/transport authorities, who can use insights from the model and build a mechanism to alert vehicle drivers about difficult driving conditions and potential hazards, thereby reducing the frequency of accidents. By identifying the severity conditions, they can also inform the police and medical authorities to be

prepared for quick action in case an accident happens. In the long run, insights from the model can be used while planning the construction of new roads (to answer questions like 'Do we need more streetlights on this road?' 'Is it better to construct a junction with traffic lights or simply a roundabout?', etc.). For the scope of this project, the government authorities of the city of Seattle, Washington are the target audience.

2. Data Understanding and Preparation

2.1 Data Source

The data source used for the project is collected by the Seattle Police Department, and maintained and owned by the Seattle Department of Transport (SDoT). It contains the records of vehicle collisions in the city of Seattle since year 2004, with attributes such as the accident severity, location, timestamp, number of people involved, number of vehicles involved, number of injuries/fatalities, whether a driver was speeding, whether a driver was driving under influence, road conditions, light/visibility conditions and weather conditions among others.

Total number of records (as on 27 Sep 2020): 221,738
Number of fields: 40
Update frequency: Weekly

The homepage for the dataset is here:
https://data-seattlecitygis.opendata.arcgis.com/datasets/5b5c745e0f1f48e7a53acec63a0022ab_0

Detailed attribute information can be found at this link:
https://www.seattle.gov/Documents/Departments/SDOT/GIS/Collisions_OD.pdf

Metadata for the dataset is here:
https://www.arcgis.com/sharing/rest/content/items/5b5c745e0f1f48e7a53acec63a0022ab/info/metadata/metadata.xml?format=default&output=html

2.2 Target Variable

Since the problem statement is to predict the severity of accidents, variable 'SEVERITYCODE' has been chosen as the target variable. Upon examination, it is found that the variable has coded values to signify the below types of severity:

Severity Code	Status Description
0	Unknown
1	Property Damage Only Collision
2	Injury Collision
2b	Serious Injury Collision
3	Fatality Collision

Table 1: Categories of the 'SEVERITYCODE' variable and their description

The code '0', corresponding to the value 'Unknown', has a lot of empty attribute values, especially for the weather, road conditions, light conditions variables. And having an unknown value in my target variable is not going to help my model in any way. Hence, I have dropped the rows having '0' severity code from the dataset.

Further, it is observed that the number of accidents classified as '2b' and '3' severity are very few compared to '1' and '2' (less than 2% of the total dataset). Including them as separate categories will not have an impact on my model. So, I have combined the categories 2, 2b, and 3 into a single category. This also makes the target variable binary, with just 2 categories:

Severity Code	Status Description
1	Only Property Damage
2	Risk of Injury/Fatality

Table 2: Binary target variable created by combining severity codes

2.3 Exploratory Data Analysis

Once my target variable has been identified, the next step in understanding the data is to plot some graphs and run descriptive statistics to understand the relationship of different features with our target variable. Also, there are certain logical assumptions that can be made based on the data we have, and using exploratory data analysis, I have tried to validate if these assumptions are correct.

For instance, a logical assumption could be that the more number of people/vehicles involved in an accident, the more severe it is. On plotting the mean values of persons, pedestrians, cyclists and vehicles for each severity category (Figure 1), it is observed that the assumption seems to hold true for persons, but not necessarily for vehicles. The average number of vehicles involved in an accident actually decreases as the severity of the accident increases (1.94 to 1.44). A possible explanation for this could be that more number of people involved in an accident increases the chances of injury for at least one of the persons, while more number of vehicles involved in an accident means that the people involved are inside their vehicles, and the chances of them getting hurt become lesser.

```
In [252]: x = dataset.groupby(['SEVERITYDESC']).mean()
          x[['PERSONCOUNT','PEDCOUNT','PEDCYLCOUNT','VEHCOUNT']]
```

Out[252]:

SEVERITYDESC	PERSONCOUNT	PEDCOUNT	PEDCYLCOUNT	VEHCOUNT
Fatality Collision	3.002841	0.426136	0.079545	1.437500
Injury Collision	2.713555	0.111944	0.083461	1.871095
Property Damage Only Collision	2.328766	0.005269	0.004994	1.945085
Serious Injury Collision	2.575378	0.315654	0.140469	1.571842
Unknown	2.000000	0.000000	1.000000	1.000000

Figure 1: Code snippet and output table to validate assumption of vehicles vs severity.

Likewise, another logical assumption could be that more vehicles are involved in an accident when the road, light and weather conditions are poor. However, looking at the data (Figure 2), this assumption doesn't seem to be always true. Despite poor visibility in low light conditions, the most number of average vehicles involved are in accidents that happen in Daylight (1.98) and Dusk (1.96). Comparatively, adverse lighting conditions have marginally lower average number of vehicles being involved (1.88 for 'Dark – No Street Lights', and 1.90 for 'Dark – Street Lights Off' conditions. The same goes for Road conditions, the accident on a dry road involves more vehicles than an accident on a wet/oily/muddy/snowy road on average. A possible explanation for this could be the fact that people drive more carefully when the weather, light and road conditions are adverse.

In [269]:
```
x = dataset.groupby(['LIGHTCOND']).mean()
x[['VEHCOUNT','SEVERITYCODE']]
```

Out[269]:

LIGHTCOND	VEHCOUNT	SEVERITYCODE
Dark - No Street Lights	1.884177	1.231646
Dark - Street Lights Off	1.909605	1.281679
Dark - Street Lights On	1.926125	1.314984
Dark - Unknown Lighting	1.583333	1.333333
Dawn	1.886930	1.346110
Daylight	1.980829	1.343215
Dusk	1.960887	1.343139
Other	1.885246	1.237705
Unknown	1.981231	1.046922

In [270]:
```
x = dataset.groupby(['WEATHER']).mean()
x[['VEHCOUNT','SEVERITYCODE']]
```

Out[270]:

WEATHER	VEHCOUNT	SEVERITYCODE
Blowing Sand/Dirt	2.017857	1.267857
Blowing Snow	2.000000	2.000000
Clear	1.969488	1.336228
Fog/Smog/Smoke	1.896014	1.336222
Other	1.948837	1.151163
Overcast	1.961409	1.327508
Partly Cloudy	2.200000	1.500000
Raining	1.942651	1.348591
Severe Crosswind	1.692308	1.307692
Sleet/Hail/Freezing Rain	1.896552	1.267241
Snowing	1.970620	1.196953
Unknown	1.975547	1.056573

```
In [271]: x = dataset.groupby(['ROADCOND']).mean()
          x[['VEHCOUNT','SEVERITYCODE']]
```

Out[271]:

ROADCOND	VEHCOUNT	SEVERITYCODE
Dry	1.970706	1.335419
Ice	1.852273	1.238636
Oil	1.718750	1.375000
Other	1.772059	1.338235
Sand/Mud/Dirt	1.623377	1.298701
Snow/Slush	1.946746	1.172584
Standing Water	1.630252	1.277311
Unknown	1.975626	1.051919
Wet	1.946240	1.343374

Figure 2: Code snippets and output tables to validate assumption of vehicles vs conditions.

Another such assumption is that a majority of accidents of least severity should be involving parked cars/stationary vehicles. When I plotted the graph of Collision Type by Severity of accidents, I observed that this assumption is indeed true. For Severity Code=1, i.e. Non-Injury type accidents, the major collision type is 'Parked Car'. While for Injury type accidents (Code = 2), the number of 'Parked Car' accidents is very less. (Figure 3)

```
In [273]: sns.catplot(y='SEVERITYCODE', hue='COLLISIONTYPE', kind='count', palette='Set1', data=dataset)
Out[273]: <seaborn.axisgrid.FacetGrid at 0x7f8feea059a0>
```

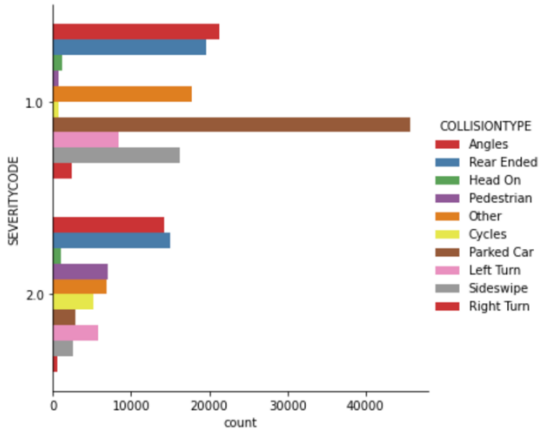


Figure 3: Code snippet and output graph to validate assumption of severity vs parked vehicle accidents.

I have also extracted the correlation matrix to understand the correlation of the number of persons, pedestrians, cyclists and injuries with the severity of the accident (Figure 4). The matrix shows a strong correlation of the injuries with the accident severity (which is understandable since the severity code is basis the extent of injury), but weak correlation between the number of persons, pedestrians or cyclists with the accident severity. Amongst the three, number of pedestrians are more correlated with the accident severity (again understandable since pedestrians are at more risk of injury from an accident, and hence an increased severity).

```
In [277]: dataset[['SEVERITYCODE','PERSONCOUNT','PEDCOUNT','PEDCYLCOUNT','INJURIES']].corr()
```

Out[277]:

	SEVERITYCODE	PERSONCOUNT	PEDCOUNT	PEDCYLCOUNT	INJURIES
SEVERITYCODE	1.000000	0.128840	0.259243	0.217701	0.811250
PERSONCOUNT	0.128840	1.000000	-0.019099	-0.038833	0.279333
PEDCOUNT	0.259243	-0.019099	1.000000	-0.019368	0.159475
PEDCYLCOUNT	0.217701	-0.038833	-0.019368	1.000000	0.115178
INJURIES	0.811250	0.279333	0.159475	0.115178	1.000000

Figure 4: Code snippet and output table for correlation matrix

In addition, I have plotted a few relation plots and box-plots to understand the variable relationships, data distributions and outliers. Refer the Jupyter Notebook and Appendix A for details.

2.4 Feature Selection

After examining the features and exploring variable relationships, I have retained only a few features, and dropped the ones that are redundant or irrelevant to my analysis. Since the problem statement is to predict the severity of accidents, **only the attributes that could be possible 'causes' for affecting the severity of accidents** need to be included in the model. For instance, the no. of fatalities/injuries/vehicles attributes are related to the severity of an accident, however, they are a consequence of the accident, not a cause for the accident. Hence they have been excluded from the model.

Also, attributes such as whether the accident happened due to driving under influence, speeding, inattention, etc. are factors which may affect the severity of accidents, but these factors are in control of the drivers, and it is impossible for the road/transport authorities to ascertain beforehand whether a person is

driving under influence and send out alerts of potential accidents to other drivers. There can certainly be a correlation between say, an area which has a lot of pubs/bars, and the possibility of severe accidents happening in that area at late evening/weekends because of more people driving under influence. And such a correlation must be explored. But it is currently beyond the scope of my analysis, and hence I have excluded these attributes from my model. Some other attributes which are simply long-form descriptions of other coded attributes in the dataset, and hence are redundant, have also been excluded.

The three features that I have chosen as the input variables for my model are:

- ROADCOND – Road Conditions
- WEATHER – Weather Conditions
- LIGHTCOND – Light Conditions

2.5 Data Cleaning and Preparation

The next step is to perform the necessary cleaning processes need to prepare the dataset for analysis. This includes – handling Missing/Null values, converting the variables into the appropriate data type, removing outliers (if any), and balancing the dataset.

While exploring the target variable, I had already dropped the rows containing Unknown values (SEVERITYCODE = 0). There are around 5000 rows having NaN values for the 3 input variables, which also need to be removed.

```
In [298]: #Looking at null values
          dataset.isna().sum()

Out[298]: OBJECTID           0
          SEVERITYCODE       1
          SEVERITYDESC       0
          INCDATE            0
          WEATHER         4987
          ROADCOND        4906
          LIGHTCOND       5076
          dtype: int64
```

Figure 5: Code snippet and output for NaN values by features

I have removed the rows where all three input variables are NaN, or have a value 'Unknown', as these are of no use to us while modelling. For the remaining rows that contain NaN values for 1 or 2 input variable, I have imputed the Mode values of the variables ('Dry' for ROADCOND, 'Clear' for WEATHER, and 'Daylight' for LIGHTCOND).

Once the missing values have been dealt with, the next step is to balance the dataset. We have seen earlier that the no. of records with 'SEVERITYCODE'= 1 are far more than those with 'SEVERITYCODE'= 2 (approx. twice as many). So, we will need to make them equal to ensure there is no bias in the model. The figure below shows the code used to balance the dataset, and the no. of records by SEVERITYCODE before and after the balancing step. Now we have a clean, balanced dataset with no Missing/Null values.

```
In [307]: dataset["SEVERITYCODE"].value_counts()

Out[307]: 1.0    124071
          2.0     60947
          Name: SEVERITYCODE, dtype: int64

In [308]: #Downsampling for balanced dataset

          dataset_1 = dataset[dataset['SEVERITYCODE'] == 1]
          dataset_2 = dataset[dataset['SEVERITYCODE'] == 2]

          resampled_dataset_1 = dataset_1.sample(n=60947,replace=False,random_state=1)
          dataset_balanced = pd.concat([resampled_dataset_1,dataset_2])

          dataset_balanced["SEVERITYCODE"].value_counts()

Out[308]: 2.0    60947
          1.0    60947
          Name: SEVERITYCODE, dtype: int64
```

Figure 6: Code snippets and output for balancing the target variable values.

2.6 Feature Encoding

The final step before our data is ready to be fed into the model is to convert the categorical input variables into numerical/binary variables. Since the categories are not ordinal, we have to convert them to binary variables. However, there are plenty of categories for each variable. For instance, the variable LIGHTCOND has 9 categories (see figure below). If we encode this variable as it is, it will create 9 more input variables in our model, which is not recommended. So I have decided to reduce the categories by grouping them based on similarity in light conditions.

```
In [311]: dataset_balanced['LIGHTCOND'].value_counts()

Out[311]: Daylight                   79844
          Dark - Street Lights On    32560
          Dusk                        4085
          Dawn                        1764
          Unknown                     1735
          Dark - No Street Lights      973
          Dark - Street Lights Off     768
          Other                        148
          Dark - Unknown Lighting       17
          Name: LIGHTCOND, dtype: int64
```

Figure 7: Code snippet and output of counts for LIGHTCOND categories.

The two categories ‘Daylight’ and ‘Dark - Street Lights On’ account for more than 90% of the total records. The next most no. of records are in the ‘Dusk’ and

'Dawn' categories. Since lighting conditions are more or less similar during dawn and dusk times of the day, I have grouped them together in one category – 'Dawn/Dusk'. All the categories where lighting conditions are dark have been grouped into 'Dark'. So we have 4 major categories mapped according to the below table:

Original Category	Grouped Category
Daylight	Daylight
Dark - Street Lights On	Dark
Dusk	Dawn/Dusk
Dawn	Dawn/Dusk
Unknown	Others (Light)
Dark - No Street Lights	Dark
Dark - Street Lights Off	Dark
Other	Others (Light)
Dark - Unknown Lighting	Dark

Table 3: Category mapping table for LIGHTCOND

Similarly, I have grouped the categories for ROADCOND and WEATHER (refer Appendix B for code and mappings) based on similarity of conditions. The result is a total of 13 categories (from the initial 30 categories across the 3 input variables), meaning that on encoding, we'll get 13 input variables for our model, which is manageable. We could have handled all 30 variables as well, but the very less number of records for some categories meant that these variables would not add anything substantial to our model, and instead, could even make our model prone to overfitting. So it is best that we use only the major categories as input variables.

The categories being finalized, I have then converted them to binary variables using one-hot encoding technique. This gives us the final dataset that can be fed into the model. You can see the 13 categories encoded as different binary variables in the figure below:

```
In [317]: #Using One Hot Encoding
dataset_final = dataset_balanced[['SEVERITYCODE']]
dataset_final = pd.concat([dataset_final,pd.get_dummies(dataset_balanced['LIGHTCOND']),
                                         pd.get_dummies(dataset_balanced['ROADCOND']),
                                         pd.get_dummies(dataset_balanced['WEATHER'])], axis=1)
dataset_final.head()
```

Out[317]:

	SEVERITYCODE	Dark	Dawn/Dusk	Daylight	Others (Light)	Dry	Ice/Snow	Others (Road)	Wet	Clear	Others (Weather)	Overcast	Raining	Snowing
137362	1.0	0	0	1	0	1	0	0	0	1	0	0	0	0
82507	1.0	0	0	1	0	1	0	0	0	1	0	0	0	0
198733	1.0	0	0	1	0	1	0	0	0	1	0	0	0	0
11193	1.0	0	0	1	0	1	0	0	0	1	0	0	0	0
120729	1.0	0	0	1	0	0	0	0	1	0	0	0	1	0

Figure 8: Code Snippet for one-hot encoding and output table of final dataset

3. Methodology

3 Machine Learning techniques have been chosen to predict the severity of accidents.

- K-Nearest Neighbours (KNN) Classification
- Decision Tree Classification
- Logistic Regression

The rationale behind selecting these models and their deployment is discussed in detail in the following sections. But before proceeding with the models, we need to separate the input and target variables, and split our dataset into a Training set (on which our models will learn) and a Test set (which will be used to validate the accuracy of the models). The All the 3 models will use the same samples for Train and Test. I have decided to use a 70-30 split, meaning 30% of our dataset will be kept aside for testing, and the remaining will be used for training the models.

```
In [319]: #Split into train and test
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=4)
```

```
In [320]: print('Train data has shape', X_train.shape)
print('Test data has shape', X_test.shape)
```

```
Train data has shape (85325, 13)
Test data has shape (36569, 13)
```

Figure 9: Code Snippet and output showing Train and Test set counts

Typically, we would have to normalize the data before feeding it into the models, however since all our variables are binary, normalization is not required. We can proceed with the model deployment.

3.1 K-Nearest Neighbours (KNN) Classification

In k-Nearest Neighbours (KNN) Classification, an object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its 'k' nearest neighbours.

In the context of our problem statement, assume a 13-dimensional space (one dimension for each input variable). Based on the current weather, road and light conditions, a data point will be generated which will have 13-dimensional coordinates and exist somewhere in that 13-dimensional space. The kNN algorithm will compare it with 'k' closest data points (from our Train set) in the 13-dimensional space, and whatever is the most common severity code among those k data points, the same will be assigned to this data point. The algorithm works on the principle that the closer two data points are, the more similar they'll be in nature.

Below is the implementation of the classification algorithm. I have initially taken an arbitrary value of k=6 and run the model. And then, I have implemented a code to run the model for different values of k, and identified which value of k gives us the highest accuracy. The graph of accuracy vs 'k' is shown in the below figure:

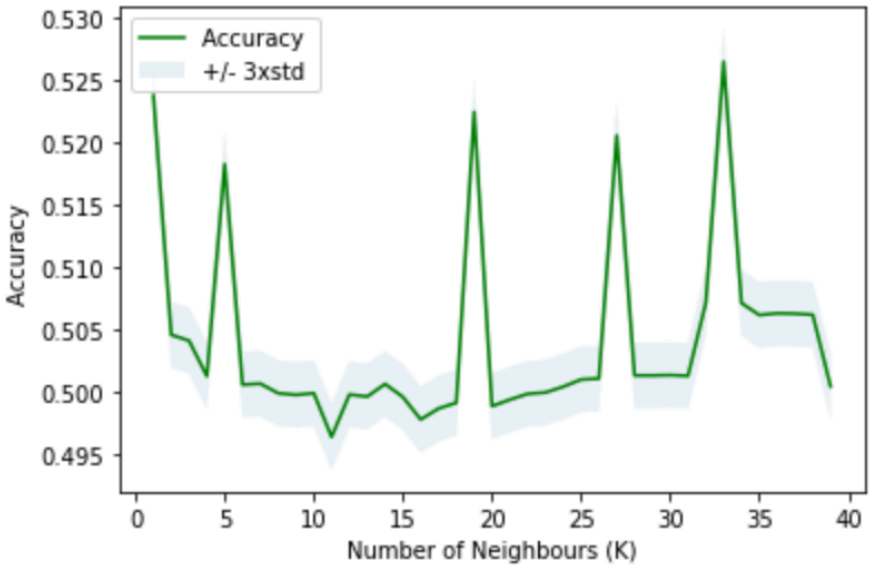


Figure 10: Graph plot showing the KNN accuracy values for different k values

For k=6, we get a Test accuracy value of 0.5006, i.e. 50.1%. From the graph, we infer that the accuracy fluctuates marginally between 49.8% and 52.6% as the value of k changes, and the maximum accuracy we get is **52.65% at k=33**.

Ideally, we would want our accuracy to be upwards of 70% for declaring our model to be a good fit, however based on the data we have, it is very difficult to find a model that will give such a high accuracy. Our data is very well-balanced and the target variable (severity) is well-distributed across the input variables (weather, road and light conditions), as can be inferred from the box plots as well. It is very difficult to find distinct patterns in such data, nevertheless, using the KNN algorithm, we can predict the severity of accidents about 53% accurately.

For detailed code and outputs, please refer Appendix C

3.2 Decision Tree Classification

A decision-tree uses a tree-like model to show each decision and its expected outcome. Starting with a root node, each additional variable adds a new level to the tree, with its categories creating branches for that level. Thus, every unique combination of variables is considered to be a unique decision, and assigned an expected value based on the frequency of its occurrences in the training set.

In the context of our problem statement, we shall have a decision tree that looks somewhat like this (illustrative only, the actual decision tree shall be plotted after the model is implemented)

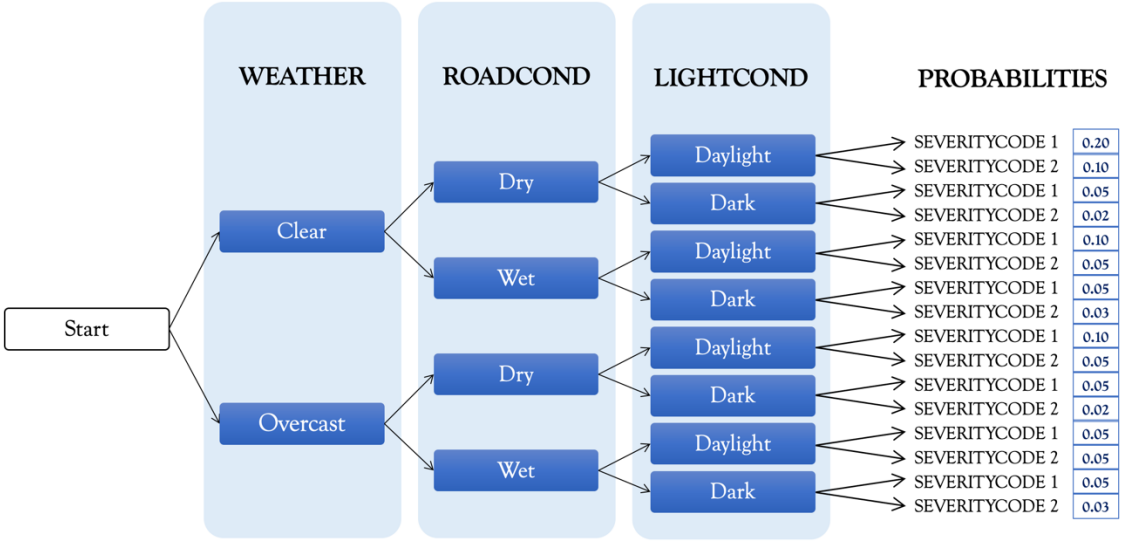


Figure 11: Illustrative Decision Tree based on input variables

Below is the implementation of the classification algorithm. The maximum depth of the tree has been chosen 13 because there are 13 input variables.

```
In [92]: from sklearn.tree import DecisionTreeClassifier
         from sklearn import tree


         model_dtree = DecisionTreeClassifier(criterion="entropy", max_depth = 6)
         model_dtree.fit(X_train,y_train)
         yhat_dtree = model_dtree.predict(X_test)
         print("Decision Tree Classifier executed successfully")

         Decision Tree Classifier executed successfully
```

Figure 12: Code Snippet for implementing the decision tree algorithm

The actual decision tree looks something like this (below). You can find a full-size image in the Github repository here. You can also find a text representation of the tree in Appendix D.

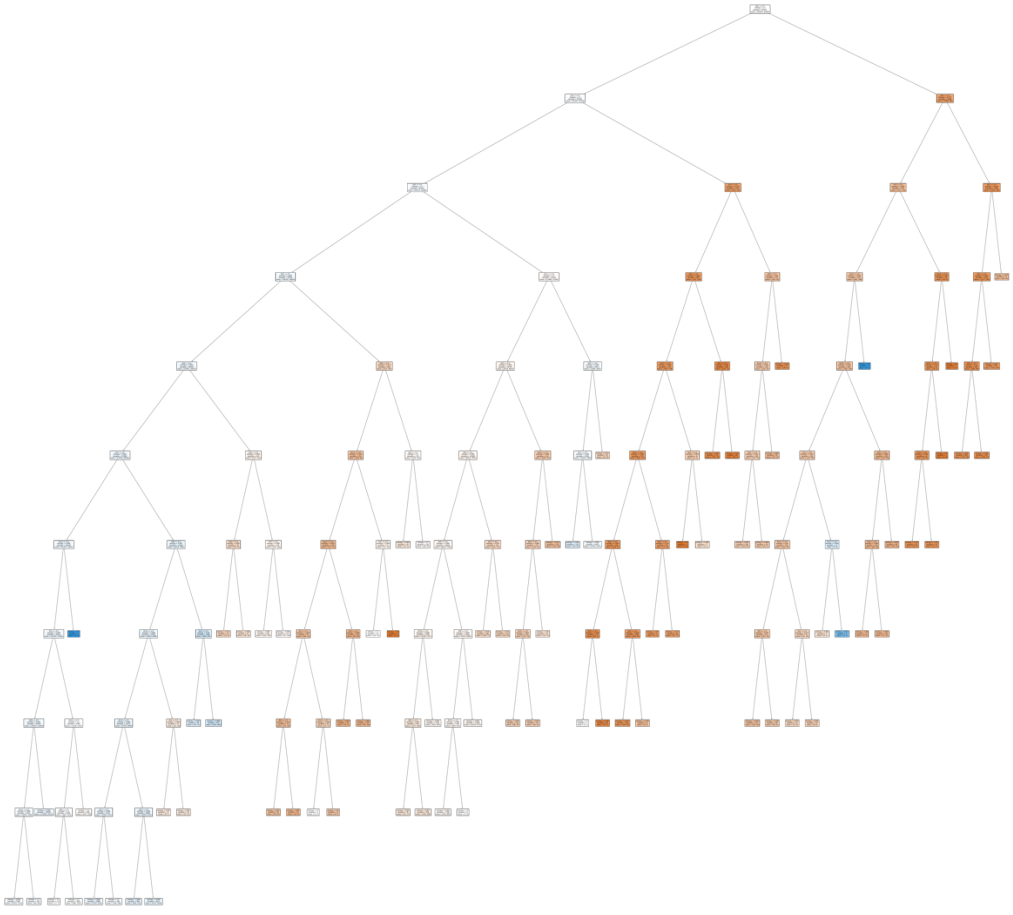


Figure 13: Actual Decision Tree for predicting accident severity created by the model

The decision tree classifier gives us a **Test accuracy of 0.5328, or 53.3%**. This is marginally better than the Test accuracy we got for KNN, but still not sufficient for us to declare it as a good fitting model.

3.3 Logistic Regression

Logistic regression is a statistical model that uses a logistic function to model binary dependent variables, and predict the probability of events having 2 outcomes, such as win/loss, pass/fail, alive/dead, yes/no, etc. Although Logistic regression is not a classification algorithm (it predicts the probability of an outcome – a value between 0 and 1), it can be used as a classifier by deciding a threshold value, and assigning classes to data points based on whether their probability values are higher or lower than the threshold value. For our dataset, Logistic regression is a good model to apply because our target variable is binary.

Below is the implementation of the Logistic regression model. The regularization parameter C has been chosen as 1 since all the input variables are binary, and there is no need for regularization.

```
In [327]: from sklearn.linear_model import LogisticRegression
          from sklearn.metrics import confusion_matrix
          from sklearn.metrics import log_loss

          model_lr = LogisticRegression(C=1, solver='liblinear')
          model_lr.fit(X_train,y_train)
          yhat_lr = model_lr.predict(X_test)
          yhat_lr_prob = model_lr.predict_proba(X_test)
          print("Logistic Regression executed successfully")

          Logistic Regression executed successfully

In [328]: print("LR Train Accuracy: ", metrics.accuracy_score(y_train, model_lr.predict(X_train)))
          print("LR Test Accuracy: ", metrics.accuracy_score(y_test, yhat_lr))
          print("Logarithmic Loss Score: ", log_loss(y_test,yhat_lr_prob))
          print("Confusion Matrix: \n", confusion_matrix(y_test, yhat_lr, labels=[1,2]))

          LR Train Accuracy:  0.5328684441840024
          LR Test Accuracy:  0.53315649867374
          Logarithmic Loss Score:  0.6843327483408435
          Confusion Matrix:
           [[ 5158 13156]
           [ 3916 14339]]
```

Figure 14: Code snippet for LR, with output accuracy values and confusion matrix

The Logistic regression model gives us a **Test accuracy of 0.5332, or 53.3%**. This is marginally better than the Test accuracy we got for the KNN and Decision Tree classifiers, but still not sufficient for us to declare it as a good fitting model. The confusion matrix for the test data is shown above, we can observe that the model has correctly predicted **5,158 cases of Severity 1 and 14,339 cases of severity 2**. We also get a **Logarithmic Loss score of 0.684**, which is decent.

3.4 Model Evaluation

We have already calculated the Test accuracy for the 3 models. Here, we will look at additional metrics to compare them and determine which is the most suitable for our data.

The **Jaccard Similarity Index** measures the similarity between two sets of data. It can range from 0 to 1. The higher the number, the more similar the two sets of data.

The **F1 Score** (also known as **Sørensen-Dice coefficient)** is another metric to measure the accuracy of a test. It is the harmonic mean of the precision and recall values calculated for the model. It can range from 0 to 1. The higher the number, higher is the precision and recall.

Below is the implementation of the metrics, and the output:

In [119]:
```
# Importing the libraries
from sklearn.metrics import jaccard_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss

#Computing jaccard and f1 score for KNN
knn_jaccard = jaccard_score(y_test, yhat_knn)
knn_f1_score = f1_score(y_test, yhat_knn, average='weighted')
knn_acc = mean_acc.max()

#Computing jaccard and f1 score for Decision Tree
dtree_jaccard = jaccard_score(y_test, yhat_dtree)
dtree_f1_score = f1_score(y_test, yhat_dtree, average='weighted')

#Computing jaccard, f1 score and Logarithmic Loss for Logistic Regression
lr_jaccard = jaccard_score(y_test, yhat_lr)
lr_f1_score = f1_score(y_test, yhat_lr, average='weighted')
lr_log_loss = log_loss(y_test, yhat_lr_prob)
```

In [120]:
```
# Tabulating a summary of the results
report = {'Test Accuracy': [mean_acc.max(), metrics.accuracy_score(y_test, yhat_dtree), metrics.accuracy_score(y_test,
          'Jaccard': [knn_jaccard, dtree_jaccard, lr_jaccard],
          'F1-score':[knn_f1_score, dtree_f1_score, lr_f1_score],
          'LogLoss': ['NA', 'NA', lr_log_loss]}
df_report = pd.DataFrame(report, index =['KNN', 'Decision Tree', 'LogisticRegression'])
df_report
```

Out[120]:

	Test Accuracy	Jaccard	F1-score	LogLoss
KNN	0.526484	0.397626	0.487656	NA
Decision Tree	0.532801	0.233994	0.502151	NA
LogisticRegression	0.533156	0.232029	0.501550	0.684333

Figure 15: Code snippet for Jaccard Similarity Score and F1- Score computation, and tabular summary of the metrics for the 3 models

4. Results

Reiterating the results of the three models in the table below. We were able to achieve fairly similar accuracy with the three models – **KNN Classification (Max accuracy of 52.65% at K=33), Decision Tree Classification** (53.28%) and **Logistic Regression** (53.32%). We had also computed Jaccard Similarity and F1 Scores for the 3 models, and the **highest Jaccard was observed for KNN**, while F1 Scores were again fairly similar.

	Test Accuracy	Jaccard	F1-Score	Log Loss
KNN	52.65%	0.40	0.49	NA
Decision Tree	53.28%	0.23	0.50	NA
Logistic Regression	53.32%	0.23	0.50	0.68

Table 4: Tabular summary of metrics for the 3 models

5. Discussion & Recommendations

From the exploratory data analysis, it was clear that our target variable was well distributed and that there was little correlation between our three input variables – Light conditions, Road conditions and Weather conditions and the severity of accidents. When the conditions were changed, the proportion of accidents of low severity vs high severity did not change much. Hence the accuracy values, Jaccard and F1 scores obtained for the three models are not surprising to me. All the three models are able to predict the severity of accidents with a >50% accuracy, meaning all of them can be used to classify accidents. However, the **Jaccard score for KNN Classification is significantly higher than the other two, and hence, I would recommend this model to be used for predicting the severity of accidents.**

Future Scope

I would suggest that the analysis will be more effective and deliver much-improved insights if another set of data is added to the analysis, that is – **The number of cars that are on the road during different weather, road and light conditions.** Currently from our dataset, we infer the fact that in rainy conditions, x_1 accidents happen which are less severe, and x_2 accidents happen which are more severe.

And in dry clear conditions, y_1 accidents happen which are less severe, and y_2 accidents happen which are more severe. Since x_1/x_2 and y_1/y_2 are fairly similar, we cannot really predict whether the conditions are causing more severe accidents. If we have the data of the number of vehicles on the road in different conditions, it will give us an idea of **what percentage of vehicles on the road get involved into an accident.** It is quite probable that in rainy conditions, the number of vehicles on the road is less, and $x_1 + x_2$ (total accidents in rainy conditions) is a significant percentage of that number, whereas $y_1 + y_2$ (total accidents in clear, dry conditions) may be a very small percentage of the number of vehicles on road in perfect driving conditions. **Rather than directly looking at the severity of accidents, we will be able to answer a more fundamental question about the likelihood of an accident happening if the conditions are adverse.** In the future, this data should be added to the accidents data and a combined analysis should be performed.

6. Conclusion

Our problem statement was to **predict the severity of potential accidents based on the weather, road and light conditions**. The approach was to extract the data from the mentioned source, identify the **target and input variables**, and perform some **exploratory data analysis** to understand relationships. Once this was done, perform the data cleaning processes and prepare the dataset to be fed into the models. I selected 3 models – **KNN, Decision Tree and Logistic Regression** to perform the classification. The data was split into training and test sets, and the models were first trained on the training set, then evaluated on the test set. The metrics used for evaluation were **Test accuracy, Jaccard Similarity Score, F1 Score and Logarithmic Loss Coefficient (for Logistic Regression).** The results show that all three models are able to predict the severity of accidents with a **fairly similar accuracy rate (52-53%), with KNN having a significantly higher Jaccard Score than the other two**. All three models can be used as classifiers for now **(KNN Recommended because of better Jaccard Score),** but a more effective classification can be performed in the future if additional data is added to the analysis.
The Jupyter Notebook, with the complete code, can be accessed here.

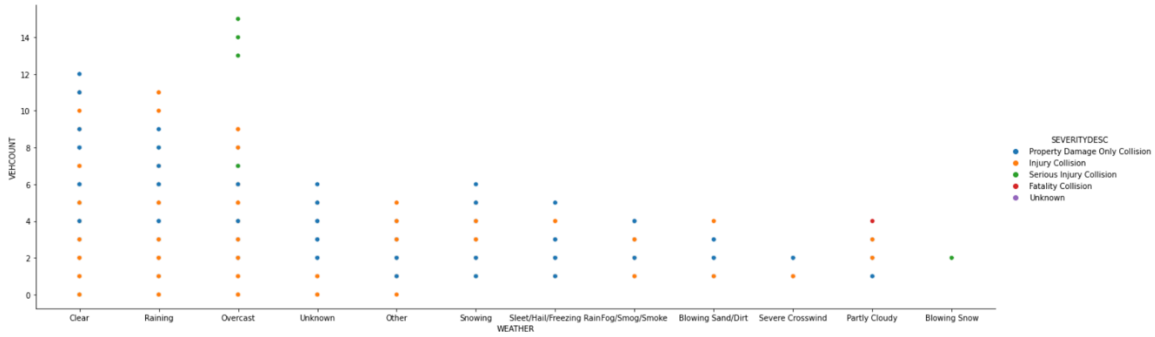
7. Appendices

7.1 Appendix A

Relation plots and Box plots to understand variable relationships, data distributions and outliers.

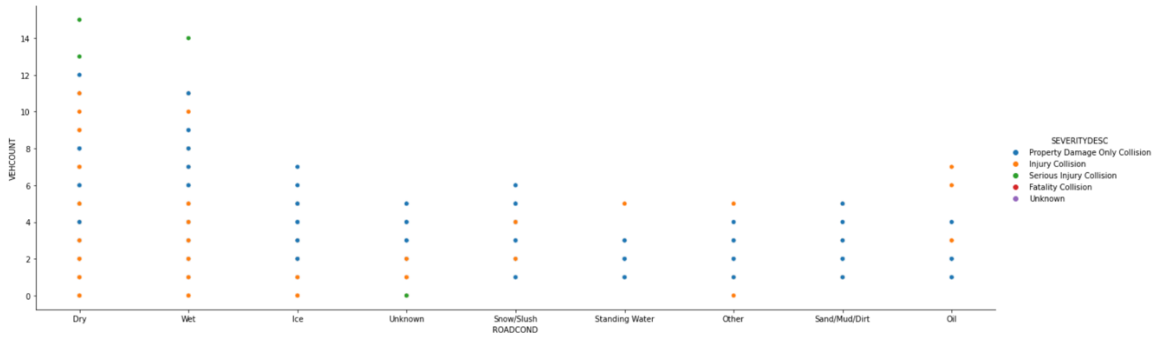
Code snippet and output rel-plot for **weather conditions vs vehicle count** by severity

```
In [283]: # Weather vs Vehicle Count by Severity of Accidents
          sns.relplot(x='WEATHER', y='VEHCOUNT', data=dataset, height = 6, aspect=3, hue='SEVERITYDESC')

Out[283]: <seaborn.axisgrid.FacetGrid at 0x7f9000513df0>
```



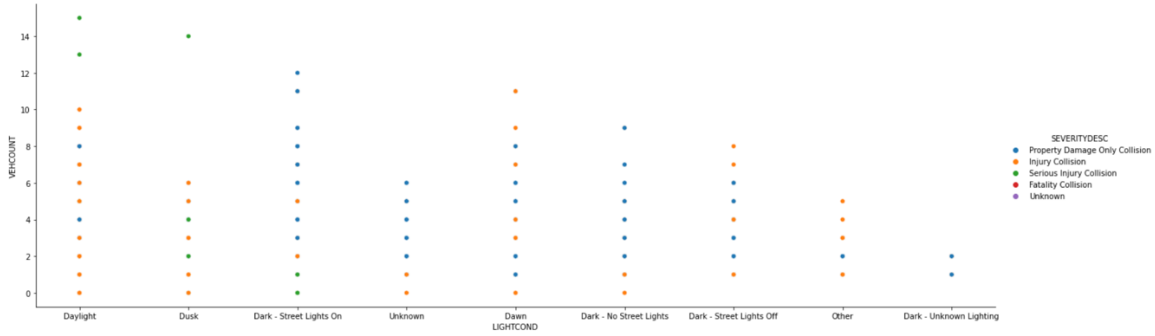
Code snippet and output rel-plot for **road conditions vs vehicle count** by severity

```
In [285]: # Road Conditions vs Vehicle Count by Severity of Accidents
          sns.relplot(x='ROADCOND', y='VEHCOUNT', data=dataset, height = 6, aspect=3, hue='SEVERITYDESC')

Out[285]: <seaborn.axisgrid.FacetGrid at 0x7f8fdb4a49a0>
```



Code snippet and output rel-plot for **light conditions vs vehicle count** by severity

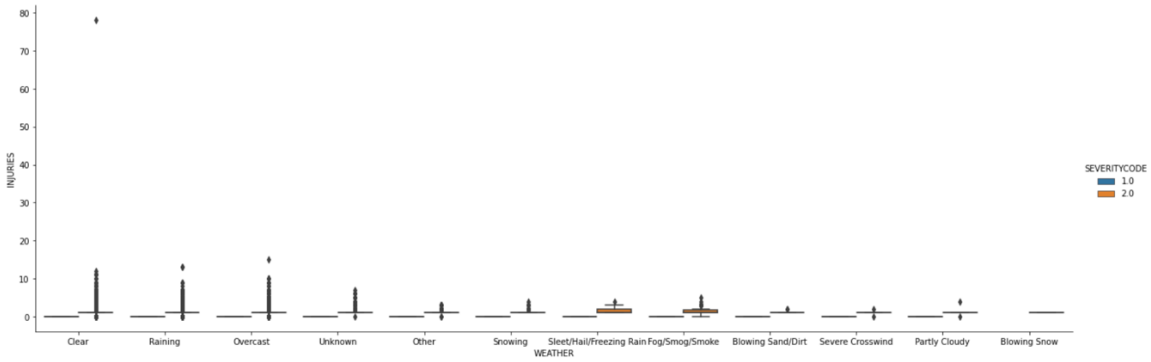
```
In [291]: # Light Conditions vs Vehicle Count by Severity of Accidents
          sns.relplot(x='LIGHTCOND', y='VEHCOUNT', data=dataset, height = 6, aspect=3, hue='SEVERITYDESC')

Out[291]: <seaborn.axisgrid.FacetGrid at 0x7f8fcbf95a30>
```



Code snippet and output box-plot for **weather conditions vs injuries** by severity

```
In [295]: # Weather vs Injuries by Severity of Accidents
          sns.catplot(x='WEATHER', y='INJURIES', data=dataset, kind = 'box', height = 6, aspect=3, hue='SEVERITYCODE')
```

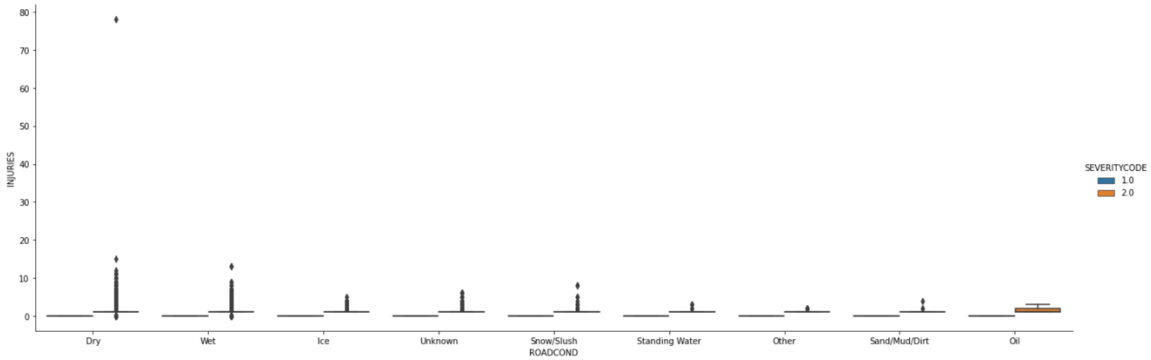
```
Out[295]: <seaborn.axisgrid.FacetGrid at 0x7f8fd01453d0>
```



Code snippet and output box-plot for **road conditions vs injuries** by severity

```
In [293]: # Road Conditions vs Injuries by Severity of Accidents
          sns.catplot(x='ROADCOND', y='INJURIES', data=dataset, kind = 'box', height = 6, aspect=3, hue='SEVERITYCODE')
```

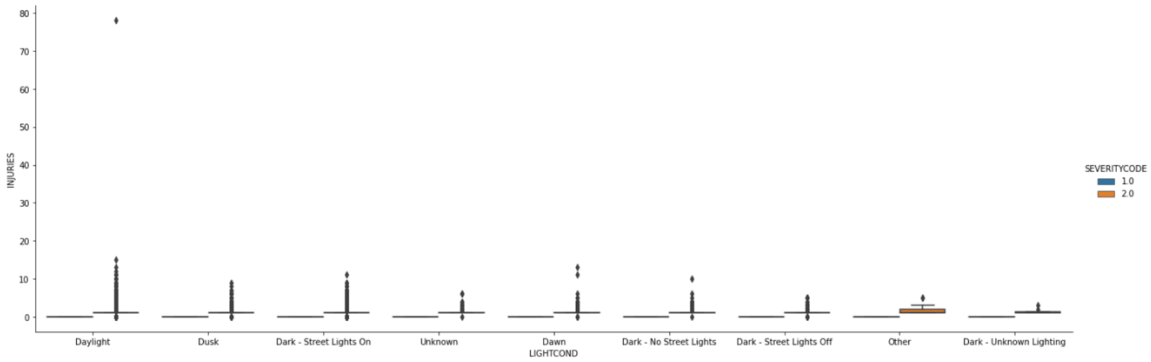
```
Out[293]: <seaborn.axisgrid.FacetGrid at 0x7f8fd17c68e0>
```



Code snippet and output box-plot for **light conditions vs injuries** by severity

```
In [296]: # Light Conditions vs Injuries by Severity of Accidents
          sns.catplot(x='LIGHTCOND', y='INJURIES', data=dataset, kind = 'box', height = 6, aspect=3, hue='SEVERITYCODE')
```

```
Out[296]: <seaborn.axisgrid.FacetGrid at 0x7f8fd49e00d0>
```



7.2 Appendix B

Grouping values of input variables into broader categories

Variable 1 – LIGHTCOND

Code Snippet and Output values for **LIGHTCOND** categories (before and after grouping)

```
In [311]: dataset_balanced['LIGHTCOND'].value_counts()

Out[311]: Daylight                    79844
          Dark - Street Lights On     32560
          Dusk                        4085
          Dawn                        1764
          Unknown                     1735
          Dark - No Street Lights      973
          Dark - Street Lights Off     768
          Other                        148
          Dark - Unknown Lighting       17
          Name: LIGHTCOND, dtype: int64
```

```
In [312]: # Daylight                  ------> Daylight
          # Dark - Street Lights On   ------> Dark
          # Dusk                      ------> Dawn/Dusk
          # Unknown                   ------> Others (Light)
          # Dawn                      ------> Dawn/Dusk
          # Dark - No Street Lights   ------> Dark
          # Dark - Street Lights Off  ------> Dark
          # Other                     ------> Others (Light)
          # Dark - Unknown Lighting   ------> Dark

          dataset_balanced["LIGHTCOND"].replace(to_replace=['Dark - Street Lights On',
                                                            'Dark - No Street Lights',
                                                            'Dark - Street Lights Off',
                                                            'Dark - Unknown Lighting',
                                                            'Other',
                                                            'Unknown',
                                                            'Dawn',
                                                            'Dusk'],
                                                value=['Dark','Dark','Dark','Dark','Others (Light)',
                                                       'Others (Light)','Dawn/Dusk','Dawn/Dusk'],inplace=True)

          dataset_balanced['LIGHTCOND'].value_counts()

Out[312]: Daylight          79844
          Dark              34318
          Dawn/Dusk          5849
          Others (Light)     1883
          Name: LIGHTCOND, dtype: int64
```

Mapping Table for **LIGHTCOND**

Original Category	Grouped Category
Daylight	Daylight
Dark - Street Lights On	Dark
Dusk	Dawn/Dusk
Dawn	Dawn/Dusk
Unknown	Others (Light)
Dark - No Street Lights	Dark
Dark - Street Lights Off	Dark
Other	Others (Light)
Dark - Unknown Lighting	Dark

Variable 2 – ROADCOND

Code Snippet and Output values for **ROADCOND** categories (before and after grouping)

```
In [313]: dataset_balanced['ROADCOND'].value_counts()
```

```
Out[313]: Dry              85229
          Wet              32401
          Unknown           2643
          Ice                782
          Snow/Slush         580
          Other               92
          Standing Water      72
          Oil                 48
          Sand/Mud/Dirt       47
          Name: ROADCOND, dtype: int64
```

```
In [314]: # Dry             ------> Dry
          # Wet             ------> Wet
          # Unknown         ------> Others (Road)
          # Ice             ------> Ice/Snow
          # Snow/Slush      ------> Ice/Snow
          # Other           ------> Others (Road)
          # Standing Water  ------> Wet
          # Oil             ------> Others (Road)
          # Sand/Mud/Dirt   ------> Others (Road)

          dataset_balanced["ROADCOND"].replace(to_replace=['Unknown',
                                                           'Ice',
                                                           'Snow/Slush',
                                                           'Other',
                                                           'Standing Water',
                                                           'Oil',
                                                           'Sand/Mud/Dirt'],
                                               value=['Others (Road)','Ice/Snow','Ice/Snow','Others (Road)','Wet',
                                                      'Others (Road)','Others (Road)'],inplace=True)

          dataset_balanced['ROADCOND'].value_counts()
```

```
Out[314]: Dry              85229
          Wet              32473
          Others (Road)     2830
          Ice/Snow          1362
          Name: ROADCOND, dtype: int64
```

Mapping Table for **ROADCOND**

Original Category	Grouped Category
Dry	Dry
Wet	Wet
Unknown	Others (Road)
Ice	Ice/Snow
Snow/Slush	Ice/Snow
Other	Others (Road)
Standing Water	Wet
Oil	Others (Road)
Sand/Mud/Dirt	Others (Road)

Variable 3 – WEATHER

Code Snippet and Output values for **WEATHER** categories (before and after grouping)

```
In [315]: dataset_balanced['WEATHER'].value_counts()
```

```
Out[315]: Clear                       76143
          Raining                     22781
          Overcast                    18779
          Unknown                      2642
          Snowing                       538
          Other                         501
          Fog/Smog/Smoke                373
          Sleet/Hail/Freezing Rain       76
          Blowing Sand/Dirt              38
          Severe Crosswind               15
          Partly Cloudy                   7
          Blowing Snow                    1
          Name: WEATHER, dtype: int64
```

```
In [316]: # Clear                     ------> Clear
          # Raining                   ------> Raining
          # Overcast                  ------> Overcast
          # Unknown                   ------> Others (Weather)
          # Snowing                   ------> Snowing
          # Other                     ------> Others (Weather)
          # Fog/Smog/Smoke            ------> Others (Weather)
          # Sleet/Hail/Freezing Rain  ------> Snowing
          # Blowing Sand/Dirt         ------> Others (Weather)
          # Severe Crosswind          ------> Others (Weather)
          # Partly Cloudy             ------> Others (Weather)
          # Blowing Snow              ------> Snowing

          dataset_balanced["WEATHER"].replace(to_replace=['Unknown',
                                                          'Other',
                                                          'Fog/Smog/Smoke',
                                                          'Sleet/Hail/Freezing Rain',
                                                          'Blowing Sand/Dirt',
                                                          'Severe Crosswind',
                                                          'Partly Cloudy',
                                                          'Blowing Snow'],
                                              value=['Others (Weather)','Others (Weather)','Snowing',
                                                     'Others (Weather)','Others (Weather)','Others (Weather)','Snowing'],
                                              inplace=True)

          dataset_balanced['WEATHER'].value_counts()
```

```
Out[316]: Clear              76143
          Raining            22781
          Overcast           18779
          Others (Weather)    3576
          Snowing              615
          Name: WEATHER, dtype: int64
```

Mapping Table for **WEATHER**

Original Category	Grouped Category
Clear	Clear
Raining	Raining
Overcast	Overcast
Unknown	Others (Weather)
Snowing	Snowing
Other	Others (Weather)
Fog/Smog/Smoke	Others (Weather)
Sleet/Hail/Freezing Rain	Snowing
Blowing Sand/Dirt	Others (Weather)
Severe Crosswind	Others (Weather)
Partly Cloudy	Others (Weather)
Blowing Snow	Others (Weather)

7.3 Appendix C

K-Nearest Neighbours (KNN) Classification

Code Snippet and Accuracy values for initial value of k=6

In [321]:
```
# Import library
from sklearn.neighbors import KNeighborsClassifier

k = 6 #inital value of k assumed to be 6
model_knn = KNeighborsClassifier(n_neighbors = k)
model_knn.fit(X_train,y_train)
yhat_knn = model_knn.predict(X_test)
print("KNN executed successfully")
```

```
KNN executed successfully
```

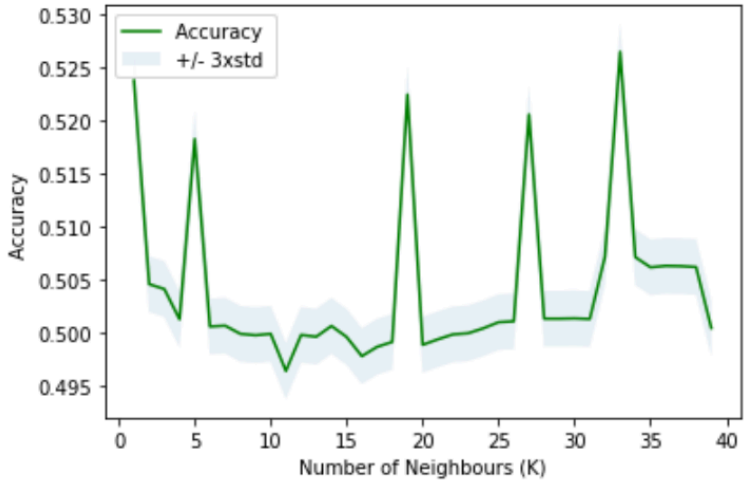
In [322]:
```
from sklearn import metrics

print("KNN Train Accuracy: ", metrics.accuracy_score(y_train, model_knn.predict(X_train)))
print("KNN Test Accuracy: ", metrics.accuracy_score(y_test, yhat_knn))
```

```
KNN Train Accuracy:  0.5019865221213009
KNN Test Accuracy:  0.500615275233121
```

Code Snippet for running KNN with value of k ranging from 1 to 39

In [323]:
```
#Changing K to find optimal value

k_optimal = 40
mean_acc = np.zeros((k_optimal-1))
std_acc = np.zeros((k_optimal-1))
Confusion_Matrix = [];
for n in range(1,k_optimal):
    model_knn = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat_knn = model_knn.predict(X_test)

    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat_knn)
    std_acc[n-1]=np.std(yhat_knn==y_test)/np.sqrt(yhat_knn.shape[0])

mean_acc

plt.plot(range(1,k_optimal),mean_acc,'g')
plt.fill_between(range(1,k_optimal),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
plt.legend(('Accuracy ', '+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbours (K)')
plt.show()

mean_acc.max()
```

Graph plot showing the KNN accuracy values for different k



Accuracy values for k = 1 to 39

```
In [324]: for i,j in zip(range(1,k_optimal),mean_acc):
              print("k:",i,"acc:",j)
```

```
k: 1 acc: 0.5237769695643851
k: 2 acc: 0.504607727856928
k: 3 acc: 0.5041428532363478
k: 4 acc: 0.5012989143810331
k: 5 acc: 0.5182805108151713
k: 6 acc: 0.500615275233121
k: 7 acc: 0.5006973119308704
k: 8 acc: 0.4999316360852088
k: 9 acc: 0.49979490825562634
k: 10 acc: 0.4999316360852088
k: 11 acc: 0.4964314036478985
k: 12 acc: 0.49982225382154283
k: 13 acc: 0.49965818042604393
k: 14 acc: 0.500669966364954
k: 15 acc: 0.49963083486012744
k: 16 acc: 0.49782602750963934
k: 17 acc: 0.49870108561896687
k: 18 acc: 0.4991659602395472
k: 19 acc: 0.5224643824003937
k: 20 acc: 0.4988925045803823
k: 21 acc: 0.49941207033279555
k: 22 acc: 0.4998769449533758
k: 23 acc: 0.4999863272170418
k: 24 acc: 0.500451201837622
k: 25 acc: 0.5010254587218682
k: 26 acc: 0.5011074954196177
k: 27 acc: 0.5205775383521561
k: 28 acc: 0.5013536055128661
k: 29 acc: 0.5013536055128661
k: 30 acc: 0.5013809510787826
k: 31 acc: 0.5013262599469496
k: 32 acc: 0.5071782110530777
k: 33 acc: 0.5264841805901174
k: 34 acc: 0.5071508654871613
k: 35 acc: 0.5061937706800842
k: 36 acc: 0.5063304985096666
k: 37 acc: 0.5063031529437502
k: 38 acc: 0.5062211162460007
k: 39 acc: 0.5004785474035385
```

7.4 Appendix D

Decision Tree Classification

```
|--- feature_6 <= 0.50
|   |--- feature_3 <= 0.50
|   |   |--- feature_0 <= 0.50
|   |   |   |--- feature_5 <= 0.50
|   |   |   |   |--- feature_9 <= 0.50
|   |   |   |   |   |--- feature_7 <= 0.50
|   |   |   |   |   |   |--- feature_12 <= 0.50
|   |   |   |   |   |   |   |--- feature_1 <= 0.50
|   |   |   |   |   |   |   |   |--- feature_8 <= 0.50
|   |   |   |   |   |   |   |   |   |--- feature_11 <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- class: 2.0
|   |   |   |   |   |   |   |   |   |--- feature_11 >  0.50
|   |   |   |   |   |   |   |   |   |   |--- class: 2.0
|   |   |   |   |   |   |   |   |--- feature_8 >  0.50
|   |   |   |   |   |   |   |   |   |--- class: 2.0
|   |   |   |   |   |   |   |--- feature_1 >  0.50
|   |   |   |   |   |   |   |   |--- feature_10 <= 0.50
|   |   |   |   |   |   |   |   |   |--- feature_8 <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |   |   |   |--- feature_8 >  0.50
|   |   |   |   |   |   |   |   |   |   |--- class: 2.0
|   |   |   |   |   |   |   |   |--- feature_10 >  0.50
|   |   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |   |--- feature_12 >  0.50
|   |   |   |   |   |   |   |   |--- class: 2.0
|   |   |   |   |   |--- feature_7 >  0.50
|   |   |   |   |   |   |--- feature_8 <= 0.50
|   |   |   |   |   |   |   |--- feature_12 <= 0.50
|   |   |   |   |   |   |   |   |--- feature_11 <= 0.50
|   |   |   |   |   |   |   |   |   |--- feature_1 <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- class: 2.0
|   |   |   |   |   |   |   |   |   |--- feature_1 >  0.50
|   |   |   |   |   |   |   |   |   |   |--- class: 2.0
|   |   |   |   |   |   |   |   |--- feature_11 >  0.50
|   |   |   |   |   |   |   |   |   |--- feature_2 <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- class: 2.0
|   |   |   |   |   |   |   |   |   |--- feature_2 >  0.50
|   |   |   |   |   |   |   |   |   |   |--- class: 2.0
|   |   |   |   |   |   |   |--- feature_12 >  0.50
|   |   |   |   |   |   |   |   |--- feature_2 <= 0.50
|   |   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |   |   |--- feature_2 >  0.50
|   |   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |--- feature_8 >  0.50
|   |   |   |   |   |   |   |--- feature_2 <= 0.50
```

```
|   |   |   |   |   |   |   |   |--- class: 2.0
|   |   |   |   |   |   |   |--- feature_2 >  0.50
|   |   |   |   |   |   |   |   |--- class: 2.0
|   |   |   |   |--- feature_9 >  0.50
|   |   |   |   |   |--- feature_2 <= 0.50
|   |   |   |   |   |   |--- feature_4 <= 0.50
|   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |--- feature_4 >  0.50
|   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |--- feature_2 >  0.50
|   |   |   |   |   |   |--- feature_7 <= 0.50
|   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |--- feature_7 >  0.50
|   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |--- feature_5 >  0.50
|   |   |   |   |--- feature_8 <= 0.50
|   |   |   |   |   |--- feature_11 <= 0.50
|   |   |   |   |   |   |--- feature_12 <= 0.50
|   |   |   |   |   |   |   |--- feature_1 <= 0.50
|   |   |   |   |   |   |   |   |--- feature_9 <= 0.50
|   |   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |   |   |--- feature_9 >  0.50
|   |   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |   |--- feature_1 >  0.50
|   |   |   |   |   |   |   |   |--- feature_10 <= 0.50
|   |   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |   |   |--- feature_10 >  0.50
|   |   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |--- feature_12 >  0.50
|   |   |   |   |   |   |   |--- feature_2 <= 0.50
|   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |   |--- feature_2 >  0.50
|   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |--- feature_11 >  0.50
|   |   |   |   |   |   |--- feature_1 <= 0.50
|   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |--- feature_1 >  0.50
|   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |--- feature_8 >  0.50
|   |   |   |   |   |--- feature_2 <= 0.50
|   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |--- feature_2 >  0.50
|   |   |   |   |   |   |--- class: 1.0
|   |   |--- feature_0 >  0.50
|   |   |   |--- feature_11 <= 0.50
|   |   |   |   |--- feature_5 <= 0.50
|   |   |   |   |   |--- feature_9 <= 0.50
|   |   |   |   |   |   |--- feature_4 <= 0.50
|   |   |   |   |   |   |   |--- feature_10 <= 0.50
|   |   |   |   |   |   |   |   |--- feature_8 <= 0.50
|   |   |   |   |   |   |   |   |   |--- class: 1.0
```

```
|   |   |   |   |   |   |   |   |--- feature_8 >  0.50
|   |   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |   |   |--- feature_10 >  0.50
|   |   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |   |--- feature_4 >  0.50
|   |   |   |   |   |   |   |   |--- feature_8 <= 0.50
|   |   |   |   |   |   |   |   |   |--- feature_12 <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |   |   |   |--- feature_12 >  0.50
|   |   |   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |   |   |--- feature_8 >  0.50
|   |   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |--- feature_9 >  0.50
|   |   |   |   |   |   |   |--- feature_7 <= 0.50
|   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |   |--- feature_7 >  0.50
|   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |--- feature_5 >  0.50
|   |   |   |   |   |   |--- feature_12 <= 0.50
|   |   |   |   |   |   |   |--- feature_9 <= 0.50
|   |   |   |   |   |   |   |   |--- feature_10 <= 0.50
|   |   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |   |   |--- feature_10 >  0.50
|   |   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |   |--- feature_9 >  0.50
|   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |--- feature_12 >  0.50
|   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |--- feature_11 >  0.50
|   |   |   |   |   |--- feature_5 <= 0.50
|   |   |   |   |   |   |--- feature_7 <= 0.50
|   |   |   |   |   |   |   |--- class: 2.0
|   |   |   |   |   |--- feature_7 >  0.50
|   |   |   |   |   |   |--- class: 2.0
|   |   |   |   |--- feature_5 >  0.50
|   |   |   |   |   |--- class: 1.0
|   |--- feature_3 >  0.50
|   |   |--- feature_7 <= 0.50
|   |   |   |--- feature_9 <= 0.50
|   |   |   |   |--- feature_11 <= 0.50
|   |   |   |   |   |--- feature_10 <= 0.50
|   |   |   |   |   |   |--- feature_8 <= 0.50
|   |   |   |   |   |   |   |--- feature_5 <= 0.50
|   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |   |--- feature_5 >  0.50
|   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |--- feature_8 >  0.50
|   |   |   |   |   |   |   |--- feature_5 <= 0.50
|   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |   |--- feature_5 >  0.50
|   |   |   |   |   |   |   |   |--- class: 1.0
```

```
|   |   |   |   |   |--- feature_10 >  0.50
|   |   |   |   |   |   |--- feature_4 <= 0.50
|   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |--- feature_4 >  0.50
|   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |--- feature_11 >  0.50
|   |   |   |   |   |--- feature_4 <= 0.50
|   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |--- feature_4 >  0.50
|   |   |   |   |   |   |--- class: 1.0
|   |   |   |--- feature_9 >  0.50
|   |   |   |   |--- feature_5 <= 0.50
|   |   |   |   |   |--- class: 1.0
|   |   |   |   |--- feature_5 >  0.50
|   |   |   |   |   |--- class: 1.0
|   |   |--- feature_7 >  0.50
|   |   |   |--- feature_9 <= 0.50
|   |   |   |   |--- feature_10 <= 0.50
|   |   |   |   |   |--- feature_8 <= 0.50
|   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |--- feature_8 >  0.50
|   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |--- feature_10 >  0.50
|   |   |   |   |   |--- class: 1.0
|   |   |   |--- feature_9 >  0.50
|   |   |   |   |--- class: 1.0
|--- feature_6 >  0.50
|   |--- feature_9 <= 0.50
|   |   |--- feature_3 <= 0.50
|   |   |   |--- feature_12 <= 0.50
|   |   |   |   |--- feature_10 <= 0.50
|   |   |   |   |   |--- feature_1 <= 0.50
|   |   |   |   |   |   |--- feature_11 <= 0.50
|   |   |   |   |   |   |   |--- feature_0 <= 0.50
|   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |   |--- feature_0 >  0.50
|   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |--- feature_11 >  0.50
|   |   |   |   |   |   |   |--- feature_0 <= 0.50
|   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |   |--- feature_0 >  0.50
|   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |--- feature_1 >  0.50
|   |   |   |   |   |   |--- feature_11 <= 0.50
|   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |--- feature_11 >  0.50
|   |   |   |   |   |   |   |--- class: 2.0
|   |   |   |   |--- feature_10 >  0.50
|   |   |   |   |   |--- feature_0 <= 0.50
|   |   |   |   |   |   |--- feature_2 <= 0.50
|   |   |   |   |   |   |   |--- class: 1.0
```

```
|   |   |   |   |   |   |--- feature_2 >  0.50
|   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |---  feature_0 >  0.50
|   |   |   |   |   |   |--- class: 1.0
|   |   |   |---  feature_12 >  0.50
|   |   |   |   |--- class: 2.0
|   |   |--- feature_3 >  0.50
|   |   |   |--- feature_12 <= 0.50
|   |   |   |   |--- feature_11 <= 0.50
|   |   |   |   |   |--- feature_10 <= 0.50
|   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |--- feature_10 >  0.50
|   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |--- feature_11 >  0.50
|   |   |   |   |   |--- class: 1.0
|   |   |   |--- feature_12 >  0.50
|   |   |   |   |--- class: 1.0
|   |--- feature_9 >  0.50
|   |   |--- feature_1 <= 0.50
|   |   |   |--- feature_2 <= 0.50
|   |   |   |   |--- feature_3 <= 0.50
|   |   |   |   |   |--- class: 1.0
|   |   |   |   |--- feature_3 >  0.50
|   |   |   |   |   |--- class: 1.0
|   |   |   |--- feature_2 >  0.50
|   |   |   |   |--- class: 1.0
|   |   |--- feature_1 >  0.50
|   |   |   |--- class: 1.0
```