CAPSTONE PROJECT
CAR ACCIDENT SEVERITY IN SEATTLE CITY

By Sam Joseph

1. Introduction

1.1 Background

Seattle, a city in US, is surrounded by water, mountains and evergreen forests, and contains thousands of acres of parkland. Washington State's largest city, it's home to a large tech industry, with Microsoft and Amazon headquartered in its metropolitan area.

Seattle is known to have well established roads network that caters to heavy traffic. There is no shortage of things to do and places to see in this metropolis, from the thriving culinary scene to the iconic Space Needle

1.2 Business Problem

This project aims to *reduce the severity of accidents* in the city of Seattle; hence we need to build an algorithm to predict the severity of an accident based on the current weather, road and visibility conditions. The main data attributes which we will use for the analysis will be

1. Weather Condition
2. Car Speeding
3. Light Condition
4. Road Condition

1.3 Target Audience

This project will benefit the drivers in Seattle city in deciding the route to be taken based on the weather and traffic conditions by reviewing the accident severity prediction model during adverse weather conditions.

2. DATA

2.1 Data Source

To proceed with this project, the data has been sourced from below repository https://s3.us.cloud-object-storage.appdomain.cloud/cf-courses-data/CognitiveClass/DP0701EN/version-2/Data-Collisions.csv

Locating the data source

```
# Get data from source path to variable
filename = "https://s3.us.cloud-object-storage.appdomain.cloud/cf-courses-data/CognitiveClass/DP0701EN/version-2/Data-Collision
```

Reading Data from source and loading to data frame

```
# Read data to data frame
df = pd.read_csv(filename)
```

The dataset has information gathered on the road traffic accidents of Seattle City. From the data extracted, the key attributes are:

a. Severity Code – The values are 1 & 2 which denotes property damage & human injury respectively. Code 2 is valued as more severe
b. Weather – Sample values: "Raining", "Clear", "Overcast", ...
c. Road Condition – Sample values: "Wet", "Dry", "Ice", ...
d. Light Condition – Sample values: "Daylight", "Dawn", ...

2.2 Data Understanding

There are many columns that will not be used for this model. The data is loaded into a panda data frame and remove the columns that is not required. The initial dataset consists of 38 columns (features/attributes) and 194673 rows. The dataset will be cleaned according to the requirements of this project. The data will be analysed to identify the set of criteria on which high severity accidents happen based on the attribute values.

The structure of data and its data type.

```
df.head()
```

YCODE	X	Y	OBJECTID	INCKEY	COLDETKEY	REPORTNO	STATUS	ADDRTYPE	INTKEY	...	ROADCOND	LIGHTCOND	PEDROWNOTGRNT
2	-122.323148	47.703140	1	1307	1307	3502005	Matched	Intersection	37475.0	...	Wet	Daylight	NaN
1	-122.347294	47.647172	2	52200	52200	2607959	Matched	Block	NaN	...	Wet	Dark - Street Lights On	NaN
1	-122.334540	47.607871	3	26700	26700	1482393	Matched	Block	NaN	...	Dry	Daylight	NaN
1	-122.334803	47.604803	4	1144	1144	3503937	Matched	Block	NaN	...	Dry	Daylight	NaN
2	-122.306426	47.545739	5	17700	17700	1807429	Matched	Intersection	34387.0	...	Wet	Daylight	NaN

Check the data types

```
# Check data type
print(df.dtypes)
```

```
SEVERITYCODE        int64
X                 float64
Y                 float64
OBJECTID            int64
INCKEY              int64
COLDETKEY           int64
REPORTNO           object
STATUS             object
ADDRTYPE           object
INTKEY            float64
LOCATION           object
EXCEPTRSNCODE      object
EXCEPTRSNDESC      object
SEVERITYCODE.1      int64
SEVERITYDESC       object
```

The statistical analysis of the data using the describe function

View statistical data of numerical values from Data Frame

```
df.describe()
```

	SEVERITYCODE	X	Y	OBJECTID	INCKEY	COLDETKEY	INTKEY	SEVERITYCODE.1
count	194673.000000	189339.000000	189339.000000	194673.000000	194673.000000	194673.000000	65070.000000	194673.000000
mean	1.298901	-122.330518	47.619543	108479.364930	141091.456350	141298.811381	37558.450576	1.298901
std	0.457778	0.029976	0.056157	62649.722558	86634.402737	86986.542110	51745.990273	0.457778
min	1.000000	-122.419091	47.495573	1.000000	1001.000000	1001.000000	23807.000000	1.000000
25%	1.000000	-122.348673	47.575956	54267.000000	70383.000000	70383.000000	28667.000000	1.000000
50%	1.000000	-122.330224	47.615369	106912.000000	123363.000000	123363.000000	29973.000000	1.000000
75%	2.000000	-122.311937	47.663664	162272.000000	203319.000000	203459.000000	33973.000000	2.000000
max	2.000000	-122.238949	47.734142	219547.000000	331454.000000	332954.000000	757580.000000	2.000000

2.3 Approach

a. Different data visualisation methods are used to study and analyse the data.

b. The predictor or target variable will be 'SEVERITYCODE' because it is used to measure the severity of an accident from 1 to 2 within the dataset. Other attributes used to weigh the severity of an accident are 'WEATHER', 'ROADCOND', 'LIGHTCOND' and 'SPEEDING'.
   - Severity code 1 stands for property damage and
   - Severity code 2 stands for injury during an accident.

2.4 Data Wrangling

The Data should be cleaned up with only required columns and remove those which are having null values. Those which are empty should be marked as NaN and then delete the rows with the missing values or Nan in the required columns.

View new dataframe

```
df1.head()
```

	SEVERITYCODE	LAT	LON	ADDRTYPE	COLLISIONTYPE	WEATHER	ROADCOND	LIGHTCOND	SPEEDING
0	2	-122.323148	47.703140	Intersection	Angles	Overcast	Wet	Daylight	NaN
1	1	-122.347294	47.647172	Block	Sideswipe	Raining	Wet	Dark - Street Lights On	NaN
2	1	-122.334540	47.607871	Block	Parked Car	Overcast	Dry	Daylight	NaN
3	1	-122.334803	47.604803	Block	Other	Clear	Dry	Daylight	NaN
4	2	-122.306426	47.545739	Intersection	Angles	Raining	Wet	Daylight	NaN

In order to find the data with null values, the data is pulled into a new data frame which are missing values. The columns which are null is shown as True.

Evaluate Missing Data

```
# Evaluating for Missing Data
missing_data = df1.isnull()
missing_data.head(5)
```

	SEVERITYCODE	LAT	LON	ADDRTYPE	COLLISIONTYPE	WEATHER	ROADCOND	LIGHTCOND	SPEEDING
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False

Count the missing values in each column (True=missing value) which need to be cleaned up for our analysis.

```
SEVERITYCODE
False    194673
Name: SEVERITYCODE, dtype: int64

LAT
False    189339
True       5334
Name: LAT, dtype: int64

LON
False    189339
True       5334
Name: LON, dtype: int64

ADDRTYPE
False    192747
True       1926
Name: ADDRTYPE, dtype: int64

COLLISIONTYPE
False    189769
True       4904
Name: COLLISIONTYPE, dtype: int64
```

As there are lot of columns with missing values, need to update them as Nan. Also, those columns which are having value as unknown. Since we cannot find the predictor variable without these attribute values, deleted these rows with missing values.

Dropped the rows of data with NaN values.

Dropping rows of data that have NaN values

```python
# Dropping rows where value is NAN
df1.dropna(subset=["WEATHER"], axis=0, inplace=True)
df1.dropna(subset=["ROADCOND"], axis=0, inplace=True)
df1.dropna(subset=["LIGHTCOND"], axis=0, inplace=True)
df1.dropna(subset=["LAT"], axis=0, inplace=True)
```

Machine Learning models require numerical data and cannot handle alphanumeric strings. For example, each entry in the “WEATHER” column contains a text string which takes one of eleven values (e.g. “Clear”, “Rain”, “Snow”, etc) which describes the prevailing weather conditions at the time of the accident. Columns such as this should be converted to numeric values for performing a proper analysis.

Label encoding has been used to covert the features like Weather condition, Road condition, Light condition and Speeding to numeric values.

View the dataframe after columns updated with numeric values

```python
df1.head()
```

	SEVERITYCODE	LAT	LON	ADDRTYPE	COLLISIONTYPE	WEATHER	ROADCOND	LIGHTCOND	SPEEDING
0	2	-122.323148	47.703140	Intersection	Angles	5	8	6	0
1	1	-122.347294	47.647172	Block	Sideswipe	7	8	3	0
2	1	-122.334540	47.607871	Block	Parked Car	5	1	6	0
3	1	-122.334803	47.604803	Block	Other	2	1	6	0
4	2	-122.306426	47.545739	Intersection	Angles	7	8	6	0

Next, should convert the datatype of these columns to integer and verify.

Verify the data type

```
# Check df1 data type
print(df1.dtypes)
```

```
SEVERITYCODE      int64
LAT             float64
LON             float64
ADDRTYPE         object
COLLISIONTYPE    object
WEATHER           int64
ROADCOND          int64
LIGHTCOND         int64
SPEEDING          int64
dtype: object
```

2.5 Balancing the Data

The data is imbalanced for the target variable SEVERITY CODE. Severity code 1 is nearly three times the size of severity code 2.

```
# Check the count of rows in both severity classes
df1['SEVERITYCODE'].value_counts()
```

```
1    128154
2     56013
Name: SEVERITYCODE, dtype: int64
```

There are 128154 rows of data with SEVERITYCODE 1 and 56013 rows of data with SEVERITYCODE 2. However, this imbalance between the real-life occurrences of different accident outcomes may bias the model if not accounted for.

This can be fixed by down sampling the majority class of severity 1 data to match with severity 2 data count.

Resample Severity 1 data frame with sample value as 56013 to match Severity 2 count

```
# Downsample majority class
dfminsev = resample(dfsev_1, replace=False,    # sample without replacement
                             n_samples=56013,     # to match minority class
                             random_state=123)
```

```
# Check Severity counts
df_sampled['SEVERITYCODE'].value_counts()
```

```
2    56013
1    56013
Name: SEVERITYCODE, dtype: int64
```

After resampling, the value counts on the new data frame (df_sampled) shows equal number of severity 1 and 2 rows which is perfectly balanced.

3. Methodology

3.1 Data Science Tools used

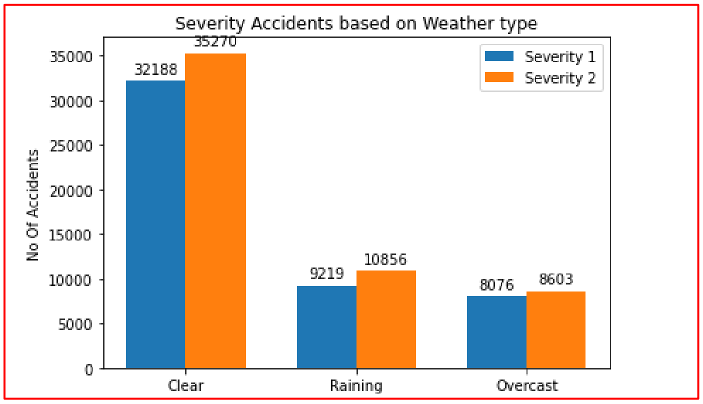
- The tools used are pandas, numpy, matplotlib libraries.
- Seaborn graphics library was used for general statistics to display graphs. Matplotlib Library was used to generate bar chart.
- Used K-Means algorithm from Scikit-Learn Library for clustering attributes on generated data.
- Folium Library was used to create maps with popups labels to allow quick identification of accident locations with popup info on Severity, weather condition, road condition etc.

3.2 Data Visualisation

Various charts and maps are plotted to clearly visualise and analyse the data using Seaborn library, Matplotlib library and Folium library.

a. Severity of accidents based on Weather Condition

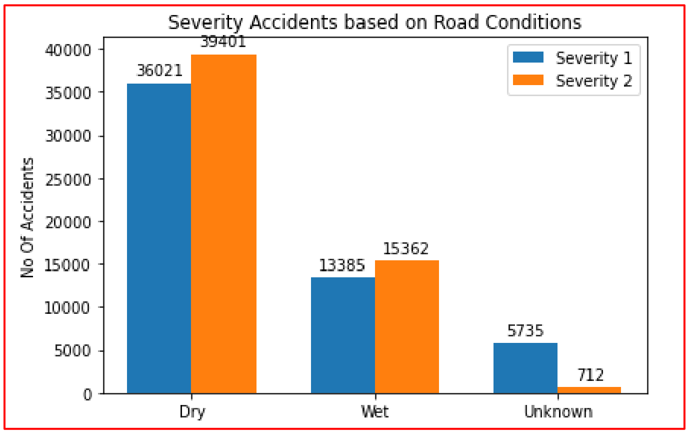
A bar chart is plotted to find the severity of accidents based on different weather condition.



The above chart shows that, from various weather conditions reported, most of the accidents happen during clear weather and severity 2 accidents happen more than severity 1 which means that its more of accidents with injuries than property damage.

b. Severity of accidents based on Road Condition

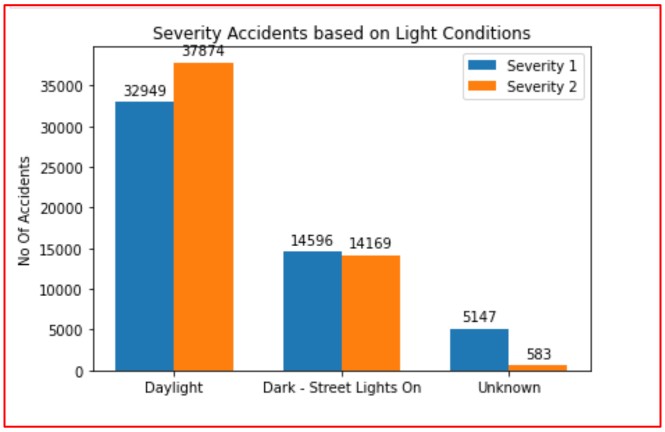
A bar chart is plotted to find the severity of accidents based on different road conditions.



The above chart shows that, from various road conditions reported, most of the accidents happen during dry road conditions and severity 2 accidents happen more that severity 1 which means that its more of accidents with injuries than property damage.

c. Severity of accidents based on Light condition

Plotting a bar chart to show the Severity accidents based on Light conditions



The above chart shows that, from various light conditions reported, most of the accidents happen during day light road conditions and severity 2 accidents happen more that severity 1 which means that its more of accidents with injuries than property damage.

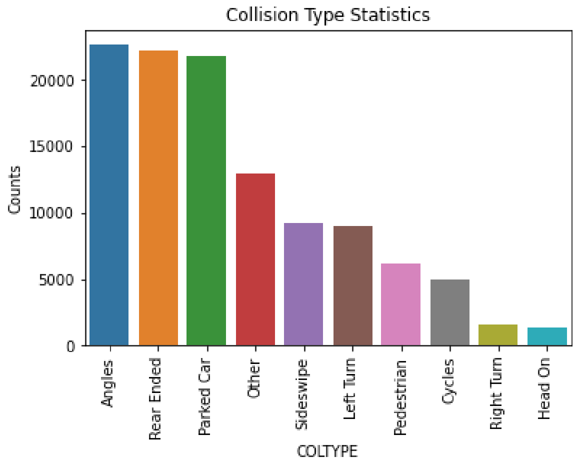
d. Number of accidents based on Collision types

Data analysis on accidents with the type of collisions plotted on a bar chart to show the collision type in accidents

Plot the data extracted to a bar chart

```
# View the collossion type rating in bar chart
#df_sampled['COLLISIONTYPE'].value_counts().plot(kind='bar', title = 'Collision Type Statistics')
barchart=sns.barplot(x='COLTYPE', y='Counts', data=df_wbar)
barchart.set_xticklabels(barchart.get_xticklabels(),rotation=90) # add to x-label to the plot
barchart.set_title('Collision Type Statistics') # add title to the plot
```

```
Text(0.5, 1.0, 'Collision Type Statistics')
```



Above graph shows that the highest number of accidents happen at road Angles followed by Rear Ended and Parked Car

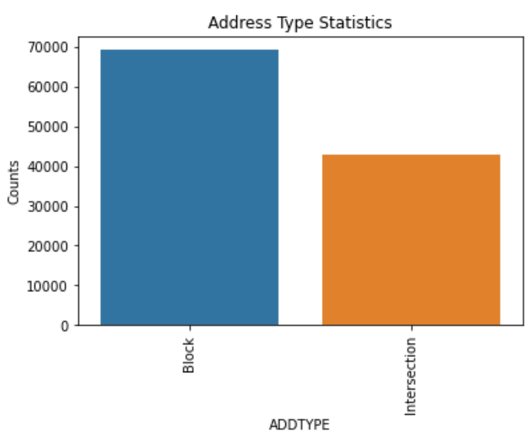
e. Number of accidents based on locations

A bar chart is plotted for the number of accidents based on address type.

	ADDTYPE	Counts
0	Block	69100
1	Intersection	42926

```
# View the address type rating in bar chart
#df_sampled['ADDRTYPE'].value_counts().plot(kind='bar',  title = 'Address Type Statistics')
barchart=sns.barplot(x='ADDTYPE', y='Counts', data=df_abar)
barchart.set_xticklabels(barchart.get_xticklabels(),rotation=90) # add to x-label to the plot
barchart.set_title('Address Type Statistics') # add title to the plot
```

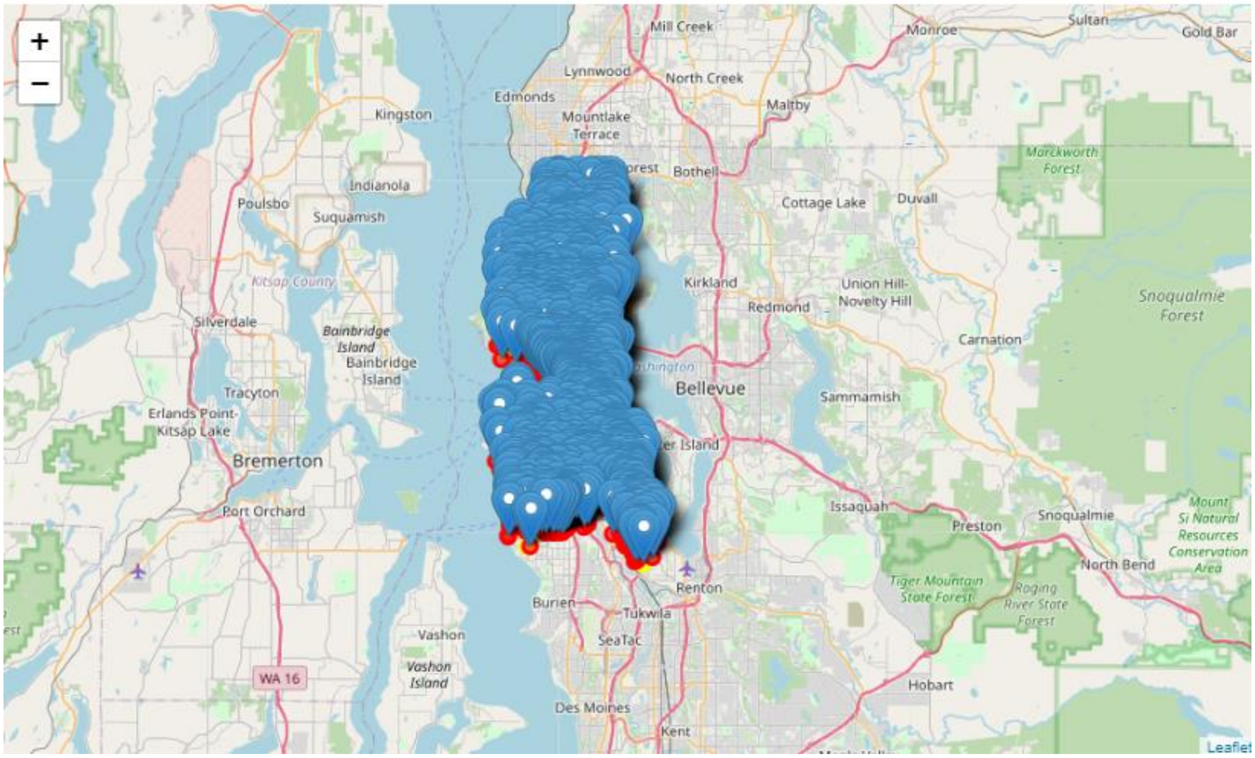
```
Text(0.5, 1.0, 'Address Type Statistics')
```



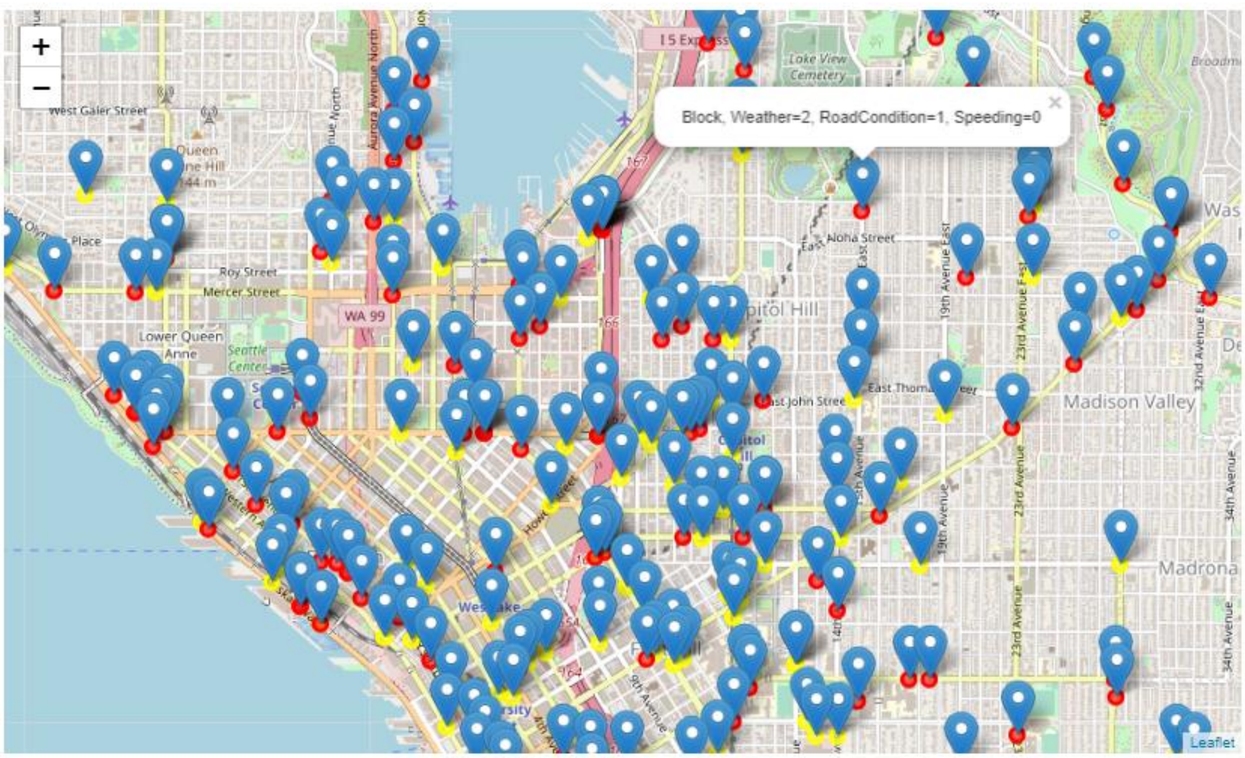
Above graph shows that the accidents happen more at Block address type.

f. Overview of Seattle city

Overview of the accident locations in Seattle City.



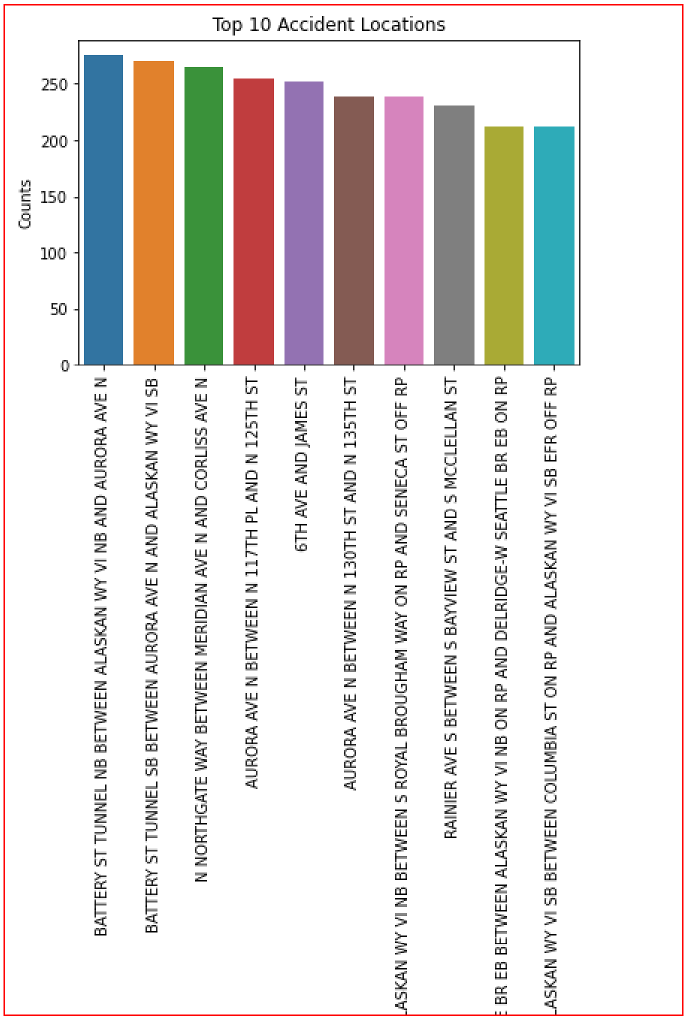
Accident locations are super imposed on the map with severity code 1 marked in yellow circle and severity code 2 marked in red circle. Click on the popup to view the Address type, Weather, Road condition and speeding status at the location. Zoom in to the map to see the roads to see where more accidents happen.



PUBLIC

Top 10 locations where highest number of accidents reported.

	LOCATION	Counts
0	BATTERY ST TUNNEL NB BETWEEN ALASKAN WY VI NB ...	276
1	BATTERY ST TUNNEL SB BETWEEN AURORA AVE N AND ...	271
2	N NORTHGATE WAY BETWEEN MERIDIAN AVE N AND COR...	265
3	AURORA AVE N BETWEEN N 117TH PL AND N 125TH ST	254
4	6TH AVE AND JAMES ST	252
5	AURORA AVE N BETWEEN N 130TH ST AND N 135TH ST	239
6	ALASKAN WY VI NB BETWEEN S ROYAL BROUGHAM WAY ...	238
7	RAINIER AVE S BETWEEN S BAYVIEW ST AND S MCCLE...	231
8	WEST SEATTLE BR EB BETWEEN ALASKAN WY VI NB ON...	212
9	ALASKAN WY VI SB BETWEEN COLUMBIA ST ON RP AND...	212



3.3 Model Development

In order to develop a model for predicting accident severity, the re-sampled, cleaned dataset was split in to testing and training sub-samples (containing 30% and 70% of the samples, respectively) using the Scikit learn "train_test_split" method.

3.3.1 Initialization

a. Define X and Y

```
X=np.asarray(df_sampled[['WEATHER','ROADCOND','LIGHTCOND','SPEEDING']])
X[0:5]
```

```
array([[5, 1, 6, 0],
       [2, 1, 6, 0],
       [5, 8, 6, 0],
       [7, 8, 3, 0],
       [2, 1, 6, 0]])
```

```
y=np.asarray(df_sampled['SEVERITYCODE'])
y[0:5]
```

```
array([1, 1, 1, 1, 1])
```

b. Normalise the dataset

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

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/sklear
rning: Data with input dtype int64 was converted to float64 by Standa
  warnings.warn(msg, DataConversionWarning)
```

```
array([[ 0.79696949, -0.57203105,  0.62221169, -0.23039919],
       [-0.58634316, -0.57203105,  0.62221169, -0.23039919],
       [ 0.79696949,  1.68308522,  0.62221169, -0.23039919],
       [ 1.71917793,  1.68308522, -1.04627678, -0.23039919],
       [-0.58634316, -0.57203105,  0.62221169, -0.23039919]])
```

c. Train/Test Split

We used 30% of the data for testing and 70% for training.

```
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)
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

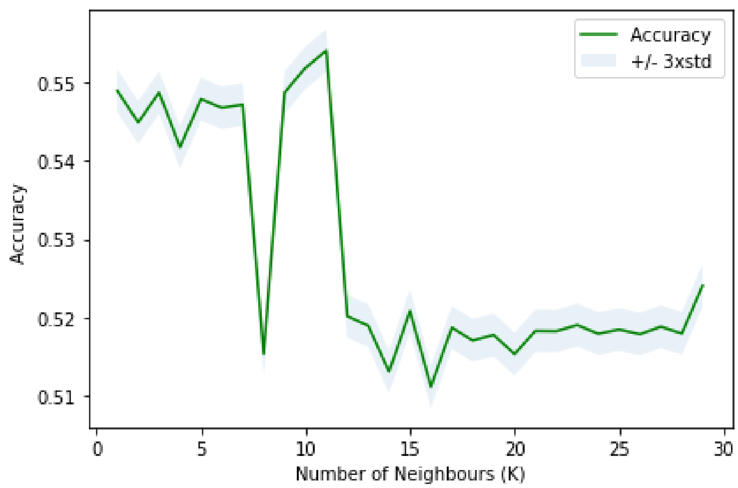
```
Train set: (78418, 4) (78418,)
Test set: (33608, 4) (33608,)
```

3.3.2 Modelling and Predictions

a. K-Nearest Neighbor (KNN)

KNN helps us to predict the severity code of an outcome by finding the most similar to data point within k distance.

Calculated the accuracy of KNN for different Ks




```
print("Best accuracy =", mean_acc.max(),"k=", mean_acc.argmax()+1 )
```

```
Best accuracy = 0.5540645084503689 k= 11
```

Built the KNN model using the best value of k and predicted the target variable.

```
k = 11
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
kyhat = neigh.predict(X_test)
kyhat[0:5]
```

```
array([2, 2, 2, 2, 1])
```

b. Decision Tree

A decision tree model gives us a layout of all possible outcomes so we can fully analyze the consequences of a decision. In this context, the decision tree observes all possible outcomes of different weather conditions.

```
from sklearn.tree import DecisionTreeClassifier

#Building the decision tree
dtree=DecisionTreeClassifier(criterion="entropy", max_depth=7)
dtree.fit(X_train,y_train)
```

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=7,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

```
#Train Model & Predict
dtyhat=dtree.predict(X_test)
dtyhat[0:5]
```

```
array([2, 2, 2, 2, 1])
```

c. Logistic Regression

As the dataset only provides us with two severity code outcomes, the model will only predict one of those two classes. This makes the data binary, which is perfect to use with logistic regression.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_train)
LR
```

```
LogisticRegression(C=0.01, 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)
```

```
LRyhat = LR.predict(X_test)
LRyhat

LRyhat_prob = LR.predict_proba(X_test)
LRyhat_prob
```

```
array([[0.47702251, 0.52297749],
       [0.47702251, 0.52297749],
       [0.47702251, 0.52297749],
       ...,
       [0.53045002, 0.46954998],
       [0.47702251, 0.52297749],
       [0.40193133, 0.59806867]])
```

4. Results and Evaluation

Checking the accuracy of our models.

```
print("KNN Jaccard index: %.2f" % jaccard_similarity_score(y_test, kyhat))
print("KNN F1-score: %.2f" % f1_score(y_test, kyhat, average='weighted') )
```

```
KNN Jaccard index: 0.55
KNN F1-score: 0.54
```

```
print("Decision Tree Jaccard index: %.2f" % jaccard_similarity_score(y_test, dtyhat))
print("Decision Tree F1-score: %.2f" % f1_score(y_test, dtyhat, average='weighted') )
```

```
Decision Tree Jaccard index: 0.56
Decision Tree F1-score: 0.53
```

```
print("LR Jaccard index: %.2f" % jaccard_similarity_score(y_test, LRyhat))
print("LR F1-score: %.2f" % f1_score(y_test, LRyhat, average='weighted') )
print("LR LogLoss: %.2f" % log_loss(y_test, LRyhat_prob))
```

```
LR Jaccard index: 0.55
LR F1-score: 0.53
LR LogLoss: 0.68
```

Evaluation metrics used to test the accuracy of models were jaccard index, f-1 score and logloss for logistic regression. Choosing different k, max depth and hyperparameter C values helped to improve our accuracy to be the best possible.

Algorithm	Jaccard	F1-score	LogLoss
KNN	0.55	0.54	NA
Decision Tree	0.56	0.53	NA
LogisticRegression	0.55	0.53	0.68

5. Discussion

The initial data had categorical data that was of type 'object'. This is not a data type that we could have fed through an algorithm, so label encoding was used to created new classes that were of type int; a numerical data type. After solving that issue we were presented with another - imbalanced data. As mentioned earlier, severity 1 was nearly three times larger than severity 2. The solution to this was down sampling the majority severity with sklearn's resample tool. The data was down sampled to match the minority severity class.

After data wrangling and analysis of the data, it was then fed through three ML models; K-Nearest Neighbour, Decision Tree and Logistic Regression. Although the first two are ideal for this project, logistic regression made most sense because of its binary nature.

Evaluation metrics used to test the accuracy of our models were jaccard index, f-1 score and logloss for logistic regression. Choosing different k, max depth and hyperparameter C values helped to improve our accuracy to be the best possible.

6. Conclusion

Based on historical data from weather conditions pointing to certain classes, we can conclude that weather conditions have impact on accident severity. The top location of accidents shown will help drivers to be careful. It also points out that during clear weather a greater number of accidents happen with severity 2 and the cause related is speeding and road conditions.

Be Safe – Be Careful & Be vigilant. Choose the safest route while driving based on this analysis published. All the best.