Capstone Project - Car accident severity

Introduction

A description of the problem and a discussion of the background.

There are a number of different factors that cause road collisions, in most cases it is related to driver factors, road and weather conditions. Collisions can result in terrible consequences, including death, injury, disability, property damage and financial costs.

For this project I will attempt to build a model which can predict the severity of an accident given the weather and the road conditions. Such a model could bring a new awareness to drivers of the dangers that can occur while travelling on the road and would encourage them to drive more safely.

The question we will attempt to answer is if you knew the weather and road conditions how severe would an accident be if a collision occurs?

Data

A description of the data and how it will be used to solve the problem.

The data we will use is provided by the SDOT Traffic Management Division and contains data of all types of collisions that happened in Seattle from 2004 to May/2020.

The data contains 194,673 samples and has 37 features that covers the weather and road conditions, collision factors and fatalities.

I will examine this data in detail in order to attempt to answer the question. I will do this by preparing the data to make the dataset readable and then apply 3 classification models on it. I will then discuss the results and apply conclusions for the report.

Import and view the data:

Import the data

```
In [22]: import pandas as pd
         import matplotlib.pyplot as plt
         %matplotlib inline
         import seaborn as sns

         path = "https://s3.us.cloud-object-storage.appdomain.cloud/cf-courses-data/CognitiveClass/DP0701EN/version-2/Data-Collisions.csv"

         df = pd.read_csv(path)
```

```
In [23]: print('Samples:', df.shape[0])
         print('Features:', df.shape[1])

         Samples: 194673
         Features: 38
```

In [24]:
```
df.describe(include="all")
```

Out[24]:

	SEVERITYCODE	X	Y	OBJECTID	INCKEY	COLDETKEY	REPORTNO	STATUS	ADDRTYPE	INTKEY	...	ROADCOND	LIGHTCOND	PEDROWNOTGRNT	SDOTCOLNUM	SPEEDING	ST_CO
count	194673.000000	189339.000000	189339.000000	194673.000000	194673.000000	194673.000000	194673	194673	192747	65070.000000	...	189661	189503	4667	1.149360e+05	9333	
unique	NaN	NaN	NaN	NaN	NaN	NaN	194670	2	3	NaN	...	9	9	1	NaN	1	
top	NaN	NaN	NaN	NaN	NaN	NaN	1780512	Matched	Block	NaN	...	Dry	Daylight	Y	NaN	Y	
freq	NaN	NaN	NaN	NaN	NaN	NaN	2	189786	126926	NaN	...	124510	116137	4667	NaN	9333	
mean	1.298901	-122.330518	47.619543	108479.364930	141091.456350	141298.811381	NaN	NaN	NaN	37558.450576	...	NaN	NaN	NaN	7.972521e+06	NaN	
std	0.457778	0.029976	0.056157	62649.722558	86634.402737	86986.542110	NaN	NaN	NaN	51745.990273	...	NaN	NaN	NaN	2.553533e+06	NaN	
min	1.000000	-122.419091	47.495573	1.000000	1001.000000	1001.000000	NaN	NaN	NaN	23807.000000	...	NaN	NaN	NaN	1.007024e+06	NaN	
25%	1.000000	-122.348673	47.575956	54267.000000	70383.000000	70383.000000	NaN	NaN	NaN	28667.000000	...	NaN	NaN	NaN	6.040015e+06	NaN	
50%	1.000000	-122.330224	47.615369	106912.000000	123363.000000	123363.000000	NaN	NaN	NaN	29973.000000	...	NaN	NaN	NaN	8.023022e+06	NaN	
75%	2.000000	-122.311937	47.663664	162272.000000	203319.000000	203459.000000	NaN	NaN	NaN	33973.000000	...	NaN	NaN	NaN	1.015501e+07	NaN	
max	2.000000	-122.238949	47.734142	219547.000000	331454.000000	332954.000000	NaN	NaN	NaN	757580.000000	...	NaN	NaN	NaN	1.307202e+07	NaN	

11 rows × 38 columns

Find out how many missing values are displayed in each column

In [25]:
```
df.isna().sum().to_frame().rename(columns={0: 'NaN Count'})
```

Out[25]:

	NaN Count
SEVERITYCODE	0
X	5334
Y	5334
OBJECTID	0
INCKEY	0
COLDETKEY	0
REPORTNO	0
STATUS	0
ADDRTYPE	1926
INTKEY	129603
LOCATION	2677
EXCEPTRSNCODE	109862
EXCEPTRSNDESC	189035
SEVERITYCODE.1	0
SEVERITYDESC	0
COLLISIONTYPE	4904
PERSONCOUNT	0
PEDCOUNT	0
PEDCYLCOUNT	0
VEHCOUNT	0
INCDATE	0
INCDTTM	0
JUNCTIONTYPE	6329
SDOT_COLCODE	0
SDOT_COLDESC	0
INATTENTIONIND	164868
UNDERINFL	4884
WEATHER	5081
ROADCOND	5012
LIGHTCOND	5170
PEDROWNOTGRNT	190006
SDOTCOLNUM	79737
SPEEDING	185340
ST_COLCODE	18
ST_COLDESC	4904
SEGLANEKEY	0
CROSSWALKKEY	0
HITPARKEDCAR	0

As we can see there is a lot of missing values in the data frame. Due to this we will not consider features with missing data. The columns in the data set which we are most interested in are:

- COLLISIONTYPE: Collision type
- WEATHER: Weather conditions during the time of the collision.
- ROADCOND: The condition of the road during the collision.
- LIGHTCOND: The light conditions during the collision.
- UNDERINFL: Whether or not a driver involved was under the influence of drugs or alcohol.

These columns do contain some missing values but it is below 3% of the total amount of samples. The target variable is SEVERITYCODE, this identifies the severity of the collision

- 1: Property Damage
- 2: Injury collision

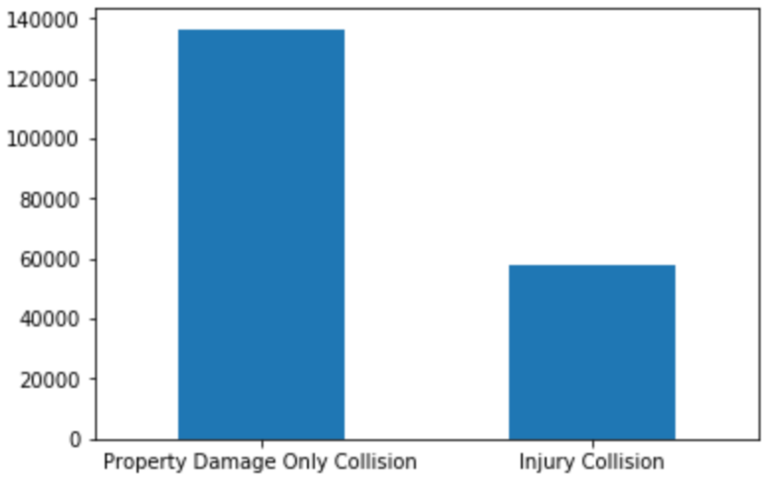
In [32]:
```
df['SEVERITYDESC'].value_counts().to_frame()
```

Out[32]:

	SEVERITYDESC
Property Damage Only Collision	136485
Injury Collision	58188

In [33]:
```
df['SEVERITYDESC'].value_counts().plot(kind='bar')
plt.xticks(rotation=0)
```

Out[33]: `(array([0, 1]), <a list of 2 Text xticklabel objects>)`



Number of Annual Collisions

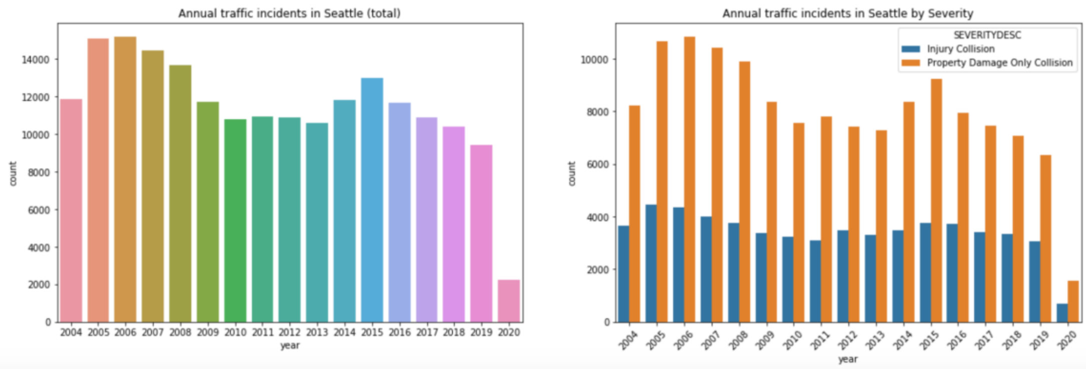
Note that the number for 2020 is lower as the data covers up to May 2020

Number of collisions year by year

In [34]:
```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(20, 6))

df['year'] = pd.DatetimeIndex(df['INCDATE']).year
df['year'].value_counts().sort_index()#.plot(kind='bar')
sns.countplot(x="year", data=df, ax=ax1)
sns.countplot(x="year", hue="SEVERITYDESC", data=df, ax=ax2)
plt.xticks(rotation=45)
ax1.set_title('Annual traffic incidents in Seattle (total)')
ax2.set_title('Annual traffic incidents in Seattle by Severity')
```

Out[34]: `Text(0.5, 1.0, 'Annual traffic incidents in Seattle by Severity')`



Collision Types

Now we will look at the different types of collisions that can occur and view how many result in injuries and property damage

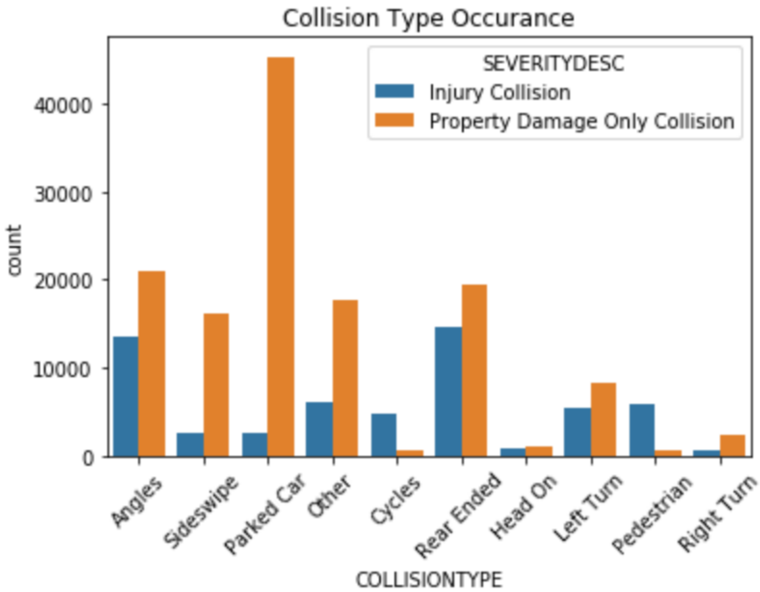
In [35]:
```
df['COLLISIONTYPE'].value_counts().sort_values(ascending=False).to_frame()
```

Out[35]:

	COLLISIONTYPE
Parked Car	47987
Angles	34674
Rear Ended	34090
Other	23703
Sideswipe	18609
Left Turn	13703
Pedestrian	6608
Cycles	5415
Right Turn	2956
Head On	2024

```
In [36]: sns.countplot(x="COLLISIONTYPE", hue="SEVERITYDESC", data=df)
         plt.xticks(rotation=45)
         plt.title('Collision Type Occurance')
```

Out[36]: Text(0.5, 1.0, 'Collision Type Occurance')



Weather Impact

Next we will view the weather conditions when the collisions occured and if they resulted in injury or property damage

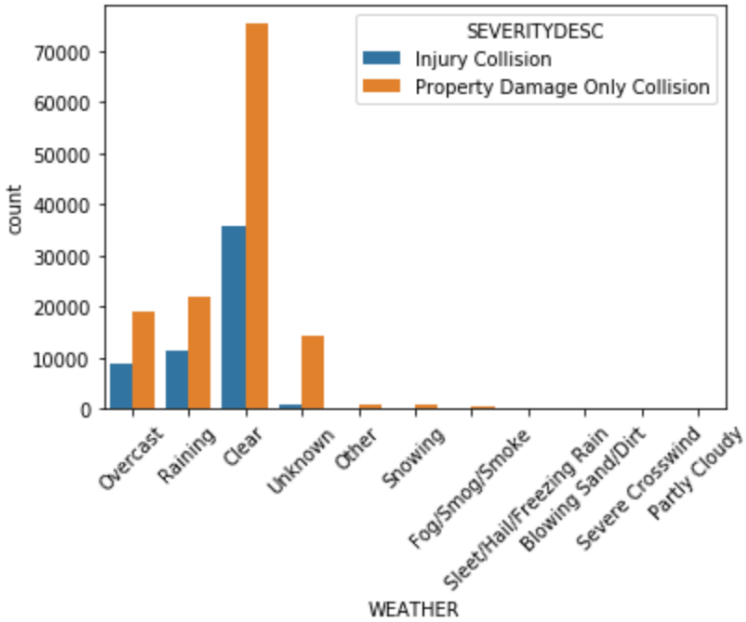
```
In [37]: df['WEATHER'].value_counts().sort_values(ascending=False).to_frame()
```

Out[37]:

	WEATHER
Clear	111135
Raining	33145
Overcast	27714
Unknown	15091
Snowing	907
Other	832
Fog/Smog/Smoke	569
Sleet/Hail/Freezing Rain	113
Blowing Sand/Dirt	56
Severe Crosswind	25
Partly Cloudy	5

```
In [38]: sns.countplot(x="WEATHER", hue="SEVERITYDESC", data=df)
         plt.xticks(rotation=45)
```

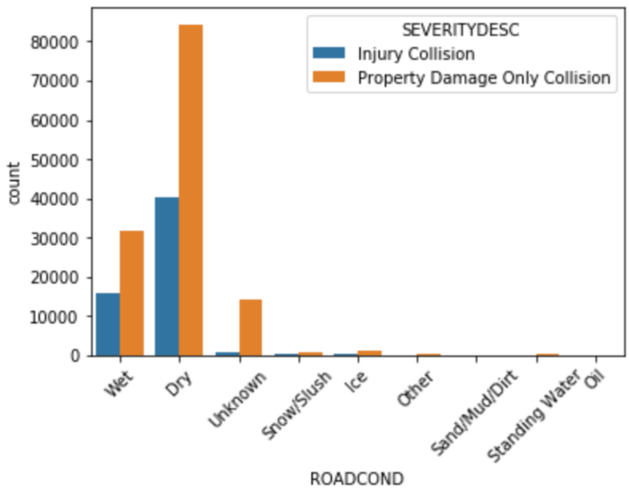
```
Out[38]: (array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
          <a list of 11 Text xticklabel objects>)
```



Condition of the Road

```
In [39]: sns.countplot(x="ROADCOND", hue="SEVERITYDESC", data=df)
         plt.xticks(rotation=45)
```

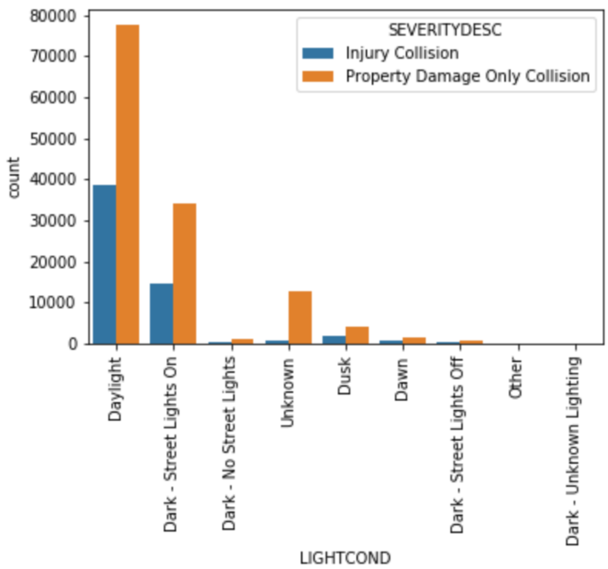
```
Out[39]: (array([0, 1, 2, 3, 4, 5, 6, 7, 8]), <a list of 9 Text xticklabel objects>)
```



Visibility

n [40]:
```
sns.countplot(x="LIGHTCOND", hue="SEVERITYDESC", data=df)
plt.xticks(rotation=90)
```

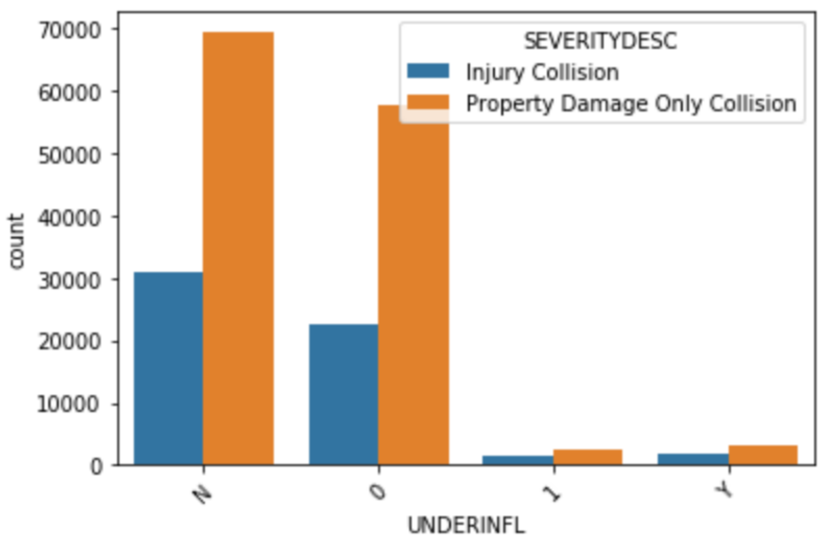
Out[40]: (array([0, 1, 2, 3, 4, 5, 6, 7, 8]), <a list of 9 Text xticklabel objects>)



Driving under the influence of Alcohol or Drugs

l]:
```
sns.countplot(x="UNDERINFL", hue="SEVERITYDESC", data=df)
plt.xticks(rotation=45)
```

[41]: (array([0, 1, 2, 3]), <a list of 4 Text xticklabel objects>)



Methodology

Data preparation and cleaning

We will clean the data to make the dataset more readable and suitable for the machine learning algorithms.

Removing Data

Of the 37 attributes, we will not consider the features with over 40% missing data, or other unclear and irrelevant data. We will use the COLLISIONTYPE, WEATHER, ROADCOND, LIGHTCOND and UNDERINFL data as attributes to classify the SEVERITYCODE. In order to do that we need to ensure the data is suitable for a binary classification model. We will use some popular machine learning algorithms to build up models and analyse their performance and predict the collision severity.

Data preparation and cleaning

In [17]:
```
data = df[['COLLISIONTYPE', 'WEATHER', 'ROADCOND', 'LIGHTCOND', 'UNDERINFL', 'SEVERITYCODE']]
data = data.dropna()
data.shape
```
Out[17]: (189316, 6)

In [18]:
```
data.head(10)
```
Out[18]:

	COLLISIONTYPE	WEATHER	ROADCOND	LIGHTCOND	UNDERINFL	SEVERITYCODE
0	Angles	Overcast	Wet	Daylight	N	2
1	Sideswipe	Raining	Wet	Dark - Street Lights On	0	1
2	Parked Car	Overcast	Dry	Daylight	0	1
3	Other	Clear	Dry	Daylight	N	1
4	Angles	Raining	Wet	Daylight	0	2
5	Angles	Clear	Dry	Daylight	N	1
6	Angles	Raining	Wet	Daylight	0	1
7	Cycles	Clear	Dry	Daylight	N	2
8	Parked Car	Clear	Dry	Daylight	0	1
9	Angles	Clear	Dry	Daylight	0	2

Working with missing values

The chosen attributes still has about 3% of data missing so we'll just drop them as there is still enough data to use.

Treating the categorical variables

As all attributes are categorical we will apply a label encoding technique on them.

Convert Categorical features to numerical values

```
In [19]: data['UNDERINFL'].replace(to_replace=['N','Y','0'], value=[0,1,0],inplace=True)
         data['UNDERINFL'].value_counts()
```

```
Out[19]: 0    180219
         1      9097
         Name: UNDERINFL, dtype: int64
```

Train/Test split and data normalization

In order to test/train the data the independent variables will be split into dataset X and the dependent variables ‘SEVERITYCODE’ to dataset Y.

```
In [22]: X = features
         y = data['SEVERITYCODE'].values
```

We will now randomly pick samples and split in the radio of 70% to train the model and 30% to test the model.
After this split the data will be normalized to ensure the features are on a similar scale.

```
In [23]: 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=0)
         X_train.head()
```

Out[23]:

	COLLISIONTYPE	WEATHER	ROADCOND	LIGHTCOND	UNDERINFL
109717	0	1	0	5	0
9615	7	1	0	5	0
133991	3	1	0	5	0
76012	5	1	0	5	0
97913	9	10	7	8	0

```
In [24]: from sklearn import preprocessing

         X= preprocessing.StandardScaler().fit(X).transform(X)
         X_train = preprocessing.StandardScaler().fit(X_train).transform(X_train.astype(float))
         X_test = preprocessing.StandardScaler().fit(X_test).transform(X_test.astype(float))
         X_train[0:5]
         X_test[0:5]
```

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/preprocessing/data.py:645: DataConversionWarning: Data with input dtype int64 we
ndardScaler.
  return self.partial_fit(X, y)
/opt/conda/envs/Python36/lib/python3.6/site-packages/ipykernel/__main__.py:3: DataConversionWarning: Data with input dtype int64 were all co
r.
  app.launch_new_instance()
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/preprocessing/data.py:645: DataConversionWarning: Data with input dtype int64 we
ndardScaler.
  return self.partial_fit(X, y)
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/preprocessing/data.py:645: DataConversionWarning: Data with input dtype int64 we
ndardScaler.
  return self.partial_fit(X, y)
```

```
Out[24]: array([[ 0.90028023, -0.72579269, -0.71028447, -1.39919831, -0.22518559],
                [-1.61258466, -0.72579269, -0.71028447,  0.35042853, -0.22518559],
                [ 0.18231884, -0.72579269, -0.71028447, -1.39919831, -0.22518559],
                [ 0.90028023,  0.32670431, -0.71028447,  0.93363748, -0.22518559],
                [-1.61258466, -0.72579269, -0.71028447,  0.35042853, -0.22518559]])
```

Classification: Modeling

We will use the dataset on 3 classification models:

- KNN: Classifies unseen data through the majority of its 'neighbours'. In this case we already know K=2 (2 classes of SEVERITY CODES). After obtaining each model's predictions we will evaluate their accuracy, precison, f1-score, log-loss and compare and discuss the results.

KNN Model

```
In [25]: from sklearn.neighbors import KNeighborsClassifier
         model_knn = KNeighborsClassifier(n_neighbors = 2).fit(X_train, y_train)
         model_knn

Out[25]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                              metric_params=None, n_jobs=None, n_neighbors=2, p=2,
                              weights='uniform')
```

- Logistic Regression: Classifies data by estimating the probability of classes.

Logistic Regression

```
In [26]: from sklearn.linear_model import LogisticRegression

         model_lr = LogisticRegression(C=0.0001, solver='liblinear')
         model_lr.fit(X_train, y_train)
         model_lr

Out[26]: LogisticRegression(C=0.0001, 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)
```

- Decision Tree: Classifies by breaking down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed.

Decision Tree

```
In [27]: from sklearn.tree import DecisionTreeClassifier

         model_tree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
         model_tree.fit(X_train, y_train)
         model_tree

Out[27]: DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=4,
                                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')
```

Evaluation using the data set

```
In [30]: from sklearn import metrics
import numpy as np
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
# KNN
yhat = model_knn.predict(X_test)
yhat_knn = yhat
print("Train set KNN Accuracy: ", metrics.accuracy_score(y_train, model_knn.predict(X_train)))
print("Test set KNN Accuracy: ", metrics.accuracy_score(y_test, yhat))
jaccard = jaccard_similarity_score(y_test, yhat)
f1_score_knn = f1_score(y_test, yhat, average='weighted')
precision_knn = precision_score(y_test, yhat, average='weighted')
knn_report = ['KNN', round(jaccard,2), round(f1_score_knn,2), round(precision_knn,2)]

# Decission tree
yhat = model_tree.predict(X_test)
yhat_tree = yhat
print("Train set Decission Tree Accuracy: ", metrics.accuracy_score(y_train, model_tree.predict(X_train)))
print("Test set Decission Tree Accuracy: ", metrics.accuracy_score(y_test, yhat))
jaccard = jaccard_similarity_score(y_test, yhat)
f1_score_tree = f1_score(y_test, yhat, average='weighted')
precision_tree = precision_score(y_test, yhat, average='weighted')
tree_report = ['Decision Tree', round(jaccard,2), round(f1_score_tree,2), round(precision_tree,2)]

# Logistic regression
yhat_proba = model_lr.predict_proba(X_test)
yhat = model_lr.predict(X_test)
yhat_lr = yhat
print("Train set Logistic regression Accuracy: ", metrics.accuracy_score(y_train, model_lr.predict(X_train)))
print("Test set Logistic regression Accuracy: ", metrics.accuracy_score(y_test, yhat))
jaccard = jaccard_similarity_score(y_test, yhat)
f1_score_lr = f1_score(y_test, yhat, average='weighted')
precision_lr = precision_score(y_test, yhat, average='weighted')
lr_report = ['Logistic Regression', round(jaccard,2), round(f1_score_lr,2), round(precision_lr,2)]
```

```
Train set KNN Accuracy:  0.7111778510575683
Test set KNN Accuracy:  0.7363676379963024
Train set Decission Tree Accuracy:  0.7479191977120607
Test set Decission Tree Accuracy:  0.7483581301170877
Train set Logistic regression Accuracy:  0.6989156435583794
Test set Logistic regression Accuracy:  0.6997799102033629
```

```
In [31]: report = pd.DataFrame(data=np.array([knn_report, tree_report, lr_report]),
                              columns=['Algorithm', 'Jaccard', 'F1-score', 'Precision'])
report
```

Out[31]:

	Algorithm	Jaccard	F1-score	Precision
0	KNN	0.74	0.67	0.74
1	Decision Tree	0.75	0.69	0.78
2	Logistic Regression	0.7	0.58	0.68

The Jaccard score is used to compare a set of predicted labels for a sample to the corresponding set of labels. Here we can see that for all 3 models it is above 70%
The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The Decision Tree model presents the best F1 score.
Precision-Recall is a useful measure of success of prediction when the classes are very imbalanced. Again the Decision Model presents the best score

In [32]:
```python
from sklearn.metrics import confusion_matrix

print('KNN Confusion Matrix')
tn, fp, fn, tp = confusion_matrix(y_test, yhat_knn).ravel()
(tn, fp, fn, tp)
```

```
KNN Confusion Matrix
```

Out[32]:
```
(38708, 999, 13974, 3114)
```

In [33]:
```python
print('Decision Tree Confusion Matrix')
tn, fp, fn, tp = confusion_matrix(y_test, yhat_tree).ravel()
(tn, fp, fn, tp)
```

```
Decision Tree Confusion Matrix
```

Out[33]:
```
(39170, 537, 13755, 3333)
```

In [34]:
```python
print('Logistic Regression Confusion Matrix')
tn, fp, fn, tp = confusion_matrix(y_test, yhat_lr).ravel()
(tn, fp, fn, tp)
```

```
Logistic Regression Confusion Matrix
```

Out[34]:
```
(39655, 52, 16999, 89)
```

The confusion matrixes show the number of samples which classified correctly. We can see a big difference when comparing false positives and true positives, while there is a smaller difference with true negatives and false negatives

Discussion

We evaluated the performance of 3 machine learning algorithms on the given dataset in order to answer the question of predicting the severity of an accident knowing the weather and road conditions. The results of this performance for the 3 models was close, there was not a big difference in each of them. The Decision Tree model proved to be have the highest score in each category, followed by the KNN and then Logistic Regression.

Conclusion

For this project 5 out of 37 features were used which proved to be an adequate amount to resolve the problem. However if more features were used then we may have gotten different results. Additional features may also extract further data which could improve our models.