Introduction and Business Problem

Car accidents can be caused due to various reasons. Some of which include; road conditions, weather and visibility conditions. The extent of these factors also affects the severity of the accidents. For example, worse weather conditions lead to accidents more severe than accidents in more favourable weather. This Project aims to draw a connection between these 3 factors and the severity of road accidents using machine learning to create a model. With the help of the model and its results, drivers can accordingly react to prevent accidents.

The Data

The data used here has been provided by the Seattle Police Department and recorded by Traffic Records. This data includes all types of collisions. Collisions will display at the intersection or mid-block of a segment. The time frame of the data is from 2004 to present day. The data had been updated on a weekly basis. For this project, the required columns from the data that will positively affect the machine learning model are "WEATHER", "ROADCOND" and "LIGHTCOND". The "SEVERITYCODE" column has some numbers and represent road accident severity in the following way:

- 3 – fatality
- 2b – serious injury
- 2 – injury
- 1 – prop damage
- 0 – unknown

Methodology

The data was first downloaded and imported to the notebook using the Pandas library. After preliminary inspection of the data, it was inferred that filtering and cleaning was required. The variable that required prediction was the "SEVERITYCODE". Predicting this variable meant trying to find independent variable(s) that have a meaningful effect on the SEVERITYCODE.

Looking at the metadata, it had been inferred that three columns could be considered as the independent variables and these were; WEATHER, ROADCOND, and LIGHTCOND. The data types of the three variables were categorical and had to be converted to their respective categorical codes (in int64).

```
In [43]: # Drop all columns with no predictive value for the context of this project
df = df.drop(columns = ['OBJECTID', 'SEVERITYCODE.1', 'REPORTNO', 'INCKEY', 'COLDETKEY',
              'X', 'Y', 'STATUS','ADDRTYPE',
              'INTKEY', 'LOCATION', 'EXCEPTRSNCODE',
              'EXCEPTRSNDESC', 'SEVERITYDESC', 'INCDATE',
              'INCDTTM', 'JUNCTIONTYPE', 'SDOT_COLCODE',
              'SDOT_COLDESC', 'PEDROWNOTGRNT', 'SDOTCOLNUM',
              'ST_COLCODE', 'ST_COLDESC', 'SEGLANEKEY',
              'CROSSWALKKEY', 'HITPARKEDCAR', 'PEDCOUNT', 'PEDCYLCOUNT',
              'PERSONCOUNT', 'VEHCOUNT', 'COLLISIONTYPE',
              'SPEEDING', 'UNDERINFL', 'INATTENTIONIND'])

# Convert column to category
df["WEATHER"] = df["WEATHER"].astype('category')
df["ROADCOND"] = df["ROADCOND"].astype('category')
df["LIGHTCOND"] = df["LIGHTCOND"].astype('category')

# Assign variable to new column for analysis
df["WEATHER_CAT"] = df["WEATHER"].cat.codes
df["ROADCOND_CAT"] = df["ROADCOND"].cat.codes
df["LIGHTCOND_CAT"] = df["LIGHTCOND"].cat.codes

df.head(5)
```

Out[43]:

	SEVERITYCODE	WEATHER	ROADCOND	LIGHTCOND	WEATHER_CAT	ROADCOND_CAT	LIGHTCOND_CAT
0	2	Overcast	Wet	Daylight	4	8	5
1	1	Raining	Wet	Dark - Street Lights On	6	8	2
2	1	Overcast	Dry	Daylight	4	0	5
3	1	Clear	Dry	Daylight	1	0	5
4	2	Raining	Wet	Daylight	6	8	5

The next step in cleaning the data was to down sample the data with SEVERITYCODE of 1. This is done to avoid bias in prediction.

```
In [47]: df["SEVERITYCODE"].value_counts()

Out[47]: 1    136485
         2     58188
         Name: SEVERITYCODE, dtype: int64

In [48]: from sklearn.utils import resample

         df_majority = df[df.SEVERITYCODE==1]
         df_minority = df[df.SEVERITYCODE==2]

         #Downsample majority class
         df_majority_downsampled = resample(df_majority,
                                            replace=False,
                                            n_samples=58188,
                                            random_state=123)

         # Combine minority class with downsampled majority class
         df_balanced = pd.concat([df_majority_downsampled, df_minority])

         # Display new class counts
         df_balanced.SEVERITYCODE.value_counts()

Out[48]: 2    58188
         1    58188
         Name: SEVERITYCODE, dtype: int64
```

The data was then put into an array. The data was then split into testing and training samples. 75% of the data was used for training and 25% of the data was used for testing.

```
In [52]: # Split data into training set and testing set
         # The data has been split such that 75% of the dataset is the training set and 25% is the testing set

         from sklearn.model_selection import train_test_split

         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=4)

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

           Train set: (87282, 3) (87282,)
           Test set: (29094, 3) (29094,)
```

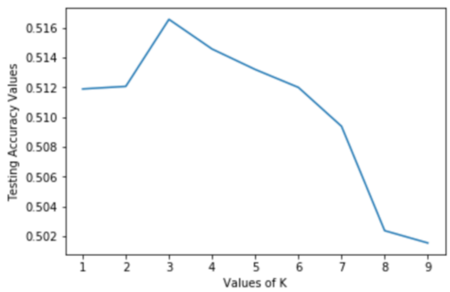
Training the model

Three different training techniques were used to generate a model:

1. K-Nearest Neighbour
   First a loop was created to create a model for different k values from 1 to 10. A graph was created to display how accurate the models were with different k values. It was determined that k = 3 gave the highest accuracy and was therefore used for training the model.

```
All Libraries Imported

k = 1 has a Score = 0.5118924864233175
k = 2 has a Score = 0.5120643431635389
k = 3 has a Score = 0.5165669897573383
k = 4 has a Score = 0.5145734515707706
k = 5 has a Score = 0.5131985976489998
k = 6 has a Score = 0.5119956004674503
k = 7 has a Score = 0.5093833780160858
k = 8 has a Score = 0.5023716230150547
k = 9 has a Score = 0.5015467106619922
```



2. Decision Tree

Decision Tree

```
In [56]: from sklearn import tree

         clf_tree = tree.DecisionTreeClassifier(criterion="entropy", max_depth = 10)
         clf_tree = clf_tree.fit(X_train, y_train)
         yhat_dt = clf_tree.predict(X_test)
         yhat_dt

Out[56]: array([2, 2, 1, ..., 2, 1, 2])

In [57]: print (yhat_dt [0:5])
         print (y_test [0:5])

         [2 2 1 1 2]
         [2 2 1 1 1]
```

3. Logistic Regression

Logistic Regression

```
In [58]: from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import confusion_matrix
         LR = LogisticRegression(C=6, solver='liblinear').fit(X_train,y_train)
         LR

Out[58]: LogisticRegression(C=6, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='warn',
                   n_jobs=None, penalty='l2', random_state=None, solver='liblinear',
                   tol=0.0001, verbose=0, warm_start=False)

In [59]: LRyhat = LR.predict(X_test)
         LRyhat

Out[59]: array([1, 2, 1, ..., 2, 1, 2])

In [60]: yhat_prob = LR.predict_proba(X_test)
         yhat_prob

Out[60]: array([[0.57263745, 0.42736255],
                [0.47083235, 0.52916765],
                [0.67438763, 0.32561237],
                ...,
                [0.47083235, 0.52916765],
                [0.68326727, 0.31673273],
                [0.46947758, 0.53052242]])

In [61]: print (LRyhat [0:5])
         print (y_test [0:5])

         [1 2 1 1 1]
         [2 2 1 1 1]
```

Results and Evaluation

The last step was to determine which technique was the best in terms of accuracy. The Jaccard Similarity Score and F1 score were calculated for each machine learning technique.

K Nearest Neighbour

In [63]:
```
# Jaccard Similarity Score
print("Jaccard Similarity Score: ", jaccard_similarity_score(y_test, knn_yhat))

# F1 SCORE
print("F1 Score:                 ", f1_score(y_test, knn_yhat, average='macro'))
```

```
Jaccard Similarity Score:  0.5165669897573383
F1 Score:                  0.46423304964062817
```

Decision Tree

In [64]:
```
# Jaccard Similarity Score
print("Jaccard Similarity Score: ", jaccard_similarity_score(y_test, yhat_dt))

# F1 SCORE
print("F1 Score:                 ", f1_score(y_test, yhat_dt, average='macro'))
```

```
Jaccard Similarity Score:  0.56688664329415
F1 Score:                  0.5431403441878632
```

Logistic Regression

In [65]:
```
# Jaccard Similarity Score
print("Jaccard Similarity Score: ", jaccard_similarity_score(y_test, LRyhat))

# F1 SCORE
print("F1 Score:                 ", f1_score(y_test, LRyhat, average='macro'))

# Log Loss
yhat_prob = LR.predict_proba(X_test)
print("Log Loss:                 ", log_loss(y_test, yhat_prob))
```

```
Jaccard Similarity Score:  0.5274283357393277
F1 Score:                  0.5127901543870825
Log Loss:                  0.6849604847680921
```

Calculating the Jaccard Similarity Score and the F1 Score revealed that the Decision Tree model outputted the most accurate results.

Conclusion

It is shown that the Decision Tree model is able to output the most accurate model and hence used to model the dataset.