1. Introduction

1.1 Background

Accidents happen all the time. And with the increasing number of vehicles out there, there is also an increase in the number of car accidents. There are a lot of factors to consider that may contribute to the occurrence of the accident such as the conditions of the road due to the weather or traffic. The condition of the drive must also be considered, whether he/she is under the influence of alcohol or is simply not paying attention at the time. We will be using machine learning algorithms to determine which of these factors they should look out for so that they may take certain precautions before heading out in the road.

1.2 Problem

We will be looking at car accident data in finding out what should a driver consider when heading out in order to avoid an accident. We will be looking at the severity of the accident.

1.3 Interest

Aside from car driver's, other parties that might be interested in this study would include insurance companies and local government agencies such as police and traffic enforcement. The model developed in this study would be able to provide some insights to the target audience on how to reduce the number of the car accidents happening.

2. Data

2.1. Data description

The data to be used for modelling will be taken from the Seattle area thru car accident information gathered by the Seattle Department of Transportation (SDOT). The data which is available in the SDOT website, will comprise variables on car accidents such as its severity, street, place of accident, weather among others.

Out[2]:

	SEVERITYCODE	X	Y	OBJECTID	INCKEY	COLDETKEY	REPORTNO	STATUS	ADDRTYPE	INTKEY	...	ROADCOND	LIGHTCOND	PEDROWNOTGRNT	SDOTCOLNUM	SPEEDING
0	2	-122.323148	47.703140	1	1307	1307	3502005	Matched	Intersection	37475.0	...	Wet	Daylight	NaN	NaN	NaN
1	1	-122.347294	47.647172	2	52200	52200	2607959	Matched	Block	NaN	...	Wet	Dark - Street Lights On	NaN	6354039.0	NaN
2	1	-122.334540	47.607871	3	26700	26700	1482393	Matched	Block	NaN	...	Dry	Daylight	NaN	4323031.0	NaN
3	1	-122.334803	47.604803	4	1144	1144	3503937	Matched	Block	NaN	...	Dry	Daylight	NaN	NaN	NaN
4	2	-122.306426	47.545739	5	17700	17700	1807429	Matched	Intersection	34387.0	...	Wet	Daylight	NaN	4028032.0	NaN

5 rows × 38 columns

The dependent variable that we will be using for this study is the SEVERITYCODE which has measurements on the degree on the accident as seen below based on information:

0: Unknown
1: Property Damage
2: Injury
2b: serious injury
3: fatality

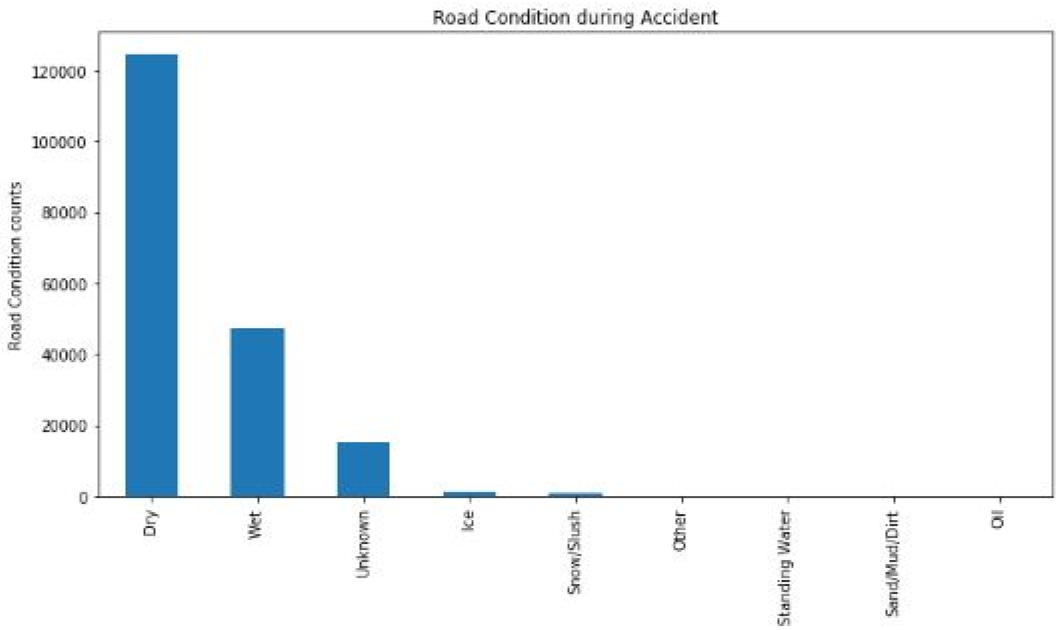
Despite the above, the data given on the SEVERITYCODE shows only ‘1’ or ‘2’.

```
Out[3]: 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
        COLLISIONTYPE       object
        PERSONCOUNT          int64
        PEDCOUNT             int64
        PEDCYLCOUNT          int64
        VEHCOUNT             int64
        INCDATE             object
        INCDTTM             object
        JUNCTIONTYPE        object
        SDOT_COLCODE         int64
        SDOT_COLDESC        object
        INATTENTIONIND      object
        UNDERINFL           object
        WEATHER             object
        ROADCOND            object
        LIGHTCOND           object
        PEDROWNOTGRNT       object
        SDOTCOLNUM         float64
        SPEEDING            object
        ST_COLCODE          object
        ST_COLDESC          object
        SEGLANEKEY           int64
        CROSSWALKKEY         int64
        HITPARKEDCAR        object
        dtype: object
```

Please see below statistical parameters of the data:

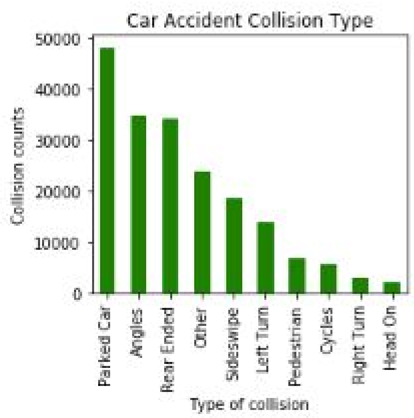
Out[4]:

	SEVERITYCODE	X	Y	OBJECTID	INCKEY	COLDETKEY	INTKEY	SEVERITYCODE.1	PERSONCOUNT	PEDCOUNT	PEDCYLCOUNT	VEHCOUNT	SDOT_COLCODE
count	194673.000000	189339.000000	189339.000000	194673.000000	194673.000000	194673.000000	65070.000000	194673.000000	194673.000000	194673.000000	194673.000000	194673.000000	194673.000000
mean	1.298901	-122.330518	47.619543	108479.364930	141091.456350	141298.811381	37558.450576	1.298901	2.444427	0.037139	0.028391	1.920780	13.867768
std	0.457778	0.029976	0.056157	62649.722558	86634.402737	86986.542110	51745.990273	0.457778	1.345929	0.198150	0.167413	0.631047	6.868755
min	1.000000	-122.419091	47.495573	1.000000	1001.000000	1001.000000	23807.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	-122.348673	47.575956	54267.000000	70383.000000	70383.000000	28667.000000	1.000000	2.000000	0.000000	0.000000	2.000000	11.000000
50%	1.000000	-122.330224	47.615369	106912.000000	123363.000000	123363.000000	29973.000000	1.000000	2.000000	0.000000	0.000000	2.000000	13.000000
75%	2.000000	-122.311937	47.663664	162272.000000	203319.000000	203459.000000	33973.000000	2.000000	3.000000	0.000000	0.000000	2.000000	14.000000
max	2.000000	-122.238949	47.734142	219547.000000	331454.000000	332954.000000	757580.000000	2.000000	81.000000	6.000000	2.000000	12.000000	69.000000


Road Condition during Accident
Road Condition counts
120000
100000
80000
60000
40000
20000
0
Dry
Wet
Unknown
Ice
Snow/Slush
Other
Standing Water
Sand/Mud/Dirt
Oil

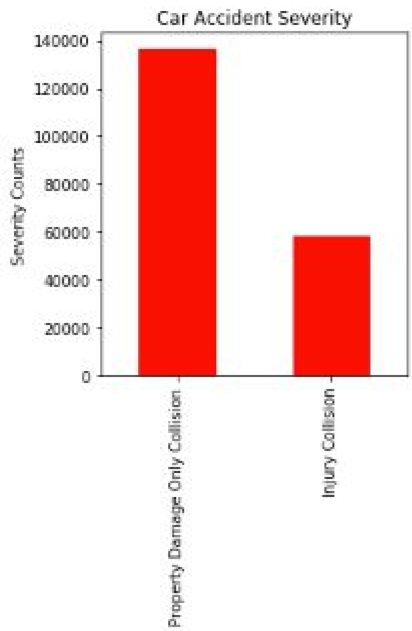
Out[21]:

	Collision Counts	Percentage of collissions
Parked Car	47987	25.287060
Angles	34674	18.271688
Rear Ended	34090	17.963946
Other	23703	12.490449
Sideswipe	18609	9.806133
Left Turn	13703	7.220884
Pedestrian	6608	3.482128
Cycles	5415	2.853469
Right Turn	2956	1.557683
Head On	2024	1.066560



Out[20]:

	Severity Counts	Severity Percentage
Property Damage Only Collision	136485	70.109877
Injury Collision	58188	29.890123



2.2 Data pre-processing

Before we continue with the modelling stage, the data must first be pre-processed . The first thing I did was remove the missing values from the dataset. Some columns were dropped because they were not relevant to the study. Another issue that was encountered was that a lot of the parameters were categorical data which needed to be converted to numerical data.

```
In [6]: #Drop unwanted variables
```

```
In [7]: df2 = 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'])
```

```
In [8]: #Change to categorical
```

```
In [9]: df2["ROADCOND"] = df2["ROADCOND"].astype('category')
        df2["WEATHER"] = df2["WEATHER"].astype('category')
        df2["LIGHTCOND"] = df2["LIGHTCOND"].astype('category')

        df2["ROADCOND_CATG"] = df2["ROADCOND"].cat.codes
        df2["WEATHER_CATG"] = df2["WEATHER"].cat.codes
        df2["LIGHTCOND_CATG"] = df2["LIGHTCOND"].cat.codes

        df2.dtypes
```

```
Out[9]: SEVERITYCODE        int64
        WEATHER          category
        ROADCOND         category
        LIGHTCOND        category
        ROADCOND_CATG        int8
        WEATHER_CATG         int8
        LIGHTCOND_CATG       int8
        dtype: object
```

Please see below new statistical parameters for the data: (Screenshot of .head)

```
In [10]: df2.head(5)
```

Out[10]:

	SEVERITYCODE	WEATHER	ROADCOND	LIGHTCOND	ROADCOND_CATG	WEATHER_CATG	LIGHTCOND_CATG
0	2	Overcast	Wet	Daylight	8	4	5
1	1	Raining	Wet	Dark - Street Lights On	8	6	2
2	1	Overcast	Dry	Daylight	0	4	5
3	1	Clear	Dry	Daylight	0	1	5
4	2	Raining	Wet	Daylight	8	6	5

We also have noticed that our target variable is not balance. This might skew our results and provide an inaccurate model. So what we did is to downsample the data to obtain a balanced dataset.

Before:

```
In [11]: df2["SEVERITYCODE"].value_counts()
```

```
Out[11]: 1    136485
         2     58188
         Name: SEVERITYCODE, dtype: int64
```

After:

```
In [16]: df2_class1 = df2[df2.SEVERITYCODE==1]
         df2_class2 = df2[df2.SEVERITYCODE==2]

         df2_class1_resampled = resample(df2_class1, replace=False,
                                         n_samples=58188,
                                         random_state=123)

         df2_bal = pd.concat([df2_class1_resampled,df2_class2])
         df2_bal.SEVERITYCODE.value_counts()

Out[16]: 2    58188
         1    58188
         Name: SEVERITYCODE, dtype: int64
```

3. Methodology

3.1 Import Python libraries

In order to create our model for car accident severity we will be using the Python programming language and its various libraries such as Pandas, Numpy, SciKitlearn, etc. All the coding will be done in Jupyter notebook and will be published in GitHub.

```
In [1]: import pandas as pd
        import numpy as np
        import pandas as pd
        import itertools
        import matplotlib.pyplot as plt
        from matplotlib.ticker import NullFormatter
        import pandas as pd
        import numpy as np
        import matplotlib.ticker as ticker
        from sklearn import preprocessing
        %matplotlib inline
        import matplotlib.pyplot as plt
        import matplotlib.image as mpimg

        from sklearn import preprocessing, svm, metrics, ensemble, tree
        from sklearn.preprocessing import OneHotEncoder, RobustScaler
        from sklearn.compose import make_column_transformer
        from sklearn.pipeline import Pipeline
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import accuracy_score, classification_report
        from sklearn.linear_model import LogisticRegression
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.naive_bayes import GaussianNB
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import f1_score
        from sklearn.metrics import jaccard_similarity_score
```

3.2 Training and Testing data

We will be splitting the data for training and testing. 30% will be used for training while 70% will be used for testing.

```
In [29]: X_train, X_test, Y_train, Y_test  = train_test_split(X,Y, test_size=0.3, random_state=42)
```

```
In [35]: print ('Training set:', X_train.shape,  Y_train.shape)
         print ('Testing set:', X_test.shape,  Y_test.shape)

         Training set: (81463, 3) (81463,)
         Testing set: (34913, 3) (34913,)
```

4. Modelling

In this section, the models we will be using are the below:

4.1 K-Nearest Neighbor (KNN)

KNN will be used to categorize the severity of an outcome based on other outcomes with the nearest data points at k distance.

```
In [36]: #K-Nearest Neighbors

In [37]: from sklearn.neighbors import KNeighborsClassifier
         k = 25

In [38]: neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train, Y_train)
         neigh

         KNNyhat = neigh.predict(X_test)
         KNNyhat[0:5]

Out[38]: array([1, 1, 1, 1, 1])
```

4.2 Decision Tree

We will be also using the Decision Tree method to categorize the possible severity code of an outcome as well as to examine the possible scenarios based on the inputs.

```
In [39]: #Decision Tree
```

```
In [40]: colDataTree = DecisionTreeClassifier(criterion="entropy", max_depth = 7)
         colDataTree
         colDataTree.fit(X_train,Y_train)
```

```
Out[40]: 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')
```

```
In [41]: DTyhat = colDataTree.predict(X_test)
         print (DTyhat [0:5])
         print (Y_test [0:5])
```

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

4.3 Logistic Regression

Logistic Regression is used to predict a binary outcome which can have only two results. Since we only have two possible outcomes in our Severity Code variable, we could expect that Logistic Regression would be a good modelling method given our data.

```
In [42]: #Logistic Regression
```

```
In [43]: lr = LogisticRegression(C=0.03, solver='liblinear').fit(X_train,Y_train)
         lr
```

```
Out[43]: LogisticRegression(C=0.03, 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 [44]: LRyhat = lr.predict(X_test)
         LRyhat

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

```
Out[44]: array([[0.40364293, 0.59635707],
                [0.53529771, 0.46470229],
                [0.46743605, 0.53256395],
                ...,
                [0.46293233, 0.53706767],
                [0.46743605, 0.53256395],
                [0.67878612, 0.32121388]])
```

4.4 Support Vector Machines(SVM)

Lastly, we will also be constructing a model using the SVM method to categorize the possible severity code of an outcome. (We will be using the RBF method since it has the highest Jaccard and F1 score compared to other SVM methods.)

```
In [52]: #Use RBF

         clf = svm.SVC(kernel='rbf')
         clf.fit(X_train, Y_train)
         RBYhat = clf.predict(X_test)
         RBYhat

           /opt/conda/envs/Python36/lib/python3.6/site-pa
           2 to account better for unscaled features. Set
             "avoid this warning.", FutureWarning)

Out[52]: array([1, 2, 2, ..., 1, 2, 1])
```

4.5 Random Forest Classification

Random Forest Classification creates a series of Decision Trees and selects the best one among them.

```
In [51]: from sklearn.ensemble import RandomForestClassifier

In [52]: clf=RandomForestClassifier(n_estimators=100)

         clf.fit(X_train,Y_train)

         RFY_pred=clf.predict(X_test)


         acc=accuracy_score(Y_test, RFY_pred)



         print("[Randon forest algorithm] accuracy_score: {:.3f}.".format(acc))

         [Randon forest algorithm] accuracy_score: 0.561.
```

5. Results

After constructing the different models as seen in the previous section, we will now be testing which model will best fit the data we have. To do this, we will be using evaluation methods such as the Jaccard Similarity Score, F-1 score and the Logloss for the Logistic Regression. The results are shown below:

K-Nearest Neighbors

In [58]:
```
#KNN Evaluation
```

In [59]:
```
# Jaccard Similarity Score
jaccard_similarity_score(Y_test, KNNyhat)
```

Out[59]: 0.5237017729785467

In [60]:
```
# F1-Score
f1_score(Y_test, KNNyhat, average='macro')
```

Out[60]: 0.5196155093297656

Decision Tree

In [61]:
```
#Decision Tree Evaluation
```

In [62]:
```
# Jaccard Similarity Score
jaccard_similarity_score(Y_test, DTyhat)
```

Out[62]: 0.5626843869045914

In [63]:
```
# F1-Score
f1_score(Y_test, DTyhat, average='macro')
```

Out[63]: 0.5385207275454998

Logistic Regression

In [64]:
```
#Logistic Regression Evaluation
```

In [65]:
```
# Jaccard Similarity Score
jaccard_similarity_score(Y_test, LRyhat)
```

Out[65]: 0.523501274596855

In [66]:
```
# F1-Score
f1_score(Y_test, LRyhat, average='macro')
```

Out[66]: 0.5098573271706865

In [67]:
```
# logloss
yhat_prob = lr.predict_proba(X_test)
log_loss(Y_test, yhat_prob)
```

Out[67]: 0.6855290309651024

Support Vector Machines (SVM)

```
In [68]: #SVM Evaluation

In [69]: # Jaccard Similarity Score
         jaccard_similarity_score(Y_test, RBYhat)

Out[69]: 0.5623979606450319

In [70]: # F1-Score
         f1_score(Y_test, RBYhat, average='macro')

Out[70]: 0.5386632235323728
```

Random Forest Classifier

```
In [79]: #Random Forest Classifier

In [80]: #Jaccard Similarity Score

In [81]: jaccard_similarity_score(Y_test, RFY_pred)

Out[81]: 0.5610231145991464

In [82]: # F1-Score
         f1_score(Y_test, RFY_pred, average='macro')

Out[82]: 0.5329956691944944
```

Based on the results above and the table below, The Decision Tree has the best Jaccard Score while closely being followed by SVM. Meanwhile, F-1 scores show the opposite where the SVM is slightly the best model with the Decision slightly next.

Model	**Jaccard SCore**	**F-1**	**Logloss**
KNN	0.5237	0.5196	
Decision Tree	0.5627	0.5385	
Logistic Regression	0.5235	0.5099	0.6855
SVM	0.5624	0.5387	
Random Forest	0.561	0.533	

6. Conclusions

For this study, we have developed several classification models to help predict the occurrence of the severity of a car accident. Given that they have shown good levels of accuracy we can say that there are definitely factors, such as the weather, road and light conditions. that lead into an accident.

7. Future directions

With respect to that, future studies could also use other models that would get additional insights. Other factors besides the variables used in this study may be used in the future. And if possible, a study on how these findings can affect existing road policies maybe done.