Prediction of Accident Severity

by

Justin Sierchio

A Report to fulfill the requirements for the

Applied Data Science Capstone Course

By

IBM/Coursera

August 27, 2020

Introduction and Business Problem

One of the leading causes of death and injury across the United States are traffic accidents. Almost 40,000 people are killed each year in traffic fatalities on American roads (https://en.wikipedia.org/wiki/Motor_vehicle_fatality_rate_in_U.S._by_year) . In order to better understand the reasons behind those accidents and possibly prevent them from happening, a data science study into the attributes leading to those accidents is warranted.

Since the sheer scale and complexity of such a task could take time and funds well beyond the scope of this course, a smaller study will be conducted. Specifically, we will be evaluating accident statistics for the municipality of Seattle, Washington for the previous 15 years. A study of this type will no doubt be of interest to drivers in the greater Seattle metropolitan area, but also pedestrians, city planners, emergency responders, road construction crews and even insurance companies.

Data

The data being used for the aforementioned study comes from the Seattle Police Department. The repository, found online at https://s3.us.cloud-object-storage.appdomain.cloud/cf-courses-data/CognitiveClass/DP0701EN/version-2/Data-Collisions.csv, contains records of every traffic incident from 2004 to the present, updated on a weekly basis. The data (contained in a .csv file), contains information such as the following.

- the severity of the accident
- type of collision
- number of fatalities and/or injuries
- weather conditions
- road conditions
- any pedestrians or non-automobiles involved and other factors.

Metadata explaining these attributes and more can be https://s3.us.cloud-object-storage.appdomain.cloud/cf-courses-data/CognitiveClass/DP0701EN/version-2/Metadata.pdf.

We will be particularly interested in how weather/road conditions, attentiveness/impairment of the driver and pedestrian/non-automobile involvement plays into traffic accident severity. Are there correlations between poor weather conditions and severity? Does the presence of pedestrians make an accident better or worse? How do road conditions affect accident severity? These are some of the questions we hope to be able to answer in this project.

Methodology

To perform this study, we will engage in each of the following steps:

- an exploratory data analysis to determine which variables are needed to construct a machine learning model
- choose an appropriate model for which to conduct the study
- training and testing the model using different classification algorithms and
- calculating the effectiveness of our model.

After uploading the initial dataset in IBM Watson Studio – Jupyter Notebook using the Python 3 programming language, we began cleaning the dataset to make it easier for conducting exploratory data analysis. We imported the Python 'Pandas' and 'Numpy' libraries to make the data manipulation easier. To begin, we noticed that several variables had the exact same data. As a result, we removed the duplicate columns with the extra variables:

```
In [7]: # Remove 'SEVERITYCODE.1', 'SEVERITYDESC'
df= df.drop(['SEVERITYDESC', 'SEVERITYCODE.1'], axis=1)
df.head()
```

Out[7]:

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

5 rows × 36 columns

Subsequently, we noticed that there were values in the WEATHER, ROADCOND, and LIGHTCOND columns that were unclear. Hence, we convert those values to 'NaN' (Not a Number) so they could easily be removed later:

```
In [9]: # Changing unclear values in 'WEATHER', 'ROADCOND' and 'LIGHTCOND'

# WEATHER -- 'Unknown', 'Other' -- NaN
df['WEATHER'].replace(to_replace=['Unknown','Other'], value=[np.nan,np.nan],inplace=True)

# ROADCOND -- 'Unknown', 'Other' -- NaN
df['ROADCOND'].replace("Unknown", np.nan, inplace = True)
df['ROADCOND'].replace("Other", np.nan, inplace = True)

# LIGHTCOND -- 'Unknown', 'Other' -- NaN
df['LIGHTCOND'].replace("Unknown", np.nan, inplace = True)
df['LIGHTCOND'].replace("Other", np.nan, inplace = True)
```

Next, we filled in missing values for remaining columns of interest. Specifically, we converted all the 'No' values to 0 and 'Yes' values to 1:

```
In [10]: # Filling in missing values for 'INATTENTIONIND', 'UNDERINFL', 'PEDROWNOTGRNT' and 'SPEEDING'

# INATTENIONIND:
# Change all 'Y' values to 1
# Make all non-'Y' values equal to 'N' (since the meaning is that the driver WAS attentive for a value of N or No)
# Change all 'N' values to 0
df['INATTENTIONIND'].fillna(0, inplace=True)
df['INATTENTIONIND'].replace("N", 0, inplace=True)
df['INATTENTIONIND'].replace("Y", 1, inplace = True)

# UNDERINFL:
# Change all values to quantitative values
# Change all empty cells to 'N' (since the meaning is that the driver was NOT under the influence)
# Change all 'N' values to 0
# Change all 'Y' values to 1
df['UNDERINFL'].fillna(0, inplace=True)
df['UNDERINFL'].replace(to_replace=['N','Y'], value=[0,1],inplace=True)
df['UNDERINFL']=df['UNDERINFL'].astype(dtype='int64')

# PEDROWNOTGRNT:
# Change all empty cells to 'N' (since the meaning is that the pedestrian right-of-way WAS granted)
# Change all 'N' values to 0
# Change all 'Y' values to 1
df['PEDROWNOTGRNT'].fillna(0, inplace=True)
df['PEDROWNOTGRNT'].replace("N", 0, inplace=True)
df['PEDROWNOTGRNT'].replace("Y", 1, inplace = True)

# SPEEDING:
# Change all empty cells to 'N' (Since the meaning is that the driver was NOT speeding)
# Change all 'N' values to 0
# Change all 'Y' values to 1
df['SPEEDING'].fillna(0, inplace=True)
df['SPEEDING'].replace(to_replace='Y', value=1,inplace=True)
```

The results of these actions demonstrated we had filled the entire dataset with actual values:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 194673 entries, 0 to 194672
Data columns (total 36 columns):
SEVERITYCODE      194673 non-null int64
X                 189339 non-null float64
Y                 189339 non-null float64
OBJECTID          194673 non-null int64
INCKEY            194673 non-null int64
COLDETKEY         194673 non-null int64
REPORTNO          194673 non-null object
STATUS            194673 non-null object
ADDRTYPE          192747 non-null object
INTKEY            65070 non-null float64
LOCATION          191996 non-null object
EXCEPTRSNCODE     84811 non-null object
EXCEPTRSNDESC     5638 non-null object
COLLISIONTYPE     189769 non-null object
PERSONCOUNT       194673 non-null int64
PEDCOUNT          194673 non-null int64
PEDCYLCOUNT       194673 non-null int64
VEHCOUNT          194673 non-null int64
INCDATE           194673 non-null object
INCDTTM           194673 non-null object
JUNCTIONTYPE      188344 non-null object
SDOT_COLCODE      194673 non-null int64
SDOT_COLDESC      194673 non-null object
INATTENTIONIND    194673 non-null int64
UNDERINFL         194673 non-null int64
WEATHER           173669 non-null object
ROADCOND          174451 non-null object
LIGHTCOND         175795 non-null object
PEDROWNOTGRNT     194673 non-null int64
SDOTCOLNUM        114936 non-null float64
SPEEDING          194673 non-null int64
ST_COLCODE        194655 non-null object
ST_COLDESC        189769 non-null object
SEGLANEKEY        194673 non-null int64
CROSSWALKKEY      194673 non-null int64
HITPARKEDCAR      194673 non-null object
dtypes: float64(4), int64(15), object(17)
memory usage: 53.5+ MB
```

From that point, we chose certain variables for our feature selection subset, including those related to weather and road conditions, location of accidents, numbers of pedestrians, number of vehicles, lighting conditions, driver impairment and attentiveness, whether the driver was speeding, and so on.

```
In [12]: # Column Choice for Feature Selections
         df_featureSelection = df[['SEVERITYCODE','ADDRTYPE','PERSONCOUNT','PEDCOUNT','PEDCYLCOUNT','VEHCOUNT','INATTENTIONIND','UNDERINFL','WEATHER','ROADCOND','LIGHTCOND','PEDROWNOTGRNT','SPEEDING']]

         # Number of 'NULL' values remaining among the Feature Selection subsete
         df_featureSelection.isnull().sum()

Out[12]: SEVERITYCODE          0
         ADDRTYPE           1926
         PERSONCOUNT           0
         PEDCOUNT              0
         PEDCYLCOUNT           0
         VEHCOUNT              0
         INATTENTIONIND        0
         UNDERINFL             0
         WEATHER           21004
         ROADCOND          20222
         LIGHTCOND         18878
         PEDROWNOTGRNT         0
         SPEEDING              0
         dtype: int64
```

Next, we converted all the categorical (binary) variables to numerical variables for easier processing later.

```
In [15]: # Change Categorical Variables into Numerical Variables

         # ADDRTYPE
         # Convert into 3 columns representing 'Alley', 'Block', 'Intersection'
         df_featureSelection[['Alley', 'Block', 'Intersection']] = pd.get_dummies(df_featureSelection['ADDRTYPE'])
         df_featureSelection.drop(['ADDRTYPE'], axis=1, inplace=True)

         # WEATHER
         # Convert into several columns
         df_featureSelection[pd.get_dummies(df_featureSelection['WEATHER']).columns] = pd.get_dummies(df_featureSelection['WEATHER'])
         df_featureSelection.drop(['WEATHER'], axis=1, inplace=True)

         # ROADCOND
         # Convert into several columns
         df_featureSelection[pd.get_dummies(df_featureSelection['ROADCOND']).columns] = pd.get_dummies(df_featureSelection['ROADCOND'])
         df_featureSelection.drop(['ROADCOND'], axis=1, inplace=True)

         # LIGHTCOND
         # Convert into several columns
         df_featureSelection[pd.get_dummies(df_featureSelection['LIGHTCOND']).columns] = pd.get_dummies(df_featureSelection['LIGHTCOND'])
         df_featureSelection.drop(['LIGHTCOND'], axis=1, inplace=True)
```

This action resulted in our final feature selection subset for which the first 5 rows are shown below:

Out[16]:

	SEVERITYCODE	PERSONCOUNT	PEDCOUNT	PEDCYLCOUNT	VEHCOUNT	INATTENTIONIND	UNDERINFL	PEDROWNOTGRNT	SPEEDING	Alley	...	Snow/Slush	Standing Water	Wet	Dark - No Street Lights	Dark - Street Lights Off	Dark - Street Lights On	Dark - Unknown Lighting	Dawn	Daylight	Dusk
0	2	2	0	0	2	0	0	0	0	0	...	0	0	1	0	0	0	0	0	1	0
1	1	2	0	0	2	0	0	0	0	0	...	0	0	1	0	0	1	0	0	0	0
2	1	4	0	0	3	0	0	0	0	0	...	0	0	0	0	0	0	0	0	1	0
3	1	3	0	0	3	0	0	0	0	0	...	0	0	0	0	0	0	0	0	1	0
4	2	2	0	0	2	0	0	0	0	0	...	0	0	1	0	0	0	0	0	1	0

5 rows × 35 columns

From this point, we began our exploratory data analysis, starting with creating a series of correlation matrices to study the potential correlations between certain types of data. Specifically, we wanted to see if the severity, denoted in our code as 'SEVERITYCODE', of the accident was related to (a) weather and road conditions (b) lighting conditions (c) driver impairment or inattentiveness (d) accident location (e) presence of pedestrians or bicyclists and (f) any speeding or failing to yield right of way to a pedestrian. The next several heatmaps, generated with the Python library Seaborn, are shown below:

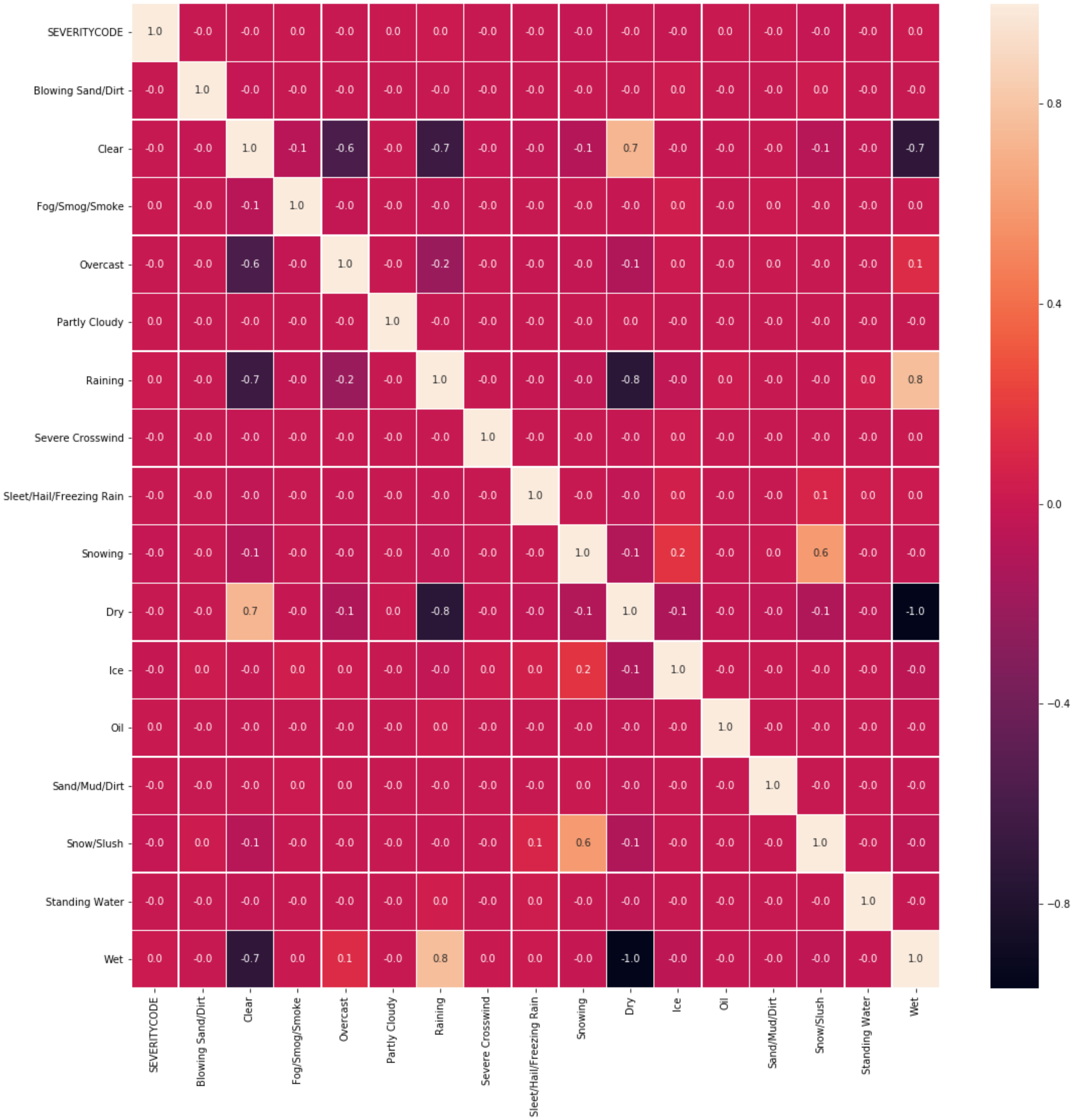

Figure 1 - Correlation Heatmap - Weather and Road Conditions vs. Accident Severity

The above heatmap showed no correlation between weather or road conditions and accident severity.

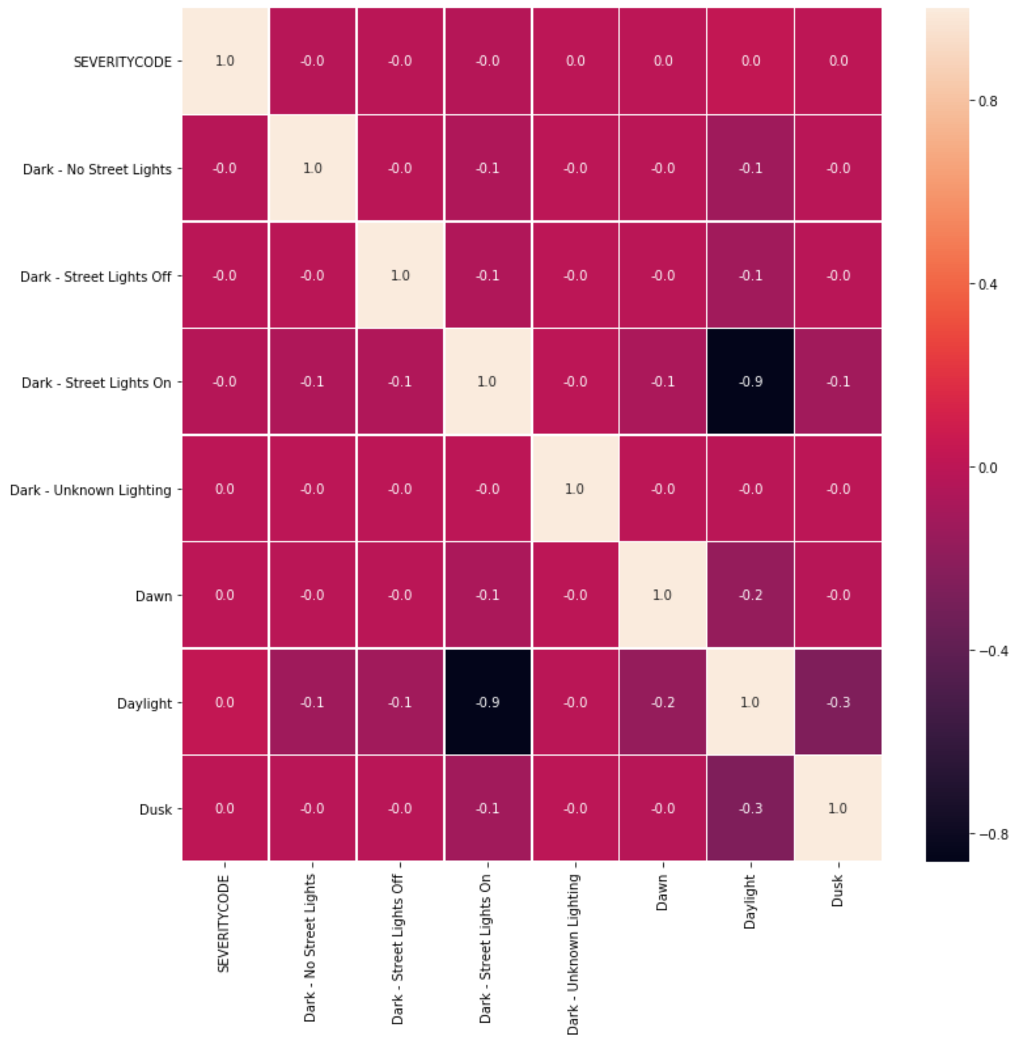

Figure 2 - Correlation Heatmap - Lighting Conditions vs. Accident Severity

The above heatmap showed no correlation between the lighting conditions and accident severity. We confirmed this finding by plotting accident frequency as a function of the relative time of day.

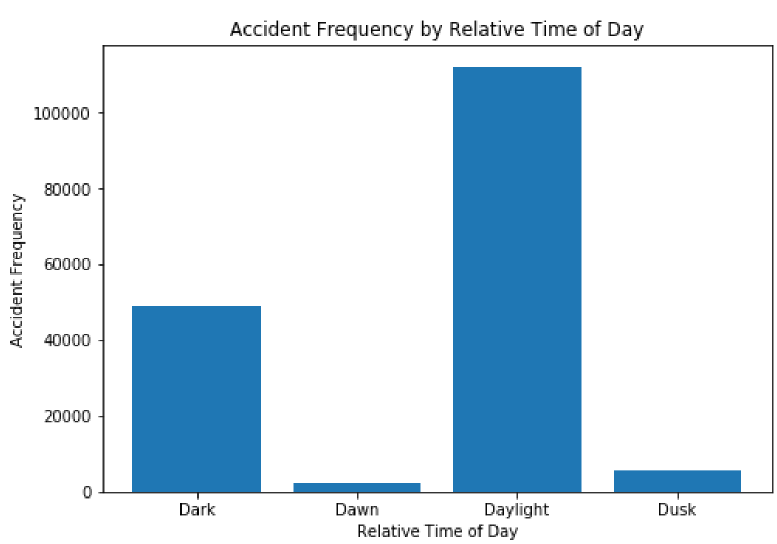

Figure 3 - Accident Frequency by Relative Time of Day

As shown in the graph above, the majority of accidents occur during the day. That statement makes logical sense, as most drivers are in their vehicles during the day (usually commuting to and from their place of work or school).

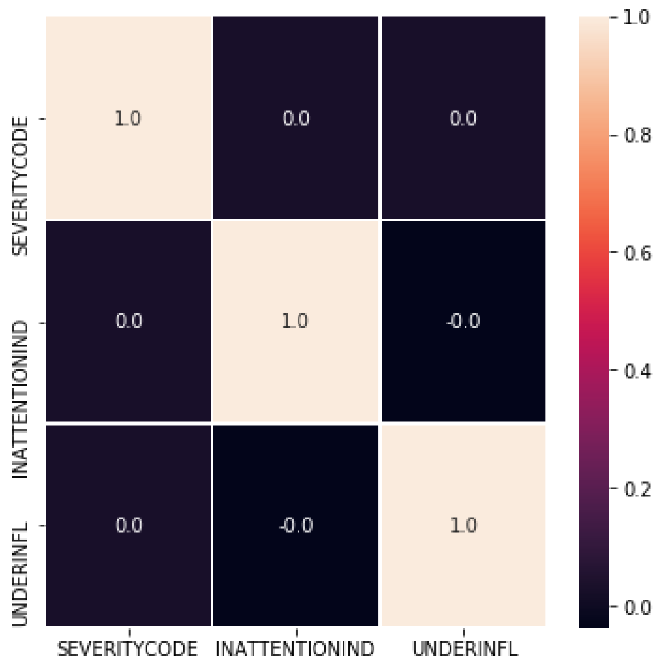

Figure 4 - Correlation Heatmap - Driver Impairment vs. Accident Severity

The above heatmap showed no correlation between the driver impairment or attentiveness and accident severity.

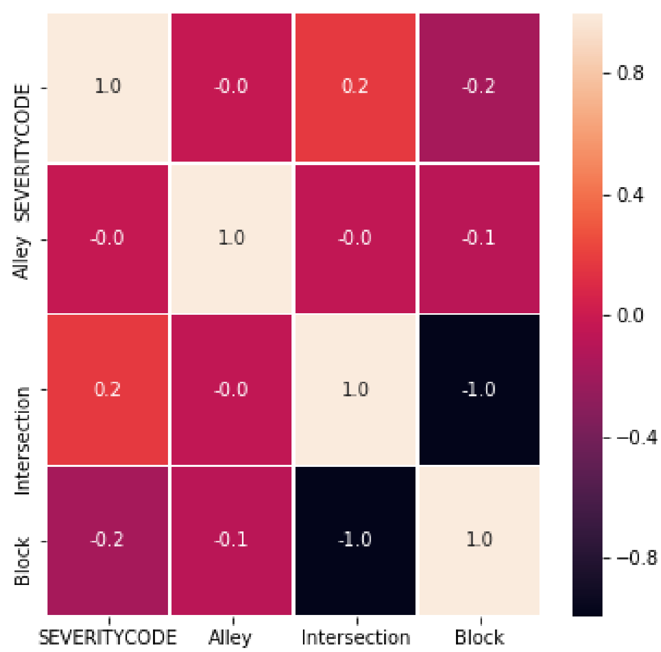

Figure 5 - Correlation Heatmap - Location Type vs. Accident Severity

The above heatmap showed a small correlation with accident severity and intersections. We the attempted to confirm this by finding the accident frequency and severity for each of the three location types collected: alleys, intersections, and blocks:

Table 1 - Location Type vs. Accident Frequency

Out[23]:

	Accident Location	Severity = 1	Severity = 2	% Difference btwn Sev=1, Sev=2
0	Alley	516	77	0.850775
1	Intersection	34463	26808	0.222122
2	Block	78726	28657	0.635991

As one can see, there is only 22.2% difference between intersection severity, as opposed to an 85.1% difference for alleys and 63.6% difference for blocks, confirming our original conclusion.

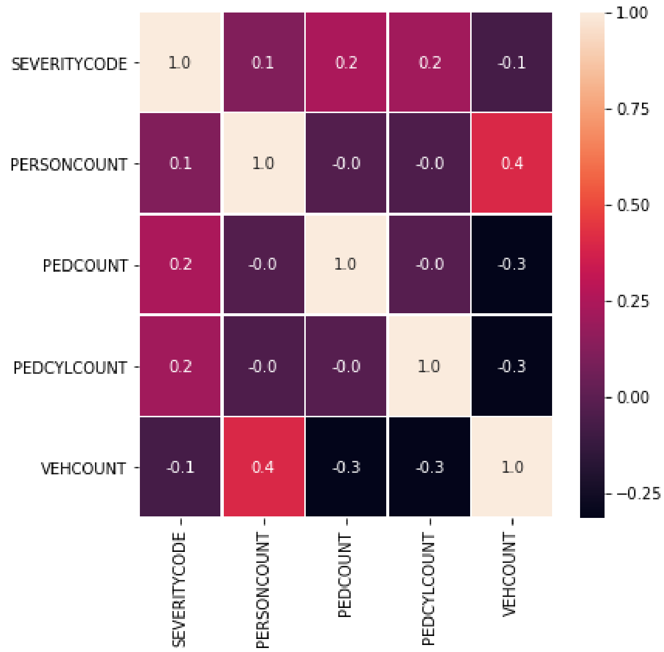

Figure 6 - Pedestrian and Bicycle Presence vs. Accident Severity

The above heatmap showed a correlation between the number of pedestrians and bicyclists and accident severity.

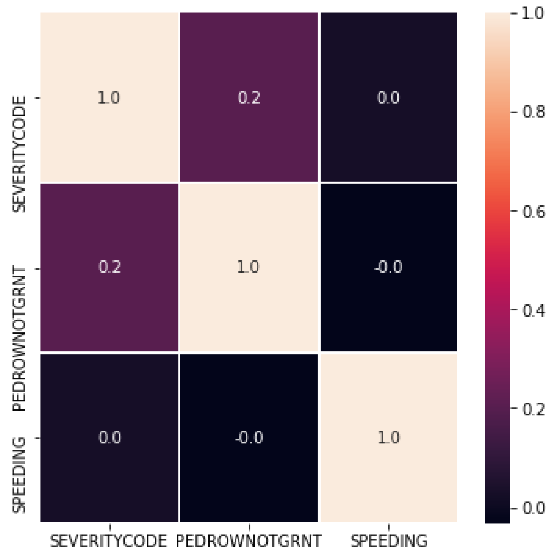

Figure 7 - Driver Speeding and Lack of Pedestrian ROW vs. Accident Severity

Finally, we showed in the above heatmap a correlation between failing to yield the right of way to a pedestrian and accident severity.

In summary from the previous heatmaps, we were able to show the following:

- weather conditions have little impact on accident severity
- road conditions have little impact on accident severity
- lighting conditions have little impact on accident severity
- accident severity is greater in intersections
- accident severity increases with pedestrians and bicycles present
- driver impairment has little impact on accident severity
- speeding has little impact on accident severity

Finally, we trained and tested machine learning models to see how well we could predict the severity of an accident. During the course of his certification program, we learned about four such classification algorithms:

- K-Nearest Neighbor
- Support Vector Machine
- Decision Trees
- Logistic Regression

While ideally it would have been worthwhile to investigate all four options, the K-Nearest Neighbor and Support Vector Machine algorithms often caused IBM Watson Studio – Jupyter Notebook to malfunction and freeze, requiring interrupting the kernel during processing. These malfunctions are likely due to the immense memory requirements (based upon the size of the original dataset and any relationships between the variables themselves). As a result, only the Decision Trees and Logistic Regression algorithms were investigated.

To begin the machine learning model development, we imported the Python library 'Sci-kit Learn,' allowing us to import the 'test-train split' methods and several summary statistical functions.

```
In [26]: # Import Machine Learning Libraries
         from sklearn import preprocessing
         from sklearn.model_selection import train_test_split

         # Import Machine Learning Metrics
         from sklearn.metrics import jaccard_similarity_score, f1_score, precision_score, recall_score
```

Next, we normalized the feature selection subset and created the training (80% of the dataset) and test sets (2% of the dataset) accordingly:

```
In [27]: # Normalize Feature Selection subset using the Standard Scaler and Fit
         machineLearningSet = df_featureSelection.drop(['SEVERITYCODE'],axis=1)
         machineLearningSet = preprocessing.StandardScaler().fit(machineLearningSet).transform(machineLearningSet)

         # Display the Normalized Set, setting the Machine Learning Set to 'X' to incorporate into the 'TrainX, TestY' methodology
         X = machineLearningSet;
         X[0:5]
```

```
In [28]: # Create the Target Value Variable
         machineLearningTarget = df_featureSelection['SEVERITYCODE'].values

         # Dsplay the Target Value Set
         y = machineLearningTarget;
         y[0:5]

Out[28]: array([2, 1, 1, 1, 2])
```

```
In [29]: # Create Training and Test Sets
         X_train , X_test , y_train, y_test = train_test_split(X,y, test_size = 0.2 , random_state=10)

         # Display the Training Set
         print ('Train set:', X_train.shape,  y_train.shape)

         # Display the Test Set
         print ('Test set:', X_test.shape,  y_test.shape)

         Train set: (135397, 34) (135397,)
         Test set: (33850, 34) (33850,)
```

We then investigated the Decision Tree algorithm to develop a machine learning model. Evaluating the Jaccard Similarity Score and f-1 Scores, we determined a depth of ‘8’ would optimize the algorithm:

In [31]:
```
# Create Jaccard and F1 Score Objects
depthRange = range(1, 9)
jaccardSimilarityScore = []
f1Score = []

# For Varying Depths, calculate jaccard similarity and f1 scores
for n in depthRange:
    dTree = DecisionTreeClassifier(criterion = 'gini', max_depth = n)
    dTree.fit(X_train, y_train)
    dTree_yhat = dTree.predict(X_test)
    jaccardSimilarityScore.append(jaccard_similarity_score(y_test, dTree_yhat))
    f1Score.append(f1_score(y_test, dTree_yhat, average = 'weighted'))
```

In [32]:
```
# Present the resulting data in a DataFrame
dTree_result = pd.DataFrame([jaccardSimilarityScore, f1Score],
                            index = ['Jaccard Sim', 'F1'], columns = ['d = 1','d = 2','d = 3','d = 4','d = 5','d = 6','d = 7', 'd = 8'])
dTree_result.columns.name = 'Depths'
dTree_result
```

Out[32]:

Depths	d = 1	d = 2	d = 3	d = 4	d = 5	d = 6	d = 7	d = 8
Jaccard Sim	0.706470	0.729897	0.729897	0.729867	0.731640	0.732555	0.734032	0.735923
F1	0.616746	0.664202	0.664202	0.664151	0.675052	0.688503	0.684060	0.690943

We performed a similar analysis with the Logistic Regression algorithm and found that the 'Lib-Linear' solver and a regularization value of 0.01 provided the highest accuracy scores:

In [35]:
```
# Create Solver and Accuracy Score Objects
solverList = ['lbfgs', 'saga', 'liblinear', 'newton-cg', 'sag']
regularizationValueSet = [0.1, 0.01, 0.001]
index = []
lrAccuracy = []
iterations = 0

for p, q in enumerate(regularizationValueSet):
    for r, s in enumerate(solverList):
        index.append(p + r *5)
        iterations +=1
        lrModel = LogisticRegression(C = q, solver = s)
        lrModel.fit(X_train, y_train)
        lr_yhat = lrModel.predict(X_test)
        y_prob = lrModel.predict_proba(X_test)
        print('Test {}: With C = {} for solver = {}, LR Accuracy is : {}'.format(iterations, q, s, log_loss(y_test, y_prob) ))
        lrAccuracy.append(log_loss(y_test, y_prob))

    print('\n')
```

```
Test 1: With C = 0.1 for solver = lbfgs, LR Accuracy is : 0.5557660528012474
Test 2: With C = 0.1 for solver = saga, LR Accuracy is : 0.5557657197099635
Test 3: With C = 0.1 for solver = liblinear, LR Accuracy is : 0.5557664324634823
Test 4: With C = 0.1 for solver = newton-cg, LR Accuracy is : 0.5557659873630711
Test 5: With C = 0.1 for solver = sag, LR Accuracy is : 0.5557665902061812


Test 6: With C = 0.01 for solver = lbfgs, LR Accuracy is : 0.5557784963316049
Test 7: With C = 0.01 for solver = saga, LR Accuracy is : 0.5557786432657238
Test 8: With C = 0.01 for solver = liblinear, LR Accuracy is : 0.5557831061701002
Test 9: With C = 0.01 for solver = newton-cg, LR Accuracy is : 0.5557784099363405
Test 10: With C = 0.01 for solver = sag, LR Accuracy is : 0.5557783761980235


Test 11: With C = 0.001 for solver = lbfgs, LR Accuracy is : 0.5561091101536626
Test 12: With C = 0.001 for solver = saga, LR Accuracy is : 0.5561102050102353
Test 13: With C = 0.001 for solver = liblinear, LR Accuracy is : 0.5562142720546198
Test 14: With C = 0.001 for solver = newton-cg, LR Accuracy is : 0.5561095344008128
Test 15: With C = 0.001 for solver = sag, LR Accuracy is : 0.5561097202183379
```

With the two machine models constructed, we compared their predictions using four summary metrics: Jaccard Similarity Scores, Precision Scores, Recall Scores, and f-1 Scores:

```python
In [39]: # Decision Tree Model Scores
f1_dTree = f1_score(y_test, dTree_yhat, average='weighted')
jss_dTree = jaccard_similarity_score(y_test, dTree_yhat)
ps_dTree = precision_score(y_test, dTree_yhat)
rs_dTree = recall_score(y_test, dTree_yhat)

#Logistic Regression Model Scores
f1_lr = f1_score(y_test, lr_yhat, average='weighted')
jss_lr = jaccard_similarity_score(y_test, lr_yhat)
ps_lr = precision_score(y_test, lr_yhat)
rs_lr = recall_score(y_test, lr_yhat)

# Present Results
ModelResults = {'Classification Model': ['Decision Trees', 'Logistic Regression'], 'f1 Score': [f1_dTree, f1_lr],
               'Jaccard': [jss_dTree, jss_lr], 'Precision Score': [ps_dTree, ps_lr], 'Recall Score': [rs_dTree, rs_lr]};
df_ModelResults = pd.DataFrame(ModelResults);
df_ModelResults
```

Out[39]:

	Classification Model	f1 Score	Jaccard	Precision Score	Recall Score
0	Decision Trees	0.690943	0.735923	0.731966	0.959823
1	Logistic Regression	0.678794	0.732201	0.724842	0.971696

Figure 8 - Machine Learning Model Metric Scores

That concluded our data analysis.

Results

As one can see from the preceding analysis, we were able to achieve a 73.6% match between the training set and the test set using the Decision Tree algorithm (and a 73.2% match with the Logistic Regression algorithm. In other words, from the accident location, number of persons involved, presence of pedestrians and/or bicyclists, number of vehicles involved, attentiveness/impairment of the driver, speeding, relative time of day and weather/road conditions, we could pick out the severity of the accident in roughly 3 out of 4 incidents.

Furthermore, we were able to show that, by themselves, weather conditions, road conditions, lighting conditions, driver impairment and speeding have little correlation with the severity of accident. However, accident severity is increased in situations involving traffic intersections and the presence of pedestrians and/or bicyclists.

Discussion

The goal of this project was to be able to predict the severity of an accident based upon a number of factors. At the beginning, we were hoping to answer the following questions:

- Are there correlations between poor weather conditions and severity?
- Does the presence of pedestrians make an accident better or worse?
- How do road conditions affect accident severity?

In each of these cases, we were able, within the constraints of time and hardware (namely IBM Watson Studio's memory) to answer these questions with some accuracy. According to our analysis, there is little correlation between poor weather conditions and accident severity. Similarly, there is little correlation between poor road conditions and accident severity. However, the presence of pedestrians and bicyclists increases the likelihood of a severe traffic accident.

The lessons that can be learned from this project are twofold. One is to be very mindful of driving in intersections, as the likelihood of a severe accident is increased. And second is that when encountering pedestrians or bicyclists, extra caution should be taken.

Conclusion

In order to better understand the potential impacts (and possibly mitigate them), it is important to understand the attributes that contribute to severe accidents. In the case of Seattle, Washington, we were able to show some of the factors that contribute to accident severity. There is no doubt that the information gathered here will be useful to drivers, pedestrians, city planners, emergency responders and insurance companies going forward.

On a final note, if one was to expand upon this work in a future exercise, evaluating the accuracy with other classification tools (i.e. K-Nearest Neighbor, Support Vector Machine) would be invaluable. Moreover, perhaps evaluating the make and model of the vehicles involved would also provide insight into traffic safety.

Thank you for reading this report!