IBM

Applied Data Science Capstone

Capstone Project Report:- Car accident severity

Submitted by

Jay Joshi

Introduction

The Seattle government is going to prevent avoidable car accidents by employing methods that alert drivers, health system, and police to remind them to be more careful in critical situations.

In most cases, not paying enough attention during driving, abusing drugs and alcohol or driving at very high speed are the main causes of occurring accidents that can be prevented by enacting harsher regulations. Besides the aforementioned reasons, weather, visibility, or road conditions are the major uncontrollable factors that can be prevented by revealing hidden patterns in the data and announcing warning to the local government, police and drivers on the targeted roads.

In an effort to reduce the frequency of car collisions in a community, an algorithim must be developed to predict the severity of an accident given the current weather, road and visibility conditions. When conditions are bad, this model will alert drivers to remind them to be more careful.

Also, the target audience of the project is local Seattle government, police, rescue groups, and last but not least, car insurance institutes. The model and its results are going to provide some advice for the target audience to make insightful decisions for reducing the number of accidents and injuries for the city.

Data Understanding

The data was collected by the Seattle Police Department and Accident Traffic Records Department from 2004 to present. The data consists of 37 independent variables and 194,673 rows. The dependent variable, “SEVERITYCODE”, contains numbers that correspond to different levels of severity caused by an accident from 0 to 3.

A code that corresponds to the severity of the collision:

• 3—fatality

• 2b—serious injury

• 2—injury

• 1—prop damage

• 0—unknown

To accurately build a model to prevent future accidents and/or reduce their severity, we will use the following attributes—ADDRTYPE, WEATHER, ROADCOND, VEHCOUNT,

PERSONCOUNT.

```
In [7]:  1  data_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 194673 entries, 0 to 194672
Data columns (total 38 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   SEVERITYCODE    194673 non-null  int64
 1   X               189339 non-null  float64
 2   Y               189339 non-null  float64
 3   OBJECTID        194673 non-null  int64
 4   INCKEY          194673 non-null  int64
 5   COLDETKEY       194673 non-null  int64
 6   REPORTNO        194673 non-null  object
 7   STATUS          194673 non-null  object
 8   ADDRTYPE        192747 non-null  object
 9   INTKEY          65070 non-null   float64
 10  LOCATION        191996 non-null  object
 11  EXCEPTRSNCODE   84811 non-null   object
 12  EXCEPTRSNDESC   5638 non-null    object
 13  SEVERITYCODE.1  194673 non-null  int64
 14  SEVERITYDESC    194673 non-null  object
 15  COLLISIONTYPE   189769 non-null  object
 16  PERSONCOUNT     194673 non-null  int64
 17  PEDCOUNT        194673 non-null  int64
 18  PEDCYLCOUNT     194673 non-null  int64
 19  VEHCOUNT        194673 non-null  int64
 20  INCDATE         194673 non-null  object
 21  INCDTTM         194673 non-null  object
 22  JUNCTIONTYPE    188344 non-null  object
 23  SDOT_COLCODE    194673 non-null  int64
 24  SDOT_COLDESC    194673 non-null  object
 25  INATTENTIONIND  29805 non-null   object
 26  UNDERINFL       189789 non-null  object
 27  WEATHER         189592 non-null  object
 28  ROADCOND        189661 non-null  object
 29  LIGHTCOND       189503 non-null  object
 30  PEDROWNOTGRNT   4667 non-null    object
 31  SDOTCOLNUM      114936 non-null  float64
 32  SPEEDING        9333 non-null    object
 33  ST_COLCODE      194655 non-null  object
 34  ST_COLDESC      189769 non-null  object
 35  SEGLANEKEY      194673 non-null  int64
 36  CROSSWALKKEY    194673 non-null  int64
 37  HITPARKEDCAR    194673 non-null  object
dtypes: float64(4), int64(12), object(22)
memory usage: 56.4+ MB
```

Load Data From CSV File

```
In [2]:  1  data_df = pd.read_csv("Data-Collisions.csv")

C:\Users\JAY\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3063: DtypeWarning: Columns (33) have mixed types.Spe
cify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

```
In [3]:  1  data_df.head()
```

Out[3]:

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

5 rows × 38 columns

Data Cleaning

The data obtained from the Seattle Open Data website are not, in their original form, usable for the purposes of model building. There are a number of issues which must be addressed:

1. Columns containing useless/redundant data. These columns can be removed from the dataframe and include:

   * _OBJECTID_ - this is just an database key, but Pandas creates its own key on import.

   * _COLDETKEY_ - this is just a duplicate of INCKEY, and probably arises due to a cross-matching of tables on the Seattle Open Data website. We will keep INCKEY as a unique identifier for each accident, but do not need to keep the duplicate column

   * _REPORTNO_ - this is just another identifier for the accident, this time tying the record to the individual piece of paperwork that was filed to report the accident. This is not useful for building our model.

   * _STATUS_ - the meaning of this column is unclear (and is not explained in the Attribute Information metadata). The values are either "Matched" or "Unmatched".

   * _EXCEPTRSNCODE, EXCEPTRSNDESC_ - these columns are listed in the Attribute Information metadata, but their meanings are not explained. EXCEPTRSNCODE is blank/NaN in ~99% of the data, with 2,480 accidents having a non-blank entry, all of which are the same ("NEI"). According to EXCEPTRSNDESC this means "Not Enough Information".

   * _INCDATE_ - this column is just a duplicate of the more easily parsable INCDTTM

   * _SDOTCOLNUM_ - this is another unique identifier for each accident, however as we are planning to keep INCKEY, keeping this second unique identifier is redundant.

2. Rows which are missing information about some of the features which we expect will be key to building the model: As we can see from data_df.head, a number of accidents (accounting for ~15% of the dataset) have "Unknown" values for attributes like _WEATHER_ , _ROADCOND_ and _LIGHTCOND_ , or have these fields blank/NaN, which in practice means the same thing. As these are expected to be among the features which influence the likelihood and severity of accidents, we have to consider discarding these rows before training the model.

3. Presence of features with categorical values: In order to construct a model using Machine Learning techniques, we must take columns which contain categorical data and re-cast them in numeric form. Different techniques will be used to accomplish this, depending on the nature of the data in these columns.

* Some columns, such as _UNDERINFL_ and _HITPARKEDCAR_ contain a mixture of alphanumeric (Y/N or 1/0), boolean (True/False) and missing (NaN) data. To prepare the data for modelling it will be important to homogenise these data by treating 1, Y and True as equivalent (and setting these to 1) and treating 0, N and False as equivalent (setting these to 0). Furthermore, we can infer that missing values (represented with NaN) are equivalent to 0/False.

* Other columns contain labelled data (e.g. _WEATHERCOND_ , which takes one of a handata_dful of values such as _RAIN_ , _CLEAR_ , _SNOWING_ , etc). These can be prepared for modelling by using One-Hot Encoding, wherein a new column is created for each of the discrete values corresponding to the original variable/column, which is filled with 1s or 0s depending on the value in that column.

4. Incorporating timestamp information: The _INCDTTM_ column contains the incident date/time in alphanumeric form. This can be parsed to an actual timestamp using Pandas, and then separated in to separate columns for Hour, Day (of month) Day (of week), Month and Year to study temporal trends

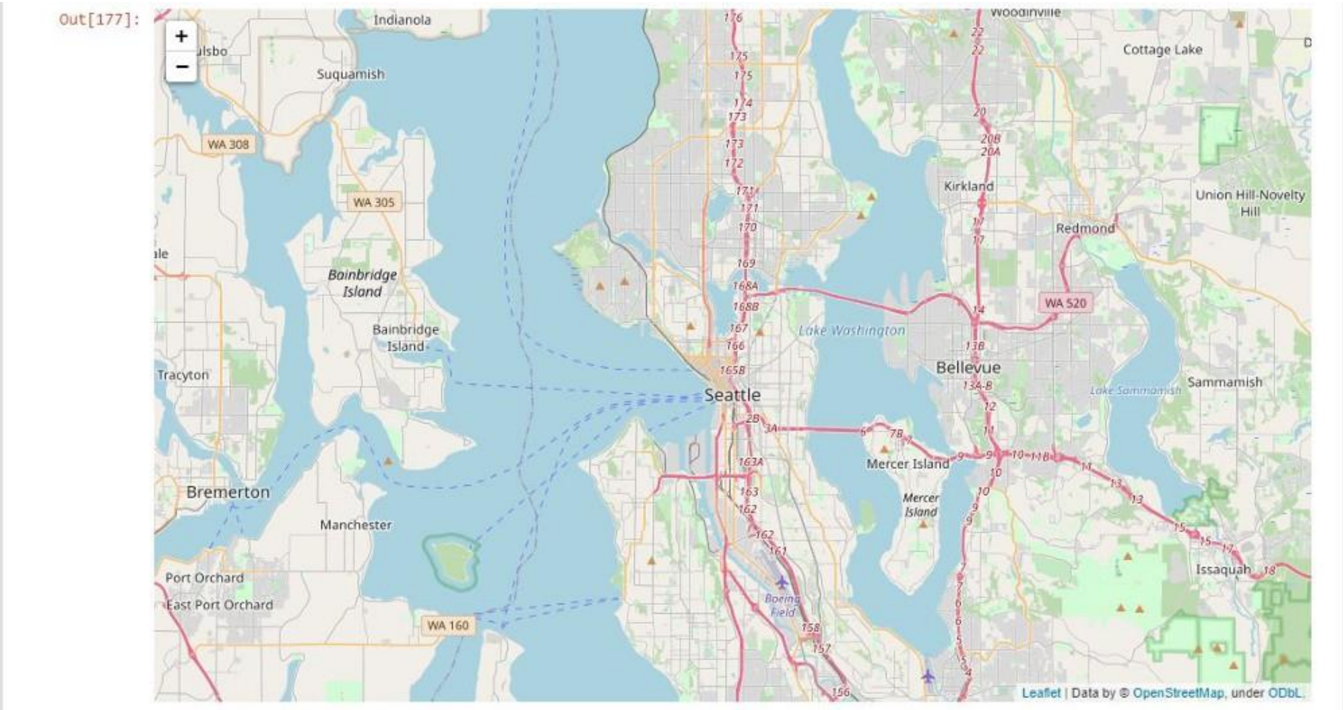
5. Other issues: after performing the above data engineering issues, we will create a copy of the data frame (minus any columns which will not be used for model-building) and perform a final check for any remaining NaNs/missing data. We will decide how to handle these later.

Okay, let's get started!

Data Exploration

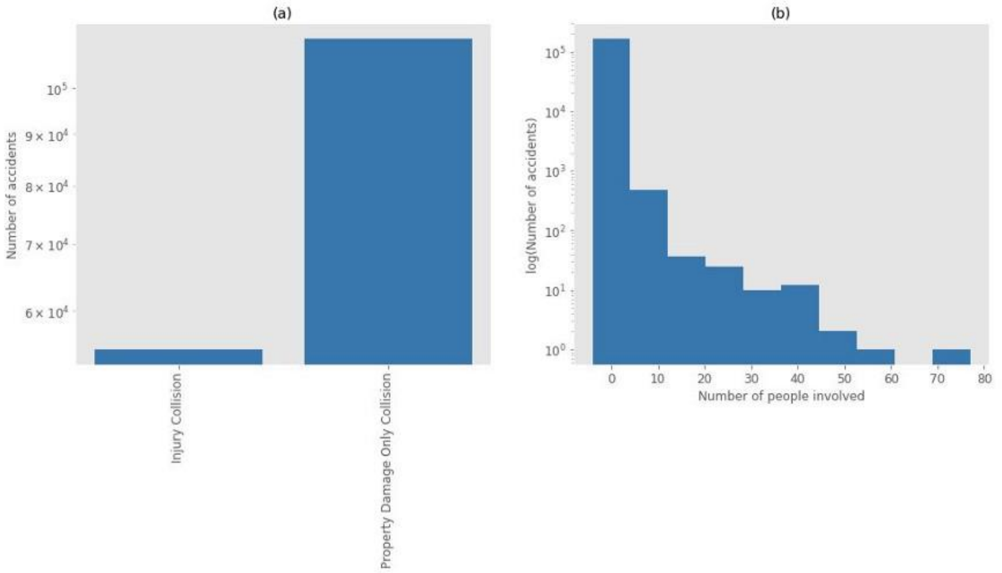
1: Where do accidents occur?

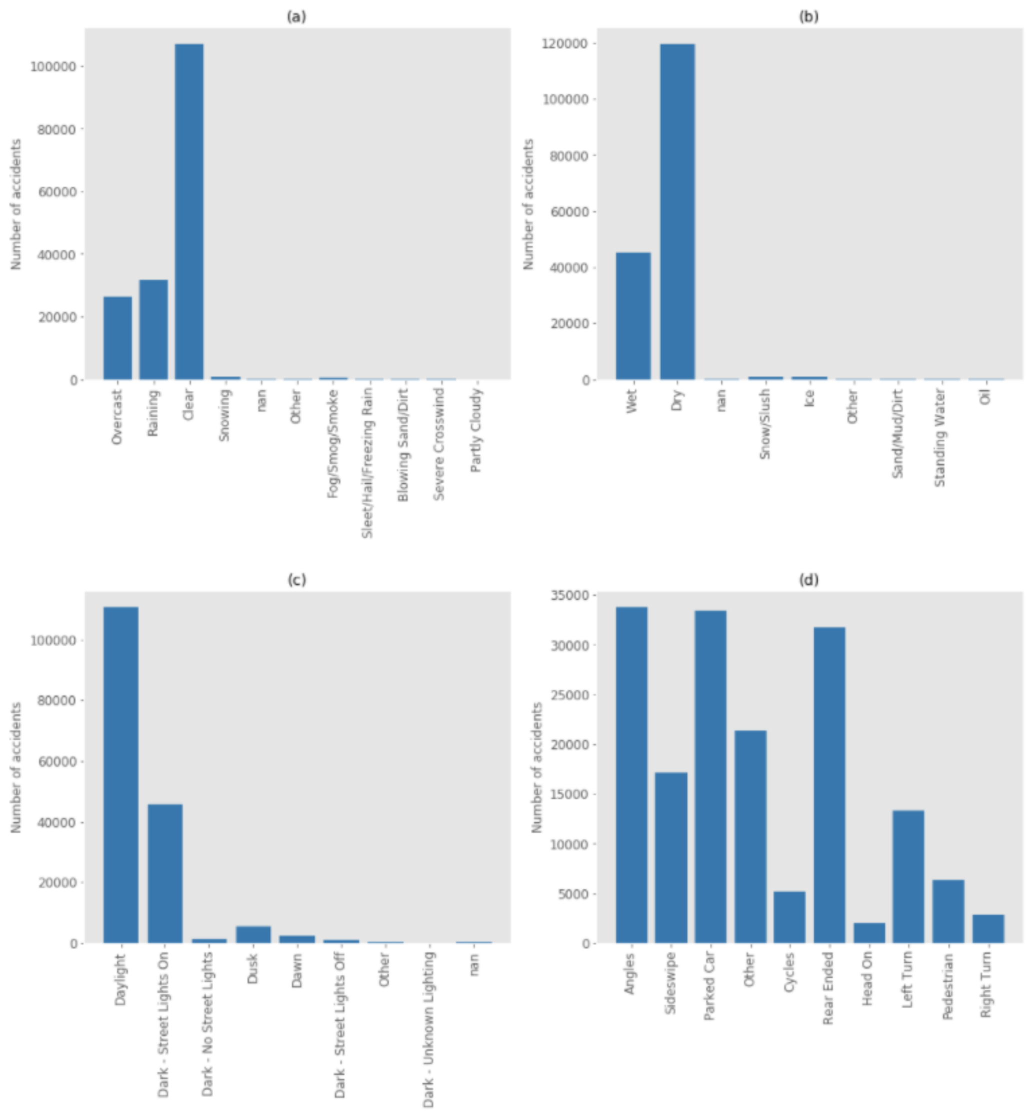
The Seattle Open Data portal record of traffic accidents include the latitude and longitude (as X, Y, respectively) of every accident that occurred in the city council area. Let's create a map in Folium to see where these occur. This might highlight some key "choke points" in the city road network and give context for some of the predictions that eventually come out of the model.

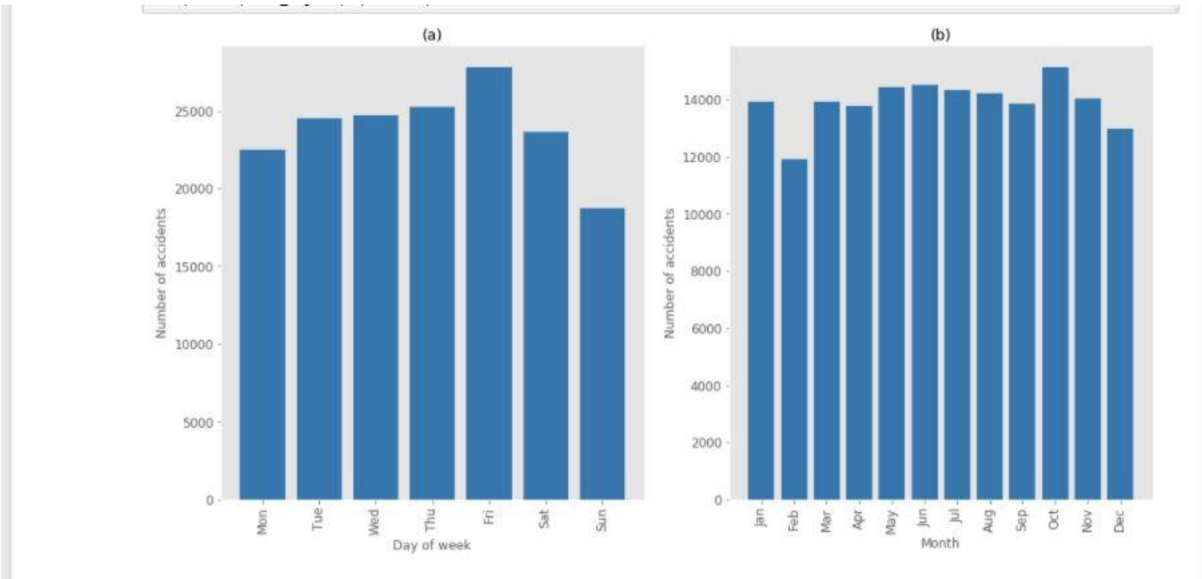


2 : Histograms & Bar Graphs

```
There are 167054 entries in df currently.
```







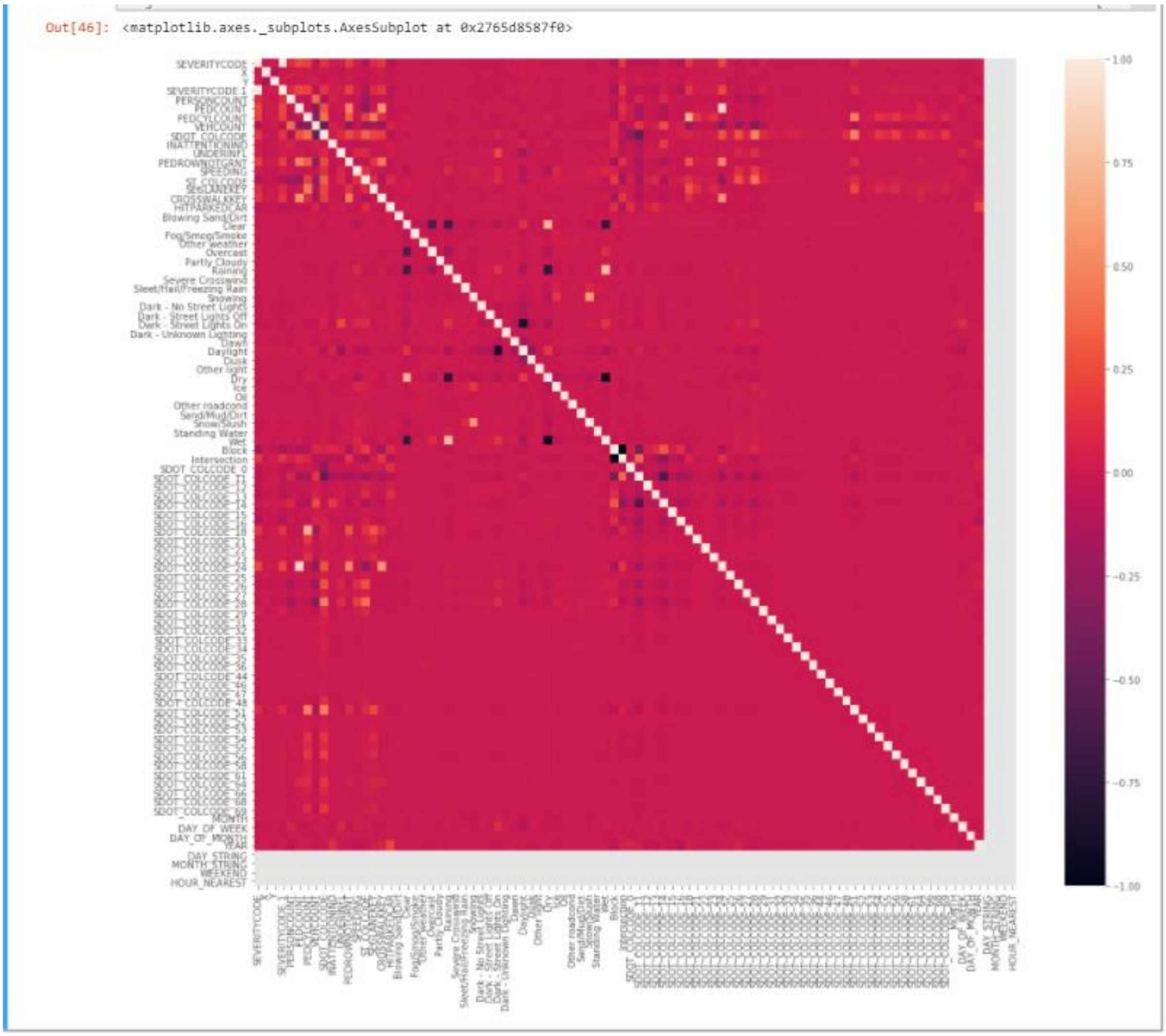
3. Pre-Processing: Feature extraction

We now enter the final phase of data preparation -- deciding which of the remaining columns from the dataframe to include in our model building. Let's begin by looking at a correlation matrix for the dataset:

In [46]:

```
if "Alley" in data:
    del data["Alley"]

plt.rcParams["figure.figsize"] = (18,16)
corr = data.corr()
plt.rc('xtick',labelsize=10)
plt.rc('ytick',labelsize=10)
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns)
```



Little is hugely surprising in here: there seem to be only weak/marginal correlations between *SEVERITYCODE* and weather/light conditions, however there are is also an interesting correlation between *PEDROWNOTGRT* and "Dark **-** Street Lights On". Presumably this is because in low light conditions drivers often don't see pedestrians?

It looks as if there is a correlation between *SEVERITYCODE* and *ST_COLCODE* , which makes sense as we would expect different types of collision to have more serious consequences, e.g. a head-on collision on a motorway is likely to have more deaths/serious injuries than a low-speed read-end collision.

There are clear correlations between *SEVERITYCODE* and *INJURIES* , *SERIOUSINJURIES* and *FATALITIES* but these should surprise no-one as a combination of the latter three features **defines** *SEVERITYCODE* . The purpose of the model is to predict the *SEVERITYCODE* of an accident based on the environmental/geographical conditions. There is little point building a model that tells us "count the injuries, and if there are a lot then the accident is serious!". We will have to consider excluding these features from the dataset.

Deciding whether to keep or remove other features is a bit more subjective, however it is helpful to think of this from the point of view of an emergency services operator: an accident has occurred, and you wish to be able to make a quick prediction as to its severity. What key features might you expect to be able to know about *before* the first responders arrive at the scene in order to be able to influence who you send? Clearly you cannot know the number of fatalities in advance, but might the person reporting the accident be able to provide some useful information to help you predict the severity of the accident? It seems logical that the time, day, date, weather and road conditions would be known at the time of the accident. It is also likely that the person reporting the accident would be able to tell you if there were pedestrians/cyclists involved, and how many cars. They may also be able to tell you if alcohol is obviously a factor, if the accident took place on a pedestrian crossing/crosswalk, whether the accident involved a parked car and to provide a brief description of the nature of the collision (which is encoded in *SDOT_COLCODE*). We should therefore keep these features.

As a reminder, the target variable is *SEVERITYCODE* , so clearly this cannot be part of the feature set.

Balancing the dataset

As is clear from the histogram of severity code shown above (and repeated below, using a linearly-scaled y axis), the vast majority of accidents involve either no injuries, or minor injuries only. Only a small number of accidents involve serious injuries or fatalities. If we train a classification model on these data, the model will be biased. To fix this issue we need to resample the data.

I propose to rebalance the data by:

1. Down-sampling *SEVERITYCODE* 1, 2 and 3 (no, minor and major injuries, respectively) to match the number of samples as is in *SEVERITYCODE* 4 (fatalities)
2. Converting *SEVERITYCODE* in to a binary variable with 0 for no/minor injuries and 1 for major injuries/fatalities.

In addition to balancing the dataset between the four values that can be taken by the target variable (*SEVERITYCODE*) this will also drastically reduce overall size of the dataset, making it much more feasible for us to build and test our models.

In [40]:
```
#Metadata for the charts that follow:
#1 - Weather conditions
print('Frequency of weather types:')
print(data_df["WEATHER"].value_counts())
print(len(data_df["WEATHER"]))
```

```
Frequency of weather types:
Clear                       106995
Raining                      31640
Overcast                     26453
Snowing                        825
Fog/Smog/Smoke                 539
Other                          244
Sleet/Hail/Freezing Rain       110
Blowing Sand/Dirt               44
Severe Crosswind                24
Partly Cloudy                    5
Name: WEATHER, dtype: int64
167054
```

In [41]:
```
#2 - Road conditions
print('Frequency of rifferent road conditions:')
print(data_df["ROADCOND"].value_counts())
```

```
Frequency of rifferent road conditions:
Dry               119599
Wet                45137
Ice                 1073
Snow/Slush           838
Other                 98
Standing Water        95
Sand/Mud/Dirt         58
Oil                   50
Name: ROADCOND, dtype: int64
```

```
Name: ROADCOND, dtype: int64
```

```
In [42]:
#3 - Light conditions
print('Frequency of different light conditions:')
print(data_df["LIGHTCOND"].value_counts())
```

```
Frequency of different light conditions:
Daylight                    110564
Dark - Street Lights On      45770
Dusk                          5556
Dawn                          2346
Dark - No Street Lights       1338
Dark - Street Lights Off      1080
Other                          160
Dark - Unknown Lighting          9
Name: LIGHTCOND, dtype: int64
```

Methodology

Our data is now ready to be fed into machine learning models.

We will use the following models:

K-Nearest Neighbor (KNN)

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

K-Nearest Neighbors (KNN) ¶

```
In [152]:
# Building the KNN Model
from sklearn.neighbors import KNeighborsClassifier

k = 25
```

```
In [153]:
#Train Model & Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,Y_train)
neigh

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

```
Out[153]: array([1, 1, 1, 1, 1], dtype=int64)
```

Decision Tree

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

Decision Tree

```
In [171]:
# Building the Decision Tree
from sklearn.tree import DecisionTreeClassifier
colDataTree = DecisionTreeClassifier(criterion="entropy", max_depth = 9)
colDataTree
colDataTree.fit(X_train,Y_train)
```

```
Out[171]: DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=9, 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='deprecated',
                       random_state=None, splitter='best')
```

```
In [172]:
# Train Model & Predict
DTyhat = colDataTree.predict(X_test)
```

Logistic Regression

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

Logistic Regression

```
In [159]:
# Building the LR Model
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[159]: LogisticRegression(C=6, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
In [160]:
# Train Model & Predicr
LRyhat = LR.predict(X_test)
LRyhat
```

```
Out[160]: array([2, 1, 2, ..., 1, 1, 1], dtype=int64)
```

```
In [161]:
yhat_prob = LR.predict_proba(X_test)
yhat_prob
```

```
Out[161]: array([[0.4937124 , 0.5062876 ],
       [0.51957358, 0.48042642],
       [0.4937124 , 0.5062876 ],
       ...,
       [0.53497271, 0.46502729],
       [0.53497271, 0.46502729],
       [0.53497271, 0.46502729]])
```

Results & Evaluation

Now we will check the accuracy of our models.

In [162]:
```
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
```

K-Nearest Neighbor

In [163]:
```
# Jaccard Similarity Score
jaccard_similarity_score(Y_test, Kyhat)
```

```
C:\Users\JAY\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:664: FutureWarning: jaccard_similarity_score has be
en deprecated and replaced with jaccard_score. It will be removed in version 0.23. This implementation has surprising behavior
for binary and multiclass classification tasks.
  FutureWarning)
```

Out[163]: 0.49124988626368626

In [164]:
```
# F1-SCORE
f1_score(Y_test, Kyhat, average='macro')
```

Out[164]: 0.47619870634644745

Model is most accurate when k is 25.

In [173]:
```
# Jaccard Similarity Score
jaccard_similarity_score(Y_test, DTyhat)
```

```
C:\Users\JAY\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:664: FutureWarning: jaccard_similarity_score has be
en deprecated and replaced with jaccard_score. It will be removed in version 0.23. This implementation has surprising behavior
for binary and multiclass classification tasks.
  FutureWarning)
```

Out[173]: 0.5183039640896545

In [174]:
```
# F1-SCORE
f1_score(Y_test, DTyhat, average='macro')
```

Out[174]: 0.48114954074540683

In [149]:
```
# Jaccard Similarity Score
jaccard_similarity_score(Y_test, LRyhat)
```

Out[149]: 0.5184556125079616

In [150]:
```
# F1-SCORE
f1_score(Y_test, LRyhat, average='macro')
```

Out[150]: 0.4958071766854718

In [151]:
```
# LOGLOSS
yhat_prob = LR.predict_proba(X_test)
log_loss(Y_test, yhat_prob)
```

Out[151]: 0.6921950516353936

Discussion

In the beginning of this notebook, we had categorical data that was of type 'object'. This is not a data type that we could have fed through an algoritim, so label encoding was used to created new classes that were of type int8; a numerical data type.

After solving that issue we were presented with another—imbalanced data. As mentioned earlier, class 1 was nearly three times larger than class 2. The solution to this was downsampling the majority class with sklearn's resample tool. We downsampled to match the minority class exactly with 58188 values each.

Once we analyzed and cleaned the data, it was then fed through three ML models; K-Nearest Neighbor, 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 hyparameter C values helped to improve our accuracy to be the best possible.

Conclusion

Based on historical data from weather conditions pointing to certain classes, we can conclude that particular weather conditions have a somewhat impact on whether or not travel could result in property damage (class 1) or injury (class 2).

Thank you for reading!