REPORT:ROAD ACCIDENT SEVERITY PREDICTOR

This Report is a part of the peer graded assignment of the final course:Applied Data Science Capstone of the IBM Data Science Professional Certificate Course.

We will be following the CRISP-DM(Cross-Industry Standard Process for Data Mining) Approach to solve the problem and build a predictor model.

BUSINESS UNDERSTANDING:

Oftentimes, while travelling from one place to another we encounter accidents on the road, sometimes severe, sometimes fatal, sometimes not so severe. What if we knew in advance the severity of accidents beforehand and avoid travelling when the probability of accidents is more.

This project is useful for anyone and everyone: If you're travelling from 1 city to another, your daily commute to your workplace and back home, including your everyday and other travels. Knowing in advance the severity of an accident will help you save you and your time by avoiding taking that route. Moreover, it is useful for the government as well to check what conditions lead to more severe accidents and how to reduce it.

A better understanding of the problem will be established in subsequent sections.

DATA UNDERSTANDING:

There are 37 attributes in the dataframe that we're using for the model development and not all of that information is required to build the model. So we drop the unnecessary columns before working on the model.

```
In [3]: df=pd.read_csv("https://s3.us.cloud-object-storage.appdomain.cloud/cf-courses-data/CognitiveClass/DP0701EN/version-2/Data-Collisions.csv")
        df.head()

/opt/conda/envs/Python36/lib/python3.6/site-packages/IPython/core/interactiveshell.py:3020: DtypeWarning: Columns (33) have mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Out[3]:

	SEVERITYCODE	X	Y	OBJECTID	INCKEY	COLDETKEY	REPORTNO	STATUS	ADDRTYPE	INTKEY	...	ROADCOND	LIGHTCOND	PEDROWNOTGRNT	SDOTCOLNU
0	2	-122.323148	47.703140	1	1307	1307	3502005	Matched	Intersection	37475.0	...	Wet	Daylight	NaN	Na
1	1	-122.347294	47.647172	2	52200	52200	2607959	Matched	Block	NaN	...	Wet	Dark - Street Lights On	NaN	6354039
2	1	-122.334540	47.607871	3	26700	26700	1482393	Matched	Block	NaN	...	Dry	Daylight	NaN	4323031
3	1	-122.334803	47.604803	4	1144	1144	3503937	Matched	Block	NaN	...	Dry	Daylight	NaN	Na
4	2	-122.306426	47.545739	5	17700	17700	1807429	Matched	Intersection	34387.0	...	Wet	Daylight	NaN	4028032

5 rows × 38 columns

To build a better understanding of data we use the dtypes method on the dataframe to know the datatypes of different columns in the table:

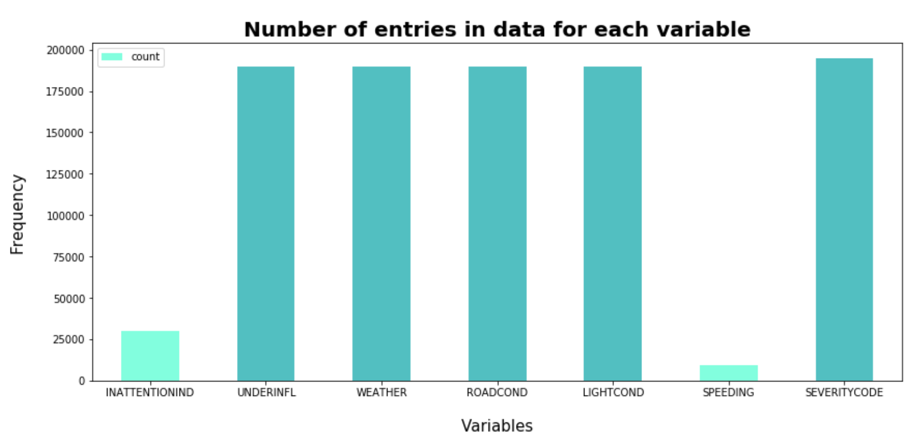
```
In [8]: df.dtypes

Out[8]: 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
```

In this model our <u>target variable (X) is SEVERITYCODE</u> and the potential Independent variables can be <u>ROADCOND,WEATHER,LIGHTCOND,SPEEDING,UNDERINFL,INATTENTIONIND.</u>

But, we see that most of these variables are of type object and difficult to be deployed in the model. So, we modified the values of these variables to int type. However, even when the SEVERITYCODE is an int type data type, we see that the values it stores are 1(for Property Damage) and 2(Injury Collision). So,we would like to change these values of 1 and 2 to 0 and 1 for a better model.

On modifying it, we use the describe() function on the modified datatype, plot a graph of the number of entries in each attribute and notice that some of our attributes have quite a less number of entries stored in them. We also can't drop all these fields, since the data may lose it's meaning. As a result, we not only need to change the datatype of these attributes but also fill the empty fields to make the data more reliable for building the model.



Moving on, we assign integers to each unique attribute in an attribute. So, the key that we have used to replace the values of different attributes is pretty simple. For variables storing binary information in the form of yes/no, we used 1 for Yes and 0 for No. These attributes include UNDERINFL,SPEEDING and INATTENTIONIND. For LIGHTCOND, we distributed the data in 3 types:Light,Medium and Dark. We've used 0 for Light,1 for Medium and 2 for Dark. Coming to ROADCOND the basis of indexing is 0 for Dry, 1 for Mushy and 2 for Wet. As for WEATHER, again we classified the data into 3 categories: 0 for clear or overcast,1 for Windy, 2 for Rain and 3 for Snow.

For attributes with null values, those were assigned the value 0. And for the ones storing values like other or unknown, we couldn't happen to delete the rows because it would've adversely affected our model. So we used another unique value for them in the attributes that fell in this category.

Our data is now ready to be used. This is what it looks like:

In [15]:
```
feature_df.head()
```

Out[15]:

	X	Y	INCKEY	INATTENTIONIND	UNDERINFL	SPEEDING	LIGHTCOND	WEATHER	ROADCOND	SEVERITYCODE
0	-122.323148	47.703140	1307	0	0	0	0	1	2	1
1	-122.347294	47.647172	52200	0	0	0	1	3	2	0
2	-122.334540	47.607871	26700	0	0	0	0	1	0	0
3	-122.334803	47.604803	1144	0	0	0	0	0	0	0
4	-122.306426	47.545739	17700	0	0	0	0	3	2	1