2 Data Section

2.1 Data

In this section we describe the data that will be used to solve the problem and the source of the data.

2.1.1 Data Analysis

Basic Insight of Dataset
After reading data into Pandas dataframe, we explore the dataset with:

- df.head(10)
- df.tail(10)
- df.shape : (194673, 38) , which is number of data sets by number of columns.

2.1.2 Data Types

In order to better learn about each attribute, it is always good for us to know the data type of each column.
df.dtypes :

```
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

2.1.3 Describe the data

If we would like to get a statistical summary of each column, such as count, column mean value, column standard deviation, etc. we use the describe method.
This method will provide various summary statistics, including NaN (Not a Number) values.
df.describe(include ="all") :

SEVERITYCODE	X	Y	OBJECTID	INCKEY	COLDETKEY	REPORTNO	STATUS	ADDRTYPE	INTKEY	...	ROADCOND	LIGHTCOND	PEDROWNOTGRNT	SDOTCOLNUM	SPEEDING	ST_COLCODE	ST_COLDESC	SEGLANEKEY	CROSSWALKKEY	HITPARKEDCAR	

count	194673.000000	189339.000000	189339.000000	194673.000000	194673.000000	194673.000000	194673	194673	192747	65070.000000	...	189661	189503	4667	1.149360e+05	9333	194655	189769	194673.000000	1.946730e+05	194673
unique	NaN	NaN	NaN	NaN	NaN	NaN	194670	2	3	NaN	...	9	9	1	NaN	1	115	62	NaN	NaN	2
top	NaN	NaN	NaN	NaN	NaN	NaN	1782439	Matched	Block	NaN	...	Dry	Daylight	Y	NaN	Y	32	One parked--one moving	NaN	NaN	N
freq	NaN	NaN	NaN	NaN	NaN	NaN	2	189786	126926	NaN	...	124510	116137	4667	NaN	9333	27612	44421	NaN	NaN	187457

mean	1.298901	-122.330518	47.619543	108479.364930	141091.456350	141298.811381	NaN	NaN	NaN	37558.450576	...	NaN	NaN	NaN	7.972521e+06	NaN	NaN	NaN	269.401114	9.782452e+03	NaN
std	0.457778	0.029976	0.056157	62649.722558	86634.402737	86986.542110	NaN	NaN	NaN	51745.990273	...	NaN	NaN	NaN	2.553533e+06	NaN	NaN	NaN	3315.776055	7.226926e+04	NaN
min	1.000000	-122.419091	47.495573	1.000000	1001.000000	1001.000000	NaN	NaN	NaN	23807.000000	...	NaN	NaN	NaN	1.007024e+06	NaN	NaN	NaN	0.000000	0.000000e+00	NaN

25%	1.000000	-122.348673	47.575956	54267.000000	70383.000000	70383.000000	NaN	NaN	NaN	28667.000000	...	NaN	NaN	NaN	6.040015e+06	NaN	NaN	NaN	0.000000	0.000000e+00	NaN
50%	1.000000	-122.330224	47.615369	106912.000000	123363.000000	123363.000000	NaN	NaN	NaN	29973.000000	...	NaN	NaN	NaN	8.023022e+06	NaN	NaN	NaN	0.000000	0.000000e+00	NaN
75%	2.000000	-122.311937	47.663664	162272.000000	203319.000000	203459.000000	NaN	NaN	NaN	33973.000000	...	NaN	NaN	NaN	1.015501e+07	NaN	NaN	NaN	0.000000	0.000000e+00	NaN

max	2.000000	-122.238949	47.734142	219547.000000	331454.000000	332954.000000	NaN	NaN	NaN	757580.000000	...	NaN	NaN	NaN	1.307202e+07	NaN	NaN	NaN	525241.000000	5.239700e+06	NaN

11 rows × 38 columns

2.1.4 Dataset Info

Another method you can use to check your dataset is dataframe.info
It provides a concise summary of your DataFrame.Look at the info of "df".
Here we are able to see the information of our dataframe, with the top 30 rows and the bottom 30 rows.
And, it also shows us the whole data frame has 194673 rows and 38 columns in total.

df.info :

```
<bound method DataFrame.info of        SEVERITYCODE          X          Y
OBJECTID  INCKEY  COLDETKEY  \
0                 2 -122.323148  47.703140         1    1307       1307
1                 1 -122.347294  47.647172         2   52200      52200
2                 1 -122.334540  47.607871         3   26700      26700
3                 1 -122.334803  47.604803         4    1144       1144
4                 2 -122.306426  47.545739         5   17700      17700
...             ...         ...        ...       ...     ...        ...
194668            2 -122.290826  47.565408    219543  309534     310814
194669            1 -122.344526  47.690924    219544  309085     310365
194670            2 -122.306689  47.683047    219545  311280     312640
194671            2 -122.355317  47.678734    219546  309514     310794
194672            1 -122.289360  47.611017    219547  308220     309500

       REPORTNO   STATUS      ADDRTYPE   INTKEY  ... ROADCOND  \
0       3502005  Matched  Intersection  37475.0  ...      Wet
1       2607959  Matched         Block      NaN  ...      Wet
2       1482393  Matched         Block      NaN  ...      Dry
3       3503937  Matched         Block      NaN  ...      Dry
```

```
4       1807429  Matched  Intersection  34387.0  ...      Wet
...         ...      ...           ...      ...  ...      ...
194668  E871089  Matched         Block      NaN  ...      Dry
194669  E876731  Matched         Block      NaN  ...      Wet
194670  3809984  Matched  Intersection  24760.0  ...      Dry
194671  3810083  Matched  Intersection  24349.0  ...      Dry
194672  E868008  Matched         Block      NaN  ...      Wet

                       LIGHTCOND PEDROWNOTGRNT  SDOTCOLNUM SPEEDING
ST_COLCODE  \
0                       Daylight          NaN         NaN      NaN         10
1       Dark - Street Lights On           NaN   6354039.0      NaN         11
2                       Daylight          NaN   4323031.0      NaN         32
3                       Daylight          NaN         NaN      NaN         23
4                       Daylight          NaN   4028032.0      NaN         10
...                          ...          ...         ...      ...        ...
194668                  Daylight          NaN         NaN      NaN         24
194669                  Daylight          NaN         NaN      NaN         13
194670                  Daylight          NaN         NaN      NaN         28
194671                      Dusk          NaN         NaN      NaN          5
194672                  Daylight          NaN         NaN      NaN         14

                                               ST_COLDESC  SEGLANEKEY  \
0                                       Entering at angle           0
1       From same direction - both going straight - bo...           0
2                                  One parked--one moving           0
3                       From same direction - all others           0
4                                       Entering at angle           0
...                                                   ...         ...
194668    From opposite direction - both moving - head-on           0
194669  From same direction - both going straight - bo...           0
194670  From opposite direction - one left turn - one ...           0
194671                       Vehicle Strikes Pedalcyclist        4308
194672  From same direction - both going straight - on...           0

        CROSSWALKKEY  HITPARKEDCAR
0                  0             N
1                  0             N
2                  0             N
3                  0             N
4                  0             N
...              ...           ...
194668             0             N
194669             0             N
194670             0             N
194671             0             N
```

194672 0 N

[194673 rows x 38 columns]>

In [135]:

2.2 Data Wrangling

2.2.1 Convert "?" to NaN if any

In the dataset, missing data comes sometimes with the question mark "?".
We replace "?" with NaN (Not a Number), which is Python's default missing value marker,
for reasons of computational speed and convenience. Here we would use the function:
replace "?" to NaN

df.replace("?", np.nan, inplace = True)

2.2.2 Missing Data
Steps for working with missing data:
1. identify missing data
2. deal with missing data
3. correct data format

2.2.2.1 Identify_missing_values
valuating for Missing Data
The missing values are converted to Python's default. We use Python's built-in functions to identify these missing values. There are two methods to detect missing data:
a. .isnull()
b. .notnull()
The output is a boolean value indicating whether the value that is passed into the argument is in fact missing data.
"True" stands for missing value, while "False" stands for not missing value.

missing_data = df.isnull()
missing_data.head(5)

Count missing values in each column
Using a for loop in Python, we can quickly figure out the number of missing values in each column.
As mentioned above, "True" represents a missing value, "False" means the

value is present in the dataset.
In the body of the for loop the method ".value_counts()" counts the number of "True" values.

```
for column in missing_data.columns.values.tolist():
    print(column)
    print (missing_data[column].value_counts())
    print("")
```

Based on the summary above, each column has 205 rows of data, seven columns containing missing data:
1. "X": 5534 missing data
2. "Y": 5534 missing data
3. "ADDRTYPE": 1926 missing data
4. "INTKEY" : 129603 missing data
5. "LOCATION": 2677 missing data
6. "EXCEPTRSNCODE": 109862 missing data
7. "EXCEPTRSNDESC": 189035 missing data
8. "COLLISIONTYPE": 4904 missing data
9. "JUNCTIONTYPE": 6329 missing data
10. "INATTENTIONIND": 164868 missing data
11. "UNDERINFL": 4884 missing data
12. "WEATHER": 5081 missing data
13. "ROADCOND": 5012 missing data
14. "LIGHTCOND": 5170 missing data
15. "PEDROWNOTGRNT": 190006 missing data
16. "SDOTCOLNUM": 79737 missing data
17. "SPEEDING": 185340 missing data
18. "ST_COLCODE": 18 missing data
19. "ST_COLDESC": 4904 missing data

```
df.dropna(axis=0, inplace=False)
df.head()
```

Select the columns we want to use

```
df_all =
df[['SEVERITYCODE','STATUS','ADDRTYPE','INTKEY','LOCATION','COLLISIONTYPE','SEVERITYCODE.1','PERSONCOUNT','PEDCOUNT','PEDCYLCOUNT','VEHCOUNT', 'INCDATE', 'INCDTTM', 'JUNCTIONTYPE', 'INATTENTIONIND', 'UNDERINFL', 'WEATHER', 'ROADCOND', 'LIGHTCOND', 'PEDROWNOTGRNT', 'SPEEDING', 'ST_COLCODE', 'SEGLANEKEY', 'CROSSWALKKEY', 'HITPARKEDCAR']]
```

For KNN: select columns without content as: plain text, koordinates, ids, keys:

cdf =
df[['SEVERITYCODE','PERSONCOUNT','PEDCOUNT','PEDCYLCOUNT','VEHCOUNT','WEATHER', 'ROADCOND', 'LIGHTCOND',]]
cdf.head(9)

For decision tree: select the accident counts and weather columns:
ddf =
df[['SEVERITYCODE','PERSONCOUNT','PEDCOUNT','PEDCYLCOUNT','VEHCOUNT','WEATHER', 'ROADCOND', 'LIGHTCOND',]]
ddf.head(9)

2.2.2.2 Deal with missing data

How to deal with missing data

1. Drop data

1. a) Drop the whole row
Whole columns should be dropped only if most entries in the column are empty. In our dataset, none of the columns are empty enough to drop entirely.
We have some freedom in choosing which method to replace data; however, some methods may seem more reasonable than others. We will apply each method to many different columns.
*   8."COLLISIONTYPE": 4904 missing data, simply delete the whole row.
    * Reason: COLLISIONTYPE is what we need to predict accident fatality. Any data entry without COLLISIONTYPE data cannot be used for prediction; therefore any row now without COLLISIONTYPE data is not useful to us.
*   3."ADDRTYPE": 1926 missing data. See reason as above.

* 12. "WEATHER": 5081 missing data, simply delete the whole row. See reason as above.
* 13. "ROADCOND": 5012 missing data, simply delete the whole row. See reason as above.
* 14. "LIGHTCOND": 5170 missing data, simply delete the whole row. See reason as above.
* 18. "ST_COLCODE": 18 missing data, simply delete the whole row.

Simply drop whole row with NaN in "COLLISIONTYPE" column
cdf.dropna(subset=["COLLISIONTYPE"], axis=0, inplace=True)

Simply drop whole row with NaN in "ADDRTYPE" column
#df.dropna(subset=["ADDRTYPE"], axis=0, inplace=True)

Simply drop whole row with NaN in "WEATHER" column
cdf.dropna(subset=["WEATHER"], axis=0, inplace=True)

ddf.dropna(subset=["WEATHER"], axis=0, inplace=True)

Simply drop whole row with NaN in "ROADCOND" column
cdf.dropna(subset=["ROADCOND"], axis=0, inplace=True)
ddf.dropna(subset=["ROADCOND"], axis=0, inplace=True)

Simply drop whole row with NaN in "LIGHTCOND" column
cdf.dropna(subset=["LIGHTCOND"], axis=0, inplace=True)
ddf.dropna(subset=["LIGHTCOND"], axis=0, inplace=True)

Reset index, because we droped some rows
cdf.reset_index(drop=True, inplace=True)
ddf.reset_index(drop=True, inplace=True)

1. b) Drop the whole column

* 4. "INTKEY" : 129603 missing data, drop the whole column.
    * Reason: The key that corresponds to the intersection associated with a collision is not used herewith as it is redundant to coordinates.
* 5. "LOCATION": 2677 missing data, drop the whole column. See reason as above.
* 6. "EXCEPTRSNCODE": 109862 missing data, drop the whole column. Code not used herewith.
* 7. "EXCEPTRSNDESC": 189035 missing data, drop the whole column. Description not used herewith.
* 9. "JUNCTIONTYPE": 6329 missing data, drop the whole column. Type of Junction not used herewith.
* 10. "INATTENTIONIND": 164868 missing data, drop the whole column. Inattention not used herewith.
* 11. "UNDERINFL": 4884 missing data, drop the whole column. Drugs and alcohol usage is not used herewith.
* 15. "PEDROWNOTGRNT": 190006 missing data, drop the whole column. Pedestrian right not used herewith.
* 16. "SDOTCOLNUM": 79737 missing data, drop the whole column. Sub cathegories of ST_COLCODE not used herewith.
* 17. "SPEEDING": 185340 missing data, drop the whole column. Speeding was a factor not used herewith.
* 19. "ST_COLDESC": 4904 missing data, drop the whole column. Description not used as this is plain text.

2. Replace data:

2. a) Replace it by mean
* 1. "X": 5334 missing data, replace them with mean.

* Reason: as this is a coordinate of the date set in the Seattle area assumed, mean value might be sufficient
* 2. "Y": 5334 missing data, replace them with mean. See reason as above.

For KNN:

Calculate the mean value for 'PERSONCOUNT' column¶
avg_PERSONCOUNT=cdf['PERSONCOUNT'].astype('float').mean(axis=0)
print("Average of PERSONCOUNT:", avg_PERSONCOUNT)
Replace NaN by mean value
cdf["PERSONCOUNT"].replace(np.nan, avg_PERSONCOUNT, inplace=True)

Calculate the mean value for 'PEDCOUNT' column¶
avg_PEDCOUNT=cdf['PEDCOUNT'].astype('float').mean(axis=0)
print("Average of PEDCOUNT:", avg_PEDCOUNT)
Replace NaN by mean value
cdf["PEDCOUNT"].replace(np.nan, avg_PEDCOUNT, inplace=True)

Calculate the mean value for 'PEDCYLCOUNT' column¶
avg_PEDCYLCOUNT=cdf['PEDCYLCOUNT'].astype('float').mean(axis=0)
print("Average of PEDCYLCOUNT:", avg_PEDCYLCOUNT)
Replace NaN by mean value
cdf["PEDCYLCOUNT"].replace(np.nan, avg_PEDCYLCOUNT, inplace=True)

Calculate the mean value for 'VEHCOUNT' column¶
avg_VEHCOUNT=cdf['VEHCOUNT'].astype('float').mean(axis=0)
print("Average of VEHCOUNT:", avg_VEHCOUNT)
Replace NaN by mean value
cdf["VEHCOUNT"].replace(np.nan, avg_VEHCOUNT, inplace=True)

For decision tree:

Calculate the mean value for 'PERSONCOUNT' column¶
avg_PERSONCOUNT=ddf['PERSONCOUNT'].astype('float').mean(axis=0)
print("Average of PERSONCOUNT:", avg_PERSONCOUNT)
Replace NaN by mean value
ddf["PERSONCOUNT"].replace(np.nan, avg_PERSONCOUNT, inplace=True)

Calculate the mean value for 'PEDCOUNT' column¶
avg_PEDCOUNT=ddf['PEDCOUNT'].astype('float').mean(axis=0)
print("Average of PEDCOUNT:", avg_PEDCOUNT)
Replace NaN by mean value
ddf["PEDCOUNT"].replace(np.nan, avg_PEDCOUNT, inplace=True)

Calculate the mean value for 'PEDCYLCOUNT' column¶
avg_PEDCYLCOUNT=ddf['PEDCYLCOUNT'].astype('float').mean(axis=0)
print("Average of PEDCYLCOUNT:", avg_PEDCYLCOUNT)
Replace NaN by mean value
ddf["PEDCYLCOUNT"].replace(np.nan, avg_PEDCYLCOUNT, inplace=True)

Calculate the mean value for 'VEHCOUNT' column¶
avg_VEHCOUNT=ddf['VEHCOUNT'].astype('float').mean(axis=0)
print("Average of VEHCOUNT:", avg_VEHCOUNT)
Replace NaN by mean value
ddf["VEHCOUNT"].replace(np.nan, avg_VEHCOUNT, inplace=True)

2. b) Replace it by frequency c) replace it based on other functions
not applicable

Calculated averages:

Average of PERSONCOUNT: 2.459566804164004
Average of PEDCOUNT: 0.03810665638517564
Average of PEDCYLCOUNT: 0.0291279570290012
Average of VEHCOUNT: 1.970412544827477

2.2.2.3 Correct Data Format

Data types corrections:

cdf.dtypes,
ddf.dtypes

```
SEVERITYCODE     int64
PERSONCOUNT      int64
PEDCOUNT         int64
PEDCYLCOUNT      int64
VEHCOUNT         int64
WEATHER         object
ROADCOND        object
LIGHTCOND       object
dtype: object
```

As we can see above, some columns are not of the correct data type.
Numerical variables should have type 'float' or 'int',
and variables with strings such as categories should have type 'object'.

For decision tree SEVERITYCODE has to be of String data type.
ddf[["SEVERITYCODE"]] = ddf[["SEVERITYCODE"]].astype("object")

Value Counts for cathgories

Specific values for categories must be converted to Numbers as the machine learning algorithm
Can not handle text categories.

```
cdf['WEATHER'].value_counts() ,
ddf['WEATHER'].value_counts()
```

```
Clear                       111008
Raining                      33117
Overcast                     27681
Unknown                      15039
Snowing                        901
Other                          824
Fog/Smog/Smoke                 569
Sleet/Hail/Freezing Rain       113
Blowing Sand/Dirt               55
Severe Crosswind                25
Partly Cloudy                    5
Name: WEATHER, dtype: int64
```

```
cdf['ROADCOND'].value_counts() ,
ddf['ROADCOND'].value_counts()
```

```
Dry               124300
Wet                47417
Unknown            15031
Ice                 1206
Snow/Slush           999
Other                131
Standing Water       115
Sand/Mud/Dirt         74
Oil                   64
Name: ROADCOND, dtype: int64
```

```
cdf['LIGHTCOND'].value_counts() ,
ddf['LIGHTCOND'].value_counts()
```

```
Daylight                  116077
Dark - Street Lights On      48440
Unknown                    13456
Dusk                        5889
Dawn                        2502
Dark - No Street Lights       1535
Dark - Street Lights Off      1192
Other                        235
Dark - Unknown Lighting        11
Name: LIGHTCOND, dtype: int64
```

```
cdf['SEVERITYCODE'].value_counts() ,
ddf['SEVERITYCODE'].value_counts()
```

```
1    132285
2     57052
Name: SEVERITYCODE, dtype: int64
```

2.3 Pre-processing

2.3.1 Pre-processing for KNN:

Get indicator variables and assign it to data frame "dummy_variable_1“:

```
dummy_variable_1 = pd.get_dummies(cdf["WEATHER"])
dummy_variable_2 = pd.get_dummies(cdf["ROADCOND"])
dummy_variable_3 = pd.get_dummies(cdf["LIGHTCOND"])
```

Change column names for clarity:

```
dummy_variable_1.rename(columns={'Unknown':'WeUnknown'}, inplace=True)
dummy_variable_1.rename(columns={'Other':'WeOther'}, inplace=True)
dummy_variable_1.rename(columns={'Fog/Smog/Smoke':'FogSmogSmoke',
'Sleet/Hail/Freezing Rain':'SleetHailFreezingRain'}, inplace=True)
dummy_variable_1.rename(columns={'Blowing Sand/
Dirt':'BlowingSandDirt','Severe Crosswind':'SevereCrosswind'}, inplace=True)
dummy_variable_1.rename(columns={'Partly Cloudy':'PartlyCloudy'},
inplace=True)

dummy_variable_2.rename(columns={'Unknown':'RoUnknown'}, inplace=True)
dummy_variable_2.rename(columns={'Snow/Slush':'SnowSlush',
'Other':'RoOther'}, inplace=True)
dummy_variable_2.rename(columns={'Standing Water':'StandingWater', 'Sand/
Mud/Dirt':'SandMudDirt'}, inplace=True)
```

dummy_variable_3.rename(columns={'Dark - Street Lights On':'DarkStreetLightsOn'}, inplace=True)
dummy_variable_3.rename(columns={'Unknown':'LiUnknown'}, inplace=True)
dummy_variable_3.rename(columns={'Dark - No Street Lights':'DarkNoStreetLights'}, inplace=True)
dummy_variable_3.rename(columns={'Dark - Street Lights Off':'DarkStreetLightsOff', 'Other':'LiOther'}, inplace=True)
dummy_variable_3.rename(columns={'Dark - Unknown Lighting':'DarkUnknownLighting'}, inplace=True)

dummy_variable_1.head()
dummy_variable_2.head()
dummy_variable_3.head()
dummy_variable_4.head()

We now have i.e. the value 0 to represent “Dry" and 1 to represent "Wet" in the column "WEATHER" etc.
We will now insert this column back into our original dataset.

Merge data frame "cdf" and "dummy_variable_1"
cdf = pd.concat([cdf, dummy_variable_1], axis=1)

Drop original column "WEATHER" from "cdf"
cdf.drop("WEATHER", axis = 1, inplace=True)

Merge data frame "cdf" and "dummy_variable_2"
cdf = pd.concat([cdf, dummy_variable_2], axis=1)

Drop original column "ROADCOND" from "cdf"
cdf.drop("ROADCOND", axis = 1, inplace=True)

Merge data frame "cdf" and "dummy_variable_3"
cdf = pd.concat([cdf, dummy_variable_3], axis=1)

Drop original column "LIGHTCOND" from "cdf"
cdf.drop("LIGHTCOND", axis = 1, inplace=True)

Merge data frame "cdf" and "dummy_variable_4"
cdf = pd.concat([cdf, dummy_variable_4], axis=1)

Drop original column "SPEEDING" from "cdf"

cdf.drop("SPEEDING", axis = 1, inplace=True)

cdf.head()

2.3.2 Pre-processing for Decision Tree

Remove columns not needed for decision tree.
dtree_df = ddf[['SEVERITYCODE', 'PERSONCOUNT', 'VEHCOUNT', 'WEATHER', 'ROADCOND', 'LIGHTCOND']]
dtree_df[0:5]

We use dtree_data as the accident data read by pandas, declare the following variables:
- X_dtree as the Feature Matrix (data of dtree_data)
- y_dtree as the response vector (target)

We remove the column containing the target name since it doesn't contain numeric values.

X_dtree = dtree_df[['PERSONCOUNT', 'VEHCOUNT', 'WEATHER', 'ROADCOND', 'LIGHTCOND']].values
X_dtree[0:5]

As we figure out, some features in this dataset are categorical such as WEATHER, ROADCOND or LIGHTCOND.
Unfortunately, Sklearn Decision Trees do not handle categorical variables.
But still we can convert these features to numerical values.
pandas.get_dummies()
Convert categorical variable into dummy/indicator variables.

```
le_weather = preprocessing.LabelEncoder()
le_weather.fit(['Blowing Sand/Dirt', 'Clear', 'Fog/Smog/Smoke', 'Other', 
'Overcast',
       'Partly Cloudy', 'Raining', 'Severe Crosswind',
       'Sleet/Hail/Freezing Rain', 'Snowing', 'Unknown'])
X_dtree[:,2] = le_weather.transform(X_dtree[:,2])

le_roadcond = preprocessing.LabelEncoder()
le_roadcond.fit([ 'Dry', 'Ice', 'Oil',
      'Other', 'Sand/Mud/Dirt', 'Snow/Slush', 'Standing Water', 'Unknown', 'Wet'])
X_dtree[:,3] = le_roadcond.transform(X_dtree[:,3])

le_lightcond = preprocessing.LabelEncoder()
```

```
le_lightcond.fit([ 'Dark - No Street Lights', 'Dark - Street Lights Off',
      'Dark - Street Lights On', 'Dark - Unknown Lighting', 'Dawn',
      'Daylight', 'Dusk', 'Other', 'Unknown'])
X_dtree[:,4] = le_lightcond.transform(X_dtree[:,4])

X_dtree[0:5]
```

Now we can fill the target variable.

```
y_dtree = dtree_df["SEVERITYCODE"].astype(str)

y_dtree[0:5]
```