Road Collisions in Junctions
And
Predictive Modelling

Introduction

- Collisions in junctions can have two consequences: property damage or/and injury
- Collisions in junctions cause long delay
- Increase traffic congestion
- Incurs costs that can be avoided
- Simple measures can reduce collision risks

Exploratory data analysis and predictive modelling can help provide insight to involved parties who can implement safety measures to reduce risk of a collision

Data Wrangling

- Use given data by the course on road collisions for Seattle City
- Read data into correct format and clean accordingly
- Remove duplicate or columns with same elements but which are either in categorical or numerical variables
- Use One Hot Encoding to turn

Dataframe after dropping unwanted columns

- We are left with categorical variables
- No null or nan values in our dataframe

In [5]:
```
# We drop all columns expect the listed above
df.drop(df.columns.difference(['SEVERITYDESC', 'ADDRTYPE', 'JUNCTIONTYPE', 'SDOT_COLDESC', 'WEATHER', 'LIGHTCOND'])\
, axis=1, inplace=True)
df.head()
```

Out[5]:

	ADDRTYPE	SEVERITYDESC	JUNCTIONTYPE	SDOT_COLDESC	WEATHER	LIGHTCOND
0	Intersection	Injury Collision	At Intersection (intersection related)	MOTOR VEHICLE STRUCK MOTOR VEHICLE, FRONT END ...	Overcast	Daylight
1	Block	Property Damage Only Collision	Mid-Block (not related to intersection)	MOTOR VEHICLE STRUCK MOTOR VEHICLE, LEFT SIDE ...	Raining	Dark - Street Lights On
2	Block	Property Damage Only Collision	Mid-Block (not related to intersection)	MOTOR VEHICLE STRUCK MOTOR VEHICLE, REAR END	Overcast	Daylight
3	Block	Property Damage Only Collision	Mid-Block (not related to intersection)	MOTOR VEHICLE STRUCK MOTOR VEHICLE, FRONT END ...	Clear	Daylight
4	Intersection	Injury Collision	At Intersection (intersection related)	MOTOR VEHICLE STRUCK MOTOR VEHICLE, FRONT END ...	Raining	Daylight

In [6]:
```
#show in a df format null value in boolean results
null_values=df.isnull()
null_values
```

Out[6]:

	ADDRTYPE	SEVERITYDESC	JUNCTIONTYPE	SDOT_COLDESC	WEATHER	LIGHTCOND
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
...	...	...	...	...	...	...
194668	False	False	False	False	False	False
194669	False	False	False	False	False	False
194670	False	False	False	False	False	False
194671	False	False	False	False	False	False
194672	False	False	False	False	False	False

Describe function: statistical summary

- Provides descriptive summary of the attributes
- Around 6% nan rows
- Use dropna function to drop all rows with nans or null variables
- We are left with no null variables

```
In [8]: #gives statistics for categorical variables
        df.describe(include='O')
Out[8]:
```

	ADDRTYPE	SEVERITYDESC	JUNCTIONTYPE	SDOT_COLDESC	WEATHER	LIGHTCOND
count	192747	194673	188344	194673	189592	189503
unique	3	2	7	39	11	9
top	Block	Property Damage Only Collision	Mid-Block (not related to intersection)	MOTOR VEHICLE STRUCK MOTOR VEHICLE, FRONT END ...	Clear	Daylight
freq	126926	136485	89800	85209	111135	116137

```
In [9]: df_with_nans=df.dropna()
        df_with_nans.shape
Out[9]: (182954, 6)

In [10]: a=(1-(182954/194673))*100
         print("%.2f" % a,"%")
         6.02 %
```

```
In [14]: #Also check to confirm no more null values
         df.isnull().sum()
Out[14]: ADDRTYPE        0
         SEVERITYDESC    0
         JUNCTIONTYPE    0
         SDOT_COLDESC    0
         WEATHER         0
         LIGHTCOND       0
         dtype: int64
```

One Hot Encoding

- Turn categorical variables into numerical (0 or 1 for true for false)

In [20]:

```
Feature=df['SEVERITYDESC']
Feature=pd.concat([Feature, pd.get_dummies(df[['ADDRTYPE','JUNCTIONTYPE','SDOT_COLDESC','WEATHER','LIGHTCOND']])], axis=1)

Feature.head()
```

Out[20]:

	SEVERITYDESC	ADDRTYPE_Alley	ADDRTYPE_Block	ADDRTYPE_Intersection	JUNCTIONTYPE_Intersection (but not related to intersection)
0	Injury Collision	0	0	1	0
1	Property Damage Only Collision	0	1	0	0
2	Property Damage Only Collision	0	1	0	0
3	Property Damage Only Collision	0	1	0	0
4	Injury Collision	0	0	1	0

5 rows × 70 columns

Preparing the final dataset before Model Development

```
In [22]: X=Feature
         X[0:5]
```

Out[22]:

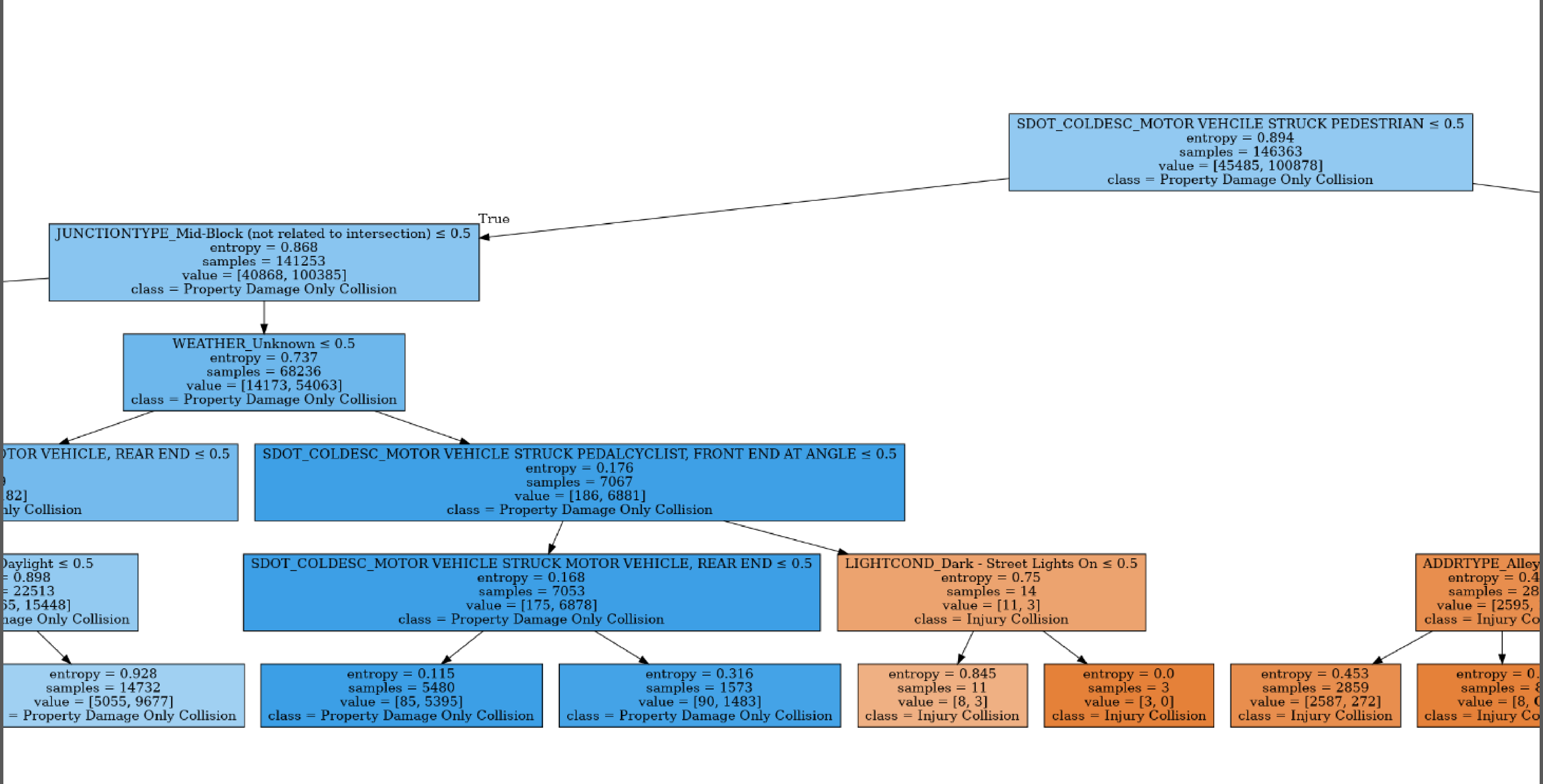
	ADDRTYPE_Alley	ADDRTYPE_Block	ADDRTYPE_Intersection	JUNCTIONTYPE_At Intersection (but not related to intersection)	JUNCTIONTY Intersection (intersection related)
0	0	0	1	0	1
1	0	1	0	0	0
2	0	1	0	0	0
3	0	1	0	0	0
4	0	0	1	0	1

5 rows × 69 columns

```
In [23]: y=df['SEVERITYDESC'].values
         y[0:5]

Out[23]: array(['Injury Collision', 'Property Damage Only Collision',
                'Property Damage Only Collision', 'Property Damage Only Collision',
                'Injury Collision'], dtype=object)

In [24]: print("Feature shape:", Feature.shape)
         print("X shape:",X.shape)
         print ("y shape:", y.shape)

         Feature shape: (182954, 69)
         X shape: (182954, 69)
         y shape: (182954,)
```

Model Development

- Import libraries and modules to perform
  - Train_test_split
  - Decision Tree Classifier
  - Metrics
  - Matplotlib for visualisation of decision tree


SDOT_COLDESC_MOTOR VEHCILE STRUCK PEDESTRIAN ≤ 0.5
entropy = 0.894
samples = 146363
value = [45485, 100878]
class = Property Damage Only Collision
True
JUNCTIONTYPE_Mid-Block (not related to intersection) ≤ 0.5
entropy = 0.868
samples = 141253
value = [40868, 100385]
class = Property Damage Only Collision
WEATHER_Unknown ≤ 0.5
entropy = 0.737
samples = 68236
value = [14173, 54063]
class = Property Damage Only Collision
SDOT_COLDESC_MOTOR VEHICLE STRUCK PEDALCYCLIST, FRONT END AT ANGLE ≤ 0.5
entropy = 0.176
samples = 7067
value = [186, 6881]
class = Property Damage Only Collision
SDOT_COLDESC_MOTOR VEHICLE STRUCK MOTOR VEHICLE, REAR END ≤ 0.5
entropy = 0.168
samples = 7053
value = [175, 6878]
class = Property Damage Only Collision
LIGHTCOND_Dark - Street Lights On ≤ 0.5
entropy = 0.75
samples = 14
value = [11, 3]
class = Injury Collision
entropy = 0.928
samples = 14732
value = [5055, 9677]
entropy = 0.115
samples = 5480
value = [85, 5395]
class = Property Damage Only Collision
entropy = 0.316
samples = 1573
value = [90, 1483]
class = Property Damage Only Collision
entropy = 0.845
samples = 11
value = [8, 3]
class = Injury Collision
entropy = 0.0
samples = 3
value = [3, 0]
class = Injury Collision
entropy = 0.453
samples = 2859
value = [2587, 272]
class = Injury Collision

Evaluation

- Jaccard Index = 0.73805
- F1-Score = 0.668051

Evaluation

```
In [33]: from sklearn.metrics import jaccard_similarity_score
         from sklearn.metrics import f1_score

In [34]: Tree_Prediction=Accident_Severity_Model.predict(X_test)
         jc=jaccard_similarity_score(y_test, Tree_Prediction)
         fs=f1_score(y_test, Tree_Prediction, average='weighted')

In [35]: list_jc = [jc]
         list_fs = [fs]

In [45]: df = pd.DataFrame(list_jc, index=['Decision Tree'])

         df.columns = ['Jaccard']
         df.insert(loc=1, column='F1-score', value=list_fs)

         df
```

Out[45]:

	Jaccard	F1-score
Decision Tree	0.73805	0.668051

Conclusion

- Drop nans if the range is between 5-10% of the total rows
- Work with categorical attributes using One Hot Encoding to solve the issue
- Use Decision Tree technique for classification model