

Jupyter_Notebook

September 27, 2020

1 Car Accident Severity Analysis using Machine Learning Algorithms

1.0.1 Introduction & Business Understanding

Road accidents are one of the major causes of death and disability all over the world. The major reasons for road accidents can be environmental conditions such as weather, traffic on road, type of road, speed and light conditions. This paper addresses the in-depth analysis that identifies as the contributory factors behind the road accidents and the quantification of the factors that affect the frequency and severity of accidents based on the crash data available. The severity of each accident can be predicted quite accurately with various classification machine learning algorithms. This can ultimately help the government, traffic police, medical institutions, individual drivers and the insurance companies by getting useful insights of the accident severity regarding the causes and consequences of the accidents. The Machine Learning 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. The model will predict the accident severity with various supervised machine learning algorithms i.e. * Algorithm A. Logistic regression * Algorithm B. The K-Nearest Neighbors (KNN) algorithm * Algorithm C. Decision Tree * Algorithm D. Random Forest And finally, the accuracy score for each considered machine learning algorithm will be plotted to check which algorithm performs better.

1.0.2 Data Understanding

The data used for this project was collected by the SDOT traffic management Division and Seattle Traffic Records Group from 2004 to present. It was downloaded from the link shared in the IBM Applied Data Science Capstone course. The data consists of 38 independent variables and 194,673 rows. The dependent variable, "SEVERITYCODE", contains numbers that correspond to different levels of severity caused by an accident from 1 to 2. Severity codes are as follows:

- Property Damage Only Collision(1)
- Injury Collision(2)

Furthermore, as there are null values in some records, the data needs to be pre-processed before proceeding further.

Importing the libraries

```
[2]: from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

```
[3]: import matplotlib.pyplot as plt
      %matplotlib inline
      import numpy as np
      import pandas as pd
      import seaborn as sns
      from sklearn.neighbors import KNeighborsClassifier

      # Import DecisionTreeClassifier from sklearn.tree
      from sklearn.tree import DecisionTreeClassifier

      # Import RandomForestClassifier
      from sklearn.ensemble import RandomForestClassifier

      # Import LogisticRegression
      from sklearn.linear_model import LogisticRegression

      from sklearn.model_selection import train_test_split
      from sklearn.feature_selection import SelectFromModel
      from sklearn.metrics import accuracy_score
```

Reading the CSV Data

```
[4]: # Reading the CSV file "Data-Collisions"

df = pd.read_csv (r"C:\Users\salma\Desktop\Data-Collisions.csv")
df.info()
pd.options.display.max_columns=200
df.head()
```

c:\users\salma\desktop\projects\venv\new\new\lib\site-packages\IPython\core\interactiveshell.py:3145: DtypeWarning: Columns (32) have mixed types.Specify dtype option on import or set low_memory=False.

```
has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 194673 entries, 0 to 194672
```

```
Data columns (total 37 columns):
```

#	Column	Non-Null Count	Dtype
0	SEVERITYCODE	194673 non-null	int64
1	longitude	189339 non-null	float64
2	latitude	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	SEVERITYDESC	194673 non-null	object
14	COLLISIONTYPE	189769 non-null	object
15	PERSONCOUNT	194673 non-null	int64
16	PEDCOUNT	194673 non-null	int64
17	PEDCYLCOUNT	194673 non-null	int64
18	VEHCOUNT	194673 non-null	int64
19	INCDATE	194673 non-null	object
20	INCDTTM	194673 non-null	object
21	JUNCTIONTYPE	188344 non-null	object
22	SDOT_COLCODE	194673 non-null	int64
23	SDOT_COLDESC	194673 non-null	object
24	INATTENTIONIND	29805 non-null	object
25	UNDERINFL	189789 non-null	object
26	WEATHER	189592 non-null	object
27	ROADCOND	189661 non-null	object
28	LIGHTCOND	189503 non-null	object
29	PEDROWNOTGRNT	4667 non-null	object
30	SDOTCOLNUM	114936 non-null	float64
31	SPEEDING	9333 non-null	object
32	ST_COLCODE	194655 non-null	object
33	ST_COLDESC	189769 non-null	object
34	SEGLANEKEY	194673 non-null	int64
35	CROSSWALKKEY	194673 non-null	int64
36	HITPARKEDCAR	194673 non-null	object

dtypes: float64(4), int64(11), object(22)

memory usage: 55.0+ MB

[4]:	SEVERITYCODE	longitude	latitude	OBJECTID	INCKEY	COLDKEY	REPORTNO	\
0	2	-122.323148	47.703140	1	1307	1307	3502005	
1	1	-122.347294	47.647172	2	52200	52200	2607959	
2	1	-122.334540	47.607871	3	26700	26700	1482393	
3	1	-122.334803	47.604803	4	1144	1144	3503937	
4	2	-122.306426	47.545739	5	17700	17700	1807429	

	STATUS	ADDRTYPE	INTKEY	\
0	Matched	Intersection	37475.0	
1	Matched	Block	NaN	
2	Matched	Block	NaN	
3	Matched	Block	NaN	
4	Matched	Intersection	34387.0	

	LOCATION	EXCEPTRSNCODE	EXCEPTRSNDESC	\
0	5TH AVE NE AND NE 103RD ST		NaN	

1	AURORA BR BETWEEN RAYE ST AND BRIDGE WAY N	NaN	NaN
2	4TH AVE BETWEEN SENECA ST AND UNIVERSITY ST	NaN	NaN
3	2ND AVE BETWEEN MARION ST AND MADISON ST		NaN
4	SWIFT AVE S AND SWIFT AV OFF RP	NaN	NaN

	SEVERITYDESC	COLLISIONTYPE	PERSONCOUNT	PEDCOUNT	\
0	Injury Collision	Angles	2	0	
1	Property Damage Only Collision	Sideswipe	2	0	
2	Property Damage Only Collision	Parked Car	4	0	
3	Property Damage Only Collision	Other	3	0	
4	Injury Collision	Angles	2	0	

	PEDCYLCOUNT	VEHCOUNT	INCDATE	INCDTTM	\
0	0	2	2013/03/27 00:00:00+00	3/27/2013 14:54	
1	0	2	2006/12/20 00:00:00+00	12/20/2006 18:55	
2	0	3	2004/11/18 00:00:00+00	11/18/2004 10:20	
3	0	3	2013/03/29 00:00:00+00	3/29/2013 9:26	
4	0	2	2004/01/28 00:00:00+00	1/28/2004 8:04	

	JUNCTIONTYPE	SDOT_COLCODE	\
0	At Intersection_related to intersection	11	
1	Mid-Block (not related to intersection)	16	
2	Mid-Block (not related to intersection)	14	
3	Mid-Block (not related to intersection)	11	
4	At Intersection_related to intersection	11	

	SDOT_COLDESC	INATTENTIONIND	UNDERINFL	\
0	MOTOR VEHICLE STRUCK MOTOR VEHICLE, FRONT END ...	NaN	N	
1	MOTOR VEHICLE STRUCK MOTOR VEHICLE, LEFT SIDE ...	NaN	N	
2	MOTOR VEHICLE STRUCK MOTOR VEHICLE, REAR END	NaN	N	
3	MOTOR VEHICLE STRUCK MOTOR VEHICLE, FRONT END ...	NaN	N	
4	MOTOR VEHICLE STRUCK MOTOR VEHICLE, FRONT END ...	NaN	N	

	WEATHER	ROADCOND	LIGHTCOND	PEDROWNOTGRNT	SDOTCOLNUM	\
0	Overcast	Wet	Daylight	NaN	NaN	
1	Raining	Wet	Dark - Street Lights On	NaN	6354039.0	
2	Overcast	Dry	Daylight	NaN	4323031.0	
3	Clear	Dry	Daylight	NaN	NaN	
4	Raining	Wet	Daylight	NaN	4028032.0	

	SPEEDING	ST_COLCODE	ST_COLDESC	\
0	NaN	10	Entering at angle	
1	NaN	11	From same direction - both going straight - bo...	
2	NaN	32	One parked--one moving	
3	NaN	23	From same direction - all others	
4	NaN	10	Entering at angle	

	SEGLANEKEY	CROSSWALKKEY	HITPARKEDCAR
0	0	0	N
1	0	0	N
2	0	0	N
3	0	0	N
4	0	0	N

Checking the percentage (%) of missing values in the columns

```
[5]: df.isna().mean().round(4) * 100
```

```
[5]: SEVERITYCODE      0.00
longitude      2.74
latitude      2.74
OBJECTID      0.00
INCKEY      0.00
COLDETKEY      0.00
REPORTNO      0.00
STATUS      0.00
ADDRTYPE      0.99
INTKEY      66.57
LOCATION      1.38
EXCEPTRSNCODE      56.43
EXCEPTRSNDESC      97.10
SEVERITYDESC      0.00
COLLISIONTYPE      2.52
PERSONCOUNT      0.00
PEDCOUNT      0.00
PEDCYLCOUNT      0.00
VEHCOUNT      0.00
INCDATE      0.00
INCDTTM      0.00
JUNCTIONTYPE      3.25
SDOT_COLCODE      0.00
SDOT_COLDESC      0.00
INATTENTIONIND      84.69
UNDERINFL      2.51
WEATHER      2.61
ROADCOND      2.57
LIGHTCOND      2.66
PEDROWNOTGRNT      97.60
SDOTCOLNUM      40.96
SPEEDING      95.21
ST_COLCODE      0.01
ST_COLDESC      2.52
SEGLANEKEY      0.00
CROSSWALKKEY      0.00
```

```
HITPARKEDCAR      0.00
dtype: float64
```

```
[6]: df.shape
```

```
[6]: (194673, 37)
```

Initial segmentation of the list of features

```
[7]: numeric_features = df[["PERSONCOUNT", "PEDCOUNT", "PEDCYLCOUNT", "VEHCOUNT",
    ↪ "SEVERITYCODE"]]

categorical_features=df[["ADDRTYPE", "LOCATION", "COLLISIONTYPE",
    ↪ "INCDATE", "INCDTTM", "JUNCTIONTYPE",
    ↪ "SDOT_COLDESC", "UNDERINFL", "WEATHER", "ROADCOND",
    ↪ "LIGHTCOND", "ST_COLDESC", "HITPARKEDCAR"]]
```

Checking the Target Variable

```
[8]: df["SEVERITYCODE"].value_counts()
```

```
[8]: 1    136485
     2     58188
     Name: SEVERITYCODE, dtype: int64
```

Description of the Numeric Features

```
[9]: numeric_features.describe()
```

```
[9]:
```

	PERSONCOUNT	PEDCOUNT	PEDCYLCOUNT	VEHCOUNT	\
count	194673.000000	194673.000000	194673.000000	194673.000000	
mean	2.444427	0.037139	0.028391	1.920780	
std	1.345929	0.198150	0.167413	0.631047	
min	0.000000	0.000000	0.000000	0.000000	
25%	2.000000	0.000000	0.000000	2.000000	
50%	2.000000	0.000000	0.000000	2.000000	
75%	3.000000	0.000000	0.000000	2.000000	
max	81.000000	6.000000	2.000000	12.000000	

	SEVERITYCODE
count	194673.000000
mean	1.298901
std	0.457778
min	1.000000
25%	1.000000
50%	1.000000
75%	2.000000
max	2.000000

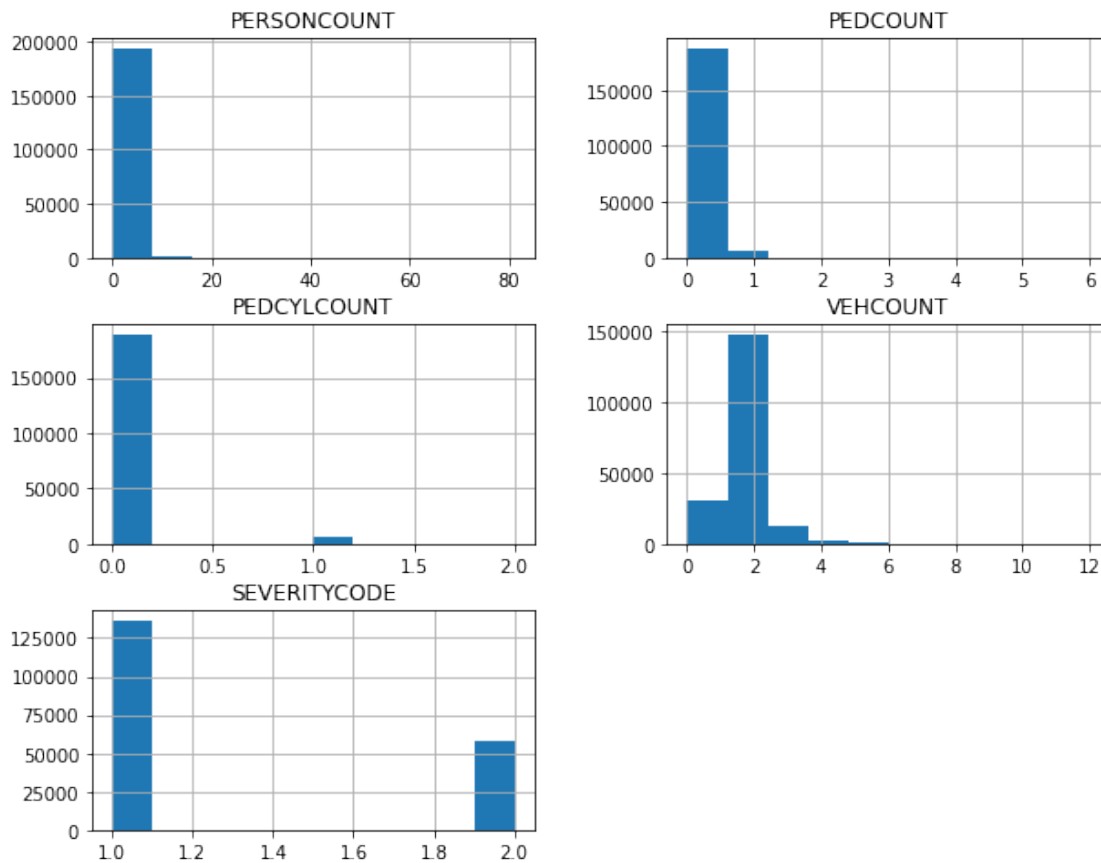
Numeric Features Distribution

```
[10]: numeric_features.hist(figsize=[10,8])  
plt.suptitle("Numeric features distribution", fontsize=20)  
plt.show()
```

```
[10]: array([[<AxesSubplot:title={'center':'PERSONCOUNT'}>,  
             <AxesSubplot:title={'center':'PEDCOUNT'}>],  
          [<AxesSubplot:title={'center':'PEDCYLCOUNT'}>,  
             <AxesSubplot:title={'center':'VEHCOUNT'}>],  
          [<AxesSubplot:title={'center':'SEVERITYCODE'}>, <AxesSubplot:>]],  
      dtype=object)
```

```
[10]: Text(0.5, 0.98, 'Numeric features distribution')
```

Numeric features distribution



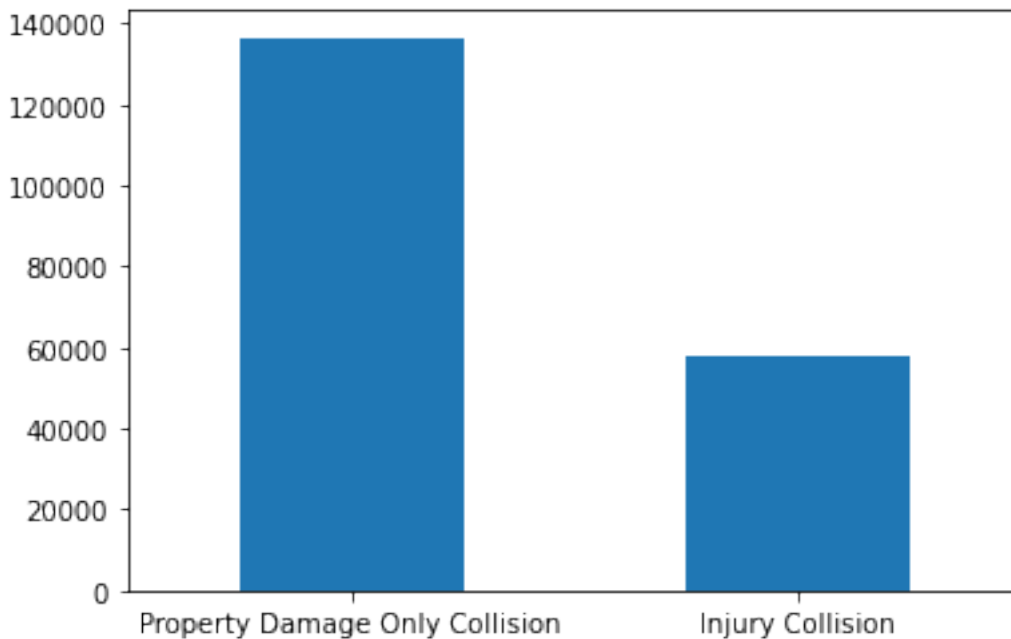
Categorical Features Distribution

“SEVERITYDESC” (Accident Severity Description)

```
[11]: df['SEVERITYDESC'].value_counts().plot(kind='bar')
plt.xticks(rotation=0)
```

```
[11]: <AxesSubplot:>
```

```
[11]: (array([0, 1]),
      [Text(0, 0, 'Property Damage Only Collision'),
       Text(1, 0, 'Injury Collision')])
```



“COLLISIONTYPE”

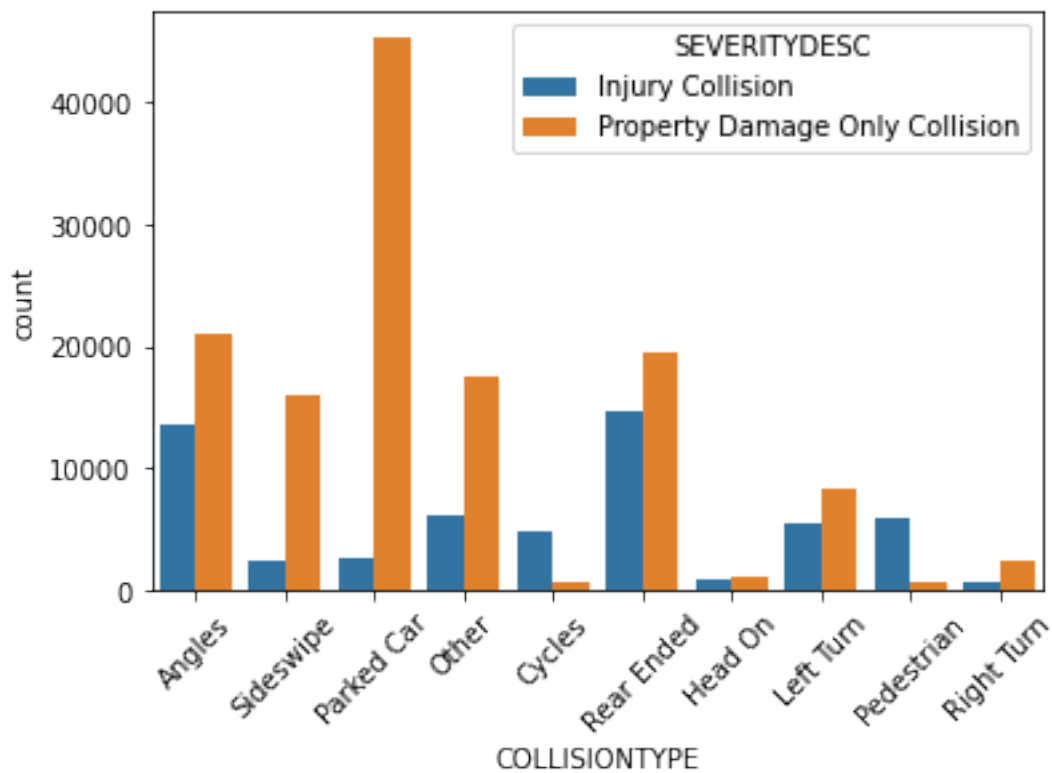
```
[12]: # Collision Type
sns.countplot(x="COLLISIONTYPE", hue="SEVERITYDESC", data=df)
plt.xticks(rotation=45)
```

```
[12]: <AxesSubplot:xlabel='COLLISIONTYPE', ylabel='count'>
```

```
[12]: (array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
      [Text(0, 0, 'Angles'),
       Text(1, 0, 'Sideswipe'),
       Text(2, 0, 'Parked Car'),
       Text(3, 0, 'Other'),
       Text(4, 0, 'Cycles'),
       Text(5, 0, 'Rear Ended'),
       Text(6, 0, 'Head On'),
       Text(7, 0, 'Left Turn'),
```



```
Text(8, 0, 'Pedestrian'),
Text(9, 0, 'Right Turn']])
```

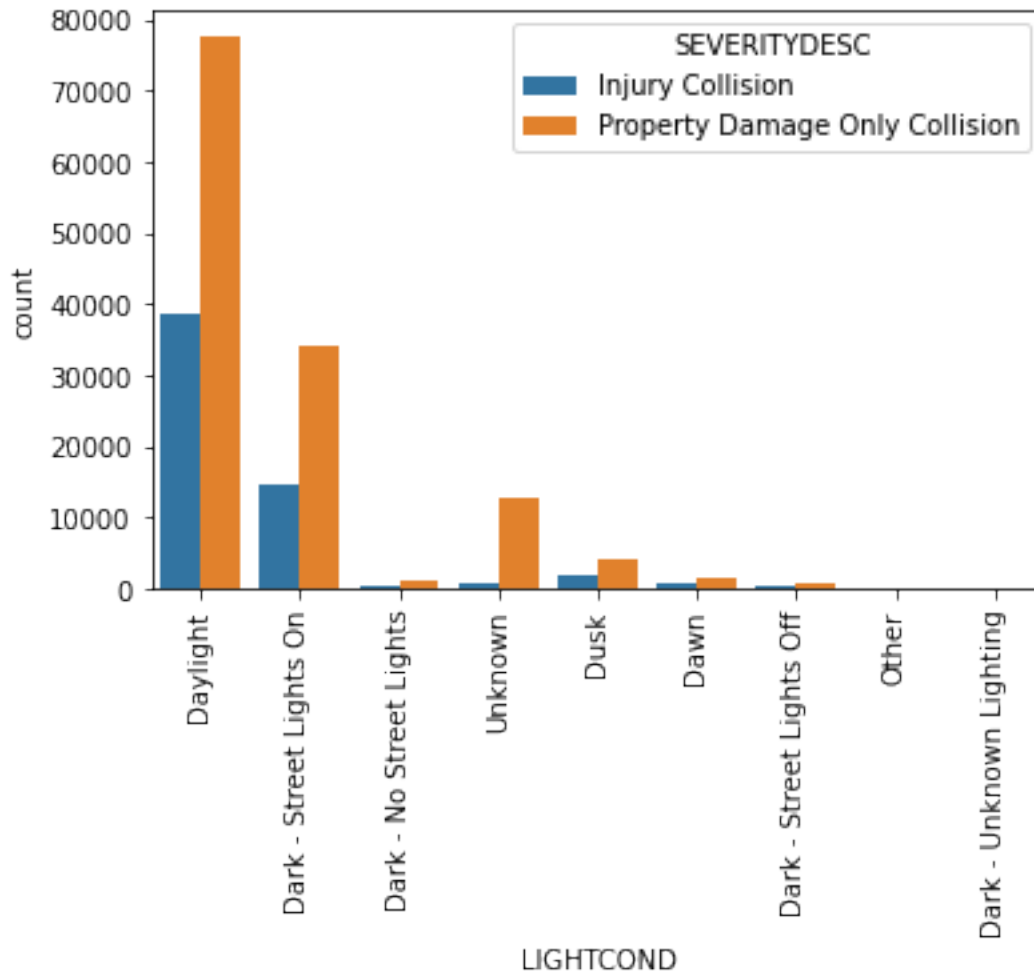


“LIGHTCOND”

```
[13]: # LIGHT CONDITIONS
sns.countplot(x="LIGHTCOND", hue="SEVERITYDESC", data=df)
plt.xticks(rotation=90)
```

```
[13]: <AxesSubplot:xlabel='LIGHTCOND', ylabel='count'>
```

```
[13]: (array([0, 1, 2, 3, 4, 5, 6, 7, 8]),
[Text(0, 0, 'Daylight'),
Text(1, 0, 'Dark - Street Lights On'),
Text(2, 0, 'Dark - No Street Lights'),
Text(3, 0, 'Unknown'),
Text(4, 0, 'Dusk'),
Text(5, 0, 'Dawn'),
Text(6, 0, 'Dark - Street Lights Off'),
Text(7, 0, 'Other'),
Text(8, 0, 'Dark - Unknown Lighting')])
```



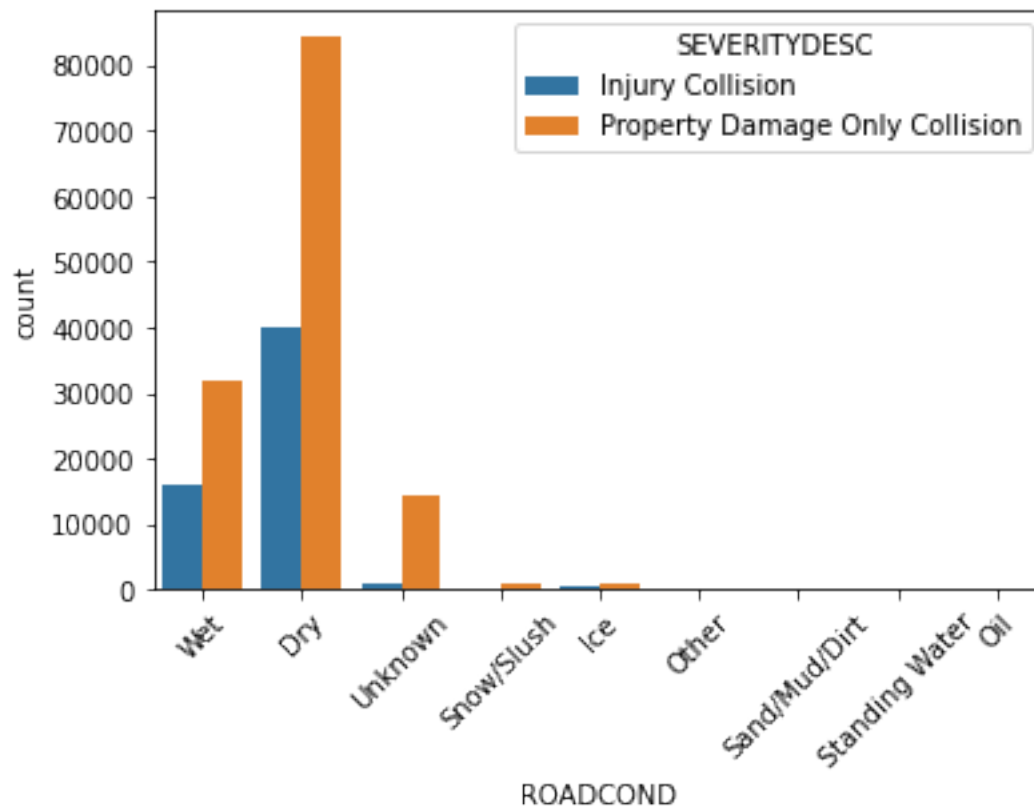
“ROADCOND”

```
[14]: # ROAD CONDITIONS
sns.countplot(x="ROADCOND", hue="SEVERITYDESC", data=df)
plt.xticks(rotation=45)
```

```
[14]: <AxesSubplot:xlabel='ROADCOND', ylabel='count'>
```

```
[14]: (array([0, 1, 2, 3, 4, 5, 6, 7, 8]),
      [Text(0, 0, 'Wet'),
       Text(1, 0, 'Dry'),
       Text(2, 0, 'Unknown'),
       Text(3, 0, 'Snow/Slush'),
       Text(4, 0, 'Ice'),
       Text(5, 0, 'Other'),
       Text(6, 0, 'Sand/Mud/Dirt'),
       Text(7, 0, 'Standing Water'),
```

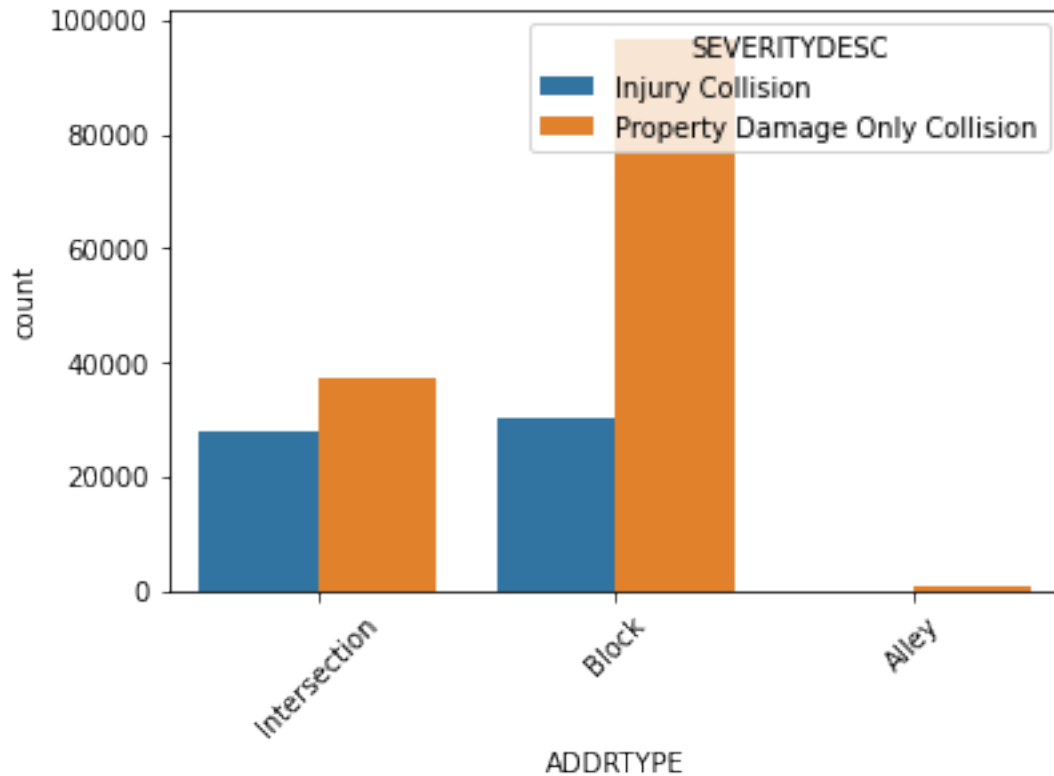
```
Text(8, 0, 'Oil'))]
```



```
[15]: # ADDRTYPE (Address Type)
sns.countplot(x="ADDRTYPE", hue="SEVERITYDESC", data=df)
plt.xticks(rotation=45)
```

```
[15]: <AxesSubplot:xlabel='ADDRTYPE', ylabel='count'>
```

```
[15]: (array([0, 1, 2]),
      [Text(0, 0, 'Intersection'), Text(1, 0, 'Block'), Text(2, 0, 'Alley')])
```



“WEATHER”

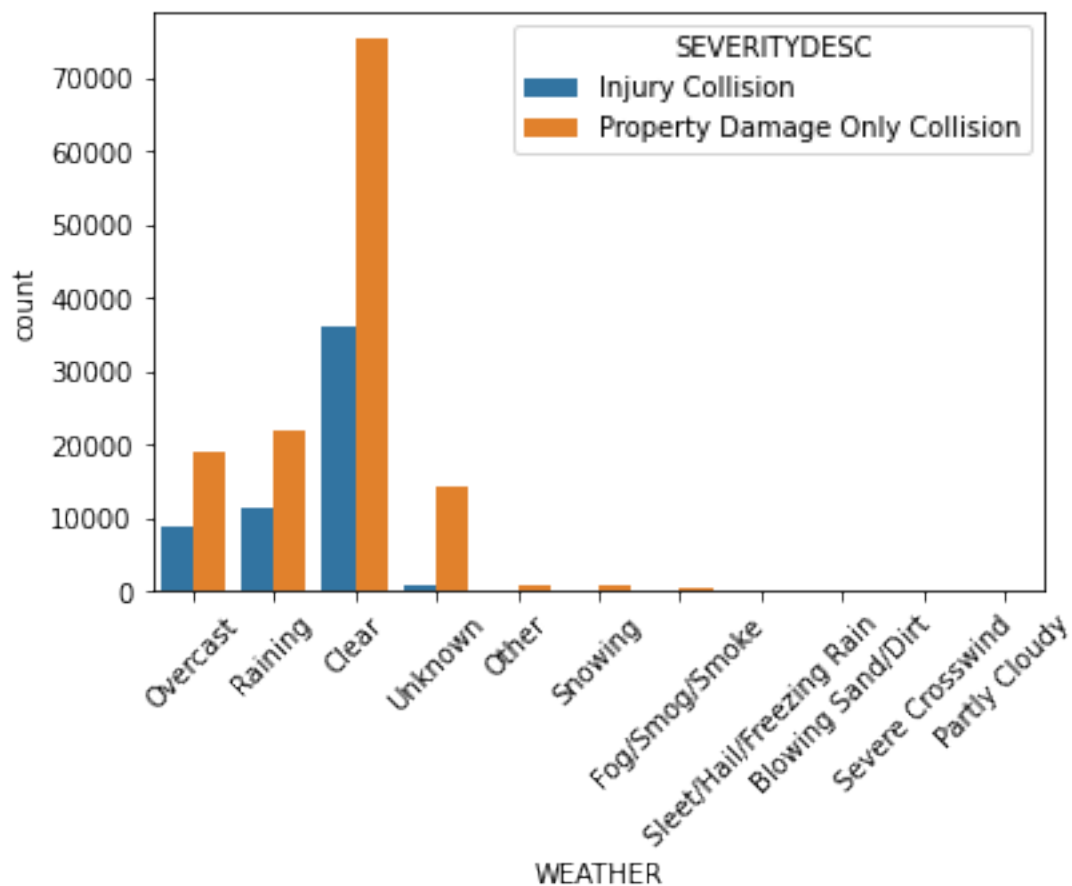
```
[16]: # WEATHER
df['WEATHER'].value_counts().sort_values(ascending=False).to_frame()
sns.countplot(x="WEATHER", hue="SEVERITYDESC", data=df)
plt.xticks(rotation=45)
```

```
[16]:
```

WEATHER	count
Clear	111135
Raining	33145
Overcast	27714
Unknown	15091
Snowing	907
Other	832
Fog/Smog/Smoke	569
Sleet/Hail/Freezing Rain	113
Blowing Sand/Dirt	56
Severe Crosswind	25
Partly Cloudy	5

```
[16]: <AxesSubplot:xlabel='WEATHER', ylabel='count'>
```

```
[16]: (array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
      [Text(0, 0, 'Overcast'),
       Text(1, 0, 'Raining'),
       Text(2, 0, 'Clear'),
       Text(3, 0, 'Unknown'),
       Text(4, 0, 'Other'),
       Text(5, 0, 'Snowing'),
       Text(6, 0, 'Fog/Smog/Smoke'),
       Text(7, 0, 'Sleet/Hail/Freezing Rain'),
       Text(8, 0, 'Blowing Sand/Dirt'),
       Text(9, 0, 'Severe Crosswind'),
       Text(10, 0, 'Partly Cloudy')])
```



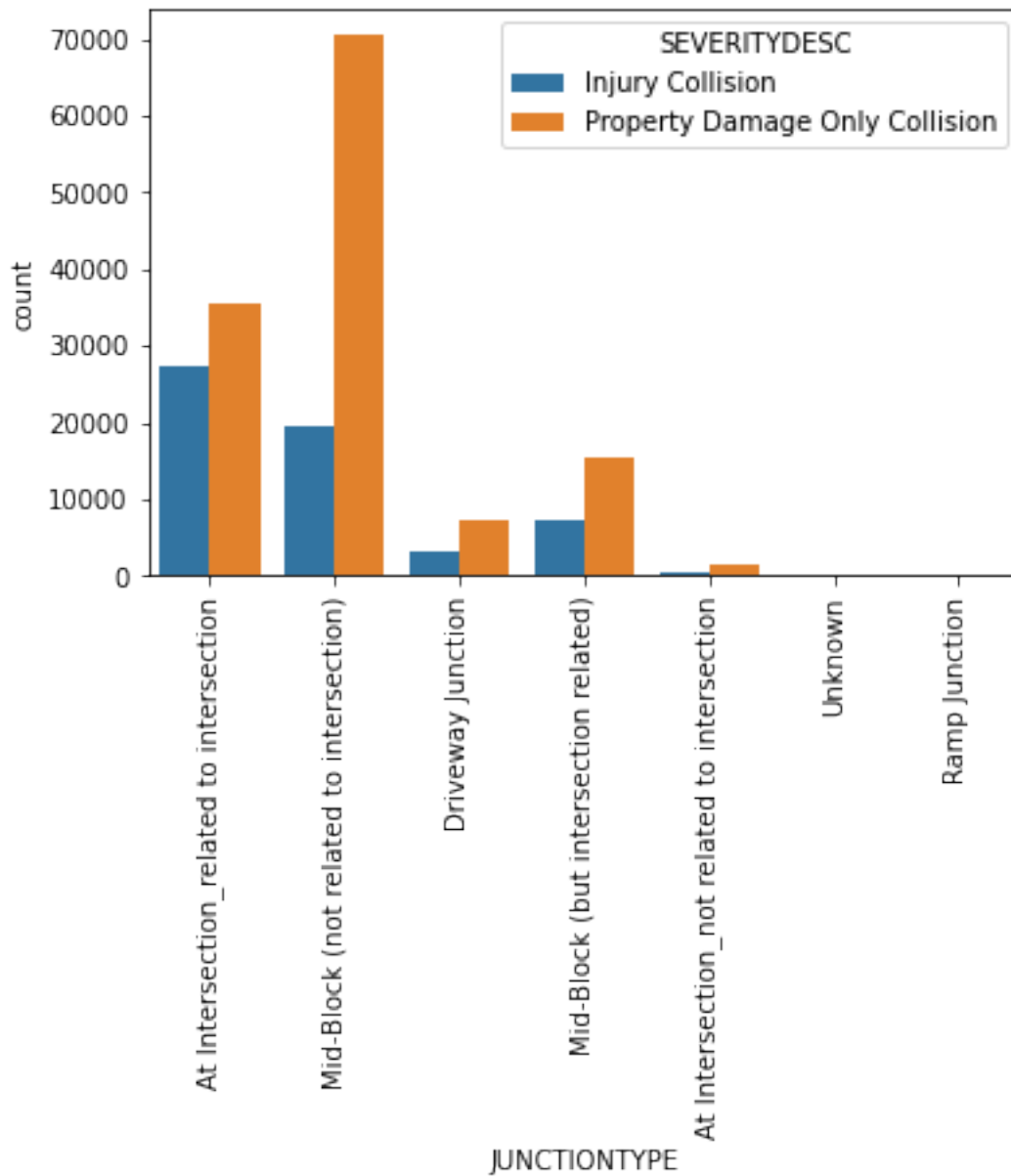
“JUNCTIONTYPE”

```
[17]: # Junction Type

sns.countplot(x="JUNCTIONTYPE", hue="SEVERITYDESC", data=df)
plt.xticks(rotation=90)
```

```
[17]: <AxesSubplot:xlabel='JUNCTIONTYPE', ylabel='count'>
```

```
[17]: (array([0, 1, 2, 3, 4, 5, 6]),  
      [Text(0, 0, 'At Intersection_related to intersection'),  
       Text(1, 0, 'Mid-Block (not related to intersection)'),  
       Text(2, 0, 'Driveway Junction'),  
       Text(3, 0, 'Mid-Block (but intersection related)'),  
       Text(4, 0, 'At Intersection_not related to intersection'),  
       Text(5, 0, 'Unknown'),  
       Text(6, 0, 'Ramp Junction')]))
```

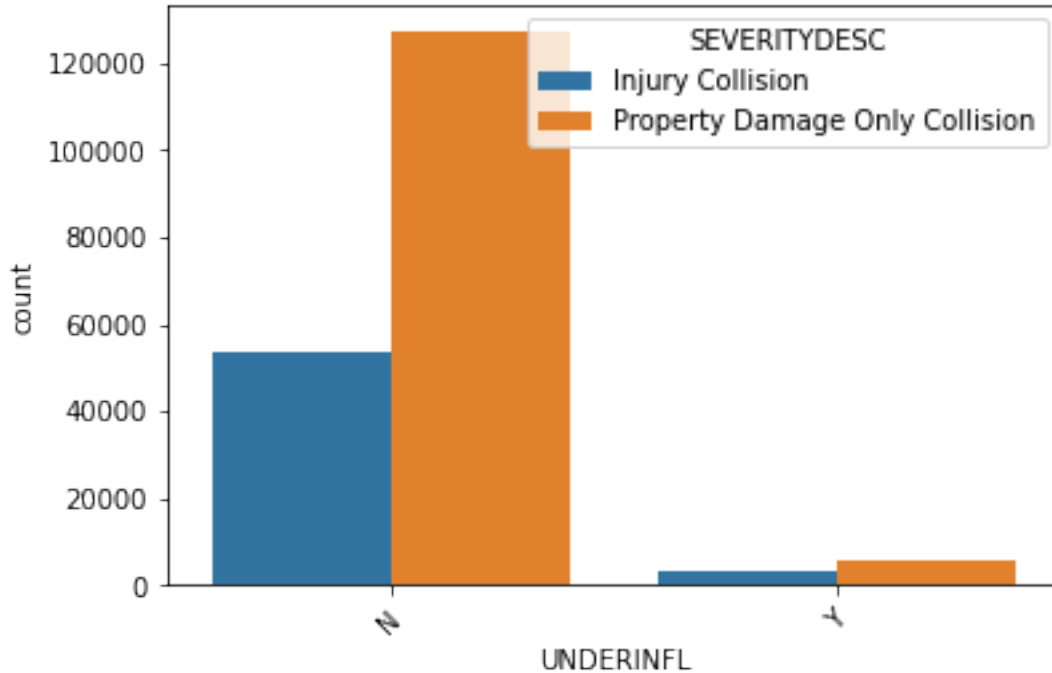


“UNDERINFL” (Under Influence of Alcohol/Drugs)

```
[60]: # UNDER INFLUENCE OF Alcohol/Drugs
sns.countplot(x="UNDERINFL", hue="SEVERITYDESC", data=df)
plt.xticks(rotation=45)
```

```
[60]: <AxesSubplot:xlabel='UNDERINFL', ylabel='count'>
```

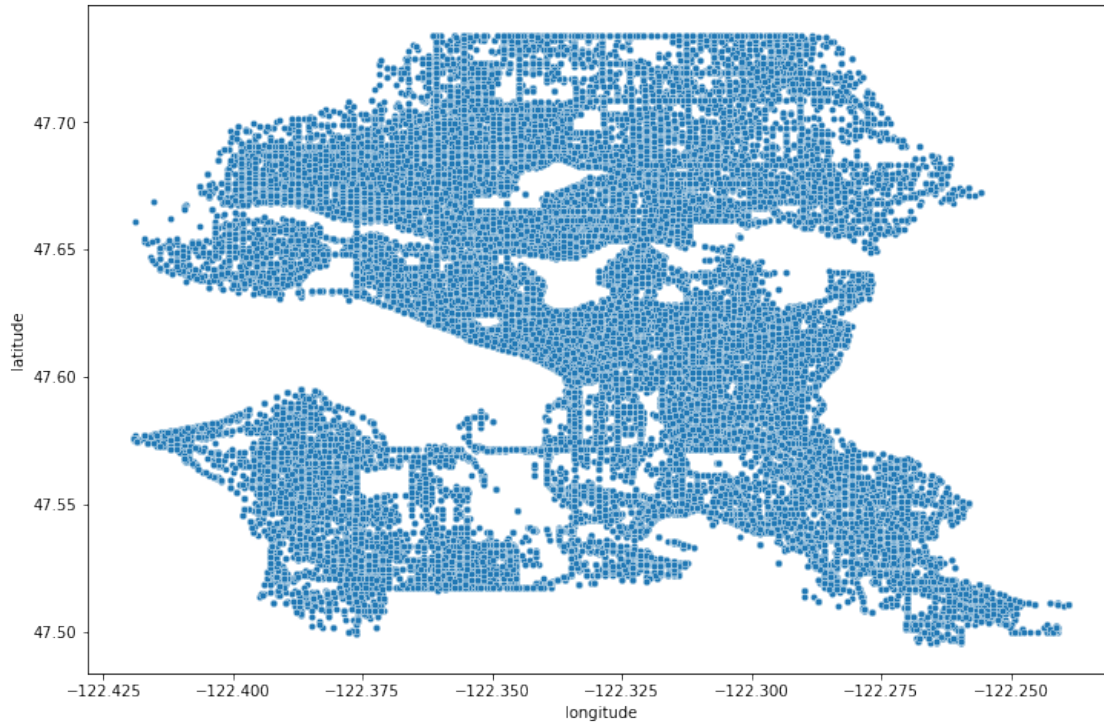
```
[60]: (array([0, 1]), [Text(0, 0, 'N'), Text(1, 0, 'Y')])
```



Scatter plot of the accident coordinates

```
[61]: fig = plt.gcf()
fig.set_size_inches(12, 8)
sns.scatterplot(x='longitude', y='latitude', data=df, legend=False, s=20)
plt.show()
```

```
[61]: <AxesSubplot:xlabel='longitude', ylabel='latitude'>
```



1.0.3 Data Preparation

Formatting the Date & time for the analysis

```
[19]: df['INCDTTM'] = pd.to_datetime(df['INCDTTM'], errors='coerce')
df['Month']=df['INCDTTM'].dt.strftime('%b')
df['Day']=df['INCDTTM'].dt.day
df['Hour']=df['INCDTTM'].dt.hour
df['Weekday']=df['INCDTTM'].dt.strftime('%a')
```

Yearly Distribution of number of accidents

```
[20]: fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(20, 6))

df['Year'] = pd.DatetimeIndex(df['INCDATE']).year
df['Year'].value_counts().sort_index()

sns.countplot(x="Year", data=df, ax=ax1)
sns.countplot(x="Year", hue="SEVERITYDESC", data=df, ax=ax2)

plt.xticks(rotation=45)
ax1.set_title('Car accidents in Seattle by Year', fontsize=20)
ax2.set_title('Car accidents in Seattle by Year & type', fontsize=20)
```



```
[20]: 2004      11865
      2005      15115
      2006      15188
      2007      14456
      2008      13660
      2009      11734
      2010      10808
      2011      10919
      2012      10907
      2013      10577
      2014      11841
      2015      12995
      2016      11659
      2017      10873
      2018      10419
      2019       9412
      2020       2245
      Name: Year, dtype: int64
```

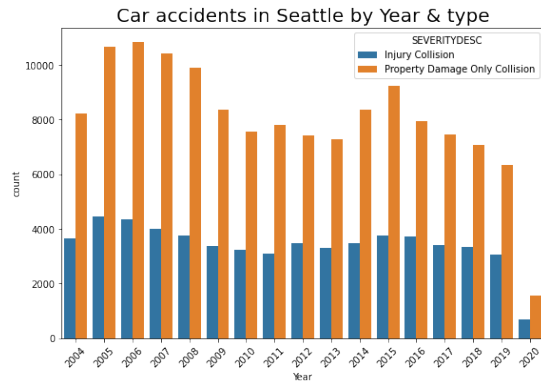
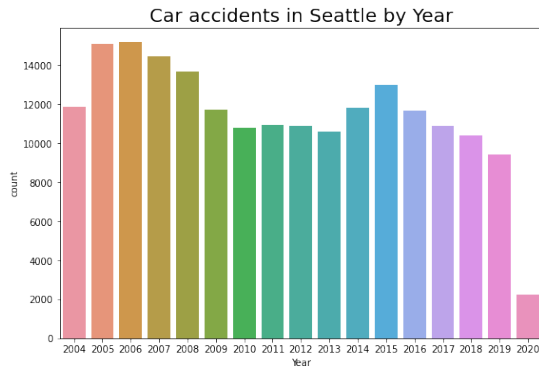
```
[20]: <AxesSubplot:xlabel='Year', ylabel='count'>
```

```
[20]: <AxesSubplot:xlabel='Year', ylabel='count'>
```

```
[20]: (array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16]),
      [Text(0, 0, '2004'),
       Text(1, 0, '2005'),
       Text(2, 0, '2006'),
       Text(3, 0, '2007'),
       Text(4, 0, '2008'),
       Text(5, 0, '2009'),
       Text(6, 0, '2010'),
       Text(7, 0, '2011'),
       Text(8, 0, '2012'),
       Text(9, 0, '2013'),
       Text(10, 0, '2014'),
       Text(11, 0, '2015'),
       Text(12, 0, '2016'),
       Text(13, 0, '2017'),
       Text(14, 0, '2018'),
       Text(15, 0, '2019'),
       Text(16, 0, '2020')])
```

```
[20]: Text(0.5, 1.0, 'Car accidents in Seattle by Year')
```

```
[20]: Text(0.5, 1.0, 'Car accidents in Seattle by Year & type')
```



Checking the Null values in the Dataframe

```
[21]: df.isnull()
```

```
[21]:
```

	SEVERITYCODE	longitude	latitude	OBJECTID	INCKEY	COLDETKEY	\
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	

	REPORTNO	STATUS	ADDRTYPE	INTKEY	LOCATION	EXCEPTRSNCODE	\
0	False	False	False	False	False	False	
1	False	False	False	True	False	True	
2	False	False	False	True	False	True	
3	False	False	False	True	False	False	
4	False	False	False	False	False	True	
...	
194668	False	False	False	True	False	False	
194669	False	False	False	True	False	False	
194670	False	False	False	False	False	False	
194671	False	False	False	False	False	False	
194672	False	False	False	True	False	False	

	EXCEPTRSNDESC	SEVERITYDESC	COLLISIONTYPE	PERSONCOUNT	PEDCOUNT	\
0	True	False	False	False	False	
1	True	False	False	False	False	

2	True	False	False	False	False	False
3	True	False	False	False	False	False
4	True	False	False	False	False	False
...
194668	True	False	False	False	False	False
194669	True	False	False	False	False	False
194670	True	False	False	False	False	False
194671	True	False	False	False	False	False
194672	True	False	False	False	False	False

	PEDCYLCOUNT	VEHCOUNT	INCDATE	INCDTTM	JUNCTIONTYPE	SDOT_COLCODE	\
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	

	SDOT_COLDESC	INATTENTIONIND	UNDERINFL	WEATHER	ROADCOND	LIGHTCOND	\
0	False	True	False	False	False	False	
1	False	True	False	False	False	False	
2	False	True	False	False	False	False	
3	False	True	False	False	False	False	
4	False	True	False	False	False	False	
...	
194668	False	True	False	False	False	False	
194669	False	False	False	False	False	False	
194670	False	True	False	False	False	False	
194671	False	True	False	False	False	False	
194672	False	True	False	False	False	False	

	PEDROWNOTGRNT	SDOTCOLNUM	SPEEDING	ST_COLCODE	ST_COLDESC	\
0	True	True	True	False	False	
1	True	False	True	False	False	
2	True	False	True	False	False	
3	True	True	True	False	False	
4	True	False	True	False	False	
...	
194668	True	True	True	False	False	
194669	True	True	True	False	False	
194670	True	True	True	False	False	
194671	True	True	True	False	False	

194672	True	True	True	False	False		
	SEGLANEKEY	CROSSWALKKEY	HITPARKEDCAR	Month	Day	Hour	Weekday \
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...
194668	False	False	False	False	False	False	False
194669	False	False	False	False	False	False	False
194670	False	False	False	False	False	False	False
194671	False	False	False	False	False	False	False
194672	False	False	False	False	False	False	False

	Year
0	False
1	False
2	False
3	False
4	False
...	...
194668	False
194669	False
194670	False
194671	False
194672	False

[194673 rows x 42 columns]

Selecting and finalizing the features for Machine Learning Model

```
[22]: selected_features = ["SEVERITYCODE", "longitude", "latitude", "PERSONCOUNT",
                           "PEDCOUNT", "PEDCYLCOUNT", "VEHCOUNT", "ADDRTYPE",
                           "COLLISIONTYPE", "WEATHER", "ROADCOND",
                           "LIGHTCOND", "SDOT_COLDESC", "HITPARKEDCAR", "Hour"]
df_sel=df[selected_features].copy()
df_sel.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 194673 entries, 0 to 194672
Data columns (total 15 columns):
#   Column          Non-Null Count  Dtype
---  -
0   SEVERITYCODE     194673 non-null  int64
1   longitude        189339 non-null  float64
2   latitude         189339 non-null  float64
3   PERSONCOUNT    194673 non-null  int64
```

4	PEDCOUNT	194673	non-null	int64
5	PEDCYLCOUNT	194673	non-null	int64
6	VEHCOUNT	194673	non-null	int64
7	ADDRTYPE	192747	non-null	object
8	COLLISIONTYPE	189769	non-null	object
9	WEATHER	189592	non-null	object
10	ROADCOND	189661	non-null	object
11	LIGHTCOND	189503	non-null	object
12	SDOT_COLDESC	194673	non-null	object
13	HITPARKEDCAR	194673	non-null	object
14	Hour	194673	non-null	int64

dtypes: float64(2), int64(6), object(7)
memory usage: 22.3+ MB

```
[23]: df.isnull().sum()
```

```
[23]: SEVERITYCODE          0
longitude          5334
latitude           5334
OBJECTID           0
INCKEY             0
COLDETKEY          0
REPORTNO           0
STATUS             0
ADDRTYPE           1926
INTKEY             129603
LOCATION            2677
EXCEPTRSNCODE      109862
EXCEPTRSNDESC      189035
SEVERITYDESC        0
COLLISIONTYPE       4904
PERSONCOUNT        0
PEDCOUNT           0
PEDCYLCOUNT         0
VEHCOUNT            0
INCDATE            0
INCDTTM            0
JUNCTIONTYPE        6329
SDOT_COLCODE         0
SDOT_COLDESC         0
INATTENTIONIND      164868
UNDERINFL           4884
WEATHER             5081
ROADCOND            5012
LIGHTCOND           5170
PEDROWNOTGRNT       190006
SDOTCOLNUM          79737
```

```

SPEEDING          185340
ST_COLCODE         18
ST_COLDESC        4904
SEGLANEKEY         0
CROSSWALKKEY       0
HITPARKEDCAR       0
Month              0
Day                0
Hour               0
Weekday            0
Year               0
dtype: int64

```

```
[24]: df_sel.shape
```

```
[24]: (194673, 15)
```

Checking the Null values in the selected dataframe and dropping the rows with the null values

```
[25]: df_sel.isnull().mean()
```

```

[25]: SEVERITYCODE      0.000000
longitude      0.027400
latitude       0.027400
PERSONCOUNT   0.000000
PEDCOUNT      0.000000
PEDCYLCOUNT    0.000000
VEHCOUNT       0.000000
ADDRTYPE       0.009894
COLLISIONTYPE  0.025191
WEATHER        0.026100
ROADCOND       0.025746
LIGHTCOND      0.026557
SDOT_COLDESC   0.000000
HITPARKEDCAR   0.000000
Hour           0.000000
dtype: float64

```

```

[26]: df_sel.dropna(subset=df_sel.columns[df_sel.isnull().mean()!=0], how='any',
↳axis=0, inplace=True)
df_sel.shape

```

```
[26]: (184146, 15)
```

```
[27]: # Export the data with selected features
```

```
df_sel.to_csv('./Data-Collisions_clean_sel.csv', index=False)
```

Generating the dummies for the Categorical Data

```
[28]: # Generate dummies for categorical data
df_dummy = pd.get_dummies(df_sel, drop_first=True)
# Export data
df_dummy.to_csv("./Data-Collisions_{}_dummy.csv", index=False)
df_dummy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 184146 entries, 0 to 194672
Data columns (total 83 columns):
```

#	Column	Non-Null Count	Dtype
0	SEVERITYCODE	184146 non-null	int64
1	longitude	184146 non-null	float64
2	latitude	184146 non-null	float64
3	PERSONCOUNT	184146 non-null	int64
4	PEDCOUNT	184146 non-null	int64
5	PEDCYLCOUNT	184146 non-null	int64
6	VEHCOUNT	184146 non-null	int64
7	Hour	184146 non-null	int64
8	ADDRTYPE_Intersection	184146 non-null	uint8
9	COLLISIONTYPE_Cycles	184146 non-null	uint8
10	COLLISIONTYPE_Head On	184146 non-null	uint8
11	COLLISIONTYPE_Left Turn	184146 non-null	uint8
12	COLLISIONTYPE_Other	184146 non-null	uint8
13	COLLISIONTYPE_Parked Car	184146 non-null	uint8
14	COLLISIONTYPE_Pedestrian	184146 non-null	uint8
15	COLLISIONTYPE_Rear Ended		

```

184146 non-null uint8
  16 COLLISIONTYPE_Right Turn
184146 non-null uint8
  17 COLLISIONTYPE_Sideswipe
184146 non-null uint8
  18 WEATHER_Clear
184146 non-null uint8
  19 WEATHER_Fog/Smog/Smoke
184146 non-null uint8
  20 WEATHER_Other
184146 non-null uint8
  21 WEATHER_Overcast
184146 non-null uint8
  22 WEATHER_Partly Cloudy
184146 non-null uint8
  23 WEATHER_Raining
184146 non-null uint8
  24 WEATHER_Severe Crosswind
184146 non-null uint8
  25 WEATHER_Sleet/Hail/Freezing Rain
184146 non-null uint8
  26 WEATHER_Snowing
184146 non-null uint8
  27 WEATHER_Unknown
184146 non-null uint8
  28 ROADCOND_Ice
184146 non-null uint8
  29 ROADCOND_Oil
184146 non-null uint8
  30 ROADCOND_Other
184146 non-null uint8
  31 ROADCOND_Sand/Mud/Dirt
184146 non-null uint8
  32 ROADCOND_Snow/Slush
184146 non-null uint8
  33 ROADCOND_Standing Water
184146 non-null uint8
  34 ROADCOND_Unknown
184146 non-null uint8
  35 ROADCOND_Wet
184146 non-null uint8
  36 LIGHTCOND_Dark - Street Lights Off
184146 non-null uint8
  37 LIGHTCOND_Dark - Street Lights On
184146 non-null uint8
  38 LIGHTCOND_Dark - Unknown Lighting
184146 non-null uint8
  39 LIGHTCOND_Dawn

```


184146 non-null uint8
40 LIGHTCOND_Daylight
184146 non-null uint8
41 LIGHTCOND_Dusk
184146 non-null uint8
42 LIGHTCOND_Other
184146 non-null uint8
43 LIGHTCOND_Unknown
184146 non-null uint8
44 SDOT_COLDESC_DRIVERLESS VEHICLE RAN OFF ROAD - NO COLLISION
184146 non-null uint8
45 SDOT_COLDESC_DRIVERLESS VEHICLE STRUCK MOTOR VEHICLE FRONT END AT ANGLE
184146 non-null uint8
46 SDOT_COLDESC_DRIVERLESS VEHICLE STRUCK MOTOR VEHICLE LEFT SIDE AT ANGLE
184146 non-null uint8
47 SDOT_COLDESC_DRIVERLESS VEHICLE STRUCK MOTOR VEHICLE LEFT SIDE SIDESWIPE
184146 non-null uint8
48 SDOT_COLDESC_DRIVERLESS VEHICLE STRUCK MOTOR VEHICLE REAR END
184146 non-null uint8
49 SDOT_COLDESC_DRIVERLESS VEHICLE STRUCK MOTOR VEHICLE RIGHT SIDE AT ANGLE
184146 non-null uint8
50 SDOT_COLDESC_DRIVERLESS VEHICLE STRUCK MOTOR VEHICLE RIGHT SIDE SIDESWIPE
184146 non-null uint8
51 SDOT_COLDESC_DRIVERLESS VEHICLE STRUCK OBJECT IN ROADWAY
184146 non-null uint8
52 SDOT_COLDESC_DRIVERLESS VEHICLE STRUCK PEDESTRIAN
184146 non-null uint8
53 SDOT_COLDESC_MOTOR VEHICLE STRUCK PEDESTRIAN
184146 non-null uint8
54 SDOT_COLDESC_MOTOR VEHICLE OVERTURNED IN ROAD
184146 non-null uint8
55 SDOT_COLDESC_MOTOR VEHICLE RAN OFF ROAD - HIT FIXED OBJECT
184146 non-null uint8
56 SDOT_COLDESC_MOTOR VEHICLE RAN OFF ROAD - NO COLLISION
184146 non-null uint8
57 SDOT_COLDESC_MOTOR VEHICLE STRUCK MOTOR VEHICLE, FRONT END AT ANGLE
184146 non-null uint8
58 SDOT_COLDESC_MOTOR VEHICLE STRUCK MOTOR VEHICLE, LEFT SIDE AT ANGLE
184146 non-null uint8
59 SDOT_COLDESC_MOTOR VEHICLE STRUCK MOTOR VEHICLE, LEFT SIDE SIDESWIPE
184146 non-null uint8
60 SDOT_COLDESC_MOTOR VEHICLE STRUCK MOTOR VEHICLE, REAR END
184146 non-null uint8
61 SDOT_COLDESC_MOTOR VEHICLE STRUCK MOTOR VEHICLE, RIGHT SIDE AT ANGLE
184146 non-null uint8
62 SDOT_COLDESC_MOTOR VEHICLE STRUCK MOTOR VEHICLE, RIGHT SIDE SIDESWIPE
184146 non-null uint8
63 SDOT_COLDESC_MOTOR VEHICLE STRUCK OBJECT IN ROAD

```

184146 non-null uint8
64 SDOT_COLDESC_MOTOR VEHICLE STRUCK PEDALCYCLIST, FRONT END AT ANGLE
184146 non-null uint8
65 SDOT_COLDESC_MOTOR VEHICLE STRUCK PEDALCYCLIST, LEFT SIDE SIDESWIPE
184146 non-null uint8
66 SDOT_COLDESC_MOTOR VEHICLE STRUCK PEDALCYCLIST, REAR END
184146 non-null uint8
67 SDOT_COLDESC_MOTOR VEHICLE STRUCK PEDALCYCLIST, RIGHT SIDE SIDESWIPE
184146 non-null uint8
68 SDOT_COLDESC_MOTOR VEHICLE STRUCK TRAIN
184146 non-null uint8
69 SDOT_COLDESC_NOT ENOUGH INFORMATION / NOT APPLICABLE
184146 non-null uint8
70 SDOT_COLDESC_PEDALCYCLIST OVERTURNED IN ROAD
184146 non-null uint8
71 SDOT_COLDESC_PEDALCYCLIST RAN OFF ROAD - HIT FIXED OBJECT
184146 non-null uint8
72 SDOT_COLDESC_PEDALCYCLIST STRUCK MOTOR VEHICLE FRONT END AT ANGLE
184146 non-null uint8
73 SDOT_COLDESC_PEDALCYCLIST STRUCK MOTOR VEHICLE LEFT SIDE AT ANGLE
184146 non-null uint8
74 SDOT_COLDESC_PEDALCYCLIST STRUCK MOTOR VEHICLE LEFT SIDE SIDESWIPE
184146 non-null uint8
75 SDOT_COLDESC_PEDALCYCLIST STRUCK MOTOR VEHICLE REAR END
184146 non-null uint8
76 SDOT_COLDESC_PEDALCYCLIST STRUCK MOTOR VEHICLE RIGHT SIDE AT ANGLE
184146 non-null uint8
77 SDOT_COLDESC_PEDALCYCLIST STRUCK MOTOR VEHICLE RIGHT SIDE SIDESWIPE
184146 non-null uint8
78 SDOT_COLDESC_PEDALCYCLIST STRUCK OBJECT IN ROAD
184146 non-null uint8
79 SDOT_COLDESC_PEDALCYCLIST STRUCK PEDALCYCLIST FRONT END AT ANGLE
184146 non-null uint8
80 SDOT_COLDESC_PEDALCYCLIST STRUCK PEDALCYCLIST REAR END
184146 non-null uint8
81 SDOT_COLDESC_PEDALCYCLIST STRUCK PEDESTRIAN
184146 non-null uint8
82 HITPARKEDCAR_Y
184146 non-null uint8
dtypes: float64(2), int64(6), uint8(75)
memory usage: 25.8 MB

```

1.0.4 Modelling

```

[29]: # Assign the data
df=df_dummy
# Set the target for the prediction

```

```

target="SEVERITYCODE"
# Create arrays for the features and the response variable
# set X and y
y = df[target]
X = df.drop(target, axis=1)

# Split the data set into training and testing data sets
# Split the data set into training and testing data sets
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0,
↳test_size=0.33, stratify=y)

```

Selecting the different Algorithms

```

[30]: algo_lst=['Logistic Regression', 'K-Nearest Neighbors', 'Decision Trees', 'Random_
↳Forest']

```

```

[31]: # Initialize an empty list for the accuracy for each algorithm
accuracy_lst=[]

```

1.0.5 Evaluation

Logistic Regression

```

[32]: # Logistic regression
lr = LogisticRegression(solver='lbfgs', max_iter=1000, dual=False).fit(X_test,
↳y_test)
y_pred=lr.predict(X_test)

# Get the accuracy score
acc=accuracy_score(y_test, y_pred)

# Append to the accuracy list
accuracy_lst.append(acc)

print("[Logistic regression algorithm] accuracy_score: {:.3f}.".format(acc))

```

[Logistic regression algorithm] accuracy_score: 0.756.

c:\users\salma\desktop\projects\venv\new\new\lib\site-packages\sklearn\linear_model_logistic.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
```

K-NN Neighbors

```
[33]: # Create a k-NN classifier with 6 neighbors
knn = KNeighborsClassifier(n_neighbors=6)

# Fit the classifier to the data
knn.fit(X_train,y_train)

# Predict the labels for the training data X
y_pred = knn.predict(X_test)

# Get the accuracy score
acc=accuracy_score(y_test, y_pred)

# Append to the accuracy list
accuracy_lst.append(acc)
print('[K-Nearest Neighbors (KNN)] knn.score: {:.3f}.'.format(knn.score(X_test,
→y_test)))
print('[K-Nearest Neighbors (KNN)] accuracy_score: {:.3f}.'.format(acc))
```

```
[33]: KNeighborsClassifier(n_neighbors=6)
```

```
[K-Nearest Neighbors (KNN)] knn.score: 0.739.
```

```
[K-Nearest Neighbors (KNN)] accuracy_score: 0.739.
```

Setting arrays for storing the train and test data accuracies

```
[34]: # Setup arrays to store train and test accuracies
neighbors = np.arange(1, 9)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

# Loop over different values of k
for i, n_neighbor in enumerate(neighbors):
    # Setup a k-NN Classifier with n_neighbor
    knn = KNeighborsClassifier(n_neighbors=n_neighbor)

    # Fit the classifier to the training data
    knn.fit(X_train,y_train)

    #Compute accuracy on the training set
    train_accuracy[i] = knn.score(X_train, y_train)
    #Compute accuracy on the testing set
    test_accuracy[i] = knn.score(X_test, y_test)
```

```
[34]: KNeighborsClassifier(n_neighbors=1)
```

```
[34]: KNeighborsClassifier(n_neighbors=2)
```

```
[34]: KNeighborsClassifier(n_neighbors=3)
```

```
[34]: KNeighborsClassifier(n_neighbors=4)
```

```
[34]: KNeighborsClassifier()
```

```
[34]: KNeighborsClassifier(n_neighbors=6)
```

```
[34]: KNeighborsClassifier(n_neighbors=7)
```

```
[34]: KNeighborsClassifier(n_neighbors=8)
```

Generating a plot for K-NN with varying number of Neighbors

```
[35]: # Generate plot

plt.title('k-NN: Varying Number of Neighbors')
plt.plot(neighbors, test_accuracy, label = 'Testing Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.show()
```

```
[35]: Text(0.5, 1.0, 'k-NN: Varying Number of Neighbors')
```

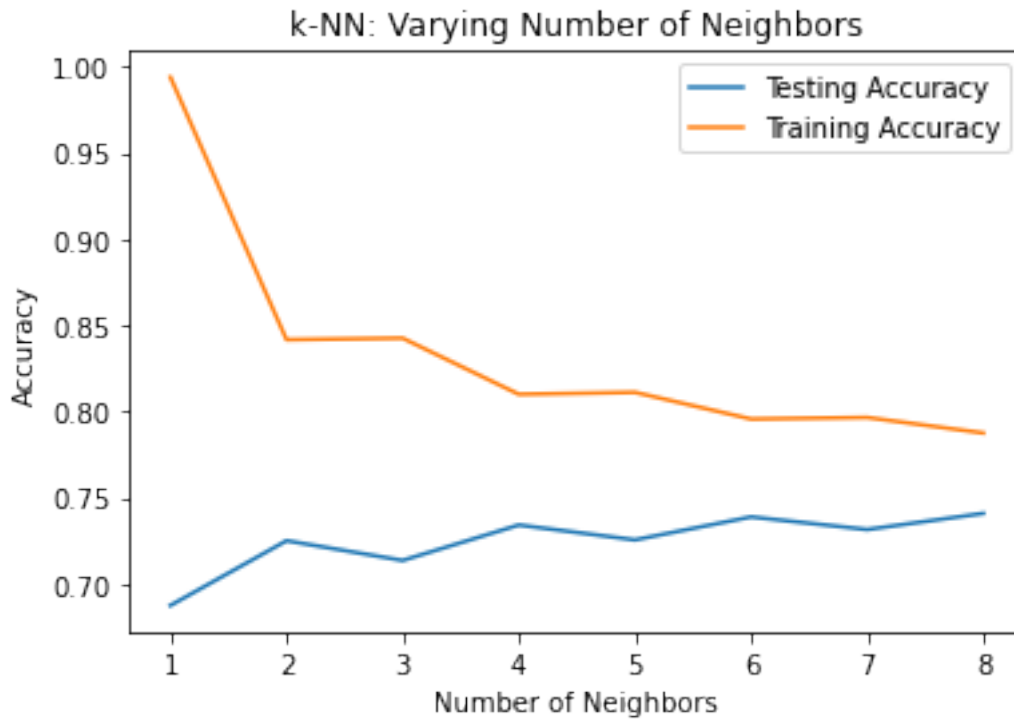
```
[35]: [<matplotlib.lines.Line2D at 0x1e01fd75760>]
```

```
[35]: [<matplotlib.lines.Line2D at 0x1e01fd75970>]
```

```
[35]: <matplotlib.legend.Legend at 0x1e01968b6d0>
```

```
[35]: Text(0.5, 0, 'Number of Neighbors')
```

```
[35]: Text(0, 0.5, 'Accuracy')
```



Decision Tree Algorithm

Instantiate `dt_entropy` & `dt_gini` by setting them as the information criterion

```
[36]: dt_entropy = DecisionTreeClassifier(max_depth=8, criterion='entropy',
    ↪ random_state=1)

# Fit dt_entropy to the training set
dt_entropy.fit(X_train, y_train)

# Use dt_entropy to predict test set labels
y_pred= dt_entropy.predict(X_test)

# Evaluate accuracy_entropy
accuracy_entropy = accuracy_score(y_test, y_pred)

# Print accuracy_entropy
print('[Decision Tree -- entropy] accuracy_score: {:.3f}.'.
    ↪ format(accuracy_entropy))

# Instantiate dt_gini, set 'gini' as the information criterion
dt_gini = DecisionTreeClassifier(max_depth=8, criterion='gini', random_state=1)
```

```

# Fit dt_entropy to the training set
dt_gini.fit(X_train, y_train)

# Use dt_entropy to predict test set labels
y_pred= dt_gini.predict(X_test)

# Evaluate accuracy_entropy
accuracy_gini = accuracy_score(y_test, y_pred)

# Append to the accuracy list
acc=accuracy_gini
accuracy_lst.append(acc)

# Print accuracy_gini
print('[Decision Tree -- gini] accuracy_score: {:.3f}'.format(accuracy_gini))

```

[36]: DecisionTreeClassifier(criterion='entropy', max_depth=8, random_state=1)

[Decision Tree -- entropy] accuracy_score: 0.754.

[36]: DecisionTreeClassifier(max_depth=8, random_state=1)

[Decision Tree -- gini] accuracy_score: 0.754.

Random Forest Algorithm

```

[37]: # Random Forest algorithm

# Create a Gaussian Classifier

clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)

clf.fit(X_train,y_train)
y_pred=clf.predict(X_test)
# Get the accuracy score
acc=accuracy_score(y_test, y_pred)

# Append to the accuracy list
accuracy_lst.append(acc)

# Model Accuracy, how often is the classifier correct?

print("[Random forest algorithm] accuracy_score: {:.3f}.".format(acc))

```

[37]: RandomForestClassifier()

[Random forest algorithm] accuracy_score: 0.736.

Random Forest Classifier

```
[38]: # Create a selector object that will use the random forest classifier to
      ↪ identify

      # features that have an importance of more than 0.03
      sfm = SelectFromModel(clf, threshold=0.03)

      # Train the selector
      sfm.fit(X_train, y_train)

      feat_labels=X.columns

      # Print the names of the most important features
      for feature_list_index in sfm.get_support(indices=True):
          print(feat_labels[feature_list_index])
```

```
[38]: SelectFromModel(estimator=RandomForestClassifier(), threshold=0.03)
```

```
longitude
latitude
PERSONCOUNT
Hour
COLLISIONTYPE_Parked Car
```

Visualizing the important features

```
[39]: feature_imp = pd.Series(clf.feature_importances_, index=X.columns).
      ↪ sort_values(ascending=False)

      # Creating a bar plot, displaying only the top k features
      k=10
      sns.barplot(x=feature_imp[:10], y=feature_imp.index[:k])
      # Add labels to your graph
      plt.xlabel('Feature Importance Score')
      plt.ylabel('Features')
      plt.title("Visualizing Important Features")
      plt.legend()
      plt.show()
```

```
[39]: <AxesSubplot:>
```

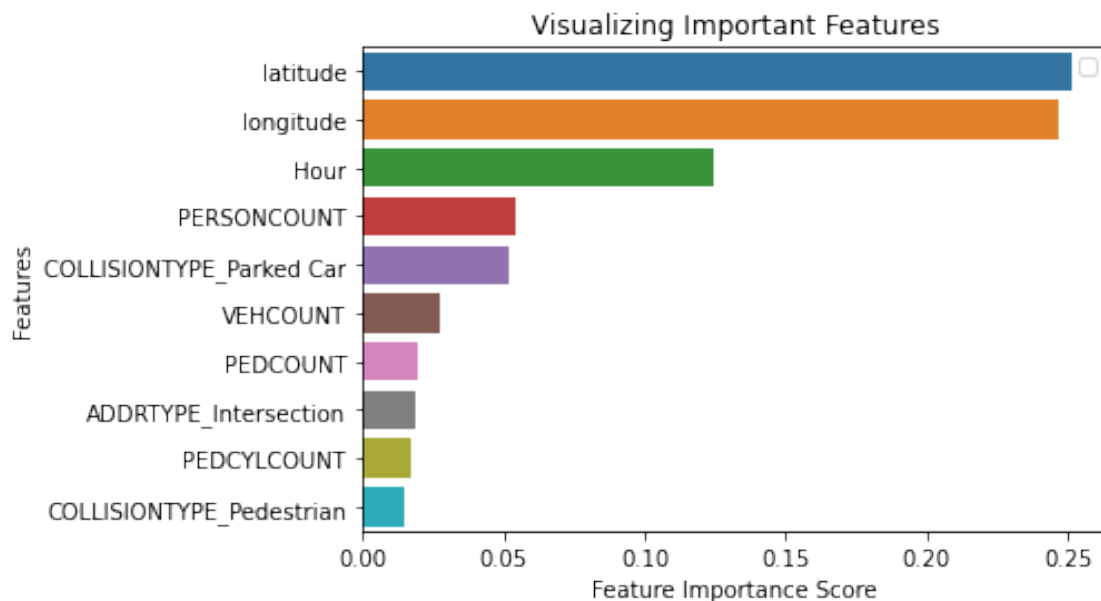
```
[39]: Text(0.5, 0, 'Feature Importance Score')
```

```
[39]: Text(0, 0.5, 'Features')
```

```
[39]: Text(0.5, 1.0, 'Visualizing Important Features')
```


No handles with labels found to put in legend.

[39]: <matplotlib.legend.Legend at 0x1e0194bff40>



Creating a new Random Forest Classifier for the most important features

```
[40]: # Transform the data to create a new dataset containing only the most important
      ↪ features

      # Note: We have to apply the transform to both the training X and test X data.
      X_important_train = sfm.transform(X_train)
      X_important_test = sfm.transform(X_test)

      # Create a new random forest classifier for the most important features
      clf_important = RandomForestClassifier(n_estimators=100, random_state=0,
      ↪ n_jobs=-1)

      # Train the new classifier on the new dataset containing the most important
      ↪ features
      clf_important.fit(X_important_train, y_train)
```

[40]: RandomForestClassifier(n_jobs=-1, random_state=0)

Checking the Accuracy of Random Forest Algorithm with full and the limited features

```
[41]: # Apply The Full Featured Classifier To The Test Data
      y_pred = clf.predict(X_test)
```

```

# View The Accuracy Of Our Full Feature Model
print('[Random forest algorithm -- Full feature] accuracy_score: {:.3f}.'.
      ↪format(accuracy_score(y_test, y_pred)))

# Apply The Full Featured Classifier To The Test Data
y_important_pred = clf_important.predict(X_important_test)

# View The Accuracy Of Our Limited Feature Model
print('[Random forest algorithm -- Limited feature] accuracy_score: {:.3f}.'.
      ↪format(accuracy_score(y_test, y_important_pred)))

```

[Random forest algorithm -- Full feature] accuracy_score: 0.736.
 [Random forest algorithm -- Limited feature] accuracy_score: 0.661.

Making a plot of the accuracy scores for different algorithms

```

[42]: # Make a plot of the accuracy scores for different algorithms

# Generate a list of ticks for y-axis
y_ticks=np.arange(len(algo_lst))

# Combine the list of algorithms and list of accuracy scores into a dataframe,
↪sort the value based on accuracy score
df_acc=pd.DataFrame(list(zip(algo_lst, accuracy_lst)),
↪columns=['Algorithm','Accuracy_Score']).
↪sort_values(by=['Accuracy_Score'],ascending = True)

# Export to a file
df_acc.to_csv('./Accuracy_scores_algorithms_{}.csv', index=False)

# Make a plot
ax=df_acc.plot.barh('Algorithm', 'Accuracy_Score',
↪align='center',legend=False,color='0.5')

# Add the data label on to the plot
for i in ax.patches:
    # get_width pulls left or right; get_y pushes up or down
    ax.text(i.get_width()+0.02, i.get_y()+0.2, str(round(i.get_width(),2)),
    ↪fontsize=10)

# Set the limit, labels, ticks and title
plt.xlim(0,1.1)
plt.xlabel('Accuracy Score')
plt.yticks(y_ticks, df_acc['Algorithm'], rotation=0)
plt.title('Accuracy Score of each Algorithm')

plt.show()

```

```
[42]: Text(0.755802794187826, -0.04999999999999999, '0.74')
```

```
[42]: Text(0.7589458440981421, 0.95, '0.74')
```

```
[42]: Text(0.7740686863367836, 1.95, '0.75')
```

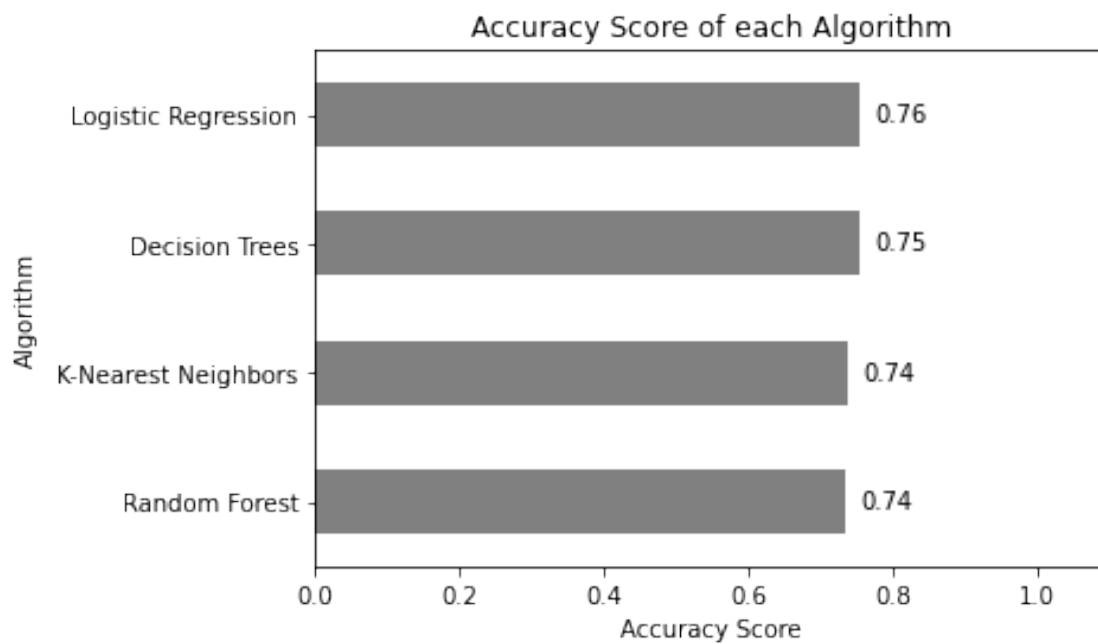
```
[42]: Text(0.7760269216212213, 2.95, '0.76')
```

```
[42]: (0.0, 1.1)
```

```
[42]: Text(0.5, 0, 'Accuracy Score')
```

```
[42]: ([<matplotlib.axis.YTick at 0x1e0115b3610>,
      <matplotlib.axis.YTick at 0x1e011653b20>,
      <matplotlib.axis.YTick at 0x1e01192a4f0>,
      <matplotlib.axis.YTick at 0x1e0118e1a90>],
      [Text(0, 0, 'Random Forest'),
       Text(0, 1, ' K-Nearest Neighbors'),
       Text(0, 2, 'Decision Trees'),
       Text(0, 3, 'Logistic Regression')])
```

```
[42]: Text(0.5, 1.0, 'Accuracy Score of each Algorithm')
```



1.0.6 Deployment

For the deployment phase as it can vary from project to project a simple pdf report has been generated.

1.0.7 Summary

- Seattle road accidents data has been analyzed in order to get useful insights.
- The data contains multiple attributes e.g. accident severity, collision type, coordinates of the incident, date and time of the incident, weather and road conditions, address types, no of persons injured and property damage and many other attributes.
- There are two accident severity types mentioned in the dataset i.e.
 - Property damage only collision(1)
 - Injury collision(2)
- All the mandatory Cross-industry standard process for data mining CRISP-DM phases are covered in this report which contains the following:
 - Business Understanding
 - Data Understanding
 - Data Preparation
 - Modeling
 - Evaluation
 - Deployment
- In the Modeling phase, four algorithms were selected where the target class was “accident severity”.
- Based on the predictions, “Logistic Regression” relatively performed better among the others having the accuracy percentage of approx.76%.

1.0.8 Conclusion

Based on the selected dataset(features) for this capstone project which includes mainly, coordinates, hour, person count and the collision type, it can be concluded that these particular classes have a somewhat impact on whether or not travelling along the Seattle roads could result in property damage (class 1) or injury (class 2). In this study, the technique of association rules with a large set of accident data to identify the reasons of road accidents were used. The results show that this model could provide good predictions against traffic accident with approx. 76% correct rate. It should be noted that due to the constraints of data and research condition, there are still some factors, such as engine capacity, traffic flows, gender, age of the driver, attaining the missing data etc. that are not considered in this model and can be taken into account for future study. The results of this study can be used in vehicle safety assistance driving and provide early warnings and proposals for safe driving, hence help in reducing the number of accidents.