US Accidents Dataset - Group 4

By: Harsh Tandon, Stuti Sanghavi

Part 1 - Dataset Description

Description

- This is a countrywide traffic accident dataset, which covers 49 states of the United States. The dataset we used contains data from February 2016 to December 2019.
- The data is continuously being collected from February 2016, using several data providers, including two APIs which provide streaming traffic event data. These APIs broadcast traffic events captured by a variety of entities, such as the US and state departments of transportation, law enforcement agencies, traffic cameras, and traffic sensors within the road-networks.
- The dataset contains around 3.0 million accident records and 48 columns.
- Each row represents one accident.
- And for each accident, some of the variables recorded in the dataset are **location** where the accident occured (columns related to location: Start_Lat, Start_Lng, End_Lat, End_Lng, Number, Street, Zipcode, City, State); the **infrastructure** where accident occured (crossing, traffic junction, bump, roundabout etc) ; **the weather conditions** (columns capturing that are: Humidity, Precipitation, Visibility, Temperature) etc.
- Our goal is to find if there are any specific trends/ patterns causing accidents which can be used to advise people to take precaution.

More detailed information about what each column means is mentioned below:

Dataset link : [https://www.kaggle.com/sobhanmoosavi/us-accidents (https://www.kaggle.com/sobhanmoosavi/us-accidents)](https://www.kaggle.com/sobhanmoosavi/us-accidents)

Description of each column

- ID: This is a unique identifier of the accident record.
- Source: Indicates source of the accident report (i.e. the API which reported the accident.).
- TMC: A traffic accident may have a Traffic Message Channel (TMC) code which provides more detailed description of the event.
- Severity: Shows the severity of the accident. 4 being the most severe
- Start_Time: Shows start time of the accident in local time zone.
- End_Time: Shows end time of the accident in local time zone.
- Start_Lat: Shows latitude in GPS coordinate of the start point.
- Start_Lng : Shows longitude in GPS coordinate of the start point.
- End_Lat: Shows latitude in GPS coordinate of the end point.
- End_Lng: Shows longitude in GPS coordinate of the end point.
- Distance(mi): The length of the road extent affected by the accident.
- Description: Shows natural language description of the accident.
- Number: Shows the street number in address field.
- Street: Shows the street name in address field.
- Side: Shows the relative side of the street (Right/Left) in address field.
- City: Shows the city in address field.
- County: Shows the county in address field.
- State: Shows the state in address field.
- Zipcode: Shows the zipcode in address field.
- Country: Shows the country in address field.
- Timezone: Shows timezone based on the location of the accident (eastern, central, etc.).
- Airport_Code: Denotes an airport-based weather station which is the closest one to location of the accident.
- Weather_Timestamp: Shows the time-stamp of weather observation record (in local time).
- Temperature(F): Shows the temperature (in Fahrenheit).
- Wind_Chill(F): Shows the wind chill (in Fahrenheit).
- Humidity(%): Shows the humidity (in percentage).
- Pressure(in): Shows the air pressure (in inches).
- Visibility(mi): Shows visibility (in miles).
- Wind_Direction: Shows wind direction.
- Wind_Speed(mph): Shows wind speed (in miles per hour).
- Precipitation(in): Shows precipitation amount in inches, if there is any.
- Weather_Condition: Shows the weather condition (rain, snow, thunderstorm, fog, etc.).
- Amenity: A Point-Of-Interest (POI) annotation which indicates presence of amenity in a nearby location.
- Bump: A POI annotation which indicates presence of speed bump or hump in a nearby location.
- Crossing: A POI annotation which indicates presence of crossing in a nearby location.
- Give_Way: A POI annotation which indicates presence of give_way sign in a nearby location.
- Junction: A POI annotation which indicates presence of junction in a nearby location.
- No_Exit: A POI annotation which indicates presence of no_exit sign in a nearby location.
- Railway: A POI annotation which indicates presence of railway in a nearby location.
- Roundabout: A POI annotation which indicates presence of roundabout in a nearby location.
- Station: A POI annotation which indicates presence of station (bus, train, etc.) in a nearby location.
- Stop: A POI annotation which indicates presence of stop sign in a nearby location.

- Traffic_Calming: A POI annotation which indicates presence of traffic_calming means in a nearby location.
- Traffic_Signal: A POI annotation which indicates presence of traffic_signal in a nearby location.
- Turning_Loop: A POI annotation which indicates presence of turning_loop in a nearby location.
- Sunrise_Sunset: Shows the period of day (i.e. day or night) based on sunrise/sunset.
- Civil_Twilight: Shows the period of day (i.e. day or night) based on civil twilight.
- Nautical_Twilight: Shows the period of day (i.e. day or night) based on nautical twilight.
- Astronomical_Twilight: Shows the period of day (i.e. day or night) based on astronomical twilight.

Part 2 - Data Exploration, Cleaning and Preperation

In [1]:

```
# print all the outputs in a cell
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [2]:

```
# importing the necessary packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
import sklearn.tree as tree
from IPython.display import Image
import pydotplus
import chart_studio.plotly as py
from plotly.graph_objs import *
pd.set_option('display.max_columns',100)
```

a) Data Exploration

In [3]:

```
# importing the datafile
df = pd.read_csv("US_Accidents.csv", index_col=0, parse_dates=False)
```

In [4]:

```
#Checking the number of rows and columns in the dataset
len(df)
df.shape
```

Out[4]:

```
2974335
```

Out[4]:

```
(2974335, 48)
```

In [5]:

```
#Looking at the columns and values that each rows consist (categorical/numerical)
df.head(1)
```

Out[5]:

	Source	TMC	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	End_Lat	End_L
ID									
A-1	MapQuest	201.0	3	2016-02-08 05:46:00	2016-02-08 11:00:00	39.865147	-84.058723	NaN	N

Check for missing values

In [6]:

```
# Checking the total NaN's in the dataset
df.isna().sum().sum()
```

Out[6]:

11817022

We have around 12 million NA values. Lets see where do these NA values occur.

In [7]:

```
# Checking the number of Nan's for each column
df.isna().sum().sort_values(ascending = False)[:24]
```

Out[7]:

```
End_Lat                  2246264
End_Lng                  2246264
Precipitation(in)        1998358
Number                   1917605
Wind_Chill(F)            1852623
TMC                       728071
Wind_Speed(mph)           440840
Weather_Condition          65932
Visibility(mi)             65691
Humidity(%)                59173
Temperature(F)             56063
Pressure(in)               48142
Wind_Direction             45101
Weather_Timestamp          36705
Airport_Code                5691
Timezone                    3163
Zipcode                      880
Nautical_Twilight             93
Astronomical_Twilight         93
Civil_Twilight                93
Sunrise_Sunset                93
City                          83
Description                    1
Side                           0
dtype: int64
```

In [8]:

```
# Calculating the percentage of Nan's in each columns
((round(df.isna().sum()/len(df),2)*100).sort_values(ascending = False))[:15]
```

Out[8]:

```
End_Lat              76.0
End_Lng              76.0
Precipitation(in)    67.0
Number               64.0
Wind_Chill(F)        62.0
TMC                  24.0
Wind_Speed(mph)      15.0
Temperature(F)        2.0
Humidity(%)           2.0
Pressure(in)          2.0
Visibility(mi)        2.0
Wind_Direction        2.0
Weather_Condition     2.0
Weather_Timestamp     1.0
County                0.0
dtype: float64
```

In [9]:

```
#Checking to see if the precipitation information is captured in Weather Condition colu
mns
df[(df['Precipitation(in)'] == 0.000000) & (df.Weather_Condition.str.contains('Rain'))]
.Weather_Condition.unique()
len(df[(df['Precipitation(in)'] == 0.000000) & (df.Weather_Condition.str.contains('Rai
n'))])
```

Out[9]:

```
array(['Light Rain', 'Rain', 'Light Rain / Windy', 'Light Rain Shower',
       'Light Thunderstorms and Rain', 'Light Rain with Thunder',
       'Thunderstorms and Rain', 'Heavy Thunderstorms and Rain',
       'Light Freezing Rain', 'Heavy Rain', 'Light Rain Showers',
       'Freezing Rain', 'Light Freezing Rain / Windy', 'Rain / Windy',
       'Heavy Rain / Windy', 'Light Rain Shower / Windy', 'Rain Shower'],
      dtype=object)
```

Out[9]:

```
49381
```

b) Data Cleaning

Dropping Unnecessary Columns:

1. End Latitude and Longitude columns doesn't capture much information as starting and ending point of accidents are the same in most the cases and the starting point of accident is captured by Start Latitude and Longitude columns. Because of this, End_Lat and End_Lng columns become redundant and also we see around 76% values are missing.
2. Even when Precipitation is 0, Weather condition shows there was rain. We'll drop precipitation column because it contains repeated information and 67% values are missing.
3. Wind Chill (F) column has around 62% missing values and we're not using this variable in our analysis.
4. We don't need Wind Speed and Wind Direction in our analysis.
5. TMC doesn't add any value to our analysis.
6. Number shows on which street number the accident occured. It doesn't add much value (since we already have many other variables to help identify exact location of the accident.
7. Zipcode captures repeated information captured by other geographic variables.
8. Weather_Timestamp captures the time when weather for the accident was recorded. It is very close to the start_time of the accident recorded, so we don't required this information.
9. Source shows where does the data come from. We have no use of it.

In [10]:

```
# Dropping columns because of the reason mentioned above
df.drop(['End_Lat','End_Lng','Precipitation(in)','Wind_Chill(F)','Wind_Speed(mph)','Win
d_Direction','TMC','Number'\
        ,'Zipcode','Weather_Timestamp','Source'], axis = 1, inplace = True)
```

In [11]:

```
#Checking if the columns are removed
df.head(1)
```

Out[11]:

	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	Distance(mi)	Description	Street
ID								
A-1	3	2016-02-08 05:46:00	2016-02-08 11:00:00	39.865147	-84.058723	0.01	Right lane blocked due to accident on I-70 Eas...	I-70 E

In [12]:

```
#Checking how many NA's are still left
df.isna().sum().sort_values(ascending = False)[:16]
```

Out[12]:

```
Weather_Condition        65932
Visibility(mi)           65691
Humidity(%)              59173
Temperature(F)           56063
Pressure(in)             48142
Airport_Code              5691
Timezone                  3163
Nautical_Twilight           93
Astronomical_Twilight       93
Civil_Twilight              93
Sunrise_Sunset              93
City                        83
Description                  1
Country                      0
Start_Time                   0
End_Time                     0
dtype: int64
```

Dealing with the remaining Nan's

- **Using Fill Na's**

In [13]:

```
# Filling the Nan's in Weather condition column with Unknows
df.Weather_Condition.fillna('Unknown', inplace = True)
```

In [14]:

```
df.Side.replace(" ",'L', inplace = True)
```

In [15]:

```
# Calculating the mean values of these columns, to use it to fill the Nan's
meanVis = df['Visibility(mi)'].mean()
meanHum = df['Humidity(%)'].mean()
meanTemp = df['Temperature(F)'].mean()
meanPre = df['Pressure(in)'].mean()
```

In [16]:

```
df['Visibility(mi)'].fillna(value = meanVis, inplace = True)
df['Humidity(%)'].fillna(value = meanHum, inplace = True)
df['Temperature(F)'].fillna(value = meanTemp, inplace = True)
df['Pressure(in)'].fillna(value = meanPre, inplace = True)
```

- **Using drop na's**

In [17]:

```
# Dropping the rows which have very few Nan's
df.dropna(how = 'any', subset = ['Airport_Code','Timezone','Nautical_Twilight','Astrono
mical_Twilight','Civil_Twilight'\
                                 ,'Sunrise_Sunset','City','Description'], inplace = Tru
e)
```

In [18]:

```
#Checking to see if there are any Nan's in the dataset
df.isna().sum().sum()
```

Out[18]:

```
0
```

Our dataset had 2974335 rows. After dropping rows containing NA values, we're down to 2968550 rows. We only lost 0.19% of 3 million records!

In [19]:

```
# Checking the number of rows and columns post cleaning the dataset - Lost only 0.19% o
f the data and around 11 columns
len(df)
df.shape
```

Out[19]:

```
2968550
```

Out[19]:

```
(2968550, 37)
```

c) Data Preperation

Converting Start_Time and End_Time attribute to date_time and then extracting information in different columns

In [20]:

```
df['Start_Time'] = pd.to_datetime(df['Start_Time'])
df['End_Time'] = pd.to_datetime(df['End_Time'])
```

In [21]:

```
df['Hour'] = df.Start_Time.dt.hour
df['Day'] = df.Start_Time.dt.day
df['Day_of_week'] = df.Start_Time.dt.day_name()
df['Month'] = df.Start_Time.dt.month_name()
df['Year'] = df.Start_Time.dt.year
```

In [22]:

```
# After extracting the relevant information, we remove these columns
df.drop(['End_Time'], axis = 1, inplace = True)
#We wont drop Start_Time yet because we want it for Time Series Analysis at the end.
```

All streets that start with the letter 'I' are interstate highways. So let's annotate them as 'Interstate' and the rest as 'Others'.

In [23]:

```
# Creating a new column Road_type to differentiate between Interstate and Other highway
s
df['Road_type'] = df.Street.apply(lambda x: 'Interstate' if x.startswith('I-') else 'Ot
hers')
```

In [24]:

```
# Checking if the necessary changes were made
df.head(1)
```

Out[24]:

	Severity	Start_Time	Start_Lat	Start_Lng	Distance(mi)	Description	Street	Side	Ci
ID									
A-1	3	2016-02-08 05:46:00	39.865147	-84.058723	0.01	Right lane blocked due to accident on I-70 Eas...	I-70 E	R	Dayt

d) Exploratory Data Analysis

1. Looking at the number of accidents by severity level

- We see that most accidents occur with Severity level of 2 and 3, and very few accidents very high or very low severity level

In [25]:

```
#Grouping by severity level
df.groupby('Severity').size()
```

Out[25]:

```
Severity
1        968
2    1989928
3     885636
4      92018
dtype: int64
```

In [26]:

```
# Plotting the results from above
df.groupby('Severity').size().plot(kind = 'bar')
plt.title('Accidents Grouped by Severity')
plt.ylabel('Count')
```

Out[26]:

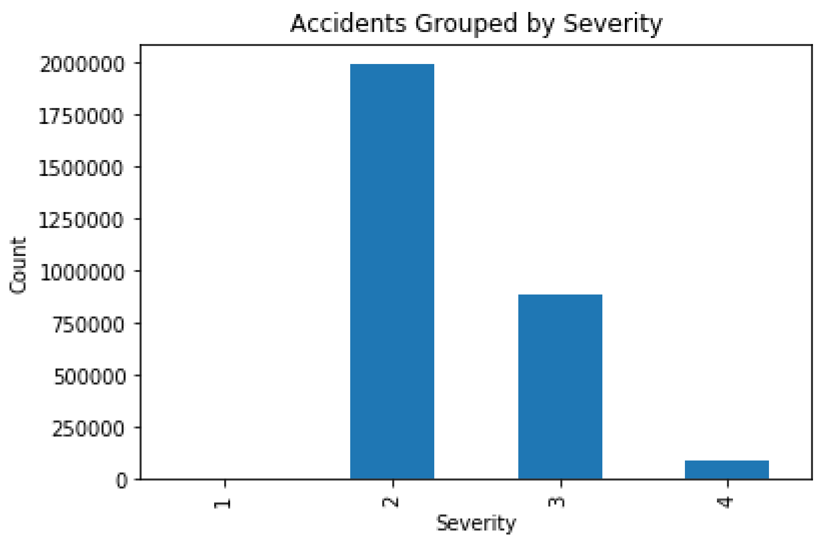
```
<matplotlib.axes._subplots.AxesSubplot at 0x18d3ae78ac8>
```

Out[26]:

```
Text(0.5, 1.0, 'Accidents Grouped by Severity')
```

Out[26]:

```
Text(0, 0.5, 'Count')
```



2. Number of accidents grouped by month:

- Most accidents occur in holiday season!

In [27]:

```
# Grouping the number of accidents by month
ByMonth = df.groupby('Month').size()
ByMonth.nlargest()
```

Out[27]:

```
Month
October      323865
December     298955
November     298482
September    291859
August       288261
dtype: int64
```

In [28]:

```
months = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'Se
ptember', 'October', 'November', 'December']

#Reindexing the months in order as above and plotting the results
ds1 = ByMonth.reindex(months, axis=0)
ds1.plot(kind = 'bar')
plt.title('Accidents Grouped by Months')
plt.ylabel('Count')
```

Out[28]:

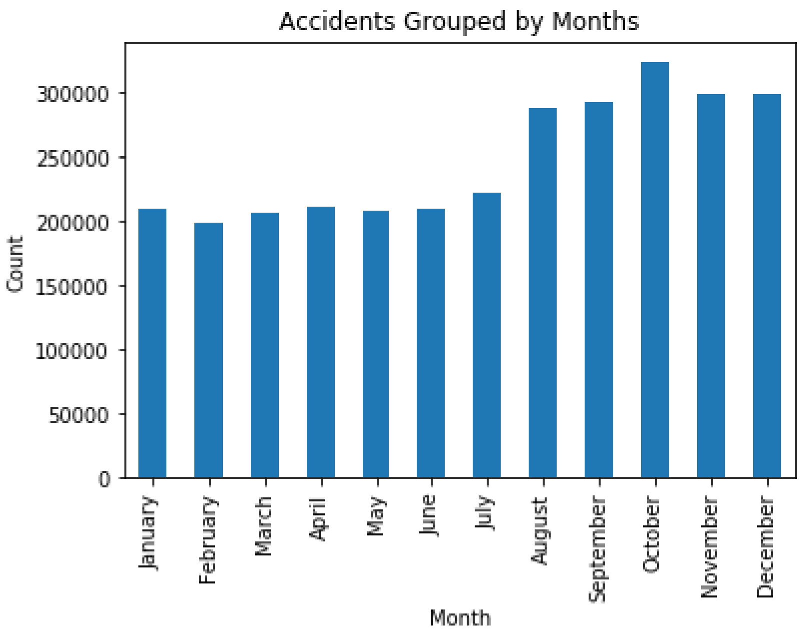
```
<matplotlib.axes._subplots.AxesSubplot at 0x18cc341e908>
```

Out[28]:

```
Text(0.5, 1.0, 'Accidents Grouped by Months')
```

Out[28]:

```
Text(0, 0.5, 'Count')
```



3. Proportion of Accidents Grouped by Hour of the Day:

- We see that majority of the accidents occuring during weekdays are during peak hours (morning and evening time) and during weekends during morning and afternoon, when people generally step out of their homes

In [29]:

```
#Finding proportion of accidents for each hour on weekends
weekend_hr = df[(df.Day_of_week == 'Saturday') | (df.Day_of_week == 'Sunday')].groupby(
'Hour').size()/\
len(df[(df.Day_of_week == 'Saturday') | (df.Day_of_week == 'Sunday')])
```

In [30]:

```
#Finding proportion of accidents for each hour on weekdays
weekday_hr = df[(df.Day_of_week != 'Saturday') & (df.Day_of_week != 'Sunday')].groupby(
'Hour').size()/\
len(df[(df.Day_of_week != 'Saturday') & (df.Day_of_week != 'Sunday')])
```

In [31]:

```
# Concating the results from above to get a line graph in the same graph
plot = pd.concat([weekend_hr,weekday_hr],axis=1).rename(columns={0:'weekends', 1:'weekd
ays'}).plot()
plt.title("Proportion of accidents by hour of the day for weekdays and weekends")
plt.xlabel("Hour of the day")
plt.ylabel("Proportion of accidents")
```

Out[31]:

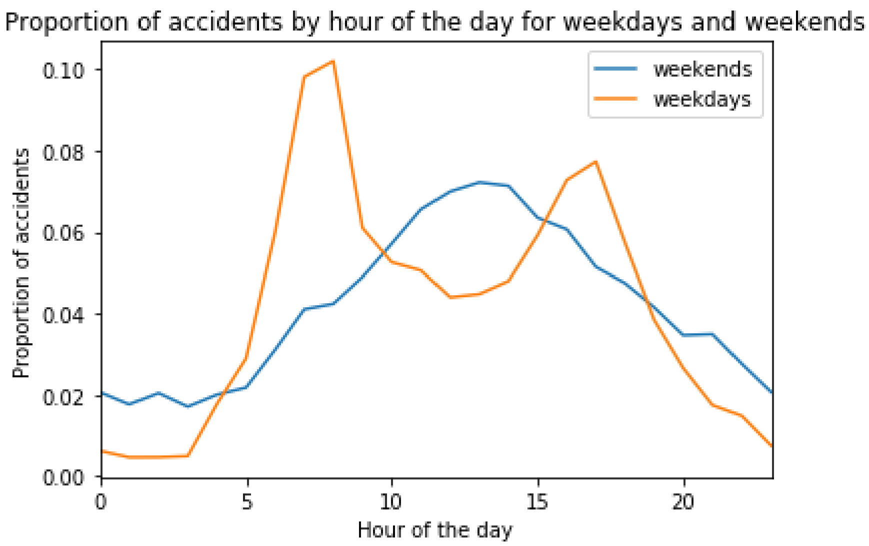
```
Text(0.5, 1.0, 'Proportion of accidents by hour of the day for weekdays an
d weekends')
```

Out[31]:

```
Text(0.5, 0, 'Hour of the day')
```

Out[31]:

```
Text(0, 0.5, 'Proportion of accidents')
```



Part 3 - Findings

Finding 1: Even though the left lane on a road is supposed to have vehicles with higher speed, we see that majority of the accidents occuring on interstate highways are on right side of the road with a higher severity level as compared to other road types

In [32]:

```
# Making sure we have only Interstate and Others listed as Road- Type
df.Road_type.unique()
```

Out[32]:

```
array(['Interstate', 'Others'], dtype=object)
```

In [33]:

```
# Finding out the number of accidents for each road-type
df_interstate = pd.DataFrame(df.groupby('Road_type').size()).reset_index()
df_interstate.rename(columns = {0:'total_count'}, inplace = True)
```

In [34]:

```
df_interstate
```

Out[34]:

	Road_type	total_count
0	Interstate	683937
1	Others	2284613

1 a) - Now we wanna see if the side of the road has anything to do with the number of accidents.

Majority of the accidents occuring on Interstate highway occur on the right side of the road

In [35]:

```
# Finding out the number of accidents for each side of each road type
df_road_side = pd.DataFrame(df.groupby(['Road_type','Side']).size()).reset_index()
df_road_side.rename(columns = {0:'counts'}, inplace = True)
```

In [36]:

```
df_road_side
```

Out[36]:

	Road_type	Side	counts
0	Interstate	L	3
1	Interstate	R	683934
2	Others	L	535254
3	Others	R	1749359

In [37]:

```
# Merging the two dataframes from above to calculate the % accidents on each side of th
e road
df_merge_interstate = df_road_side.merge(df_interstate)
df_merge_interstate['%_side'] = df_merge_interstate['counts']/df_merge_interstate['tota
l_count']*100
df_merge_interstate
```

Out[37]:

	Road_type	Side	counts	total_count	%_side
0	Interstate	L	3	683937	0.000439
1	Interstate	R	683934	683937	99.999561
2	Others	L	535254	2284613	23.428651
3	Others	R	1749359	2284613	76.571349

In [38]:

```
#Plotting the above results
sns.catplot(x='Road_type', y='%_side', hue= 'Side', data=df_merge_interstate[df_merge_i
nterstate.Side != " "]\
                                                                                , kind =
'bar')
plt.xlabel('Interstate vs Other Roads')
plt.ylabel('Percentage of accidents by side')
plt.title('% accidents for Interstate vs Others Roads by side of the road')
```

Out[38]:

```
<seaborn.axisgrid.FacetGrid at 0x18cc34f2048>
```

Out[38]:

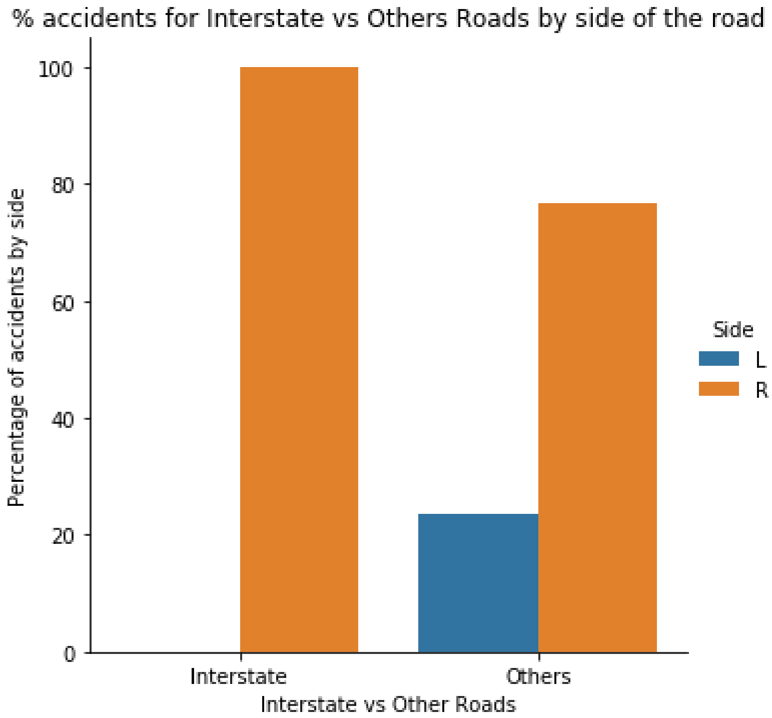
```
Text(0.5, 21.706250000000002, 'Interstate vs Other Roads')
```

Out[38]:

```
Text(27.849433593750014, 0.5, 'Percentage of accidents by side')
```

Out[38]:

```
Text(0.5, 1, '% accidents for Interstate vs Others Roads by side of the ro
ad')
```



1 b) - Looking deeper at severity of accidents for the above results:

Majority of the accidents occuring on Interstate highways are of higher severity level than those occuring on the other roads

In [39]:

```
#Finding out the number of accidents for each severity level
df_interstate_severity = pd.DataFrame(df.groupby(['Road_type','Severity']).size()).rese
t_index()
df_interstate_severity.rename(columns = {0:'counts'}, inplace = True)
```

In [40]:

```
df_interstate_severity
```

Out[40]:

	Road_type	Severity	counts
0	Interstate	1	21
1	Interstate	2	238246
2	Interstate	3	423675
3	Interstate	4	21995
4	Others	1	947
5	Others	2	1751682
6	Others	3	461961
7	Others	4	70023

In [41]:

```
# Getting the number of accidents for each Road_type
df_interstate = pd.DataFrame(df.groupby('Road_type').size()).reset_index()
```

In [42]:

```
df_interstate.rename(columns = {0:'total_for_interstate'}, inplace = True)
```

In [43]:

```
df_interstate
```

Out[43]:

	Road_type	total_for_interstate
0	Interstate	683937
1	Others	2284613

In [44]:

```
# merging the above two dataset to get the percentage accidents by severity
df_merge_severity = df_interstate_severity.merge(df_interstate)
df_merge_severity['pct_acc_by_severity'] = df_merge_severity['counts']/df_merge_severit
y['total_for_interstate']*100
df_merge_severity
```

Out[44]:

	Road_type	Severity	counts	total_for_interstate	pct_acc_by_severity
0	Interstate	1	21	683937	0.003070
1	Interstate	2	238246	683937	34.834495
2	Interstate	3	423675	683937	61.946495
3	Interstate	4	21995	683937	3.215939
4	Others	1	947	2284613	0.041451
5	Others	2	1751682	2284613	76.673030
6	Others	3	461961	2284613	20.220536
7	Others	4	70023	2284613	3.064983

In [45]:

```
# Plotting the above results
sns.catplot(x='Road_type', y='pct_acc_by_severity', hue= 'Severity', data=df_merge_seve
rity, kind ='bar')
plt.xlabel("Interstate vs Others")
plt.ylabel("% accident by severity")
plt.title("% of accidents by severity for road type")
```

Out[45]:

```
<seaborn.axisgrid.FacetGrid at 0x18cc3538e48>
```

Out[45]:

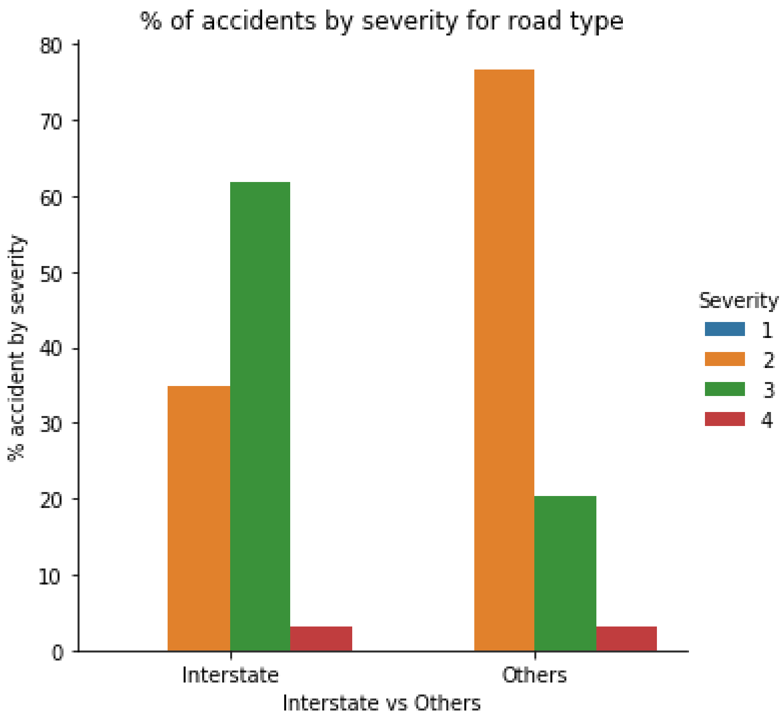
```
Text(0.5, 21.706249999999983, 'Interstate vs Others')
```

Out[45]:

```
Text(27.89207682291667, 0.5, '% accident by severity')
```

Out[45]:

```
Text(0.5, 1, '% of accidents by severity for road type')
```



Managerial Insight
This finding explains that even though the traffic is slower on right side, accidents might be occuring because of vehicles entering/exiting the highway. Using this, we suggest taking extra precaution while doing that and there could be extra infrastructure improvements such as a longer entry and exit lane, allowing drivers enough time to know thier surroundings and facilitate a smooth entry/exit.

Finding 2: South Carolina has the higest accident rate and Maryland has the most severe ones.

Grouped by City: *Top 3 accident affected cities are in Texas and only 1 in California!*

In [46]:

```
# number of accidents grouped by city
df.groupby('City').size().nlargest()
```

Out[46]:

```
City
Houston        93288
Charlotte      68054
Los Angeles    65851
Austin         58703
Dallas         58036
dtype: int64
```

Grouped by State: *Even though the accidents are highest in 3 'cities' from Texas, we see that in accidents by States, accidents are highest in CA and then Texas. This could be attributed to the fact that Texas in nearly twice as big as California, and yet the population is approximately 11 million lesser than that of California.*
In other words, population in more spread out in Texas than in California, and thus State level accidents are lesser in number.

In [47]:

```
#number of accidents grouped by state
df.groupby('State').size().nlargest()
```

Out[47]:

```
State
CA    662985
TX    298036
FL    223403
SC    145297
NC    142453
dtype: int64
```

Merge Population Dataset

Number of accidents seem to depend on population of the state as well. Lets merge a population dataset!

In [48]:

```
# Reading in the csv file
popdf = pd.read_csv('State Populations.csv')
popdf.rename({'2018 Population':'Population'}, axis = 1, inplace = True)
```

In [49]:

```
# 1 row represents population for each state
popdf.head()
```

Out[49]:

	State	Population
0	CA	39776830
1	TX	28704330
2	FL	21312211
3	NY	19862512
4	PA	12823989

In [50]:

```
# Finding the number of accidents for each state
State_df = df.State.value_counts().reset_index()
State_df.rename(columns={'State':'Num_of_Accidents','index':'State'},inplace=True)
```

In [51]:

```
State_df.head(1)
```

Out[51]:

	State	Num_of_Accidents
0	CA	662985

In [52]:

```
# merging this with the population data
State_df = State_df.merge(popdf, how = 'right')
```

In [53]:

```
# View of the merged data
State_df.head(1)
```

Out[53]:

	State	Num_of_Accidents	Population
0	CA	662985.0	39776830

Lets normalize number of accidents in each state by that state's population

In [54]:

```
# Getting the rate of accident per person
State_df['Accident_Rate'] = (State_df.Num_of_Accidents/State_df.Population * 100)
```

2 a) - After normalizing the data with population for each state, we see which state has the highest accident rate.

We see that California is no longer at the top of the list!
In fact, South Carolina has a higher accident rate, even though the total number of accidents is lesser.
Even Orlando comes above California.

In [55]:

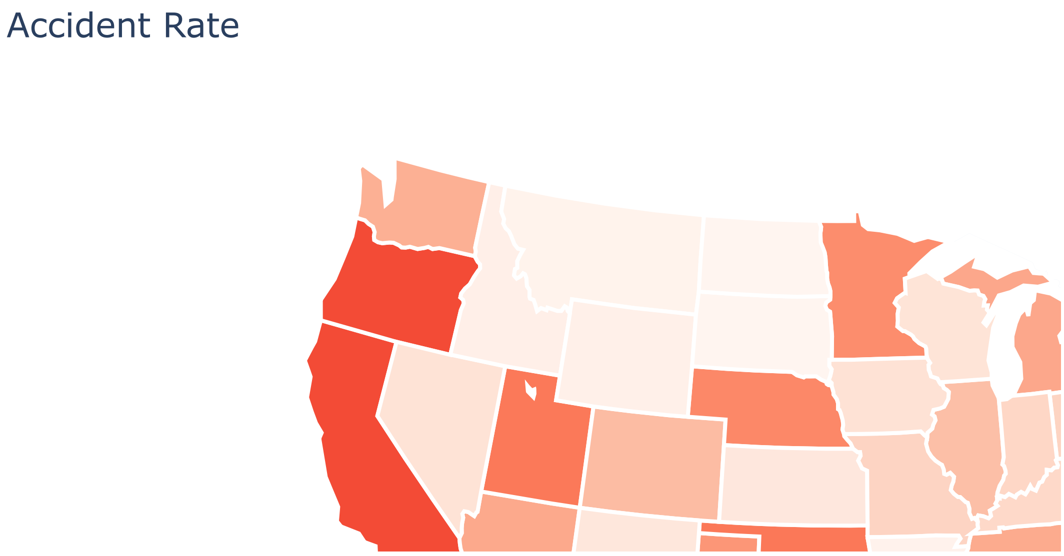
```
# Top 5 states with highest number of accidents per person
accident_rate_by_state = State_df.sort_values(by = 'Accident_Rate', ascending = False)
accident_rate_by_state.head()
```

Out[55]:

	State	Num_of_Accidents	Population	Accident_Rate
3	SC	145297.0	5088916	2.855166
11	OR	70063.0	4199563	1.668340
0	CA	662985.0	39776830	1.666762
4	NC	142453.0	10390149	1.371039
18	OK	51297.0	3940521	1.301782

In [56]:

```
import plotly.graph_objects as go


fig = go.Figure(data=go.Choropleth(
    locations=State_df['State'],
    z = State_df['Accident_Rate'].astype(float),
    locationmode = 'USA-states',
    text = State_df['State'],
    colorscale = 'Reds',
    colorbar_title = "Count Accidents",
    marker = dict(
            line = dict (
                color = 'rgb(255,255,255)',
                width = 2
            ) ),
))

fig.update_layout(
    title_text = 'Accident Rate',
    geo_scope='usa',
)
```



2 b) For each severity level, which state has the highest accident rate

In the above finding we don't know how severe the accident was. Lets look into severity.

In [57]:

```
# For each severity level and state, looking at the number of accidents
State_Severity_df = df.groupby(['Severity','State']).size().reset_index()
State_Severity_df = State_Severity_df.rename(columns = {0:'Number_of_accidents'})
```

In [58]:

```
State_Severity_df.head(1)
```

Out[58]:

	Severity	State	Number_of_accidents
0	1	AL	18

In [59]:

```
# Merging this with population dataset
State_Severity_df = State_Severity_df.merge(popdf, how='inner')
```

In [60]:

```
State_Severity_df.head(1)
```

Out[60]:

	Severity	State	Number_of_accidents	Population
0	1	AL	18	4888949

In [61]:

```
# Creating a new column to calculate the per person accident rate for each severity lev
el
State_Severity_df['Accident_Rate'] = (State_Severity_df.Number_of_accidents/State_Sever
ity_df.Population * 100)
```

In the previous finding of 2a, we saw that South Carolina topped the list with the highest accident rate in a state. Though the accidents per person is highest in South Carolina with accidents of severity level 1, 2 and 3 we also see that the most severe accidents (Severity = 4) happen in Maryland!

In [62]:

```
# Looking at accident rate for each severity level
State_Severity_df.groupby('Severity').apply(lambda x: x[x.Accident_Rate == x.Accident_R
ate.max()]).sort_index(ascending = False)
```

Out[62]:

		Severity	State	Number_of_accidents	Population	Accident_Rate
Severity						
4	**63**	4	MD	4652	6079602	0.076518
3	**130**	3	SC	31096	5088916	0.611054
2	**129**	2	SC	113222	5088916	2.224875
1	**128**	1	SC	40	5088916	0.000786

Managerial Suggestion

This finding explains that even though the accident rate is low in Maryland, the ones which do happen, are most severe. However, the accident rate is highest in South Carolina for severity level of 1, 2 and 3, indicating a lot of accidents occuring in the state of SC. We could use this finding to advise the emergency services to be prepared for frequent and severe accidents, which in turn could save lives.

Finding 3: Even though Missisippi has higher % of people drinking, the rate of accident is more than 100% lower than states having much stringent laws like Utah or Tennese.

In [63]:

```
# Looking at the accident rate for each state
State_df.sort_values(by = 'Accident_Rate', ascending = False).head()
```

Out[63]:

	State	Num_of_Accidents	Population	Accident_Rate
3	SC	145297.0	5088916	2.855166
11	OR	70063.0	4199563	1.668340
0	CA	662985.0	39776830	1.666762
4	NC	142453.0	10390149	1.371039
18	OK	51297.0	3940521	1.301782

Did you know Mississippi is the only state in the country to allow an open container of alcohol to be present while driving.

Riding In Cars With Beers

by The Awl · April 28, 2016

by Owen Phillips

TitlePhoto

If you ask someone from Mississippi how long it takes to drive from Jackson, the capital, down to the Gulf Coast they might tell you "about six beers." Walk into most gas stations there and you'll find waist-high barrels filled with tallboys (sixteen to twenty-four ounces) and hog legs (thirty-two ounces) covered in ice. The beers are located near the door, sold individually, and placed in brown paper bags for a reason. Mississippi is the only state that doesn't have an open-container law that prohibits drivers or passengers from drinking inside a motor vehicle. And while some counties

After we came to know about the law mentioned above, it would be quite interesting to see if it has any effect on accident rate.

We know rate of accident for each state, let's look into what role drinking plays into accident rate. Let's merge dataset about % of people drinking in each state obtained from CDC website. [https://www.cdc.gov/alcohol/data-stats.htm (https://www.cdc.gov/alcohol/data-stats.htm)](https://www.cdc.gov/alcohol/data-stats.htm)

In [64]:

```
# Reading in the file
drunkState = pd.read_csv('DrunkState.csv')
drunkState.rename({'Percentage':'Drunk%'}, axis = 1, inplace = True)
```

In [65]:

```
# sorting by the % of people drinking in each state
drunkState.sort_values(by = 'Drunk%').head()
```

Out[65]:

	State	Drunk%
42	TN	10.9
44	UT	11.4
48	WV	11.8
0	AL	12.2
24	MS	12.5

In [66]:

```
State_df.head()
```

Out[66]:

	State	Num_of_Accidents	Population	Accident_Rate
0	CA	662985.0	39776830	1.666762
1	TX	298036.0	28704330	1.038296
2	FL	223403.0	21312211	1.048239
3	SC	145297.0	5088916	2.855166
4	NC	142453.0	10390149	1.371039

In [67]:

```
# Merging the accident rate per state dataset with the % people drinking in each state
 dataset
State_df = State_df.merge(drunkState, how = 'inner').sort_values('Drunk%', ascending =
False)
```

In [68]:

```
# Looking at the dataset
drunk_vs_rate = State_df.tail()
```

In [69]:

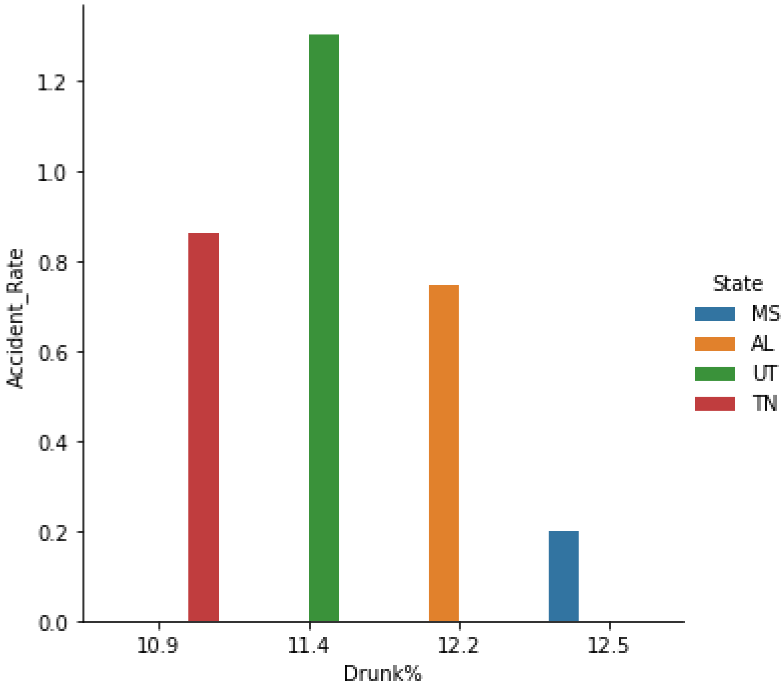
```
# Looking at these states to compare and see if the law mentioned above has any effects
on accident rate
temp = drunk_vs_rate[(drunk_vs_rate.State == 'MS') | (drunk_vs_rate.State == 'UT') | (d
runk_vs_rate.State == 'TN') | (drunk_vs_rate.State == 'AL')]
```

In [70]:

```
#Plotting the results above
sns.catplot(data= temp, x = 'Drunk%', y = 'Accident_Rate', hue = 'State', kind='bar')
```

Out[70]:

```
<seaborn.axisgrid.FacetGrid at 0x18cc48ab9b0>
```



Managerial Suggestion
Looking into this finding we can assume that drinking alcohol (within limits) while driving is not the sole reason for accidents. States like Utah can look into other accident preventive measures adopted by Missisippi and prevent accidents.

Part 4 - Machine Learning Algorithims

Run classification

In [71]:

```
df.head()
```

Out[71]:

	Severity	Start_Time	Start_Lat	Start_Lng	Distance(mi)	Description	Street	Side
ID								
A-1	3	2016-02-08 05:46:00	39.865147	-84.058723	0.01	Right lane blocked due to accident on I-70 Eas...	I-70 E	R
A-2	2	2016-02-08 06:07:59	39.928059	-82.831184	0.01	Accident on Brice Rd at Tussing Rd. Expect del...	Brice Rd	L
A-3	2	2016-02-08 06:49:27	39.063148	-84.032608	0.01	Accident on OH-32 State Route 32 Westbound at ...	State Route 32	R
A-4	3	2016-02-08 07:23:34	39.747753	-84.205582	0.01	Accident on I-75 Southbound at Exits 52 52B US...	I-75 S	R
A-5	2	2016-02-08 07:39:07	39.627781	-84.188354	0.01	Accident on McEwen Rd at OH-725 Miamisburg Cen...	Miamisburg Centerville Rd	R

In [72]:

```
sub_df=df.loc[:,['Severity','State','Timezone','Temperature(F)','Visibility(mi)','Bump'
,'Crossing','Give_Way','Junction','No_Exit',\
           'Railway','Stop']]
```

In [73]:

```
sub_df.head()
```

Out[73]:

ID	Severity	State	Timezone	Temperature(F)	Visibility(mi)	Bump	Crossing	Give_Way	Ju
A-1	3	OH	US/Eastern	36.9	10.0	False	False	False	
A-2	2	OH	US/Eastern	37.9	10.0	False	False	False	
A-3	2	OH	US/Eastern	36.0	10.0	False	False	False	
A-4	3	OH	US/Eastern	35.1	9.0	False	False	False	
A-5	2	OH	US/Eastern	36.0	6.0	False	False	False	

In [74]:

```
sub_df['State'] = sub_df['State'].astype('category')
sub_df['State_category'] = sub_df['State'].cat.codes
```

In [75]:

```
sub_df['Timezone'] = sub_df['Timezone'].astype('category')
sub_df['tz_category'] = sub_df['Timezone'].cat.codes
```

In [76]:

```
sub_df['Bump'] = sub_df.Bump + 0.0
sub_df['Crossing'] = sub_df.Crossing + 0.0
sub_df['Give_Way'] = sub_df.Give_Way + 0.0
sub_df['Junction'] = sub_df.Junction + 0.0
sub_df['No_Exit'] = sub_df.No_Exit + 0.0
sub_df['Railway'] = sub_df.Railway + 0.0
sub_df['Stop'] = sub_df.Stop + 0.0
```

In [77]:

```
sub_df.head()
```

Out[77]:

	Severity	State	Timezone	Temperature(F)	Visibility(mi)	Bump	Crossing	Give_Way	Ju
ID									
A-1	3	OH	US/Eastern	36.9	10.0	0.0	0.0	0.0	
A-2	2	OH	US/Eastern	37.9	10.0	0.0	0.0	0.0	
A-3	2	OH	US/Eastern	36.0	10.0	0.0	0.0	0.0	
A-4	3	OH	US/Eastern	35.1	9.0	0.0	0.0	0.0	
A-5	2	OH	US/Eastern	36.0	6.0	0.0	0.0	0.0	

In [78]:

```
sub_df.isna().any()
```

Out[78]:

```
Severity          False
State             False
Timezone          False
Temperature(F)    False
Visibility(mi)    False
Bump              False
Crossing          False
Give_Way          False
Junction          False
No_Exit           False
Railway           False
Stop              False
State_category    False
tz_category       False
dtype: bool
```

In [79]:

```
# Importing the dataset
dataset = sub_df
X = dataset.iloc[:, 5:14].values
y = dataset.iloc[:, 0].values
```

In [80]:

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_stat
e = 0)
```

In [81]:

```
# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

```
C:\Users\harsh\AppData\Roaming\Python\Python37\site-packages\sklearn\linea
r_model\_logistic.py:940: 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 i
n:
    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-reg
ression
```

Out[81]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
e,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=0, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [82]:

```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
```

In [83]:

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

In [84]:

```
cm #pretty good classification
```

Out[84]:

```
array([[     0,    256,      0,      0],
       [     0, 497576,      0,      0],
       [     0, 221116,      0,      0],
       [     0,  23190,      0,      0]], dtype=int64)
```

In [85]:

```
Accuracy = classifier.score(X_test, y_test)
print("Accuracy : ",Accuracy)
```

```
Accuracy :  0.6704629058207503
```

Time-Series Forecasting

In [86]:

```
df.head()
```

Out[86]:

	Severity	Start_Time	Start_Lat	Start_Lng	Distance(mi)	Description	Street	Side
ID								
A-1	3	2016-02-08 05:46:00	39.865147	-84.058723	0.01	Right lane blocked due to accident on I-70 Eas...	I-70 E	R
A-2	2	2016-02-08 06:07:59	39.928059	-82.831184	0.01	Accident on Brice Rd at Tussing Rd. Expect del...	Brice Rd	L
A-3	2	2016-02-08 06:49:27	39.063148	-84.032608	0.01	Accident on OH-32 State Route 32 Westbound at ...	State Route 32	R
A-4	3	2016-02-08 07:23:34	39.747753	-84.205582	0.01	Accident on I-75 Southbound at Exits 52 52B US...	I-75 S	R
A-5	2	2016-02-08 07:39:07	39.627781	-84.188354	0.01	Accident on McEwen Rd at OH-725 Miamisburg Cen...	Miamisburg Centerville Rd	R

In [87]:

```
df['Start_Time'] = pd.to_datetime(df['Start_Time']).dt.strftime('%Y-%m')
```

In [88]:

```
df.head(1)
```

Out[88]:

	Severity	Start_Time	Start_Lat	Start_Lng	Distance(mi)	Description	Street	Side	Ci
ID									
A-1	3	2016-02	39.865147	-84.058723	0.01	Right lane blocked due to accident on I-70 Eas...	I-70 E	R	Dayt

In [89]:

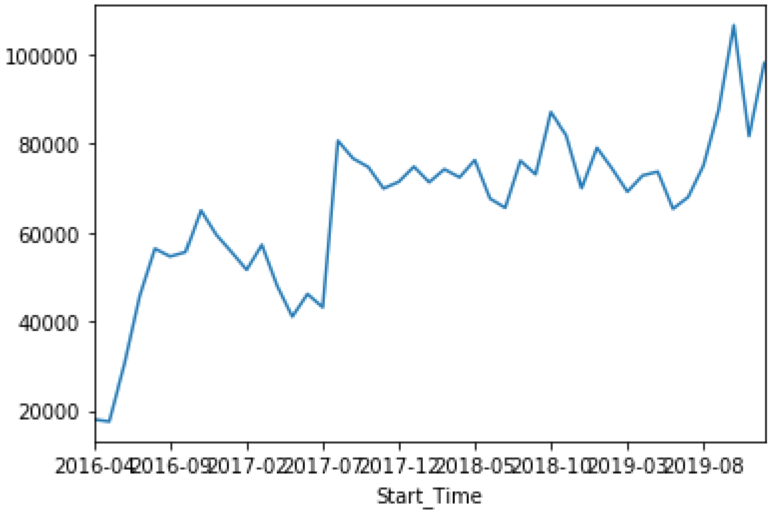
```
df_ts = df[(df.Start_Time > '2016-03') & (df.Start_Time < '2020-01')].groupby('Start_Time').size()
```

In [90]:

```
df_ts.plot()
```

Out[90]:

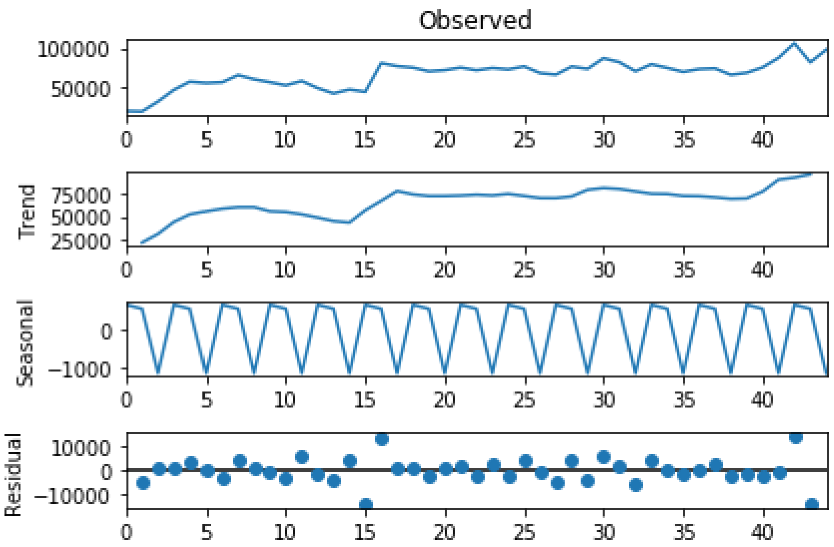
```
<matplotlib.axes._subplots.AxesSubplot at 0x18cc4967128>
```



In [91]:

```
#from chart_studio.plotly import plot_mpl
from statsmodels.tsa.seasonal import seasonal_decompose
decomposed = seasonal_decompose(np.asarray(df_ts), freq=3)
fig = decomposed.plot()
#plot_mpl(fig)
```

```
C:\Users\harsh\AppData\Local\Continuum\anaconda3\lib\site-packages\ipykern
el_launcher.py:3: FutureWarning:

the 'freq' keyword is deprecated, use 'period' instead
```



In [92]:

```
from pmdarima import auto_arima
stepwise_model = auto_arima(df_ts, start_p=1, start_q=1,
                            max_p=3, max_q=3, m=12,
                            start_P=0, seasonal=True,
                            d=1, D=1, trace=True,
                            error_action='ignore',
                            suppress_warnings=True,
                            stepwise=True)
print(stepwise_model.aic())
```

```
Performing stepwise search to minimize aic
Fit ARIMA: (1, 1, 1)x(0, 1, 1, 12) (constant=True); AIC=693.193, BIC=700.5
22, Time=0.298 seconds
Fit ARIMA: (0, 1, 0)x(0, 1, 0, 12) (constant=True); AIC=696.035, BIC=698.9
66, Time=0.007 seconds
Fit ARIMA: (1, 1, 0)x(1, 1, 0, 12) (constant=True); AIC=693.724, BIC=699.5
87, Time=0.108 seconds
Fit ARIMA: (0, 1, 1)x(0, 1, 1, 12) (constant=True); AIC=691.531, BIC=697.3
94, Time=0.136 seconds
Fit ARIMA: (0, 1, 0)x(0, 1, 0, 12) (constant=False); AIC=694.156, BIC=695.
622, Time=0.013 seconds
Fit ARIMA: (0, 1, 1)x(0, 1, 0, 12) (constant=True); AIC=695.425, BIC=699.8
22, Time=0.122 seconds
Fit ARIMA: (0, 1, 1)x(1, 1, 1, 12) (constant=True); AIC=693.401, BIC=700.7
30, Time=0.285 seconds
Fit ARIMA: (0, 1, 1)x(0, 1, 2, 12) (constant=True); AIC=692.896, BIC=700.2
25, Time=0.422 seconds
Fit ARIMA: (0, 1, 1)x(1, 1, 0, 12) (constant=True); AIC=693.593, BIC=699.4
56, Time=0.098 seconds
Fit ARIMA: (0, 1, 1)x(1, 1, 2, 12) (constant=True); AIC=694.279, BIC=703.0
73, Time=0.919 seconds
Fit ARIMA: (0, 1, 0)x(0, 1, 1, 12) (constant=True); AIC=692.342, BIC=696.7
39, Time=0.212 seconds
Fit ARIMA: (0, 1, 2)x(0, 1, 1, 12) (constant=True); AIC=691.472, BIC=698.8
00, Time=0.167 seconds
Fit ARIMA: (0, 1, 2)x(0, 1, 0, 12) (constant=True); AIC=697.712, BIC=703.5
74, Time=0.048 seconds
Fit ARIMA: (0, 1, 2)x(1, 1, 1, 12) (constant=True); AIC=693.315, BIC=702.1
09, Time=0.302 seconds
Fit ARIMA: (0, 1, 2)x(0, 1, 2, 12) (constant=True); AIC=693.295, BIC=702.0
90, Time=0.512 seconds
Fit ARIMA: (0, 1, 2)x(1, 1, 0, 12) (constant=True); AIC=694.434, BIC=701.7
62, Time=0.128 seconds
Fit ARIMA: (0, 1, 2)x(1, 1, 2, 12) (constant=True); AIC=695.226, BIC=705.4
86, Time=0.941 seconds
Fit ARIMA: (1, 1, 2)x(0, 1, 1, 12) (constant=True); AIC=689.101, BIC=697.8
96, Time=0.656 seconds
Near non-invertible roots for order (1, 1, 2)(0, 1, 1, 12); setting score
to inf (at least one inverse root too close to the border of the unit circ
le: 0.998)
Fit ARIMA: (0, 1, 3)x(0, 1, 1, 12) (constant=True); AIC=694.393, BIC=703.1
87, Time=0.368 seconds
Fit ARIMA: (1, 1, 3)x(0, 1, 1, 12) (constant=True); AIC=693.413, BIC=703.6
73, Time=0.473 seconds
Total fit time: 6.225 seconds
689.1014821347418
```

In [93]:

```
df.Start_Time.unique()
```

Out[93]:

```
array(['2016-02', '2016-03', '2016-06', '2016-07', '2016-08', '2016-11',
       '2016-12', '2017-01', '2016-10', '2016-09', '2016-04', '2016-05',
       '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07',
       '2019-12', '2019-11', '2019-10', '2019-08', '2019-07', '2019-09',
       '2019-06', '2019-05', '2019-04', '2019-03', '2019-02', '2019-01',
       '2018-12', '2018-11', '2018-10', '2018-09', '2018-08', '2018-07',
       '2018-06', '2018-05', '2018-04', '2018-03', '2018-02', '2018-01',
       '2017-12', '2017-11', '2017-10', '2017-09', '2017-08', '2016-01'],
      dtype=object)
```

In [94]:

```
train = df_ts.loc['2016-04':'2019-10']
test = df_ts.loc['2019-11':]
```

In [95]:

```
stepwise_model.fit(train)
```

Out[95]:

```
ARIMA(maxiter=50, method='lbfgs', order=(1, 1, 2), out_of_sample_size=0,
      scoring='mse', scoring_args=None, seasonal_order=(0, 1, 1, 12),
      start_params=None, suppress_warnings=True, trend=None,
      with_intercept=True)
```

In [96]:

```
test_forecast = stepwise_model.predict(n_periods=2)
# This returns an array of predictions:
print(test_forecast)
```

```
[93339.46966028 78868.73784324]
```

In [97]:

```
test_forecast = pd.DataFrame(test_forecast,index = test.index,columns=['Prediction'])
```

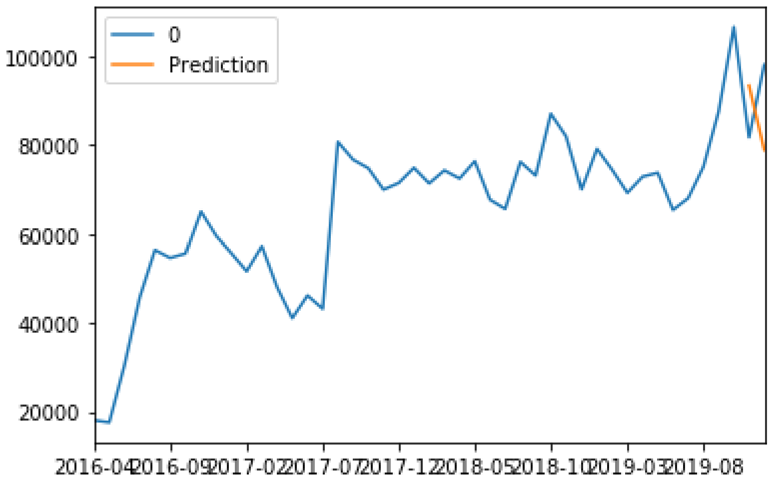
In [98]:

```
pd.concat([df_ts,test_forecast],axis=1).plot()
```

```
C:\Users\harsh\AppData\Local\Continuum\anaconda3\lib\site-packages\ipykern
el_launcher.py:1: FutureWarning:

Sorting because non-concatenation axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.
```

Out[98]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18cc3622e48>
```



In [99]:

```
#Final forecasting of next 6 months
stepwise_model.fit(df_ts['2016-04':'2019-12'])
forecast = pd.Series(stepwise_model.predict(n_periods=6))
```

Out[99]:

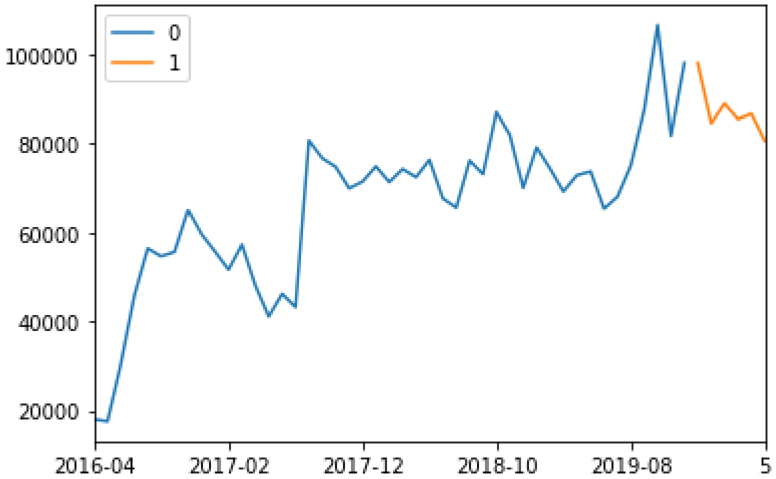
```
ARIMA(maxiter=50, method='lbfgs', order=(1, 1, 2), out_of_sample_size=0,
      scoring='mse', scoring_args=None, seasonal_order=(0, 1, 1, 12),
      start_params=None, suppress_warnings=True, trend=None,
      with_intercept=True)
```

In [100]:

```
pd.concat([df_ts,forecast],axis=1).plot()
```

Out[100]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18d489f4dd8>
```



K Means

In [101]:

```
kmeans = KMeans(n_clusters=2,random_state=0)
```

In [102]:

```
df.head(1)
```

Out[102]:

	Severity	Start_Time	Start_Lat	Start_Lng	Distance(mi)	Description	Street	Side	Ci
ID									
A-1	3	2016-02	39.865147	-84.058723	0.01	Right lane blocked due to accident on I-70 Eas...	I-70 E	R	Dayt

In [103]:

```
dfK = df[['Severity','Amenity','Bump','Crossing','Give_Way','Junction','No_Exit','Railw
ay','Roundabout'\
          ,'Station','Stop','Traffic_Signal','Turning_Loop']]
```

In [104]:

```
dfK.replace([1,2],0, inplace = True)
dfK.replace([3,4],1, inplace = True)
```

```
C:\Users\harsh\AppData\Local\Continuum\anaconda3\lib\site-packages\pandas
\core\frame.py:4042: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy
```

In [105]:

```
kmeans.fit(dfK)
```

Out[105]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=0, tol=0.0001, verbose=0)
```

In [106]:

```
df2 = pd.DataFrame.copy(dfK)
df2['Cluster'] = kmeans.labels_
df2.groupby('Cluster').mean()
```

Out[106]:

	Severity	Amenity	Bump	Crossing	Give_Way	Junction	No_Exit	Railway	Rc
Cluster									
0	0	0.016041	0.000202	0.095086	0.002907	0.067793	0.001319	0.010136	
1	1	0.003352	0.000052	0.018649	0.001622	0.105329	0.000787	0.005093	

In [107]:

```
df.groupby('Severity')[['Amenity','Bump','Crossing','Give_Way','Junction','No_Exit','Ra
ilway','Roundabout'\
          ,'Station','Stop','Traffic_Signal','Turning_Loop']].sum()
```

Out[107]:

	Amenity	Bump	Crossing	Give_Way	Junction	No_Exit	Railway	Roundabout	St
Severity									
1	19.0	0.0	87.0	1.0	13.0	3.0	7.0	0.0	
2	31917.0	403.0	189220.0	5787.0	134956.0	2622.0	20173.0	160.0	484
3	2424.0	47.0	14559.0	1308.0	92599.0	677.0	4380.0	2.0	68
4	853.0	4.0	3673.0	278.0	10376.0	92.0	599.0	6.0	10

Decision Tree

In [108]:

```
dt = tree.DecisionTreeClassifier(max_depth=5)
```

In [124]:

```
df2 = pd.DataFrame.copy(df)
```

In [125]:

```
df2.head(1)
```

Out[125]:

	Severity	Start_Time	Start_Lat	Start_Lng	Distance(mi)	Description	Street	Side	Ci
ID									
A-1	0	2016-02	39.865147	-84.058723	0.01	Right lane blocked due to accident on I-70 Eas...	I-70 E	R	Dayt

In [126]:

```
X = df2.drop(['Severity','Description','Street','Side','City','County','State','Countr
y','Timezone'\
              ,'Airport_Code', 'Distance(mi)','Start_Lng','Start_Lat','Start_Time','Roa
d_type'],axis=1)
Y = df.Severity
```

In [127]:

```
Y.replace([1,2],0, inplace = True)
Y.replace([3,4],1, inplace = True)
```

In [128]:

```
X = pd.get_dummies(columns=['Weather_Condition','Sunrise_Sunset','Civil_Twilight','Naut
ical_Twilight','Astronomical_Twilight'\
                             ,'Day_of_week','Month'],data=X)
```

In [129]:

```
dt.fit(X,Y)
```

Out[129]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=5, max_features=None, max_leaf_nodes=Non
e,
                       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')
```