Project

June 24, 2021

[46]:
```
#CSCE 5300 Road Accident Dataset Analysis
#Abdulrahman Ghadi, Paul Phillips, Sumukha Parameshwara Bhat

#This code is based on https://github.com/JoaquimCSantos/
↪US-Car-Accident-Data-Exploration/blob/main/
↪How%20to%20avoid%20car%20accidents.ipynb
#For the machine learning code, https://github.com/RonghuiZhou/us-accidents/
↪blob/master/Machine%20Learning%20for%20US%20Accidents_PA_Mont_RZhou.ipynb

!pip install plotly
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
pd.options.mode.chained_assignment = None
pd.options.display.max_columns = 999
import plotly.graph_objects as go
import os
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_curve, auc
```

```
Requirement already satisfied: plotly in c:\users\paul\anaconda3\lib\site-
packages (5.0.0)
Requirement already satisfied: tenacity>=6.2.0 in
c:\users\paul\anaconda3\lib\site-packages (from plotly) (7.0.0)
Requirement already satisfied: six in c:\users\paul\anaconda3\lib\site-packages
(from plotly) (1.15.0)
```

import dataset as dataframe

[2]:
```python
train_df = pd.read_csv("US_Accidents_Dec20_Updated.
 ↪csv",parse_dates=['Start_Time','End_Time'])
train_df.head(5)
```

[2]:
```
    ID  Severity          Start_Time            End_Time  Start_Lat  \
0  A-1         2 2019-05-21 08:29:55 2019-05-21 09:29:40  34.808868
1  A-2         2 2019-10-07 17:43:09 2019-10-07 19:42:50  35.090080
2  A-3         2 2020-12-13 21:53:00 2020-12-13 22:44:00  37.145730
3  A-4         2 2018-04-17 16:51:23 2018-04-17 17:50:46  39.110390
4  A-5         3 2016-08-31 17:40:49 2016-08-31 18:10:49  26.102942

    Start_Lng    End_Lat     End_Lng  Distance(mi)  \
0  -82.269157  34.808868  -82.269157           0.0
1  -80.745560  35.090080  -80.745560           0.0
2 -121.985052  37.165850 -121.988062           1.4
3 -119.773781  39.110390 -119.773781           0.0
4  -80.265091  26.102942  -80.265091           0.0

                                        Description  Number  \
0           Accident on Tanner Rd at Pennbrooke Ln.   439.0
1  Accident on Houston Branch Rd at Providence Br…  3299.0
2  Stationary traffic on CA-17 from Summit Rd (CA…     NaN
3        Accident on US-395 Southbound at Topsy Ln.     NaN
4  Accident on I-595 Westbound at Exit 4 / Pine I…     NaN

                 Street Side             City       County State     Zipcode  \
0              Tanner Rd    R       Greenville   Greenville    SC  29607-6027
1  Providence Branch Ln    R        Charlotte  Mecklenburg    NC  28270-8560
2         Santa Cruz Hwy    R        Los Gatos  Santa Clara    CA       95033
3      US Highway 395 S    R      Carson City      Douglas    NV       89705
4               I-595 W    R  Fort Lauderdale      Broward    FL       33324

  Country    Timezone Airport_Code    Weather_Timestamp  Temperature(F)  \
0      US  US/Eastern         KGMU  2019-05-21 08:53:00            76.0
1      US  US/Eastern         KEQY  2019-10-07 17:53:00            76.0
2      US  US/Pacific         KSJC  2020-12-13 21:53:00            51.0
3      US  US/Pacific         KCXP  2018-04-17 16:55:00            53.6
4      US  US/Eastern         KHWO  2016-08-31 17:53:00            84.2

   Wind_Chill(F)  Humidity(%)  Pressure(in)  Visibility(mi) Wind_Direction  \
0           76.0         52.0         28.91            10.0              N
1           76.0         62.0         29.30            10.0            VAR
2           51.0         80.0         30.17            10.0              W
3            NaN         16.0         30.16            10.0            SSW
4            NaN         84.0         29.92            10.0            SSE

   Wind_Speed(mph)  Precipitation(in) Weather_Condition  Amenity   Bump  \
```

```
0            7.0              0.0           Fair    False   False
1            3.0              0.0         Cloudy    False   False
2            6.0              0.0           Fair    False   False
3            4.6              NaN          Clear    False   False
4           13.8              NaN       Overcast    False   False

  Crossing  Give_Way  Junction  No_Exit  Railway  Roundabout  Station   Stop  \
0    False     False     False    False    False       False    False  False
1    False     False     False    False    False       False    False  False
2    False     False     False    False    False       False    False  False
3    False     False     False    False    False       False    False  False
4    False     False      True    False    False       False    False  False

  Traffic_Calming  Traffic_Signal  Turning_Loop Sunrise_Sunset  \
0           False           False         False            Day
1           False           False         False            Day
2           False           False         False          Night
3           False            True         False            Day
4           False            True         False            Day

 Civil_Twilight Nautical_Twilight Astronomical_Twilight
0           Day               Day                   Day
1           Day               Day                   Day
2         Night             Night                 Night
3           Day               Day                   Day
4           Day               Day                   Day
```

[3]:
```python
print (train_df.shape)
train_df.head(2)
```

```
(2906610, 47)
```

[3]:
```
    ID  Severity           Start_Time             End_Time  Start_Lat  \
0  A-1         2  2019-05-21 08:29:55  2019-05-21 09:29:40  34.808868
1  A-2         2  2019-10-07 17:43:09  2019-10-07 19:42:50  35.090080

   Start_Lng    End_Lat    End_Lng  Distance(mi)  \
0 -82.269157  34.808868 -82.269157           0.0
1 -80.745560  35.090080 -80.745560           0.0

                                        Description  Number  \
0            Accident on Tanner Rd at Pennbrooke Ln.   439.0
1  Accident on Houston Branch Rd at Providence Br…  3299.0

                  Street Side        City       County State     Zipcode  \
0              Tanner Rd    R  Greenville   Greenville    SC  29607-6027
1  Providence Branch Ln    R   Charlotte  Mecklenburg    NC  28270-8560
```

```
  Country    Timezone Airport_Code    Weather_Timestamp  Temperature(F)  \
0      US  US/Eastern         KGMU  2019-05-21 08:53:00            76.0
1      US  US/Eastern         KEQY  2019-10-07 17:53:00            76.0

   Wind_Chill(F)  Humidity(%)  Pressure(in)  Visibility(mi) Wind_Direction  \
0           76.0         52.0         28.91            10.0              N
1           76.0         62.0         29.30            10.0            VAR

   Wind_Speed(mph)  Precipitation(in) Weather_Condition  Amenity   Bump  \
0              7.0                0.0              Fair    False  False
1              3.0                0.0            Cloudy    False  False

   Crossing  Give_Way  Junction  No_Exit  Railway  Roundabout  Station   Stop  \
0     False     False     False    False    False       False    False  False
1     False     False     False    False    False       False    False  False

   Traffic_Calming  Traffic_Signal  Turning_Loop Sunrise_Sunset  \
0            False           False         False            Day
1            False           False         False            Day

  Civil_Twilight Nautical_Twilight Astronomical_Twilight
0            Day               Day                   Day
1            Day               Day                   Day
```

[4]:
```python
print (train_df.shape)
train_df.head(1)
```

```
(2906610, 47)
```

[4]:
```
    ID  Severity           Start_Time             End_Time  Start_Lat  \
0  A-1         2 2019-05-21 08:29:55 2019-05-21 09:29:40  34.808868

   Start_Lng    End_Lat    End_Lng  Distance(mi)  \
0 -82.269157  34.808868 -82.269157           0.0

                              Description  Number     Street Side  \
0  Accident on Tanner Rd at Pennbrooke Ln.   439.0  Tanner Rd    R

         City      County State     Zipcode Country    Timezone Airport_Code  \
0  Greenville  Greenville    SC  29607-6027      US  US/Eastern         KGMU

     Weather_Timestamp  Temperature(F)  Wind_Chill(F)  Humidity(%)  \
0  2019-05-21 08:53:00            76.0           76.0         52.0

   Pressure(in)  Visibility(mi) Wind_Direction  Wind_Speed(mph)  \
0         28.91            10.0              N              7.0

   Precipitation(in) Weather_Condition  Amenity   Bump  Crossing  Give_Way  \
```

```
0                0.0              Fair    False  False     False     False

   Junction  No_Exit  Railway  Roundabout  Station   Stop  Traffic_Calming  \
0     False    False    False       False    False  False            False

   Traffic_Signal  Turning_Loop Sunrise_Sunset Civil_Twilight  \
0           False         False            Day            Day

  Nautical_Twilight Astronomical_Twilight
0               Day                   Day
```

create a new column "impact"

[5]:
```python
#data Types
train_df.dtypes
```

[5]:
```
ID                               object
Severity                          int64
Start_Time               datetime64[ns]
End_Time                 datetime64[ns]
Start_Lat                       float64
Start_Lng                       float64
End_Lat                         float64
End_Lng                         float64
Distance(mi)                    float64
Description                      object
Number                          float64
Street                           object
Side                             object
City                             object
County                           object
State                            object
Zipcode                          object
Country                          object
Timezone                         object
Airport_Code                     object
Weather_Timestamp                object
Temperature(F)                  float64
Wind_Chill(F)                   float64
Humidity(%)                     float64
Pressure(in)                    float64
Visibility(mi)                  float64
Wind_Direction                   object
Wind_Speed(mph)                 float64
Precipitation(in)               float64
Weather_Condition                object
Amenity                            bool
Bump                               bool
```

```
Crossing                             bool
Give_Way                             bool
Junction                             bool
No_Exit                              bool
Railway                              bool
Roundabout                           bool
Station                              bool
Stop                                 bool
Traffic_Calming                      bool
Traffic_Signal                       bool
Turning_Loop                         bool
Sunrise_Sunset                     object
Civil_Twilight                     object
Nautical_Twilight                  object
Astronomical_Twilight              object
dtype: object
```

[6]:
```
train_df['Month'] = train_df['Start_Time'].dt.month
train_df['Year'] = train_df['Start_Time'].dt.year
train_df['Hour'] = train_df['Start_Time'].dt.hour
train_df['Weekday'] = train_df['Start_Time'].dt.weekday
train_df['Impact'] = (train_df['End_Time'] - train_df['Start_Time']).dt.
↪total_seconds()/60
```

clean the data based on the condition that the impact on traffic is between zero-one week,and drop duplicates & display uniques states

[7]:
```
oneweek = 60*24*7
df_clean = train_df[(train_df['Impact']>0) & (train_df['Impact']< oneweek)].
↪drop_duplicates(subset=['Start_Time','End_Time','City','Street','Number','Description'])

states = df_clean.State.unique()
states
```

[7]:
```
array(['SC', 'NC', 'CA', 'NV', 'FL', 'CO', 'TN', 'NY', 'TX', 'AZ', 'NJ',
       'MI', 'GA', 'VA', 'IN', 'LA', 'PA', 'MN', 'OH', 'MD', 'CT', 'IL',
       'MO', 'OR', 'NE', 'OK', 'UT', 'WA', 'AL', 'WI', 'MA', 'DC', 'MS',
       'KS', 'KY', 'ME', 'IA', 'WV', 'AR', 'ID', 'RI', 'WY', 'NM', 'MT',
       'NH', 'DE', 'ND', 'SD', 'VT'], dtype=object)
```

[8]:
```
city = df_clean.City.unique()
city
```

[8]:
```
array(['Greenville', 'Charlotte', 'Los Gatos', …, 'Allons', 'Adolphus',
       'Gowanda'], dtype=object)
```

dataset description

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

[9]:
```
           Severity     Start_Lat     Start_Lng       End_Lat       End_Lng  \
count  2.769380e+06  2.769380e+06  2.769380e+06  2.486668e+06  2.486668e+06
mean   2.297543e+00  3.655674e+01 -9.654857e+01  3.654613e+01 -9.632724e+01
std    5.565735e-01  4.996480e+00  1.774095e+01  4.997367e+00  1.764146e+01
min    1.000000e+00  2.455527e+01 -1.246238e+02  2.455527e+01 -1.246238e+02
25%    2.000000e+00  3.367997e+01 -1.178352e+02  3.366795e+01 -1.177351e+02
50%    2.000000e+00  3.610597e+01 -9.199359e+01  3.606112e+01 -9.111529e+01
75%    3.000000e+00  4.041993e+01 -8.090346e+01  4.037918e+01 -8.088122e+01
max    4.000000e+00  4.900220e+01 -6.711317e+01  4.907500e+01 -6.710924e+01

       Distance(mi)        Number  Temperature(F)  Wind_Chill(F)  \
count  2.769380e+06  9.644480e+05    2.706980e+06   1.597437e+06
mean   3.833527e-01  6.772075e+03    6.123598e+01   5.507805e+01
std    1.582932e+00  1.706855e+04    1.848095e+01   2.242770e+01
min    0.000000e+00  0.000000e+00   -8.900000e+01  -8.900000e+01
25%    0.000000e+00  9.540000e+02    4.900000e+01   3.900000e+01
50%    0.000000e+00  3.066000e+03    6.300000e+01   5.800000e+01
75%    2.510000e-01  7.923000e+03    7.500000e+01   7.300000e+01
max    3.336300e+02  9.999997e+06    2.030000e+02   1.740000e+02

         Humidity(%)  Pressure(in)  Visibility(mi)  Wind_Speed(mph)  \
count  2.703286e+06  2.716584e+06    2.702062e+06     2.468477e+06
mean   6.525573e+01  2.966493e+01    9.118533e+00     7.877233e+00
std    2.287986e+01  9.049384e-01    2.851076e+00     5.418822e+00
min    1.000000e+00  0.000000e+00    0.000000e+00     0.000000e+00
25%    4.900000e+01  2.960000e+01    1.000000e+01     4.600000e+00
50%    6.800000e+01  2.992000e+01    1.000000e+01     7.000000e+00
75%    8.400000e+01  3.007000e+01    1.000000e+01     1.040000e+01
max    1.000000e+02  5.804000e+01    1.400000e+02     9.840000e+02

       Precipitation(in)         Month          Year          Hour  \
count       1.482516e+06  2.769380e+06  2.769380e+06  2.769380e+06
mean        1.176244e-02  7.199289e+00  2.018521e+03  1.223750e+01
std         1.586527e-01  3.581208e+00  1.344426e+00  5.562745e+00
min         0.000000e+00  1.000000e+00  2.016000e+03  0.000000e+00
25%         0.000000e+00  4.000000e+00  2.017000e+03  8.000000e+00
50%         0.000000e+00  8.000000e+00  2.019000e+03  1.300000e+01
75%         0.000000e+00  1.000000e+01  2.020000e+03  1.700000e+01
max         2.400000e+01  1.200000e+01  2.020000e+03  2.300000e+01

            Weekday        Impact
count  2.769380e+06  2.769380e+06
mean   2.495801e+00  1.252292e+02
std    1.771157e+00  1.956720e+02
min    0.000000e+00  1.216667e+00
```

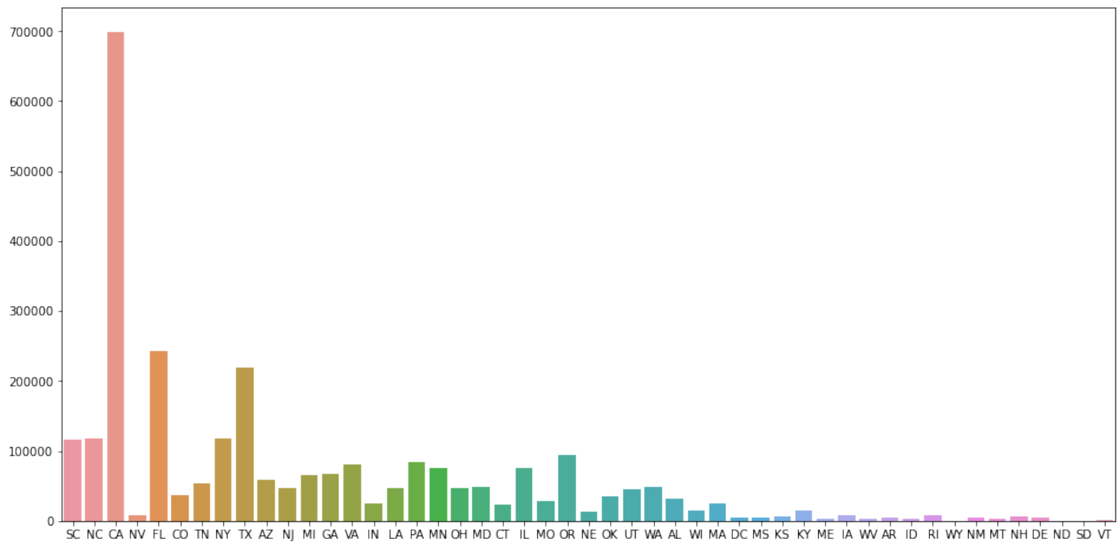
```
25%     1.000000e+00  2.981667e+01
50%     2.000000e+00  5.948333e+01
75%     4.000000e+00  1.374833e+02
max     6.000000e+00  1.007577e+04
```

display accidents per states

[10]:
```python
count_by_state=[]
for i in df_clean.State.unique():
    count_by_state.append(df_clean[df_clean['State']==i].count()['ID'])

fig,ax = plt.subplots(figsize=(16,8))
sns.barplot(states,count_by_state)
```

```
C:\Users\Paul\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

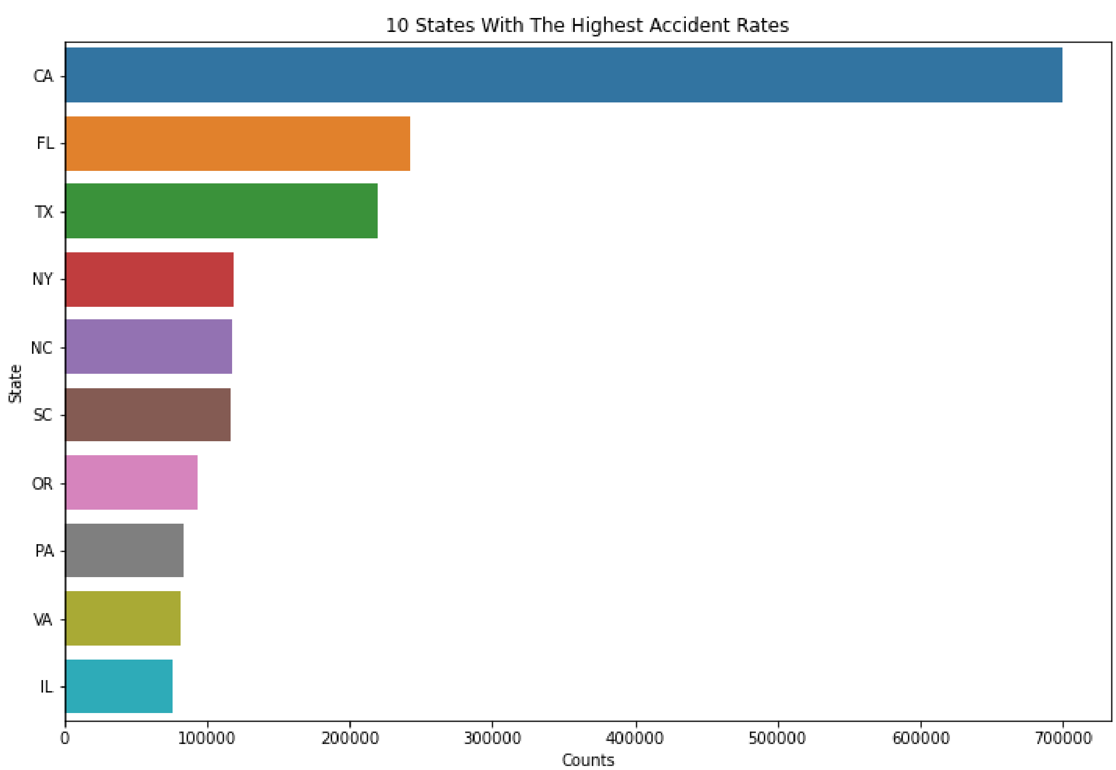
[10]: <AxesSubplot:>



[11]:
```python
df_st_ct = pd.value_counts(df_clean['State'])

fig = go.Figure(data=go.Choropleth(
    locations=df_st_ct.index,
    z = df_st_ct.values.astype(float),  # Data to be color-coded
    locationmode = 'USA-states',     # set of locations match entries in␣
 ↪`locations`
```

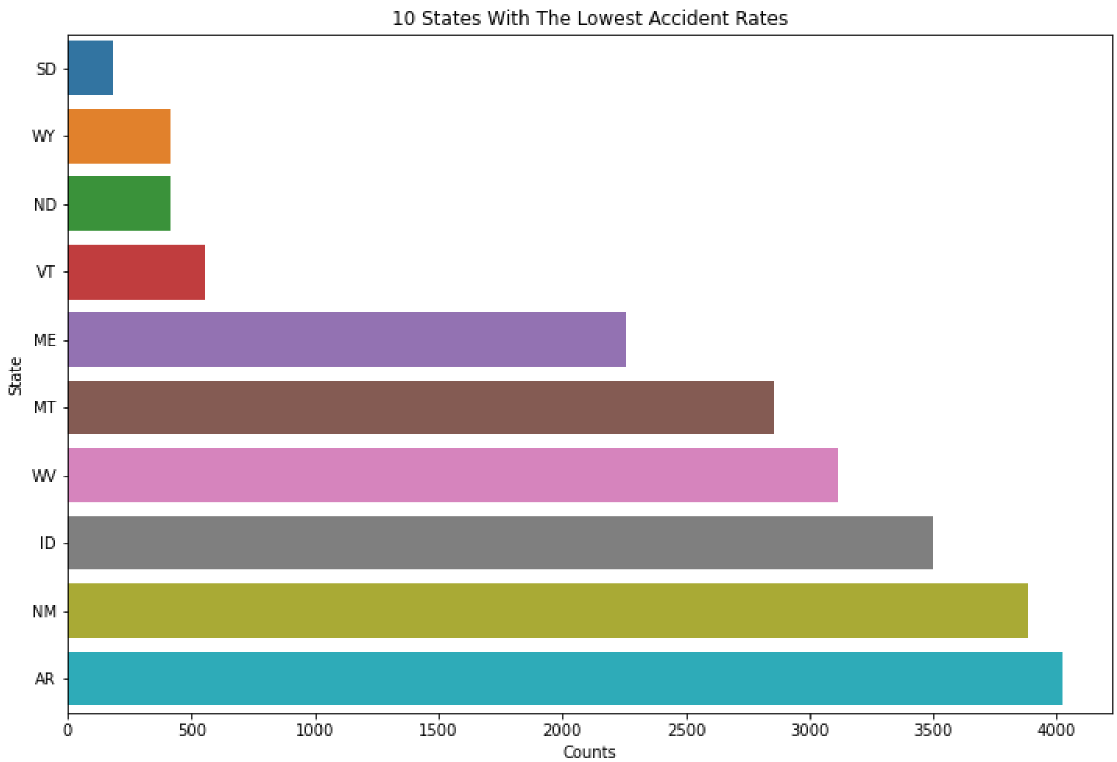
```
    colorscale = 'YlGnBu',
    colorbar_title = "Count",
))

fig.update_layout(
    title_text = 'US Accidents by State',
    geo_scope='usa',
)

fig.show()
```

[12]:
```
#10 states with the highest accident rates
df_st = df_clean.groupby('State').size().to_frame('Counts')
df_st = df_st.reset_index().sort_values('Counts', ascending = False)[:10]

fig, ax = plt.subplots(figsize = (12,8))
b = sns.barplot(y = 'State',x = 'Counts', data = df_st )

b.set_title("10 States With The Highest Accident Rates")

plt.show()
# these states are consistent with the states with largest population in the U.
 ↪S.
```



[13]:

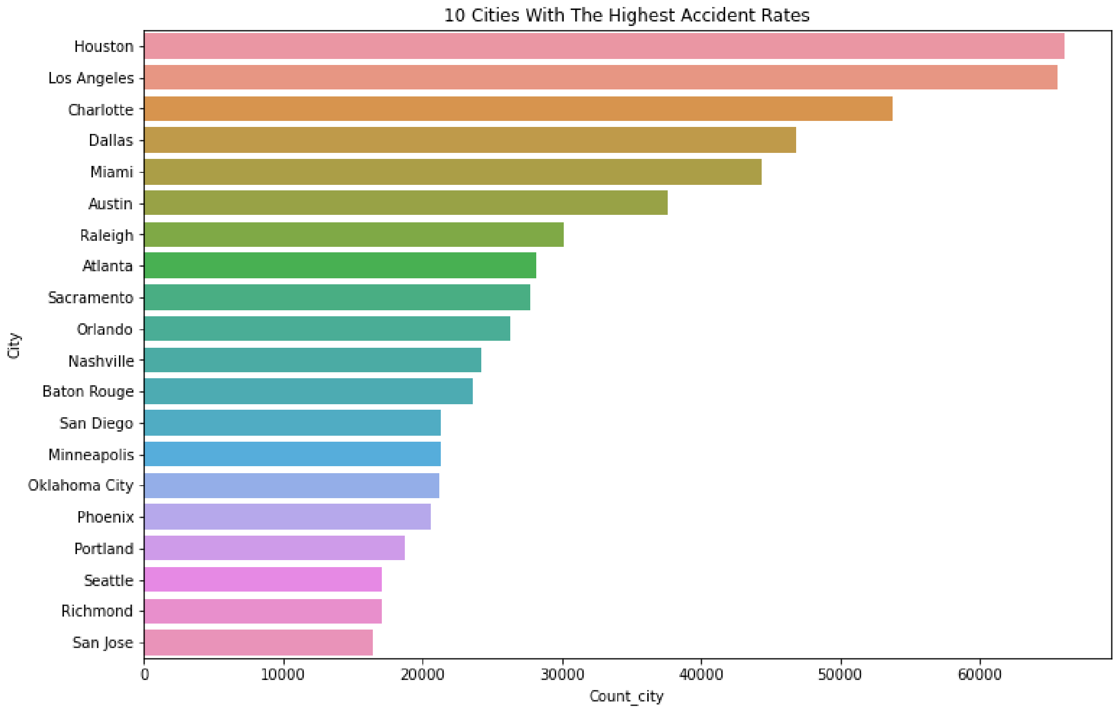
```
#10 states with the lowest accident rates
df_st = df_clean.groupby('State').size().to_frame('Counts')
df_st = df_st.reset_index().sort_values('Counts', ascending = True)[:10]

fig, ax = plt.subplots(figsize = (12,8))
b = sns.barplot(y = 'State',x = 'Counts', data = df_st )

b.set_title("10 States With The Lowest Accident Rates")

plt.show()
```



[14]:

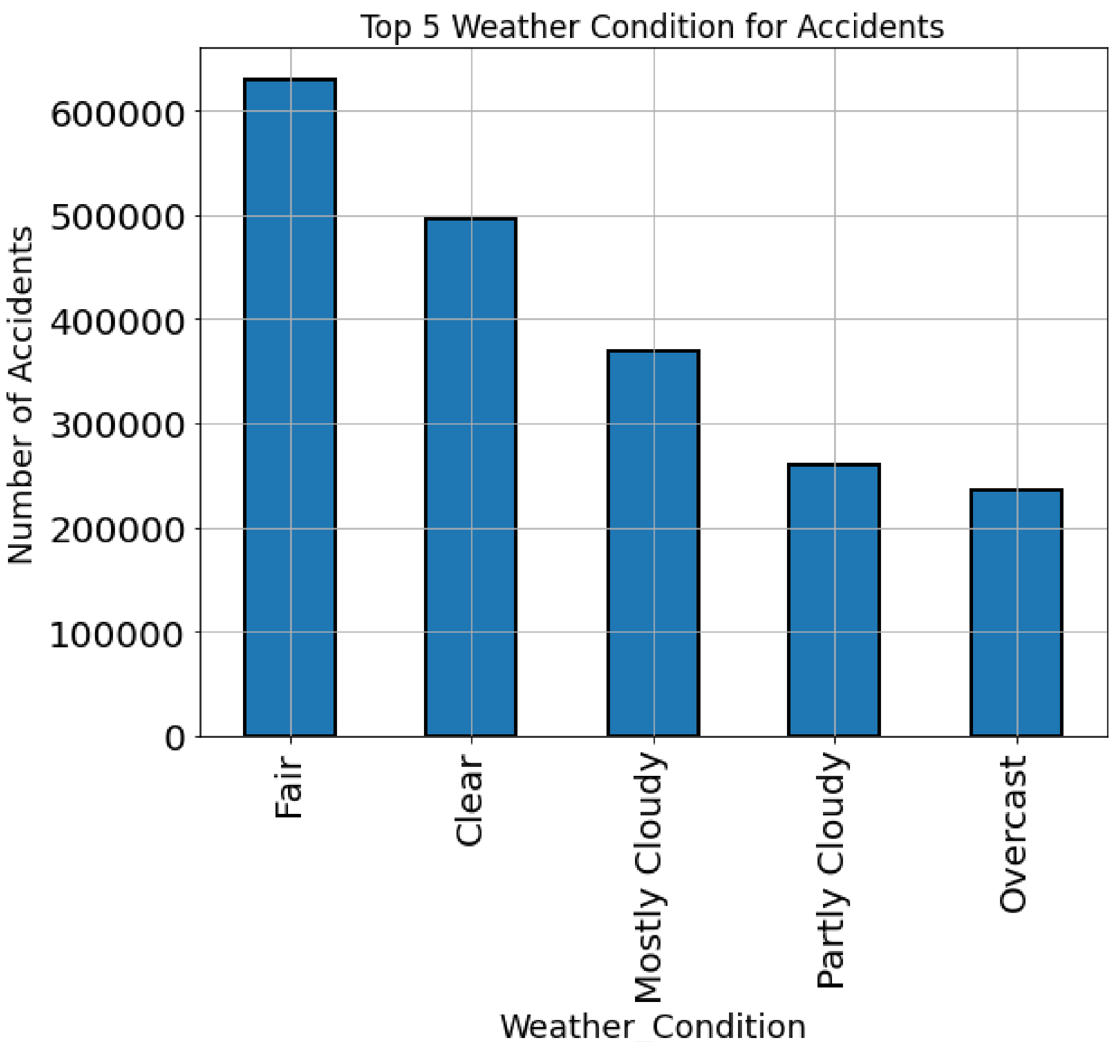
```
#20 cities with the highest accident rates
df_ci_cnt = df_clean.groupby('City').size().to_frame('Count_city')
df_ci_cnt = df_ci_cnt.reset_index().sort_values('Count_city', ascending =␣
↪False)[:20]

fig, ax = plt.subplots(figsize = (12,8))
b = sns.barplot(y = 'City',x = 'Count_city', data = df_ci_cnt )

b.set_title("10 Cities With The Highest Accident Rates")
```

```
plt.show()
```

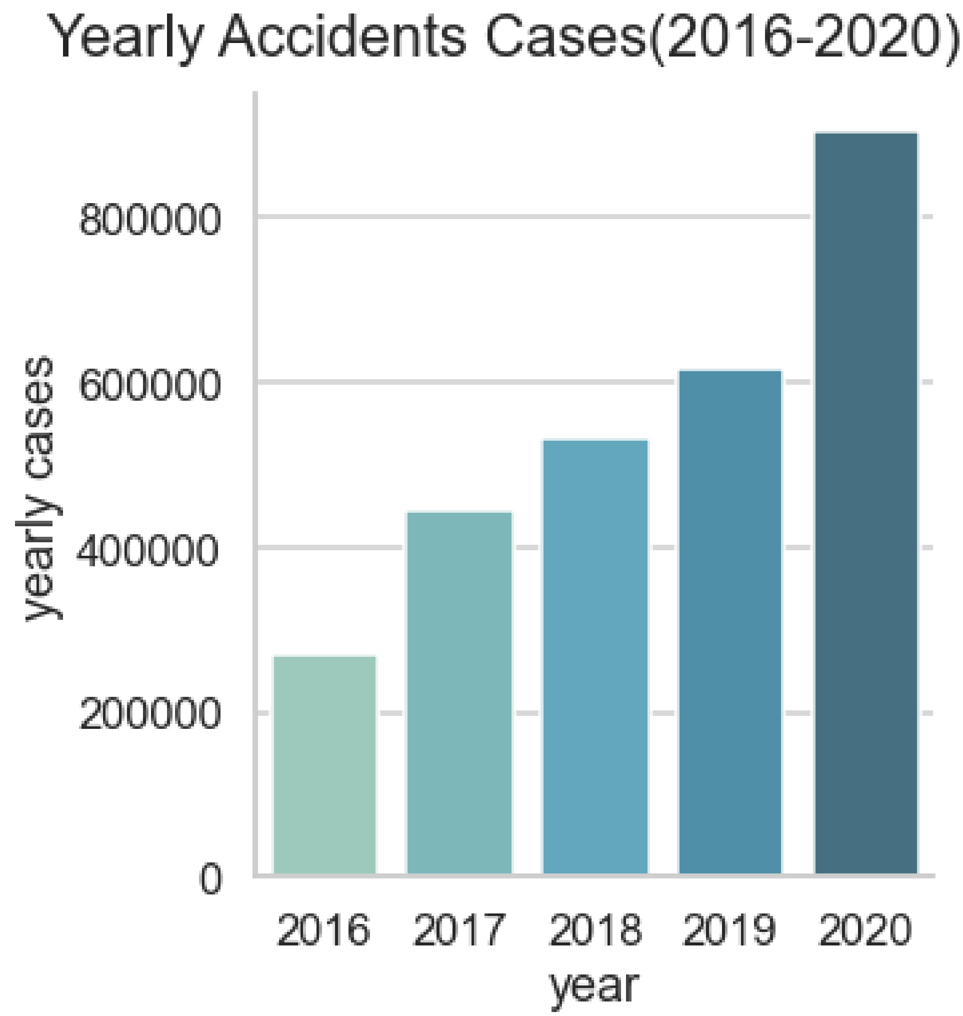


display top 5 Weather Condition for Accidents

[15]:
```
fig, ax=plt.subplots(figsize=(9,7))
df_clean['Weather_Condition'].value_counts().sort_values(ascending=False).
 ↪head(5).plot.bar(width=0.5,edgecolor='k',align='center',linewidth=2)
plt.xlabel('Weather_Condition',fontsize=18)
plt.ylabel('Number of Accidents',fontsize=18)
ax.tick_params(labelsize=20)
plt.title('Top 5 Weather Condition for Accidents',fontsize=17)
plt.grid()
plt.ioff()
```



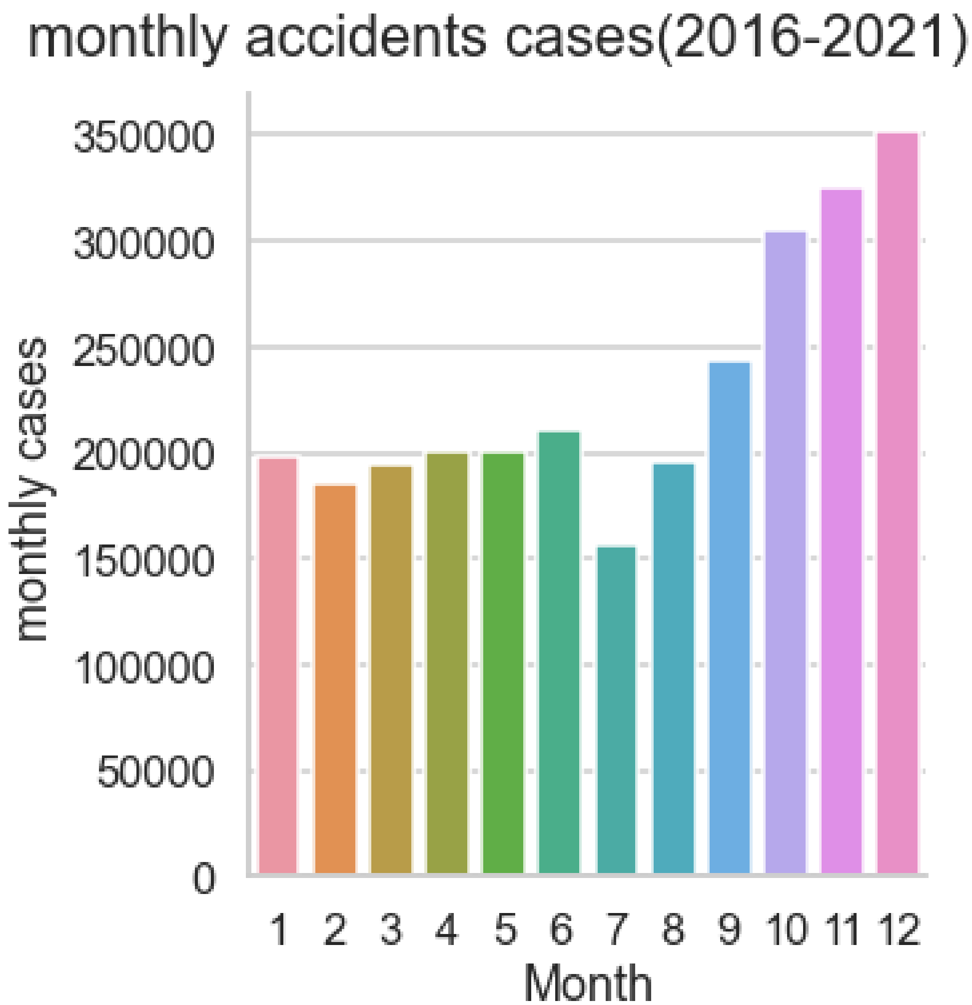
[16]:
```
#time series analysis
df1 =␣
↪df_clean[['Country','Start_Time','End_Time','Year','Month','Weekday','Hour','Impact','Sever
sns.set_style('whitegrid')
sns.set_context('talk')
sns.set_palette('GnBu_d')
a = sns.catplot(x='Year',data=df_clean[df_clean['Year'] < 2021],kind='count')
a.fig.suptitle('Yearly Accidents Cases(2016-2020)',y=1.03)
a.set(ylabel='yearly cases',xlabel='year')
plt.show()
# there is a growing trend of year accidents cases
```



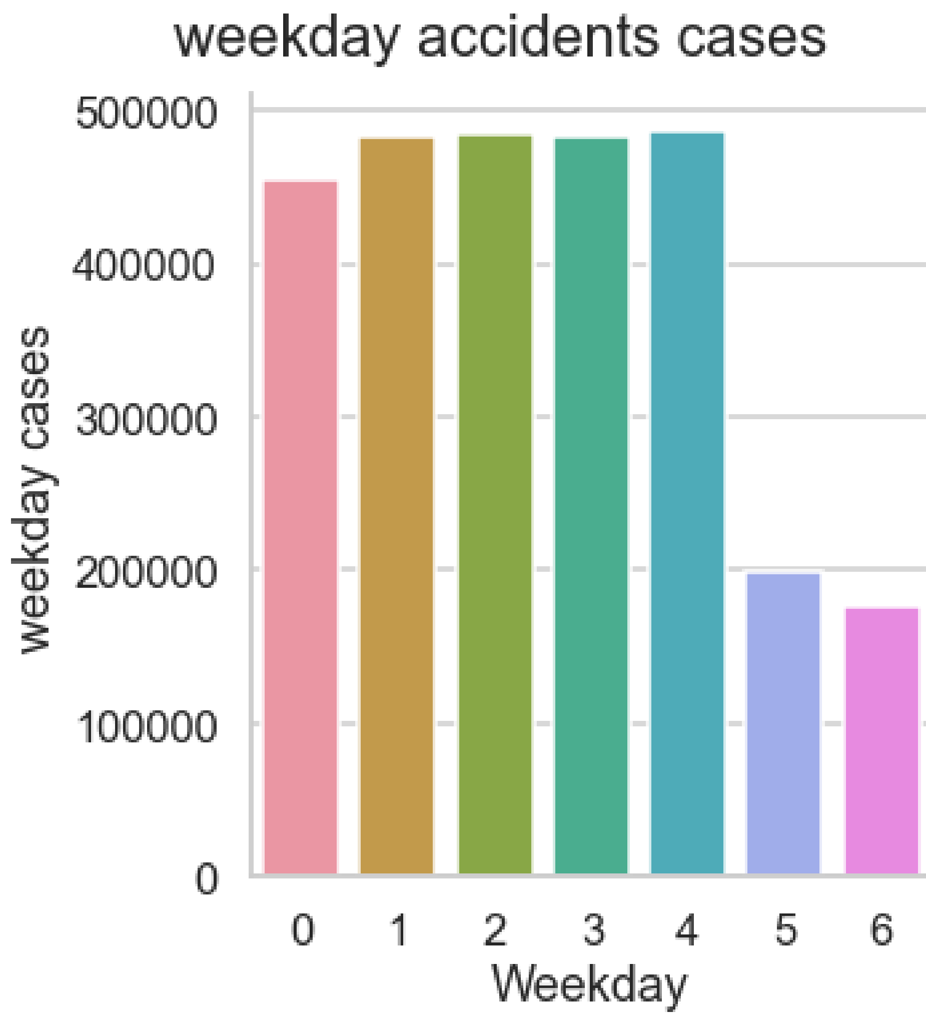
[ ]:

[ ]:

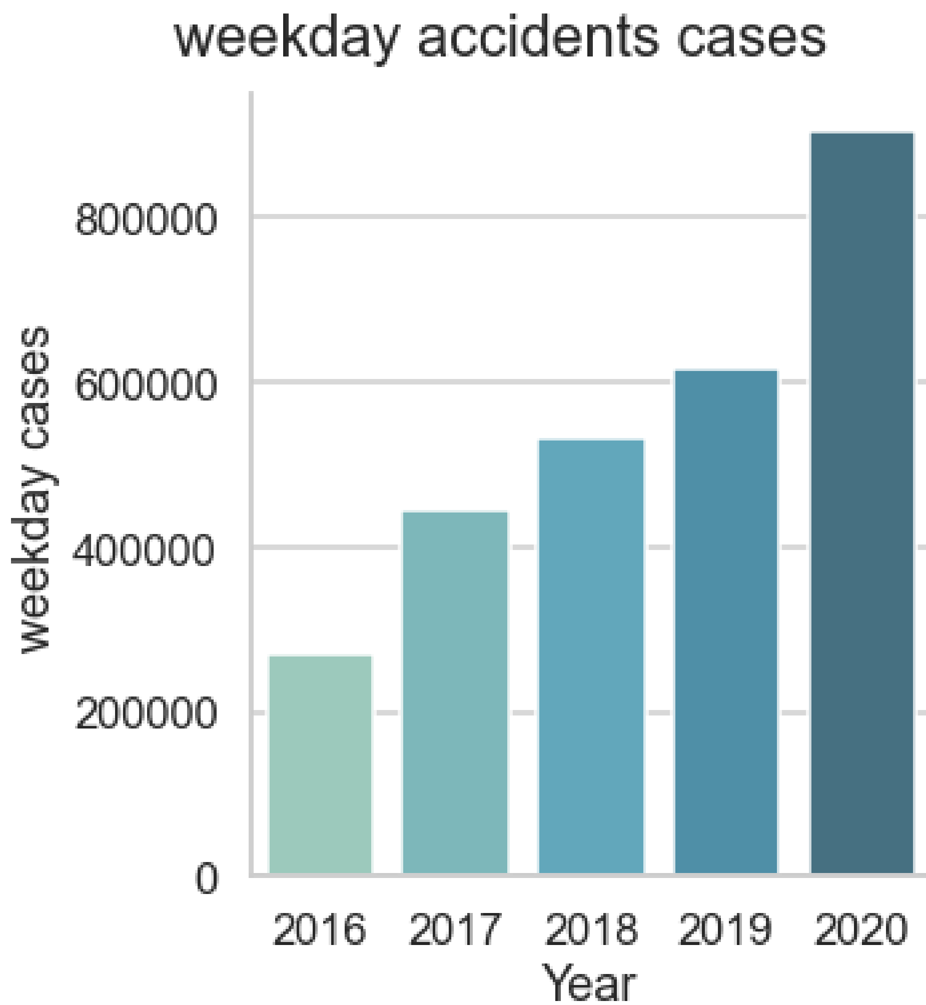
[17]:
```
sns.set_context('talk')
m = sns.catplot(x='Month',data=df1[df1['Year'] < 2021],kind='count')
m.fig.suptitle('monthly accidents cases(2016-2021)',y=1.03)
m.set(ylabel='monthly cases')
plt.show()
```



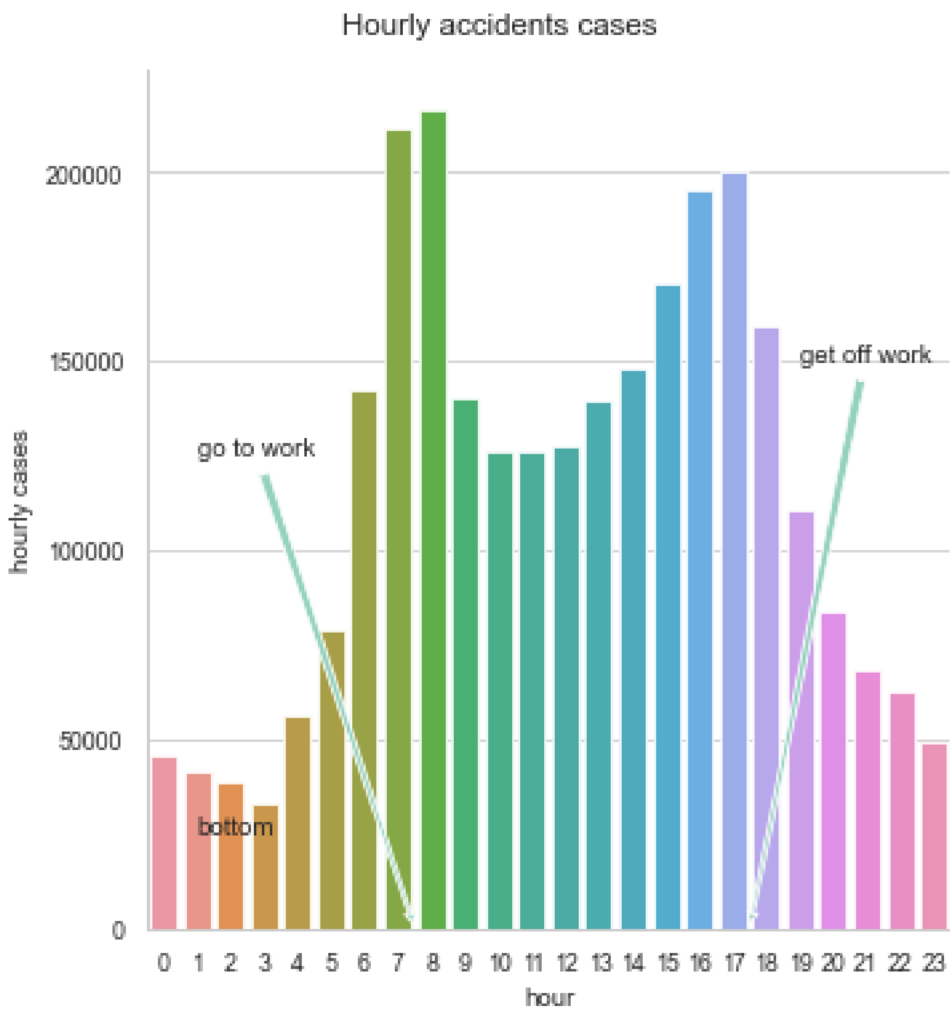
[18]:
```
sns.set_context('talk')
w = sns.catplot(x='Weekday',data=df1,kind='count')
w.fig.suptitle('weekday accidents cases',y=1.03)
w.set(ylabel='weekday cases')
plt.show()
```



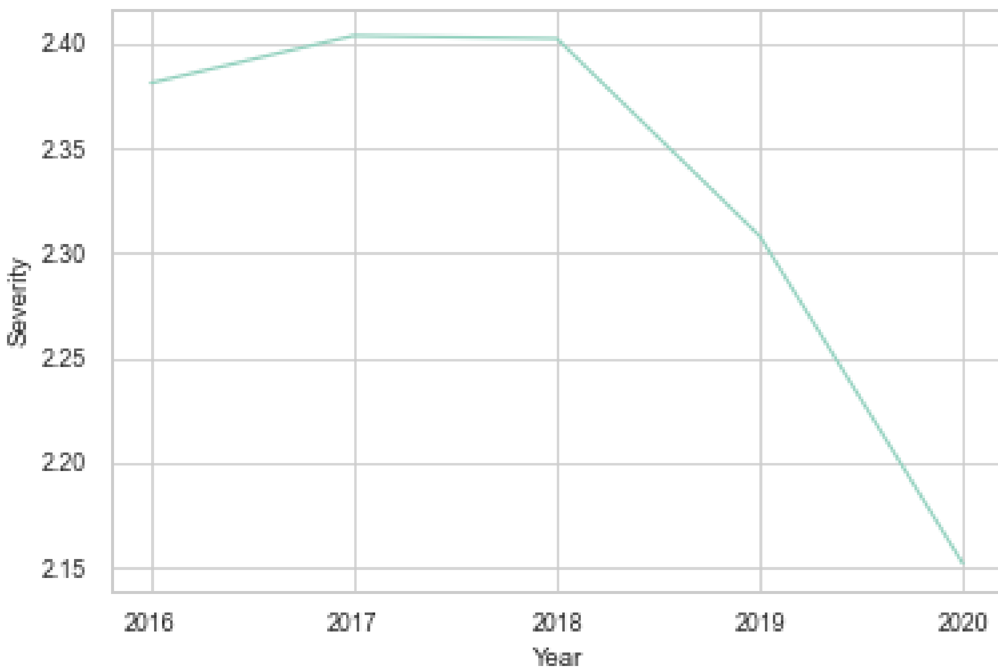
[19]:
```
sns.set_context('talk')
w = sns.catplot(x='Year',data=df1,kind='count')
w.fig.suptitle('weekday accidents cases',y=1.03)
w.set(ylabel='weekday cases')
plt.show()
```



[20]:
```
sns.set_context('paper')
h = sns.catplot(x='Hour',data=df1,kind='count')
h.fig.suptitle('Hourly accidents cases',y=1.03)
h.set(ylabel='hourly cases',xlabel='hour')
plt.annotate('morning peak',xy=(6,330000))
plt.annotate('afternoon peak',xy=(15,270000))
plt.annotate('bottom',xy=(1,25000))
plt.annotate('go to work',xy=(7.5,0),xytext=(1,125000),arrowprops={'arrowstyle':
↪'fancy'})
plt.annotate('get off work',xy=(17.
↪5,0),xytext=(19,150000),arrowprops={'arrowstyle':'fancy'})
plt.show()
```



[21]:
```
df1.groupby('Year')['Severity'].mean().plot(kind='line')
plt.xticks([2016,2017,2018,2019,2020])
plt.ylabel('Severity')
plt.show()
```



[22]:
```
dtype_df = df_clean.dtypes.reset_index()
dtype_df.columns = ["Count", "Column Type"]
dtype_df
```

[22]:

	Count	Column Type
0	ID	object
1	Severity	int64
2	Start_Time	datetime64[ns]
3	End_Time	datetime64[ns]
4	Start_Lat	float64
5	Start_Lng	float64
6	End_Lat	float64
7	End_Lng	float64
8	Distance(mi)	float64
9	Description	object
10	Number	float64
11	Street	object
12	Side	object
13	City	object
14	County	object
15	State	object
16	Zipcode	object
17	Country	object
18	Timezone	object

```
19             Airport_Code          object
20        Weather_Timestamp          object
21           Temperature(F)         float64
22             Wind_Chill(F)        float64
23               Humidity(%)        float64
24              Pressure(in)        float64
25            Visibility(mi)        float64
26            Wind_Direction         object
27            Wind_Speed(mph)       float64
28         Precipitation(in)        float64
29         Weather_Condition         object
30                   Amenity           bool
31                      Bump           bool
32                  Crossing           bool
33                  Give_Way           bool
34                  Junction           bool
35                   No_Exit           bool
36                   Railway           bool
37                Roundabout           bool
38                   Station           bool
39                      Stop           bool
40           Traffic_Calming           bool
41            Traffic_Signal           bool
42              Turning_Loop           bool
43            Sunrise_Sunset         object
44            Civil_Twilight         object
45         Nautical_Twilight         object
46     Astronomical_Twilight         object
47                     Month          int64
48                      Year          int64
49                      Hour          int64
50                   Weekday          int64
51                    Impact        float64
```

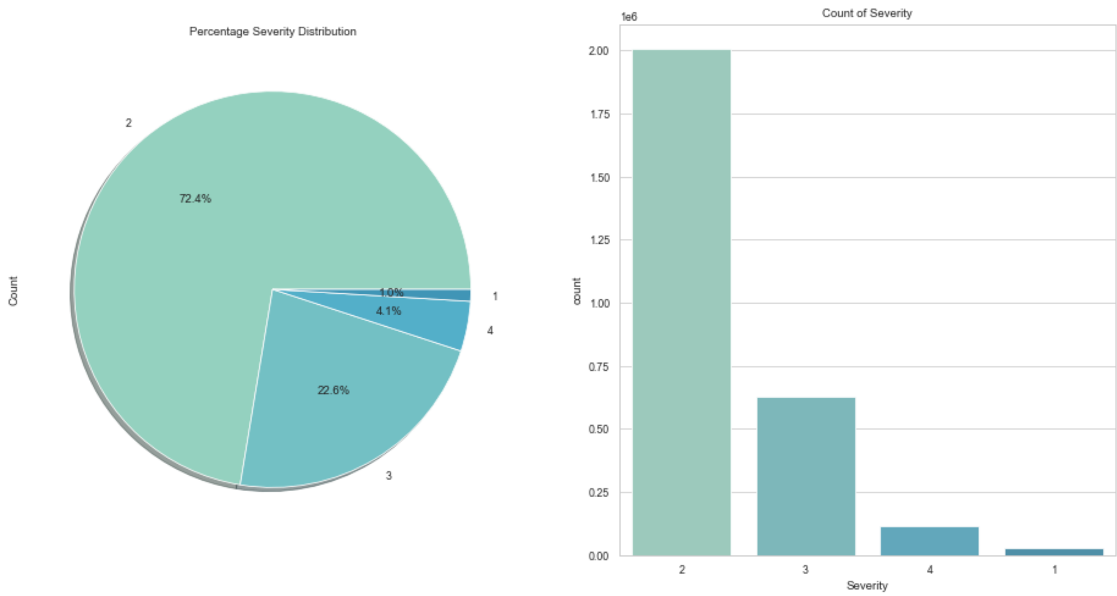
[23]:
```
dtype_df.groupby("Column Type").aggregate('count').reset_index()
```

[23]:
```
       Column Type  Count
0            int64      5
1   datetime64[ns]      2
2             bool     13
3          float64     14
4           object     18
```

[24]:
```
f,ax=plt.subplots(1,2,figsize=(16,8))
df_clean['Severity'].value_counts().plot.pie(autopct='%1.
 ↪1f%%',ax=ax[0],shadow=True)
ax[0].set_title('Percentage Severity Distribution')
```

```
ax[0].set_ylabel('Count')
sns.countplot('Severity',data=df_clean,ax=ax[1],order=df_clean['Severity'].
 ↪value_counts().index)
ax[1].set_title('Count of Severity')
plt.show()
```

```
C:\Users\Paul\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning:

Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
```



0.1 Machine Learning Algorithms

[25]:
```
# Convert Start_Time and End_Time to datetypes
train_df['Start_Time'] = pd.to_datetime(train_df['Start_Time'], errors='coerce')
train_df['End_Time'] = pd.to_datetime(train_df['End_Time'], errors='coerce')

# Extract year, month, day, hour and weekday
train_df['Year']=train_df['Start_Time'].dt.year
train_df['Month']=train_df['Start_Time'].dt.strftime('%b')
train_df['Day']=train_df['Start_Time'].dt.day
train_df['Hour']=train_df['Start_Time'].dt.hour
train_df['Weekday']=train_df['Start_Time'].dt.strftime('%a')
```

```
# Extract the amount of time in the unit of minutes for each accident, round to␣
↪the nearest integer
td='Time_Duration(min)'
train_df[td]=round((train_df['End_Time']-train_df['Start_Time'])/np.
↪timedelta64(1,'m'))
# Check if there is any negative time_duration values
train_df[td][train_df[td]<=0]

# Drop the rows with td<0
neg_outliers=train_df[td]<=0

# Set outliers to NAN
train_df[neg_outliers] = np.nan

# Drop rows with negative td
train_df.dropna(subset=[td],axis=0,inplace=True)

# Double check to make sure no more negative td
train_df[td][train_df[td]<=0]
```

[25]:
```
Series([], Name: Time_Duration(min), dtype: float64)
```

0.1.1 Deal with outliers

Fill outliers with median values

[26]:
```
# Remove outliers for Time_Duration(min): n * standard_deviation (n=3),␣
↪backfill with median

n=3

median = train_df[td].median()
std = train_df[td].std()
outliers = (train_df[td] - median).abs() > std*n

# Set outliers to NAN
train_df[outliers] = np.nan

# Fill NAN with median
train_df[td].fillna(median, inplace=True)

# train_df.info()
```

Select a list of features for machine learning algorithms

[27]:
```
# Set the list of features to include in Machine Learning
```

```
feature_lst=['Severity','Start_Lng','Start_Lat','Distance(mi)','Side','City','County','State',
↪'Visibility(mi)',␣
↪'Wind_Direction','Weather_Condition','Amenity','Bump','Crossing','Give_Way','Junction','No_
↪'Time_Duration(min)']
```

[28]:
```
df_sel=train_df[feature_lst].copy()
# Check missing values
df_sel.isnull().mean()

#drop na
df_sel.dropna(subset=df_sel.columns[df_sel.isnull().mean()!=0], how='any',␣
↪axis=0, inplace=True)
df_sel.shape
```

[28]:
```
(2800301, 32)
```

Select the state of interest: TX; and County of interest: Denton

Due to the limitation of personal laptop, the whole US dataset is too big to handle

[29]:
```
state='TX'

# Select the state of Texas
df_state=df_sel.loc[df_sel.State==state].copy()
df_state.drop('State',axis=1, inplace=True)
df_state.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 220074 entries, 12 to 2906605
Data columns (total 31 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   Severity           220074 non-null  float64
 1   Start_Lng          220074 non-null  float64
 2   Start_Lat          220074 non-null  float64
 3   Distance(mi)       220074 non-null  float64
 4   Side               220074 non-null  object
 5   City               220074 non-null  object
 6   County             220074 non-null  object
 7   Timezone           220074 non-null  object
 8   Temperature(F)     220074 non-null  float64
 9   Humidity(%)        220074 non-null  float64
 10  Pressure(in)       220074 non-null  float64
 11  Visibility(mi)     220074 non-null  float64
 12  Wind_Direction     220074 non-null  object
 13  Weather_Condition  220074 non-null  object
 14  Amenity            220074 non-null  float64
 15  Bump               220074 non-null  float64
 16  Crossing           220074 non-null  float64
```

```
 17  Give_Way           220074 non-null  float64
 18  Junction           220074 non-null  float64
 19  No_Exit            220074 non-null  float64
 20  Railway            220074 non-null  float64
 21  Roundabout         220074 non-null  float64
 22  Station            220074 non-null  float64
 23  Stop               220074 non-null  float64
 24  Traffic_Calming    220074 non-null  float64
 25  Traffic_Signal     220074 non-null  float64
 26  Turning_Loop       220074 non-null  float64
 27  Sunrise_Sunset     220074 non-null  object
 28  Hour               220074 non-null  float64
 29  Weekday            220074 non-null  object
 30  Time_Duration(min) 220074 non-null  float64
dtypes: float64(23), object(8)
memory usage: 53.7+ MB
```

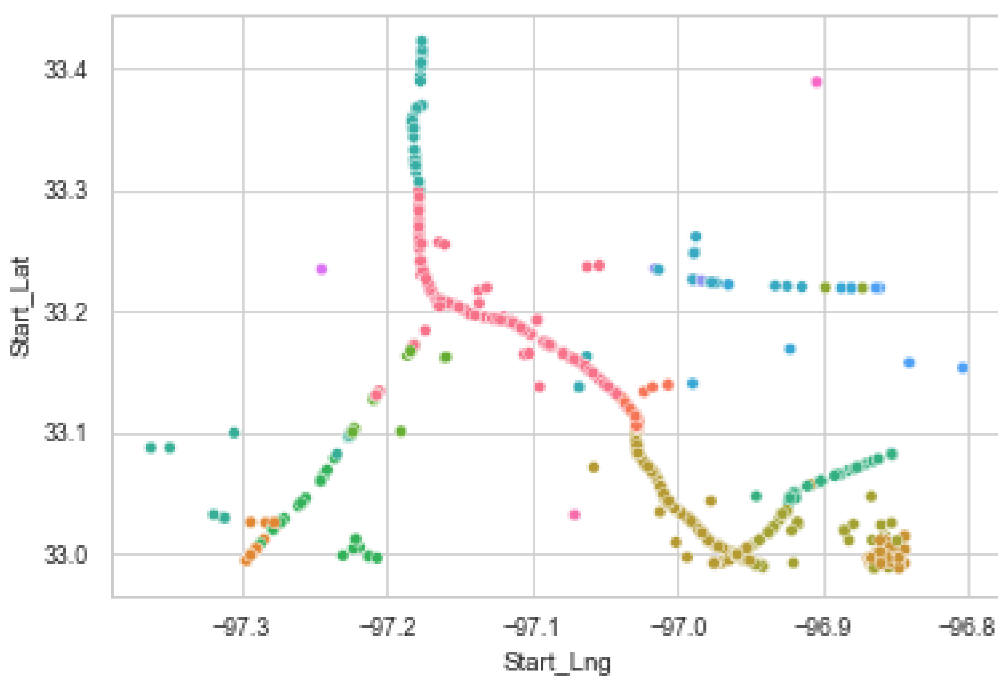
[30]:

```python
# Set county
county='Denton'

# Select the state of Pennsylvania
df_county=df_state.loc[df_state.County==county].copy()
df_county.drop('County',axis=1, inplace=True)
df_county.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2006 entries, 309 to 2906450
Data columns (total 30 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Severity            2006 non-null   float64
 1   Start_Lng           2006 non-null   float64
 2   Start_Lat           2006 non-null   float64
 3   Distance(mi)        2006 non-null   float64
 4   Side                2006 non-null   object
 5   City                2006 non-null   object
 6   Timezone            2006 non-null   object
 7   Temperature(F)      2006 non-null   float64
 8   Humidity(%)         2006 non-null   float64
 9   Pressure(in)        2006 non-null   float64
 10  Visibility(mi)      2006 non-null   float64
 11  Wind_Direction      2006 non-null   object
 12  Weather_Condition   2006 non-null   object
 13  Amenity             2006 non-null   float64
 14  Bump                2006 non-null   float64
 15  Crossing            2006 non-null   float64
 16  Give_Way            2006 non-null   float64
 17  Junction            2006 non-null   float64
```

```
 18  No_Exit              2006 non-null   float64
 19  Railway              2006 non-null   float64
 20  Roundabout           2006 non-null   float64
 21  Station              2006 non-null   float64
 22  Stop                 2006 non-null   float64
 23  Traffic_Calming      2006 non-null   float64
 24  Traffic_Signal       2006 non-null   float64
 25  Turning_Loop         2006 non-null   float64
 26  Sunrise_Sunset       2006 non-null   object
 27  Hour                 2006 non-null   float64
 28  Weekday              2006 non-null   object
 29  Time_Duration(min)   2006 non-null   float64
dtypes: float64(23), object(7)
memory usage: 485.8+ KB
```

[31]:
```
# Map of accidents, color code by city

sns.scatterplot(x='Start_Lng', y='Start_Lat', data=df_county, hue='City',␣
↪legend=False, s=20)
plt.show()
```



Deal with categorical data: pd.get_dummies()

[32]:
```
# Generate dummies for categorical data
df_county_dummy = pd.get_dummies(df_county,drop_first=True)
```

```
# Export data
# df_county_dummy.to_csv('./US_Accidents_May19_{}_dummy.csv'.
 ↪format(state),index=False)

df_county_dummy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2006 entries, 309 to 2906450
Columns: 113 entries, Severity to Weekday_Wed
dtypes: float64(23), uint8(90)
memory usage: 552.4 KB
```

Predict the accident severity with various supervised machine learning algorithms Data preparation: train_test_split

[33]:
```
# Assign the data
df=df_county_dummy

# Set the target for the prediction
target='Severity'


# 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
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=21, stratify=y)
```

[34]:
```
# List of classification algorithms
algo_lst=['Logistic Regression',' K-Nearest Neighbors','Decision Trees','Random␣
 ↪Forest']

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

Predict the accident severity with various supervised machine learning algorithms Algorithm A. Logistic regression

[35]:
```
# Logistic regression
lr = LogisticRegression(random_state=0)
lr.fit(X_train,y_train)
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: 0.689.
```

```
C:\Users\Paul\anaconda3\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
```

Algorithm B. The K-Nearest Neighbors (KNN) algorithm

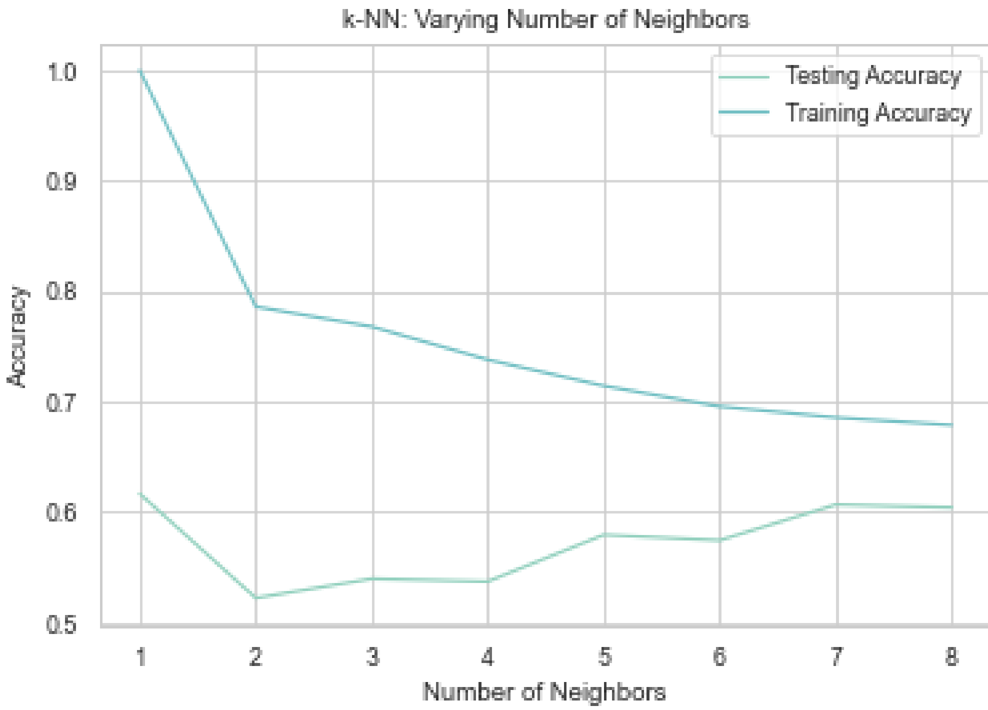
[36]:
```
# Create a k-NN classifier with 3 neighbors
knn = KNeighborsClassifier(n_neighbors=3)

# 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))
```

```
[K-Nearest Neighbors (KNN)] knn.score: 0.540.
[K-Nearest Neighbors (KNN)] accuracy_score: 0.540.
```

Optmize the number of neighors: plot the accuracy versus number of neighbors

[37]:

```python
# 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)

# 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()
```



Algorithm C. Decision Tree

[38]:
```
# Decision tree algorithm

# Instantiate dt_entropy, set 'entropy' as the information criterion
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))
```

```
[Decision Tree -- entropy] accuracy_score: 0.734.
[Decision Tree -- gini] accuracy_score: 0.746.
```

Algorithm D. Random Forest

[39]:
```
# 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("[Randon forest algorithm] accuracy_score: {:.3f}.".format(acc))
```

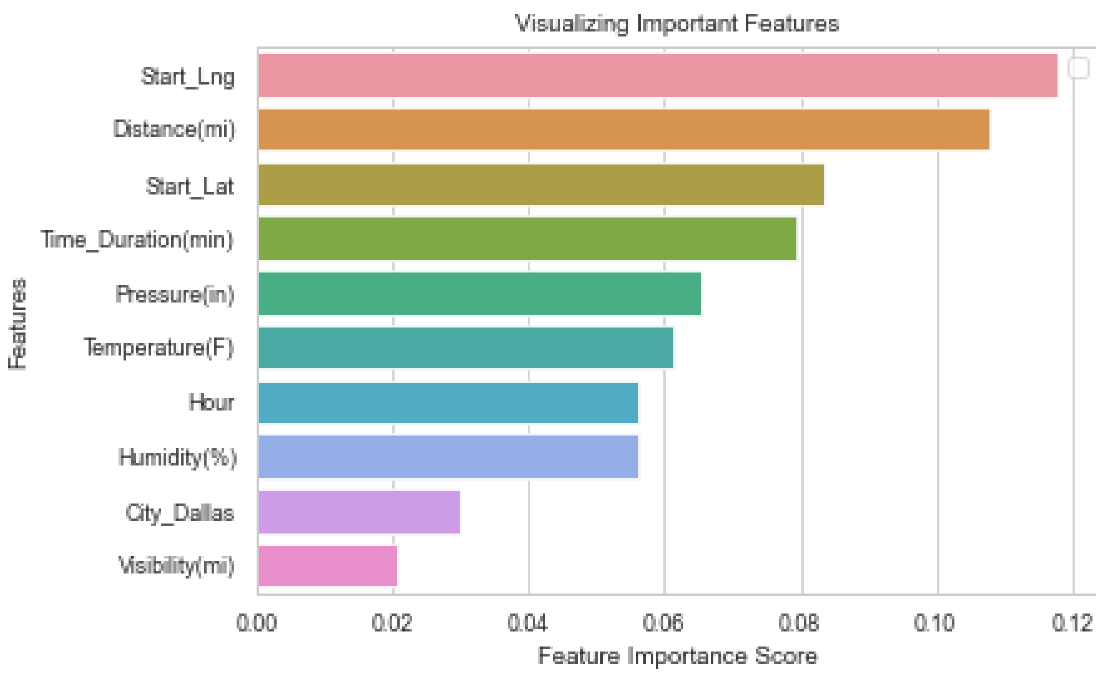
```
[Randon forest algorithm] accuracy_score: 0.781.
```

Algorithm D. Random Forest. Visualize important features

[40]:
```
feature_imp = pd.Series(clf.feature_importances_,index=X.columns).
↪sort_values(ascending=False)
```

```python
# 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()
```

No handles with labels found to put in legend.



[41]:
```python
# List top k important features
k=20
feature_imp.sort_values(ascending=False)[:k]
```

[41]:
```
Start_Lng              0.117871
Distance(mi)           0.107999
Start_Lat              0.083536
Time_Duration(min)     0.079458
Pressure(in)           0.065367
Temperature(F)         0.061386
Hour                   0.056313
Humidity(%)            0.056242
City_Dallas            0.029932
```

```
Visibility(mi)           0.020592
Sunrise_Sunset_Night     0.015713
City_Lewisville          0.015371
Side_R                   0.012284
City_Denton              0.010434
City_The Colony          0.010096
Weekday_Tue              0.009790
Weekday_Thu              0.009785
Weekday_Wed              0.009524
Junction                 0.008733
Weekday_Mon              0.008607
dtype: float64
```

Algorithm D. Random Forest. Select the top important features, set the threshold

[42]:
```
# 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])
```

```
Start_Lng
Start_Lat
Distance(mi)
Temperature(F)
Humidity(%)
Pressure(in)
Hour
Time_Duration(min)
City_Dallas
```

[43]:
```
# 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)
```

[43]:

```
RandomForestClassifier(n_jobs=-1, random_state=0)
```

[44]:

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

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

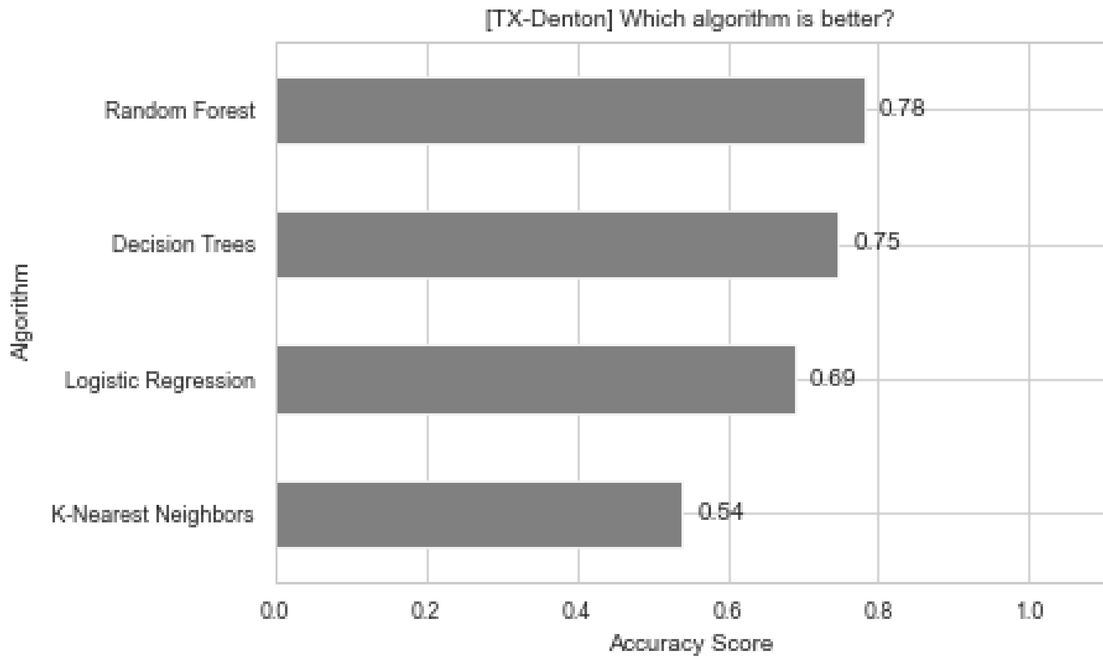
```
[Randon forest algorithm -- Full feature] accuracy_score: 0.781.
[Randon forest algorithm -- Limited feature] accuracy_score: 0.764.
```

Plot the accuracy score versus algorithm

[45]:

```
# 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)
# 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, lables, ticks and title
plt.xlim(0,1.1)
plt.xlabel('Accuracy Score')
```

```
plt.yticks(y_ticks, df_acc['Algorithm'], rotation=0)
plt.title('[{}-{}] Which algorithm is better?'.format(state, county))

plt.show()
```



[ ]:

[ ]: