```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
df = pd.read_csv("/content/drive/MyDrive/Dataset/US_Accidents_Dec21_updated.csv")
```

```
df.head()
```

	ID	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	End_Lat	End_Lng	Di
0	A-1	3	2016-02-08 00:37:08	2016-02-08 06:37:08	40.108910	-83.092860	40.112060	-83.031870	
1	A-2	2	2016-02-08 05:56:20	2016-02-08 11:56:20	39.865420	-84.062800	39.865010	-84.048730	
2	A-3	2	2016-02-08 06:15:39	2016-02-08 12:15:39	39.102660	-84.524680	39.102090	-84.523960	
3	A-4	2	2016-02-08 06:51:45	2016-02-08 12:51:45	41.062130	-81.537840	41.062170	-81.535470	
4	A-5	3	2016-02-08 07:53:43	2016-02-08 13:53:43	39.172393	-84.492792	39.170476	-84.501798	

5 rows × 47 columns

```
number_of_row = df.shape[0]
number_of_column = df.shape[1]
```

```
number_of_row
```

```
2845342
```

```
number_of_column
```

```
47
```

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2845342 entries, 0 to 2845341
Data columns (total 47 columns):
 #   Column                 Dtype  
---  ------                 -----  
 0   ID                     object 
 1   Severity               int64  
 2   Start_Time             object 
 3   End_Time               object 
 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 
dtypes: bool(13), float64(13), int64(1), object(20)
memory usage: 773.4+ MB
```

```
1
```

```
1 df.describe()
```

	Severity	Start_Lat	Start_Lng	End_Lat	End_Lng	Distance
count	2.845342e+06	2.845342e+06	2.845342e+06	2.845342e+06	2.845342e+06	2.845342
mean	2.137572e+00	3.624520e+01	-9.711463e+01	3.624532e+01	-9.711439e+01	7.026779
std	4.787216e-01	5.363797e+00	1.831782e+01	5.363873e+00	1.831763e+01	1.560361
min	1.000000e+00	2.456603e+01	-1.245481e+02	2.456601e+01	-1.245457e+02	0.000000
25%	2.000000e+00	3.344517e+01	-1.180331e+02	3.344628e+01	-1.180333e+02	5.200000
50%	2.000000e+00	3.609861e+01	-9.241808e+01	3.609799e+01	-9.241772e+01	2.440000
75%	2.000000e+00	4.016024e+01	-8.037243e+01	4.016105e+01	-8.037338e+01	7.640000
max	4.000000e+00	4.900058e+01	-6.711317e+01	4.907500e+01	-6.710924e+01	1.551860

Insights-

- This tells us that majority of the accidents (> 75%) had a severity rating of ~2 i.e. the traffic was not highly impacted.
- The average length of the road extent affected by the accidents was ~0.7 miles. So, there was almost no traffic disturbance.
- Accidents took place despite having average visibility of ~9 miles. This could mean that low visibility was not a reason.
- For about 75% of the accidents, no precipitation was reported. This could mean that rain was not a problem.

```
1 df.Start_Time = pd.to_datetime(df.Start_Time)
2 df.End_Time = pd.to_datetime(df.End_Time)
```

```
1 df['duration'] =  df['End_Time'] - df['Start_Time']
```

```
1 df['duration_minute'] = df['duration'].dt.total_seconds().div(60).astype(int)
```

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

	ID	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	End_Lat	End_Lng	Di
0	A-1	3	2016-02-08 00:37:08	2016-02-08 06:37:08	40.108910	-83.092860	40.112060	-83.031870	
1	A-2	2	2016-02-08 05:56:20	2016-02-08 11:56:20	39.865420	-84.062800	39.865010	-84.048730	
2	A-3	2	2016-02-08 06:15:39	2016-02-08 12:15:39	39.102660	-84.524680	39.102090	-84.523960	
3	A-4	2	2016-02-08 06:51:45	2016-02-08 12:51:45	41.062130	-81.537840	41.062170	-81.535470	
4	A-5	3	2016-02-08 07:53:43	2016-02-08 13:53:43	39.172393	-84.492792	39.170476	-84.501798	

5 rows × 49 columns

Double-click (or enter) to edit

```
fd =df
```

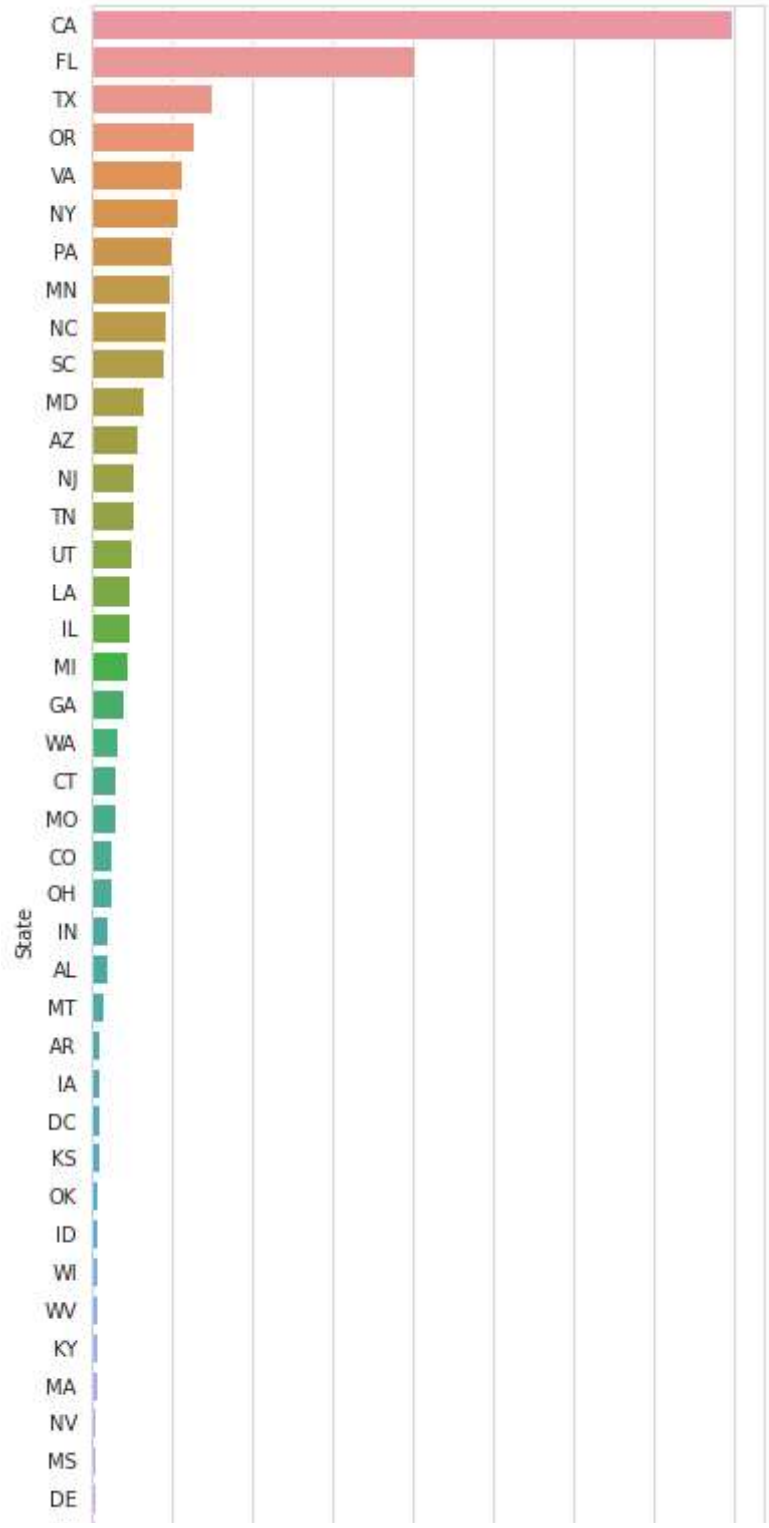
```
rr =fd
```

states with maximum accidents

```
import seaborn as sns
state_wise_counts = rr.groupby('State')['ID'].count().reset_index()
state_wise_counts = state_wise_counts.sort_values(by = "ID",ascending=False)
sns.set_style("whitegrid")
f, ax = plt.subplots(figsize=(6, 17))
sns.barplot(y="State", x="ID", data=state_wise_counts)

```

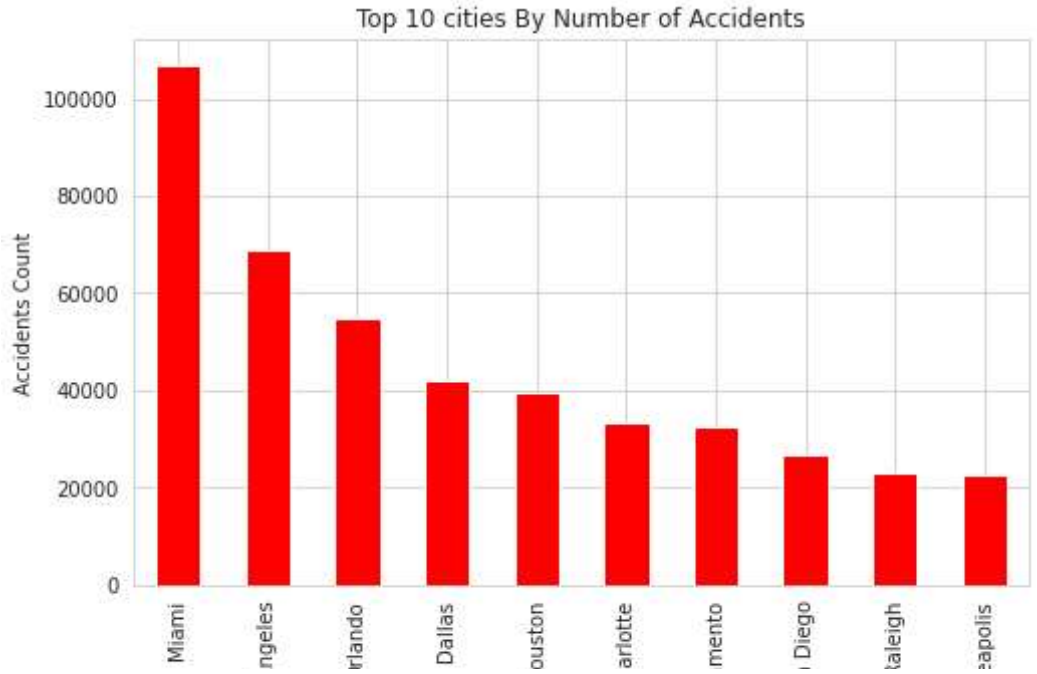
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f85bc4e0750>
```



City_wise count of accidents

```
1 city_wise_accidents = rr.City.value_counts()
2 fig, ax = plt.subplots(figsize=(8,5))
3 city_wise_accidents[:10].plot(kind='bar',color='red')
4 ax.set(title = 'Top 10 cities By Number of Accidents',
5        xlabel = 'Cities',
6        ylabel = 'Accidents Count')
```

```
[Text(0, 0.5, 'Accidents Count'),
 Text(0.5, 0, 'Cities'),
 Text(0.5, 1.0, 'Top 10 cities By Number of Accidents')]
```



Accident analysis by severity

```
accidents_severity = rr.groupby('Severity').count()['ID']
fig, ax = plt.subplots(figsize=(8, 6), subplot_kw=dict(aspect="equal"))
label = [1,2,3,4]
plt.pie(accidents_severity, labels=label,
        autopct='%1.1f%%', pctdistance=0.85)
circle = plt.Circle( (0,0), 0.7, color='white')
p=plt.gcf()
p.gca().add_artist(circle)
ax.set_title("Accident by Severity",fontdict={'fontsize': 16})
plt.tight_layout()
plt.show()
```

bold text

```
1 rr.head(1)
```

	ID	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	End_Lat	End_Lng	Dist
0	A-1	3	2016-02-08 00:37:08	2016-02-08 06:37:08	40.10891	-83.09286	40.11206	-83.03187	

1 rows × 49 columns

Traffic_calming is used to limit the speed of the vehicle.

Analysis shows that maximum accident takes place when there is no traffic calming

```
1 from pandas.core.reshape.pivot import crosstab
2 crosstb = pd.crosstab(rr.Traffic_Calming , rr.duration_minute , margins = True)
3 crosstb
```

duration_minute	2	3	4	5	6	7	8	9	10	11	...	1039752	115708
Traffic_Calming													
False	5	16	22	58	378	329	585	855	2056	3176	...	1	2
True	0	0	0	0	0	0	0	0	0	0	...	0	0
All	5	16	22	58	378	329	585	855	2056	3176	...	1	2

3 rows × 5780 columns

```
1 cross_signal = pd.crosstab(rr.Traffic_Signal , rr.duration_minute , margins = True)
2 cross_signal
```

duration_minute	2	3	4	5	6	7	8	9	10	11	...	1039752	115708
Traffic_Signal													
False	5	16	22	55	373	326	577	849	2029	3131	...	1	2
True	0	0	0	3	5	3	8	6	27	45	...	0	0
All	5	16	22	58	378	329	585	855	2056	3176	...	1	2

3 rows × 5780 columns

Popular belief that with increase in precipitation the severity of accidents or duration of incident and number of accidents increases but this cannot be varified here as the incidents with precipitation are very less compared to accidents with no precipitation. Analysis of this issue is futher dampened by lack of avaibility of precipitatiom distribution data.

```
rr["Precipitation(cm)"] = (rr["Precipitation(in)"]*2.54).round(decimals = 3)
crosstb = pd.crosstab(rr["Precipitation(cm)"] , rr.duration_minute , margins = True)
crosstb
```

duration_minute	2	3	4	5	6	7	8	9	10	11	...	679133	69479
Precipitation(cm)													
0.0	4	15	20	50	332	302	519	746	1819	2786	...	1	
0.025	0	1	0	1	6	6	7	10	30	35	...	0	
0.051	1	0	0	0	1	1	5	6	15	22	...	0	
0.076	0	0	0	0	2	1	1	6	16	17	...	0	
0.102	0	0	0	2	1	0	1	9	5	16	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	.
25.451	0	0	0	0	0	0	0	0	0	0	...	0	
25.527	0	0	0	0	0	0	0	0	0	0	...	0	
26.416	0	0	0	0	0	0	0	0	0	0	...	0	
60.96	0	0	0	0	0	0	0	0	0	0	...	0	
All	5	16	20	55	356	321	545	810	1966	2998	...	1	

231 rows × 5355 columns

```
Astronomical_Twilight = pd.crosstab(rr.Astronomical_Twilight , rr.duration_minute , mar
Astronomical_Twilight
```

duration_minute	2	3	4	5	6	7	8	9	10	11	...	1039752	1
Astronomical_Twilight													
Day	5	14	17	47	284	251	428	642	1673	2542	...	1	
Night	0	2	5	11	93	78	156	213	383	633	...	0	
All	5	16	22	58	377	329	584	855	2056	3175	...	1	

3 rows × 5758 columns

```
Nautical_Twilight = pd.crosstab(rr.Nautical_Twilight , rr.duration_minute , margins = T
Nautical_Twilight
```

duration_minute	2	3	4	5	6	7	8	9	10	11	...	1039752	11570
Nautical_Twilight													
Day	5	13	17	45	272	235	401	605	1590	2433	...	1	
Night	0	3	5	13	105	94	183	250	466	742	...	0	
All	5	16	22	58	377	329	584	855	2056	3175	...	1	

3 rows × 5758 columns

```
Civil_Twilight = pd.crosstab(rr.Civil_Twilight , rr.duration_minute , margins = True)
Civil_Twilight
```

duration_minute	2	3	4	5	6	7	8	9	10	11	...	1039752	115708
Civil_Twilight													
Day	4	11	13	45	247	218	367	554	1495	2286	...	1	
Night	1	5	9	13	130	111	217	301	561	889	...	0	
All	5	16	22	58	377	329	584	855	2056	3175	...	1	

3 rows × 5758 columns

```
Turning_Loop = pd.crosstab(rr.Turning_Loop , rr.duration_minute , margins = True)
Turning_Loop

```

duration_minute	2	3	4	5	6	7	8	9	10	11	...	1039752	115708
Turning_Loop													
False	5	16	22	58	378	329	585	855	2056	3176	...	1	2
All	5	16	22	58	378	329	585	855	2056	3176	...	1	2

2 rows × 5780 columns

```
Traffic_Signal = pd.crosstab(rr.Traffic_Signal , rr.duration_minute , margins = True)
Traffic_Signal
```

duration_minute	2	3	4	5	6	7	8	9	10	11	...	1039752	115708
Traffic_Signal													
False	5	16	22	55	373	326	577	849	2029	3131	...	1	2
True	0	0	0	3	5	3	8	6	27	45	...	0	0
All	5	16	22	58	378	329	585	855	2056	3176	...	1	2

3 rows × 5780 columns

```
Stop = pd.crosstab(rr.Stop , rr.duration_minute , margins = True)
Stop
```

duration_minute	2	3	4	5	6	7	8	9	10	11	...	1039752	115708
Stop													
False	5	16	22	58	376	329	585	854	2051	3171	...	1	2
True	0	0	0	0	2	0	0	1	5	5	...	0	0
All	5	16	22	58	378	329	585	855	2056	3176	...	1	2

3 rows × 5780 columns

```
Roundabout = pd.crosstab(rr.Roundabout , rr.duration_minute , margins = True)
Roundabout

```

duration_minute	2	3	4	5	6	7	8	9	10	11	...	1039752	115708
Roundabout													
False	5	16	22	58	378	329	585	855	2056	3176	...	1	2
True	0	0	0	0	0	0	0	0	0	0	...	0	0
All	5	16	22	58	378	329	585	855	2056	3176	...	1	2

```
No_Exit = pd.crosstab(rr.No_Exit , rr.duration_minute , margins = True)
No_Exit

```

duration_minute	2	3	4	5	6	7	8	9	10	11	...	1039752	115708
No_Exit													
False	5	16	22	58	378	329	585	855	2056	3172	...	1	2
True	0	0	0	0	0	0	0	0	0	4	...	0	0
All	5	16	22	58	378	329	585	855	2056	3176	...	1	2

3 rows × 5780 columns

```
Give_Way = pd.crosstab(rr.Give_Way , rr.duration_minute , margins = True)
Give_Way

```

duration_minute	2	3	4	5	6	7	8	9	10	11	...	1039752	115708
Give_Way													
False	5	16	22	58	378	329	585	855	2053	3176	...	1	2
True	0	0	0	0	0	0	0	0	3	0	...	0	0
All	5	16	22	58	378	329	585	855	2056	3176	...	1	2

3 rows × 5780 columns

```
Crossing = pd.crosstab(rr.Crossing , rr.duration_minute , margins = True)
Crossing

```

duration_minute	2	3	4	5	6	7	8	9	10	11	...	1039752	115708
Crossing													
False	5	16	22	56	375	327	578	849	2035	3158	...	1	2
True	0	0	0	2	3	2	7	6	21	18	...	0	0
All	5	16	22	58	378	329	585	855	2056	3176	...	1	2

3 rows × 5780 columns

```
Bump = pd.crosstab(rr.Bump , rr.duration_minute , margins = True)
Bump

```

duration_minute	2	3	4	5	6	7	8	9	10	11	...	1039752	115708
Bump													
False	5	16	22	58	378	329	585	855	2056	3176	...	1	2
True	0	0	0	0	0	0	0	0	0	0	...	0	0
All	5	16	22	58	378	329	585	855	2056	3176	...	1	2

3 rows × 5780 columns

```
Weather_Condition = pd.crosstab(rr.Weather_Condition , rr.duration_minute , margins = T
Weather_Condition

```

duration_minute	2	3	4	5	6	7	8	9	10	11	...	1037765	1039
Weather_Condition													
Blowing Dust	0	0	0	0	0	0	0	0	0	0	...	0	
Blowing Dust / Windy	0	0	0	0	0	0	0	0	0	0	...	0	
Blowing Sand	0	0	0	0	0	0	0	0	0	0		0	

```
rr.groupby(["Weather_Condition",'Severity']).size().reset_index(name='counts').max()
```

```
Weather_Condition    Wintry Mix / Windy
Severity                              4
counts                          1043277
dtype: object
```

```
# df[df['duration_minute'] < 60].count()
# df[df['duration_minute'] < 120].count()
# df[df['duration_minute'] < 180].count()
# df[df['duration_minute'] > 180].count()
```

```
df[df['duration_minute'] < 60].count()
```

```
ID                     618632
Severity               618632
Start_Time             618632
End_Time               618632
Start_Lat              618632
Start_Lng              618632
End_Lat                618632
End_Lng                618632
Distance(mi)           618632
Description            618632
Number                 171451
Street                 618632
Side                   618632
City                   618598
County                 618632
State                  618632
Zipcode                618252
Country                618632
Timezone               617977
Airport_Code           617083
Weather_Timestamp      609642
Temperature(F)         605794
Wind_Chill(F)          530349
Humidity(%)            605040
Pressure(in)           607822
Visibility(mi)         605230
Wind_Direction         603418
Wind_Speed(mph)        584357
Precipitation(in)      501694
Weather_Condition      605043
Amenity                618632
Bump                   618632
Crossing               618632
```

```
Give_Way                618632
Junction                618632
No_Exit                 618632
Railway                 618632
Roundabout              618632
Station                 618632
Stop                    618632
Traffic_Calming         618632
Traffic_Signal          618632
Turning_Loop            618632
Sunrise_Sunset          618434
Civil_Twilight          618434
Nautical_Twilight       618434
Astronomical_Twilight   618434
duration                618632
duration_minute         618632
Precipitation(cm)       501694
dtype: int64
```

```
df[df['duration_minute'] < 120].count()
```

```
ID                      1415006
Severity                1415006
Start_Time              1415006
End_Time                1415006
Start_Lat               1415006
Start_Lng               1415006
End_Lat                 1415006
End_Lng                 1415006
Distance(mi)            1415006
Description             1415006
Number                   513618
Street                  1415005
Side                    1415006
City                    1414949
County                  1415006
State                   1415006
Zipcode                 1414323
Country                 1415006
Timezone                1413426
Airport_Code            1410102
Weather_Timestamp       1389485
Temperature(F)          1381222
Wind_Chill(F)           1298986
Humidity(%)             1379275
Pressure(in)            1386154
Visibility(mi)          1380655
Wind_Direction          1375741
Wind_Speed(mph)         1356078
Precipitation(in)       1255209
Weather_Condition       1380272
Amenity                 1415006
Bump                    1415006
Crossing                1415006
Give_Way                1415006
Junction                1415006
No_Exit                 1415006
Railway                 1415006
Roundabout              1415006
```

```
Station                 1415006
Stop                    1415006
Traffic_Calming         1415006
Traffic_Signal          1415006
Turning_Loop            1415006
Sunrise_Sunset          1413709
Civil_Twilight          1413709
Nautical_Twilight       1413709
Astronomical_Twilight   1413709
duration                1415006
duration_minute         1415006
Precipitation(cm)       1255209
dtype: int64
```

```
df[df['duration_minute'] < 180].count()
```

```
ID                      2011902
Severity                2011902
Start_Time              2011902
End_Time                2011902
Start_Lat               2011902
Start_Lng               2011902
End_Lat                 2011902
End_Lng                 2011902
Distance(mi)            2011902
Description             2011902
Number                   808661
Street                  2011900
Side                    2011902
City                    2011821
County                  2011902
State                   2011902
Zipcode                 2011014
Country                 2011902
Timezone                2009741
Airport_Code            2005072
Weather_Timestamp       1974269
Temperature(F)          1962761
Wind_Chill(F)           1876346
Humidity(%)             1959976
Pressure(in)            1969913
Visibility(mi)          1962796
Wind_Direction          1955679
Wind_Speed(mph)         1935853
Precipitation(in)       1821896
Weather_Condition       1962301
Amenity                 2011902
Bump                    2011902
Crossing                2011902
Give_Way                2011902
Junction                2011902
No_Exit                 2011902
Railway                 2011902
Roundabout              2011902
Station                 2011902
Stop                    2011902
Traffic_Calming         2011902
Traffic_Signal          2011902
Turning_Loop            2011902
```

```
Sunrise_Sunset           2009933
Civil_Twilight           2009933
Nautical_Twilight        2009933
Astronomical_Twilight    2009933
duration                 2011902
duration_minute          2011902
Precipitation(cm)        1821896
dtype: int64
```

```
df[df['duration_minute'] > 180].count()
```

```
ID                       824064
Severity                 824064
Start_Time               824064
End_Time                 824064
Start_Lat                824064
Start_Lng                824064
End_Lat                  824064
End_Lng                  824064
Distance(mi)             824064
Description              824064
Number                   287759
Street                   824064
Side                     824064
City                     824008
County                   824064
State                    824064
Zipcode                  823635
Country                  824064
Timezone                 822568
Airport_Code             821362
Weather_Timestamp        811234
Temperature(F)           804250
Wind_Chill(F)            490415
Humidity(%)              803232
Pressure(in)             807147
Visibility(mi)           802946
Wind_Direction           806897
Wind_Speed(mph)          742560
Precipitation(in)        465059
Weather_Condition        803348
Amenity                  824064
Bump                     824064
Crossing                 824064
Give_Way                 824064
Junction                 824064
No_Exit                  824064
Railway                  824064
Roundabout               824064
Station                  824064
Stop                     824064
Traffic_Calming          824064
Traffic_Signal           824064
Turning_Loop             824064
Sunrise_Sunset           823215
Civil_Twilight           823215
Nautical_Twilight        823215
Astronomical_Twilight    823215
duration                 824064
```

```
duration_minute          824064
Precipitation(cm)        465059
dtype: int64
```

```
df[df['duration_minute'] > 1111180].count()
```

```
ID                       10
Severity                 10
Start_Time               10
End_Time                 10
Start_Lat                10
Start_Lng                10
End_Lat                  10
End_Lng                  10
Distance(mi)             10
Description              10
Number                    4
Street                   10
Side                     10
City                     10
County                   10
State                    10
Zipcode                  10
Country                  10
Timezone                 10
Airport_Code             10
Weather_Timestamp        10
Temperature(F)           10
Wind_Chill(F)             3
Humidity(%)              10
Pressure(in)             10
Visibility(mi)            9
Wind_Direction           10
Wind_Speed(mph)           6
Precipitation(in)         0
Weather_Condition         9
Amenity                  10
Bump                     10
Crossing                 10
Give_Way                 10
Junction                 10
No_Exit                  10
Railway                  10
Roundabout               10
Station                  10
Stop                     10
Traffic_Calming          10
Traffic_Signal           10
Turning_Loop             10
Sunrise_Sunset           10
Civil_Twilight           10
Nautical_Twilight        10
Astronomical_Twilight    10
duration                 10
duration_minute          10
Precipitation(cm)         0
dtype: int64
```

```
gk = df.groupby('Severity')
```

```
gk.first()
```

Severity	ID	Start_Time	End_Time	Start_Lat	Start_Lng	End_Lat	End_Lng
1	A-1735389	2020-08-03 14:35:16	2020-08-03 15:20:16	32.206760	-110.980500	32.206760	-110.98050(
2	A-2	2016-02-08 05:56:20	2016-02-08 11:56:20	39.865420	-84.062800	39.865010	-84.04873(
3	A-1	2016-02-08 00:37:08	2016-02-08 06:37:08	40.108910	-83.092860	40.112060	-83.03187(
4	A-20	2016-02-08 21:00:17	2016-02-09 03:00:17	41.679361	-83.573037	41.666124	-83.56633!

4 rows × 49 columns

```
1
```

```
# Test1 = df[(df['duration_minute'] <=60)]
# Test2 = df[(df['duration_minute'] >60) & (df['duration_minute'] <=120)]
# Test3 = df[(df['duration_minute'] >120) & (df['duration_minute'] <=180)]
# Test4 = df[(df['duration_minute'] >180)]


```

```
df['duration_minute'].max()
```

```
1682579
```

```
 df['duration_minute'].idxmax()
```

```
415412
```

```
df['Start_Time'][415412]
```

```
Timestamp('2018-09-28 07:55:52')
```

```
df['End_Time'][415412]
```

```
Timestamp('2021-12-09 18:55:09')
```

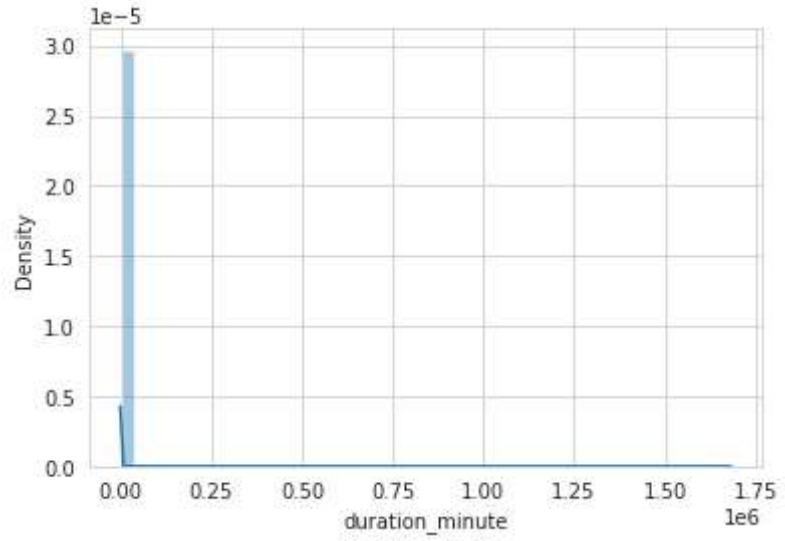
```
df['duration_minute'].min()
```

```
2
```

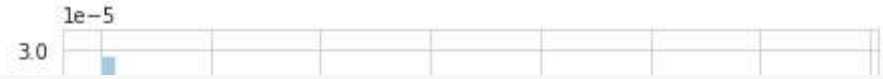
```
import seaborn as sns
```

```
sns.distplot(df['duration_minute'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8578ac0350>
```




```
import warnings
warnings.filterwarnings('ignore')
plt.figure(figsize=(16,5))
plt.subplot(1,2,1)
sns.distplot(df['duration_minute'])
plt.show()
```




```
df['duration_minute'].describe()
```

```
count    2.845342e+06
mean     3.586833e+02
std      9.329923e+03
min      2.000000e+00
25%      7.300000e+01
50%      1.200000e+02
75%      2.230000e+02
max      1.682579e+06
Name: duration_minute, dtype: float64
```

```
Q1 = df.duration_minute.quantile(0.25)
Q3 = df.duration_minute.quantile(0.75)
Q1,Q3
```

```
(73.0, 223.0)
```

```
IQR = Q3 - Q1
IQR
```

```
150.0
```

```
lower_limit = Q1 - 1.5*IQR
upper_limit = Q3 + 1.5*IQR
lower_limit, upper_limit
```

```
(-152.0, 448.0)
```

```
df[(df.duration_minute<lower_limit)|(df.duration_minute>upper_limit)]
```

	ID	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	End_Lat
136	A-137	2	2016-02-12 10:07:44	2016-02-12 19:07:44	41.468220	-81.947600	41.468660
512	A-513	3	2016-02-26 12:55:26	2016-02-26 21:07:14	41.675710	-83.693850	41.675710
587	A-588	3	2016-03-02 05:10:40	2016-03-02 19:10:40	38.077590	-84.457120	38.100840
1668	A-1669	2	2016-04-03 15:40:55	2016-04-06 23:41:50	38.376860	-122.202290	38.445560
1670	A-1671	2	2016-04-03 15:40:56	2016-04-06 23:41:50	38.445560	-122.196610	38.376860
...	...	...	...	...	...	...	...
2845187	A-2845188	2	2019-08-23 09:45:12	2019-08-26 13:45:00	45.561531	-117.917390	45.560740
2845199	A-2845200	2	2019-08-23 17:19:55	2019-08-24 03:11:30	44.119763	-121.317222	44.119050

```
1
```

08:57:00

```
1 new_df = df[(df.duration_minute>lower_limit)&(df.duration_minute<upper_limit)]
```

2019-08-23 2019-08-

```
1 new_df['duration_minute'].describe()
```

```
count    2.705900e+06
mean     1.422295e+02
std      1.103949e+02
min      2.000000e+00
25%      6.900000e+01
50%      1.130000e+02
75%      1.820000e+02
max      4.470000e+02
Name: duration_minute, dtype: float64
```

```
1 df['duration_minute'].describe()
```

```
count    2.845342e+06
mean     3.586833e+02
std      9.329923e+03
```

```
min      2.000000e+00
25%      7.300000e+01
50%      1.200000e+02
75%      2.230000e+02
max      1.682579e+06
Name: duration_minute, dtype: float64
```

```
df.shape[0] - new_df.shape[0]
```

```
139442
```

```
df.shape
```

```
(2845342, 50)
```

```
new_df.isnull().any()
```

```
ID                     False
Severity               False
Start_Time             False
End_Time               False
Start_Lat              False
Start_Lng              False
End_Lat                False
End_Lng                False
Distance(mi)           False
Description            False
Number                  True
Street                  True
Side                   False
City                    True
County                 False
State                  False
Zipcode                 True
Country                False
Timezone                True
Airport_Code            True
Weather_Timestamp       True
Temperature(F)          True
Wind_Chill(F)           True
Humidity(%)             True
Pressure(in)            True
Visibility(mi)          True
Wind_Direction          True
Wind_Speed(mph)         True
Precipitation(in)       True
Weather_Condition       True
Amenity                False
Bump                   False
Crossing               False
Give_Way               False
Junction               False
No_Exit                False
Railway                False
Roundabout             False
Station                False
Stop                   False
```

```
Traffic_Calming          False
Traffic_Signal           False
Turning_Loop             False
Sunrise_Sunset            True
Civil_Twilight            True
Nautical_Twilight         True
Astronomical_Twilight     True
duration                 False
duration_minute          False
Precipitation(cm)         True
dtype: bool
```

```
percent_missing = df.isnull().sum() * 100 / len(df)
missing_value_df = pd.DataFrame({'column_name': df.columns,
                                 'percent_missing': percent_missing})
```

```
missing_value_df
```

	column_name	percent_missing
ID	ID	0.000000
Severity	Severity	0.000000
Start_Time	Start_Time	0.000000
End_Time	End_Time	0.000000
Start_Lat	Start_Lat	0.000000
Start_Lng	Start_Lng	0.000000
End_Lat	End_Lat	0.000000
End_Lng	End_Lng	0.000000
Distance(mi)	Distance(mi)	0.000000

```
new_df.drop(['Precipitation(in)', 'Wind_Chill(F)','Number'], axis = 1,inplace = True)
```

```
percent_missing = new_df.isnull().sum() * 100 / len(new_df)
missing_value_df = pd.DataFrame({'column_name': new_df.columns,
                                 'percent_missing': percent_missing})
```

```
missing_value_df
```

	column_name	percent_missing
ID	ID	0.000000
Severity	Severity	0.000000
Start_Time	Start_Time	0.000000
End_Time	End_Time	0.000000
Start_Lat	Start_Lat	0.000000
Start_Lng	Start_Lng	0.000000
End_Lat	End_Lat	0.000000
End_Lng	End_Lng	0.000000
Distance(mi)	Distance(mi)	0.000000
Description	Description	0.000000
Street	Street	0.000074
Side	Side	0.000000
City	City	0.004693

```
dff = new_df.dropna()
dff
```

	ID	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	End_Lat
0	A-1	3	2016-02-08 00:37:08	2016-02-08 06:37:08	40.108910	-83.092860	40.112060
4	A-5	3	2016-02-08 07:53:43	2016-02-08 13:53:43	39.172393	-84.492792	39.170476
7	A-8	2	2016-02-08 11:51:46	2016-02-08 17:51:46	41.375310	-81.820170	41.367860
9	A-10	2	2016-02-08 15:16:43	2016-02-08 21:16:43	40.109310	-82.968490	40.110780
10	A-11	2	2016-02-08 15:43:50	2016-02-08 21:43:50	39.192880	-84.477230	39.196150

```
1 dff.describe()
```

	Severity	Start_Lat	Start_Lng	End_Lat	End_Lng	Distance
count	2.111263e+06	2.111263e+06	2.111263e+06	2.111263e+06	2.111263e+06	2.111263
mean	2.081831e+00	3.613390e+01	-9.671673e+01	3.613405e+01	-9.671654e+01	6.78659
std	3.959759e-01	5.389858e+00	1.827758e+01	5.389964e+00	1.827742e+01	1.441641
min	1.000000e+00	2.456603e+01	-1.245481e+02	2.456601e+01	-1.245457e+02	0.000000
25%	2.000000e+00	3.324983e+01	-1.179906e+02	3.324876e+01	-1.179919e+02	4.20000
50%	2.000000e+00	3.585001e+01	-9.112819e+01	3.585079e+01	-9.113061e+01	2.07000
75%	2.000000e+00	4.009866e+01	-8.033706e+01	4.009820e+01	-8.033661e+01	7.69000
max	4.000000e+00	4.900058e+01	-6.748413e+01	4.907500e+01	-6.748413e+01	1.55186

```
1 dff.isnull().any()
```

```
ID                  False
Severity            False
Start_Time          False
End_Time            False
Start_Lat           False
Start_Lng           False
```

```
End_Lat                 False
End_Lng                 False
Distance(mi)            False
Description             False
Street                  False
Side                    False
City                    False
County                  False
State                   False
Zipcode                 False
Country                 False
Timezone                False
Airport_Code            False
Weather_Timestamp       False
Temperature(F)          False
Humidity(%)             False
Pressure(in)            False
Visibility(mi)          False
Wind_Direction          False
Wind_Speed(mph)         False
Weather_Condition       False
Amenity                 False
Bump                    False
Crossing                False
Give_Way                False
Junction                False
No_Exit                 False
Railway                 False
Roundabout              False
Station                 False
Stop                    False
Traffic_Calming         False
Traffic_Signal          False
Turning_Loop            False
Sunrise_Sunset          False
Civil_Twilight          False
Nautical_Twilight       False
Astronomical_Twilight   False
duration                False
duration_minute         False
Precipitation(cm)       False
dtype: bool
```

```
1 dff.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2111263 entries, 0 to 2845341
Data columns (total 47 columns):
 #   Column                Dtype
---  ------                -----
 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  Street                 object
 11  Side                   object
 12  City                   object
 13  County                 object
 14  State                  object
 15  Zipcode                object
 16  Country                object
 17  Timezone               object
 18  Airport_Code           object
 19  Weather_Timestamp      object
 20  Temperature(F)         float64
 21  Humidity(%)            float64
 22  Pressure(in)           float64
 23  Visibility(mi)         float64
 24  Wind_Direction         object
 25  Wind_Speed(mph)        float64
 26  Weather_Condition      object
 27  Amenity                bool
 28  Bump                   bool
 29  Crossing               bool
 30  Give_Way               bool
 31  Junction               bool
 32  No_Exit                bool
 33  Railway                bool
 34  Roundabout             bool
 35  Station                bool
 36  Stop                   bool
 37  Traffic_Calming        bool
 38  Traffic_Signal         bool
 39  Turning_Loop           bool
 40  Sunrise_Sunset         object
 41  Civil_Twilight         object
 42  Nautical_Twilight      object
 43  Astronomical_Twilight  object
 44  duration               timedelta64[ns]
 45  duration_minute        int64
 46  Precipitation(cm)      float64
dtypes: bool(13), datetime64[ns](2), float64(11), int64(2), object(18), timedelta64[r
memory usage: 589.9+ MB
```

```
X = dff[['Amenity','Bump','Crossing','Give_Way','Junction','No_Exit','Railway','Roundab
Y = dff[['duration_minute']]
```

```
X
```

	Amenity	Bump	Crossing	Give_Way	Junction	No_Exit	Railway	Roundabout
0	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False
7	False	False	False	False	True	False	False	False
9	False	False	False	False	False	False	False	False
10	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...
2845337	False	False	False	False	False	False	False	False
2845338	False	False	False	False	False	False	False	False

```
X_new = X.copy()                                    # Create copy of DataFrame
X_new = X_new.astype(int)
```

	Amenity	Bump	Crossing	Give_Way	Junction	No_Exit	Railway	Roundabout
2845341	False	False	False	False	False	False	False	False

```
X_new
```

	Amenity	Bump	Crossing	Give_Way	Junction	No_Exit	Railway	Roundabout
0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
7	0	0	0	0	1	0	0	0
9	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...
2845337	0	0	0	0	0	0	0	0
2845338	0	0	0	0	0	0	0	0
2845339	0	0	0	0	1	0	0	0
2845340	0	0	0	0	0	0	0	0
2845341	0	0	0	0	0	0	0	0

2111263 rows × 13 columns

```
1
```

```
n1 = dff[dff['duration_minute'] < 60].count()
n2 = dff[dff['duration_minute'] < 120].count()
n3 = dff[dff['duration_minute'] < 180].count()
```

```
n4 = dff[dff['duration_minute'] > 180].count()
n1,n2,n3,n4
```

```
(ID                      489864
 Severity                489864
 Start_Time              489864
 End_Time                489864
 Start_Lat               489864
 Start_Lng               489864
 End_Lat                 489864
 End_Lng                 489864
 Distance(mi)            489864
 Description             489864
 Street                  489864
 Side                    489864
 City                    489864
 County                  489864
 State                   489864
 Zipcode                 489864
 Country                 489864
 Timezone                489864
 Airport_Code            489864
 Weather_Timestamp       489864
 Temperature(F)          489864
 Humidity(%)             489864
 Pressure(in)            489864
 Visibility(mi)          489864
 Wind_Direction          489864
 Wind_Speed(mph)         489864
 Weather_Condition       489864
 Amenity                 489864
 Bump                    489864
 Crossing                489864
 Give_Way                489864
 Junction                489864
 No_Exit                 489864
 Railway                 489864
 Roundabout              489864
 Station                 489864
 Stop                    489864
 Traffic_Calming         489864
 Traffic_Signal          489864
 Turning_Loop            489864
 Sunrise_Sunset          489864
 Civil_Twilight          489864
 Nautical_Twilight       489864
 Astronomical_Twilight   489864
 duration                489864
 duration_minute         489864
 Precipitation(cm)       489864
 dtype: int64, ID                    1227064
 Severity                1227064
 Start_Time              1227064
 End_Time                1227064
 Start_Lat               1227064
 Start_Lng               1227064
 End_Lat                 1227064
 End_Lng                 1227064
 Distance(mi)            1227064
 Description             1227064
```

```
Street                    1227064
```

```
Test1 = dff[(dff['duration_minute'] <=60)]
Test2 = dff[(dff['duration_minute'] >60) & (df['duration_minute'] <=120)]
Test3 = dff[(dff['duration_minute'] >120) & (df['duration_minute'] <=180)]
Test4 = dff[(dff['duration_minute'] >180)]
```

```
Test1
```

	ID	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	End_Lat
46848	A-46849	2	2017-01-24 07:14:00	2017-01-24 08:14:00	43.981403	-69.944528	43.976240
46900	A-46901	2	2017-01-24 08:20:00	2017-01-24	43.783670	-70.192130	43.785240

```
1
```

Regression

```
1 from sklearn import linear_model
2 from sklearn.model_selection import train_test_split
3 import sklearn.metrics as metrics
```

Training all dataset at once

Double-click (or enter) to edit

```
1  x_train, x_test, y_train, y_test = train_test_split(X_new, Y)
```

```
1 reg = linear_model.LinearRegression()
2 reg.fit(x_train,y_train)
```

```
LinearRegression()
```

```
1 reg = linear_model.LinearRegression()
2 reg.fit(x_train,y_train)
3 y_predicted = reg.predict(x_test)
4
5 mae = metrics.mean_absolute_error(y_test, y_predicted)
6 mse = metrics.mean_squared_error(y_test, y_predicted)
7 rmse = np.sqrt(mse) # or mse**(0.5)
8 r2 = metrics.r2_score(y_test,y_predicted)
9
10
11 print("MAE:",mae)
12 print("MSE:", mse)
13 print("RMSE:", rmse)
14 print("R-Squared:", r2)
```

```
MAE: 58.45082544977815
MSE: 6460.636556727899
RMSE: 80.37808505263048
R-Squared: 0.005825196434314561
```

```
1 reg.coef_
```

```
array([[ -1.73090608,  -1.16875905,  -1.78156591,  -3.37989047,
         -6.1734658 ,  -1.65267486,  -1.00017663,   7.09349909,
          1.33487689,   3.30416334,   0.06733694, -18.95471405,
          0.        ]])
```

```
reg.intercept_
```

```
array([117.63957431])
```

```
y_predicted = reg.predict(x_test)
```

```
mae = metrics.mean_absolute_error(y_test, y_predicted)
mse = metrics.mean_squared_error(y_test, y_predicted)
rmse = np.sqrt(mse) # or mse**(0.5)
r2 = metrics.r2_score(y_test,y_predicted)
```

```
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
MAE: 58.45082544977815
MSE: 6460.636556727899
RMSE: 80.37808505263048
R-Squared: 0.005825196434314561
```

```
dff.describe()
```

	Severity	Start_Lat	Start_Lng	End_Lat	End_Lng	Distance
count	2.111263e+06	2.111263e+06	2.111263e+06	2.111263e+06	2.111263e+06	2.111263
mean	2.081831e+00	3.613390e+01	-9.671673e+01	3.613405e+01	-9.671654e+01	6.78659
std	3.959759e-01	5.389858e+00	1.827758e+01	5.389964e+00	1.827742e+01	1.441641
min	1.000000e+00	2.456603e+01	-1.245481e+02	2.456601e+01	-1.245457e+02	0.000000
25%	2.000000e+00	3.324983e+01	-1.179906e+02	3.324876e+01	-1.179919e+02	4.200000
50%	2.000000e+00	3.585001e+01	-9.112819e+01	3.585079e+01	-9.113061e+01	2.070000
75%	2.000000e+00	4.009866e+01	-8.033706e+01	4.009820e+01	-8.033661e+01	7.690000
max	4.000000e+00	4.900058e+01	-6.748413e+01	4.907500e+01	-6.748413e+01	1.551860

```
1
```

training each group separately

```
X1 = Test1[['Amenity','Bump','Crossing','Give_Way','Junction','No_Exit','Railway','Roun
Y1 = Test1[['duration_minute']]

X_new1 = X1.copy()                                    # Create copy of DataFrame
X_new1 = X_new1.astype(int)

x_train, x_test, y_train, y_test = train_test_split(X_new1, Y1)

reg = linear_model.LinearRegression()
reg.fit(x_train,y_train)
y_predicted = reg.predict(x_test)

mae = metrics.mean_absolute_error(y_test, y_predicted)
mse = metrics.mean_squared_error(y_test, y_predicted)
rmse = np.sqrt(mse) # or mse**(0.5)
r2 = metrics.r2_score(y_test,y_predicted)


print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
MAE: 10.78432889984507
MSE: 179.0007786699877
RMSE: 13.379117260491729
R-Squared: 0.0030637670205264644
```

```
X2 = Test2[['Amenity','Bump','Crossing','Give_Way','Junction','No_Exit','Railway','Roun
Y2 = Test2[['duration_minute']]

X_new2 = X2.copy()                                    # Create copy of DataFrame
X_new2 = X_new2.astype(int)

x_train, x_test, y_train, y_test = train_test_split(X_new2, Y2)

reg = linear_model.LinearRegression()
reg.fit(x_train,y_train)
y_predicted = reg.predict(x_test)

mae = metrics.mean_absolute_error(y_test, y_predicted)
mse = metrics.mean_squared_error(y_test, y_predicted)
rmse = np.sqrt(mse) # or mse**(0.5)
r2 = metrics.r2_score(y_test,y_predicted)


print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
MAE: 13.740660884984367
MSE: 247.01891959917765
RMSE: 15.71683554660981
R-Squared: 0.0029455295424258843
```

```
X3 = Test3[['Amenity','Bump','Crossing','Give_Way','Junction','No_Exit','Railway','Roun
Y3 = Test3[['duration_minute']]

X_new3 = X3.copy()                                    # Create copy of DataFrame
X_new3 = X_new3.astype(int)

x_train, x_test, y_train, y_test = train_test_split(X_new3, Y3)

reg = linear_model.LinearRegression()
reg.fit(x_train,y_train)
y_predicted = reg.predict(x_test)

mae = metrics.mean_absolute_error(y_test, y_predicted)
mse = metrics.mean_squared_error(y_test, y_predicted)
rmse = np.sqrt(mse) # or mse**(0.5)
r2 = metrics.r2_score(y_test,y_predicted)


print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
MAE: 14.236097868267482
MSE: 286.43591652445025
RMSE: 16.924417760279088
R-Squared: 0.004641644885399421
```

```
X4 = Test4[['Amenity','Bump','Crossing','Give_Way','Junction','No_Exit','Railway','Roun
Y4 = Test4[['duration_minute']]

X_new4 = X4.copy()                                    # Create copy of DataFrame
X_new4 = X_new4.astype(int)

x_train, x_test, y_train, y_test = train_test_split(X_new4, Y4)

reg = linear_model.LinearRegression()
reg.fit(x_train,y_train)
y_predicted = reg.predict(x_test)

mae = metrics.mean_absolute_error(y_test, y_predicted)
mse = metrics.mean_squared_error(y_test, y_predicted)
rmse = np.sqrt(mse) # or mse**(0.5)
r2 = metrics.r2_score(y_test,y_predicted)


print("MAE:",mae)
print("MSE:", mse)
```

```
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
MAE: 59.166567710783234
MSE: 4805.799103615084
RMSE: 69.32387109513637
R-Squared: 0.011171424613769076
```

```

```

```

```

Decision Tree

```
from sklearn import tree

```

```
X1 = Test1[['Amenity','Bump','Crossing','Give_Way','Junction','No_Exit','Railway','Roun
Y1 = Test1[['duration_minute']]

X_new1 = X1.copy()                                    # Create copy of DataFrame
X_new1 = X_new1.astype(int)

x_train, x_test, y_train, y_test = train_test_split(X_new1, Y1)

model = tree.DecisionTreeClassifier()
model.fit(x_train,y_train)
y_predicted = model.predict(x_test)

mae = metrics.mean_absolute_error(y_test, y_predicted)
mse = metrics.mean_squared_error(y_test, y_predicted)
rmse = np.sqrt(mse) # or mse**(0.5)



print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)

```

```
MAE: 10.520787492762015
MSE: 209.72602200347424
RMSE: 14.481920521929204
```

```
X2 = Test2[['Amenity','Bump','Crossing','Give_Way','Junction','No_Exit','Railway','Roun
Y2 = Test2[['duration_minute']]

X_new2 = X2.copy()                                    # Create copy of DataFrame
X_new2 = X_new2.astype(int)

```

```
x_train, x_test, y_train, y_test = train_test_split(X_new2, Y2)

model = tree.DecisionTreeClassifier()
model.fit(x_train,y_train)
y_predicted = model.predict(x_test)

mae = metrics.mean_absolute_error(y_test, y_predicted)
mse = metrics.mean_squared_error(y_test, y_predicted)
rmse = np.sqrt(mse) # or mse**(0.5)



print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)

```

```
MAE: 15.029238360729277
MSE: 445.8178570284213
RMSE: 21.114399281732393
```

```
X3 = Test3[['Amenity','Bump','Crossing','Give_Way','Junction','No_Exit','Railway','Roun
Y3 = Test3[['duration_minute']]

X_new3 = X3.copy()                                    # Create copy of DataFrame
X_new3 = X_new3.astype(int)

x_train, x_test, y_train, y_test = train_test_split(X_new3, Y3)

model = tree.DecisionTreeClassifier()
model.fit(x_train,y_train)
y_predicted = model.predict(x_test)

mae = metrics.mean_absolute_error(y_test, y_predicted)
mse = metrics.mean_squared_error(y_test, y_predicted)
rmse = np.sqrt(mse) # or mse**(0.5)



print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)

```

```
MAE: 18.657983738596023
MSE: 633.4482116272856
RMSE: 25.16839708100787
```

```
X4 = Test4[['Amenity','Bump','Crossing','Give_Way','Junction','No_Exit','Railway','Roun
Y4 = Test4[['duration_minute']]

X_new4 = X4.copy()                                    # Create copy of DataFrame
X_new4 = X_new4.astype(int)
```

```
x_train, x_test, y_train, y_test = train_test_split(X_new4, Y4)

model = tree.DecisionTreeClassifier()
model.fit(x_train,y_train)
y_predicted = model.predict(x_test)

mae = metrics.mean_absolute_error(y_test, y_predicted)
mse = metrics.mean_squared_error(y_test, y_predicted)
rmse = np.sqrt(mse) # or mse**(0.5)



print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)

```

```
MAE: 59.791825415662124
MSE: 6481.858506239616
RMSE: 80.50999010209613
```

```

```

Deep Neural Network

```
from keras.callbacks import ModelCheckpoint
from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error


import warnings
warnings.filterwarnings('ignore')
warnings.filterwarnings('ignore', category=DeprecationWarning)
from xgboost import XGBRegressor
```

```
X1 = Test1[['Amenity','Bump','Crossing','Give_Way','Junction','No_Exit','Railway','Roun
Y1 = Test1[['duration_minute']]

X_new1 = X1.copy()                                    # Create copy of DataFrame
X_new1 = X_new1.astype(int)

x_train, x_test, y_train, y_test = train_test_split(X_new1, Y1)

NN_model = Sequential()

# The Input Layer :
NN_model.add(Dense(128, kernel_initializer='normal',input_dim = x_train.shape[1], activ
```

```
13
14 # The Hidden Layers :
15 NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
16 NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
17 NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
18
19 # The Output Layer :
20 NN_model.add(Dense(1, kernel_initializer='normal',activation='linear'))
21
22 # Compile the network :
23 NN_model.compile(loss='mean_absolute_error', optimizer='adam', metrics=['mean_absolute_
24 NN_model.summary()
25
26
27 NN_model.fit(x_train, y_train, epochs=5, batch_size=32, validation_split = 0.2)
28
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_15 (Dense)            (None, 128)               1792

 dense_16 (Dense)            (None, 256)               33024

 dense_17 (Dense)            (None, 256)               65792

 dense_18 (Dense)            (None, 256)               65792

 dense_19 (Dense)            (None, 1)                 257

=================================================================
Total params: 166,657
Trainable params: 166,657
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
9715/9715 [==============================] - 40s 4ms/step - loss: 10.4427 - mean_absc
Epoch 2/5
9715/9715 [==============================] - 35s 4ms/step - loss: 10.3281 - mean_absc
Epoch 3/5
9715/9715 [==============================] - 32s 3ms/step - loss: 10.3173 - mean_absc
Epoch 4/5
9715/9715 [==============================] - 31s 3ms/step - loss: 10.3093 - mean_absc
Epoch 5/5
9715/9715 [==============================] - 32s 3ms/step - loss: 10.3035 - mean_absc
<keras.callbacks.History at 0x7f8578e31890>
```

```
1 y_predicted = NN_model.predict(x_test)
2
3 mae = metrics.mean_absolute_error(y_test, y_predicted)
4 mse = metrics.mean_squared_error(y_test, y_predicted)
5 rmse = np.sqrt(mse) # or mse**(0.5)
6
7
8
```

```
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)

```

```
MAE: 10.292275375236624
MSE: 199.38845875348173
RMSE: 14.120497822438193
```

```
X2 = Test2[['Amenity','Bump','Crossing','Give_Way','Junction','No_Exit','Railway','Roun
Y2 = Test2[['duration_minute']]

X_new2 = X2.copy()                                    # Create copy of DataFrame
X_new2 = X_new2.astype(int)

x_train, x_test, y_train, y_test = train_test_split(X_new2, Y2)

NN_model = Sequential()

# The Input Layer :
NN_model.add(Dense(128, kernel_initializer='normal',input_dim = x_train.shape[1], activ

# The Hidden Layers :
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))

# The Output Layer :
NN_model.add(Dense(1, kernel_initializer='normal',activation='linear'))

# Compile the network :
NN_model.compile(loss='mean_absolute_error', optimizer='adam', metrics=['mean_absolute_
NN_model.summary()


NN_model.fit(x_train, y_train, epochs=5, batch_size=32, validation_split = 0.2)

y_predicted = NN_model.predict(x_test)

mae = metrics.mean_absolute_error(y_test, y_predicted)
mse = metrics.mean_squared_error(y_test, y_predicted)
rmse = np.sqrt(mse) # or mse**(0.5)


print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)

```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 dense_20 (Dense)            (None, 128)               1792

 dense_21 (Dense)            (None, 256)               33024

 dense_22 (Dense)            (None, 256)               65792

 dense_23 (Dense)            (None, 256)               65792

 dense_24 (Dense)            (None, 1)                 257

=================================================================
Total params: 166,657
Trainable params: 166,657
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
14045/14045 [==============================] - 57s 4ms/step - loss: 13.5375 - mean_ab
Epoch 2/5
14045/14045 [==============================] - 48s 3ms/step - loss: 13.2268 - mean_ab
Epoch 3/5
14045/14045 [==============================] - 50s 4ms/step - loss: 13.2121 - mean_ab
Epoch 4/5
14045/14045 [==============================] - 45s 3ms/step - loss: 13.1921 - mean_ab
Epoch 5/5
14045/14045 [==============================] - 49s 3ms/step - loss: 13.1805 - mean_ab
MAE: 13.206372268106184
MSE: 270.3104614606409
RMSE: 16.441121052429512
```

```
X3 = Test3[['Amenity','Bump','Crossing','Give_Way','Junction','No_Exit','Railway','Roun
Y3 = Test3[['duration_minute']]

X_new3 = X3.copy()                                  # Create copy of DataFrame
X_new3 = X_new3.astype(int)

x_train, x_test, y_train, y_test = train_test_split(X_new3, Y3)

NN_model = Sequential()

# The Input Layer :
NN_model.add(Dense(128, kernel_initializer='normal',input_dim = x_train.shape[1], activ

# The Hidden Layers :
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))

# The Output Layer :
NN_model.add(Dense(1, kernel_initializer='normal',activation='linear'))

# Compile the network :
NN_model.compile(loss='mean_absolute_error', optimizer='adam', metrics=['mean_absolute_
NN_model.summary()


NN_model.fit(x_train, y_train, epochs=5, batch_size=32, validation_split = 0.2)
```

```
28
29 y_predicted = NN_model.predict(x_test)
30
31 mae = metrics.mean_absolute_error(y_test, y_predicted)
32 mse = metrics.mean_squared_error(y_test, y_predicted)
33 rmse = np.sqrt(mse) # or mse**(0.5)
34
35
36
37 print("MAE:",mae)
38 print("MSE:", mse)
39 print("RMSE:", rmse)
40
41
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_25 (Dense)            (None, 128)               1792

 dense_26 (Dense)            (None, 256)               33024

 dense_27 (Dense)            (None, 256)               65792

 dense_28 (Dense)            (None, 256)               65792

 dense_29 (Dense)            (None, 1)                 257

=================================================================
Total params: 166,657
Trainable params: 166,657
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
9824/9824 [==============================] - 33s 3ms/step - loss: 14.7167 - mean_absc
Epoch 2/5
9824/9824 [==============================] - 31s 3ms/step - loss: 14.0449 - mean_absc
Epoch 3/5
9824/9824 [==============================] - 32s 3ms/step - loss: 13.9811 - mean_absc
Epoch 4/5
9824/9824 [==============================] - 32s 3ms/step - loss: 13.9570 - mean_absc
Epoch 5/5
9824/9824 [==============================] - 32s 3ms/step - loss: 13.9399 - mean_absc
MAE: 13.767303168167981
MSE: 315.01508368657835
RMSE: 17.74866427894162
```

```
1 X4 = Test4[['Amenity','Bump','Crossing','Give_Way','Junction','No_Exit','Railway','Roun
2 Y4 = Test4[['duration_minute']]
3
4 X_new4 = X4.copy()                                    # Create copy of DataFrame
5 X_new4 = X_new4.astype(int)
6
7 x_train, x_test, y_train, y_test = train_test_split(X_new4, Y4)
8
```

```
NN_model = Sequential()

# The Input Layer :
NN_model.add(Dense(128, kernel_initializer='normal',input_dim = x_train.shape[1], activ

# The Hidden Layers :
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))

# The Output Layer :
NN_model.add(Dense(1, kernel_initializer='normal',activation='linear'))

# Compile the network :
NN_model.compile(loss='mean_absolute_error', optimizer='adam', metrics=['mean_absolute_
NN_model.summary()


NN_model.fit(x_train, y_train, epochs=5, batch_size=32, validation_split = 0.2)

y_predicted = NN_model.predict(x_test)

mae = metrics.mean_absolute_error(y_test, y_predicted)
mse = metrics.mean_squared_error(y_test, y_predicted)
rmse = np.sqrt(mse) # or mse**(0.5)



print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)


```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_30 (Dense)            (None, 128)               1792

 dense_31 (Dense)            (None, 256)               33024

 dense_32 (Dense)            (None, 256)               65792

 dense_33 (Dense)            (None, 256)               65792

 dense_34 (Dense)            (None, 1)                 257

=================================================================
Total params: 166,657
Trainable params: 166,657
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
6004/6004 [==============================] - 20s 3ms/step - loss: 58.4166 - mean_absc
Epoch 2/5
6004/6004 [==============================] - 20s 3ms/step - loss: 56.8716 - mean_absc
```

```
Epoch 3/5
6004/6004 [==============================] - 19s 3ms/step - loss: 56.6230 - mean_absc
Epoch 4/5
6004/6004 [==============================] - 19s 3ms/step - loss: 56.4552 - mean_absc
Epoch 5/5
6004/6004 [==============================] - 19s 3ms/step - loss: 56.3197 - mean_absc
MAE: 56.02572536398219
MSE: 5270.41291167738
RMSE: 72.59760954520046
```