DATA CLEANING

In [2]:
```
import numpy as np
import pandas as pd
```

In [3]:
```
# reading the file

df = pd.read_csv("US_Accidents_March23.csv")
```

In [4]:
```
# raw data ( before processing and cleaning )

df.head()
```

Out[4]:

	ID	Source	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	End_Lat	End_Lng	Distance(mi
0	A-1	Source2	3	2016-02-08 05:46:00	2016-02-08 11:00:00	39.865147	-84.058723	NaN	NaN	0.0
1	A-2	Source2	2	2016-02-08 06:07:59	2016-02-08 06:37:59	39.928059	-82.831184	NaN	NaN	0.0
2	A-3	Source2	2	2016-02-08 06:49:27	2016-02-08 07:19:27	39.063148	-84.032608	NaN	NaN	0.0
3	A-4	Source2	3	2016-02-08 07:23:34	2016-02-08 07:53:34	39.747753	-84.205582	NaN	NaN	0.0
4	A-5	Source2	2	2016-02-08 07:39:07	2016-02-08 08:09:07	39.627781	-84.188354	NaN	NaN	0.0

5 rows × 46 columns

In [5]:
```
df.shape  # 77 lakh rows and 46 columns (2016 - 2023) such a huge number of accidents.
```

Out[5]:
```
(7728394, 46)
```

In [6]:
```
# data types

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7728394 entries, 0 to 7728393
Data columns (total 46 columns):
 #   Column                 Dtype  
---  ------                 -----  
 0   ID                     object 
 1   Source                 object 
 2   Severity               int64  
 3   Start_Time             object 
 4   End_Time               object 
 5   Start_Lat              float64
 6   Start_Lng              float64
 7   End_Lat                float64
 8   End_Lng                float64
 9   Distance(mi)           float64
 10  Description            object 
 11  Street                 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  Wind_Chill(F)          float64
 22  Humidity(%)            float64
 23  Pressure(in)           float64
 24  Visibility(mi)         float64
 25  Wind_Direction         object 
 26  Wind_Speed(mph)        float64
 27  Precipitation(in)      float64
 28  Weather_Condition      object 
 29  Amenity                bool   
 30  Bump                   bool   
 31  Crossing               bool   
 32  Give_Way               bool   
 33  Junction               bool   
 34  No_Exit                bool   
 35  Railway                bool   
 36  Roundabout             bool   
 37  Station                bool   
 38  Stop                   bool   
 39  Traffic_Calming        bool   
 40  Traffic_Signal         bool   
 41  Turning_Loop           bool   
 42  Sunrise_Sunset         object 
 43  Civil_Twilight         object 
 44  Nautical_Twilight      object 
 45  Astronomical_Twilight  object 
dtypes: bool(13), float64(12), int64(1), object(20)
memory usage: 2.0+ GB
```

In [7]:
```
# missing values

miss = df.isnull().sum()
# df[miss.index]
```

In [8]:
```
# looking for percentage of missing values and sorting them in descending order to gre

missing_percent_values = round(df.isnull().sum().sort_values(ascending = False) / len(
```

In [9]:
```
missing_percent_values
```

Out[9]:
```
End_Lat                 44.03
End_Lng                 44.03
Precipitation(in)       28.51
Wind_Chill(F)           25.87
Wind_Speed(mph)          7.39
Visibility(mi)           2.29
Wind_Direction           2.27
Humidity(%)              2.25
Weather_Condition        2.24
Temperature(F)           2.12
Pressure(in)             1.82
Weather_Timestamp        1.56
Nautical_Twilight        0.30
Civil_Twilight           0.30
Sunrise_Sunset           0.30
Astronomical_Twilight    0.30
Airport_Code             0.29
Street                   0.14
Timezone                 0.10
Zipcode                  0.02
City                     0.00
Description              0.00
Traffic_Signal           0.00
Roundabout               0.00
Station                  0.00
Stop                     0.00
Traffic_Calming          0.00
Country                  0.00
Turning_Loop             0.00
No_Exit                  0.00
End_Time                 0.00
Start_Time               0.00
Severity                 0.00
Railway                  0.00
Crossing                 0.00
Junction                 0.00
Give_Way                 0.00
Bump                     0.00
Amenity                  0.00
Start_Lat                0.00
Start_Lng                0.00
Distance(mi)             0.00
Source                   0.00
County                   0.00
State                    0.00
ID                       0.00
dtype: float64
```

Getting columns that only have missing values to deal with them

In [10]:
```
cols_missing = missing_percent_values[missing_percent_values > 0]
```

In [11]:
```
cols_missing
```

Out[11]:
```
End_Lat                  44.03
End_Lng                  44.03
Precipitation(in)        28.51
Wind_Chill(F)            25.87
Wind_Speed(mph)           7.39
Visibility(mi)            2.29
Wind_Direction            2.27
Humidity(%)               2.25
Weather_Condition         2.24
Temperature(F)            2.12
Pressure(in)              1.82
Weather_Timestamp         1.56
Nautical_Twilight         0.30
Civil_Twilight            0.30
Sunrise_Sunset            0.30
Astronomical_Twilight     0.30
Airport_Code              0.29
Street                    0.14
Timezone                  0.10
Zipcode                   0.02
dtype: float64
```

In [12]:
```
# creating a data frame only for the missing values.

missing = df.isnull().sum().sort_values(ascending = False)
missing_percentage = round(missing / len(df) * 100 , 2).sort_values(ascending = False)
missing_cols_df = pd.concat([missing, missing_percentage, df[missing.index].dtypes], a
                            keys = ['Count', 'Percent', 'Data Types'], sort = False)
```

In [13]:
```
missing.dtype
```

Out[13]:
```
dtype('int64')
```

In [14]:
```
missing_cols_df = missing_cols_df[missing_cols_df['Count'] >= 1]
missing_cols_df
```

Out[14]:

	Count	Percent	Data Types
End_Lat	3402762	44.03	float64
End_Lng	3402762	44.03	float64
Precipitation(in)	2203586	28.51	float64
Wind_Chill(F)	1999019	25.87	float64
Wind_Speed(mph)	571233	7.39	float64
Visibility(mi)	177098	2.29	float64
Wind_Direction	175206	2.27	object
Humidity(%)	174144	2.25	float64
Weather_Condition	173459	2.24	object
Temperature(F)	163853	2.12	float64
Pressure(in)	140679	1.82	float64
Weather_Timestamp	120228	1.56	object
Nautical_Twilight	23246	0.30	object
Civil_Twilight	23246	0.30	object
Sunrise_Sunset	23246	0.30	object
Astronomical_Twilight	23246	0.30	object
Airport_Code	22635	0.29	object
Street	10869	0.14	object
Timezone	7808	0.10	object
Zipcode	1915	0.02	object
City	253	0.00	object
Description	5	0.00	object

In [15]:
```
# filtering out the float values

missing_cols_df_float = missing_cols_df[missing_cols_df['Data Types'] == 'float64']
missing_cols_df_float
```

Out[15]:

	Count	Percent	Data Types
End_Lat	3402762	44.03	float64
End_Lng	3402762	44.03	float64
Precipitation(in)	2203586	28.51	float64
Wind_Chill(F)	1999019	25.87	float64
Wind_Speed(mph)	571233	7.39	float64
Visibility(mi)	177098	2.29	float64
Humidity(%)	174144	2.25	float64
Temperature(F)	163853	2.12	float64
Pressure(in)	140679	1.82	float64

Let us now deal with the missing values of numerical columns.. (int, float etc)

In [16]:
```
# We don't require End latitude and End longitude as of now. because it doesn't give m
# It is just how far the car might be dragged or tossed.
# And if it is missing then it means that car is at the same point after accident.
# We will deal with this 2 columns later.
```

In [17]:
```
# Let us consider the variables Temperature and Pressure.
# There has to be some temperature or pressure associated to a day
# So i can't fill it with zero value neither I can drop them.
# There are very few values that is appx 2% values missing
# So option is to fill with mean, median, mode.
```

In [18]:
```
# Let us see the characteristics for all the numerical columns
df.describe().T
```

Out[18]:

	count	mean	std	min	25%	50%	75%
Severity	7728394.0	2.212384	0.487531	1.000000	2.000000	2.000000	2.000000
Start_Lat	7728394.0	36.201195	5.076079	24.554800	33.399631	35.823974	40.084959
Start_Lng	7728394.0	-94.702545	17.391756	-124.623833	-117.219396	-87.766616	-80.353676
End_Lat	4325632.0	36.261829	5.272905	24.566013	33.462070	36.183495	40.178920
End_Lng	4325632.0	-95.725570	18.107928	-124.545748	-117.754345	-88.027890	-80.247086
Distance(mi)	7728394.0	0.561842	1.776811	0.000000	0.000000	0.030000	0.464000
Temperature(F)	7564541.0	61.663286	19.013653	-89.000000	49.000000	64.000000	76.000000
Wind_Chill(F)	5729375.0	58.251048	22.389832	-89.000000	43.000000	62.000000	75.000000
Humidity(%)	7554250.0	64.831041	22.820968	1.000000	48.000000	67.000000	84.000000
Pressure(in)	7587715.0	29.538986	1.006190	0.000000	29.370000	29.860000	30.030000
Visibility(mi)	7551296.0	9.090376	2.688316	0.000000	10.000000	10.000000	10.000000
Wind_Speed(mph)	7157161.0	7.685490	5.424983	0.000000	4.600000	7.000000	10.400000
Precipitation(in)	5524808.0	0.008407	0.110225	0.000000	0.000000	0.000000	0.000000

In [19]:
```
# Looking into the above table
# The mean and median values for pressure are exactly same.
  #so it follows normal dist and we can replace missing values with mean

# In the case of Temperature it differs a bit but not lot skewed.
  #so we can also replace missing values with mean.
```

In [20]:
```
# Replacing missing values
lst = ['Temperature(F)','Pressure(in)']

for i in lst:
    df[i] = df[i].fillna(df[i].mean())
```

In [21]:
```
# Now let us deal with other three columns
# Let us fill WIndspeed, visibility, humidity with mean as well. because they as mean

lst = ['Humidity(%)','Wind_Speed(mph)','Visibility(mi)']

for i in lst:
    df[i] = df[i].fillna(df[i].mean())
```

In [22]:
```
# The remaining two columns 'Precipitation' and 'Wind chill' are not recorded.
# There is a posibility that there will be a zero values for both of these.
# We can either completely remove the column or we can replace them with 0.
# Since I have decided that windchill doesn't make any sense for accidents I decided t
# I would also like to drop 'End_lat' and 'End_lon' columns as they don't have signifi
```

In [23]:
```
# Dropping un-necessary columns.

df = df.drop(['Precipitation(in)','Wind_Chill(F)','End_Lat','End_Lng'], axis = 1)
```

In [24]:
```
df.isnull().sum()
```

Out[24]:
```
ID                          0
Source                      0
Severity                    0
Start_Time                  0
End_Time                    0
Start_Lat                   0
Start_Lng                   0
Distance(mi)                0
Description                 5
Street                  10869
City                      253
County                      0
State                       0
Zipcode                  1915
Country                     0
Timezone                 7808
Airport_Code            22635
Weather_Timestamp      120228
Temperature(F)              0
Humidity(%)                 0
Pressure(in)                0
Visibility(mi)              0
Wind_Direction         175206
Wind_Speed(mph)             0
Weather_Condition      173459
Amenity                     0
Bump                        0
Crossing                    0
Give_Way                    0
Junction                    0
No_Exit                     0
Railway                     0
Roundabout                  0
Station                     0
Stop                        0
Traffic_Calming             0
Traffic_Signal              0
Turning_Loop                0
Sunrise_Sunset          23246
Civil_Twilight          23246
Nautical_Twilight       23246
Astronomical_Twilight   23246
dtype: int64
```

Let us deal with non-numeric columns now

In [25]:
```
missing_cols_df_non_numeric = missing_cols_df[missing_cols_df['Data Types'] == 'object
missing_cols_df_non_numeric
```

Out[25]:

	Count	Percent	Data Types
Wind_Direction	175206	2.27	object
Weather_Condition	173459	2.24	object
Weather_Timestamp	120228	1.56	object
Nautical_Twilight	23246	0.30	object
Civil_Twilight	23246	0.30	object
Sunrise_Sunset	23246	0.30	object
Astronomical_Twilight	23246	0.30	object
Airport_Code	22635	0.29	object
Street	10869	0.14	object
Timezone	7808	0.10	object
Zipcode	1915	0.02	object
City	253	0.00	object
Description	5	0.00	object

In [26]:
```
df[missing_cols_df_non_numeric.index]
```

Out[26]:

	Wind_Direction	Weather_Condition	Weather_Timestamp	Nautical_Twilight	Civil_Twilight	Su
0	Calm	Light Rain	2016-02-08 05:58:00	Night	Night	
1	Calm	Light Rain	2016-02-08 05:51:00	Night	Night	
2	SW	Overcast	2016-02-08 06:56:00	Day	Night	
3	SW	Mostly Cloudy	2016-02-08 07:38:00	Day	Day	
4	SW	Mostly Cloudy	2016-02-08 07:53:00	Day	Day	
...	...	...	...	...	...	
7728389	W	Fair	2019-08-23 17:53:00	Day	Day	
7728390	SW	Fair	2019-08-23 18:53:00	Day	Day	
7728391	SSW	Partly Cloudy	2019-08-23 18:53:00	Day	Day	
7728392	SW	Fair	2019-08-23 18:51:00	Day	Day	
7728393	SW	Fair	2019-08-23 20:50:00	Day	Day	

7728394 rows × 13 columns

In [32]:
```
# Let us consider State, Zipcode and City columns. There is an obvious relation betwee
# Let us dig deeper into these columns.

df[df['Zipcode'] == '45424']['State'].value_counts()
# It is evident that all 45425 have same state.
```

Out[32]:
```
State
OH    367
Name: count, dtype: int64
```

In [37]:
```
# We can group by the state with city and see the results.

df.groupby('State')['City'].count()
```

Out[37]:
```
State
AL     101044
AR      22780
AZ     170609
CA    1741422
CO      90877
CT      71005
DC      18493
DE      14097
FL     880159
GA     169234
IA      26306
ID      11376
IL     168956
IN      67219
KS      20991
KY      32254
LA     149701
MA      61996
MD     140408
ME       2698
MI     162189
MN     192079
MO      77323
MS      15181
MT      28496
NC     338199
ND       3487
NE      28870
NH      10213
NJ     140719
NM      10325
NV      21665
NY     347932
OH     118115
OK      83647
OR     179655
PA     296620
RI      16971
SC     382557
SD        289
TN     167386
TX     582837
UT      97079
VA     303301
VT        926
WA     108221
WI      34686
WV      13791
WY       3757
Name: City, dtype: int64
```

In [39]:
```
# Now we can fill the missing values of the city with the most frequent city of that p

df['City'] = df.groupby('State')['City'].transform(lambda x: x.fillna(x.mode().iloc[0]
```

In [41]:
```
df.groupby(['City','Zipcode'])['ID'].count()
```

Out[41]:
```
City        Zipcode
Aaronsburg  16820          9
            16820-8900     1
            16820-9113     1
            16820-9115     7
            16820-9202     1
                          ..
Zuni        23898-2835     1
            23898-2857     3
            23898-3311     1
Zwingle     52079         10
            52079-9603     4
Name: ID, Length: 827394, dtype: int64
```

In [42]:
```
# Now lets use the same method to fill the missing values of Zipcode and Airport Code

df['Zipcode'] = df.groupby('State')['Zipcode'].transform(lambda x: x.fillna(x.mode().i
df['Airport_Code'] = df.groupby('State')['Airport_Code'].transform(lambda x: x.fillna(
```

In [44]:
```
# Description has very few missing values.
# If we are doing some analysis for text data we can use this column
# Currently I am not using any textual column
# So I am dropping this column completely
```

In [45]:
```
df = df.drop(['Description'], axis = 1)
```

In [46]:
```
df.shape
```

Out[46]:
```
(7728394, 41)
```

In [49]:
```
df.isnull().sum().sort_values(ascending=False)
```

Out[49]:
```
Wind_Direction           175206
Weather_Condition        173459
Weather_Timestamp        120228
Sunrise_Sunset            23246
Astronomical_Twilight     23246
Civil_Twilight            23246
Nautical_Twilight         23246
Street                    10869
Timezone                   7808
Turning_Loop                  0
Junction                      0
Amenity                       0
Bump                          0
Crossing                      0
Give_Way                      0
No_Exit                       0
Traffic_Signal                0
Railway                       0
Station                       0
Stop                          0
Traffic_Calming               0
Roundabout                    0
ID                            0
Wind_Speed(mph)               0
Source                        0
Severity                      0
Start_Time                    0
End_Time                      0
Start_Lat                     0
Start_Lng                     0
Distance(mi)                  0
City                          0
County                        0
State                         0
Zipcode                       0
Country                       0
Airport_Code                  0
Temperature(F)                0
Humidity(%)                   0
Pressure(in)                  0
Visibility(mi)                0
dtype: int64
```

In [51]:
```
# Honestly speaking for categorical values the best possible way to impute the values
# So for the following columns I would use the same method.
# Anyways I would not be using some of these columns.

columns_to_impute = ['Astronomical_Twilight','Civil_Twilight','Nautical_Twilight','Wir

# function definition

def impute_mode(x):
    df[x].fillna(df[x].mode().iloc[0], inplace = True)

for col in columns_to_impute:
    impute_mode(col)
```

In [58]:
```
df_time = df[['Weather_Timestamp','Start_Time']]
df_time
```

Out[58]:

	Weather_Timestamp	Start_Time
0	2016-02-08 05:58:00	2016-02-08 05:46:00
1	2016-02-08 05:51:00	2016-02-08 06:07:59
2	2016-02-08 06:56:00	2016-02-08 06:49:27
3	2016-02-08 07:38:00	2016-02-08 07:23:34
4	2016-02-08 07:53:00	2016-02-08 07:39:07
...	...	...
7728389	2019-08-23 17:53:00	2019-08-23 18:03:25
7728390	2019-08-23 18:53:00	2019-08-23 19:11:30
7728391	2019-08-23 18:53:00	2019-08-23 19:00:21
7728392	2019-08-23 18:51:00	2019-08-23 19:00:21
7728393	2019-08-23 20:50:00	2019-08-23 18:52:06

7728394 rows × 2 columns

In [57]:
```
mask = df_time.isnull().any(axis = 1)
df_time[mask]
```

Out[57]:

	Weather_Timestamp	Start_Time
601	NaN	2016-03-11 07:28:40
1957	NaN	2016-07-03 03:54:45
1968	NaN	2016-07-03 08:41:05
1973	NaN	2016-07-03 11:59:28
1978	NaN	2016-07-03 13:12:18
...	...	...
7728155	NaN	2019-08-23 12:23:00
7728156	NaN	2019-08-23 12:23:00
7728162	NaN	2019-08-23 13:23:00
7728250	NaN	2019-08-23 17:19:55
7728305	NaN	2019-08-23 23:48:20

120228 rows × 2 columns

In [59]:
```
# We can observe from the above data that almost weather_timestamp and Start_time(acci
# We can just impute the start time value in the missing values place.

df['Weather_Timestamp'].fillna(df['Start_Time'], inplace = True)
```

In [60]:
```
# Same methos for time zone as well - mode after grouping by state.
```

```
df['Timezone'] = df.groupby('State')['Timezone'].transform(lambda x: x.fillna(x.mode()
```

In [64]:
```
# Nautical - we can do some substitution like
# Day if hour is between 6:00 am and 6:00 pm else night. We can do this for the other

df['Start_Time'] = pd.to_datetime(df['Start_Time'], format="ISO8601")
df['End_Time'] = pd.to_datetime(df['End_Time'], format="ISO8601")

def filler(df, columns):
    lst = df[df[columns].isna()].index
    for i in lst:
        if 6 <= df.loc[i, 'Start_Time'].hour < 18:
            df.loc[i, columns] = 'Day'
        else:
            df.loc[i, columns] = 'Night'

# Calling the function
filler(df, 'Nautical_Twilight')
```

In [66]:
```
df['Street'] = df.groupby(['City'])['Street'].transform(lambda x: x.fillna(x.mode().il
```

In [67]:
```
df.isnull().sum()
```

Out[67]:
```
ID                          0
Source                      0
Severity                    0
Start_Time                  0
End_Time                    0
Start_Lat                   0
Start_Lng                   0
Distance(mi)                0
Street                      0
City                        0
County                      0
State                       0
Zipcode                     0
Country                     0
Timezone                    0
Airport_Code                0
Weather_Timestamp           0
Temperature(F)              0
Humidity(%)                 0
Pressure(in)                0
Visibility(mi)              0
Wind_Direction              0
Wind_Speed(mph)             0
Weather_Condition           0
Amenity                     0
Bump                        0
Crossing                    0
Give_Way                    0
Junction                    0
No_Exit                     0
Railway                     0
Roundabout                  0
Station                     0
Stop                        0
Traffic_Calming             0
Traffic_Signal              0
Turning_Loop                0
Sunrise_Sunset          23246
Civil_Twilight              0
Nautical_Twilight           0
Astronomical_Twilight       0
dtype: int64
```

In [68]:
```
impute_mode('Sunrise_Sunset')
```

In [69]:
```
df.isnull().sum()
```

Out[69]:
```
ID                    0
Source                0
Severity              0
Start_Time            0
End_Time              0
Start_Lat             0
Start_Lng             0
Distance(mi)          0
Street                0
City                  0
County                0
State                 0
Zipcode               0
Country               0
Timezone              0
Airport_Code          0
Weather_Timestamp     0
Temperature(F)        0
Humidity(%)           0
Pressure(in)          0
Visibility(mi)        0
Wind_Direction        0
Wind_Speed(mph)       0
Weather_Condition     0
Amenity               0
Bump                  0
Crossing              0
Give_Way              0
Junction              0
No_Exit               0
Railway               0
Roundabout            0
Station               0
Stop                  0
Traffic_Calming       0
Traffic_Signal        0
Turning_Loop          0
Sunrise_Sunset        0
Civil_Twilight        0
Nautical_Twilight     0
Astronomical_Twilight 0
dtype: int64
```

HURRAY!!! Data is Cleaned

In [70]:
```
# Finally let us check if there are any duplicates
```

In [71]:
```
df[df.duplicated]
```

Out[71]:

ID	Source	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	Distance(mi)	Street	City	...	Rou

0 rows × 41 columns

In [ ]:
```

```