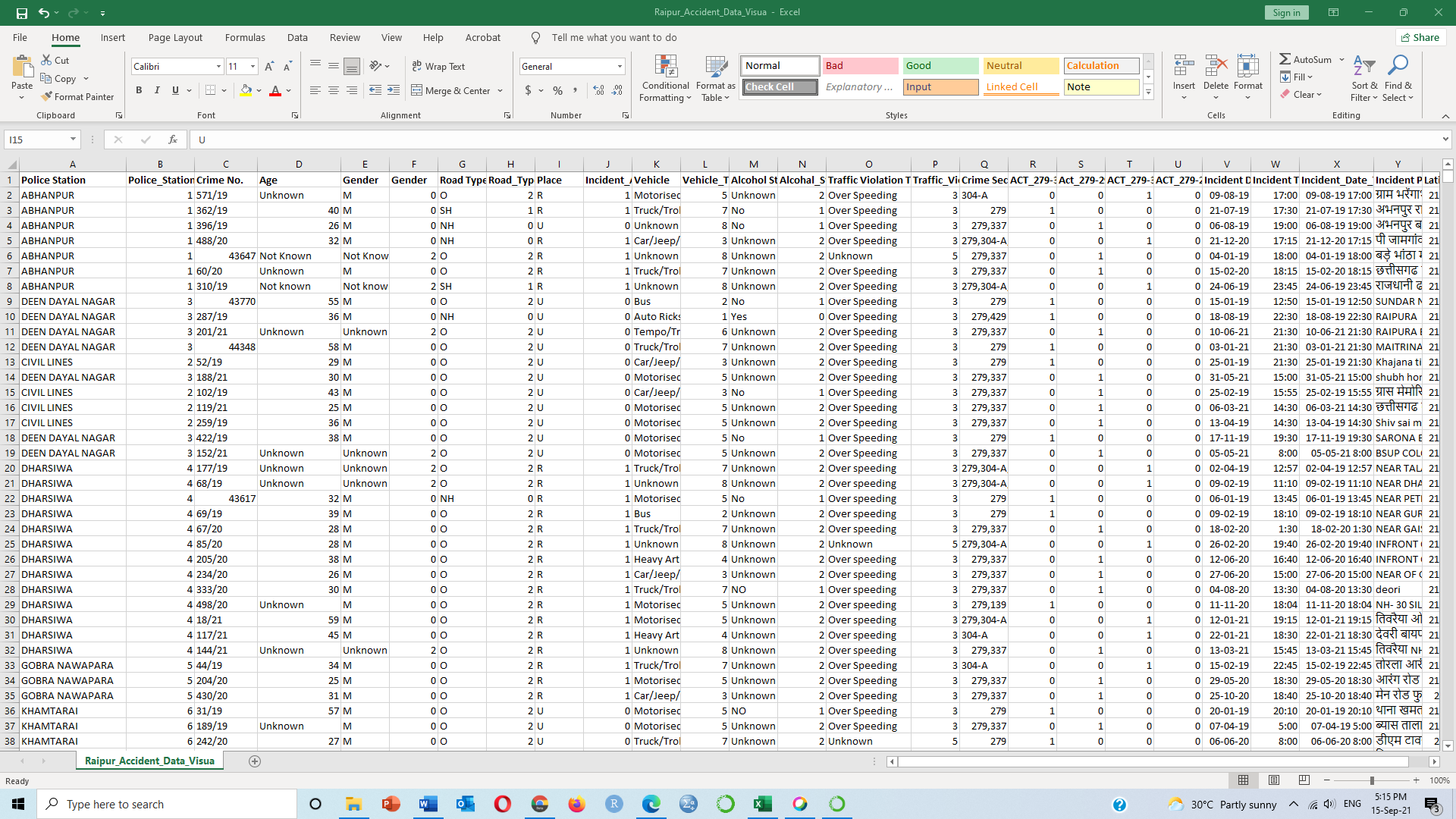
**Visualisation, clustering and classification of**

**Raipur Road Accident Data**

**For Year 2019 & 2020 by using Python**

**Structure of data**



**#import required library files**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

sns.set\_style('whitegrid')

%matplotlib inline

import folium

from sklearn.cluster import DBSCAN

from collections import Counter

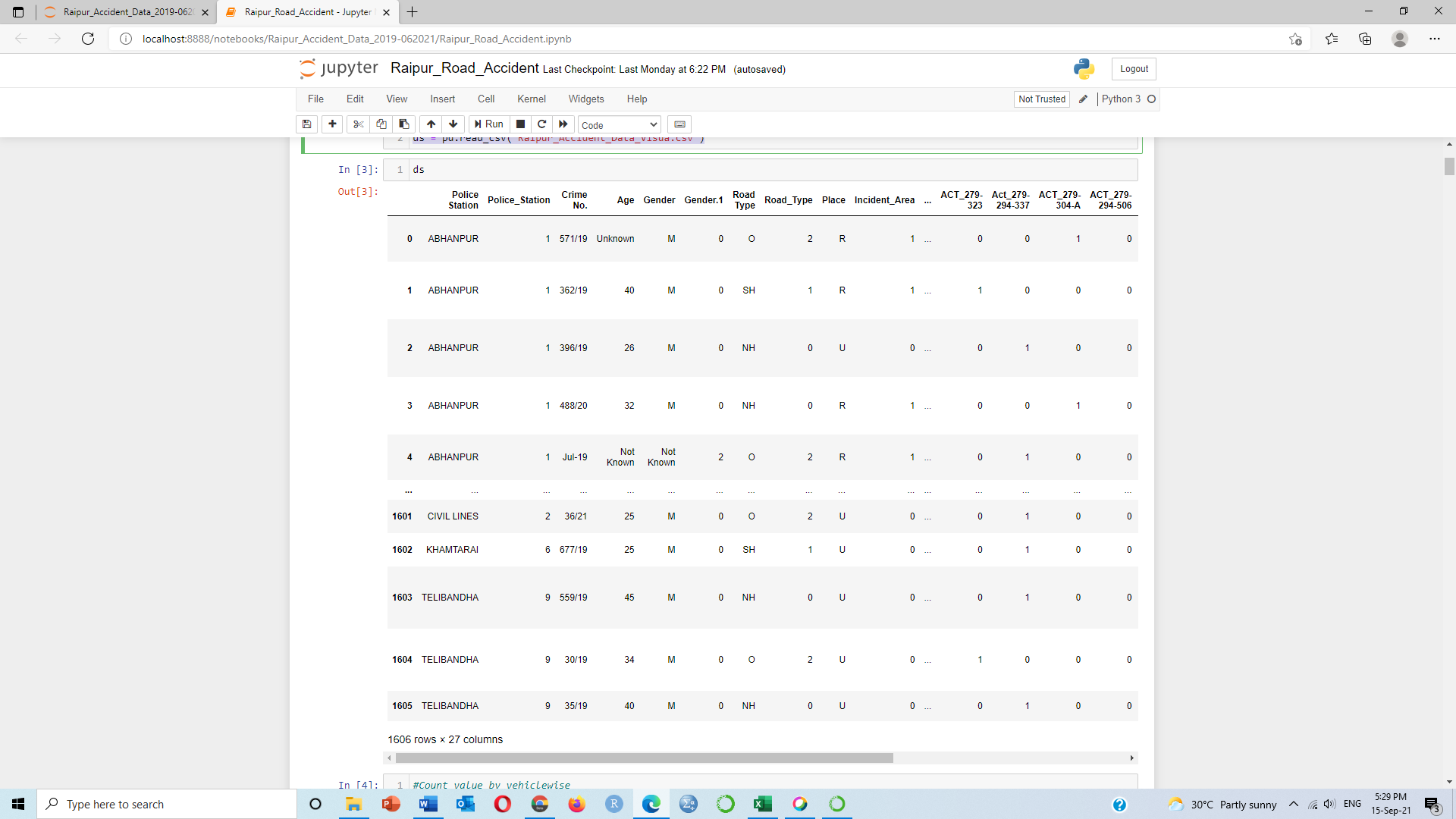
from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

from sklearn.preprocessing import StandardScaler

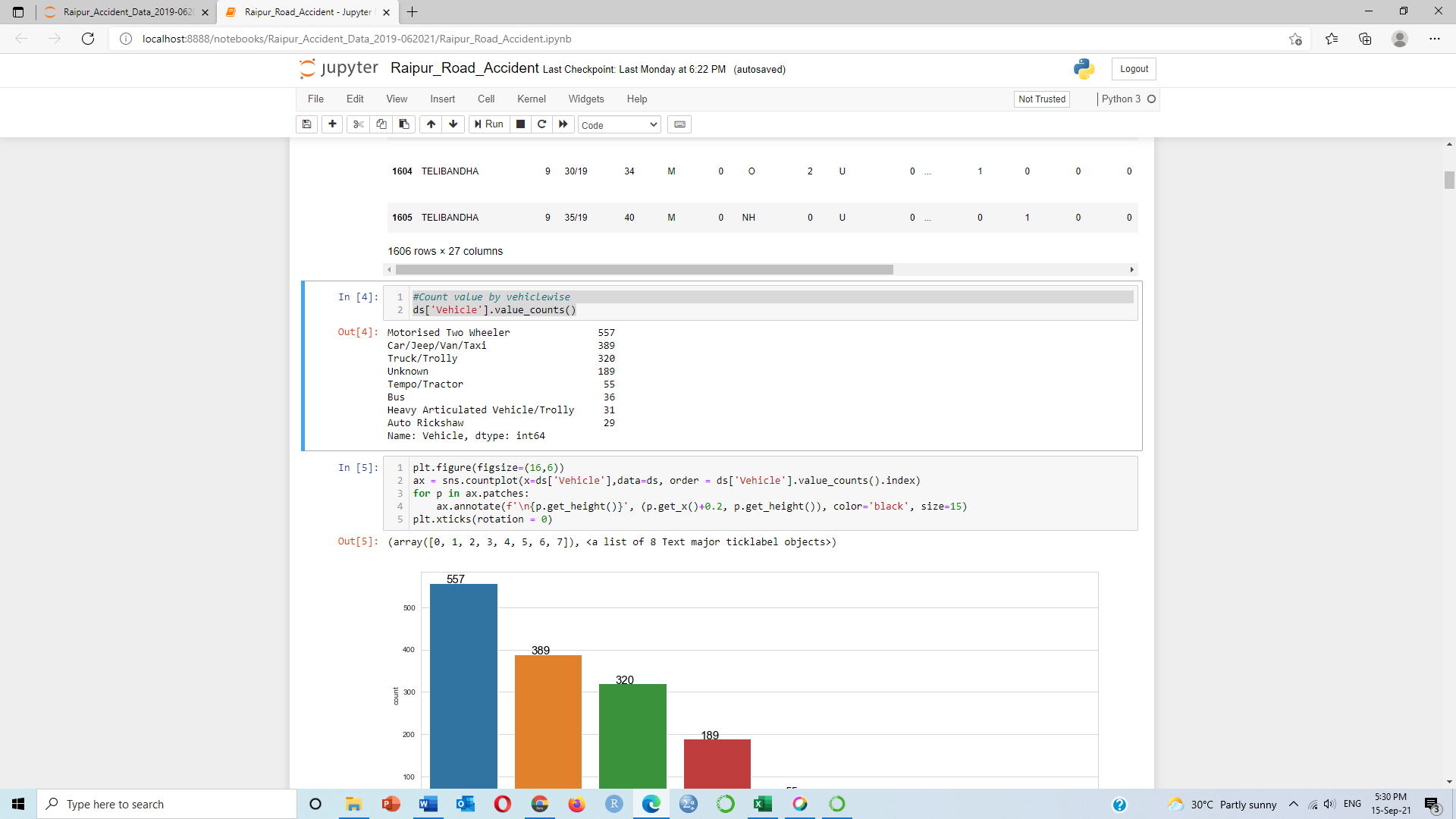
**# Read the data**

ds = pd.read\_csv("Raipur\_Accident\_Data\_Visua.csv")



**#Count value by vehiclewise**

ds['Vehicle'].value\_counts()



**#Plot figure**

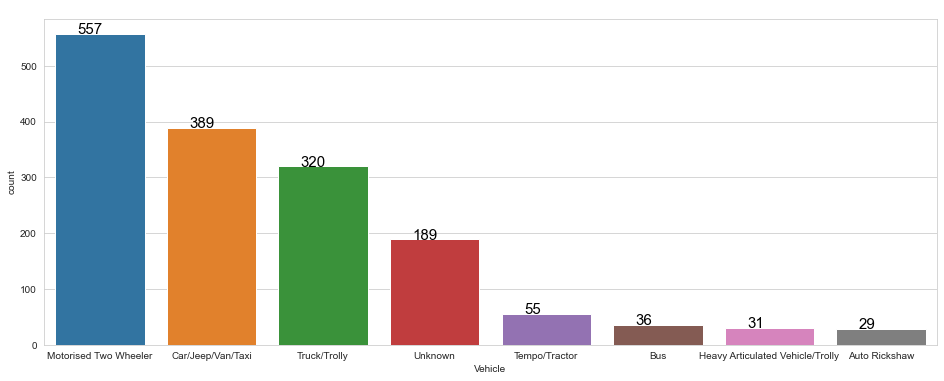
plt.figure(figsize=(16,6))

ax = sns.countplot(x=ds['Vehicle'],data=ds, order = ds['Vehicle'].value\_counts().index)

for p in ax.patches:

ax.annotate(f'\n{p.get\_height()}', (p.get\_x()+0.2, p.get\_height()), color='black', size=15)

plt.xticks(rotation = 0)



**#Count value by National highway, State highway or Other**

ds['Road Type'].value\_counts()

O 744

NH 575

SH 287

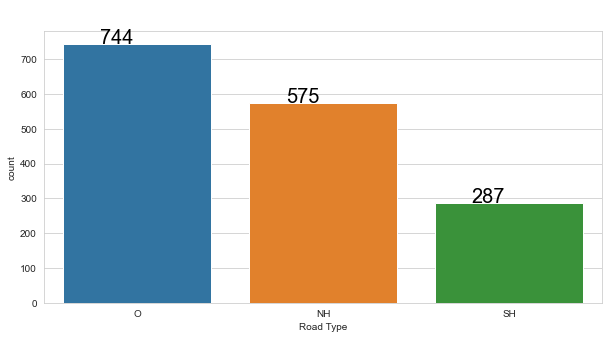
Name: Road Type, dtype: int64

plt.figure(figsize=(10,5))

ax = sns.countplot(x=ds['Road Type'],data=ds, order = ds['Road Type'].value\_counts().index)

for p in ax.patches:

ax.annotate(f'\n{p.get\_height()}', (p.get\_x()+0.2, p.get\_height()), color='black', size=20)



**#Count value by Urban or Rural area wise**

ds['Place'].value\_counts()

R 891

U 715

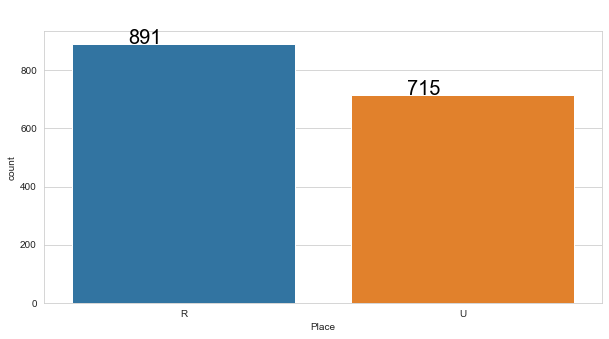
Name: Place, dtype: int64

plt.figure(figsize=(10,5))

ax = sns.countplot(x=ds['Place'],data=ds, order = ds['Place'].value\_counts().index)

for p in ax.patches:

ax.annotate(f'\n{p.get\_height()}', (p.get\_x()+0.2, p.get\_height()), color='black', size=20)



**#Count value by Police Station wise**

ds['Police Station'].value\_counts()

DHARSIWA 310

GOBRA NAWAPARA 231

KHARORA 193

TELIBANDHA 190

ABHANPUR 173

KHAMTARAI 153

CIVIL LINES 144

DEEN DAYAL NAGAR 122

KOTWALI 90

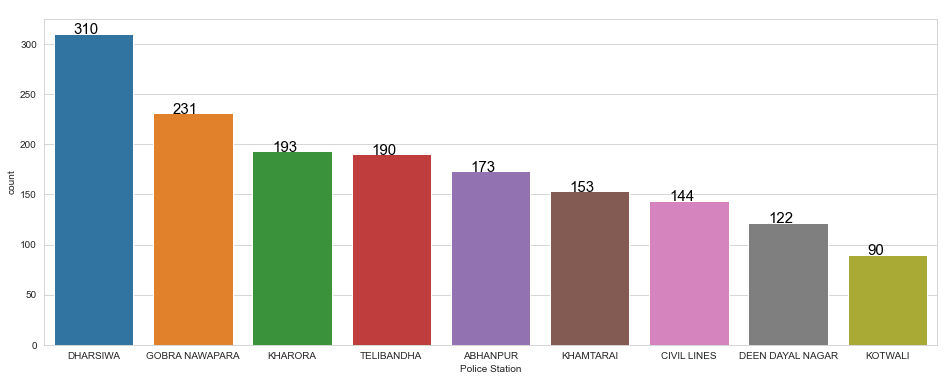
Name: Police Station, dtype: int64

plt.figure(figsize=(16,6))

ax = sns.countplot(x=ds['Police Station'],data=ds, order = ds['Police Station'].value\_counts().index)

for p in ax.patches:

ax.annotate(f'\n{p.get\_height()}', (p.get\_x()+0.2, p.get\_height()), color='black', size=15)



**#Count Hour, Month & Weeks of Incident\_Date\_Time**

type(ds['Incident\_Date\_Time'].iloc[1])

ds['Incident\_Date\_Time'] = pd.to\_datetime(ds['Incident\_Date\_Time'])

ds['Hour'] = ds['Incident\_Date\_Time'].apply(lambda time: time.hour)

ds['Month'] = ds['Incident\_Date\_Time'].apply(lambda time: time.month)

ds['Day\_of\_Week'] = ds['Incident\_Date\_Time'].apply(lambda time: time.dayofweek)

**#Count no of accidents in days of week and plot figure**

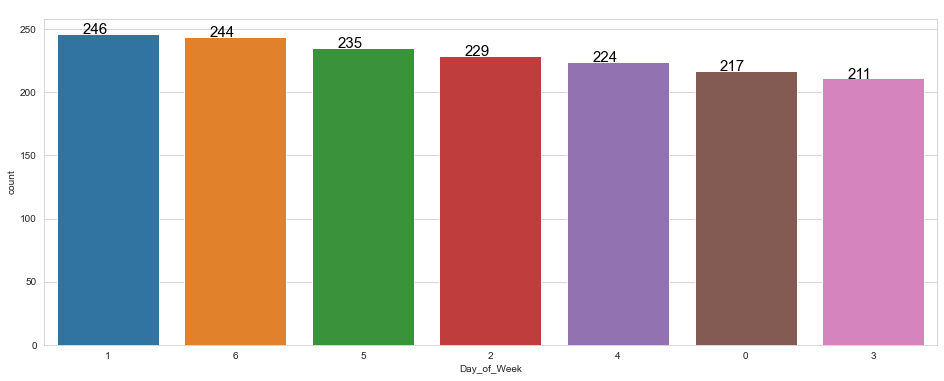
plt.figure(figsize=(16,6))

ax = sns.countplot(x=ds['Day\_of\_Week'],data=ds, order = ds['Day\_of\_Week'].value\_counts().index)

for p in ax.patches:

ax.annotate(f'\n{p.get\_height()}', (p.get\_x()+0.2, p.get\_height()), color='black', size=15)

plt.xticks(rotation = 0)



**#Count no of accidents in hour and plot figure**

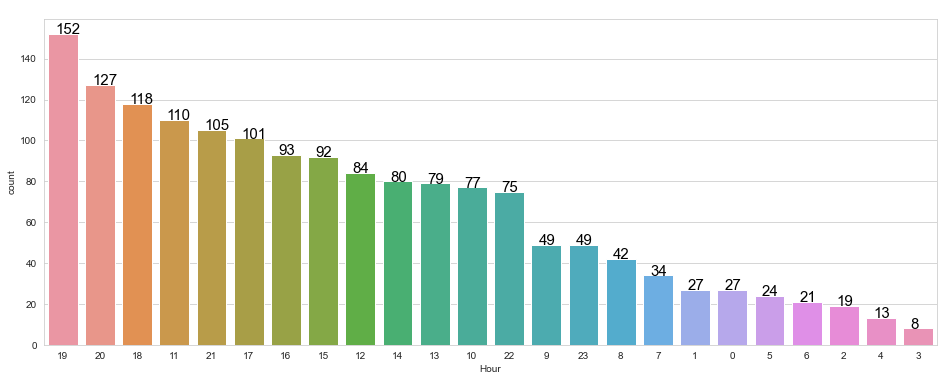
plt.figure(figsize=(16,6))

ax = sns.countplot(x=ds['Hour'],data=ds, order = ds['Hour'].value\_counts().index)

for p in ax.patches:

ax.annotate(f'\n{p.get\_height()}', (p.get\_x()+0.2, p.get\_height()), color='black', size=15)

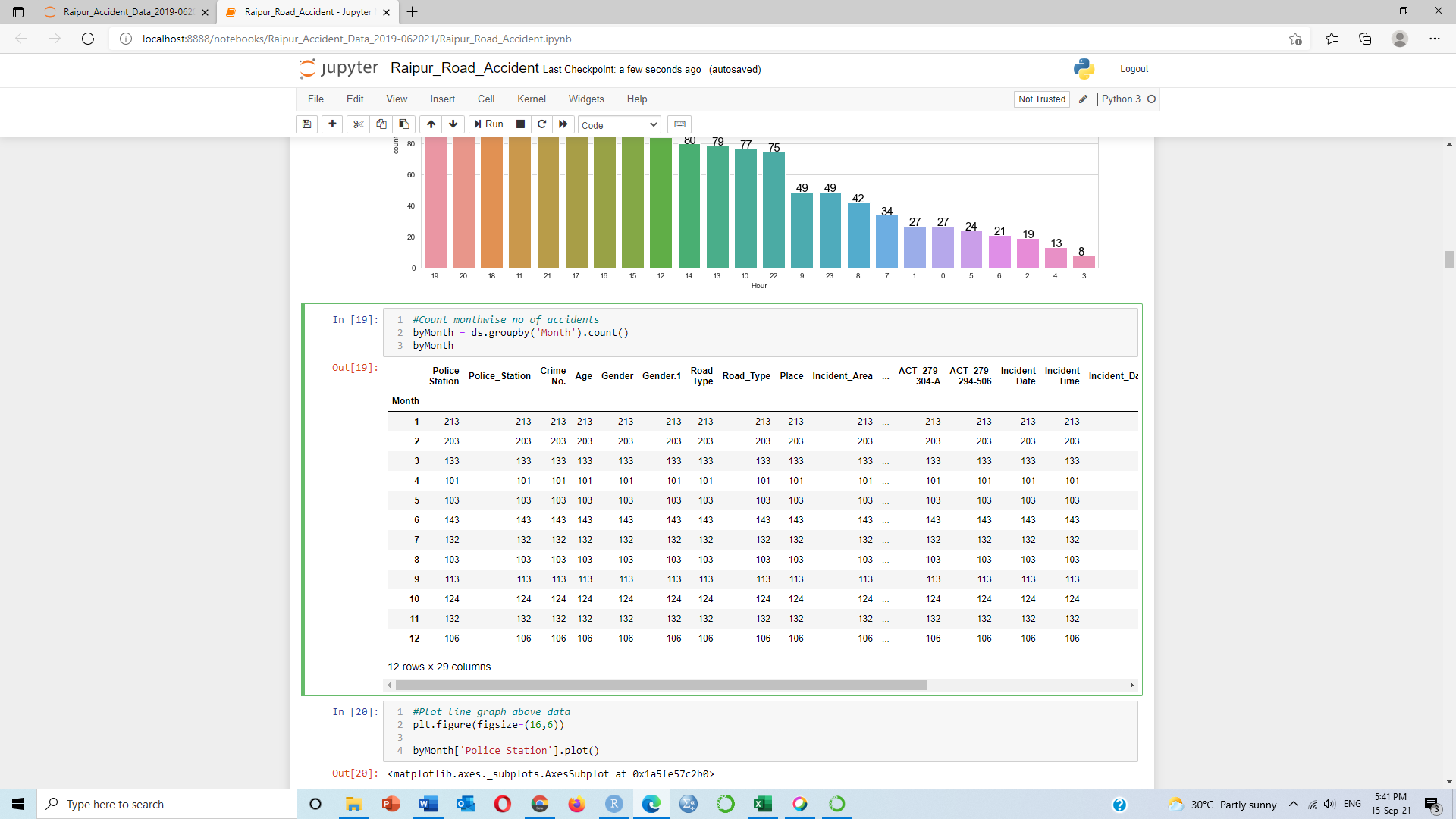
plt.xticks(rotation = 0)



**#Count month wise no of accidents**

byMonth = ds.groupby('Month').count()

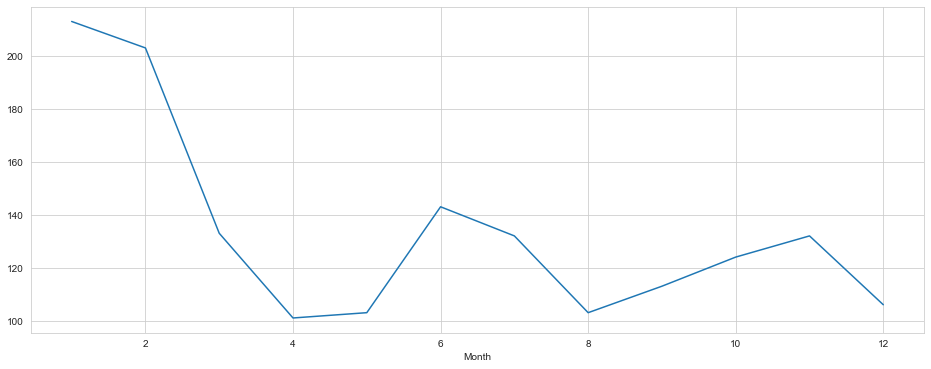
byMonth



**#Plot line graph above data**

plt.figure(figsize=(16,6))

byMonth['Police Station'].plot()



type(ds['Incident\_Date\_Time'].iloc[0])

ds['Incident\_Date\_Time'] = pd.to\_datetime(ds['Incident\_Date\_Time'])

ds['Hour'] = ds['Incident\_Date\_Time'].apply(lambda time: time.hour)

ds['Month'] = ds['Incident\_Date\_Time'].apply(lambda time: time.month)

ds['Day\_of\_Week'] = ds['Incident\_Date\_Time'].apply(lambda time: time.dayofweek)

dmap = {0:'Mon',1:'Tue',2:'Wed',3:'Thu',4:'Fri',5:'Sat',6:'Sun'}

ds['Day\_of\_Week'] = ds['Day\_of\_Week'].map(dmap)

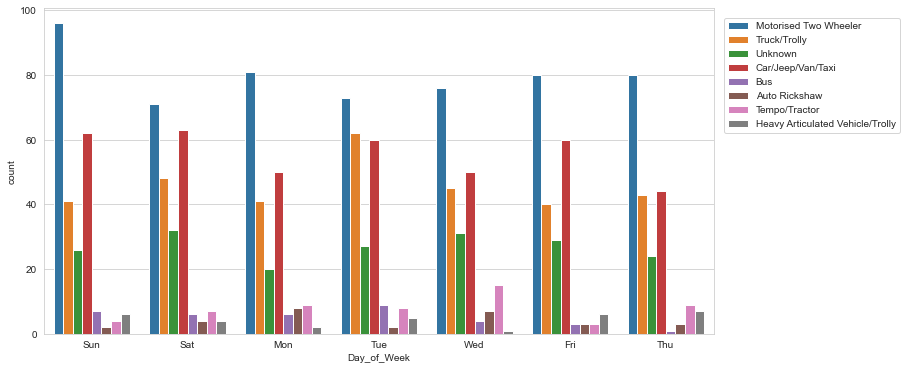
**#Vehiclewise no. of accidents by weekday**

plt.figure(figsize=(12, 6))

sns.countplot(x='Day\_of\_Week',data=ds,hue=ds['Vehicle'])

**# To relocate the legend**

plt.legend(bbox\_to\_anchor=(1, 1), loc=2, borderaxespad=1)



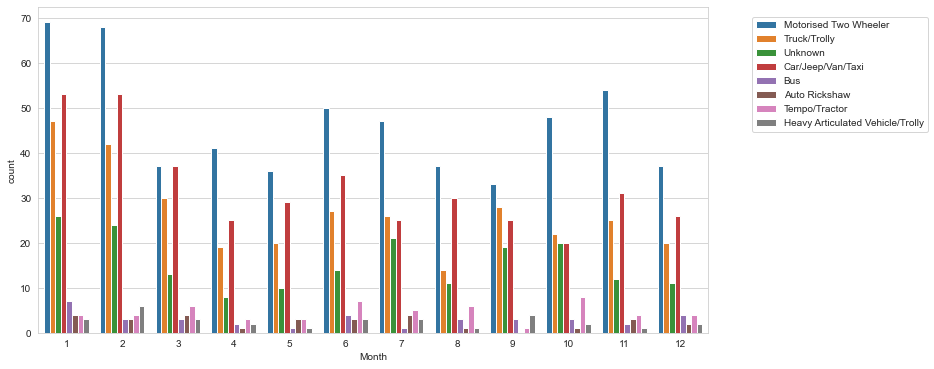
**#Vehiclewise nos. of accidents by month**

plt.figure(figsize=(12, 6))

sns.countplot(x='Month',data=ds,hue=ds['Vehicle'])

**# To relocate the legend**

plt.legend(bbox\_to\_anchor=(1.05, 1), loc=2, borderaxespad=1)



**#Count datewise no. of accidents**

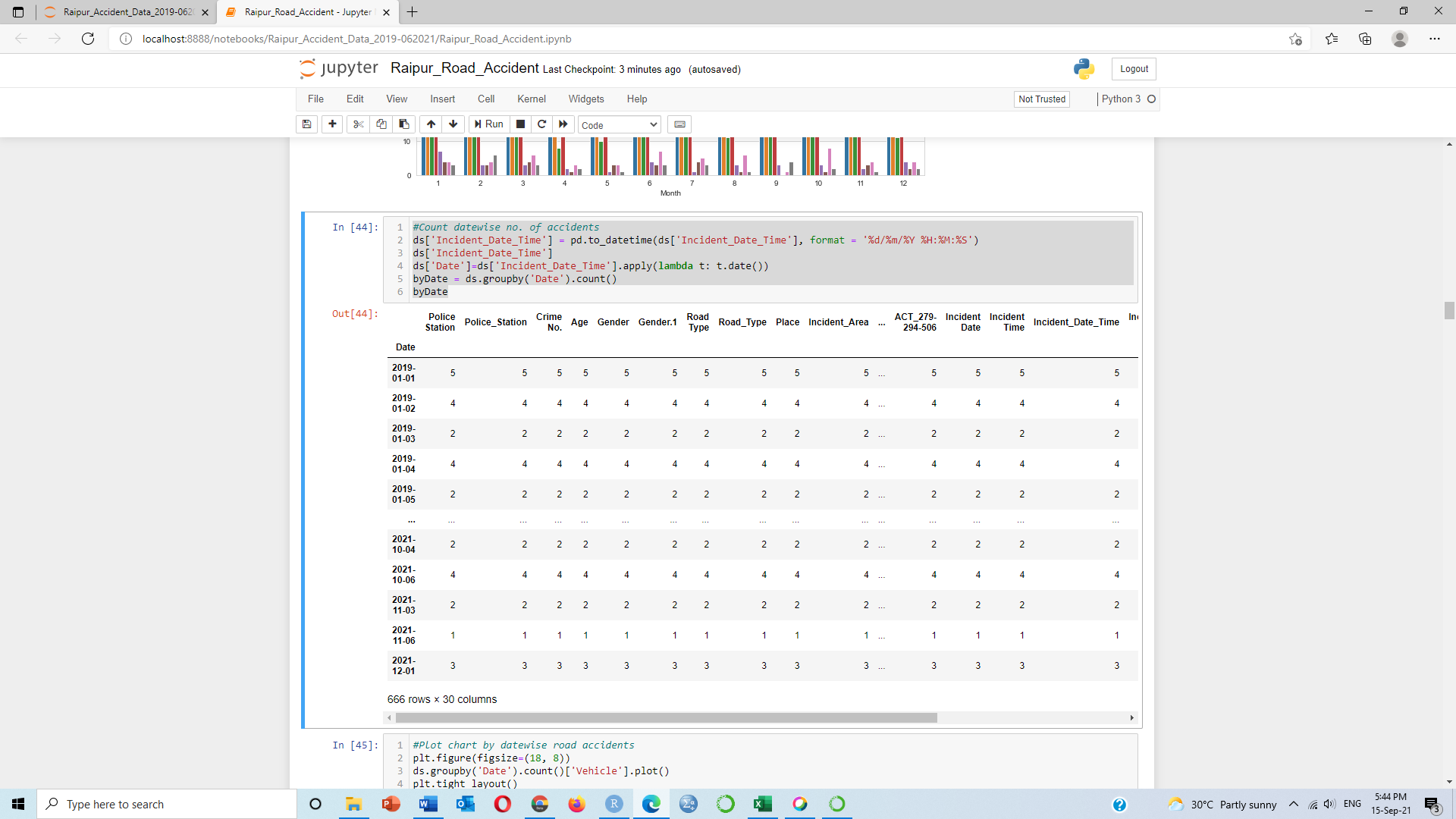
ds['Incident\_Date\_Time'] = pd.to\_datetime(ds['Incident\_Date\_Time'], format = '%d/%m/%Y %H:%M:%S')

ds['Incident\_Date\_Time']

ds['Date']=ds['Incident\_Date\_Time'].apply(lambda t: t.date())

byDate = ds.groupby('Date').count()

byDate



**#Plot chart by date wise road accidents**

plt.figure(figsize=(18, 8))

ds.groupby('Date').count()['Vehicle'].plot()

plt.tight\_layout()



**#Count every weekday, hour wise no of accidents**

dayHour = ds.groupby(by=['Day\_of\_Week','Hour']).count()['Vehicle'].unstack()

dayHour

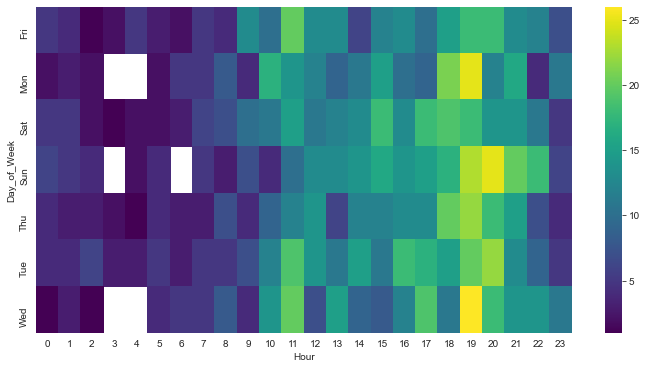
| **Hour** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **...** | **14** | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Day\_of\_Week** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Fri** | 5.0 | 4.0 | 1.0 | 2.0 | 5.0 | 3.0 | 2.0 | 5.0 | 4.0 | 13.0 | ... | 6.0 | 12.0 | 13.0 | 10.0 | 15.0 | 18.0 | 18.0 | 13.0 | 12.0 | 7.0 |
| **Mon** | 2.0 | 3.0 | 2.0 | NaN | NaN | 2.0 | 5.0 | 5.0 | 8.0 | 4.0 | ... | 11.0 | 15.0 | 10.0 | 9.0 | 21.0 | 25.0 | 12.0 | 16.0 | 4.0 | 11.0 |
| **Sat** | 5.0 | 5.0 | 2.0 | 1.0 | 2.0 | 2.0 | 3.0 | 6.0 | 7.0 | 10.0 | ... | 13.0 | 18.0 | 13.0 | 18.0 | 19.0 | 18.0 | 14.0 | 14.0 | 11.0 | 5.0 |
| **Sun** | 6.0 | 5.0 | 4.0 | NaN | 2.0 | 4.0 | NaN | 5.0 | 3.0 | 7.0 | ... | 14.0 | 16.0 | 14.0 | 15.0 | 17.0 | 23.0 | 25.0 | 20.0 | 18.0 | 6.0 |
| **Thu** | 4.0 | 3.0 | 3.0 | 2.0 | 1.0 | 4.0 | 3.0 | 3.0 | 7.0 | 4.0 | ... | 12.0 | 12.0 | 13.0 | 13.0 | 20.0 | 22.0 | 18.0 | 15.0 | 7.0 | 4.0 |
| **Tue** | 4.0 | 4.0 | 6.0 | 3.0 | 3.0 | 5.0 | 3.0 | 5.0 | 5.0 | 7.0 | ... | 15.0 | 11.0 | 18.0 | 17.0 | 15.0 | 20.0 | 22.0 | 13.0 | 9.0 | 5.0 |
| **Wed** | 1.0 | 3.0 | 1.0 | NaN | NaN | 4.0 | 5.0 | 5.0 | 8.0 | 4.0 | ... | 9.0 | 8.0 | 12.0 | 19.0 | 11.0 | 26.0 | 18.0 | 14.0 | 14.0 | 11.0 |

7 rows × 24 columns

**#Heatmap of above output**

plt.figure(figsize=(12,6))

sns.heatmap(dayHour,cmap='viridis')



**#Count every weekday, month wise no of accidents**

dayMonth = ds.groupby(by=['Day\_of\_Week','Month']).count()['Vehicle'].unstack()

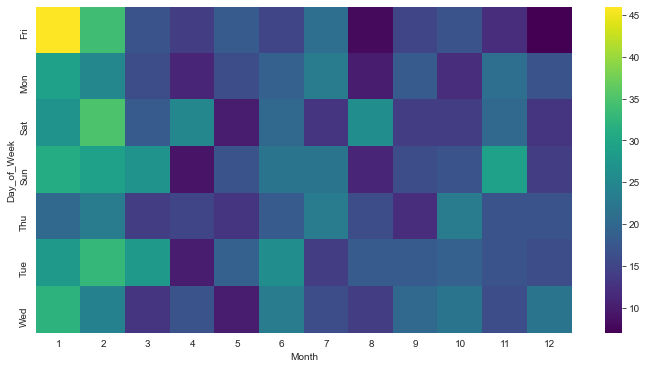
dayMonth

| **Month** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Day\_of\_Week** |  |  |  |  |  |  |  |  |  |  |  |  |
| **Fri** | 46 | 34 | 17 | 14 | 18 | 15 | 21 | 8 | 15 | 17 | 12 | 7 |
| **Mon** | 29 | 25 | 16 | 11 | 16 | 19 | 23 | 10 | 18 | 12 | 21 | 17 |
| **Sat** | 27 | 35 | 18 | 25 | 10 | 20 | 13 | 26 | 14 | 14 | 20 | 13 |
| **Sun** | 31 | 29 | 27 | 9 | 17 | 22 | 22 | 11 | 16 | 17 | 29 | 14 |
| **Thu** | 20 | 23 | 14 | 15 | 13 | 18 | 23 | 16 | 12 | 23 | 17 | 17 |
| **Tue** | 28 | 33 | 28 | 10 | 19 | 26 | 14 | 18 | 18 | 19 | 17 | 16 |
| **Wed** | 32 | 24 | 13 | 17 | 10 | 23 | 16 | 14 | 20 | 22 | 16 | 22 |

**#Draw heatmap of above output**

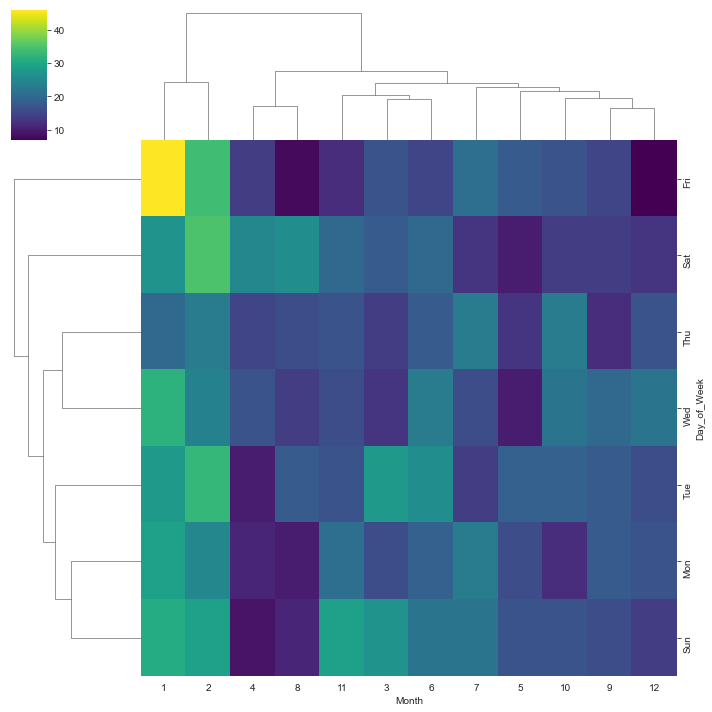
plt.figure(figsize=(12,6))

sns.heatmap(dayMonth,cmap='viridis')



**#Cluster map of Weekday and Month wise**

sns.clustermap(dayMonth,cmap='viridis')



**Cluster**

**#Crime data according to Latitude and Longitude**

plt.figure(figsize=(14,8))

plt.plot(ds['Longitude'], ds['Latitude'],

marker='\*', linewidth=0, color='#128128')

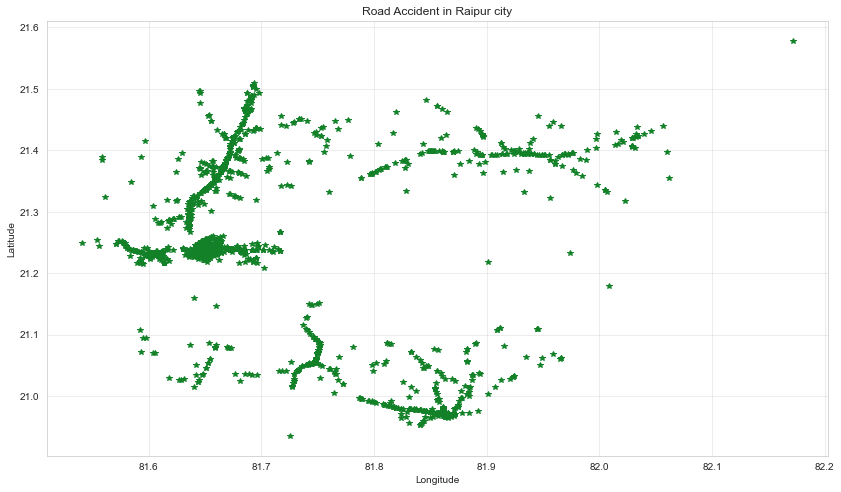
plt.grid(which='major', color='#cccccc', alpha=0.45)

plt.title('Road Accident in Raipur city', family='Arial', fontsize=12)

plt.xlabel('Longitude')

plt.ylabel('Latitude')

plt.show()



dbscan\_data = ds[['Longitude', 'Latitude']]

dbscan\_data = dbscan\_data.values.astype('float32', copy=False)

dbscan\_data

array([[81.63955 , 21.32514 ],

[81.75226 , 21.050568],

[81.73591 , 21.04815 ],

...,

[81.68838 , 21.223051],

[81.689735, 21.221146],

[81.67191 , 21.238594]], dtype=float32)

dbscan\_data\_scaler = StandardScaler().fit(dbscan\_data)

dbscan\_data = dbscan\_data\_scaler.transform(dbscan\_data)

dbscan\_data

array([[-0.76666385, 0.57104236],

[ 0.254068 , -1.3079445 ],

[ 0.10599867, -1.3244952 ],

...,

[-0.32445955, -0.12758492],

[-0.31216076, -0.14062446],

[-0.4735653 , -0.0212193 ]], dtype=float32)

model = DBSCAN(eps = 0.25, min\_samples = 6, metric='euclidean').\

fit(dbscan\_data)

model

DBSCAN(eps=0.25, min\_samples=6)

outliers\_df = ds[model.labels\_ == -1]

clusters\_df = ds[model.labels\_ != -1]

colors = model.labels\_

colors\_clusters = colors[colors != -1]

color\_outliers = 'black'

clusters = Counter(model.labels\_)

print(clusters)

print(ds[model.labels\_ == -1].head())

print('Number of clusters = {}'.format(len(clusters)-1))

Counter({0: 998, 1: 393, 2: 180, -1: 29, 3: 6})

Police Station Police\_Station Crime No. Age Gender Gender.1 Road Type \

52 KHARORA 7 280/20 42 M 0 SH

55 KHARORA 7 121/21 45 M 0 SH

227 GOBRA NAWAPARA 5 85/20 25 M 0 O

242 GOBRA NAWAPARA 5 57/20 29 M 0 O

248 GOBRA NAWAPARA 5 267/20 25 M 0 O

Road\_Type Place Incident\_Area ... Incident Date Incident Time \

52 1 R 1 ... 31-08-20 6:00

55 1 R 1 ... 21-03-21 5:15

227 2 R 1 ... 20-02-20 12:30

242 2 R 1 ... 28-12-19 17:00

248 2 R 1 ... 12-07-20 18:30

Incident\_Date\_Time Incident Place Latitude \

52 2020-08-31 06:00:00 gram chicholi mukhaya marg 21.482079

55 2021-03-21 05:15:00 gram broad bhatha 21.331386

227 2020-02-20 12:30:00 ग्राम आलेखुटा कुर्रा मार्ग 20.934855

242 2019-12-28 17:00:00 ग्राम सुन्दरकेरा पारागांव रोड 21.179444

248 2020-12-07 18:30:00 तोरला के आगे भुरका के पास मेन रोड 21.108753

Longitude Hour Month Day\_of\_Week Date

52 81.845829 6 8 Mon 2020-08-31

55 81.759960 5 3 Sun 2021-03-21

227 81.725153 12 2 Thu 2020-02-20

242 82.008471 17 12 Sat 2019-12-28

248 81.945013 18 12 Mon 2020-12-07

[5 rows x 31 columns]

Number of clusters = 4

fig = plt.figure(figsize=(14,8))

ax = fig.add\_axes([.1, .1, 1,1])

ax.scatter(clusters\_df['Longitude'], clusters\_df['Latitude'],

c = colors\_clusters, edgecolors='black', s=50)

ax.scatter(outliers\_df['Longitude'], outliers\_df['Latitude'],

c = color\_outliers, edgecolors='black', s=50)

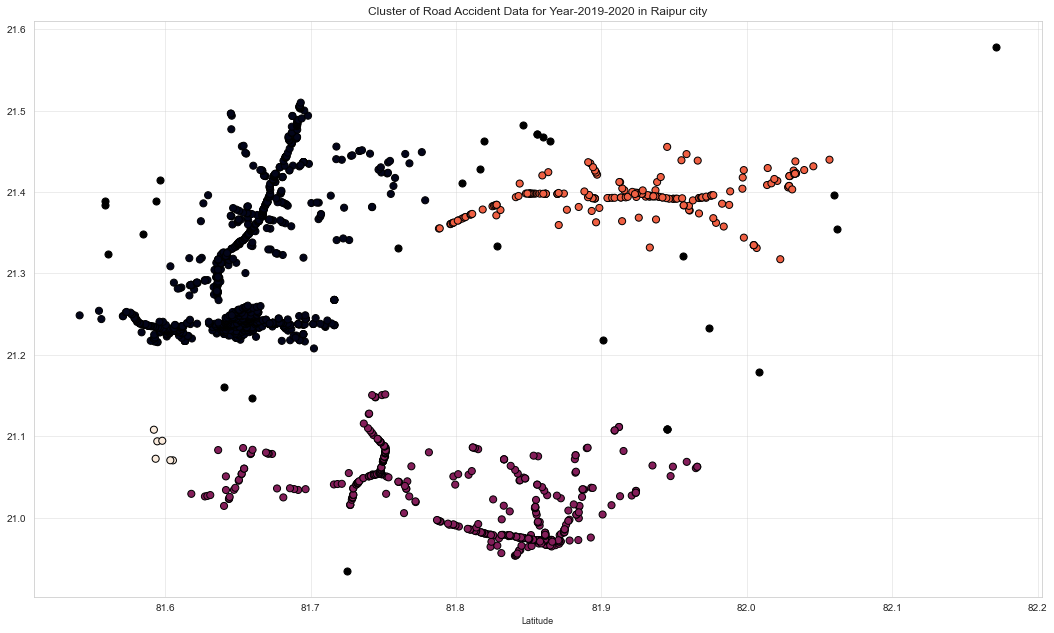
ax.set\_xlabel('Longitude', family='Arial', fontsize=9)

ax.set\_xlabel('Latitude', family='Arial', fontsize=9)

plt.title('Cluster of Road Accident Data for Year-2019-2020 in Raipur city', Family='Arial', fontsize=12)

plt.grid(which='major', color='#cccccc', alpha=0.45)

plt.show()



**#Show location wise accidents in Map**

locations = ds[['Latitude', 'Longitude']]

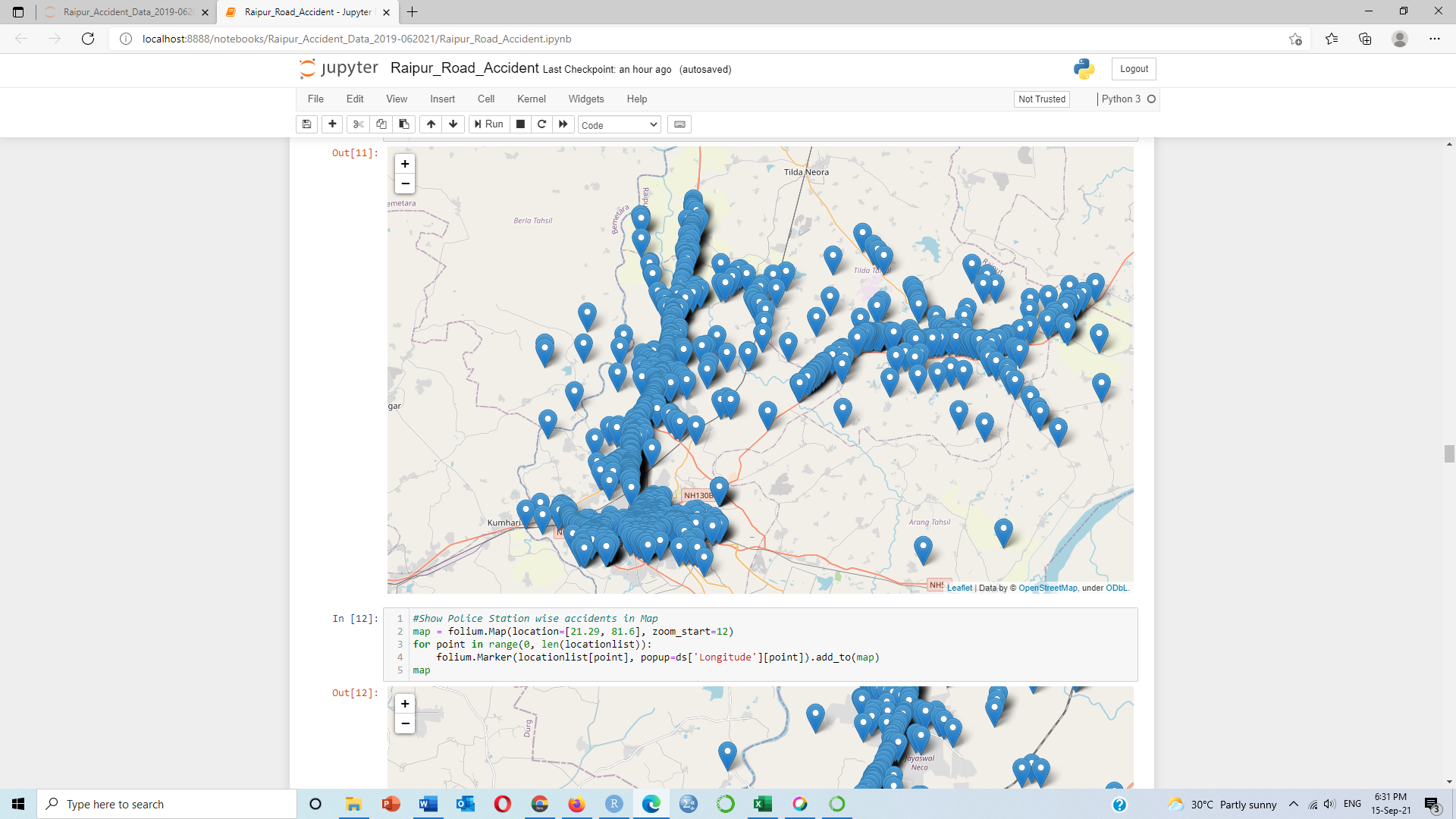
locationlist = locations.values.tolist()

map = folium.Map(location=[21.29, 81.6], zoom\_start=12)

for point in range(0, len(locationlist)):

folium.Marker(locationlist[point], popup=ds['Latitude'][point]).add\_to(map)

map



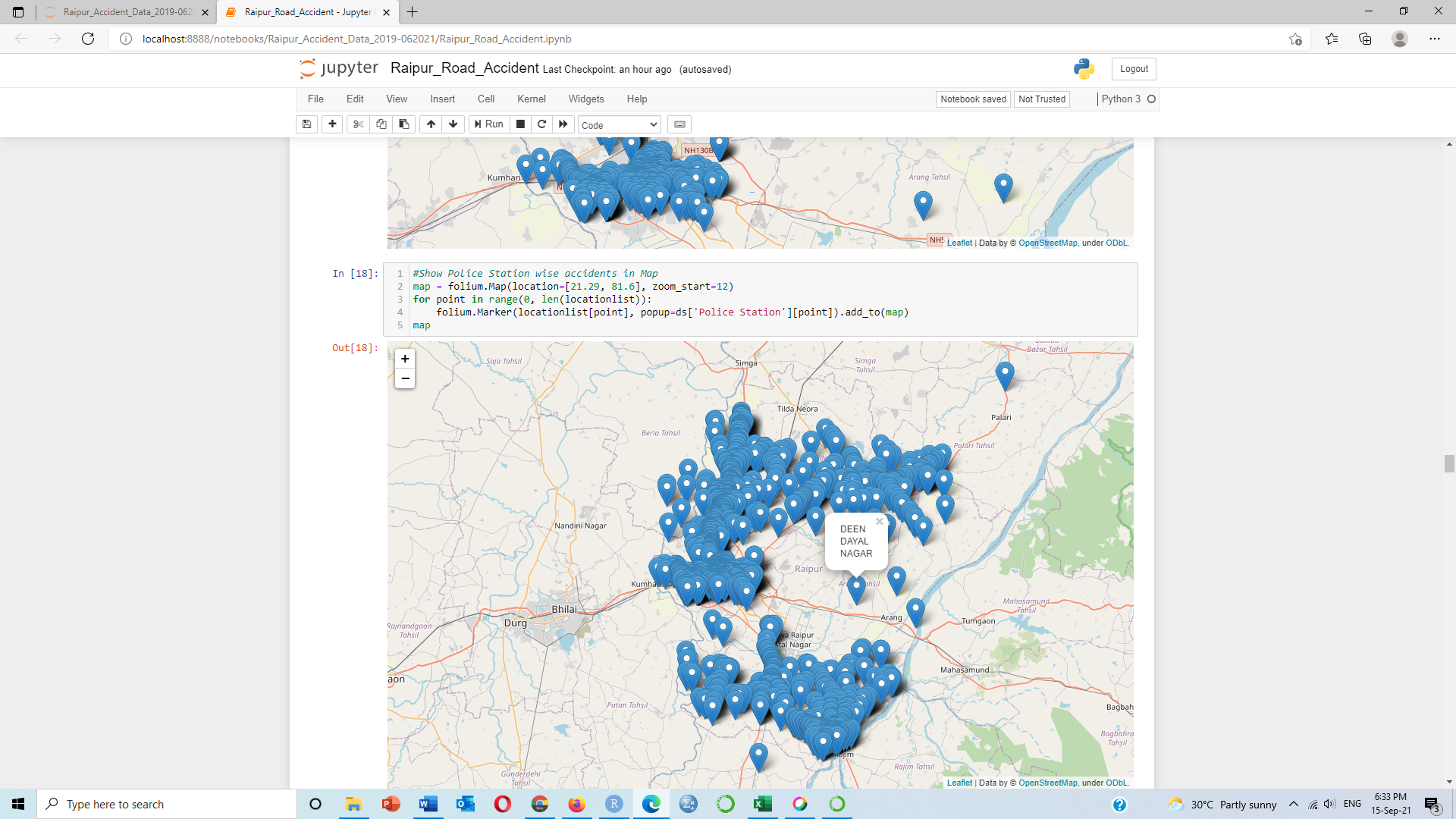
**#Show Police Station wise accidents in Map**

map = folium.Map(location=[21.29, 81.6], zoom\_start=12)

for point in range(0, len(locationlist)):

folium.Marker(locationlist[point], popup=ds['Police Station'][point]).add\_to(map)

map



**#Count location wise accidents and give the color and size according to the no of accidents**

location=ds.groupby(['Latitude','Longitude','Police\_Station','Road\_Type']).size().reset\_index(name='count').sort\_values(by='count',ascending=False)

location=location[location['count']>0]

location['color']=location['count'].apply(lambda count:"Black" if count>=11 else

"blue" if count>=9 and count<11 else

"Orange" if count>=7 and count<9 else

"red" if count>=5 and count<7 else

"navyblue" if count>=3 and count<5 else

"green" if count>=1 and count<3 else

"grey")

location['size']=location['count'].apply(lambda count:12 if count>=11 else

10 if count>=9 and count<11 else

8 if count>=7 and count<9 else

6 if count>=5 and count<7 else

4 if count>=3 and count<5 else

2 if count>=1 and count<3 else

1)

location

| **Latitude** | **Longitude** | **Police\_Station** | **Road\_Type** | **count** | **color** | **size** |
| --- | --- | --- | --- | --- | --- | --- |
| **933** | 21.304821 | 81.636473 | 6 | 2 | 13 | Black | 12 |
| **669** | 21.238622 | 81.671455 | 9 | 0 | 12 | Black | 12 |
| **702** | 21.239277 | 81.674033 | 9 | 0 | 7 | Orange | 8 |
| **396** | 21.217220 | 81.613670 | 9 | 1 | 6 | red | 6 |
| **900** | 21.288320 | 81.636330 | 6 | 0 | 6 | red | 6 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **498** | 21.230702 | 81.609828 | 3 | 2 | 1 | green | 2 |
| **497** | 21.230677 | 81.656739 | 2 | 2 | 1 | green | 2 |
| **496** | 21.230673 | 81.641675 | 8 | 2 | 1 | green | 2 |
| **495** | 21.230537 | 81.600911 | 3 | 2 | 1 | green | 2 |
| **1480** | 21.578364 | 82.171362 | 7 | 2 | 1 | green | 2 |

1481 rows × 7 columns

**#Show the density of accidents**

m=folium.Map([21.29, 81.6],zoom\_start=11)

#location=location[0:2000]

for lat,lon,area,loc,color,count,size in zip(location['Latitude'],location['Longitude'],location['Police\_Station'],location['Road\_Type'],location['color'],location['count'],location['size']):

text = """Police Station:{}<br>

Latitide:{}<br>

Longitude:{}<br>

Location:{}<br>

No. of incidents: {} <br>"""

text = text.format(area,lat,lon,loc, count)

folium.CircleMarker([lat, lon],

popup=text,

radius=size,

color='b',

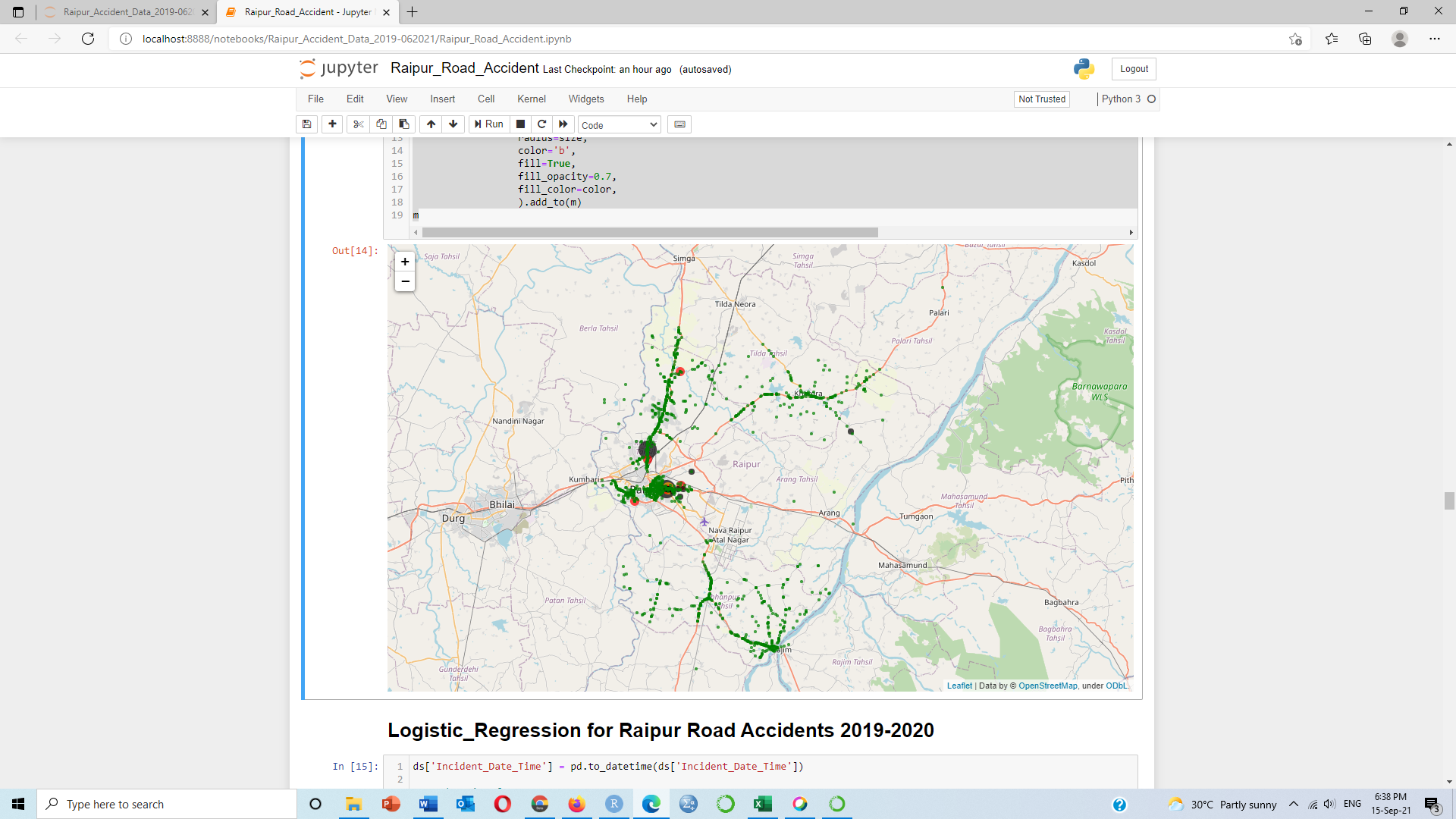
fill=True,

fill\_opacity=0.7,

fill\_color=color,

).add\_to(m)

m



**Logistic\_Regression for Raipur Road Accidents 2019-2020**

ds['Incident\_Date\_Time'] = pd.to\_datetime(ds['Incident\_Date\_Time'])

#Set datetime format

ds['Incident\_Date\_Time'] = pd.to\_datetime(ds['Incident\_Date\_Time'], format = '%d/%m/%Y %H:%M:%S')

ds['Incident\_Date\_Time']

0 2019-09-08 17:00:00

1 2019-07-21 17:30:00

2 2019-06-08 19:00:00

3 2020-12-21 17:15:00

4 2019-04-01 18:00:00

...

1601 2021-12-01 23:00:00

1602 2019-12-13 07:55:00

1603 2019-11-23 22:30:00

1604 2019-11-17 22:43:00

1605 2019-01-20 18:00:00

Name: Incident\_Date\_Time, Length: 1606, dtype: datetime64[ns]

**#Convert data into feature columns**

column\_1 = ds.iloc[:,23]

db=pd.DataFrame({"year": column\_1.dt.year,

"month": column\_1.dt.month,

"day": column\_1.dt.day,

"hour": column\_1.dt.hour,

"dayofyear": column\_1.dt.dayofyear,

"week": column\_1.dt.week,

"weekofyear": column\_1.dt.weekofyear,

"dayofweek": column\_1.dt.dayofweek,

"weekday": column\_1.dt.weekday,

"quarter": column\_1.dt.quarter,

})

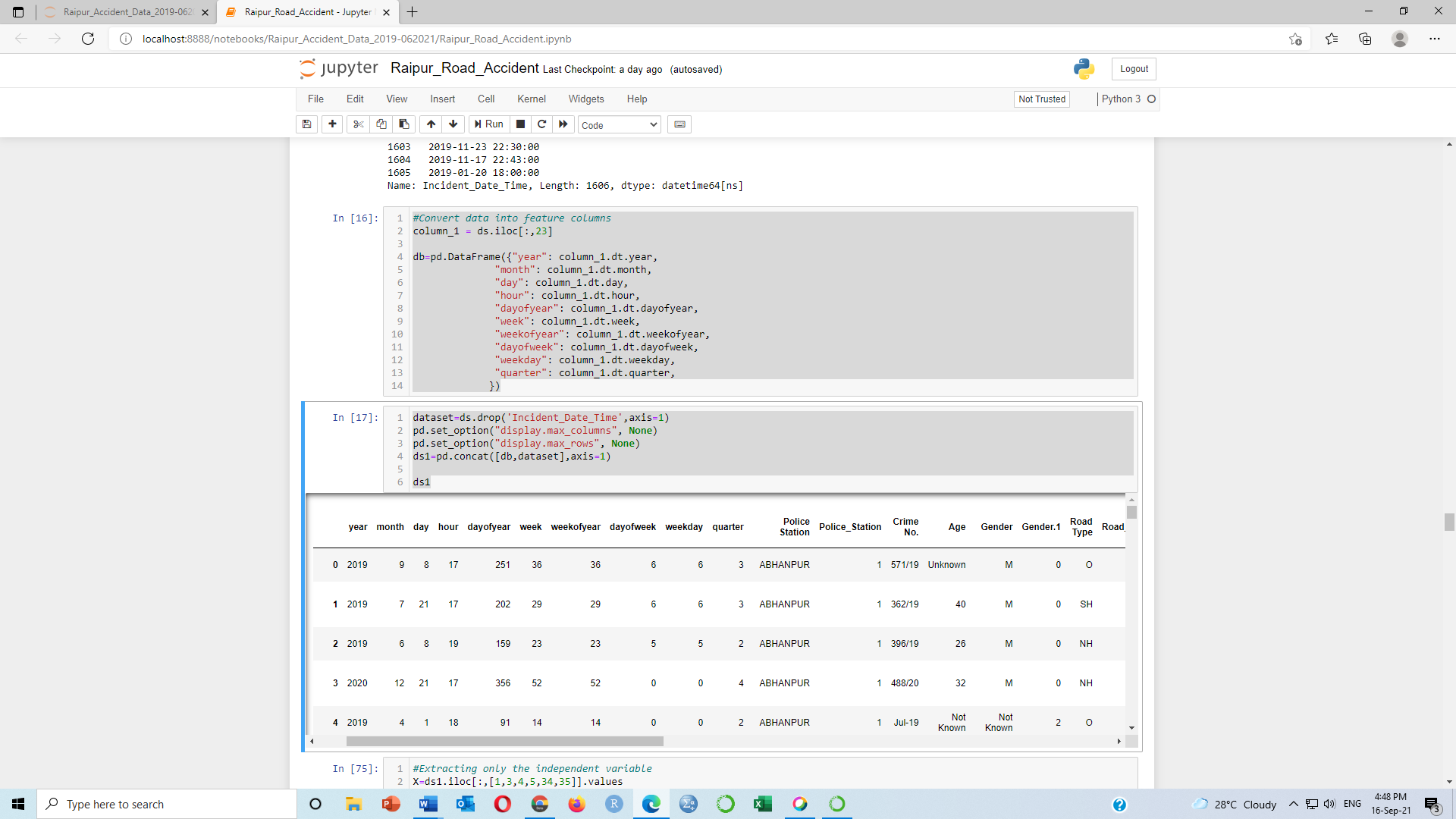
dataset=ds.drop('Incident\_Date\_Time',axis=1)

pd.set\_option("display.max\_columns", None)

pd.set\_option("display.max\_rows", None)

ds1=pd.concat([db,dataset],axis=1)

ds1



**#Extracting only the independent variable**

X=ds1.iloc[:,[1,3,4,5,34,35]].values

X

array([[ 9. , 17. , 251. , 36. , 21.32514 ,

81.639546],

[ 7. , 17. , 202. , 29. , 21.050567,

81.752261],

[ 6. , 19. , 159. , 23. , 21.048149,

81.73591 ],

...,

[ 11. , 22. , 327. , 47. , 21.223051,

81.688381],

[ 11. , 22. , 321. , 46. , 21.221146,

81.689735],

[ 1. , 18. , 20. , 3. , 21.238594,

81.671913]])

**#ACT\_279-304-A**

y=ds1.iloc[:,[29]].values

array([[1],

[0],

[0],

...,

[0],

[0],

[0]], dtype=int64)

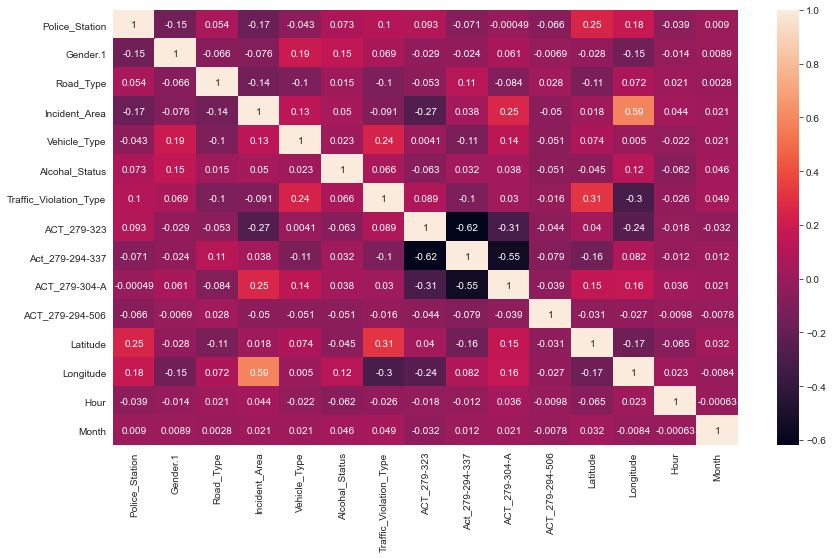
ds.corr()

| **Police\_Station** | **Gender.1** | **Road\_Type** | **Incident\_Area** | **Vehicle\_Type** | **Alcohal\_Status** | **Traffic\_Violation\_Type** | **ACT\_279-323** | **Act\_279-294-337** | **ACT\_279-304-A** | **ACT\_279-294-506** | **Latitude** | **Longitude** | **Hour** | **Month** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Police\_Station** | 1.000000 | -0.145291 | 0.053526 | -0.171674 | -0.042861 | 0.072582 | 0.104197 | 0.092558 | -0.070670 | -0.000490 | -0.066047 | 0.248858 | 0.176565 | -0.039079 | 0.009044 |
| **Gender.1** | -0.145291 | 1.000000 | -0.065653 | -0.075574 | 0.190825 | 0.153263 | 0.068831 | -0.028543 | -0.023981 | 0.060957 | -0.006888 | -0.028022 | -0.146211 | -0.013524 | 0.008894 |
| **Road\_Type** | 0.053526 | -0.065653 | 1.000000 | -0.137464 | -0.104367 | 0.015403 | -0.101481 | -0.052833 | 0.111221 | -0.084466 | 0.028291 | -0.109770 | 0.072339 | 0.020688 | 0.002833 |
| **Incident\_Area** | -0.171674 | -0.075574 | -0.137464 | 1.000000 | 0.128678 | 0.050295 | -0.090938 | -0.274188 | 0.038406 | 0.254803 | -0.050235 | 0.017747 | 0.587941 | 0.043789 | 0.021218 |
| **Vehicle\_Type** | -0.042861 | 0.190825 | -0.104367 | 0.128678 | 1.000000 | 0.022836 | 0.239134 | 0.004092 | -0.107793 | 0.136498 | -0.051429 | 0.073622 | 0.005014 | -0.021804 | 0.020921 |
| **Alcohal\_Status** | 0.072582 | 0.153263 | 0.015403 | 0.050295 | 0.022836 | 1.000000 | 0.065673 | -0.063299 | 0.031863 | 0.037920 | -0.050794 | -0.044563 | 0.116085 | -0.062119 | 0.045586 |
| **Traffic\_Violation\_Type** | 0.104197 | 0.068831 | -0.101481 | -0.090938 | 0.239134 | 0.065673 | 1.000000 | 0.088937 | -0.099850 | 0.029875 | -0.016186 | 0.312524 | -0.303619 | -0.026070 | 0.049455 |
| **ACT\_279-323** | 0.092558 | -0.028543 | -0.052833 | -0.274188 | 0.004092 | -0.063299 | 0.088937 | 1.000000 | -0.617445 | -0.305553 | -0.044098 | 0.039618 | -0.241071 | -0.018334 | -0.032096 |
| **Act\_279-294-337** | -0.070670 | -0.023981 | 0.111221 | 0.038406 | -0.107793 | 0.031863 | -0.099850 | -0.617445 | 1.000000 | -0.546753 | -0.078908 | -0.156176 | 0.081683 | -0.011901 | 0.011671 |
| **ACT\_279-304-A** | -0.000490 | 0.060957 | -0.084466 | 0.254803 | 0.136498 | 0.037920 | 0.029875 | -0.305553 | -0.546753 | 1.000000 | -0.039049 | 0.153857 | 0.162388 | 0.035867 | 0.021424 |
| **ACT\_279-294-506** | -0.066047 | -0.006888 | 0.028291 | -0.050235 | -0.051429 | -0.050794 | -0.016186 | -0.044098 | -0.078908 | -0.039049 | 1.000000 | -0.030796 | -0.026655 | -0.009830 | -0.007792 |
| **Latitude** | 0.248858 | -0.028022 | -0.109770 | 0.017747 | 0.073622 | -0.044563 | 0.312524 | 0.039618 | -0.156176 | 0.153857 | -0.030796 | 1.000000 | -0.166642 | -0.065362 | 0.031802 |
| **Longitude** | 0.176565 | -0.146211 | 0.072339 | 0.587941 | 0.005014 | 0.116085 | -0.303619 | -0.241071 | 0.081683 | 0.162388 | -0.026655 | -0.166642 | 1.000000 | 0.023266 | -0.008390 |
| **Hour** | -0.039079 | -0.013524 | 0.020688 | 0.043789 | -0.021804 | -0.062119 | -0.026070 | -0.018334 | -0.011901 | 0.035867 | -0.009830 | -0.065362 | 0.023266 | 1.000000 | -0.000627 |
| **Month** | 0.009044 | 0.008894 | 0.002833 | 0.021218 | 0.020921 | 0.045586 | 0.049455 | -0.032096 | 0.011671 | 0.021424 | -0.007792 | 0.031802 | -0.008390 | -0.000627 | 1.000000 |

**#Visualising the dataset by drawing the correlation map using numerical features**

fig = plt.figure(figsize=(14,8))

sns.heatmap(ds.corr(), annot=True)



**#Splitting the data into Training and Test data**

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.30, random\_state = 0)

X\_train

array([[ 9. , 14. , 253. , 37. , 21.36426 ,

81.624626],

[ 11. , 16. , 321. , 46. , 21.405463,

81.672005],

[ 5. , 0. , 128. , 19. , 21.357707,

81.984145],

...,

[ 8. , 18. , 224. , 33. , 21.372357,

81.666728],

[ 8. , 11. , 215. , 31. , 21.229605,

81.605794],

[ 8. , 18. , 215. , 31. , 21.236911,

81.685999]])

**#Feature scaling**

from sklearn.preprocessing import StandardScaler

ss\_X = StandardScaler()

X\_train = ss\_X.fit\_transform(X\_train)

X\_test = ss\_X.transform(X\_test)

**#Apply logistic regression**

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression()

model.fit(X\_train, y\_train)

X\_test

array([[ 0.31136517, -0.13747802, 0.33719605, 0.31229063, 0.43972424,

-0.80045702],

[ 1.42151411, -1.21306013, 1.34407716, 1.33157096, 1.37892877,

-0.38173725],

[-1.076321 , 1.47589514, -1.01751672, -1.02551481, -1.90253598,

1.06203963],

...,

[-0.2437093 , 0.22104935, -0.21201183, -0.19734954, 0.03289567,

-0.74099859],

[ 0.58890241, 0.7588404 , 0.62095345, 0.63081573, 0.9030193 ,

-0.59194307],

[ 1.14397688, -0.13747802, 1.20677519, 1.20416092, 1.0713065 ,

1.22629498]])

**#Predict the model**

y\_pred= model.predict(X\_test)

y\_pred

array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,

1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

dtype=int64)

**#Predict the probability**

model.predict\_proba(X\_test)

array([[0.80929464, 0.19070536],

[0.73596722, 0.26403278],

[0.8284424 , 0.1715576 ],

[0.74798976, 0.25201024],

[0.76368397, 0.23631603],

[0.67700093, 0.32299907],

[0.82016768, 0.17983232],

[0.80755299, 0.19244701],

[0.84574586, 0.15425414],

[0.76221737, 0.23778263],

[0.87854094, 0.12145906],

[0.8029636 , 0.1970364 ],

[0.85618432, 0.14381568],

[0.75956135, 0.24043865],

[0.85523121, 0.14476879],

[0.74910706, 0.25089294],

[0.84237299, 0.15762701],

[0.85855292, 0.14144708],

[0.79147472, 0.20852528],

**#Check the score/accuracy of testing\_training data**

B=model.score(X\_test,y\_test)

B = B\*100

print('Accuracy of Testing data: %.3f' %B)

C=model.score(X\_train,y\_train)

C = C\*100

print('Accuracy of Training data: %.3f' %C)

**Accuracy of Testing data: 80.083**

**Accuracy of Training data: 77.847**

**#Create confusion matrix**

from sklearn import metrics

from sklearn.metrics import confusion\_matrix,accuracy\_score

cm = metrics.confusion\_matrix(y\_test, y\_pred)

print(cm)

[[380 5]

[ 91 6]]

**#Create classification report**

from sklearn.metrics import classification\_report

print(classification\_report(y\_test, y\_pred))

precision recall f1-score support

0 0.81 0.99 0.89 385

1 0.55 0.06 0.11 97

accuracy 0.80 482

macro avg 0.68 0.52 0.50 482

weighted avg 0.75 0.80 0.73 482

**Decision\_Tree for Raipur Road Accident 2019-2020**

dtree = DecisionTreeClassifier(max\_leaf\_nodes=10, random\_state=0)

dtree.fit(X\_train,y\_train)

DecisionTreeClassifier(max\_leaf\_nodes=10, random\_state=0)

**#Predict the data**

y\_pred=dtree.predict(X\_test)

y\_pred

array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

dtype=int64)

**#Check accuarcy**

dtree.score(X\_test,y\_test)

**0.7987551867219918**

**#Create confusion matrix**

from sklearn import metrics

from sklearn.metrics import confusion\_matrix,accuracy\_score

cm = metrics.confusion\_matrix(y\_test, y\_pred)

print(cm)

[[380 5]

[ 92 5]]

**#Evaluation of the model**

from sklearn.metrics import classification\_report, confusion\_matrix

print(classification\_report(y\_test, y\_pred))

precision recall f1-score support

0 0.81 0.99 0.89 385

1 0.50 0.05 0.09 97

accuracy 0.80 482

macro avg 0.65 0.52 0.49 482

weighted avg 0.74 0.80 0.73 482

**Plot Decision Tree**

**#Put latitude into X1**

X1 = ds.iloc[:,[26]].values

**#Put ACT\_279-304-A into y1**

y1 = ds.iloc[:,[19]].values

**# Fit the classifier with default hyper-parameters**

clf = DecisionTreeClassifier(max\_leaf\_nodes=20,random\_state=0)

model = clf.fit(X1, y1)

text\_representation = tree.export\_text(clf)

print(text\_representation)

|--- feature\_0 <= 81.89

| |--- feature\_0 <= 81.67

| | |--- feature\_0 <= 81.56

| | | |--- class: 1

| | |--- feature\_0 > 81.56

| | | |--- class: 0

| |--- feature\_0 > 81.67

| | |--- feature\_0 <= 81.67

| | | |--- class: 1

| | |--- feature\_0 > 81.67

| | | |--- feature\_0 <= 81.83

| | | | |--- feature\_0 <= 81.75

| | | | | |--- feature\_0 <= 81.75

| | | | | | |--- class: 0

| | | | | |--- feature\_0 > 81.75

| | | | | | |--- class: 0

| | | | |--- feature\_0 > 81.75

| | | | | |--- feature\_0 <= 81.76

| | | | | | |--- class: 1

| | | | | |--- feature\_0 > 81.76

| | | | | | |--- class: 0

| | | |--- feature\_0 > 81.83

| | | | |--- class: 0

|--- feature\_0 > 81.89

| |--- feature\_0 <= 82.03

| | |--- feature\_0 <= 81.98

| | | |--- feature\_0 <= 81.96

| | | | |--- feature\_0 <= 81.96

| | | | | |--- feature\_0 <= 81.89

| | | | | | |--- class: 1

| | | | | |--- feature\_0 > 81.89

| | | | | | |--- feature\_0 <= 81.93

| | | | | | | |--- feature\_0 <= 81.93

| | | | | | | | |--- feature\_0 <= 81.92

| | | | | | | | | |--- feature\_0 <= 81.92

| | | | | | | | | | |--- class: 0

| | | | | | | | | |--- feature\_0 > 81.92

| | | | | | | | | | |--- class: 0

| | | | | | | | |--- feature\_0 > 81.92

| | | | | | | | | |--- class: 1

| | | | | | | |--- feature\_0 > 81.93

| | | | | | | | |--- class: 0

| | | | | | |--- feature\_0 > 81.93

| | | | | | | |--- feature\_0 <= 81.94

| | | | | | | | |--- class: 1

| | | | | | | |--- feature\_0 > 81.94

| | | | | | | | |--- class: 0

| | | | |--- feature\_0 > 81.96

| | | | | |--- class: 1

| | | |--- feature\_0 > 81.96

| | | | |--- class: 0

| | |--- feature\_0 > 81.98

| | | |--- class: 1

| |--- feature\_0 > 82.03

| | |--- feature\_0 <= 82.05

| | | |--- class: 0

| | |--- feature\_0 > 82.05

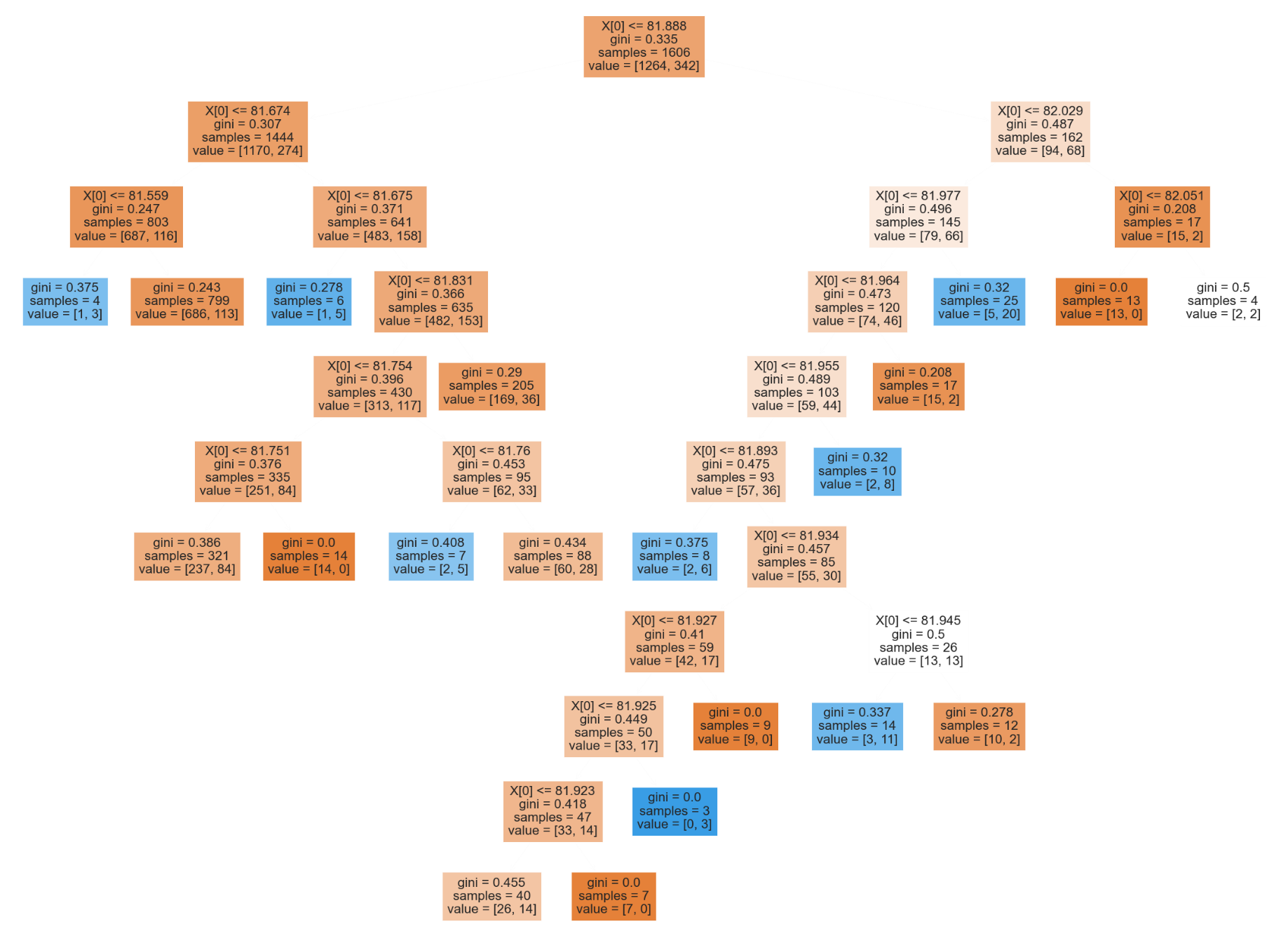
| | | |--- class: 0

**#Longitude value with ACT\_279-304-A**

fig = plt.figure(figsize=(40,30))

\_ = tree.plot\_tree(clf,

filled=True)



**Time Series Forecasting**

df = pd.read\_csv('Raipur\_Monthwise\_Accidents.csv', parse\_dates=['Month'], index\_col='Month')

df

**Accidents**

**Month**

2019-01-01 91

2019-02-01 71

2019-03-01 58

2019-04-01 73

2019-05-01 62

2019-06-01 70

2019-07-01 68

2019-08-01 56

2019-09-01 53

2019-10-01 48

2019-11-01 52

2019-12-01 57

2020-01-01 93

2020-02-01 83

2020-03-01 54

2020-04-01 9

2020-05-01 37

2020-06-01 52

2020-07-01 52

2020-08-01 60

2020-09-01 51

2020-10-01 57

2020-11-01 65

2020-12-01 41

2021-01-01 51

2021-02-01 68

2021-03-01 41

import statsmodels.api as sm

model = sm.tsa.statespace.SARIMAX(df['Accidents'], order = (1,1,1), seasonal\_order=(1,1,1,12))

results = model.fit()

from pandas.tseries.offsets import DateOffset

future\_dates = [df.index[-1]+DateOffset(months=x) for x in range(0,24)]

**#Orange line are Forecast value respect of original value**

df['forecast']=results.predict(start=10,end=100,dynamic=True)

df[['Accidents','forecast']].plot(figsize=(12,8))

future\_dates\_ds = pd.DataFrame(index= future\_dates[1:], columns=df.columns)

future\_dates\_ds.tail()

**Accidents forecast**

2022-10-01 NaN NaN

2022-11-01 NaN NaN

2022-12-01 NaN NaN

2023-01-01 NaN NaN

2023-02-01 NaN NaN

future\_Dataset = pd.concat([df, future\_dates\_ds])

future\_Dataset['forecast'] = results.predict(start = 20, end = 120, dynamic = True)

future\_Dataset[['Accidents', 'forecast']].plot(figsize=(12,8))

