usa-car-accidents-severity-prediction

April 24, 2022

1 OVERVIEW & PREPROCESSING

[ ]:
```
import numpy as np
import pandas as pd
import json
import matplotlib
import matplotlib.pyplot as plt
from matplotlib import cm
from datetime import datetime
import glob
import seaborn as sns
import re
import os
import io
from scipy.stats import boxcox
```

[ ]:
```
df = pd.read_csv(r'C:\Users\idiot\DIVA\US_Accidents_Dec21_updated.csv')
print("The shape of data is:",(df.shape))
display(df.head(3))
```

```
The shape of data is: (3015853, 47)
```

	ID	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	End_Lat	End_Lng	Distance(mi)	Description	...	Roundabout	Station	Stop	Traffic_Calming	Traffic_Signal	Turning_Loop	Sunrise_Sunset
0	A-3166784	3	2016-02-08 00:37:08	2016-02-08 06:37:08	40.10891	-83.09286	40.11206	-83.03187	3.230	Between Sawmill Rd/Exit 20 and OH-315/Olentang...	...	False	False	False	False	False	False	Night
1	A-3166785	2	2016-02-08 05:56:20	2016-02-08 11:56:20	39.86542	-84.06280	39.86501	-84.04873	0.747	At OH-4/OH-235/Exit 41 - Accident.	...	False	False	False	False	False	False	Night
2	A-3166786	2	2016-02-08 06:15:39	2016-02-08 12:15:39	39.10266	-84.52468	39.10209	-84.52396	0.055	At I-71/US-50/Exit 1 - Accident.	...	False	False	False	False	False	False	Night

```
1  False           False          False        False          Night
2  False           False          False        False          Night

  Civil_Twilight Nautical_Twilight Astronomical_Twilight
0          Night             Night                 Night
1          Night             Night                 Night
2          Night             Night                   Day

[3 rows x 47 columns]
```

[ ]:
```
# fix datetime type
df['Start_Time'] = pd.to_datetime(df['Start_Time'])
df['End_Time'] = pd.to_datetime(df['End_Time'])
df['Weather_Timestamp'] = pd.to_datetime(df['Weather_Timestamp'])

# calculate duration as the difference between end time and start time in minute
df['Duration'] = df.End_Time - df.Start_Time
df['Duration'] = df['Duration'].apply(lambda x:round(x.total_seconds() / 60) )
print("The overall mean duration is: ", (round(df['Duration'].mean(),3)), 'min')
```

```
The overall mean duration is:  409.926 min
```

[ ]:
```
df = df.drop(['ID','Description','Distance(mi)', 'End_Time', 'Duration',
              'End_Lat', 'End_Lng'], axis=1)
```

[ ]:
```
cat_names = ['Side', 'Country', 'Timezone', 'Amenity', 'Bump', 'Crossing',
             'Give_Way', 'Junction', 'No_Exit', 'Railway', 'Roundabout',␣
 ↪'Station',
             'Stop', 'Traffic_Calming', 'Traffic_Signal', 'Turning_Loop',␣
 ↪'Sunrise_Sunset',
             'Civil_Twilight', 'Nautical_Twilight', 'Astronomical_Twilight']
print("Unique count of categorical features:")
for i in cat_names:
  print(i,df[i].unique().size)
```

```
Unique count of categorical features:
Side 3
Country 1
Timezone 5
Amenity 2
Bump 2
Crossing 2
Give_Way 2
Junction 2
No_Exit 2
Railway 2
Roundabout 2
```

```
Station 2
Stop 2
Traffic_Calming 2
Traffic_Signal 2
Turning_Loop 1
Sunrise_Sunset 3
Civil_Twilight 3
Nautical_Twilight 3
Astronomical_Twilight 3
```

[ ]:
```
df = df.drop(['Country','Turning_Loop'], axis=1)
```

[ ]:
```
print("Wind Direction: ", df['Wind_Direction'].unique())
```

```
Wind Direction:  ['SW' 'Calm' 'WSW' 'WNW' 'West' 'NNW' 'South' 'W' 'NW' 'North'
'SSE' 'SSW'
 'ESE' 'SE' nan 'East' 'Variable' 'NNE' 'NE' 'ENE' 'CALM' 'S' 'VAR' 'N'
 'E']
```

[ ]:
```
df.loc[df['Wind_Direction']=='Calm','Wind_Direction'] = 'CALM'
df.
 ↪loc[(df['Wind_Direction']=='West')|(df['Wind_Direction']=='WSW')|(df['Wind_Direction']=='WN
 ↪= 'W'
df.
 ↪loc[(df['Wind_Direction']=='South')|(df['Wind_Direction']=='SSW')|(df['Wind_Direction']=='S
 ↪= 'S'
df.
 ↪loc[(df['Wind_Direction']=='North')|(df['Wind_Direction']=='NNW')|(df['Wind_Direction']=='N
 ↪= 'N'
df.
 ↪loc[(df['Wind_Direction']=='East')|(df['Wind_Direction']=='ESE')|(df['Wind_Direction']=='EN
 ↪= 'E'
df.loc[df['Wind_Direction']=='Variable','Wind_Direction'] = 'VAR'
print("Wind Direction after simplification: ", df['Wind_Direction'].unique())
```

```
Wind Direction after simplification:  ['SW' 'CALM' 'W' 'N' 'S' 'NW' 'E' 'SE' nan
'VAR' 'NE']
```

[ ]:
```
# show distinctive weather conditions
weather ='!'.join(df['Weather_Condition'].dropna().unique().tolist())
weather = np.unique(np.array(re.split(
    "!|\s/\s|\sand\s|\swith\s|Partly\s|Mostly\s|Blowing\s|Freezing\s",␣
 ↪weather))).tolist()
print("Weather Conditions: ", weather)
```

```
Weather Conditions:  ['', 'Clear', 'Cloudy', 'Drifting Snow', 'Drizzle', 'Dust',
'Dust Whirls', 'Dust Whirls Nearby', 'Dust Whirlwinds', 'Duststorm', 'Fair',
'Fog', 'Funnel Cloud', 'Hail', 'Haze', 'Heavy ', 'Heavy Drizzle', 'Heavy Ice
```

```
Pellets', 'Heavy Rain', 'Heavy Rain Shower', 'Heavy Rain Showers', 'Heavy
Sleet', 'Heavy Snow', 'Heavy T-Storm', 'Heavy Thunderstorms', 'Ice Pellets',
'Light ', 'Light Drizzle', 'Light Fog', 'Light Haze', 'Light Ice Pellets',
'Light Rain', 'Light Rain Shower', 'Light Rain Showers', 'Light Sleet', 'Light
Snow', 'Light Snow Shower', 'Light Snow Showers', 'Light Thunderstorms', 'Low
Drifting Snow', 'Mist', 'N/A Precipitation', 'Overcast', 'Partial Fog', 'Patches
of Fog', 'Rain', 'Rain Shower', 'Rain Showers', 'Sand', 'Scattered Clouds',
'Shallow Fog', 'Showers in the Vicinity', 'Sleet', 'Small Hail', 'Smoke',
'Snow', 'Snow Grains', 'Snow Nearby', 'Squalls', 'T-Storm', 'Thunder', 'Thunder
in the Vicinity', 'Thunderstorm', 'Thunderstorms', 'Tornado', 'Volcanic Ash',
'Widespread Dust', 'Windy', 'Wintry Mix']
```

[ ]:
```
df['Clear'] = np.where(df['Weather_Condition'].str.contains('Clear',␣
↪case=False, na = False), True, False)
df['Cloud'] = np.where(df['Weather_Condition'].str.contains('Cloud|Overcast',␣
↪case=False, na = False), True, False)
df['Rain'] = np.where(df['Weather_Condition'].str.contains('Rain|storm',␣
↪case=False, na = False), True, False)
df['Heavy_Rain'] = np.where(df['Weather_Condition'].str.contains('Heavy␣
↪Rain|Rain Shower|Heavy T-Storm|Heavy Thunderstorms', case=False, na =␣
↪False), True, False)
df['Snow'] = np.where(df['Weather_Condition'].str.contains('Snow|Sleet|Ice',␣
↪case=False, na = False), True, False)
df['Heavy_Snow'] = np.where(df['Weather_Condition'].str.contains('Heavy␣
↪Snow|Heavy Sleet|Heavy Ice Pellets|Snow Showers|Squalls', case=False, na =␣
↪False), True, False)
df['Fog'] = np.where(df['Weather_Condition'].str.contains('Fog', case=False, na␣
↪= False), True, False)
```

[ ]:
```
# Assign NA to created weather features where 'Weather_Condition' is null.
weather = ['Clear','Cloud','Rain','Heavy_Rain','Snow','Heavy_Snow','Fog']
for i in weather:
    df.loc[df['Weather_Condition'].isnull(),i] = df.loc[df['Weather_Condition'].
↪isnull(),'Weather_Condition']
    df[i] = df[i].astype('bool')

df.loc[:,['Weather_Condition'] + weather]

df = df.drop(['Weather_Condition'], axis=1)
```

[ ]:
```
# average difference between weather time and start time
print("Mean difference between 'Start_Time' and 'Weather_Timestamp': ",
(df.Weather_Timestamp - df.Start_Time).mean())
```

```
Mean difference between 'Start_Time' and 'Weather_Timestamp':  0 days
00:01:14.478512055
```

[ ]:
```
df = df.drop(["Weather_Timestamp"], axis=1)

df['Year'] = df['Start_Time'].dt.year

nmonth = df['Start_Time'].dt.month
df['Month'] = nmonth

df['Weekday']= df['Start_Time'].dt.weekday

days_each_month = np.cumsum(np.array([0,31,28,31,30,31,30,31,31,30,31,30,31]))
nday = [days_each_month[arg-1] for arg in nmonth.values]
nday = nday + df["Start_Time"].dt.day.values
df['Day'] = nday

df['Hour'] = df['Start_Time'].dt.hour

df['Minute']=df['Hour']*60.0+df["Start_Time"].dt.minute

df.loc[:4,['Start_Time', 'Year', 'Month', 'Weekday', 'Day', 'Hour', 'Minute']]
```

[ ]:

	Start_Time	Year	Month	Weekday	Day	Hour	Minute
0	2016-02-08 00:37:08	2016	2	0	39	0	37.0
1	2016-02-08 05:56:20	2016	2	0	39	5	356.0
2	2016-02-08 06:15:39	2016	2	0	39	6	375.0
3	2016-02-08 06:15:39	2016	2	0	39	6	375.0
4	2016-02-08 06:51:45	2016	2	0	39	6	411.0

[ ]:
```
missing = pd.DataFrame(df.isnull().sum()).reset_index()
missing.columns = ['Feature', 'Missing_Percent(%)']
missing['Missing_Percent(%)'] = missing['Missing_Percent(%)'].apply(lambda x: x␣
↪/ df.shape[0] * 100)
missing.loc[missing['Missing_Percent(%)']>0,:]
```

[ ]:

	Feature	Missing_Percent(%)
4	Number	61.048101
5	Street	0.000066
7	City	0.004708
10	Zipcode	0.047781
11	Timezone	0.128090
12	Airport_Code	0.340567
13	Temperature(F)	2.470479
14	Wind_Chill(F)	16.278910
15	Humidity(%)	2.608284
16	Pressure(in)	2.116482
17	Visibility(mi)	2.520116
18	Wind_Direction	2.643696
19	Wind_Speed(mph)	5.528419

```
20      Precipitation(in)           19.017804
33         Sunrise_Sunset            0.113732
34         Civil_Twilight            0.113732
35      Nautical_Twilight            0.113732
36  Astronomical_Twilight            0.113732
```

[ ]:
```
df = df.drop(['Number','Wind_Chill(F)'], axis=1)
```

[ ]:
```
df['Precipitation_NA'] = 0
df.loc[df['Precipitation(in)'].isnull(),'Precipitation_NA'] = 1
df['Precipitation(in)'] = df['Precipitation(in)'].
 ↪fillna(df['Precipitation(in)'].median())
df.loc[:5,['Precipitation(in)','Precipitation_NA']]
```

[ ]:

	Precipitation(in)	Precipitation_NA
0	0.00	0
1	0.02	0
2	0.02	0
3	0.02	0
4	0.00	1
5	0.01	0

[ ]:
```
df = df.dropna(subset=['City','Zipcode','Airport_Code',
                       ␣
 ↪'Sunrise_Sunset','Civil_Twilight','Nautical_Twilight','Astronomical_Twilight'])
```

[ ]:
```
# group data by 'Airport_Code' and 'Start_Month' then fill NAs with median value
Weather_data=['Temperature(F)','Humidity(%)','Pressure(in)','Visibility(mi)','Wind_Speed(mph)'
print("The number of remaining missing values: ")
for i in Weather_data:
  df[i] = df.groupby(['Airport_Code','Month'])[i].apply(lambda x: x.fillna(x.
 ↪median()))
  print( i + " : " + df[i].isnull().sum().astype(str))
```

```
The number of remaining missing values:
Temperature(F) : 6953
Humidity(%) : 6956
Pressure(in) : 6941
Visibility(mi) : 17986
Wind_Speed(mph) : 17838
```

[ ]:
```
df = df.dropna(subset=Weather_data)
```

[ ]:
```
# group data by 'Airport_Code' and 'Start_Month' then fill NAs with majority␣
 ↪value
from collections import Counter
weather_cat = ['Wind_Direction'] + weather
```

```
print("Count of missing values that will be dropped: ")
for i in weather_cat:
  df[i] = df.groupby(['Airport_Code','Month'])[i].apply(lambda x: x.
 ↪fillna(Counter(x).most_common()[0][0]) if all(x.isnull())==False else x)
  print(i + " : " + df[i].isnull().sum().astype(str))

# drop na
df = df.dropna(subset=weather_cat)
```

```
Count of missing values that will be dropped:
Wind_Direction : 21089
Clear : 0
Cloud : 0
Rain : 0
Heavy_Rain : 0
Snow : 0
Heavy_Snow : 0
Fog : 0
```

[ ]:
```
df['Severity4'] = 0
df.loc[df['Severity'] == 4, 'Severity4'] = 1
df = df.drop(['Severity'], axis = 1)
df.Severity4.value_counts()
```

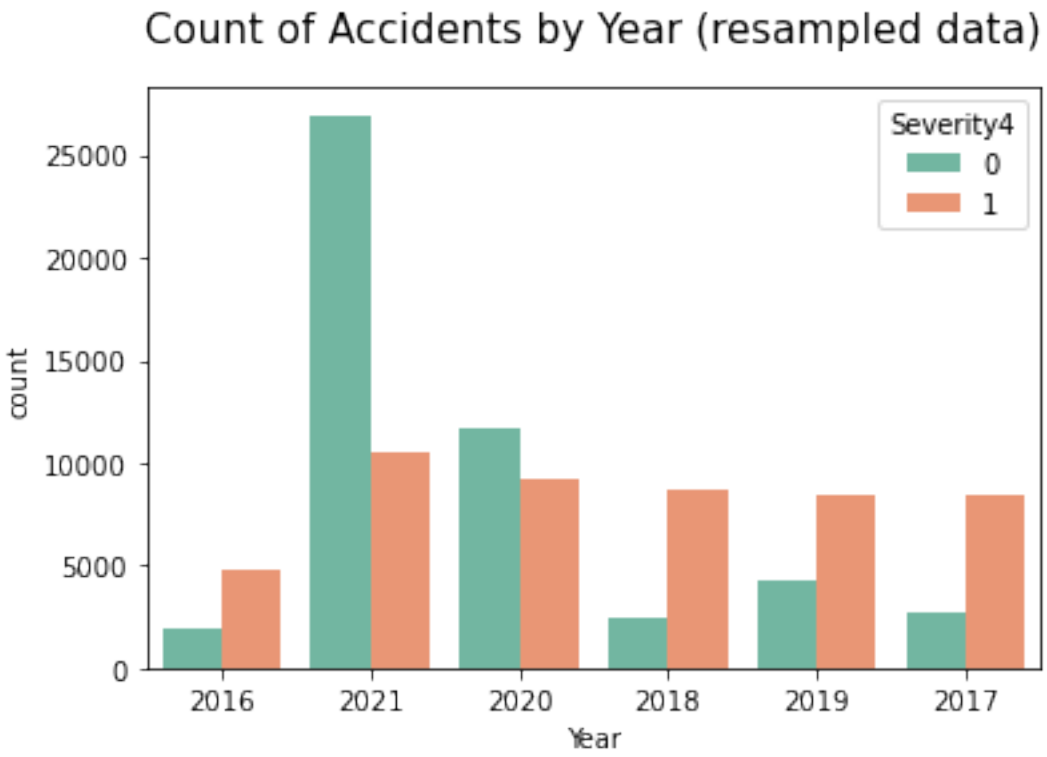
[ ]:
```
0    2820720
1     142225
Name: Severity4, dtype: int64
```

[ ]:
```
def resample(dat, col, n):
    return pd.concat([dat[dat[col]==1].sample(n, replace = True),
                      dat[dat[col]==0].sample(n)], axis=0)
```

[ ]:
```
df_bl = resample(df, 'Severity4', 50000)
print('resampled data:', df_bl.Severity4.value_counts())
```

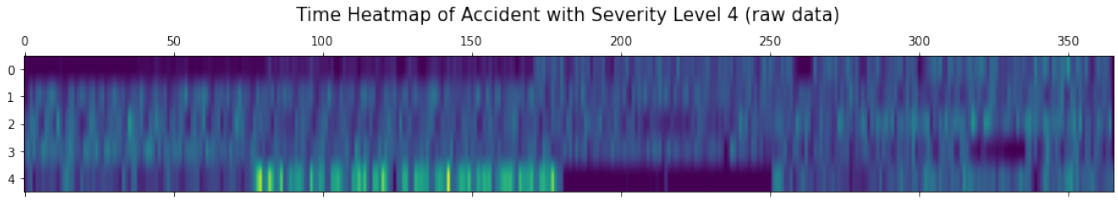
```
resampled data: 1    50000
0    50000
Name: Severity4, dtype: int64
```

[ ]:
```
df_bl.Year = df_bl.Year.astype(str)
sns.countplot(x='Year', hue='Severity4', data=df_bl ,palette="Set2")
plt.title('Count of Accidents by Year (resampled data)', size=15, y=1.05)
plt.show()
```



[ ]:
```
# create a dataframe used to plot heatmap
df_date = df.loc[:,['Start_Time','Severity4']]          # create a new dateframe␣
 ↪only containing time and severity
df_date['date'] = df_date['Start_Time'].dt.normalize() # keep only the date␣
 ↪part of start time
df_date = df_date.drop(['Start_Time'], axis = 1)
df_date = df_date.groupby('date').sum()                # sum the number of␣
 ↪accidents with severity level 4 by date
df_date = df_date.reset_index().drop_duplicates()

# join the dataframe with full range of date from 2016 to 2020
full_date = pd.DataFrame(pd.date_range(start="2016-01-02",end="2020-12-31"))
df_date = full_date.merge(df_date, how = 'left',left_on = 0, right_on = 'date')
df_date['date'] = df_date.iloc[:,0]
df_date = df_date.fillna(0)
df_date = df_date.iloc[:,1:].set_index('date')

# group by date
groups = df_date['Severity4'].groupby(pd.Grouper(freq='A'))
years = pd.DataFrame()
for name, group in groups:
    if name.year != 2020:
```

```
        years[name.year] = np.append(group.values,0)
    else:
        years[name.year] = group.values


# plot
years = years.T
plt.matshow(years, interpolation=None, aspect='auto')
plt.title('Time Heatmap of Accident with Severity Level 4 (raw data)', y=1.2,␣
 ↪fontsize=15)
plt.show()
```

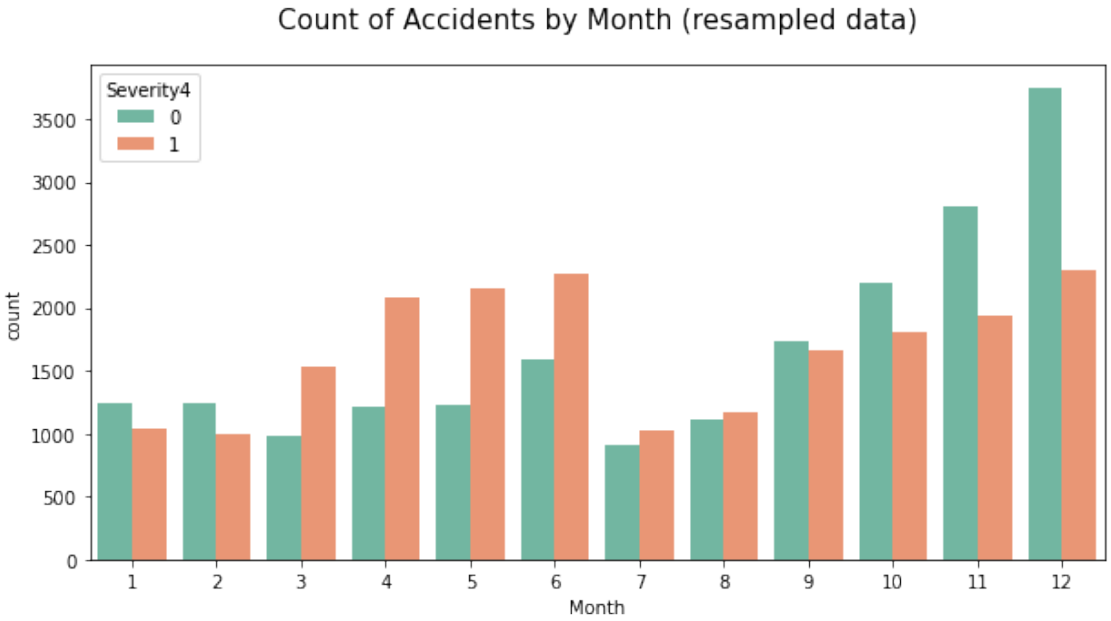


[ ]:
```
df = df.loc[df['Start_Time'] > "2019-03-10",:]
df = df.drop(['Year', 'Start_Time'], axis=1)
df['Severity4'].value_counts()
```

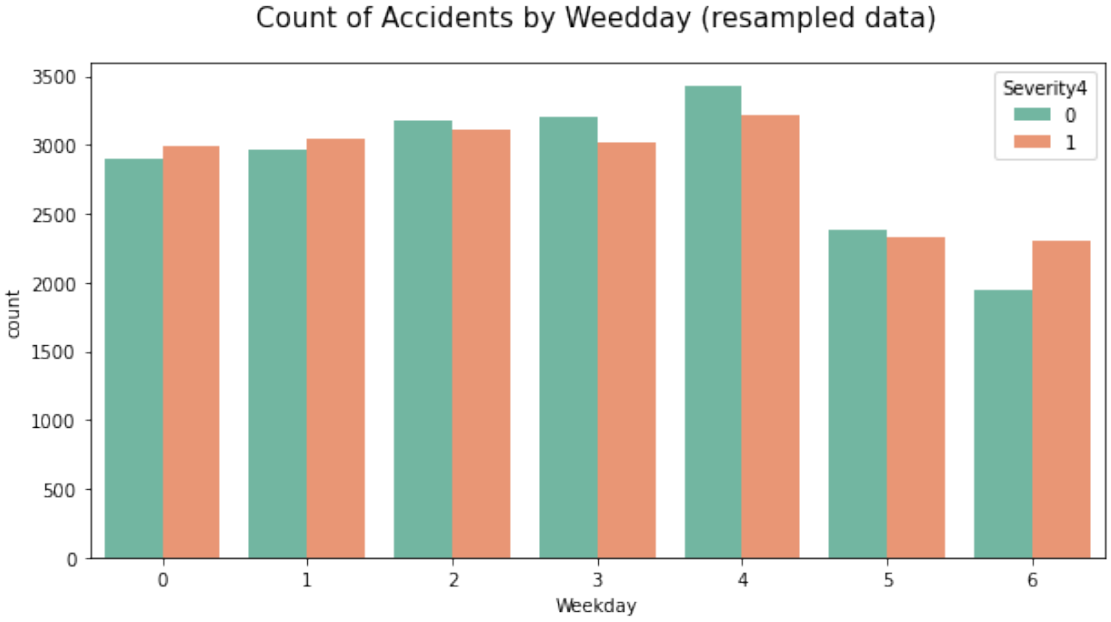
[ ]:
```
0    2388158
1      74307
Name: Severity4, dtype: int64
```

[ ]:
```
df_bl = resample(df, 'Severity4', 20000)
```

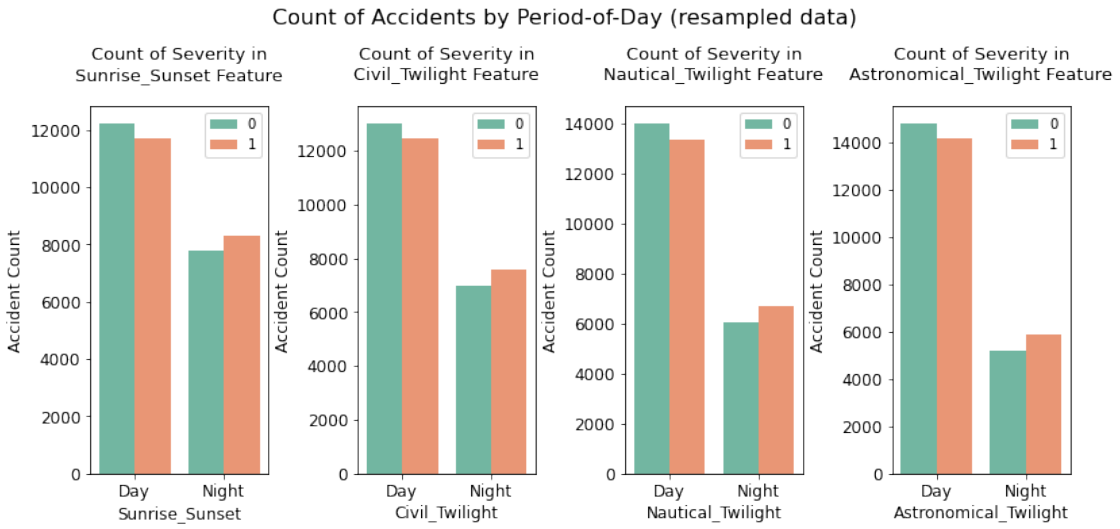
[ ]:
```
plt.figure(figsize=(10,5))
sns.countplot(x='Month', hue='Severity4', data=df_bl ,palette="Set2")
plt.title('Count of Accidents by Month (resampled data)', size=15, y=1.05)
plt.show()
```



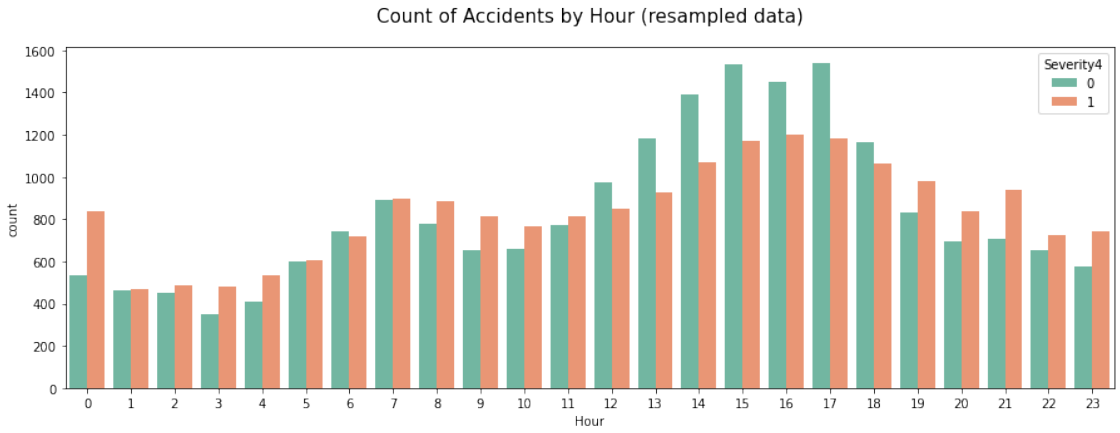
[ ]:
```
plt.figure(figsize=(10,5))
sns.countplot(x='Weekday', hue='Severity4', data=df_bl ,palette="Set2")
plt.title('Count of Accidents by Weedday (resampled data)', size=15, y=1.05)
plt.show()
```



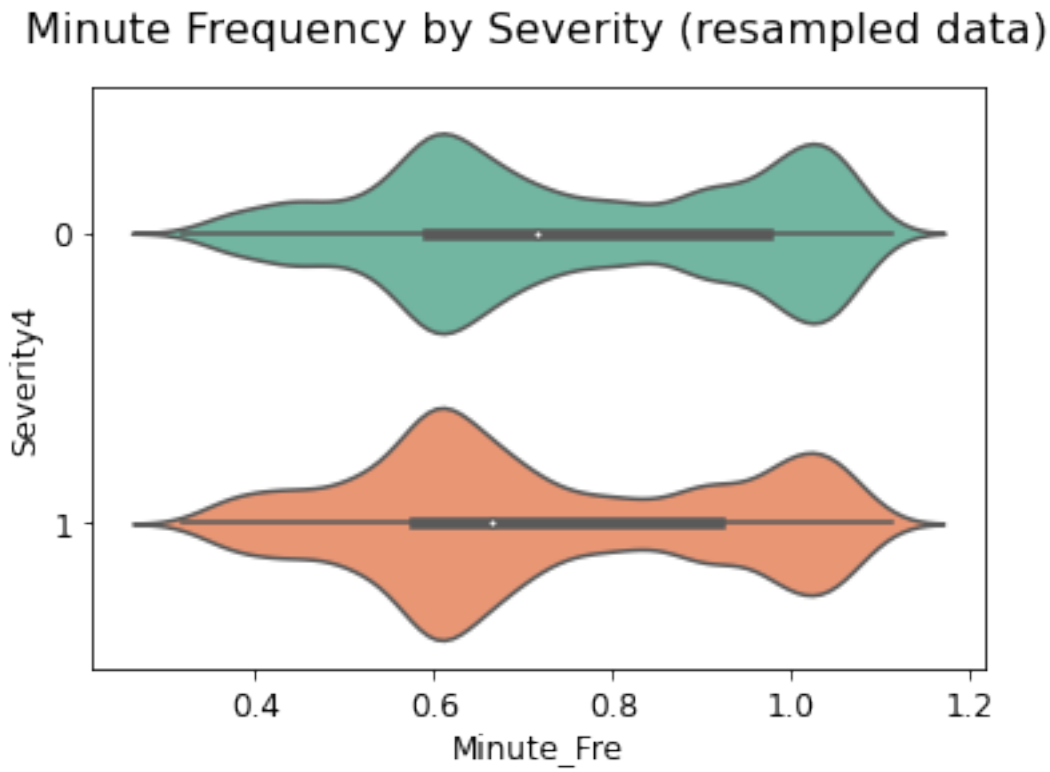
[ ]:
```
period_features =␣
 ↪['Sunrise_Sunset','Civil_Twilight','Nautical_Twilight','Astronomical_Twilight']
fig, axs = plt.subplots(ncols=1, nrows=4, figsize=(13, 5))

plt.subplots_adjust(wspace = 0.5)
for i, feature in enumerate(period_features, 1):
    plt.subplot(1, 4, i)
    sns.countplot(x=feature, hue='Severity4', data=df_bl ,palette="Set2")

    plt.xlabel('{}'.format(feature), size=12, labelpad=3)
    plt.ylabel('Accident Count', size=12, labelpad=3)
    plt.tick_params(axis='x', labelsize=12)
    plt.tick_params(axis='y', labelsize=12)

    plt.legend(['0', '1'], loc='upper right', prop={'size': 10})
    plt.title('Count of Severity in\n{} Feature'.format(feature), size=13, y=1.
 ↪05)
fig.suptitle('Count of Accidents by Period-of-Day (resampled data)',y=1.08,␣
 ↪fontsize=16)
plt.show()
```



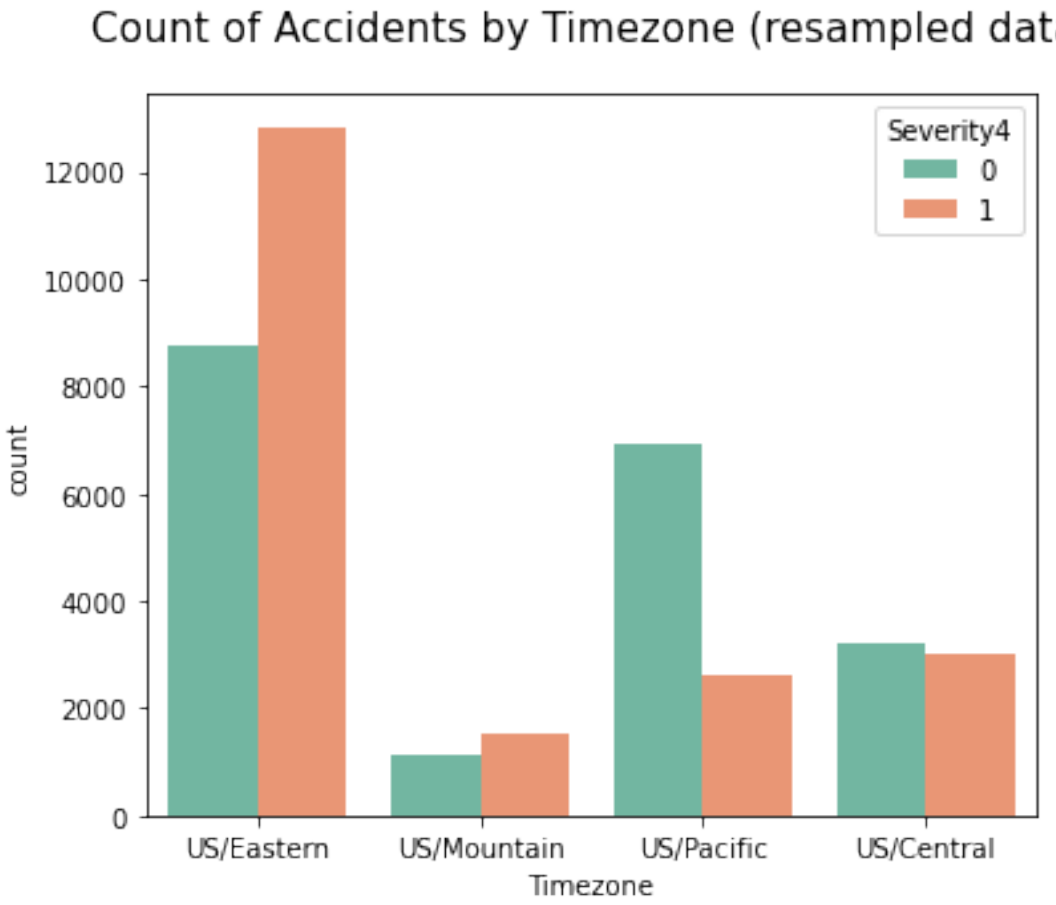
[ ]:
```
plt.figure(figsize=(15,5))
sns.countplot(x='Hour', hue='Severity4', data=df_bl ,palette="Set2")
plt.title('Count of Accidents by Hour (resampled data)', size=15, y=1.05)
plt.show()
```



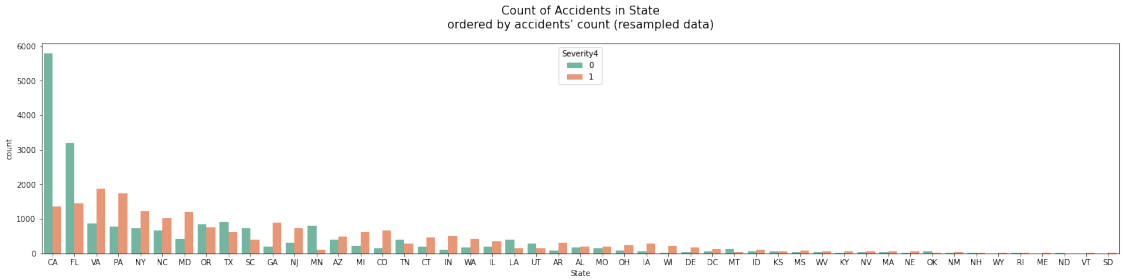
[ ]:
```
# frequence encoding and log-transform
df['Minute_Freq'] = df.groupby(['Minute'])['Minute'].transform('count')
df['Minute_Freq'] = df['Minute_Freq']/df.shape[0]*24*60
df['Minute_Freq'] = df['Minute_Freq'].apply(lambda x: np.log(x+1))

# resampling
df_bl = resample(df, 'Severity4', 20000)

# plot
df_bl['Severity4'] = df_bl['Severity4'].astype('category')
sns.violinplot(x='Minute_Freq', y="Severity4", data=df_bl, palette="Set2")
plt.xlabel('Minute_Fre', size=12, labelpad=3)
plt.ylabel('Severity4', size=12, labelpad=3)
plt.tick_params(axis='x', labelsize=12)
plt.tick_params(axis='y', labelsize=12)
plt.title('Minute Frequency by Severity (resampled data)', size=16, y=1.05)
plt.show()
```



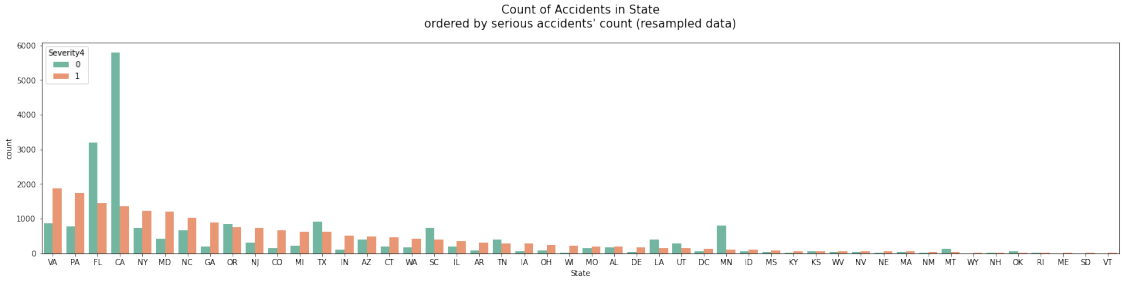
[ ]:
```
plt.figure(figsize=(6,5))
chart = sns.countplot(x='Timezone', hue='Severity4', data=df_bl ,palette="Set2")
plt.title("Count of Accidents by Timezone (resampled data)", size=15, y=1.05)
plt.show()
```



[ ]:
```
plt.figure(figsize=(25,5))
chart = sns.countplot(x='State', hue='Severity4',
                      data=df_bl ,palette="Set2", order=df_bl['State'].
 ↪value_counts().index)
plt.title("Count of Accidents in State\nordered by accidents' count (resampled␣
 ↪data)", size=15, y=1.05)
plt.show()
```



[ ]:
```
plt.figure(figsize=(25,5))
chart = sns.countplot(x='State', hue='Severity4', data=df_bl ,palette="Set2",␣
↪order=df_bl[df_bl['Severity4']==1]['State'].value_counts().index)
plt.title("Count of Accidents in State\nordered by serious accidents' count␣
↪(resampled data)", size=15, y=1.05)
plt.show()
```



[ ]:
```
!pip install -q censusdata
import censusdata

# download data
county = censusdata.download('acs5', 2018, censusdata.censusgeo([('county',␣
↪'*')]),
                                    ['DP05_0001E', ␣
↪'DP03_0019PE','DP03_0021PE','DP03_0022PE','DP03_0062E'],
                                    tabletype='profile')
# rename columns
county.columns =␣
↪['Population_County','Drive_County','Transit_County','Walk_County','MedianHouseholdIncome_C
county = county.reset_index()
# extract county name and state name
county['County_y'] = county['index'].apply(lambda x : x.name.split('␣
↪County')[0].split(',')[0]).str.lower()
county['State_y'] = county['index'].apply(lambda x : x.name.split(':')[0].
↪split(', ')[1])
```

[ ]:
```
us_state_abbrev = {
    'Alabama': 'AL',
    'Alaska': 'AK',
    'American Samoa': 'AS',
    'Arizona': 'AZ',
    'Arkansas': 'AR',
    'California': 'CA',
    'Colorado': 'CO',
    'Connecticut': 'CT',
    'Delaware': 'DE',
```

```
    'District of Columbia': 'DC',
    'Florida': 'FL',
    'Georgia': 'GA',
    'Guam': 'GU',
    'Hawaii': 'HI',
    'Idaho': 'ID',
    'Illinois': 'IL',
    'Indiana': 'IN',
    'Iowa': 'IA',
    'Kansas': 'KS',
    'Kentucky': 'KY',
    'Louisiana': 'LA',
    'Maine': 'ME',
    'Maryland': 'MD',
    'Massachusetts': 'MA',
    'Michigan': 'MI',
    'Minnesota': 'MN',
    'Mississippi': 'MS',
    'Missouri': 'MO',
    'Montana': 'MT',
    'Nebraska': 'NE',
    'Nevada': 'NV',
    'New Hampshire': 'NH',
    'New Jersey': 'NJ',
    'New Mexico': 'NM',
    'New York': 'NY',
    'North Carolina': 'NC',
    'North Dakota': 'ND',
    'Northern Mariana Islands':'MP',
    'Ohio': 'OH',
    'Oklahoma': 'OK',
    'Oregon': 'OR',
    'Pennsylvania': 'PA',
    'Puerto Rico': 'PR',
    'Rhode Island': 'RI',
    'South Carolina': 'SC',
    'South Dakota': 'SD',
    'Tennessee': 'TN',
    'Texas': 'TX',
    'Utah': 'UT',
    'Vermont': 'VT',
    'Virgin Islands': 'VI',
    'Virginia': 'VA',
    'Washington': 'WA',
    'West Virginia': 'WV',
    'Wisconsin': 'WI',
    'Wyoming': 'WY'
```

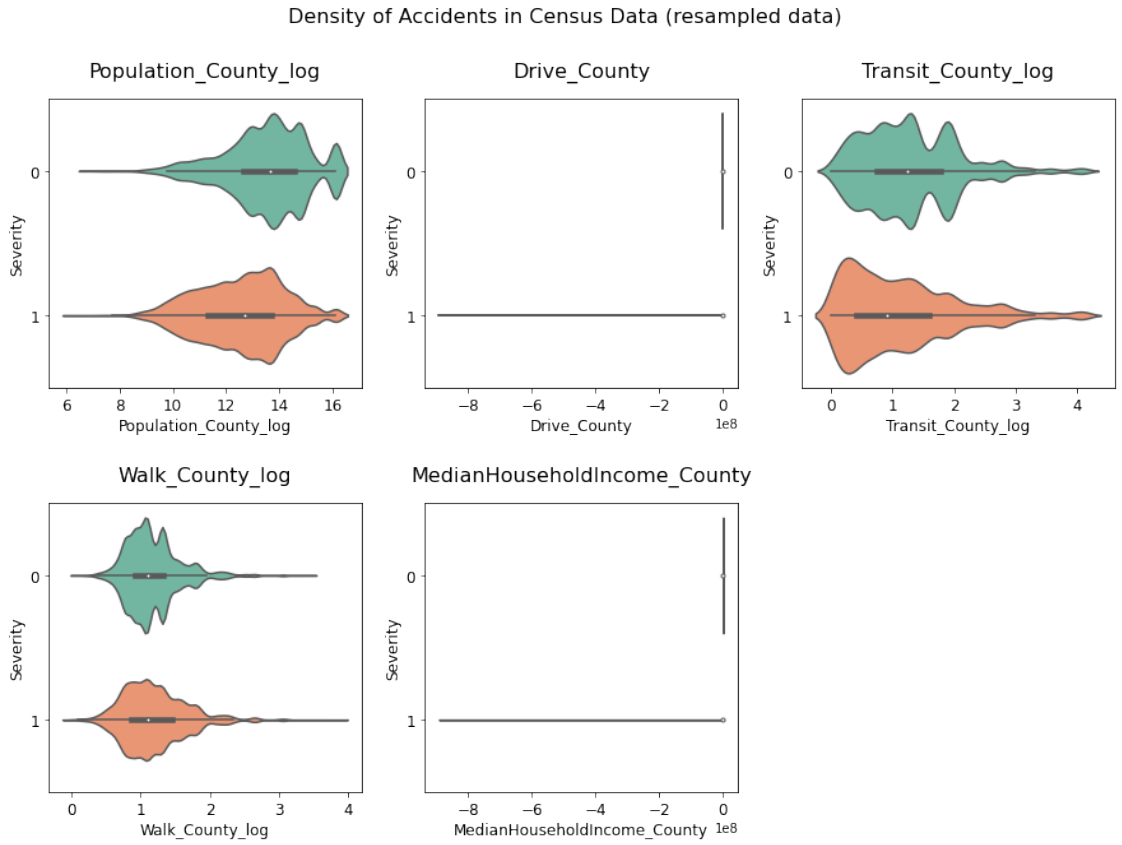
```
}
county['State_y'] = county['State_y'].replace(us_state_abbrev)
```

[ ]:
```
# convert all county name to lowercase
df['County'] = df['County'].str.lower()

# left join df with census data
df = df.merge(county, left_on = ['County','State'],␣
↪right_on=['County_y','State_y'],how = 'left').drop(['County_y','State_y'],␣
↪axis = 1)
join_var = county.columns.to_list()[:-2]

# check how many miss match we got
print('Count of missing values before: ', df[join_var].isnull().sum())

# add "city" and match again
df_city = df[df['Walk_County'].isnull()].drop(join_var, axis=1)
df_city['County_city'] = df_city['County'].apply(lambda x : x + ' city')
df_city = df_city.merge(county,left_on= ['County_city','State'],right_on =␣
↪['County_y','State_y'], how = 'left').
↪drop(['County_city','County_y','State_y'], axis=1)
df = pd.concat((df[df['Walk_County'].isnull()==False], df_city), axis=0)

# add "parish" and match again
df_parish = df[df['Walk_County'].isnull()].drop(join_var, axis=1)
df_parish['County_parish'] = df_parish['County'].apply(lambda x : x + ' parish')
df_parish = df_parish.merge(county,left_on= ['County_parish','State'],right_on␣
↪= ['County_y','State_y'], how = 'left').
↪drop(['County_parish','County_y','State_y'], axis=1)
df = pd.concat((df[df['Walk_County'].isnull()==False], df_parish), axis=0)
print('Count of missing values after: ', df[join_var].isnull().sum())
```

```
Count of missing values before:  index                          90126
Population_County               90126
Drive_County                    90126
Transit_County                  90126
Walk_County                     90126
MedianHouseholdIncome_County    90126
dtype: int64
Count of missing values after:  index                          26108
Population_County               26108
Drive_County                    26108
Transit_County                  26108
Walk_County                     26108
MedianHouseholdIncome_County    26108
dtype: int64
```

[ ]:
```
# drop na
df = df.drop('index', axis = 1).dropna()

# log-transform
for i in ['Population_County','Transit_County','Walk_County']:
    df[i + '_log'] = df[i].apply(lambda x: np.log(x+1))
df = df.drop(['Population_County','Transit_County','Walk_County'], axis = 1)
```

[ ]:
```
# resample again
df_bl = resample(df, 'Severity4', 20000)

# plot
df_bl['Severity4'] = df_bl['Severity4'].astype('category')
census_features =␣
 ↪['Population_County_log','Drive_County','Transit_County_log','Walk_County_log','MedianHousel
fig, axs = plt.subplots(ncols=2, nrows=3, figsize=(15, 10))
plt.subplots_adjust(hspace=0.4,wspace = 0.2)
for i, feature in enumerate(census_features, 1):
    plt.subplot(2, 3, i)
    sns.violinplot(x=feature, y="Severity4", data=df_bl, palette="Set2")

    plt.xlabel('{}'.format(feature), size=12, labelpad=3)
    plt.ylabel('Severity', size=12, labelpad=3)
    plt.tick_params(axis='x', labelsize=12)
    plt.tick_params(axis='y', labelsize=12)

    plt.title('{}'.format(feature), size=16, y=1.05)
fig.suptitle('Density of Accidents in Census Data (resampled data)',␣
 ↪fontsize=16)
plt.show()
```



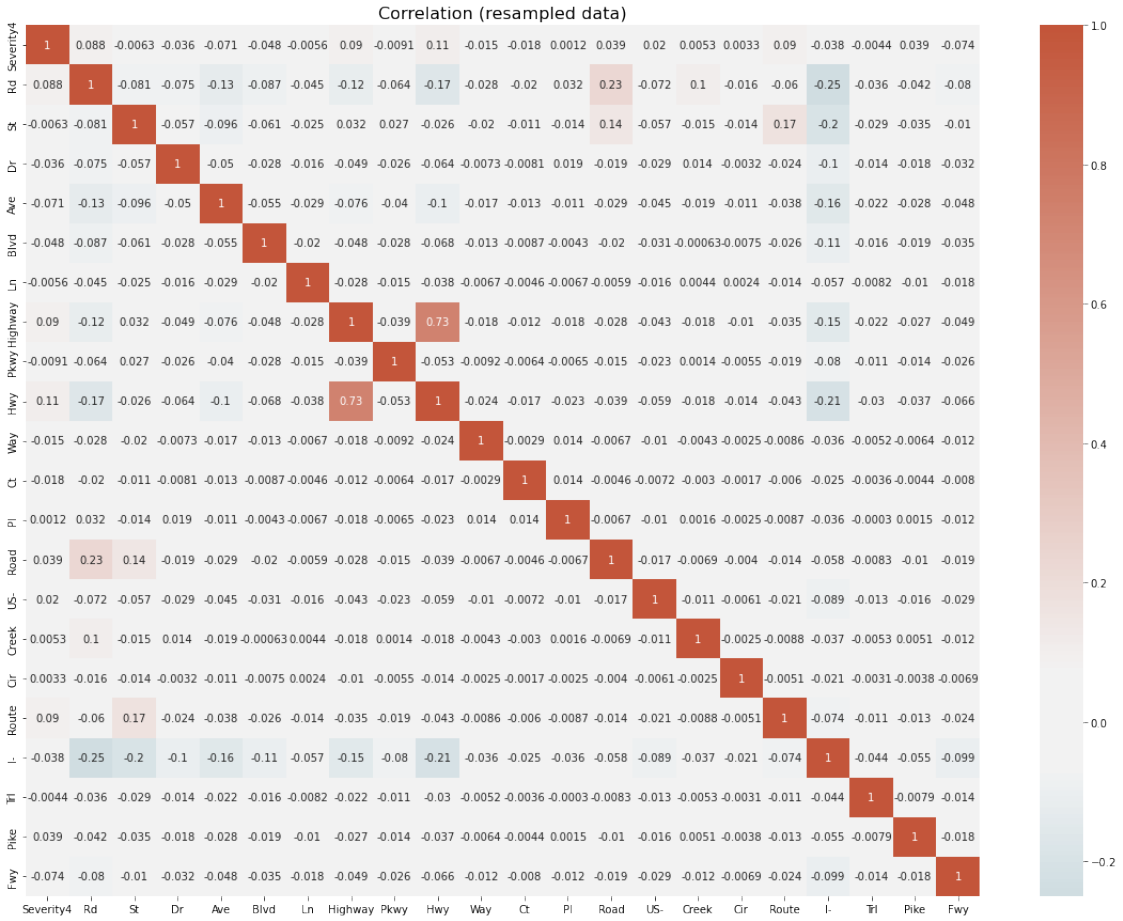
[ ]:
```
# create a list of top 40 most common words in street name
st_type =' '.join(df['Street'].unique().tolist()) # flat the array of street␣
 ↪name
st_type = re.split(" |-", st_type) # split the long string by space and hyphen
st_type = [x[0] for x in Counter(st_type).most_common(40)] # select the 40 most␣
 ↪common words
print('the 40 most common words')
print(*st_type, sep = ", ")
```

```
the 40 most common words
, Rd, Dr, St, Ave, N, S, W, Ln, E, Blvd, Highway, Way, Ct, SW, Hwy, NW, NE,
State, Pl, Pkwy, Road, SE, Cir, Creek, Old, US, Route, Lake, Hill, County, Park,
Trl, Valley, Ter, Ridge, Avenue, River, Mill, Canyon
```

[ ]:
```
# Remove some irrelevant words and add spaces and hyphen back
st_type= [' Rd', ' St', ' Dr', ' Ave', ' Blvd', ' Ln', ' Highway', ' Pkwy', '␣
 ↪Hwy',
          ' Way', ' Ct', 'Pl', ' Road', 'US-', 'Creek', ' Cir',  'Route',
          'I-', 'Trl', 'Pike', ' Fwy']
print(*st_type, sep = ", ")
```

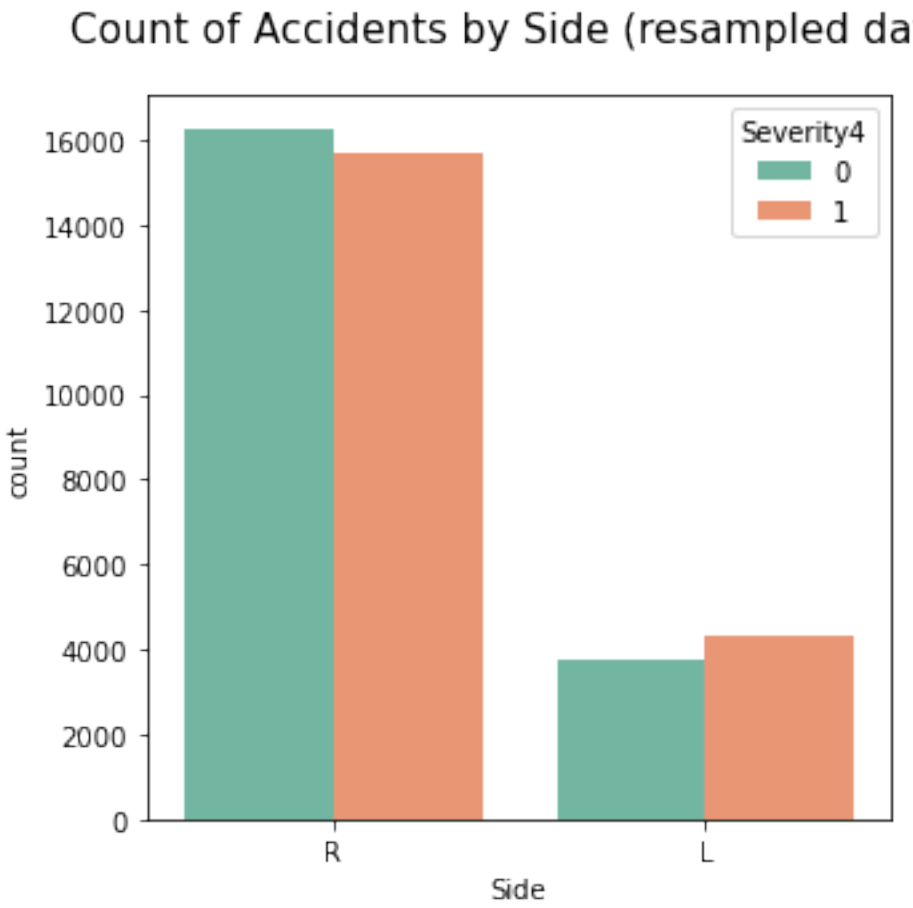
Rd, St, Dr, Ave, Blvd, Ln, Highway, Pkwy, Hwy, Way, Ct, Pl, Road, US-, Creek, Cir, Route, I-, Trl, Pike, Fwy

[ ]:
```
# for each word create a boolean column
for i in st_type:
  df[i.strip()] = np.where(df['Street'].str.contains(i, case=True, na = False),␣
 ↪True, False)
df.loc[df['Road']==1,'Rd'] = True
df.loc[df['Highway']==1,'Hwy'] = True

# resample again
df_bl = resample(df, 'Severity4', 20000)

# plot correlation
df_bl['Severity4'] = df_bl['Severity4'].astype(int)
street_corr  = df_bl.loc[:,['Severity4']+[x.strip() for x in st_type]].corr()
plt.figure(figsize=(20,15))
cmap = sns.diverging_palette(220, 20, sep=20, as_cmap=True)
sns.heatmap(street_corr, annot=True, cmap=cmap, center=0).
 ↪set_title("Correlation (resampled data)", fontsize=16)
plt.show()
```



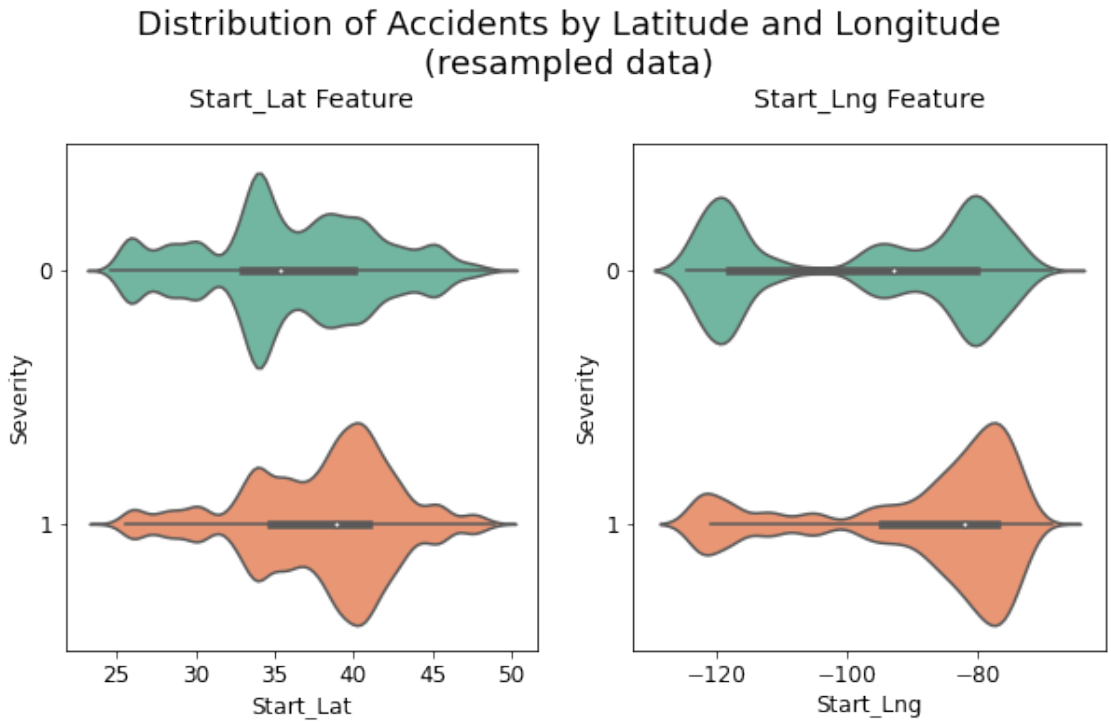
[ ]:
```
drop_list = street_corr.index[street_corr['Severity4'].abs()<0.1].to_list()
df = df.drop(drop_list, axis=1)

# resample again
df_bl = resample(df, 'Severity4', 20000)
```

[ ]:
```
plt.figure(figsize=(5,5))
chart = sns.countplot(x='Side', hue='Severity4', data=df_bl ,palette="Set2")
plt.title("Count of Accidents by Side (resampled data)", size=15, y=1.05)
plt.show()
```



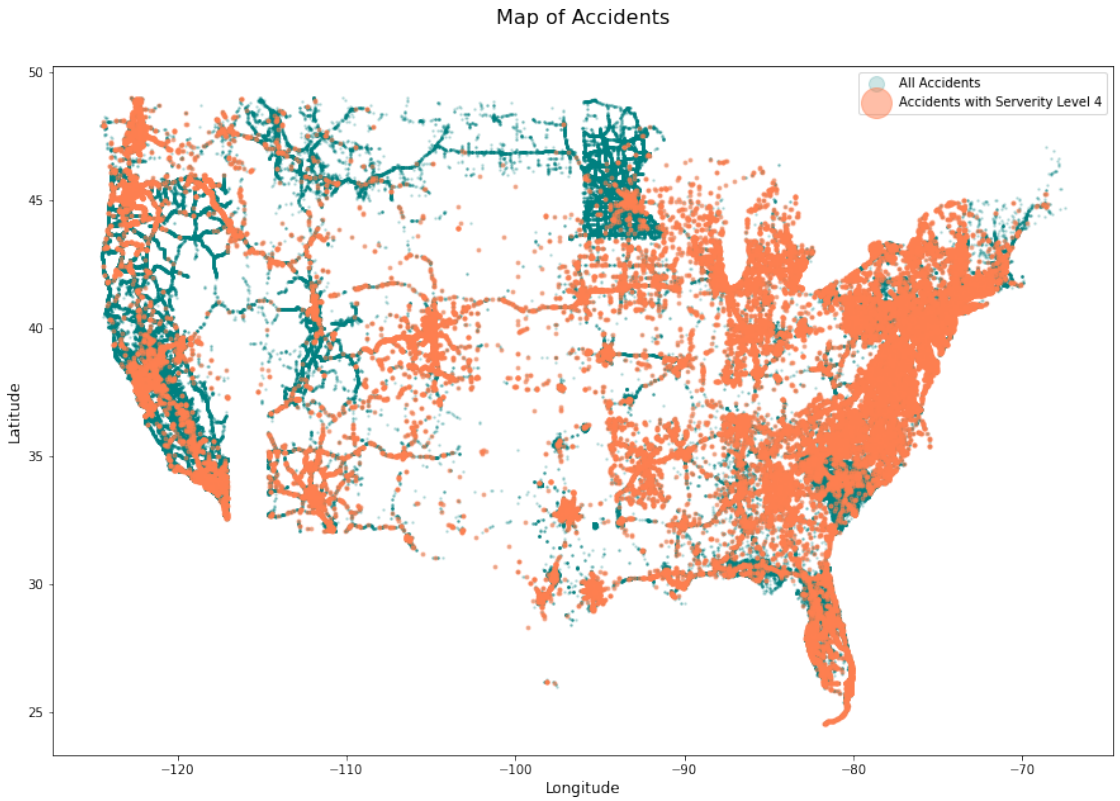
[ ]:
```
df_bl['Severity4'] = df_bl['Severity4'].astype('category')
num_features = ['Start_Lat', 'Start_Lng']
fig, axs = plt.subplots(ncols=1, nrows=2, figsize=(10, 5))
plt.subplots_adjust(hspace=0.4,wspace = 0.2)
for i, feature in enumerate(num_features, 1):
```

```
    plt.subplot(1, 2, i)
    sns.violinplot(x=feature, y="Severity4", data=df_bl, palette="Set2")

    plt.xlabel('{}'.format(feature), size=12, labelpad=3)
    plt.ylabel('Severity', size=12, labelpad=3)
    plt.tick_params(axis='x', labelsize=12)
    plt.tick_params(axis='y', labelsize=12)

    plt.title('{} Feature'.format(feature), size=14, y=1.05)
fig.suptitle('Distribution of Accidents by Latitude and Longitude\n(resampled␣
 ↪data)', fontsize=18,y=1.08)
plt.show()
```



[ ]:
```
df_4 = df[df['Severity4']==1]

plt.figure(figsize=(15,10))

plt.plot( 'Start_Lng', 'Start_Lat', data=df, linestyle='', marker='o',␣
 ↪markersize=1.5, color="teal", alpha=0.2, label='All Accidents')
plt.plot( 'Start_Lng', 'Start_Lat', data=df_4, linestyle='', marker='o',␣
 ↪markersize=3, color="coral", alpha=0.5, label='Accidents with Serverity␣
 ↪Level 4')
plt.legend(markerscale=8)
```

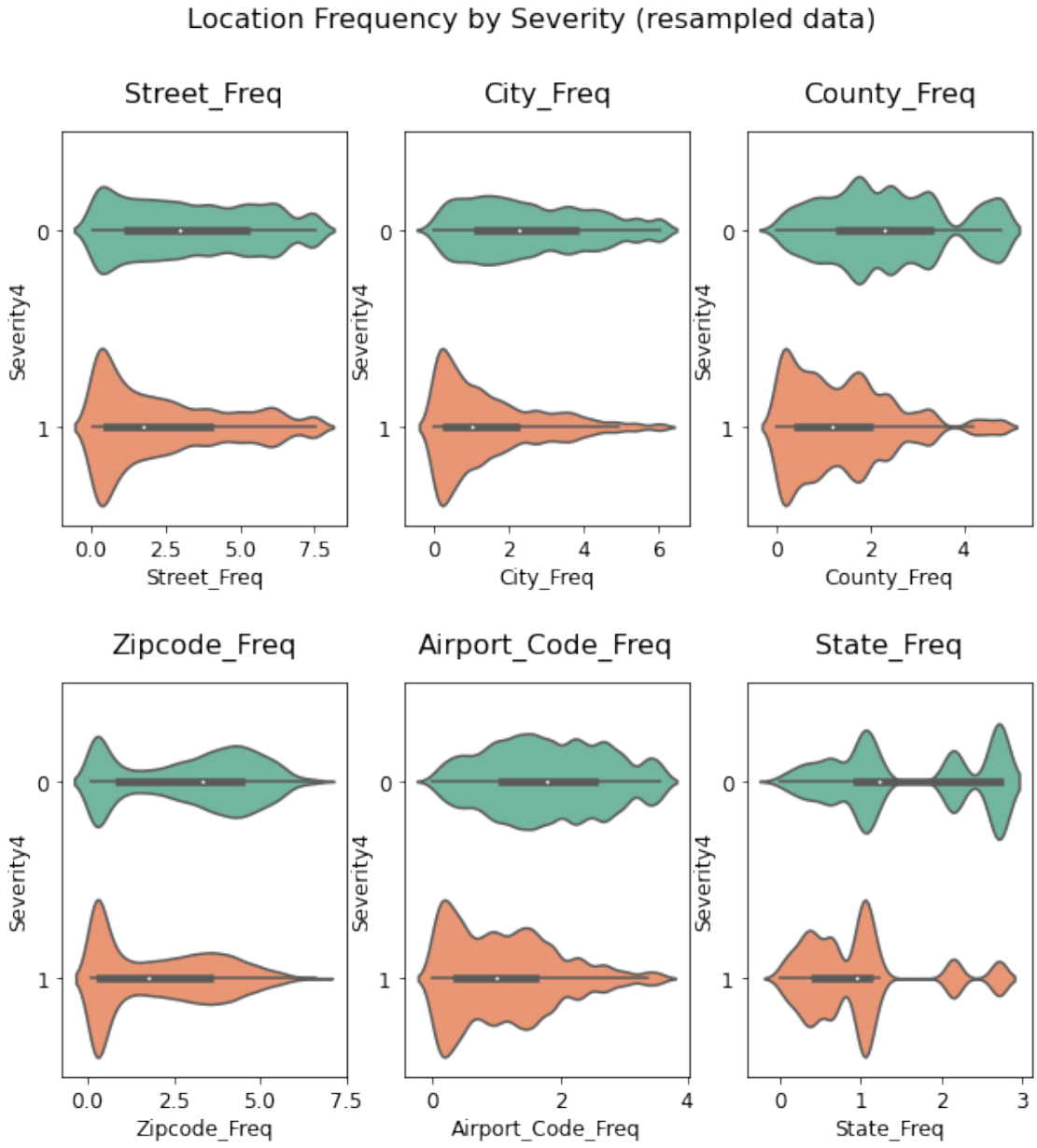
```
plt.xlabel('Longitude', size=12, labelpad=3)
plt.ylabel('Latitude', size=12, labelpad=3)
plt.title('Map of Accidents', size=16, y=1.05)
plt.show()
```



[ ]:
```
fre_list = ['Street', 'City', 'County', 'Zipcode', 'Airport_Code','State']
for i in fre_list:
    newname = i + '_Freq'
    df[newname] = df.groupby([i])[i].transform('count')
    df[newname] = df[newname]/df.shape[0]*df[i].unique().size
    df[newname] = df[newname].apply(lambda x: np.log(x+1))
```

[ ]:
```
# resample again
df_bl = resample(df, 'Severity4', 20000)

df_bl['Severity4'] = df_bl['Severity4'].astype('category')
fig, axs = plt.subplots(ncols=2, nrows=3, figsize=(10, 10))
plt.subplots_adjust(hspace=0.4,wspace = 0.2)
fig.suptitle('Location Frequency by Severity (resampled data)', fontsize=16)
for i, feature in enumerate(fre_list, 1):
    feature = feature + '_Freq'
    plt.subplot(2, 3, i)
```

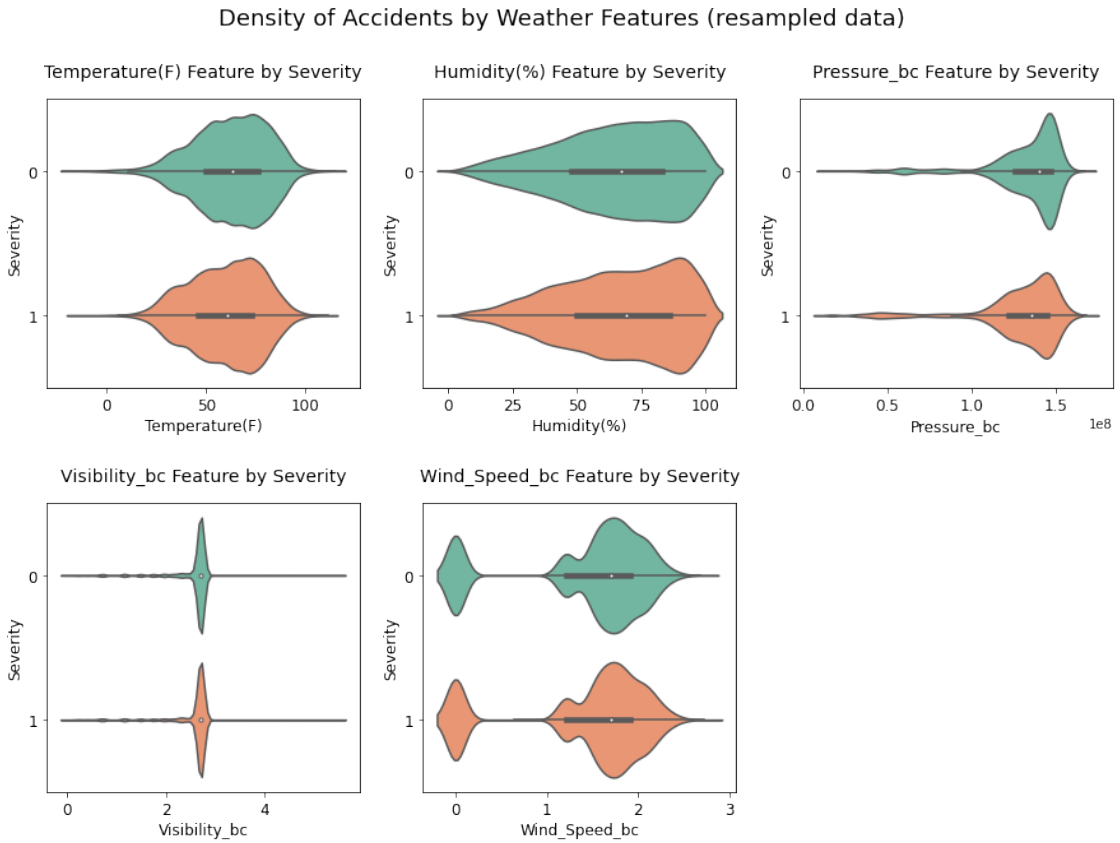
```
    sns.violinplot(x=feature, y="Severity4", data=df_bl, palette="Set2")

    plt.xlabel('{}'.format(feature), size=12, labelpad=3)
    plt.ylabel('Severity4', size=12, labelpad=3)
    plt.tick_params(axis='x', labelsize=12)
    plt.tick_params(axis='y', labelsize=12)

    plt.title('{}'.format(feature), size=16, y=1.05)
plt.show()
```



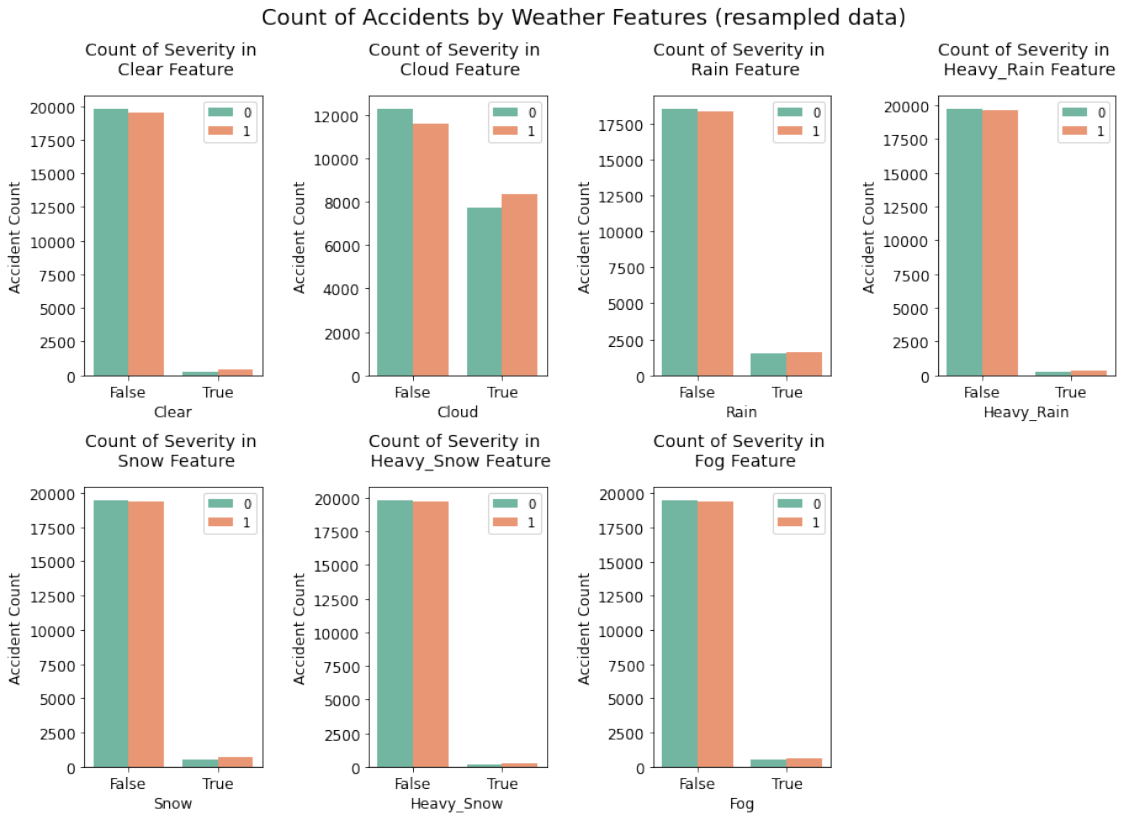
[ ]:
```
df = df.drop(fre_list, axis  = 1)
```

[ ]:
```
df['Pressure_bc']= boxcox(df['Pressure(in)'].apply(lambda x: x+1),lmbda=6)
df['Visibility_bc']= boxcox(df['Visibility(mi)'].apply(lambda x: x+1),lmbda = 0.
↪1)
df['Wind_Speed_bc']= boxcox(df['Wind_Speed(mph)'].apply(lambda x: x+1),lmbda=-0.
↪2)
df = df.drop(['Pressure(in)','Visibility(mi)','Wind_Speed(mph)'], axis=1)
```

[ ]:
```
# resample again
df_bl = resample(df, 'Severity4', 20000)

df_bl['Severity4'] = df_bl['Severity4'].astype('category')
num_features = ['Temperature(F)', 'Humidity(%)', 'Pressure_bc',␣
↪'Visibility_bc', 'Wind_Speed_bc']
fig, axs = plt.subplots(ncols=2, nrows=3, figsize=(15, 10))
plt.subplots_adjust(hspace=0.4,wspace = 0.2)
for i, feature in enumerate(num_features, 1):
    plt.subplot(2, 3, i)
    sns.violinplot(x=feature, y="Severity4", data=df_bl, palette="Set2")

    plt.xlabel('{}'.format(feature), size=12, labelpad=3)
    plt.ylabel('Severity', size=12, labelpad=3)
    plt.tick_params(axis='x', labelsize=12)
    plt.tick_params(axis='y', labelsize=12)

    plt.title('{} Feature by Severity'.format(feature), size=14, y=1.05)
fig.suptitle('Density of Accidents by Weather Features (resampled data)',␣
↪fontsize=18)
plt.show()
```

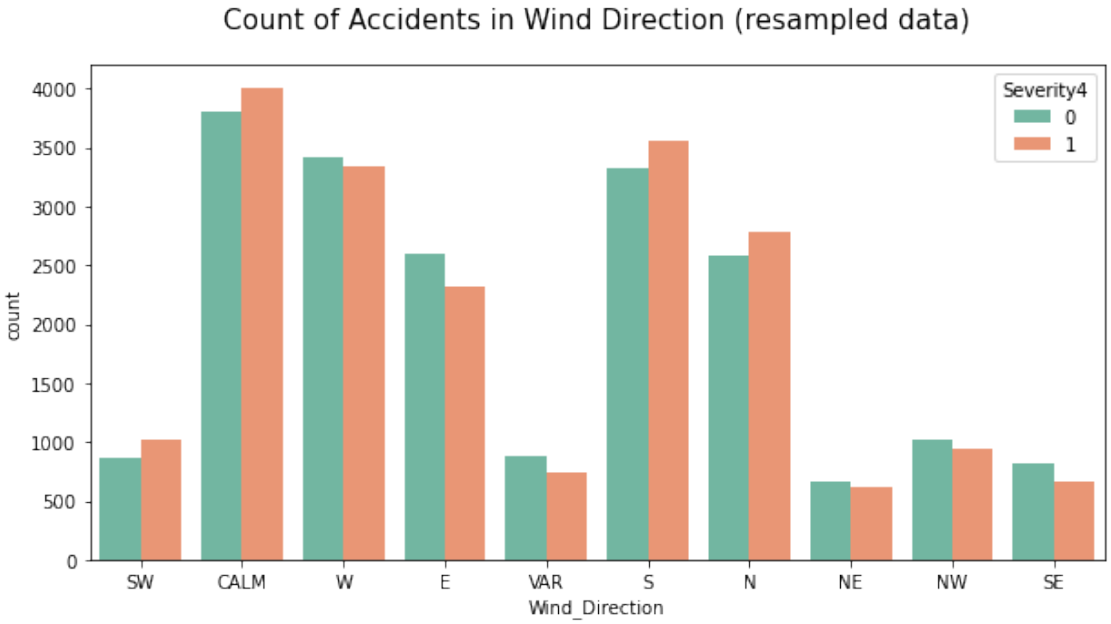


[ ]:
```
fig, axs = plt.subplots(ncols=2, nrows=4, figsize=(15, 10))
plt.subplots_adjust(hspace=0.4,wspace = 0.6)
for i, feature in enumerate(weather, 1):
    plt.subplot(2, 4, i)
    sns.countplot(x=feature, hue='Severity4', data=df_bl ,palette="Set2")

    plt.xlabel('{}'.format(feature), size=12, labelpad=3)
    plt.ylabel('Accident Count', size=12, labelpad=3)
    plt.tick_params(axis='x', labelsize=12)
    plt.tick_params(axis='y', labelsize=12)

    plt.legend(['0', '1'], loc='upper right', prop={'size': 10})
    plt.title('Count of Severity in \n {} Feature'.format(feature), size=14,␣
 ↪y=1.05)
fig.suptitle('Count of Accidents by Weather Features (resampled data)',␣
 ↪fontsize=18)
plt.show()
```



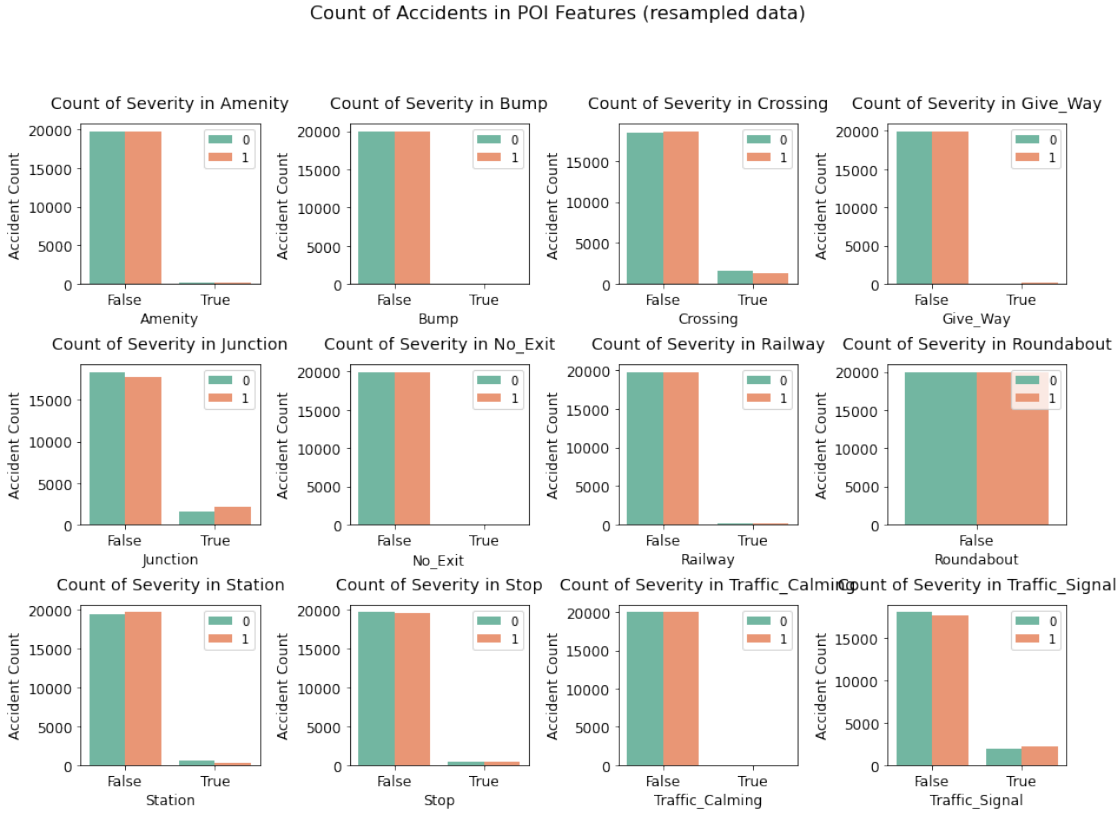
[ ]:
```
df = df.drop(['Heavy_Rain','Heavy_Snow','Fog'], axis  = 1)
```

[ ]:
```
plt.figure(figsize=(10,5))
chart = sns.countplot(x='Wind_Direction', hue='Severity4', data=df_bl␣
↪,palette="Set2")
plt.title("Count of Accidents in Wind Direction (resampled data)", size=15, y=1.
↪05)
plt.show()
```



[ ]:
```
df = df.drop(['Wind_Direction'], axis=1)
```

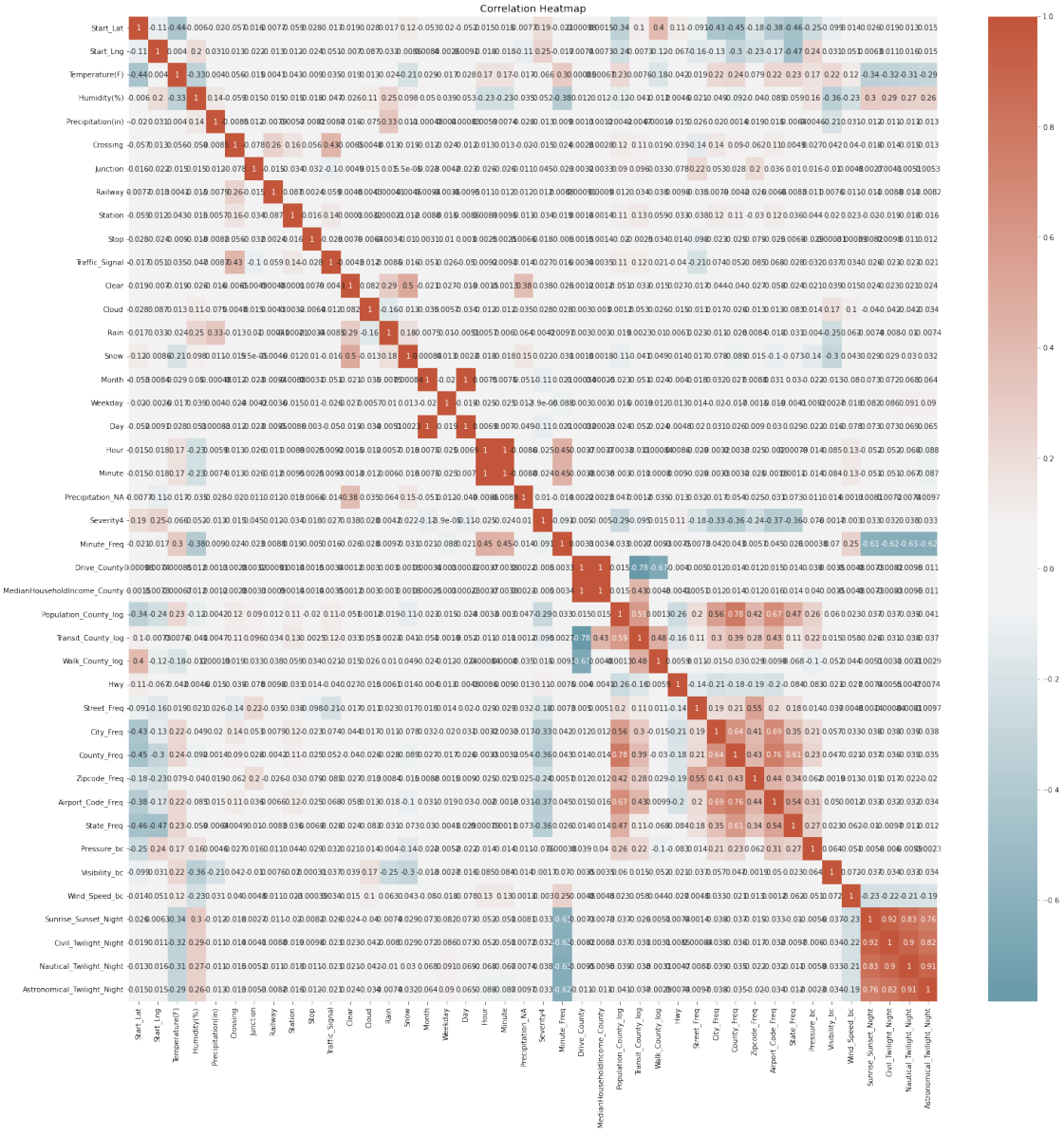
[ ]:
```
POI_features =␣
 ↪['Amenity','Bump','Crossing','Give_Way','Junction','No_Exit','Railway','Roundabout','Statio

fig, axs = plt.subplots(ncols=3, nrows=4, figsize=(15, 10))

plt.subplots_adjust(hspace=0.5,wspace = 0.5)
for i, feature in enumerate(POI_features, 1):
    plt.subplot(3, 4, i)
    sns.countplot(x=feature, hue='Severity4', data=df_bl ,palette="Set2")

    plt.xlabel('{}'.format(feature), size=12, labelpad=3)
    plt.ylabel('Accident Count', size=12, labelpad=3)
    plt.tick_params(axis='x', labelsize=12)
    plt.tick_params(axis='y', labelsize=12)

    plt.legend(['0', '1'], loc='upper right', prop={'size': 10})
    plt.title('Count of Severity in {}'.format(feature), size=14, y=1.05)
fig.suptitle('Count of Accidents in POI Features (resampled data)',y=1.02,␣
 ↪fontsize=16)
plt.show()
```



[ ]:
```
df= df.
 ↪drop(['Amenity','Bump','Give_Way','No_Exit','Roundabout','Traffic_Calming'],␣
 ↪axis=1)
```

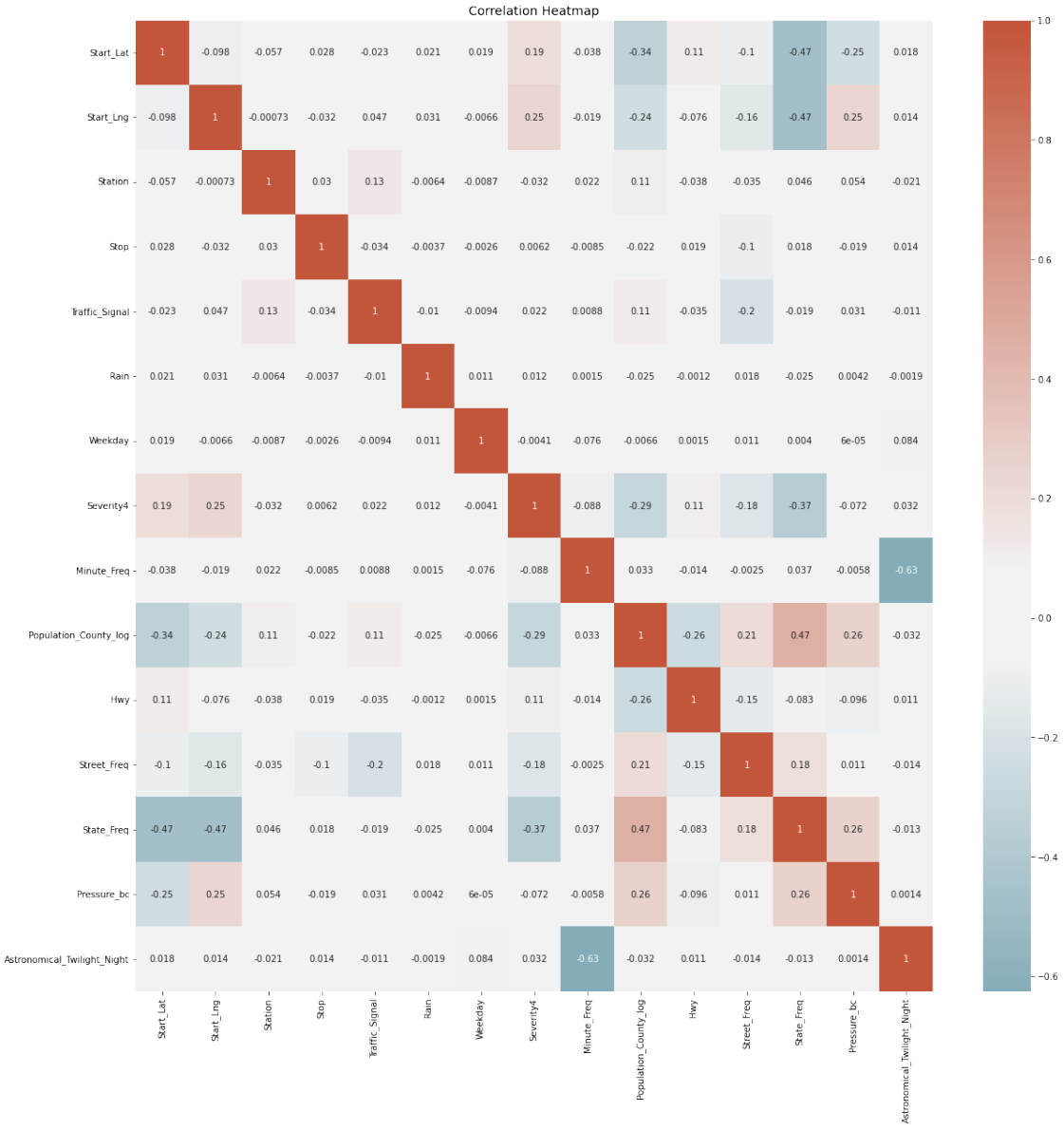
[ ]:
```
# one-hot encoding
df[period_features] = df[period_features].astype('category')
df = pd.get_dummies(df, columns=period_features, drop_first=True)
```

[ ]:
```
# resample again
df_bl = resample(df, 'Severity4', 20000)

# plot correlation
df_bl['Severity4'] = df_bl['Severity4'].astype(int)
plt.figure(figsize=(25,25))
cmap = sns.diverging_palette(220, 20, sep=20, as_cmap=True)
sns.heatmap(df_bl.corr(), annot=True,cmap=cmap, center=0).
 ↪set_title("Correlation Heatmap", fontsize=14)
plt.show()
```




```
[ ]: df = df.drop(['Temperature(F)', 'Humidity(%)', 'Precipitation(in)',␣
     ↪'Precipitation_NA','Visibility_bc', 'Wind_Speed_bc',
                'Clear','Cloud','Snow','Crossing','Junction','Railway','Month',
                'Hour', 'Day','Minute','MedianHouseholdIncome_County',␣
     ↪'Transit_County_log',
                'Walk_County_log','Drive_County',␣
     ↪'City_Freq','County_Freq','Airport_Code_Freq','Zipcode_Freq',
                'Sunrise_Sunset_Night', 'Civil_Twilight_Night',␣
     ↪'Nautical_Twilight_Night'], axis=1)
```

[ ]:
```
# resample again
df_bl = resample(df, 'Severity4', 20000)

# plot correlation
df_bl['Severity4'] = df_bl['Severity4'].astype(int)
plt.figure(figsize=(20,20))
cmap = sns.diverging_palette(220, 20, sep=20, as_cmap=True)
sns.heatmap(df_bl.corr(), annot=True,cmap=cmap, center=0).
 ↪set_title("Correlation Heatmap", fontsize=14)
plt.show()
```



3.7 One-hot Encoding One-hot encode categorical features.

[ ]:
```
df = df.replace([True, False], [1,0])

cat = ['Side','Timezone','Weekday']
df[cat] = df[cat].astype('category')
df = pd.get_dummies(df, columns=cat, drop_first=True)

df_int = df.select_dtypes(include=['int']).apply(pd.
 ↪to_numeric,downcast='unsigned')
df_float = df.select_dtypes(include=['float']).apply(pd.
 ↪to_numeric,downcast='float')
df = pd.concat([df.select_dtypes(include=['uint8']),df_int,df_float],axis=1)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2436356 entries, 0 to 73317
Data columns (total 24 columns):
 #   Column                       Dtype
---  ------                       -----
 0   Astronomical_Twilight_Night  uint8
 1   Side_R                       uint8
 2   Timezone_US/Eastern          uint8
 3   Timezone_US/Mountain         uint8
 4   Timezone_US/Pacific          uint8
 5   Weekday_1                    uint8
 6   Weekday_2                    uint8
 7   Weekday_3                    uint8
 8   Weekday_4                    uint8
 9   Weekday_5                    uint8
 10  Weekday_6                    uint8
 11  Station                      uint8
 12  Stop                         uint8
 13  Traffic_Signal               uint8
 14  Rain                         uint8
 15  Severity4                    uint8
 16  Hwy                          uint8
 17  Start_Lat                    float32
 18  Start_Lng                    float32
 19  Minute_Freq                  float32
 20  Population_County_log        float32
 21  Street_Freq                  float32
 22  State_Freq                   float32
 23  Pressure_bc                  float32
dtypes: float32(7), uint8(17)
memory usage: 123.1 MB
```

4 Model

Imbalance ratio of this dataset is about 100, which is the key problem we need to deal with. There are several ways to handle it: 1. **under-sampling** (I didn't use over-sampling because this dataset is large enough and over-sampling is very likely to casue overfitting) 2. **modify the loss function** 3. **ensemble methods** * EasyEnsemble * BalanceCascade

[ ]:
```
from sklearn.model_selection import GridSearchCV, KFold, train_test_split,␣
↪cross_val_predict
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier
from imblearn.under_sampling import RandomUnderSampler
from collections import Counter
import warnings
warnings.filterwarnings('ignore')
```

4.1 Train Test Split

[ ]:
```
# split X, y
X = df.drop('Severity4', axis=1)
y= df['Severity4']

# split train, test
X_train, X_test, y_train, y_test = train_test_split(\
  X, y, test_size=0.30, random_state=42)
```

4 Random Forest with balanced class weights under-sampling + modify the loss function

[ ]:
```
# Randomly undersample majority class to about 10 times of minority class
rus = RandomUnderSampler(sampling_strategy = 0.1, random_state=42)
X_train_res, y_train_res = rus.fit_resample(X_train, y_train)
print ("Distribution of class labels before resampling {}".
↪format(Counter(y_train)))
print ("Distribution of class labels after resampling {}".
↪format(Counter(y_train_res)))
```

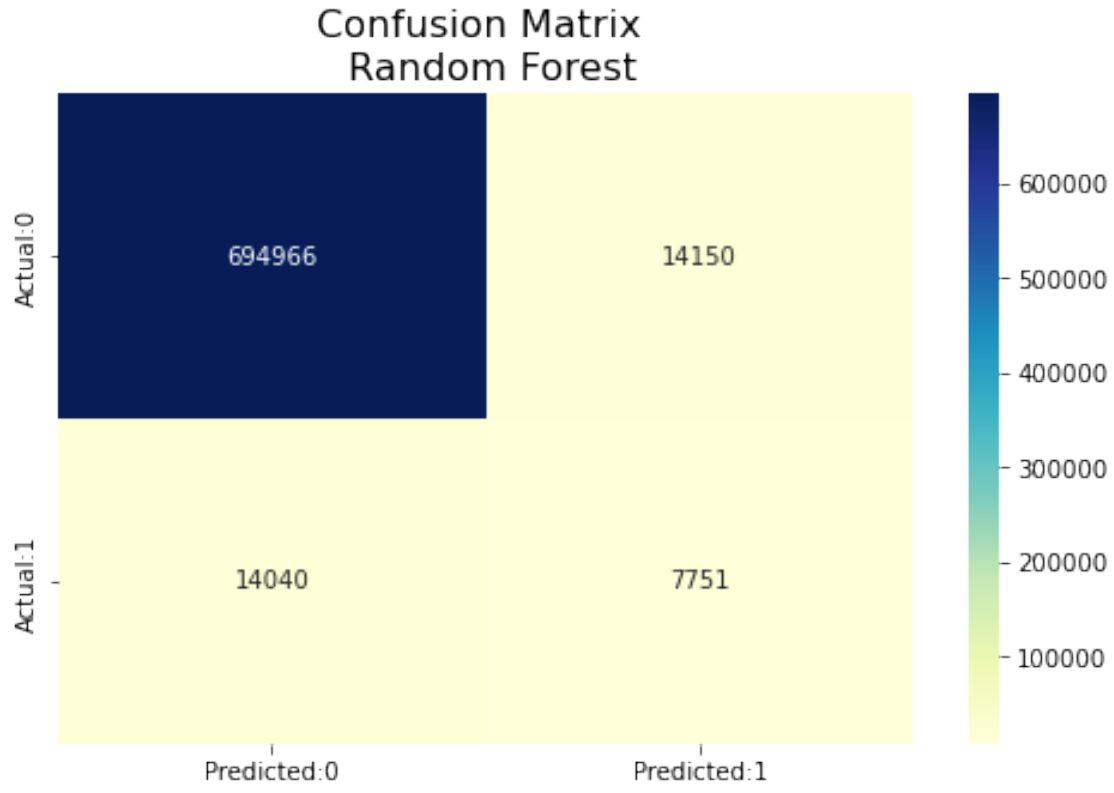
```
Distribution of class labels before resampling Counter({0: 1654176, 1: 51273})
Distribution of class labels after resampling Counter({0: 512730, 1: 51273})
```

[ ]:
```
clf_base = RandomForestClassifier()
grid = {'n_estimators': [10, 50, 100],
        'max_features': ['auto','sqrt']}
clf_rf = GridSearchCV(clf_base, grid, cv=5, n_jobs=8, scoring='f1_macro')

clf_rf.fit(X_train_res, y_train_res)
y_pred = clf_rf.predict(X_test)

print (classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support
```

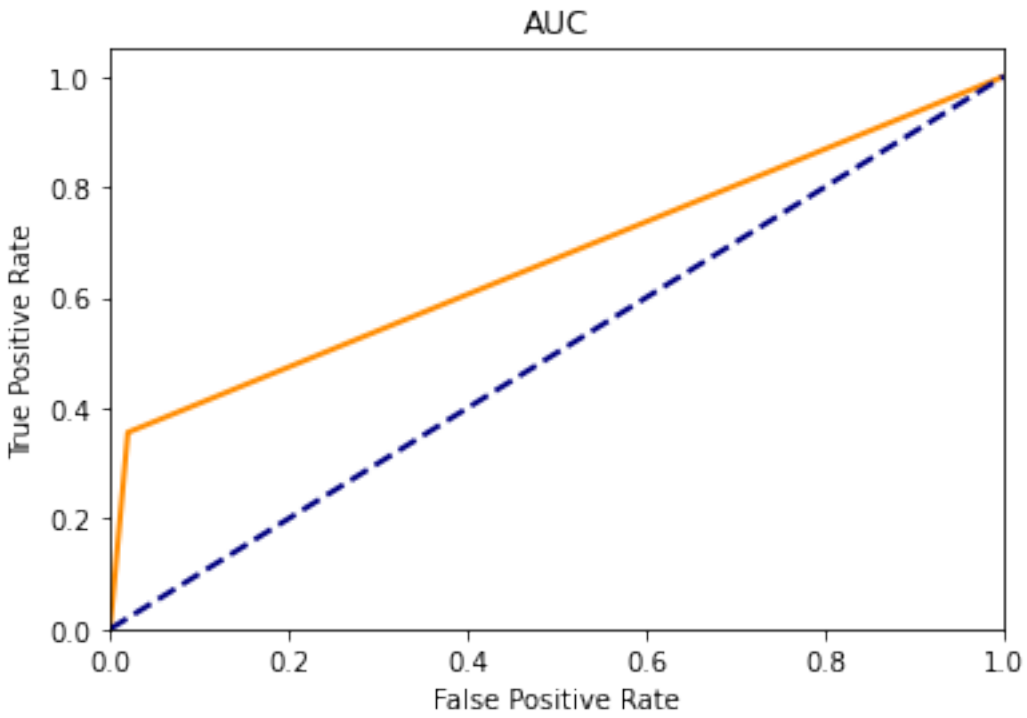
```
              0       0.98      0.98      0.98    709116
              1       0.35      0.36      0.35     21791

       accuracy                           0.96    730907
      macro avg       0.67      0.67      0.67    730907
   weighted avg       0.96      0.96      0.96    730907
```

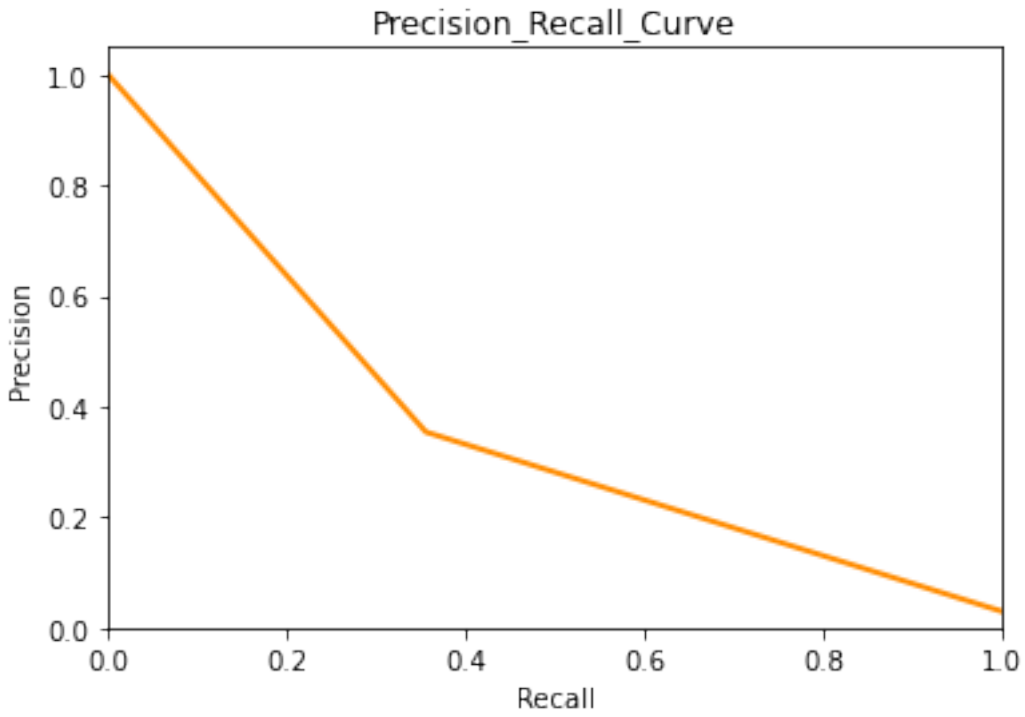
[ ]:
```
confmat = confusion_matrix(y_true=y_test, y_pred=y_pred)

conf_matrix = pd.DataFrame(data=confmat,
                           columns=['Predicted:0','Predicted:1'],index=['Actual:
 ↪0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu").set_title(
    "Confusion Matrix \n Random Forest", fontsize=16)
plt.show()
```



[ ]:
```
from sklearn.metrics import roc_curve, auc,precision_recall_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred,pos_label=1)
```

```
precision, recall, thresholds = precision_recall_curve(y_test,y_pred,␣
 ↪pos_label=1)
plt.figure();
plt.plot(fpr, tpr, color='darkorange', lw=2);
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--');
plt.xlim([0.0, 1.0]);
plt.ylim([0.0, 1.05]);
plt.xlabel('False Positive Rate');
plt.ylabel('True Positive Rate');
plt.title('AUC');
plt.show();
plt.figure();
plt.plot(recall, precision, color='darkorange', lw=2);
plt.xlim([0.0, 1.0]);
plt.ylim([0.0, 1.05]);
plt.xlabel('Recall');
plt.ylabel('Precision');
plt.title('Precision_Recall_Curve');
plt.show();
```





[ ]:
```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('usa-car-accidents-severity-prediction.ipynb')
```

```
--2022-04-24 00:17:54--  https://raw.githubusercontent.com/brpy/colab-
pdf/master/colab_pdf.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
185.199.108.133, 185.199.109.133, 185.199.110.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 1864 (1.8K) [text/plain]
Saving to: ‘colab_pdf.py’

colab_pdf.py        100%[===================>]   1.82K  --.-KB/s    in 0s

2022-04-24 00:17:54 (34.5 MB/s) - ‘colab_pdf.py’ saved [1864/1864]


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.
```

```
Extracting templates from packages: 100%
```