Open in Colab
(https://colab.research.google.com/github/pcarugno/Coursera_Capstone/blob/coursera_capstone/Coursera_Ca
%20Report.ipynb)

Coursera Capstone Project

Predict the severity of an accident in Seattle - Final Report

1. Introduction

The aim of this project is to develop a machine learning tool that will help predicting the severity of an accident based on some key features (such as weather, driving conditions, road conditions etc). This problem is particularly important for the city of Seattle: the city has been moving towards achieving 0 traffic deaths and serious injuries by 2030 and has launched Vision Zero (https://www.seattle.gov/visionzero) in 2015, but as explained in this Seattle Transit Blog entry from 2018 (https://seattletransitblog.com/2018/01/05/traffic-volumes-collision-rate/), collisions with fatal or serious injury jumped *16.5%* in 2016, even as traffic volumes remained nearly unchanged. It is hence of paramount importance to develop techniques that can help understand, predict and even avoid accidents based on their severity and that's what this project is about.

2. Data

The data being used comprises every accident occurred in the city of Seattle between 2004 and 2020. The dimensions of the dataset are:

In [4]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import folium
```

In [5]:

```
import warnings
warnings.filterwarnings("ignore")
```

In [6]:

```
coll_df = pd.read_csv(r'../Python/Data-Collisions.csv')
print('Dimensions of dataset:', coll_df.shape)
```

```
Dimensions of dataset: (194673, 38)
```

The dataset is hence comprised of 194673 records and 38 features. More information on the data and the adopted methodology are presented in the next section

3. Methodology

In this section the exploratory data analysis, statistical testing and machine learning methodology adopted will be described.

3.a Exploratory Data Analysis

To perform a meaningful exploratory data analysis we start by looking at the provided dataset global characteristics, such as number of features, whether these can be grouped in categories and what are the global statistics of the dataset.

In [7]:

```
coll_df.head()
```

Out[7]:

	SEVERITYCODE	X	Y	OBJECTID	INCKEY	COLDETKEY	REPORTNO
0	2	-122.323148	47.703140	1	1307	1307	3502005
1	1	-122.347294	47.647172	2	52200	52200	2607959
2	1	-122.334540	47.607871	3	26700	26700	1482393
3	1	-122.334803	47.604803	4	1144	1144	3503937
4	2	-122.306426	47.545739	5	17700	17700	1807429

5 rows × 38 columns

In [8]:

```
coll_df.columns
```

Out[8]:

```
Index(['SEVERITYCODE', 'X', 'Y', 'OBJECTID', 'INCKEY', 'COLDETKEY', 'REPOR
TNO',
       'STATUS', 'ADDRTYPE', 'INTKEY', 'LOCATION', 'EXCEPTRSNCODE',
       'EXCEPTRSNDESC', 'SEVERITYCODE.1', 'SEVERITYDESC', 'COLLISIONTYPE',
       'PERSONCOUNT', 'PEDCOUNT', 'PEDCYLCOUNT', 'VEHCOUNT', 'INCDATE',
       'INCDTTM', 'JUNCTIONTYPE', 'SDOT_COLCODE', 'SDOT_COLDESC',
       'INATTENTIONIND', 'UNDERINFL', 'WEATHER', 'ROADCOND', 'LIGHTCOND',
       'PEDROWNOTGRNT', 'SDOTCOLNUM', 'SPEEDING', 'ST_COLCODE', 'ST_COLDES
C',
       'SEGLANEKEY', 'CROSSWALKKEY', 'HITPARKEDCAR'],
      dtype='object')
```

We create a dictionary that contains the description of each feature in order to have a better idea of what each predictor/label means. To do so, we use the metadata associated with this dataset.

In [9]:

```
coll_dict = {
    "SEVERITYCODE": "A code that corresponds to the severity of the collision 3-fatalit
y, 2b-serious injury, 2-injury, 1-prop damage, 0-unknown",
    "X": "X location of accident (lat)",
    "Y": "Y location of accident (long)",
    "OBJECTID": "ESRI unique identifier",
    "INCKEY": "A unique key for the incident",
    "COLDETKEY": "Secondary key for the incident",
    "REPORTNO": "Number of report (?)",
    "STATUS": "Status of report (?)",
    "ADDRTYPE": "Collision address type: Alley, Block, Intersection",
    "INTKEY": "Key that corresponds to the intersection associated with a collision",
    "LOCATION": "Description of the general location of the collision",
    "EXCEPTRSNCODE": "",
    "EXCEPTRSNDESC": "",
    "SEVERITYCODE.1": "",
    "SEVERITYDESC": "A detailed description of the severity of the collision",
    "COLLISIONTYPE": "Collision type",
    "PERSONCOUNT": "The total number of people involved in the collision",
    "PEDCOUNT": "The number of pedestrians involved in the collision. This is entered b
y the state.",
    "PEDCYLCOUNT": "The number of bicycles involved in the collision. This is entered b
y the state.",
    "VEHCOUNT": "The number of vehicles involved in the collision. This is entered by t
he state.",
    "INCDATE": "The date of the incident",
    "INCDTTM": "The date and time of the incident",
    "JUNCTIONTYPE": "Category of junction at which collision took place",
    "SDOT_COLCODE": "A code given to the collision by SDOT",
    "SDOT_COLDESC": "A description of the collision corresponding to the collision cod
e",
    "INATTENTIONIND": "Whether or not collision was due to inattention (Y/N)",
    "UNDERINFL": "Whether or not a driver involved was under the influence of drugs or
 alcohol",
    "WEATHER": "A description of the weather conditions during the time of the collisio
n",
    "ROADCOND": "The condition of the road during the collision",
    "LIGHTCOND": "The light conditions during the collision",
    "PEDROWNOTGRNT": "Whether or not the pedestrian right of way was not granted (Y/N)"
,
    "SDOTCOLNUM": "A number given to the collision by SDOT",
    "SPEEDING": "Whether or not speeding was a factor in the collision (Y/N)",
    "ST_COLCODE": "State-defined collision code",
    "ST_COLDESC": "A description taht corresponds to the state's coding designation",
    "SEGLANEKEY": "A key for the lane segment in which the collision occurred",
    "CROSSWALKKEY" : "A key for the crosswalk at which the collision occurred",
    "HITPARKEDCAR" : "Whether or not the collision in volved hitting a parked car (Y/
N)"
}
```

From these preliminaries we can already see a potential grouping of features of the dataset:

- location -> X, Y, ADDRTYPE, LOCATION, JUNCTIONTYPE;
- date/time - season too -> INCDATE, INCDTTM;
- collision type -> SEVERITYDESC, COLLISIONTYPE, SDOT_COLCODE, SDOT_COLDESC
- collision causes and circumstances (including weather, road and light conditions) -> INATTENTIONIND, UNDERINFL, PEDROWNOTGRNT, SPEEDING, HITPARKEDCAR,WEATHER, ROADCOND, LIGHTCOND;
- number of pedestrian, vehicle, bicycles involved -> PERSONCOUNT, PEDCOUNT, PEDCYLCOUNT, VEHCOUNT.

3.a.1 - Location

So we start by looking at the first group of features: **location**. We do so by creating a location dataframe: loc_df.

In [10]:

```
loc_df = coll_df[['SEVERITYCODE','X','Y','ADDRTYPE','LOCATION','JUNCTIONTYPE']]
loc_df.head()
```

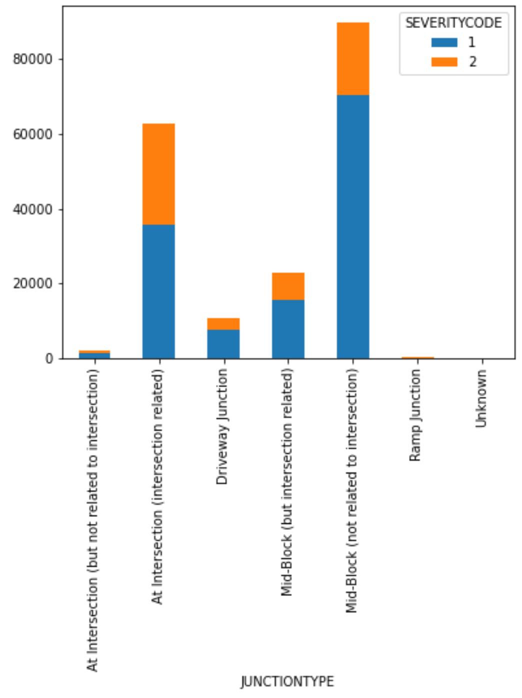
Out[10]:

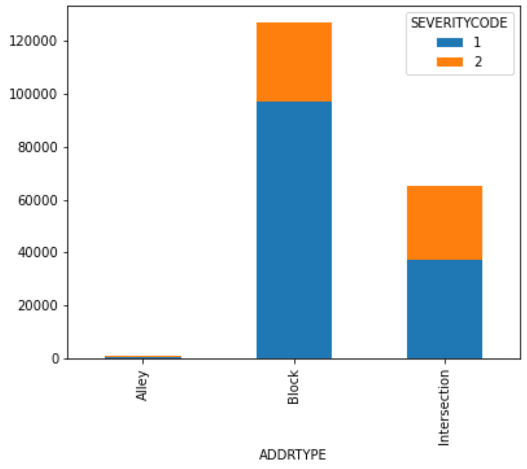
	SEVERITYCODE	X	Y	ADDRTYPE	LOCATION	JUNCTIONTYPE
0	2	-122.323148	47.703140	Intersection	5TH AVE NE AND NE 103RD ST	At Intersection (intersection related)
1	1	-122.347294	47.647172	Block	AURORA BR BETWEEN RAYE ST AND BRIDGE WAY N	Mid-Block (not related to intersection)
2	1	-122.334540	47.607871	Block	4TH AVE BETWEEN SENECA ST AND UNIVERSITY ST	Mid-Block (not related to intersection)
3	1	-122.334803	47.604803	Block	2ND AVE BETWEEN MARION ST AND MADISON ST	Mid-Block (not related to intersection)
4	2	-122.306426	47.545739	Intersection	SWIFT AVE S AND SWIFT AV OFF RP	At Intersection (intersection related)

We can have a look at the number of occurrences for each severity code based on ADDRTYPE and JUNCTIONTYPE to understand whether accidents occurred in some particular location rather than others.

In [11]:

```
figure, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5));
# figure.tight_layout()
plt.subplots_adjust(wspace=0.4)

pd.crosstab(loc_df['JUNCTIONTYPE'],loc_df['SEVERITYCODE'] ).plot(kind='bar', stacked=True,ax=ax1);
pd.crosstab(loc_df['ADDRTYPE'],loc_df['SEVERITYCODE'] ).plot(kind='bar', stacked=True,ax=ax2);
```





From this first 2 plots we can observe that:

- almost no accidents are occurring at ramp junction or at intersections (but not related to intersections)
- the majority of the accidents are either happening at intersection (intersection related) or at mid-block but not related to intersections
- of these two, the accidents happening at intersection (intersection related) are split half-half in terms of severity while for the mid-block related accidents the majority of the accidents are of severity 1
- when accidents are occurring at intersection, the severity is split between 1 and 2 almost equally in terms of occurrences while the type of accidents occurring at "block" are dominated by severity 1 accidents.

We can also have a look at the statistics of the loc_df database and observe where the mean location of accidents is within Seattle. Using the standard deviation along the x and y axes (i.e. longitude and latitude) we can draw a square around this mean location which could represent a rough estimation of the area where accidents occurred between 2004 and 2020.

In [12]:

```
X_std = loc_df.X.std()
X_mean = loc_df.X.mean()
Y_std = loc_df.Y.std()
Y_mean = loc_df.Y.mean()
```

In [13]:

```
# define the world map centered around Canada with a low zoom level
seattle = folium.Map(location=[Y_mean, X_mean], zoom_start=12)
folium.Marker([Y_mean, X_mean]).add_to(seattle)
# now I define the square of the box that should contain > 95% of collisions
points=[[Y_mean+Y_std,X_mean-X_std],[Y_std+Y_mean,X_mean+X_std],[Y_mean-Y_std,X_mean+X_
std],[Y_mean-Y_std,X_mean-X_std],[Y_mean+Y_std,X_mean-X_std]]
folium.PolyLine(points,color="red", weight=2.5, opacity=1).add_to(seattle)
seattle
```

Out[13]:

Make this Notebook Trusted to load map: File -> Trust Notebook

3.a.2 Date/time

Now we move on to analysing the **date/time** group of features, i.e. INCDATE, INCDTTM. We create a new dataframe called time_df.

In [14]:

```
time_df = coll_df[['SEVERITYCODE','INCDATE','INCDTTM']]
time_df.head()
```

Out[14]:

	SEVERITYCODE	INCDATE	INCDTTM
0	2	2013/03/27 00:00:00+00	3/27/2013 2:54:00 PM
1	1	2006/12/20 00:00:00+00	12/20/2006 6:55:00 PM
2	1	2004/11/18 00:00:00+00	11/18/2004 10:20:00 AM
3	1	2013/03/29 00:00:00+00	3/29/2013 9:26:00 AM
4	2	2004/01/28 00:00:00+00	1/28/2004 8:04:00 AM

We notice already that INCDATE is superficial, so we can drop it, split the INCDTTM in year, month, day, hours, minutes, seconds

In [15]:

```
import datetime
```

In [16]:

```
time_df['INCDTTM'] =  pd.to_datetime(time_df['INCDTTM'], infer_datetime_format=True)
time_df['Time'] = [datetime.datetime.time(d) for d in time_df['INCDTTM']]
time_df['Date'] = [datetime.datetime.date(d) for d in time_df['INCDTTM']]
time_df['Year'] = [datetime.datetime.date(d).year for d in time_df['INCDTTM']]
time_df['Month'] = [datetime.datetime.date(d).month for d in time_df['INCDTTM']]
time_df['Day'] = [datetime.datetime.date(d).day for d in time_df['INCDTTM']]
time_df['Hours'] = [datetime.datetime.time(d).hour for d in time_df['INCDTTM']]
time_df['Minutes'] = [datetime.datetime.time(d).minute for d in time_df['INCDTTM']]
time_df['Seconds'] = [datetime.datetime.time(d).second for d in time_df['INCDTTM']]
time_df = time_df.drop(columns = ['INCDATE'])
```

In [17]:

```
time_df.head()
```

Out[17]:

	SEVERITYCODE	INCDTTM	Time	Date	Year	Month	Day	Hours	Minutes	Seconds
0	2	2013-03-27 14:54:00	14:54:00	2013-03-27	2013	3	27	14	54	0
1	1	2006-12-20 18:55:00	18:55:00	2006-12-20	2006	12	20	18	55	0
2	1	2004-11-18 10:20:00	10:20:00	2004-11-18	2004	11	18	10	20	0
3	1	2013-03-29 09:26:00	09:26:00	2013-03-29	2013	3	29	9	26	0
4	2	2004-01-28 08:04:00	08:04:00	2004-01-28	2004	1	28	8	4	0

We perform a One Hot Encoding operation to convert the SEVERITYCODE feature into columns "Severity = 1" and "Severity = 2"

In [18]:

```
time_df['Severity = 1']= time_df['SEVERITYCODE'].apply(lambda x: 1 if (x<2)  else 0)
time_df['Severity = 2']= time_df['SEVERITYCODE'].apply(lambda x: 1 if (x>=2)  else 0)
time_df.head()
#Alternative method using pd.get_dummies:
#time_df = pd.merge(time_df, pd.get_dummies(time_df.SEVERITYCODE),left_index=True, righ
t_index=True)
```

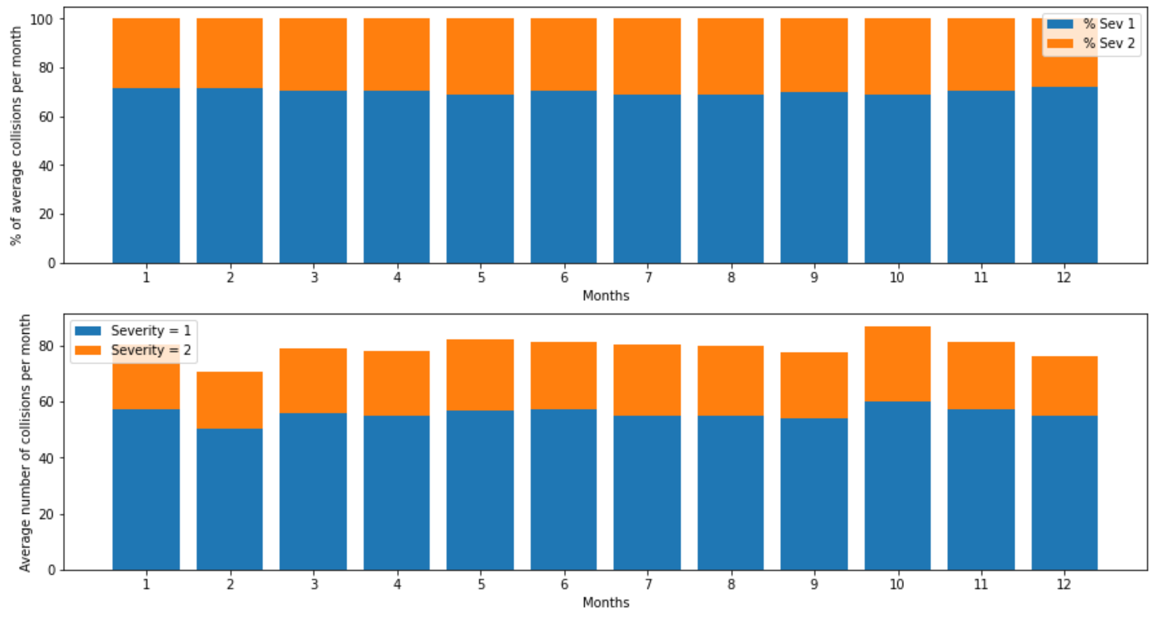
Out[18]:

	SEVERITYCODE	INCDTTM	Time	Date	Year	Month	Day	Hours	Minutes	Seconds
0	2	2013-03-27 14:54:00	14:54:00	2013-03-27	2013	3	27	14	54	0
1	1	2006-12-20 18:55:00	18:55:00	2006-12-20	2006	12	20	18	55	0
2	1	2004-11-18 10:20:00	10:20:00	2004-11-18	2004	11	18	10	20	0
3	1	2013-03-29 09:26:00	09:26:00	2013-03-29	2013	3	29	9	26	0
4	2	2004-01-28 08:04:00	08:04:00	2004-01-28	2004	1	28	8	4	0

We now look at the average number of collisions per month averaged between 2004 and 2020 to see if there are any obvious spikes or odd behaviours - this is shown as percentage of Severity 1 and percentage of Severity 2 accidents.

In [19]:

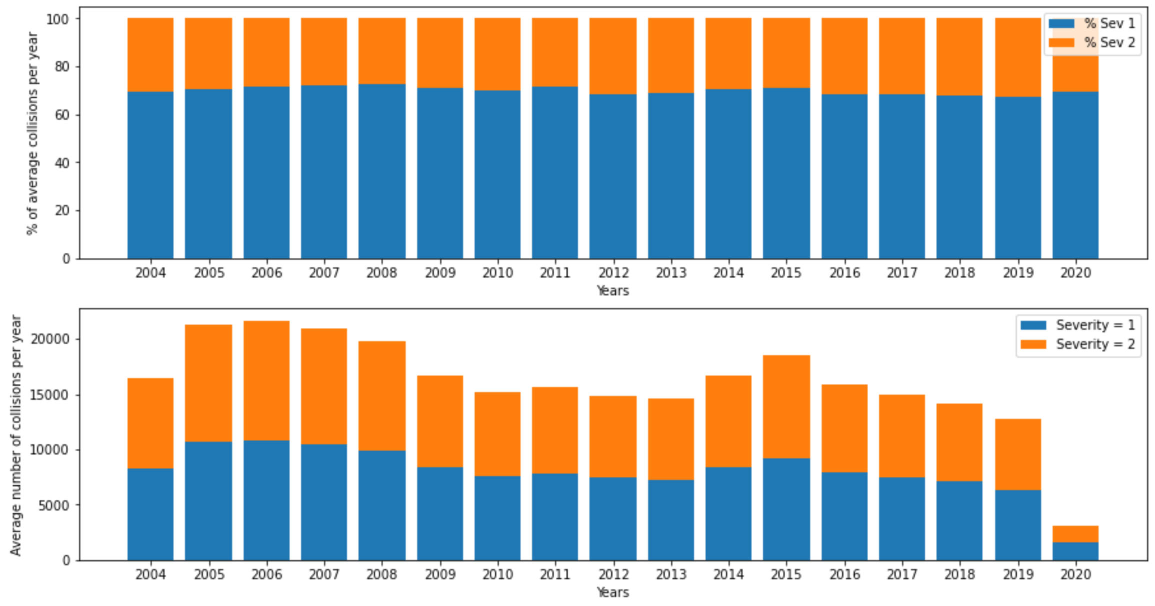
```
monthly_average = time_df.groupby(['Month'])['Severity = 1','Severity = 2'].sum()/(17*1
2)
monthly_average = monthly_average.reset_index()
monthly_average.head()

from matplotlib.pyplot import figure;

monthly_average['% Sev 1'] = 100*monthly_average['Severity = 1'] / (monthly_average['Se
verity = 1']+monthly_average['Severity = 2'])
monthly_average['% Sev 2'] = 100*monthly_average['Severity = 2'] / (monthly_average['Se
verity = 1']+monthly_average['Severity = 2'])

plt.figure(figsize=(15,8))
plt.subplot(2,1,1)
plt.bar(monthly_average.Month, monthly_average['% Sev 1']);
plt.bar(monthly_average.Month, monthly_average['% Sev 2'], bottom =monthly_average['% S
ev 1']);
plt.legend(['% Sev 1', '% Sev 2']);
plt.xticks(monthly_average.Month, monthly_average['Month'], rotation='horizontal');
plt.ylabel('% of average collisions per month');
plt.xlabel('Months');

plt.subplot(2,1,2)
plt.bar(monthly_average.Month, monthly_average['Severity = 1']);
plt.bar(monthly_average.Month, monthly_average['Severity = 2'], bottom =monthly_average
['Severity = 1']);
plt.legend(['Severity = 1', 'Severity = 2']);
plt.xticks(monthly_average.Month, monthly_average['Month'], rotation='horizontal');
plt.ylabel('Average number of collisions per month');
plt.xlabel('Months');

#Alternative way (without doing reset_index above)
#monthly_average.plot.bar(stacked=True)
```



We can see from the plot above that there is no apparent trend and that Severity 1 accidents made up ~70% of total accidents per month between 2004 and 2020. Even looking at the average number of collisions per month doesn't seem to offer much more information. We now turn our attention to the annual trends.

In [20]:

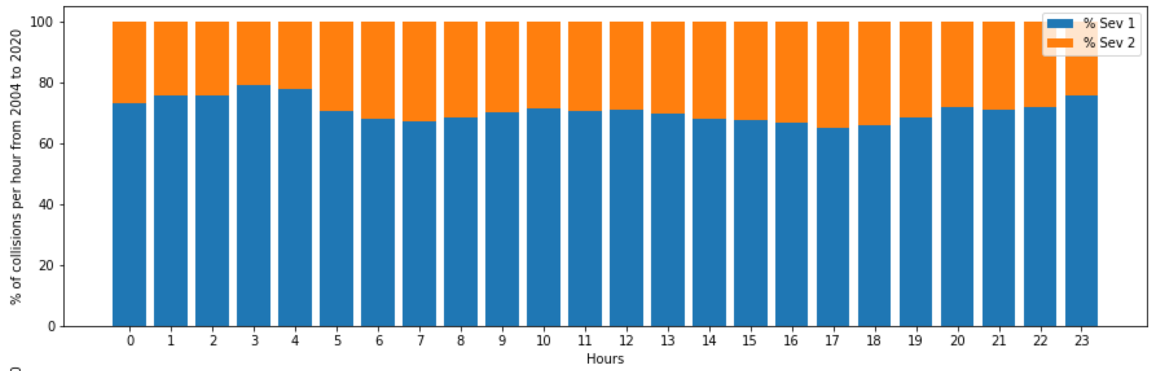
```
annual_trends_df = time_df.groupby(['Year'])['Severity = 1','Severity = 2'].sum();
annual_trends_df = annual_trends_df.reset_index()
plt.figure(figsize=(15,8))
annual_trends_df['% Sev 1'] = 100*annual_trends_df['Severity = 1'] / (annual_trends_df[
'Severity = 1']+annual_trends_df['Severity = 2'])
annual_trends_df['% Sev 2'] = 100*annual_trends_df['Severity = 2'] / (annual_trends_df[
'Severity = 1']+annual_trends_df['Severity = 2'])
plt.subplot(2,1,1)
plt.bar(annual_trends_df.Year, annual_trends_df['% Sev 1']);
plt.bar(annual_trends_df.Year, annual_trends_df['% Sev 2'], bottom =annual_trends_df['%
Sev 1']);
plt.legend(['% Sev 1', '% Sev 2']);
plt.xticks(annual_trends_df.Year, annual_trends_df['Year'], rotation='horizontal');
plt.ylabel('% of average collisions per year');
plt.xlabel('Years');

plt.subplot(2,1,2)
plt.bar(annual_trends_df.Year, annual_trends_df['Severity = 1']);
plt.bar(annual_trends_df.Year, annual_trends_df['Severity = 1'], bottom =annual_trends_
df['Severity = 1']);
plt.legend(['Severity = 1','Severity = 2']);
plt.xticks(annual_trends_df.Year, annual_trends_df['Year'], rotation='horizontal');
plt.ylabel('Average number of collisions per year');
plt.xlabel('Years');
```

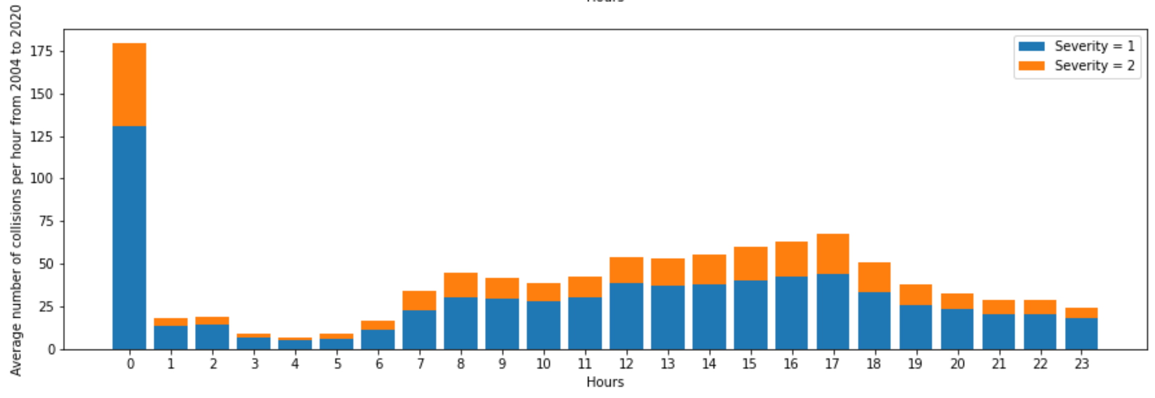


From the first plot we can see that Severity 1 accidents made up ~70% of total accidents per year between 2004 and 2020. However, when looking at the average number of collisions per year we can see that between 2011/2012 to 2015 there was a constant increase in the total number accidents - we then observe a decrease from 2015 to 2019 - this coincides with the introduction of Vision Zero (https://www.seattle.gov/visionzero) in 2015. The 2020 numbers are obviously only partial as the database stops at May 2020.

In [21]:

```
hourly_trends_df = time_df.groupby(['Hours'])['Severity = 1','Severity = 2'].sum()/(16*
12);
hourly_trends_df = hourly_trends_df.reset_index()
plt.figure(figsize=(15,10))
hourly_trends_df['% Sev 1'] = 100*hourly_trends_df['Severity = 1'] / (hourly_trends_df[
'Severity = 1']+hourly_trends_df['Severity = 2'])
hourly_trends_df['% Sev 2'] = 100*hourly_trends_df['Severity = 2'] / (hourly_trends_df[
'Severity = 1']+hourly_trends_df['Severity = 2'])
plt.subplot(2,1,1)
plt.bar(hourly_trends_df.Hours, hourly_trends_df['% Sev 1']);
plt.bar(hourly_trends_df.Hours, hourly_trends_df['% Sev 2'], bottom =hourly_trends_df[
'% Sev 1']);
plt.legend(['% Sev 1', '% Sev 2']);
plt.xticks(hourly_trends_df.Hours, hourly_trends_df['Hours'], rotation='horizontal');
plt.ylabel('% of collisions per hour from 2004 to 2020');
plt.xlabel('Hours');

plt.subplot(2,1,2)
plt.bar(hourly_trends_df.Hours, hourly_trends_df['Severity = 1']);
plt.bar(hourly_trends_df.Hours, hourly_trends_df['Severity = 2'], bottom =hourly_trends
_df['Severity = 1']);
plt.legend(['Severity = 1', 'Severity = 2']);
plt.xticks(hourly_trends_df.Hours, hourly_trends_df['Hours'], rotation='horizontal');
plt.ylabel('Average number of collisions per hour from 2004 to 2020');
plt.xlabel('Hours');
```





These last two plots are a bit more interesting and show the trend of number of collisions per hour, averaged on year and month. We can see 2 peaks: one at midnight and one at 5pm. In particular, the peak at midnight seems to be much higher than the other hourly values - let's dig more into the hours results to see if something odd is happening in the database. We start with searching for all the occurrences in the dataframe where the time of the collisions was 00:00:00 and all the occurrences in the dataframe where the collisions where recorded at other times than 00:00:00 (but still at 00 hours)

In [22]:

```
len(time_df.loc[(time_df.Hours == 0) & (time_df.Minutes==0) & (time_df.Seconds==0)])
```

Out[22]:

```
30526
```

In [23]:

```
len(time_df.loc[(time_df.Hours == 0)])
```

Out[23]:

```
34381
```

In [24]:

```
corrected_midnight = (len(time_df.loc[(time_df.Hours == 0)])-len(time_df.loc[(time_df.H
ours == 0) & (time_df.Minutes==0) & (time_df.Seconds==0)]))/(16*12)
corrected_midnight
```

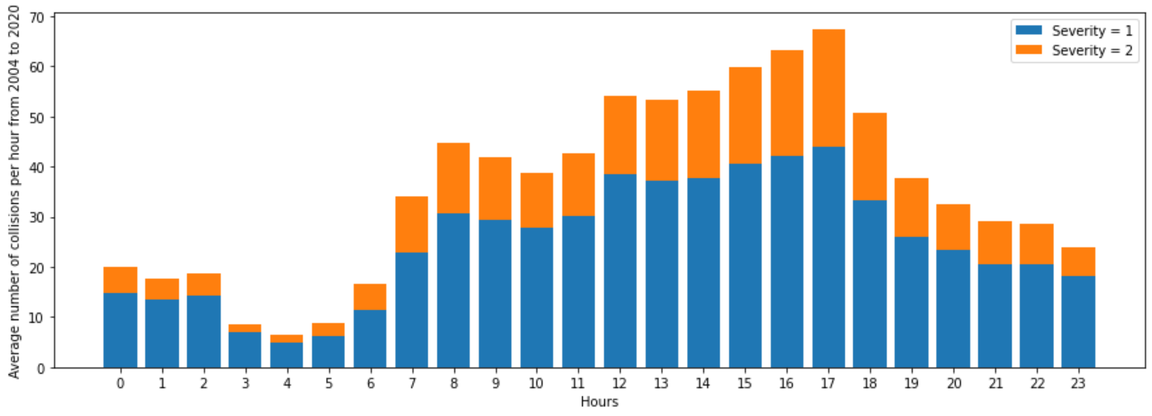
Out[24]:

```
20.078125
```

From this first simple analysis we see that collisions recorded exactly at 00:00:00 time account for 30526 out of the 34381 that happened at 00 hours (basically between 12am and 1am). We can hypothize that those collisions recorded exactly at 00:00:00 were not updated with the correct time and when we look at the collisions that were recorded betwen 12am and 1am (but not exactly at 00:00:00), the average number of collisions per year, per month between 12am and 1am is much more in line with the rest of chart, that's to say ~20. With this information we can try to modify the plot above to a more realistic one - shown below - which now shows 2 relative peaks: one at 8am and one at 5pm, basically coinciding with peak hours with school runs and office commute, with the majority of Severity 2 accidents occuring towards the late afternoon/early evening.

In [25]:

```
hourly_trends_corrected = hourly_trends_df
hourly_trends_corrected['Severity = 1'][0] = hourly_trends_df['% Sev 1'][0]*corrected_m
idnight/100
hourly_trends_corrected['Severity = 2'][0] = hourly_trends_df['% Sev 2'][0]*corrected_m
idnight/100

plt.figure(figsize=(15,5))
plt.bar(hourly_trends_corrected.Hours, hourly_trends_corrected['Severity = 1']);
plt.bar(hourly_trends_corrected.Hours, hourly_trends_corrected['Severity = 2'], bottom
=hourly_trends_corrected['Severity = 1']);
plt.legend(['Severity = 1', 'Severity = 2']);
plt.xticks(hourly_trends_corrected.Hours, hourly_trends_corrected['Hours'], rotation='h
orizontal');
plt.ylabel('Average number of collisions per hour from 2004 to 2020');
plt.xlabel('Hours');
```



3.a.3 Collision type

Now we consider the third group of features, i.e. **collision type** made up of COLLISIONTYPE, SDOT_COLCODE, SDOT_COLDESC. We invoke the dictionary defined at the beginning of this report to remind ourselves what these features are.

In [26]:

```
print('COLLISIONTYPE:',coll_dict['COLLISIONTYPE'], '\nSDOT_COLCODE:', coll_dict['SDOT_C
OLCODE'], '\nSDOT_COLDESC:', coll_dict['SDOT_COLDESC'])

#coll_df.COLLISIONTYPE.unique()
```

```
COLLISIONTYPE: Collision type
SDOT_COLCODE: A code given to the collision by SDOT
SDOT_COLDESC: A description of the collision corresponding to the collisio
n code
```

We assemble a new dataframe called coltype_df with the features above to examine them.

In [27]:

```
colltype_df = coll_df[['SEVERITYCODE','COLLISIONTYPE','SDOT_COLCODE','SDOT_COLDESC']]
colltype_df.head()
```

Out[27]:

	SEVERITYCODE	COLLISIONTYPE	SDOT_COLCODE	SDOT_COLDESC
0	2	Angles	11	MOTOR VEHICLE STRUCK MOTOR VEHICLE, FRONT END ...
1	1	Sideswipe	16	MOTOR VEHICLE STRUCK MOTOR VEHICLE, LEFT SIDE ...
2	1	Parked Car	14	MOTOR VEHICLE STRUCK MOTOR VEHICLE, REAR END
3	1	Other	11	MOTOR VEHICLE STRUCK MOTOR VEHICLE, FRONT END ...
4	2	Angles	11	MOTOR VEHICLE STRUCK MOTOR VEHICLE, FRONT END ...

Looking at this dataframe we notice how some of the information is redundant and in fact can be retrieved by including also the group of features that is number of pedestrian, vehicle, bicycles involved and dropping the columns SDOT_COLCODE and SDOT_COLDESC.

In [28]:

```
colltype_df['PERSONCOUNT'] = coll_df['PERSONCOUNT']
colltype_df['PEDCOUNT'] = coll_df['PEDCOUNT']
colltype_df['PEDCYLCOUNT'] = coll_df['PEDCYLCOUNT']
colltype_df['VEHCOUNT'] = coll_df['VEHCOUNT']
colltype_df = colltype_df.drop(columns=['SDOT_COLCODE','SDOT_COLDESC'])
#one hot encoding to transform SEVERITYCODE column into 2 separate columns
colltype_df['Severity = 1']= colltype_df['SEVERITYCODE'].apply(lambda x: 1 if (x<2)  el
se 0)
colltype_df['Severity = 2']= colltype_df['SEVERITYCODE'].apply(lambda x: 1 if (x>=2)  e
lse 0)
colltype_df['Tot_collisions'] = colltype_df['Severity = 1']+colltype_df['Severity = 2']
colltype_df.head()
```

Out[28]:

	SEVERITYCODE	COLLISIONTYPE	PERSONCOUNT	PEDCOUNT	PEDCYLCOUNT	VEHCOU
0	2	Angles	2	0	0	
1	1	Sideswipe	2	0	0	
2	1	Parked Car	4	0	0	
3	1	Other	3	0	0	
4	2	Angles	2	0	0	

We can now run some statistics on this dataframe. For example we can have a look at how many accidents occured per collision type and how many vehicles, pedestrians and bicycles were involved.

In [29]:

```
colltype_df.groupby('COLLISIONTYPE')['Severity = 1','Severity = 2','Tot_collisions','PE
DCOUNT','PEDCYLCOUNT','VEHCOUNT'].sum()
```

Out[29]:

	Severity = 1	Severity = 2	Tot_collisions	PEDCOUNT	PEDCYLCOUNT	VEHCOUNT
COLLISIONTYPE						
Angles	21050	13624	34674	60	17	71978
Cycles	671	4744	5415	98	5447	5295
Head On	1152	872	2024	0	1	4305
Left Turn	8292	5411	13703	22	14	28117
Other	17591	6112	23703	71	7	34276
Parked Car	45325	2662	47987	72	6	102983
Pedestrian	672	5936	6608	6857	4	6702
Rear Ended	19419	14671	34090	27	7	75753
Right Turn	2347	609	2956	5	3	5985
Sideswipe	16103	2506	18609	18	5	38505

From this dataframe we can see that between 2004 and 2020 the majority of accidents of Severity 1 occurred with parked cars, while the majority of accidents of Severity 2 involved either rear-ended type collisions or angles type collision. Looking at the total amount of collisions, we can also deduce that whenever a rear-ended type collision took place, more than 2 vehicles were involved given the fact that for this type of collisions, VEHCOUNT is more than double the total amount of accidents - this is the same for the collisions type "Angles" and "Parked Car" and it seems to suggest that these were accidents that occurred with multiple cars close to each other e.g. in stationary traffic. From this dataframe we can also read that whenever a collision involved a bicycle, this was of severity 2 for the majority of the cases and a small number of these collisions involved more than one bicycle per single occurrence (PEDCYLCOUNT > VEHCOUNT). This same rationale can be applied to accidents involving pedestrian (PEDCOUNT > VEHCOUNT).

3.a.4 Collision causes and circumstances

Lastly we turn our attention to the group of features named **collisions causes and circumstances**, which is made up not only of features indicating the circumstances surrounding the people involved in the accident but also information about weather, road and light conditions. First, let's remind ourselves what each of these features mean by invoking the *coll_dict* dictionary and then let's create a dataframe called cond_df to start making some analyses.

In [30]:

```
print('INATTENTIONIND:',coll_dict['INATTENTIONIND'], '\nUNDERINFL:', coll_dict['UNDERIN
FL'], '\nPEDROWNOTGRNT:', coll_dict['PEDROWNOTGRNT'], '\nSPEEDING:', coll_dict['SPEEDIN
G'])
print('HITPARKEDCAR:',coll_dict['HITPARKEDCAR'], '\nWEATHER:', coll_dict['WEATHER'], '
\nROADCOND:', coll_dict['ROADCOND'], '\nLIGHTCOND:', coll_dict['LIGHTCOND'])
```

```
INATTENTIONIND: Whether or not collision was due to inattention (Y/N)
UNDERINFL: Whether or not a driver involved was under the influence of dru
gs or alcohol
PEDROWNOTGRNT: Whether or not the pedestrian right of way was not granted
(Y/N)
SPEEDING: Whether or not speeding was a factor in the collision (Y/N)
HITPARKEDCAR: Whether or not the collision in volved hitting a parked car
(Y/N)
WEATHER: A description of the weather conditions during the time of the co
llision
ROADCOND: The condition of the road during the collision
LIGHTCOND: The light conditions during the collision
```

In [31]:

```
cond_df = coll_df[['SEVERITYCODE','INATTENTIONIND','UNDERINFL','PEDROWNOTGRNT','SPEEDIN
G','HITPARKEDCAR','WEATHER','ROADCOND','LIGHTCOND']]
cond_df.head(10)
```

Out[31]:

	SEVERITYCODE	INATTENTIONIND	UNDERINFL	PEDROWNOTGRNT	SPEEDING	HITPARK
0	2	NaN	N	NaN	NaN	
1	1	NaN	0	NaN	NaN	
2	1	NaN	0	NaN	NaN	
3	1	NaN	N	NaN	NaN	
4	2	NaN	0	NaN	NaN	
5	1	NaN	N	NaN	NaN	
6	1	NaN	0	NaN	NaN	
7	2	NaN	N	NaN	NaN	
8	1	NaN	0	NaN	NaN	
9	2	NaN	0	NaN	NaN	

Focusing on the driver/accident conditions first, except weather, road and light conditions, we can first see how many NaN values, incomplete or missing values there are and replace these to create a new df called cond_df_driver.

In [32]:

```
cond_df_driver = cond_df[['SEVERITYCODE','INATTENTIONIND','UNDERINFL','PEDROWNOTGRNT',
'SPEEDING','HITPARKEDCAR']]
for c in cond_df_driver.columns:
    print ("---- %s ---" % c)
    print (cond_df_driver[c].value_counts())
    print ("is nan:", cond_df_driver[c].isnull().sum())
```

```
---- SEVERITYCODE ---
1    136485
2     58188
Name: SEVERITYCODE, dtype: int64
is nan: 0
---- INATTENTIONIND ---
Y    29805
Name: INATTENTIONIND, dtype: int64
is nan: 164868
---- UNDERINFL ---
N    100274
0     80394
Y      5126
1      3995
Name: UNDERINFL, dtype: int64
is nan: 4884
---- PEDROWNOTGRNT ---
Y    4667
Name: PEDROWNOTGRNT, dtype: int64
is nan: 190006
---- SPEEDING ---
Y    9333
Name: SPEEDING, dtype: int64
is nan: 185340
---- HITPARKEDCAR ---
N    187457
Y      7216
Name: HITPARKEDCAR, dtype: int64
is nan: 0
```

In [33]:

```
#Encoding INATTENTIONIND (0 = No, 1 = Yes)
cond_df_driver["INATTENTIONIND"].replace("Y", 1, inplace=True)
cond_df_driver["INATTENTIONIND"].replace(np.nan, 0, inplace=True)

#Encoding UNDERINFL (0 = No, 1 = Yes)
cond_df_driver["UNDERINFL"].replace("N", 0, inplace=True)
cond_df_driver["UNDERINFL"].replace("Y", 1, inplace=True)

#Encoding SPEEDING (0 = No, 1 = Yes)
cond_df_driver["SPEEDING"].replace("Y", 1, inplace=True)
cond_df_driver["SPEEDING"].replace(np.nan, 0, inplace=True)

#Encoding PEDROWNOTGRNT(0 = No, 1 = Yes)
cond_df_driver["PEDROWNOTGRNT"].replace("Y", 1, inplace=True)
cond_df_driver["PEDROWNOTGRNT"].replace(np.nan, 0, inplace=True)

#Encoding HITPARKEDCAR(0 = No, 1 = Yes)
cond_df_driver["HITPARKEDCAR"].replace("Y", 1, inplace=True)
cond_df_driver["HITPARKEDCAR"].replace("N", 0, inplace=True)
cond_df_driver["HITPARKEDCAR"].replace(np.nan, 0, inplace=True)

cond_df_driver.head()
```

Out[33]:

	SEVERITYCODE	INATTENTIONIND	UNDERINFL	PEDROWNOTGRNT	SPEEDING	HITPARK
0	2	0.0	0	0.0	0.0	
1	1	0.0	0	0.0	0.0	
2	1	0.0	0	0.0	0.0	
3	1	0.0	0	0.0	0.0	
4	2	0.0	0	0.0	0.0	

In [34]:

```
cond_df_driver['Severity = 1']= cond_df_driver['SEVERITYCODE'].apply(lambda x: 1 if (x<
2)  else 0)
cond_df_driver['Severity = 2']= cond_df_driver['SEVERITYCODE'].apply(lambda x: 1 if (x>
=2)  else 0)
cond_df_driver.head()
```

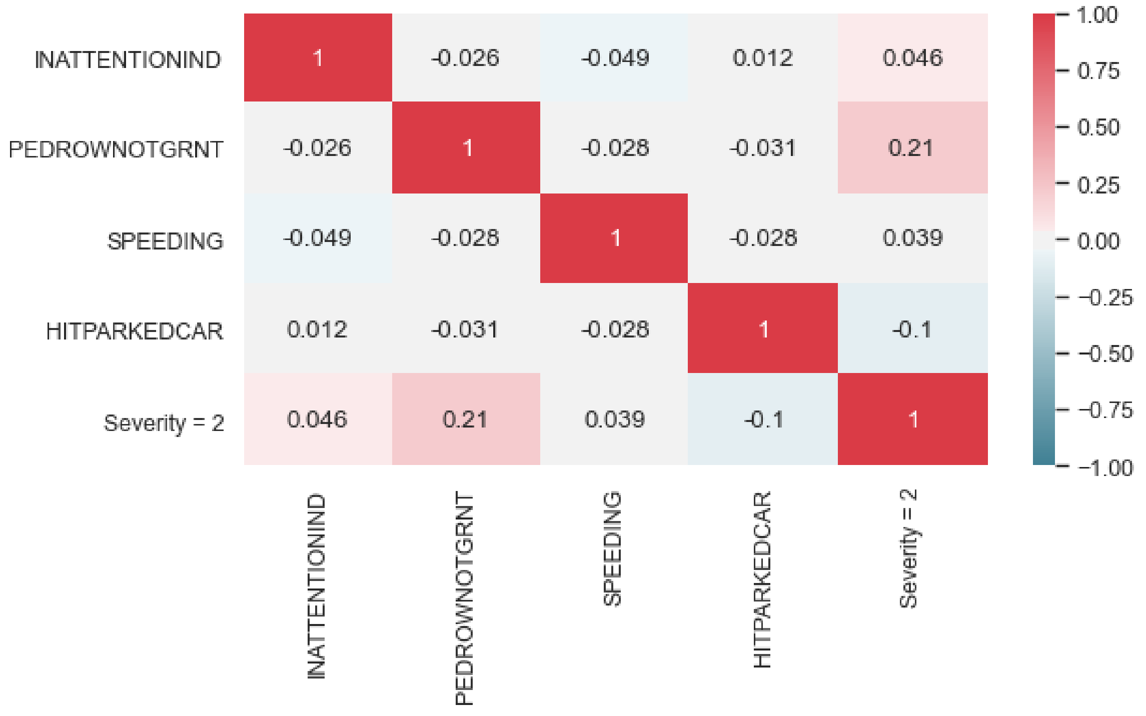
Out[34]:

	SEVERITYCODE	INATTENTIONIND	UNDERINFL	PEDROWNOTGRNT	SPEEDING	HITPARK
0	2	0.0	0	0.0	0.0	
1	1	0.0	0	0.0	0.0	
2	1	0.0	0	0.0	0.0	
3	1	0.0	0	0.0	0.0	
4	2	0.0	0	0.0	0.0	

We now look at the correlation matrix for these driver/accident conditions to see whether there's any strong correlation or not.

In [35]:

```
cond_df_driver_corr = cond_df_driver.drop(columns = ['SEVERITYCODE','Severity = 1']).co
rr()
fig = plt.figure(num=None, figsize=(8, 4), dpi=80, facecolor='w', edgecolor='k')
sns.set(font_scale=1)
cmap = sns.diverging_palette(220, 10, as_cmap=True)
sns.heatmap(cond_df_driver_corr, cmap=cmap,annot=True, vmin=-1, vmax=1);
```



From this correlation matrix we can see that there's no strong 1 to 1 correlation with the output. We continue the analyses by looking at weather, road and light conditions.

In [36]:

```
cond_df['Severity = 1']= cond_df['SEVERITYCODE'].apply(lambda x: 1 if (x<2)  else 0)
cond_df['Severity = 2']= cond_df['SEVERITYCODE'].apply(lambda x: 1 if (x>=2)  else 0)
cond_df.head()
```

Out[36]:

	SEVERITYCODE	INATTENTIONIND	UNDERINFL	PEDROWNOTGRNT	SPEEDING	HITPARK
0	2	NaN	N	NaN	NaN	
1	1	NaN	0	NaN	NaN	
2	1	NaN	0	NaN	NaN	
3	1	NaN	N	NaN	NaN	
4	2	NaN	0	NaN	NaN	

In [37]:

```
cond_df.groupby('WEATHER')['Severity = 1','Severity = 2'].sum()
```

Out[37]:

	Severity = 1	Severity = 2
WEATHER		
Blowing Sand/Dirt	41	15
Clear	75295	35840
Fog/Smog/Smoke	382	187
Other	716	116
Overcast	18969	8745
Partly Cloudy	2	3
Raining	21969	11176
Severe Crosswind	18	7
Sleet/Hail/Freezing Rain	85	28
Snowing	736	171
Unknown	14275	816

In [38]:

```
cond_df.groupby('ROADCOND')['Severity = 1','Severity = 2'].sum()
```

Out[38]:

	Severity = 1	Severity = 2
ROADCOND		
Dry	84446	40064
Ice	936	273
Oil	40	24
Other	89	43
Sand/Mud/Dirt	52	23
Snow/Slush	837	167
Standing Water	85	30
Unknown	14329	749
Wet	31719	15755

In [39]:

```
cond_df.groupby('LIGHTCOND')['Severity = 1','Severity = 2'].sum()
```

Out[39]:

LIGHTCOND	Severity = 1	Severity = 2
Dark - No Street Lights	1203	334
Dark - Street Lights Off	883	316
Dark - Street Lights On	34032	14475
Dark - Unknown Lighting	7	4
Dawn	1678	824
Daylight	77593	38544
Dusk	3958	1944
Other	183	52
Unknown	12868	605

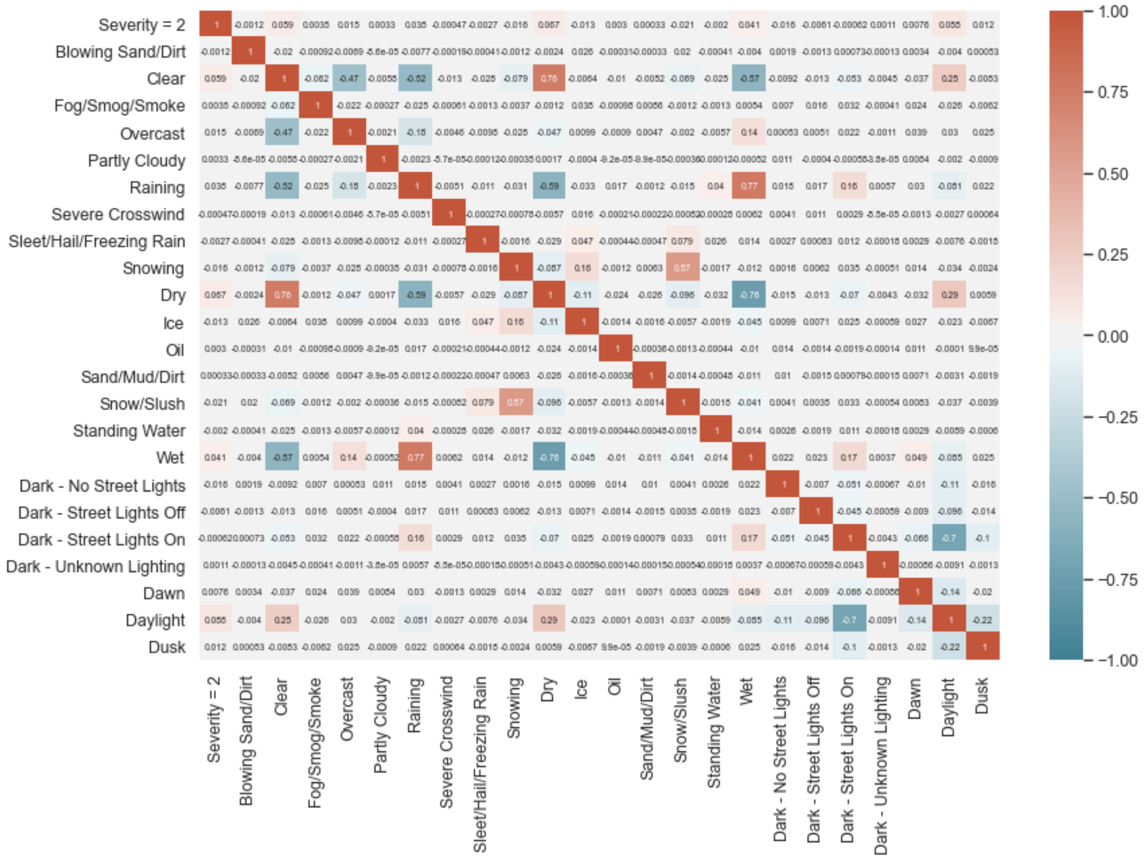
From these 3 groupby we can see:

- most of the collisions of severity 1 and severity 2 occurred with clear weather followed by rain
- most of the collisions of severity 1 and severity 2 occurred on dry roads followed by wet roads
- most of the collisions of severity 1 and 2 occurred in daylight, followed by dark - street lights on

We can now plot a correlation matrix to observe whether there's a strong 1 to 1 correlation with the accident severity. To do, we exclude "Severity = 1" label from the dataset (otherwise it would show a strong negative correlation with "Severity = 2").

In [40]:

```
weather_df = pd.get_dummies(cond_df.WEATHER).drop(columns = 'Unknown')
roadcond_df = pd.get_dummies(cond_df.ROADCOND).drop(columns = 'Unknown')
light_df = pd.get_dummies(cond_df.LIGHTCOND).drop(columns = 'Unknown')
cond_df_environment = pd.concat([cond_df, weather_df, roadcond_df, light_df], axis=1).d
rop(columns=['SEVERITYCODE','WEATHER','ROADCOND','LIGHTCOND','Other', 'Severity = 1'])
fig = plt.figure(num=None, figsize=(12, 8), dpi=80, facecolor='w', edgecolor='k')
sns.set(font_scale=1)
cmap = sns.diverging_palette(220, 20, as_cmap=True)
sns.heatmap(cond_df_environment.corr(), cmap=cmap,annot=True, annot_kws={"size": 6}, vm
in=-1, vmax=1);
```



Even in this case we can see that there isn't a strong 1 to 1 correlation with any of these features.

3.b Data cleaning and preparation for ML

Analysing these features has been very useful in understanding how to clean up the data and organise the dataframe to proceed with the implementation of a classification algorithm (such as decision tree or logistic regression). We have identified and built several dataframes:

- time_df with information on the time and date of accidents;
- colltype_df with information on the type of collision;
- cond_df_driver with information on the condition of the driver and other non-environmental factors;
- cond_df_environment with information on the environmental conditions of the accidents.

We can use these dataframes, combined, in a decision tree algorithm to predict the probability of future accidents given certain conditions.

In [41]:

```
time_clean = time_df.drop(columns = ['SEVERITYCODE','INCDTTM','Time','Date','Month','Da
y','Minutes','Seconds'])
time_clean = time_clean [['Severity = 1','Severity = 2','Year','Hours']]
time_clean.head()
```

Out[41]:

	Severity = 1	Severity = 2	Year	Hours
0	0	1	2013	14
1	1	0	2006	18
2	1	0	2004	10
3	1	0	2013	9
4	0	1	2004	8

In [42]:

```
colltype_clean = colltype_df.drop(columns = ['SEVERITYCODE','Severity = 1','Severity =
 2','Tot_collisions'])
colltype_clean.head()
```

Out[42]:

	COLLISIONTYPE	PERSONCOUNT	PEDCOUNT	PEDCYLCOUNT	VEHCOUNT
0	Angles	2	0	0	2
1	Sideswipe	2	0	0	2
2	Parked Car	4	0	0	3
3	Other	3	0	0	3
4	Angles	2	0	0	2

In [43]:

```
cond_driver_clean = cond_df_driver.drop(columns = ['SEVERITYCODE','Severity = 1','Sever
ity = 2'])
cond_driver_clean.head()
```

Out[43]:

	INATTENTIONIND	UNDERINFL	PEDROWNOTGRNT	SPEEDING	HITPARKEDCAR
0	0.0	0	0.0	0.0	0
1	0.0	0	0.0	0.0	0
2	0.0	0	0.0	0.0	0
3	0.0	0	0.0	0.0	0
4	0.0	0	0.0	0.0	0

In [44]:

```
cond_env_clean = cond_df_environment.drop(columns = ['INATTENTIONIND','UNDERINFL','PEDR
OWNOTGRNT','SPEEDING','HITPARKEDCAR','Severity = 2'])
cond_env_clean.head()
```

Out[44]:

	Blowing Sand/Dirt	Clear	Fog/Smog/Smoke	Overcast	Partly Cloudy	Raining	Severe Crosswind	Sleet/Hail/Freez F
0	0	0	0	1	0	0	0	
1	0	0	0	0	0	1	0	
2	0	0	0	1	0	0	0	
3	0	1	0	0	0	0	0	
4	0	0	0	0	0	1	0	

5 rows × 23 columns

In [45]:

```
# We can stitch these 4 dataframes together and check how to clean the data
coll_clean = pd.concat([time_clean, pd.get_dummies(colltype_clean.COLLISIONTYPE), cond_
driver_clean, cond_env_clean], axis=1)
coll_clean.head()
```

Out[45]:

	Severity = 1	Severity = 2	Year	Hours	Angles	Cycles	Head On	Left Turn	Other	Parked Car	...	Snow/Sl
0	0	1	2013	14	1	0	0	0	0	0	...	
1	1	0	2006	18	0	0	0	0	0	0	...	
2	1	0	2004	10	0	0	0	0	0	1	...	
3	1	0	2013	9	0	0	0	0	1	0	...	
4	0	1	2004	8	1	0	0	0	0	0	...	

5 rows × 42 columns

Now we see how many NaN values there are in this dataframe for each column.

In [46]:

```
for c in coll_clean.columns:
    print ("---- %s ---" % c)
    print ("is nan:", coll_clean[c].isnull().sum())
```

```
---- Severity = 1 ---
is nan: 0
---- Severity = 2 ---
is nan: 0
---- Year ---
is nan: 0
---- Hours ---
is nan: 0
---- Angles ---
is nan: 0
---- Cycles ---
is nan: 0
---- Head On ---
is nan: 0
---- Left Turn ---
is nan: 0
---- Other ---
is nan: 0
---- Parked Car ---
is nan: 0
---- Pedestrian ---
is nan: 0
---- Rear Ended ---
is nan: 0
---- Right Turn ---
is nan: 0
---- Sideswipe ---
is nan: 0
---- INATTENTIONIND ---
is nan: 0
---- UNDERINFL ---
is nan: 4884
---- PEDROWNOTGRNT ---
is nan: 0
---- SPEEDING ---
is nan: 0
---- HITPARKEDCAR ---
is nan: 0
---- Blowing Sand/Dirt ---
is nan: 0
---- Clear ---
is nan: 0
---- Fog/Smog/Smoke ---
is nan: 0
---- Overcast ---
is nan: 0
---- Partly Cloudy ---
is nan: 0
---- Raining ---
is nan: 0
---- Severe Crosswind ---
is nan: 0
---- Sleet/Hail/Freezing Rain ---
is nan: 0
---- Snowing ---
is nan: 0
---- Dry ---
is nan: 0
---- Ice ---
is nan: 0
---- Oil ---
```

```
is nan: 0
---- Sand/Mud/Dirt ---
is nan: 0
---- Snow/Slush ---
is nan: 0
---- Standing Water ---
is nan: 0
---- Wet ---
is nan: 0
---- Dark - No Street Lights ---
is nan: 0
---- Dark - Street Lights Off ---
is nan: 0
---- Dark - Street Lights On ---
is nan: 0
---- Dark - Unknown Lighting ---
is nan: 0
---- Dawn ---
is nan: 0
---- Daylight ---
is nan: 0
---- Dusk ---
is nan: 0
```

We notice there are 4884 NaN for UNDERINFL feature. We get rid of these NaN and then check the dataframe again.

In [47]:

```
coll_clean = coll_clean.dropna(axis=0)
for c in coll_clean.columns:
    print ("---- %s ---" % c)
    print ("is nan:", coll_clean[c].isnull().sum())
```

```
---- Severity = 1 ---
is nan: 0
---- Severity = 2 ---
is nan: 0
---- Year ---
is nan: 0
---- Hours ---
is nan: 0
---- Angles ---
is nan: 0
---- Cycles ---
is nan: 0
---- Head On ---
is nan: 0
---- Left Turn ---
is nan: 0
---- Other ---
is nan: 0
---- Parked Car ---
is nan: 0
---- Pedestrian ---
is nan: 0
---- Rear Ended ---
is nan: 0
---- Right Turn ---
is nan: 0
---- Sideswipe ---
is nan: 0
---- INATTENTIONIND ---
is nan: 0
---- UNDERINFL ---
is nan: 0
---- PEDROWNOTGRNT ---
is nan: 0
---- SPEEDING ---
is nan: 0
---- HITPARKEDCAR ---
is nan: 0
---- Blowing Sand/Dirt ---
is nan: 0
---- Clear ---
is nan: 0
---- Fog/Smog/Smoke ---
is nan: 0
---- Overcast ---
is nan: 0
---- Partly Cloudy ---
is nan: 0
---- Raining ---
is nan: 0
---- Severe Crosswind ---
is nan: 0
---- Sleet/Hail/Freezing Rain ---
is nan: 0
---- Snowing ---
is nan: 0
---- Dry ---
is nan: 0
---- Ice ---
is nan: 0
---- Oil ---
```

```
is nan: 0
---- Sand/Mud/Dirt ---
is nan: 0
---- Snow/Slush ---
is nan: 0
---- Standing Water ---
is nan: 0
---- Wet ---
is nan: 0
---- Dark - No Street Lights ---
is nan: 0
---- Dark - Street Lights Off ---
is nan: 0
---- Dark - Street Lights On ---
is nan: 0
---- Dark - Unknown Lighting ---
is nan: 0
---- Dawn ---
is nan: 0
---- Daylight ---
is nan: 0
---- Dusk ---
is nan: 0
```

There are now 0 NaN in the whole dataframe. We now turn our attention to the Severity 1 and Severity 2 feature to check if the dataframe is imbalanced or not.

In [48]:

```
print("Sum of Severity 1 accidents: ", coll_clean['Severity = 1'].sum())
print("Sum of Severity 2 accidents: ", coll_clean['Severity = 2'].sum())
perc_sev1 = coll_clean['Severity = 1'].sum() / ((coll_clean['Severity = 1'].sum()) + (c
oll_clean['Severity = 2'].sum()))
print("Percentages of Severity 1 wrt to total number of accidents: ", perc_sev1*100, "
 %")
print("Percentages of Severity 2 wrt to total number of accidents: ", (1-perc_sev1)*100
, " %")
```

```
Sum of Severity 1 accidents:  132630
Sum of Severity 2 accidents:  57159
Percentages of Severity 1 wrt to total number of accidents:  69.8828699239
682  %
Percentages of Severity 2 wrt to total number of accidents:  30.1171300760
31806  %
```

We see that the dataframe is actually imbalanced and so we perform an **undersampling** following the instructions here (https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets#t1).

In [49]:

```
coll_clean['Severity = 2'].value_counts()
```

Out[49]:

```
0    132630
1     57159
Name: Severity = 2, dtype: int64
```

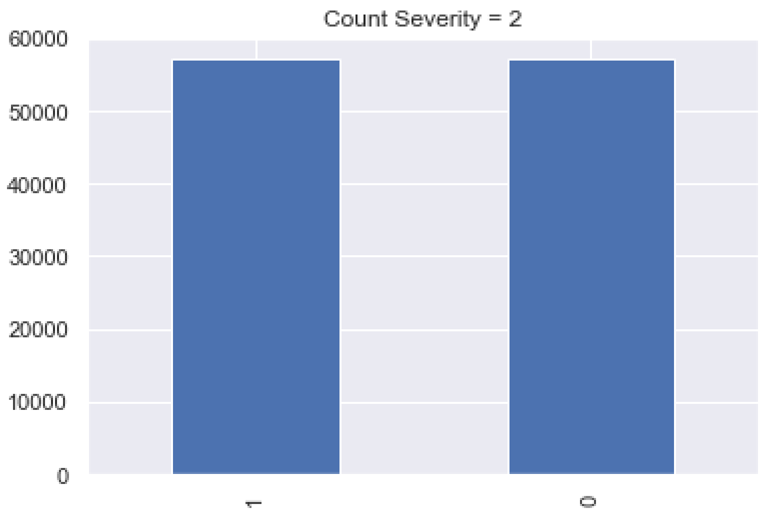
In [50]:

```
# Class count
count_class_0, count_class_1 = coll_clean['Severity = 2'].value_counts()

# Divide by class
df_class_0 = coll_clean[coll_clean['Severity = 2'] == 0]
df_class_1 = coll_clean[coll_clean['Severity = 2'] == 1]
```

In [51]:

```
df_class_0_under = df_class_0.sample(count_class_1)
coll_clean_under = pd.concat([df_class_0_under, df_class_1], axis=0)

print('Random under-sampling:')
print(coll_clean_under['Severity = 2'].value_counts())

coll_clean_under['Severity = 2'].value_counts().plot(kind='bar', title='Count ''Severit
y = 2''');
```

```
Random under-sampling:
1    57159
0    57159
Name: Severity = 2, dtype: int64
```



In [52]:

```
coll_clean_under['Severity = 2'].value_counts()
```

Out[52]:

```
1    57159
0    57159
Name: Severity = 2, dtype: int64
```

Now that we have a balanced dataframe, we can now drop the column "Severity = 1" as we will only use "Severity = 2" as label.

In [53]:

```
coll_clean_under = coll_clean_under.reset_index().drop(columns = 'index').drop(columns
='Severity = 1')
coll_clean_under.shape
coll_clean_under.head()
```

Out[53]:

	Severity = 2	Year	Hours	Angles	Cycles	Head On	Left Turn	Other	Parked Car	Pedestrian	...	Snow
0	0	2006	15	0	0	0	0	0	0	0	...	
1	0	2013	16	0	0	0	0	0	1	0	...	
2	0	2011	23	0	0	0	0	0	0	0	...	
3	0	2014	17	0	0	0	0	0	0	0	...	
4	0	2020	17	0	0	0	0	0	0	0	...	

5 rows × 41 columns

In [54]:

```
## Getting rid of Year and Hours for now
```

In [55]:

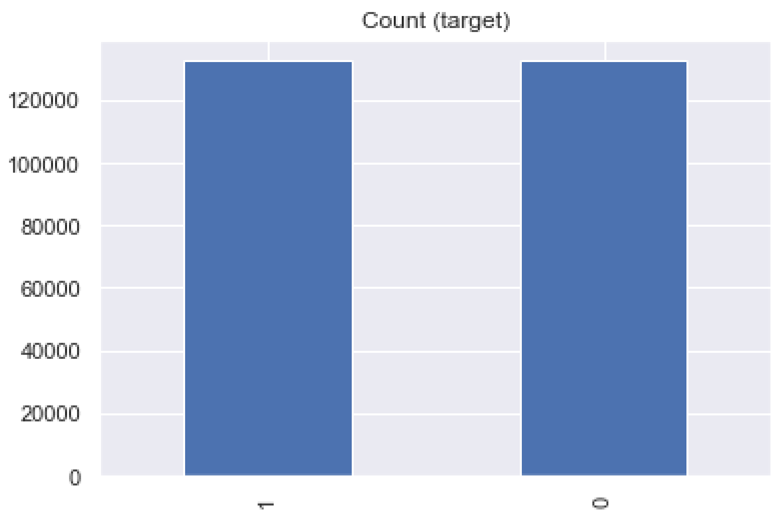
```
coll_clean_under = coll_clean_under.drop(columns = ['Year','Hours'])
```

We now perform an **over-sampling** to use both sampling techniques and compare them when using the machine learning algorithms.

In [56]:

```
df_class_1_over = df_class_1.sample(count_class_0, replace=True)
coll_clean_over = pd.concat([df_class_0, df_class_1_over], axis=0)

print('Random over-sampling:')
print(coll_clean_over['Severity = 2'].value_counts())

coll_clean_over['Severity = 2'].value_counts().plot(kind='bar', title='Count (target)'
);
```

```
Random over-sampling:
1    132630
0    132630
Name: Severity = 2, dtype: int64
```



In [57]:

```
coll_clean_over = coll_clean_over.reset_index().drop(columns = 'index').drop(columns =
'Severity = 1')
coll_clean_over = coll_clean_over.drop(columns = ['Year','Hours'])
coll_clean_over.shape
coll_clean_over.head()
```

Out[57]:

	Severity = 2	Angles	Cycles	Head On	Left Turn	Other	Parked Car	Pedestrian	Rear Ended	Right Turn	...	Snov
0	0	0	0	0	0	0	0	0	0	0	...	
1	0	0	0	0	0	0	1	0	0	0	...	
2	0	0	0	0	0	1	0	0	0	0	...	
3	0	1	0	0	0	0	0	0	0	0	...	
4	0	1	0	0	0	0	0	0	0	0	...	

5 rows × 39 columns

3.c Application of classification algorithms

In our dataframe *coll_clean_under*, we now have identified 40 features ranging from environmental condition to time of the day and we're now trying to build a model that is capable of classifying accidents based on these features. We can try to employ different models like random forest classifier, decision tree and logistic regression.

3.c.1 Decision Tree, Random Forest and Logistic Regression

First thing is identifying label and features and then importing the relevant libraries.

In [58]:

```
X_under = coll_clean_under.loc[:, coll_clean_under.columns != 'Severity = 2']
y_under = coll_clean_under['Severity = 2']
```

In [59]:

```
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import preprocessing
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
```

In [61]:

```
### Creating function to plot confusion matrix
# Compute confusion matrix
from sklearn.metrics import classification_report, confusion_matrix
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

We now apply **Decision Tree, Random Forest** and **Logistic Regression** to both under- and over-sampled dataframes. For each application of ML algorithms we also look at their confusion matrix which gives an indication of true positives, true negatives, false positives and false negatives.

In [62]:

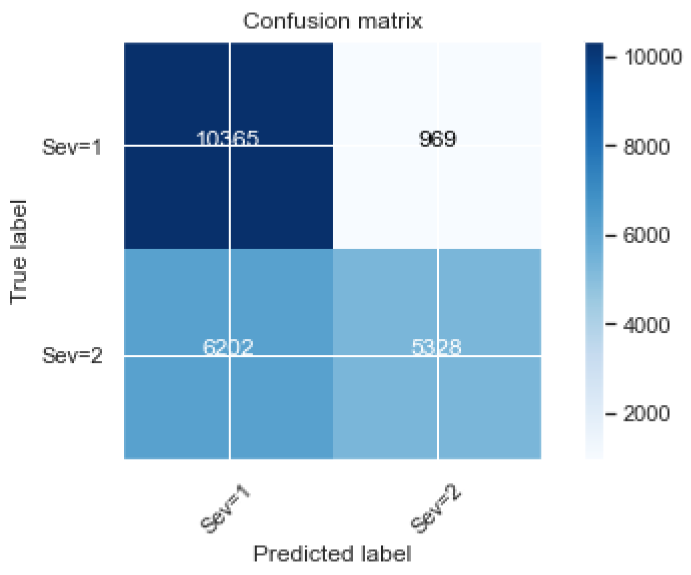
```
X_trainset_under, X_testset_under, y_trainset_under, y_testset_under = train_test_split
(X_under, y_under, test_size=0.2, random_state=3)
tree_under = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
tree_under.fit(X_trainset_under,y_trainset_under)
predTree_under = tree_under.predict(X_testset_under)
print("DT's Metrics: ")
print(metrics.classification_report(y_testset_under, predTree_under))
```

```
DT's Metrics: 
              precision    recall  f1-score   support

           0       0.85      0.46      0.60     11530
           1       0.63      0.91      0.74     11334

    accuracy                           0.69     22864
   macro avg       0.74      0.69      0.67     22864
weighted avg       0.74      0.69      0.67     22864
```

In [63]:

```
cnf_matrix_dt_under = confusion_matrix(y_testset_under, predTree_under, labels=[1,0])
np.set_printoptions(precision=2)
# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix_dt_under, classes=['Sev=1','Sev=2'],normalize= False,
title='Confusion matrix')
```

```
Confusion matrix, without normalization
[[10365   969]
 [ 6202  5328]]
```



In [65]:

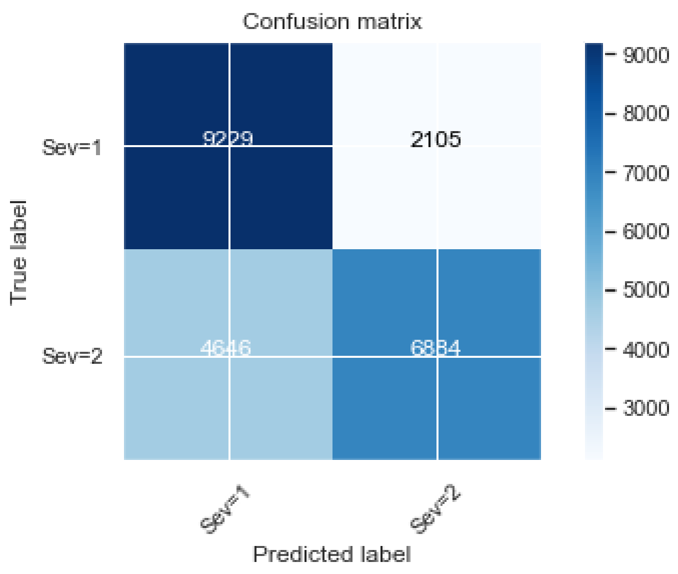
```
from sklearn.ensemble import RandomForestClassifier
rfc_under = RandomForestClassifier(n_estimators=500)
rfc_under.fit(X_trainset_under , y_trainset_under)
rfcpred_under = rfc_under.predict(X_testset_under)
print("RFC's Metrics: ")
print(metrics.classification_report(y_testset_under, rfcpred_under))
```

```
RFC's Metrics: 
              precision    recall  f1-score   support

           0       0.77      0.60      0.67     11530
           1       0.67      0.81      0.73     11334

    accuracy                           0.70     22864
   macro avg       0.72      0.71      0.70     22864
weighted avg       0.72      0.70      0.70     22864
```

In [66]:

```
cnf_matrix_rfc_under = confusion_matrix(y_testset_under, rfcpred_under, labels=[1,0])
np.set_printoptions(precision=2)
# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix_rfc_under, classes=['Sev=1','Sev=2'],normalize= False,
title='Confusion matrix')
```

```
Confusion matrix, without normalization
[[9229 2105]
 [4646 6884]]
```



In [67]:

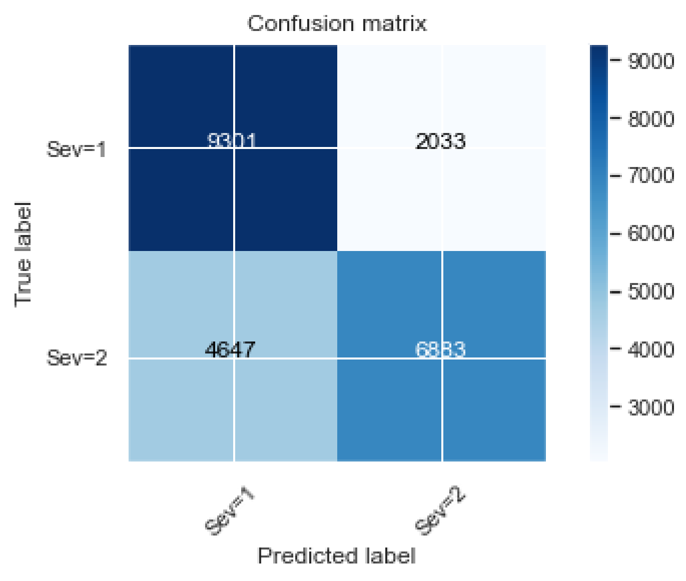
```
LR_under=LogisticRegression()
LR_under.fit(X_trainset_under , y_trainset_under)
LRpredict_under = LR_under.predict(X_testset_under)
print("LR's Metrics: ")
print(metrics.classification_report(y_testset_under, LRpredict_under))
```

```
LR's Metrics:
              precision    recall  f1-score   support

           0       0.77      0.60      0.67     11530
           1       0.67      0.82      0.74     11334

    accuracy                           0.71     22864
   macro avg       0.72      0.71      0.70     22864
weighted avg       0.72      0.71      0.70     22864
```

In [68]:

```
cnf_matrix_lr_under = confusion_matrix(y_testset_under, LRpredict_under, labels=[1,0])
np.set_printoptions(precision=2)
# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix_lr_under, classes=['Sev=1','Sev=2'],normalize= False,
title='Confusion matrix')
```

```
Confusion matrix, without normalization
[[9301 2033]
 [4647 6883]]
```



Now looking to apply the same methologies to the over-sampled dataframe

In [69]:

```
X_over = coll_clean_over.loc[:, coll_clean_over.columns != 'Severity = 2']
y_over = coll_clean_over['Severity = 2']
X_over.head()
```

Out[69]:

	Angles	Cycles	Head On	Left Turn	Other	Parked Car	Pedestrian	Rear Ended	Right Turn	Sideswipe	...	Sn
0	0	0	0	0	0	0	0	0	0	1	...	
1	0	0	0	0	0	1	0	0	0	0	...	
2	0	0	0	0	1	0	0	0	0	0	...	
3	1	0	0	0	0	0	0	0	0	0	...	
4	1	0	0	0	0	0	0	0	0	0	...	

5 rows × 38 columns

In [70]:

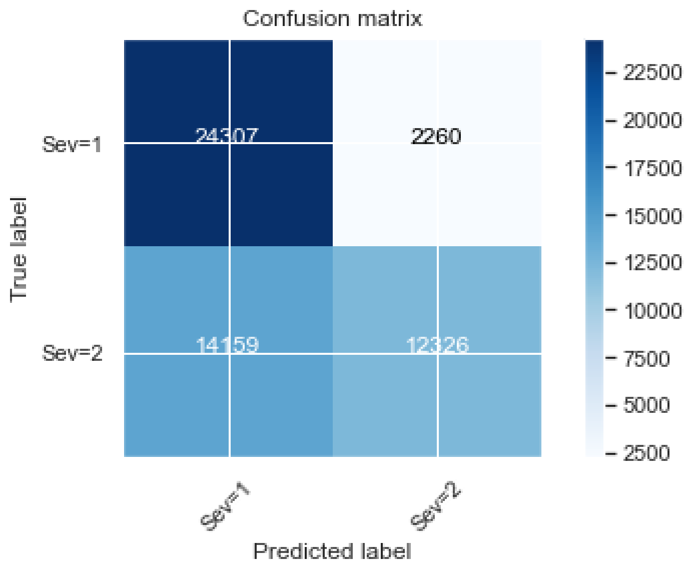
```
X_trainset_over, X_testset_over, y_trainset_over, y_testset_over = train_test_split(X_o
ver, y_over, test_size=0.2, random_state=3)
tree_over = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
tree_over.fit(X_trainset_over,y_trainset_over)
predTree_over = tree_over.predict(X_testset_over)
print("DT's Metrics: ")
print(metrics.classification_report(y_testset_over, predTree_over))
```

```
DT's Metrics: 
              precision    recall  f1-score   support

           0       0.85      0.47      0.60     26485
           1       0.63      0.91      0.75     26567

    accuracy                           0.69     53052
   macro avg       0.74      0.69      0.67     53052
weighted avg       0.74      0.69      0.67     53052
```

In [71]:

```
cnf_matrix_dt_over = confusion_matrix(y_testset_over, predTree_over, labels=[1,0])
np.set_printoptions(precision=2)
# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix_dt_over, classes=['Sev=1','Sev=2'],normalize= False,
title='Confusion matrix')
```

```
Confusion matrix, without normalization
[[24307  2260]
 [14159 12326]]
```



In [72]:

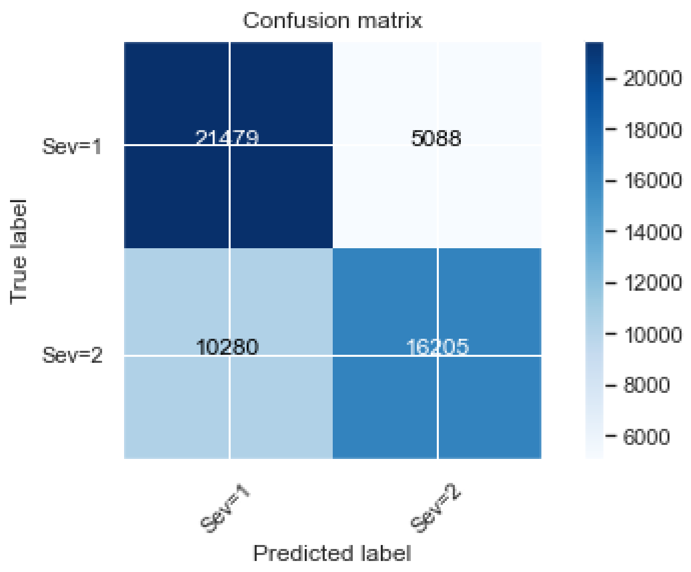
```
rfc_over = RandomForestClassifier(n_estimators=500)
rfc_over.fit(X_trainset_over , y_trainset_over)
rfcpred_over = rfc_over.predict(X_testset_over)
print("RFC's Metrics: ")
print(metrics.classification_report(y_testset_over, rfcpred_over))
```

```
RFC's Metrics:
              precision    recall  f1-score   support

           0       0.76      0.61      0.68     26485
           1       0.68      0.81      0.74     26567

    accuracy                           0.71     53052
   macro avg       0.72      0.71      0.71     53052
weighted avg       0.72      0.71      0.71     53052
```

In [73]:

```
cnf_matrix_rfc_over = confusion_matrix(y_testset_over, rfcpred_over, labels=[1,0])
np.set_printoptions(precision=2)
# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix_rfc_over, classes=['Sev=1','Sev=2'],normalize= False,
title='Confusion matrix')
```

```
Confusion matrix, without normalization
[[21479  5088]
 [10280 16205]]
```



In [74]:

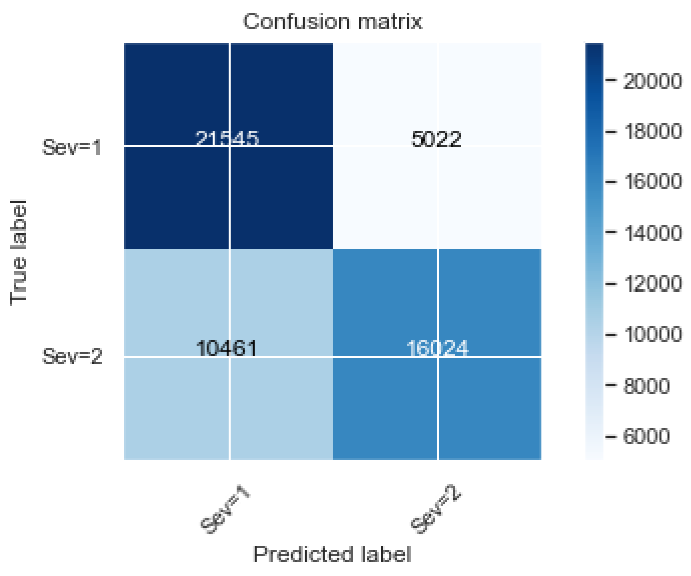
```
LR_over=LogisticRegression()
LR_over.fit(X_trainset_over , y_trainset_over)
LRpredict_over = LR_over.predict(X_testset_over)
print("LR's Metrics: ")
print(metrics.classification_report(y_testset_over, LRpredict_over))
```

```
LR's Metrics: 
              precision    recall  f1-score   support

           0       0.76      0.61      0.67     26485
           1       0.67      0.81      0.74     26567

    accuracy                           0.71     53052
   macro avg       0.72      0.71      0.70     53052
weighted avg       0.72      0.71      0.71     53052
```

In [75]:

```
cnf_matrix_lr_over = confusion_matrix(y_testset_over, LRpredict_over, labels=[1,0])
np.set_printoptions(precision=2)
# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix_lr_over, classes=['Sev=1','Sev=2'],normalize= False,
title='Confusion matrix')
```

```
Confusion matrix, without normalization
[[21545  5022]
 [10461 16024]]
```



3.c.2 Gridsearchcv

Now we want to try and improve the predictions and this can be done by introducing **gridsearchcv** which looks for the best set of parameters off a parameter grid defined by the user.

Decision Tree

In [149]:

```
param_grid_dt = {'max_leaf_nodes': list(range(2, 100)), 'min_samples_split': [2, 3, 4],
'criterion': ['gini', 'entropy']}

dt_gs_under = GridSearchCV(DecisionTreeClassifier(random_state=42), param_grid_dt, verb
ose=1, cv=10, scoring = 'accuracy')
dt_gs_under.fit(X_trainset_under,y_trainset_under)
dtpredict_gs_under = dt_gs_under.predict(X_testset_under)
print("DT GS's best params: ")
print(dt_gs_under.best_params_)
print("DT GS's Metrics: ")
print(metrics.classification_report(y_testset_under, dtpredict_gs_under))
```

```
Fitting 10 folds for each of 588 candidates, totalling 5880 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wo
rkers.
[Parallel(n_jobs=1)]: Done 5880 out of 5880 | elapsed: 45.4min finished

DT GS's best params:
{'criterion': 'gini', 'max_leaf_nodes': 98, 'min_samples_split': 2}
DT GS's Metrics:
              precision    recall  f1-score   support

           0       0.77      0.61      0.68     11530
           1       0.67      0.81      0.73     11334

    accuracy                           0.71     22864
   macro avg       0.72      0.71      0.71     22864
weighted avg       0.72      0.71      0.71     22864
```

In [150]:

```
dt_gs_over = GridSearchCV(DecisionTreeClassifier(random_state=42), param_grid_dt, verbo
se=1, cv=10, scoring = 'accuracy')
dt_gs_over.fit(X_trainset_over,y_trainset_over)
dtpredict_gs_over = dt_gs_over.predict(X_testset_over)
print("DT GS's best params: ")
print(dt_gs_over.best_params_)
print("DT GS's Metrics: ")
print(metrics.classification_report(y_testset_over, dtpredict_gs_over))
```

```
Fitting 10 folds for each of 588 candidates, totalling 5880 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wo
rkers.
[Parallel(n_jobs=1)]: Done 5880 out of 5880 | elapsed: 158.8min finished

DT GS's best params:
{'criterion': 'gini', 'max_leaf_nodes': 99, 'min_samples_split': 2}
DT GS's Metrics:
              precision    recall  f1-score   support

           0       0.76      0.60      0.67     26485
           1       0.67      0.81      0.73     26567

    accuracy                           0.71     53052
   macro avg       0.72      0.71      0.70     53052
weighted avg       0.72      0.71      0.70     53052
```

Random Forest Classifier

In [151]:

```
param_grid_rfc = {'n_estimators': [200, 500, 1000],'max_features': ['auto','log2'],'max
_depth' : [4,6,8],'criterion' :['gini', 'entropy']}

rfc_gs_under=RandomForestClassifier()
rfc_gs_under=GridSearchCV(rfc_gs_under,param_grid_rfc,verbose=1, cv=5,scoring = 'accura
cy')
rfc_gs_under.fit(X_trainset_under,y_trainset_under)
rfcpredict_gs_under = rfc_gs_under.predict(X_testset_under)
print("RFC GS's best params: ")
print(rfc_gs_under.best_params_)
print("RFC GS's Metrics: ")
print(metrics.classification_report(y_testset_under, rfcpredict_gs_under))
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wo
rkers.
[Parallel(n_jobs=1)]: Done 180 out of 180 | elapsed: 84.6min finished

RFC GS's best params:
{'criterion': 'gini', 'max_depth': 8, 'max_features': 'log2', 'n_estimator
s': 200}
RFC GS's Metrics:
              precision    recall  f1-score   support

           0       0.77      0.61      0.68     11530
           1       0.67      0.82      0.74     11334

    accuracy                           0.71     22864
   macro avg       0.72      0.71      0.71     22864
weighted avg       0.72      0.71      0.71     22864
```

In [152]:

```
rfc_gs_over=RandomForestClassifier()
rfc_gs_over=GridSearchCV(rfc_gs_over,param_grid_rfc,verbose=1, cv=5,scoring = 'accurac
y')
rfc_gs_over.fit(X_trainset_over,y_trainset_over)
rfcpredict_gs_over = rfc_gs_over.predict(X_testset_over)
print("RFC GS's best params: ")
print(rfc_gs_over.best_params_)
print("RFC GS's Metrics: ")
print(metrics.classification_report(y_testset_over, rfcpredict_gs_over))
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wo
rkers.
[Parallel(n_jobs=1)]: Done 180 out of 180 | elapsed: 121.9min finished

RFC GS's best params:
{'criterion': 'entropy', 'max_depth': 8, 'max_features': 'log2', 'n_estima
tors': 500}
RFC GS's Metrics:
              precision    recall  f1-score   support

           0       0.76      0.60      0.67     26485
           1       0.67      0.81      0.73     26567

    accuracy                           0.71     53052
   macro avg       0.72      0.71      0.70     53052
weighted avg       0.72      0.71      0.70     53052
```

Logistic regression

In [153]:

```
param_grid_LR = {'penalty' : ['l1', 'l2'],'C' : np.logspace(-10, 4, 20),'solver' : ['li
blinear'] }

LR_gs_under=LogisticRegression()
LR_gs_under=GridSearchCV(LR_gs_under,param_grid_LR,cv=5,verbose=1, scoring = 'accuracy'
)
LR_gs_under.fit(X_trainset_under,y_trainset_under)
LRpredict_gs_under = LR_gs_under.predict(X_testset_under)
print("LR GS's best params: ")
print(LR_gs_under.best_params_)
print("LR GS's Metrics: ")
print(metrics.classification_report(y_testset_under, LRpredict_gs_under))
```

```
Fitting 5 folds for each of 40 candidates, totalling 200 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wo
rkers.
[Parallel(n_jobs=1)]: Done 200 out of 200 | elapsed:  4.2min finished

LR GS's best params:
{'C': 61.58482110660255, 'penalty': 'l1', 'solver': 'liblinear'}
LR GS's Metrics:
              precision    recall  f1-score   support

           0       0.77      0.60      0.68     11530
           1       0.67      0.82      0.74     11334

    accuracy                           0.71     22864
   macro avg       0.72      0.71      0.71     22864
weighted avg       0.72      0.71      0.71     22864
```

In [154]:

```
LR_gs_over=LogisticRegression()
LR_gs_over=GridSearchCV(LR_gs_over,param_grid_LR,cv=5,verbose=1, scoring = 'accuracy')
LR_gs_over.fit(X_trainset_over,y_trainset_over)
LRpredict_gs_over = LR_gs_over.predict(X_testset_over)
print("LR GS's best params: ")
print(LR_gs_over.best_params_)
print("LR GS's Metrics: ")
print(metrics.classification_report(y_testset_over, LRpredict_gs_over))
```

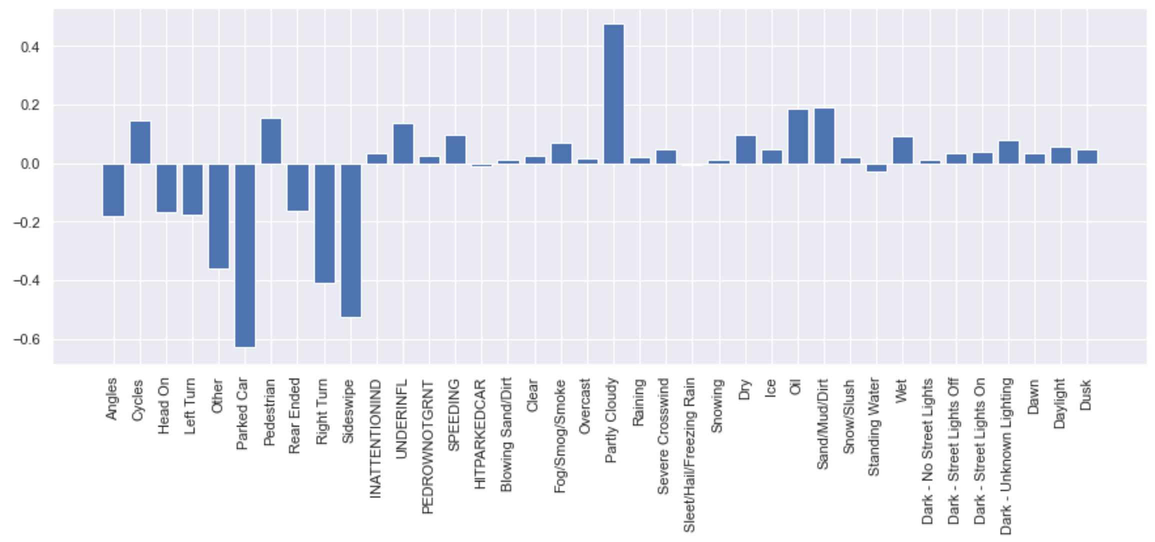
```
Fitting 5 folds for each of 40 candidates, totalling 200 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wo
rkers.
[Parallel(n_jobs=1)]: Done 200 out of 200 | elapsed:  6.8min finished

LR GS's best params:
{'C': 0.012742749857031322, 'penalty': 'l2', 'solver': 'liblinear'}
LR GS's Metrics:
              precision    recall  f1-score   support

           0       0.76      0.60      0.67     26485
           1       0.67      0.81      0.73     26567

    accuracy                           0.71     53052
   macro avg       0.72      0.71      0.70     53052
weighted avg       0.72      0.71      0.70     53052
```

Finally we run a simple regression model to understand features importance on the label.

In [98]:

```
from sklearn.linear_model import LinearRegression
from matplotlib import pyplot
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

model = LinearRegression()
# fit the model
model.fit(X_trainset_over, y_trainset_over)
ypred_linear = model.predict(X_testset_over)
# get importance
importance = model.coef_
# summarize feature importance
# The mean squared error
print('Mean squared error: %.2f'
      % mean_squared_error(y_testset_over, ypred_linear))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(y_testset_over, ypred_linear))
print('')
for i,v in zip(X_trainset_over.columns, importance):
    print('Feature: ', i ,' Score: %.5f' % (v))
# plot feature importance

plt.figure(figsize=(15,5))
plt.bar(X_trainset_over.columns,importance);
plt.xticks(X_trainset_over.columns, rotation='vertical');
```

```
Mean squared error: 0.19
Coefficient of determination: 0.26

Feature:  Angles  Score: -0.18111
Feature:  Cycles  Score: 0.14564
Feature:  Head On  Score: -0.16593
Feature:  Left Turn  Score: -0.17763
Feature:  Other  Score: -0.35987
Feature:  Parked Car  Score: -0.62869
Feature:  Pedestrian  Score: 0.15523
Feature:  Rear Ended  Score: -0.16068
Feature:  Right Turn  Score: -0.40695
Feature:  Sideswipe  Score: -0.52226
Feature:  INATTENTIONIND  Score: 0.03577
Feature:  UNDERINFL  Score: 0.13818
Feature:  PEDROWNOTGRNT  Score: 0.02712
Feature:  SPEEDING  Score: 0.09569
Feature:  HITPARKEDCAR  Score: -0.01173
Feature:  Blowing Sand/Dirt  Score: 0.01103
Feature:  Clear  Score: 0.02357
Feature:  Fog/Smog/Smoke  Score: 0.06869
Feature:  Overcast  Score: 0.01669
Feature:  Partly Cloudy  Score: 0.47595
Feature:  Raining  Score: 0.02045
Feature:  Severe Crosswind  Score: 0.04782
Feature:  Sleet/Hail/Freezing Rain  Score: -0.00736
Feature:  Snowing  Score: 0.01346
Feature:  Dry  Score: 0.09768
Feature:  Ice  Score: 0.04646
Feature:  Oil  Score: 0.18723
Feature:  Sand/Mud/Dirt  Score: 0.19161
Feature:  Snow/Slush  Score: 0.01911
Feature:  Standing Water  Score: -0.02728
Feature:  Wet  Score: 0.09068
Feature:  Dark - No Street Lights  Score: 0.01419
Feature:  Dark - Street Lights Off  Score: 0.03549
Feature:  Dark - Street Lights On  Score: 0.03694
Feature:  Dark - Unknown Lighting  Score: 0.08027
Feature:  Dawn  Score: 0.03262
Feature:  Daylight  Score: 0.05548
Feature:  Dusk  Score: 0.04918
```



4. Results and discussion

Looking at the initial exploratory data analysis, in order to expand it even further one could look at working day vs weekend incidence on number of Severity 1/2 accidents. Moreover, hotspots of accidents could be identified by using geographical data and also weather, road and light conditions could be grouped by type i.e. good/bad weather, clean/dirty road, light/no light.

The results given by the machine learning algorithms show that by doing under- or over-sampling doesn't seem to affect the results too much in terms of precision and recall. Moreover, we've also noticed how the results slightly improve by implementing a random tree forest algorithm which is one step forward with respect to the decision tree classification algorithm - random forests consist of multiple single trees each based on a random sample of the training data. They are typically more accurate than single decision trees. This can be clearly seen when observing the confusion matrices of decision tree vs random forest vs logistic regression. We have also run a linear regression analysis to understand the most importance factors influencing the label (prediction of Severity = 2 accidents) and we can see that the type of accident, along with cloudy weather and dirt road conditions are the most important features in predicting the label.

5. Conclusions

From this analyses, we've been able to draw the following conclusions:

1) Severity 1 accidents made up ~70% of total accidents per year between 2004 and 2020

2) From 2015 onwards, there has been a constant decline in total number of accidents probably due to the introduction of the Vision Zero programme in the city of Seattle.

3) 2 peaks of accidents were found during the day: one at 8am and one at 5pm - both representing peak hours.

4) Rear-ended type collisions seem to occur in stationary traffic because of the number of vehicle involved (>2)

5) Whenever a collision involved a bicycle, this was of severity 2 for majority of cases and the same can be said for accidents involving pedestrian

6) There doesn't seem to be a direct 1 to 1 correlation between weather and road conditions and severity/number of accidents.

7) There does some however to be some correlation with driver's conditions (under influence y/n, speeding y/n) and type of accident (sidesweep, rear-end, etc) in determining whether an accident is going to be of Severity 1 or 2 - although for this last one the prediction mean square error is quite big and hence some further investigation is required to make a robust conclusion.

Finally, from a global point of view it seems that the Vision Zero initiative undertaken by the city of Seattle is coming to fruition given the steady decline in number of accidents between 2015 and now. Even better conclusions and analyses could've been carried out if data were provided in a more concise way (i.e. grouping weather, road, light conditions or grouping them in a single category giving weights to each sub-parameter) and if data were 100% reliable (for example, why are so many accidents reported at 00:00:00?) but these analyses have shown what an initial exploration could lead to.