Road accidents' forecast in Belgium

I. Business Problem and Data Understanding

In this section we will firstly discuss the problem that we face, and secondly we will describe the data set that we used to help us solve it.

Purpose of the notebook

This notebook presents the results of the investigations on the probabilities of getting a severe road accident in Belgium. Severity is defined in terms of fatality and severe injuries. This notebook would thus be useful for anyone owning a car, since one should be able to see whether it is a good idea to take her car under specific circumstances. The model is going to warn people when they should be more careful than normal, meaning that they should drive slower for instance or try to find an alternative to the car.

Data set

The open data used for making the present analysis is gathered from the website of "Statbel", the Belgian statistical office. The data can be retrieve on the following link: https://statbel.fgov.be/en/open-data?category=162 (https://statbel.fgov.be/en/open-data?category=162). It contains 655 467 observation (i.e. road accidents) and 35 attributes (among others day of the week, road type, build up area, type of collision, light conditions, municipality and district, number of deadly accidents in the last 30 days etc.).

In [1]:

```
#Import of libraries
#------------------------------
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns

# Import of data set
#------------------------------
df = pd.read_csv(r'C:\Users\NJ5866\Desktop\Final project\Road_accidents_Bel
gium.csv')
df.head(10)
```

Out[1]:

	DT_DAY	DT_HOUR	CD_DAY_OF_WEEK	TX_DAY_OF_WEEK_DESCR_FR
0	27/09/2019	18	5	Vendredi
1	20/11/2019	12	3	Mercredi
2	15/07/2019	14	1	Lundi
3	20/04/2019	2	6	Samedi
4	25/10/2019	12	5	Vendredi
5	28/07/2019	21	7	Dimanche
6	18/10/2019	17	5	Vendredi
7	7/10/2019	13	1	Lundi
8	29/06/2019	14	6	Samedi
9	26/09/2019	6	4	Jeudi

10 rows × 35 columns

As one might notice, some variables are simply translations of others (French and Dutch), so in order to avoid using twice the same variables, we can simplify our data set by removing one of the two columns for each of such variables. Arbitrarily, French has been chosen to be kept. Let's check all variable which need to be removed (column names ending with '_NL'). As one can see, 9 variables will be withdrawn.

In [2]:

```
df_accidents = df.drop(["TX_DAY_OF_WEEK_DESCR_NL", "TX_BUILD_UP_AREA_DESCR_
NL", "TX_COLL_TYPE_DESCR_NL", "TX_LIGHT_COND_DESCR_NL", "TX_ROAD_TYPE_DESCR
_NL", "TX_MUNTY_DESCR_NL", "TX_ADM_DSTR_DESCR_NL", "TX_PROV_DESCR_NL", "TX_
RGN_DESCR_NL" ], axis=1)
df_accidents.shape
```

Out[2]:

```
(655468, 26)
```

The new data set contains now 26 columns (and the same number of lines, i.e. 655 467 observations).
A description of the variables is described, together with the data type needed, are described in the following table:

Variable names	Meaning	Type of data required
DT_DAY	Date of the accident	Date
DT_HOUR	Hour of the accident	Categorical (1-24)
CD_DAY_OF_WEEK	Day of week of the accident (1 to 7)	Categorical (1-7)
TX_DAY_OF_WEEK_DESCR_FR	Day of week of the accident in French	Categorical
CD_BUILD_UP_AREA	Code of buildup area: 1 = urban area, 2 = extra-urban	Categorical (1-2)
TX_BUILD_UP_AREA_DESCR_FR	Name of buildup area in French	Categorical
CD_COLL_TYPE	Type of collision code	Categorical (1-8)
TX_COLL_TYPE_DESCR_FR	Collision type name in French	Categorical
CD_LIGHT_COND	light conditions code	Categorical (1-4)
TX_LIGHT_COND_DESCR_FR	Description of light conditions in French	Categorical
CD_ROAD_TYPE	Road type code	Categorical (1-2)
TX_ROAD_TYPE_DESCR_FR	Road type description in French	Categorical
CD_MUNTY_REFNIS	NSI-code of the municipality	Categorical
TX_MUNTY_DESCR_FR	Municipality name in French	Categorical
CD_DSTR_REFNIS	NSI-code of the district	Categorical
TX_ADM_DSTR_DESCR_FR	Name of the district in French	Categorical
CD_PROV_REFNIS	NSI-code of the district	Categorical
TX_PROV_DESCR_FR	Name of the province in French	Categorical
CD_RGN_REFNIS	NSI-code of the region	Categorical
TX_RGN_DESCR_FR	Name of the region in French	Categorical
MS_ACCT	Number of accidents with dead or injured	Integer
MS_ACCT_WITH_DEAD	Number of accidents with dead	Integer
MS_ACCT_WITH_DEAD_30_DAYS	Number of accidents with dead, in the 30 days following the accident	Integer
MS_ACCT_WITH_MORY_INJ	Number of accidents with mortally injured	Integer
MS_ACCT_WITH_SERLY_INJ	Number of accidents with severely injured	Integer
MS_ACCT_WITH_SLY_INJ	Number of accidents with slightly injured	Integer

Let's now convert the type of some variables (e.g. DT_DAY to datetime, DT_HOUR to categorical...)

In [3]:

```
# DT_DAY typecasting to datetime, and renaming to 'DATE'
df_accidents['DATE'] = pd.to_datetime(df_accidents['DT_DAY'])
df_accidents = df_accidents.drop('DT_DAY', axis=1)
```

In [4]:

```
#Changing the dataframe to the right data formats: lots of castings to cate
gorical, dropping of French descriptions
categorical_variables = ['DT_HOUR', 'CD_DAY_OF_WEEK', 'CD_BUILD_UP_AREA',
'CD_COLL_TYPE', 'CD_LIGHT_COND', 'CD_ROAD_TYPE', 'CD_MUNTY_REFNIS', 'CD_DST
R_REFNIS', 'CD_PROV_REFNIS', 'CD_RGN_REFNIS']
df_cat = df_accidents[categorical_variables].astype('category')

df_accidents = pd.concat([df_accidents['DATE'], df_cat, df_accidents[['MS_A
CCT','MS_ACCT_WITH_DEAD', 'MS_ACCT_WITH_DEAD_30_DAYS', 'MS_ACCT_WITH_MORY_I
NJ', 'MS_ACCT_WITH_SERLY_INJ', 'MS_ACCT_WITH_SLY_INJ'] ]], axis=1)
```

In [5]:

```
# Check that all variables are the same type as expected and described earl
ier
df_accidents.dtypes
```

Out[5]:

```
DATE                         datetime64[ns]
DT_HOUR                            category
CD_DAY_OF_WEEK                     category
CD_BUILD_UP_AREA                   category
CD_COLL_TYPE                       category
CD_LIGHT_COND                      category
CD_ROAD_TYPE                       category
CD_MUNTY_REFNIS                    category
CD_DSTR_REFNIS                     category
CD_PROV_REFNIS                     category
CD_RGN_REFNIS                      category
MS_ACCT                               int64
MS_ACCT_WITH_DEAD                     int64
MS_ACCT_WITH_DEAD_30_DAYS             int64
MS_ACCT_WITH_MORY_INJ                 int64
MS_ACCT_WITH_SERLY_INJ                int64
MS_ACCT_WITH_SLY_INJ                  int64
dtype: object
```

Now let us define additional variables, which are new definition of "DATE" in terms of "DAY" (day of the month), "MONTH", "QUARTER" and "YEAR". This will allow us to check whether there exist general trends or seasonal effects for example. Once those new variables are added to the data set "df_accidents" we can start the Exploratory Data Analysis.

In [6]:

```
df_accidents['DAY'] = df_accidents['DATE'].apply(lambda date: date.day)
df_accidents['MONTH'] = df_accidents['DATE'].apply(lambda date: date.month)
df_accidents['QUARTER'] = df_accidents['DATE'].apply(lambda date: date.quar
ter)
df_accidents['YEAR'] = df_accidents['DATE'].apply(lambda date: date.year)
```

In [7]:

```
df_accidents.index = pd.DatetimeIndex(df_accidents['DATE'])
```

II. Methodology

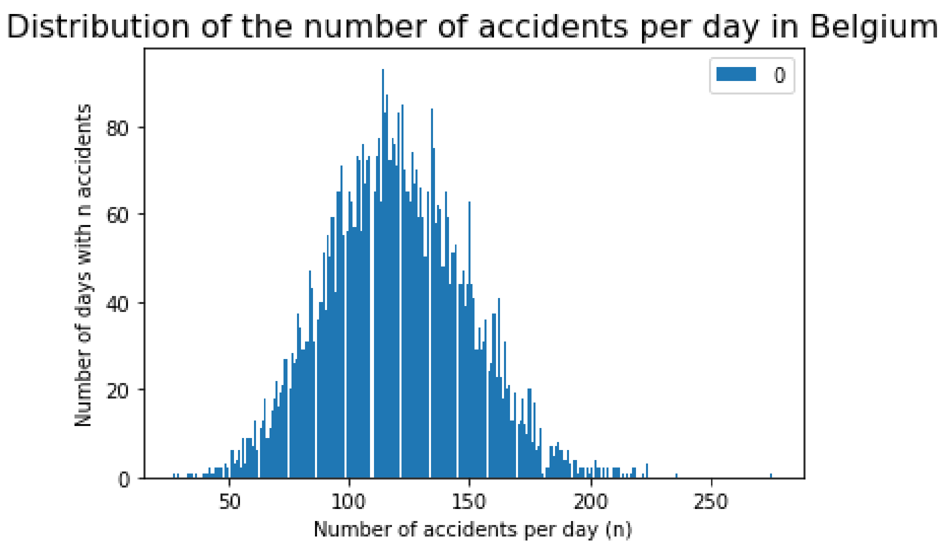
II a. Exploratory Data Analysis

First of all, let's get acquainted with the data that we are dealing with. A first step is to check the distribution plot of the number of accidents that we have. From what we can see below, this is pretty well approximated by a normal distribution.

In [8]:

```
daily_accidents = pd.DataFrame(df_accidents.resample('D').size())
plt.figure(figsize=(12,8))
daily_accidents.plot(kind = 'hist', bins=270)

plt.title("Distribution of the number of accidents per day in Belgium", fon
tsize = 16)
plt.xlabel("Number of accidents per day (n)")
plt.ylabel("Number of days with n accidents")

plt.show()
```

```
<Figure size 864x576 with 0 Axes>
```



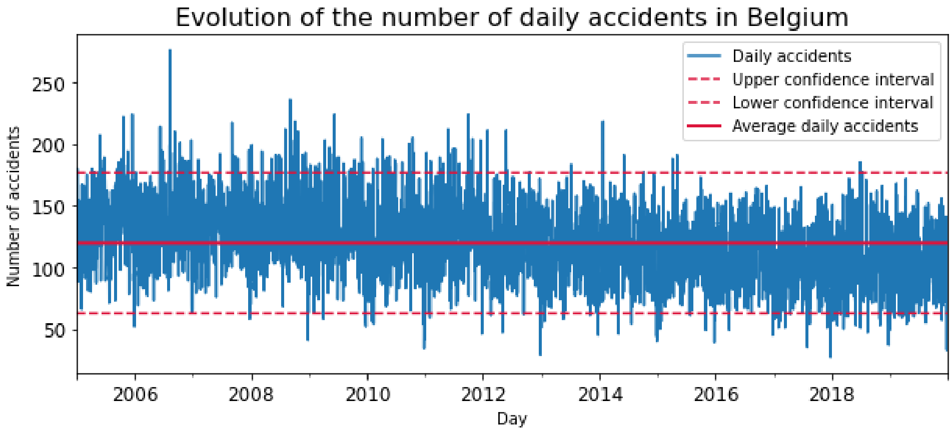
Now that we know how the accidents are distributed, it would be interesting to know how those accidents are evolving with time. From what we can see below, there is a general trend of decreasing number of accidents with time (in years), while we can also guess some kind of seasonal effects.

In [9]:

```
# Adding Mean and Standard deviation to the data set of "daily_accidents"
daily_accidents['MEAN'] = df_accidents.resample('D').size().mean()
daily_accidents['STD'] = df_accidents.resample('D').size().std()

# Under the assumption of normal distribution, 95% of data contained in mea
n +/- 1.96*std
up_conf_int = daily_accidents['MEAN'] + 1.96 * daily_accidents['STD']
low_conf_int = daily_accidents['MEAN'] - 1.96 * daily_accidents['STD']

plt.figure(figsize=(10,4))
df_accidents.resample('D').size().plot(label='Daily accidents')
up_conf_int.plot(color='crimson', ls='--', linewidth=1.5, label='Upper conf
idence interval')
low_conf_int.plot(color='crimson', ls='--', linewidth=1.5, label='Lower con
fidence interval')
daily_accidents['MEAN'].plot(color='crimson', linewidth=2, label='Average d
aily accidents')

plt.title('Evolution of the number of daily accidents in Belgium', fontsize
=16)
plt.xlabel('Day')
plt.ylabel('Number of accidents')
plt.tick_params(labelsize=12)
plt.legend(prop={'size':10})
```

Out[9]:

```
<matplotlib.legend.Legend at 0x128fadd8>
```



Accidents depend on the moment(s) of the year(s)

In [10]:

```
# Renaming column 0 by DAILY_ACC => the number of daily accidents
daily_accidents.rename(columns={0:'DAILY_ACC'}, inplace = 'TRUE')

#Adding a column DATE in the daily_accidents dataframe
daily_accidents['DATE'] = daily_accidents.index.values

#creating new variables corresponding to DAY, MONTH, QUARTER and YEAR
daily_accidents['DAY'] = daily_accidents['DATE'].apply(lambda date: date.da
y)
daily_accidents['MONTH'] = daily_accidents['DATE'].apply(lambda date: date.
month)
daily_accidents['QUARTER'] = daily_accidents['DATE'].apply(lambda date: dat
e.quarter)
daily_accidents['YEAR'] = daily_accidents['DATE'].apply(lambda date: date.y
ear)
daily_accidents.head()
```

Out[10]:

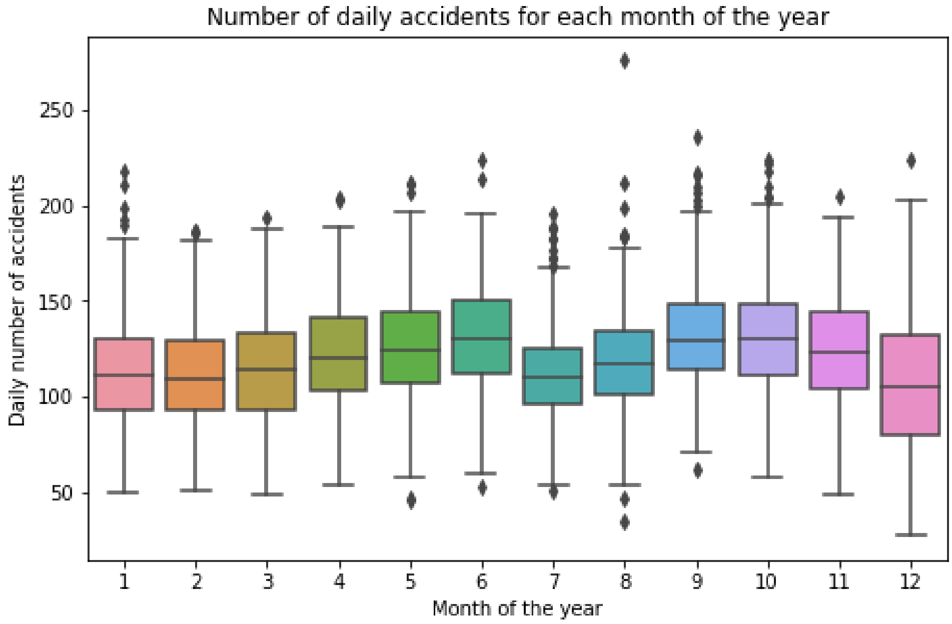
	DAILY_ACC	MEAN	STD	DATE	DAY	MONTH	QUARTER
DATE							
2005-01-01	94	119.654618	28.961461	2005-01-01	1	1	1
2005-01-02	103	119.654618	28.961461	2005-01-02	2	1	1
2005-01-03	96	119.654618	28.961461	2005-01-03	3	1	1
2005-01-04	138	119.654618	28.961461	2005-01-04	4	1	1
2005-01-05	180	119.654618	28.961461	2005-01-05	5	1	1

In [11]:

```
plt.figure(figsize=(8,5))
ax = sns.boxplot( x=daily_accidents["MONTH"], y=daily_accidents['DAILY_ACC'
] )
ax.set(xlabel ='Month of the year', ylabel = 'Daily number of accidents', t
itle= 'Number of daily accidents for each month of the year')
```

Out[11]:

```
[Text(0, 0.5, 'Daily number of accidents'),
 Text(0.5, 0, 'Month of the year'),
 Text(0.5, 1.0, 'Number of daily accidents for each month of t
he year')]
```

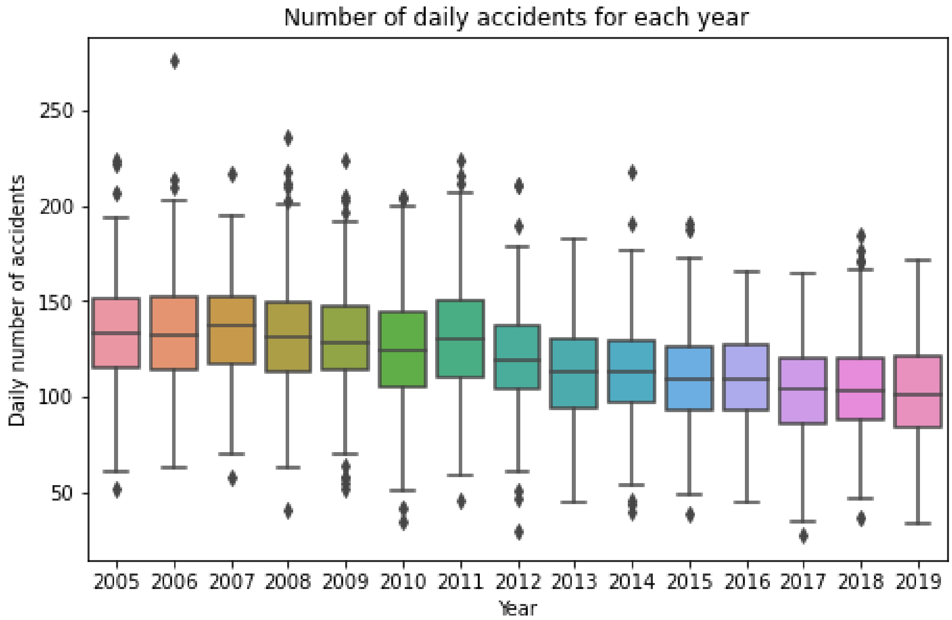


In [12]:

```
plt.figure(figsize=(8,5))
ax = sns.boxplot( x=daily_accidents["YEAR"], y=daily_accidents['DAILY_ACC']
)
ax.set(xlabel ='Year', ylabel = 'Daily number of accidents', title= 'Number
of daily accidents for each year')
```

Out[12]:

```
[Text(0, 0.5, 'Daily number of accidents'),
 Text(0.5, 0, 'Year'),
 Text(0.5, 1.0, 'Number of daily accidents for each year')]
```

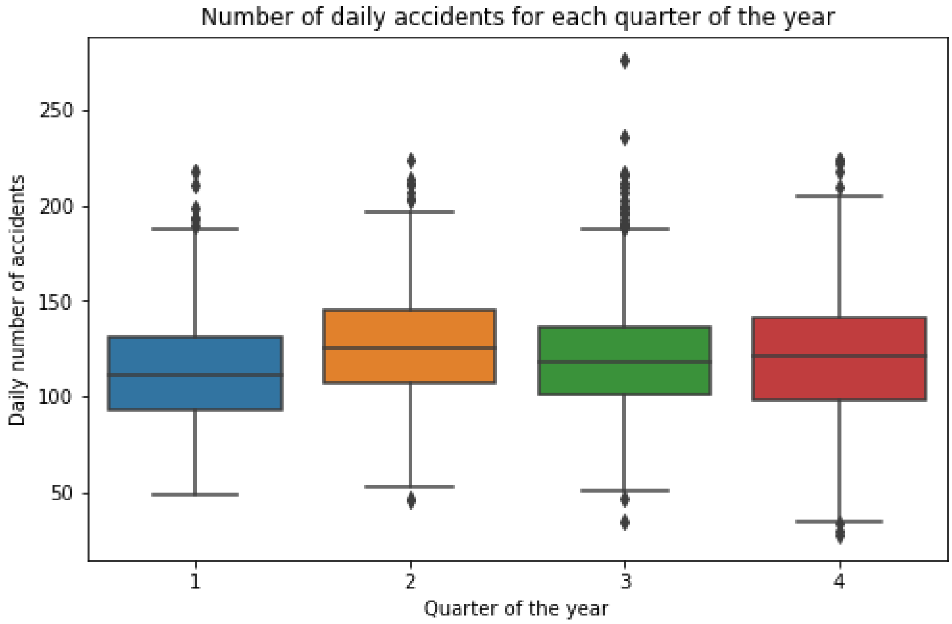


In [13]:

```
plt.figure(figsize=(8,5))
ax = sns.boxplot( x=daily_accidents["QUARTER"], y=daily_accidents['DAILY_AC
C'] )
ax.set(xlabel ='Quarter of the year', ylabel = 'Daily number of accidents',
title= 'Number of daily accidents for each quarter of the year')
```

Out[13]:

```
[Text(0, 0.5, 'Daily number of accidents'),
 Text(0.5, 0, 'Quarter of the year'),
 Text(0.5, 1.0, 'Number of daily accidents for each quarter of
the year')]
```



From the boxplots above, one can clearly see that the number of daily accidents is clearly dependent on the the timing, e.g. in July the number of accidents sharply collapses, in comparison to the month of June. This can probably be explained by the fact that many belgian citizen are leaving the country and go on holiday abroad (reducing the number of cars, hence the number of potential accidents). We also see that the number of accidents is slowly going down year after year (as already mentioned before).

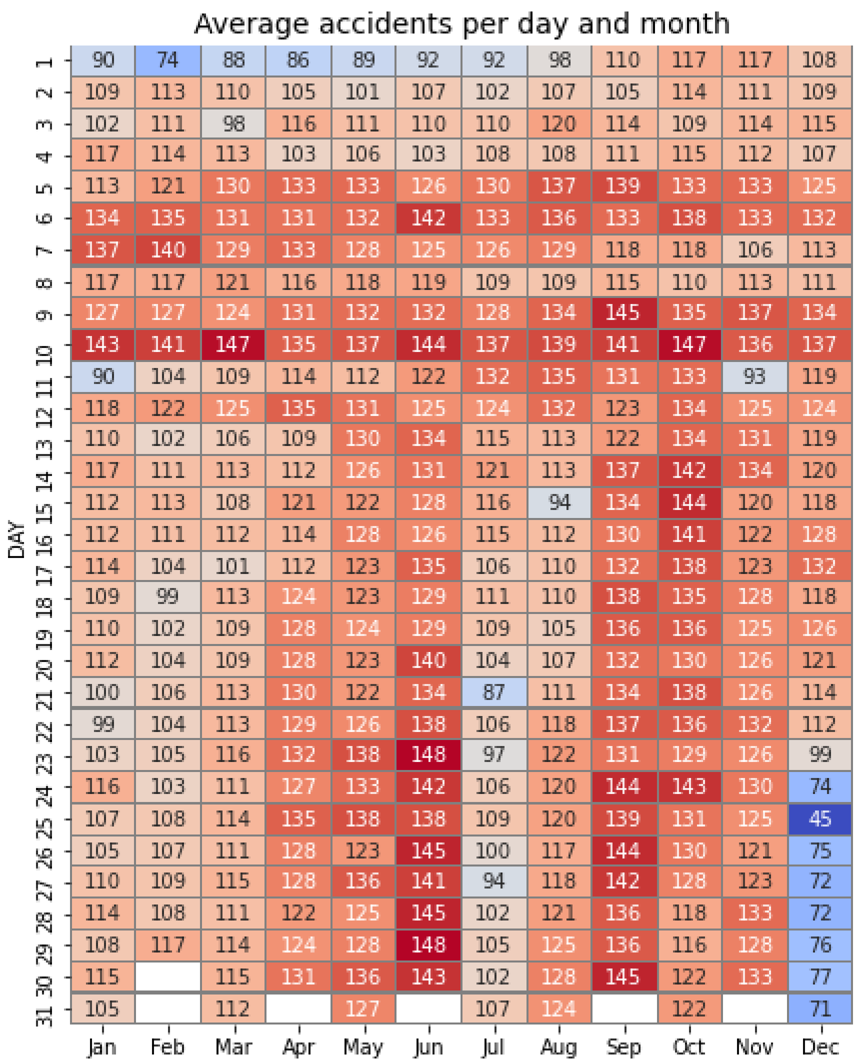
We can now create a table to see at which moment of the year we are more prone to get an accident. As predicted, we can see some patterns in the number of accidents per day. Just before christmas until 1st of January for example, the number of daily accidents is sharply reduced. On the contrary, the two last 10 days of June are significantly more dangerous than most of the year.

In [14]:

```
pivot = df_accidents.pivot_table(values='DATE', index='DAY', columns='MONT
H', aggfunc=len)
pivot_date_count = df_accidents.pivot_table(values='DATE', index='DAY', col
umns='MONTH', aggfunc=lambda x: len(x.unique()))
avg_accidents = pivot/pivot_date_count
avg_accidents.columns = ['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','S
ep','Oct','Nov','Dec']

plt.figure(figsize=(7,9))
plt.title('Average accidents per day and month', fontsize=14)
sns.heatmap(avg_accidents.round(), cmap='coolwarm', linecolor='grey',linewi
dths=0.1, cbar=False, annot=True, fmt=".0f")
```

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x138f23e8>
```



In [15]:

```
# Adding day of week to the data frame daily_accidents
DOW = list()
for i in range (0, len(daily_accidents['DATE'])):
    DOW.append(daily_accidents['DATE'][i].weekday())
daily_accidents['DOW'] = DOW
daily_accidents.head()
```

Out[15]:

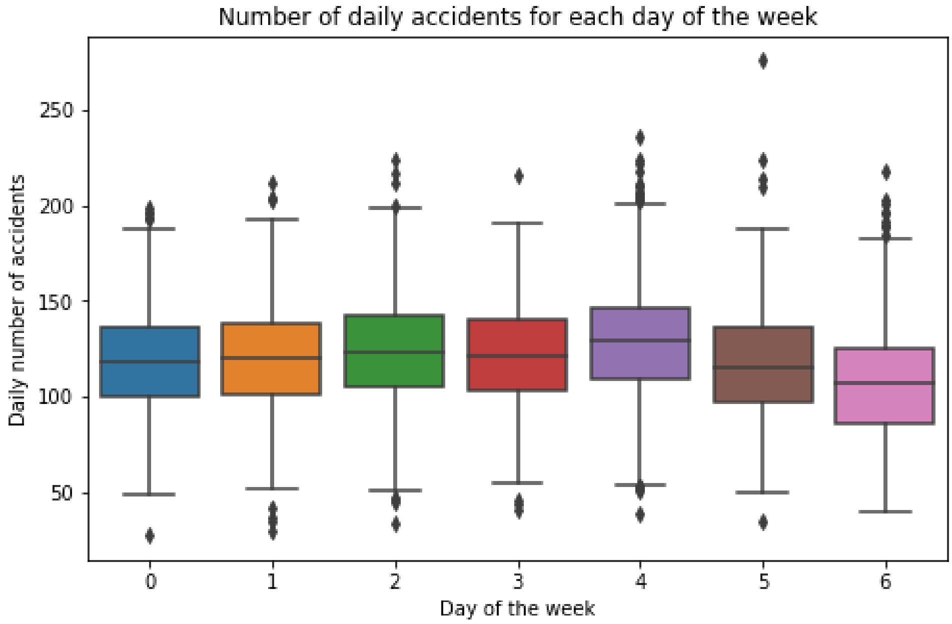
	DAILY_ACC	MEAN	STD	DATE	DAY	MONTH	QUARTER
DATE							
2005-01-01	94	119.654618	28.961461	2005-01-01	1	1	1
2005-01-02	103	119.654618	28.961461	2005-01-02	2	1	1
2005-01-03	96	119.654618	28.961461	2005-01-03	3	1	1
2005-01-04	138	119.654618	28.961461	2005-01-04	4	1	1
2005-01-05	180	119.654618	28.961461	2005-01-05	5	1	1

In [16]:

```
plt.figure(figsize=(8,5))
ax = sns.boxplot( x=daily_accidents['DOW'], y=daily_accidents['DAILY_ACC']
)
ax.set(xlabel ='Day of the week', ylabel = 'Daily number of accidents', tit
le= 'Number of daily accidents for each day of the week')
```

Out[16]:

```
[Text(0, 0.5, 'Daily number of accidents'),
 Text(0.5, 0, 'Day of the week'),
 Text(0.5, 1.0, 'Number of daily accidents for each day of the
week')]
```

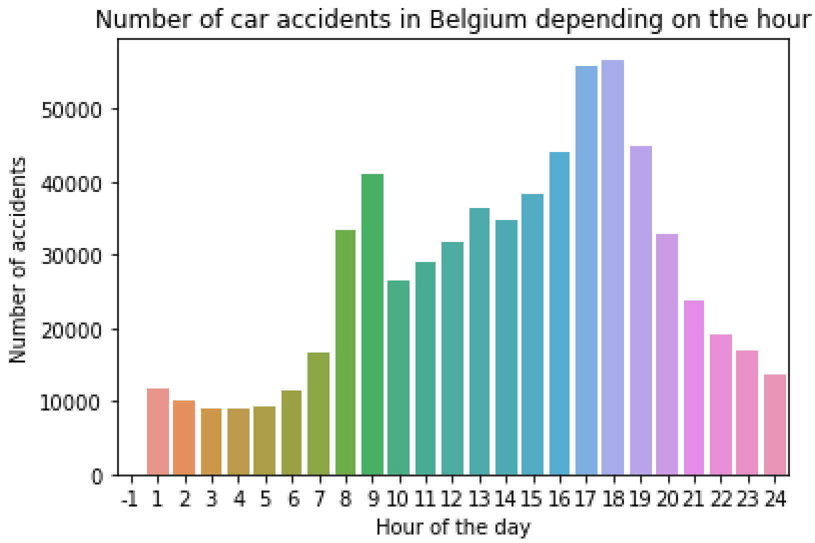


In [17]:

```
ax = sns.countplot(df_accidents['DT_HOUR'])
ax.set(xlabel='Hour of the day', ylabel ='Number of accidents', title ='Num
ber of car accidents in Belgium depending on the hour')
```

Out[17]:

```
[Text(0, 0.5, 'Number of accidents'),
 Text(0.5, 0, 'Hour of the day'),
 Text(0.5, 1.0, 'Number of car accidents in Belgium depending
on the hour')]
```

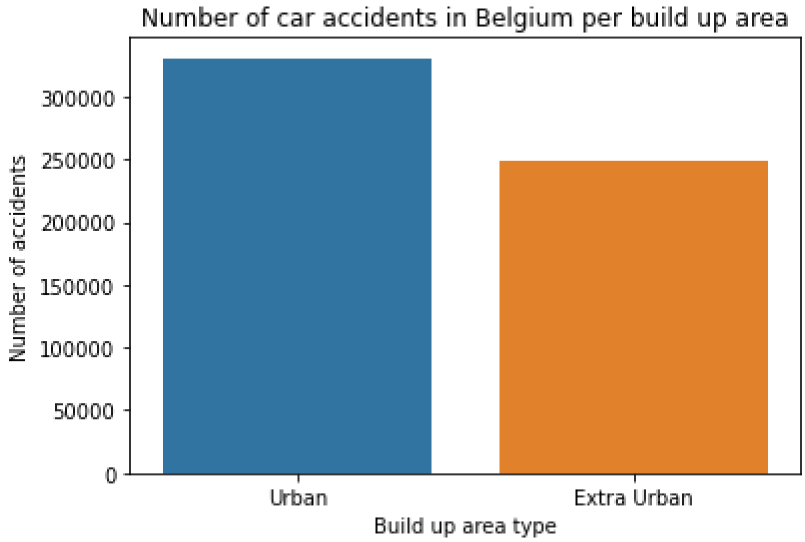


In [18]:

```
df_accidents['CD_BUILD_UP_AREA'] = df_accidents['CD_BUILD_UP_AREA'].cat.ren
ame_categories(["Urban", "Extra Urban"])
ax = sns.countplot(df_accidents['CD_BUILD_UP_AREA'])
ax.set(xlabel='Build up area type', ylabel ='Number of accidents', title =
'Number of car accidents in Belgium per build up area')
```

Out[18]:

```
[Text(0, 0.5, 'Number of accidents'),
 Text(0.5, 0, 'Build up area type'),
 Text(0.5, 1.0, 'Number of car accidents in Belgium per build
up area')]
```

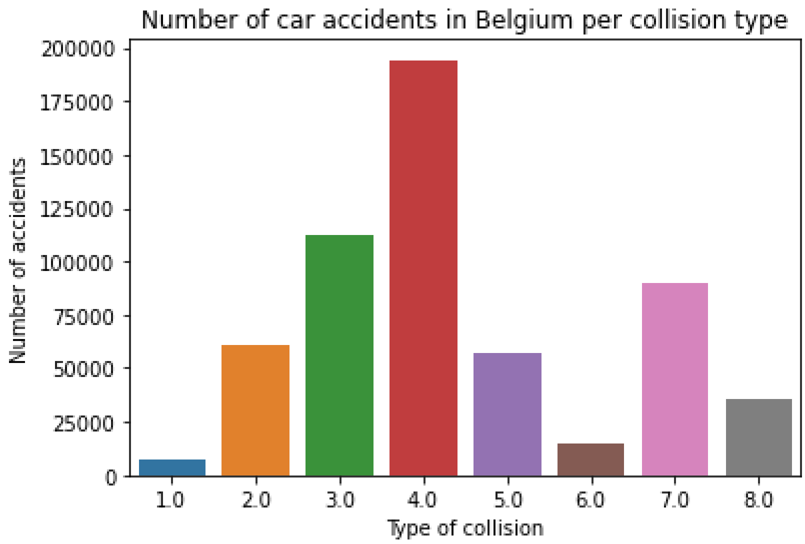


In [19]:

```
ax = sns.countplot(df_accidents['CD_COLL_TYPE'])
ax.set(xlabel='Type of collision', ylabel ='Number of accidents', title ='N
umber of car accidents in Belgium per collision type')
```

Out[19]:

```
[Text(0, 0.5, 'Number of accidents'),
 Text(0.5, 0, 'Type of collision'),
 Text(0.5, 1.0, 'Number of car accidents in Belgium per collis
ion type')]
```

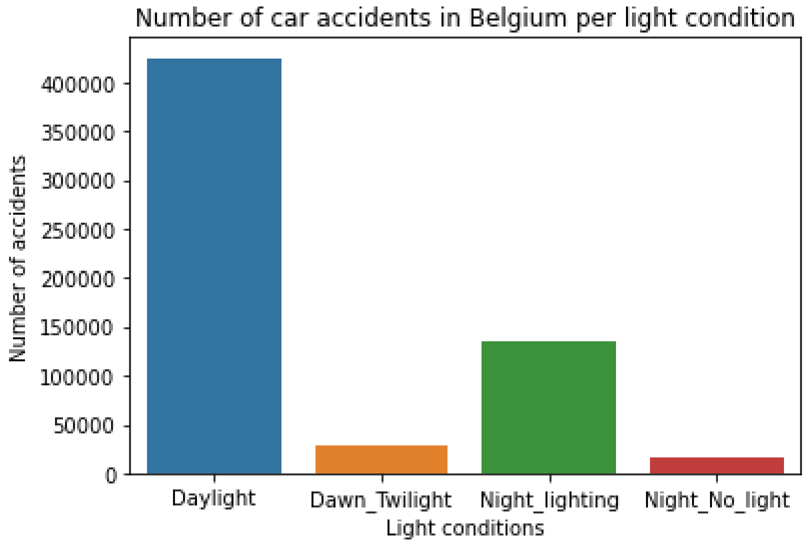


In [20]:

```
df_accidents['CD_LIGHT_COND'] = df_accidents['CD_LIGHT_COND'].cat.rename_categories(['Daylight', 'Dawn_Twilight', 'Night_lighting', 'Night_No_light'])
ax = sns.countplot(df_accidents['CD_LIGHT_COND'])
ax.set(xlabel='Light conditions', ylabel ='Number of accidents', title ='Number of car accidents in Belgium per light condition')
```

Out[20]:

```
[Text(0, 0.5, 'Number of accidents'),
 Text(0.5, 0, 'Light conditions'),
 Text(0.5, 1.0, 'Number of car accidents in Belgium per light
condition')]
```

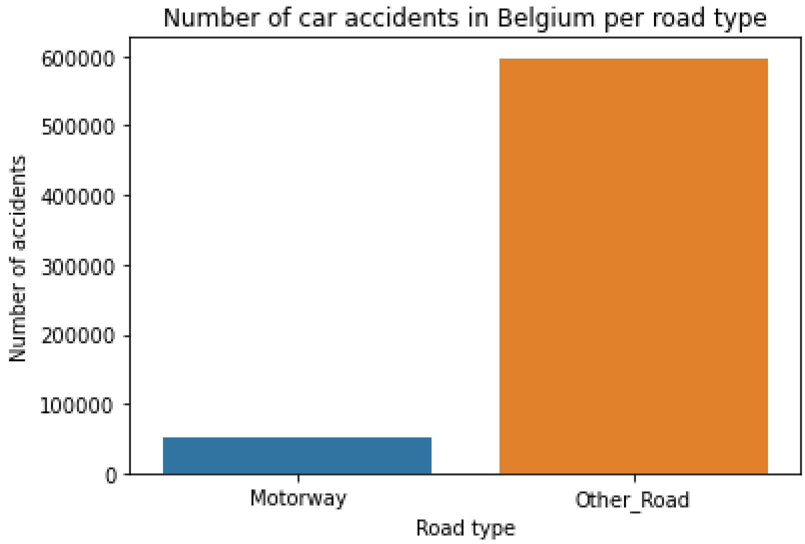


In [21]:

```
df_accidents['CD_ROAD_TYPE'] = df_accidents['CD_ROAD_TYPE'].cat.rename_cate
gories(['Motorway', 'Other_Road'])
ax = sns.countplot(df_accidents['CD_ROAD_TYPE'])
ax.set(xlabel='Road type', ylabel ='Number of accidents', title ='Number of
car accidents in Belgium per road type')
```

Out[21]:

```
[Text(0, 0.5, 'Number of accidents'),
 Text(0.5, 0, 'Road type'),
 Text(0.5, 1.0, 'Number of car accidents in Belgium per road t
ype')]
```

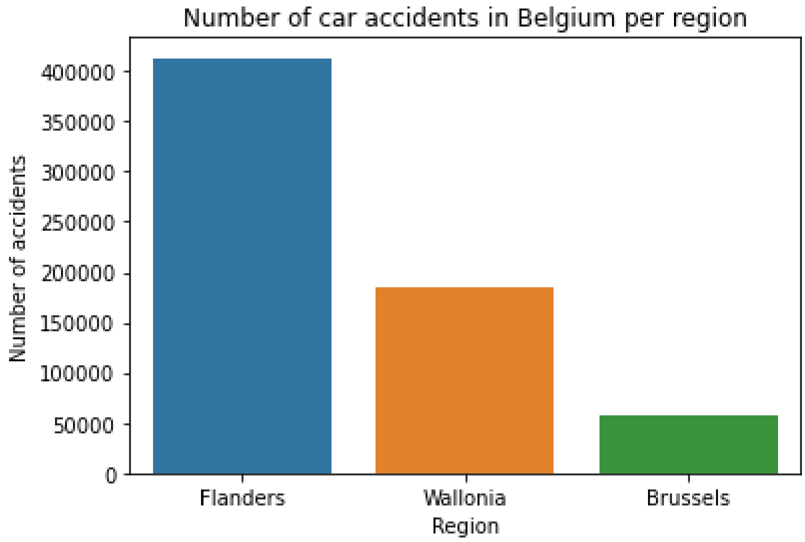


In [22]:

```
df_accidents['CD_RGN_REFNIS'] = df_accidents['CD_RGN_REFNIS'].cat.rename_ca
tegories(['Flanders', 'Wallonia', 'Brussels'])
ax = sns.countplot(df_accidents['CD_RGN_REFNIS'])
ax.set(xlabel='Region', ylabel ='Number of accidents', title ='Number of ca
r accidents in Belgium per region')
```

Out[22]:

```
[Text(0, 0.5, 'Number of accidents'),
 Text(0.5, 0, 'Region'),
 Text(0.5, 1.0, 'Number of car accidents in Belgium per regio
n')]
```



The main objective of this project is to warn drivers when there are high chances of severe car accidents. One can define a severe accident as an accident which is either with dead, with dead in the 30 days following the accident, as mortally injured or severely injured. One can now define that variable (called "SEVERE_ACC") and add it to our Data Frame df_accidents.

In [25]:

```
# TO DO ONLY ONCE (SINCE TIME CONSUMING): Compute SEVERE_ACC and add it to
 df_accidents so that not needed anymore. Then, export the DataFrame to CS
V, can be openened next time.
# ------------------------------------------------------------------------
---------------------------------------
# SEVERE_ACC = [None]*(len(df_accidents))
# for i in range(len(df_accidents)):
#       SEVERE_ACC[i] = df_accidents['MS_ACCT_WITH_DEAD'][i] or df_accident
s['MS_ACCT_WITH_DEAD_30_DAYS'][i] or df_accidents['MS_ACCT_WITH_MORY_INJ']
[i] or df_accidents['MS_ACCT_WITH_SERLY_INJ'][i]
# df_accidents['SEVERE_ACC'] = SEVERE_ACC
# df_accidents.to_csv(r'C:\Users\NJ5866\Desktop\Final project\Road_accident
s_Belgium_SEVERE.csv', index=False)
```

In [23]:

```
df_SEVERE = pd.read_csv(r'C:\Users\NJ5866\Desktop\Final project\Road_accide
nts_Belgium_SEVERE.csv')
df_SEVERE.head()
```

Out[23]:

	DATE	DT_HOUR	CD_DAY_OF_WEEK	CD_BUILD_UP_AREA	CD_COLL_
0	2019-09-27	18	5	Extra Urban	
1	2019-11-20	12	3	Urban	
2	2019-07-15	14	1	Urban	
3	2019-04-20	2	6	Extra Urban	
4	2019-10-25	12	5	Extra Urban	

5 rows × 22 columns

Since we are interested in the occurence of severe accidents in general, and not in particular cases, we can remove the specific attributes related to the severity of the accidents (except of course for 'SEVERE_ACC', which is the main objective of this research). Apart from that, we can also remove variables if they are duplicates of others. This will be done via the correlation matrix, taking into account all numerical variables.

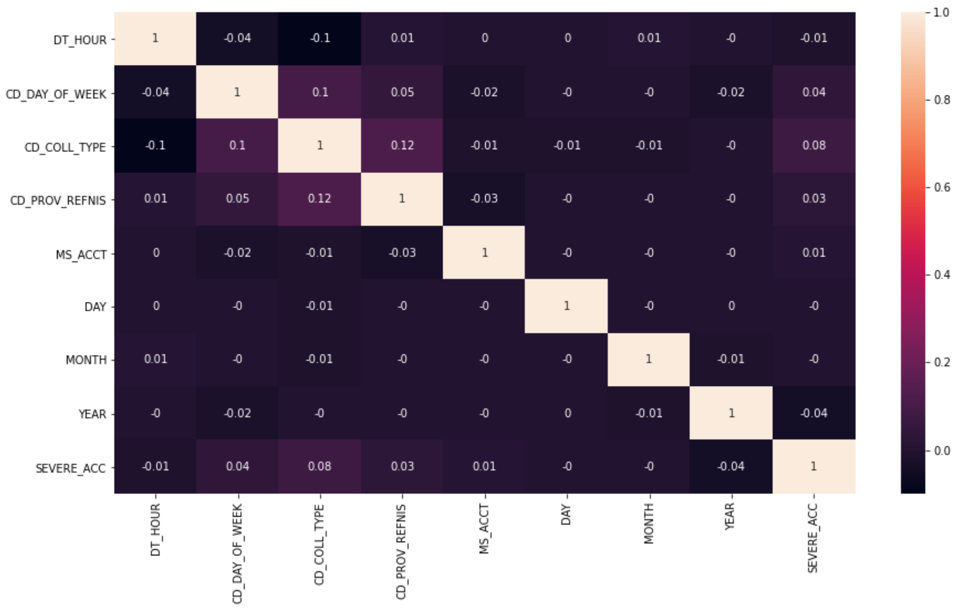
We can observe that some attributes are highly correlated and redundent, that is why we can drop some of them. Variables Quarter and Month are highly correlated (97%), hence one can drop one of the two (Quarter will be dissmissed), since almost all the information is contained in each variable. The municipality, district and province also display the same information, this is why districts and municipalities will be dropped.

In [24]:

```
df_SEVERE = df_SEVERE.drop(['MS_ACCT_WITH_DEAD','MS_ACCT_WITH_DEAD_30_DAYS'
, 'MS_ACCT_WITH_SLY_INJ', 'MS_ACCT_WITH_MORY_INJ', 'MS_ACCT_WITH_SERLY_INJ'
, 'QUARTER','CD_DSTR_REFNIS', 'CD_MUNTY_REFNIS'], axis=1)

plt.figure(figsize=(15,8))
sns.heatmap(df_SEVERE.corr().round(2),annot=True)
```

Out[24]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1de45cb8>
```



	DT_HOUR	CD_DAY_OF_WEEK	CD_COLL_TYPE	CD_PROV_REFNIS	MS_ACCT	DAY	MONTH	YEAR	SEVERE_ACC
DT_HOUR	1	-0.04	-0.1	0.01	0	0	0.01	-0	-0.01
CD_DAY_OF_WEEK	-0.04	1	0.1	0.05	-0.02	-0	-0	-0.02	0.04
CD_COLL_TYPE	-0.1	0.1	1	0.12	-0.01	-0.01	-0.01	-0	0.08
CD_PROV_REFNIS	0.01	0.05	0.12	1	-0.03	-0	-0	-0	0.03
MS_ACCT	0	-0.02	-0.01	-0.03	1	-0	-0	-0	0.01
DAY	0	-0	-0.01	-0	-0	1	-0	0	-0
MONTH	0.01	-0	-0.01	-0	-0	-0	1	-0.01	-0
YEAR	-0	-0.02	-0	-0	-0	0	-0.01	1	-0.04
SEVERE_ACC	-0.01	0.04	0.08	0.03	0.01	-0	-0	-0.04	1

In [25]:

```
df_SEVERE.head()
```

Out[25]:

	DATE	DT_HOUR	CD_DAY_OF_WEEK	CD_BUILD_UP_AREA	CD_COLL_TYI
0	2019-09-27	18	5	Extra Urban	1
1	2019-11-20	12	3	Urban	4
2	2019-07-15	14	1	Urban	Na
3	2019-04-20	2	6	Extra Urban	1
4	2019-10-25	12	5	Extra Urban	5

II b. Data Cleaning

NaN values

In [26]:

```
df_SEVERE.isnull().sum()
```

Out[26]:

```
DATE                    0
DT_HOUR                 0
CD_DAY_OF_WEEK          0
CD_BUILD_UP_AREA    76521
CD_COLL_TYPE        83748
CD_LIGHT_COND       50663
CD_ROAD_TYPE         6118
CD_PROV_REFNIS      57442
CD_RGN_REFNIS           0
MS_ACCT                 0
DAY                     0
MONTH                   0
YEAR                    0
SEVERE_ACC              0
dtype: int64
```

In [27]:

```
missing = [i for i in df_SEVERE.columns if df_SEVERE[i].isnull().sum() >0]
missing
```

Out[27]:

```
['CD_BUILD_UP_AREA',
 'CD_COLL_TYPE',
 'CD_LIGHT_COND',
 'CD_ROAD_TYPE',
 'CD_PROV_REFNIS']
```

For each missing value of an attribute, we will allocate the value with highest likelyhood of occurring (i.e. the most observed values in our data set df_accidents). For example, 330600 accidents happend in urban areas, compared to 248347 in Extra-urban, hence all missing values will be set as urban accidents.

In [28]:

```
for i in missing:
    print(df_SEVERE[i].value_counts())
```

```
Urban           330600
Extra Urban     248347
Name: CD_BUILD_UP_AREA, dtype: int64
4.0    194245
3.0    112384
7.0     90087
2.0     60768
5.0     57399
8.0     35150
6.0     14692
1.0      6995
Name: CD_COLL_TYPE, dtype: int64
Daylight          424555
Night_lighting    134218
Dawn_Twilight      29186
Night_No_light     16846
Name: CD_LIGHT_COND, dtype: int64
Other_Road    597112
Motorway       52238
Name: CD_ROAD_TYPE, dtype: int64
10000.0    114083
40000.0    107814
30000.0     82789
50000.0     64612
60000.0     58275
20001.0     55487
70000.0     52787
90000.0     26949
20002.0     18474
80000.0     16756
Name: CD_PROV_REFNIS, dtype: int64
```

In [29]:

```
for i in missing:

    if i == 'CD_BUILD_UP_AREA':
        df_SEVERE[i].fillna("Urban", inplace = True)
    elif i == 'CD_COLL_TYPE':
        df_SEVERE[i].fillna(4, inplace = True)
    elif i == 'CD_LIGHT_COND':
        df_SEVERE[i].fillna("Daylight", inplace = True)
    elif i == 'CD_ROAD_TYPE':
        df_SEVERE[i].fillna("Other_Road", inplace = True)
    else:
        df_SEVERE[i].fillna(10000.0, inplace = True)
```

II c. Models Building

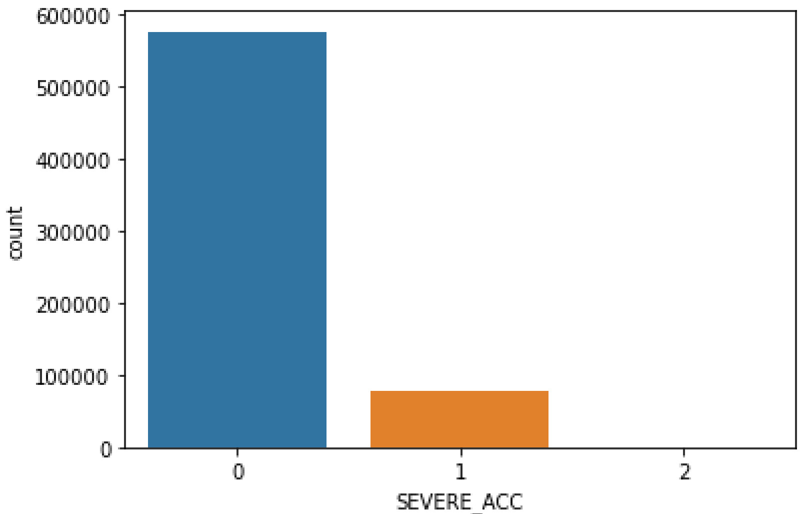
Undersampling

Since our data set is unbalanced, i.e. the number of severe accidents in comparison with the total number of accidents (observations in the data set) is really low, we must firstly rebalance it. As one can observe below, only 12% of the accidents are defined as severe. Hence, as a first step, we need to resample the data set. Since the number of total severe accidents is already around 80k, it has been decided to undersample our dataset (reduce the number of samples which are benign (not severe)).

In [30]:

```
sns.countplot(df_SEVERE['SEVERE_ACC'])
print("Number of benign vs severe accidents:")
print(df_SEVERE['SEVERE_ACC'].value_counts()/len(df_SEVERE))
```

```
Number of benign vs severe accidents:
0    0.879477
1    0.120468
2    0.000055
Name: SEVERE_ACC, dtype: float64
```



In [31]:

```
# Class count
Class_count_0, Class_count_1, Class_count_2 = df_SEVERE.SEVERE_ACC.value_co
unts()

print('Class_0:', Class_count_0)
print('Class_1: ', Class_count_1)
print('Class_2: ', Class_count_2 )
```

```
Class_0: 576469
Class_1:  78963
Class_2:  36
```

In [32]:

```
# Divide data set by class,  we drop Class_2, since they represent only 36
 observations and are kind of outliers
Accidents_class_0 = df_SEVERE[df_SEVERE['SEVERE_ACC'] == 0]
Accidents_class_1 = df_SEVERE[df_SEVERE['SEVERE_ACC'] == 1]
```

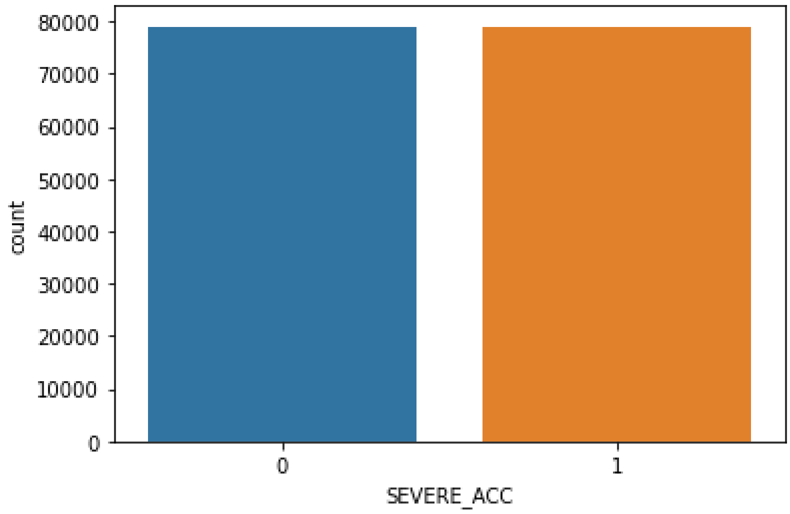
In [33]:

```
df_class_0_under = Accidents_class_0.sample(Class_count_1)
df_SEVERE_sampled = pd.concat([df_class_0_under, Accidents_class_1], axis =
0)

print('Random under-sampling: ')
print(df_SEVERE_sampled.SEVERE_ACC.value_counts())

sns.countplot(df_SEVERE_sampled['SEVERE_ACC'])
```

```
Random under-sampling:
1    78963
0    78963
Name: SEVERE_ACC, dtype: int64
```

Out[33]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x6b34640>
```



Preprocessing

The next step consists in casting the object variables into dummy variables in order to be able to apply the machine learning algorithms.

In [35]:

```
# Casting String variables into categorical
df_SEVERE_sampled['CD_BUILD_UP_AREA'] = df_SEVERE_sampled['CD_BUILD_UP_ARE
A'].astype('category')
df_SEVERE_sampled['CD_LIGHT_COND'] = df_SEVERE_sampled['CD_LIGHT_COND'].ast
ype('category')
df_SEVERE_sampled['CD_ROAD_TYPE'] = df_SEVERE_sampled['CD_ROAD_TYPE'].astyp
e('category')
df_SEVERE_sampled['CD_PROV_REFNIS'] = df_SEVERE_sampled['CD_PROV_REFNIS'].a
stype('category')
df_SEVERE_sampled['CD_RGN_REFNIS'] = df_SEVERE_sampled['CD_RGN_REFNIS'].ast
ype('category')



# Casting categorical into Dummy variables => in order to be able to perfor
m ML algos
from sklearn import preprocessing
le_BUILD_UP_AREA = preprocessing.LabelEncoder()
le_BUILD_UP_AREA.fit(['Urban','Extra Urban'])
df_SEVERE_sampled['CD_BUILD_UP_AREA'] = le_BUILD_UP_AREA.transform(df_SEVER
E_sampled['CD_BUILD_UP_AREA'])

le_LIGHT_COND = preprocessing.LabelEncoder()
le_LIGHT_COND.fit(['Daylight', 'Dawn_Twilight', 'Night_lighting', 'Night_No
_light'])
df_SEVERE_sampled['CD_LIGHT_COND'] = le_LIGHT_COND.transform(df_SEVERE_samp
led['CD_LIGHT_COND'])

le_ROAD_TYPE = preprocessing.LabelEncoder()
le_ROAD_TYPE.fit(['Motorway', 'Other_Road'])
df_SEVERE_sampled['CD_ROAD_TYPE'] = le_ROAD_TYPE.transform(df_SEVERE_sample
d['CD_ROAD_TYPE'])

le_REGION = preprocessing.LabelEncoder()
le_REGION.fit(['Flanders', 'Wallonia', 'Brussels'])
df_SEVERE_sampled['CD_RGN_REFNIS'] = le_REGION.transform(df_SEVERE_sampled[
'CD_RGN_REFNIS'])

le_PROVINCE = preprocessing.LabelEncoder()
le_PROVINCE.fit([10000.0, 20001.0, 20002.0, 30000.0, 40000.0, 50000.0, 6000
0.0, 70000.0, 80000.0, 90000.0])
df_SEVERE_sampled['CD_PROV_REFNIS'] = le_PROVINCE.transform(df_SEVERE_sampl
ed['CD_PROV_REFNIS'])

df_SEVERE_sampled.head()
```

Out[35]:

	DATE	DT_HOUR	CD_DAY_OF_WEEK	CD_BUILD_UP_AREA	CD_COL
102372	2017-05-25	15	4	0	
325552	2011-03-08	18	3	1	
413151	2009-05-01	18	1	1	
363726	2011-10-25	8	2	1	
160510	2015-01-28	13	3	1	

Splitting Train and Test sets

Let's now split our data set df_SEVERE_sampled into training and test sets, with 80% of the data used to train our models and 20% to test them.

In [53]:

```
df_SEVERE_sampled.columns
```

Out[53]:

```
Index(['DATE', 'DT_HOUR', 'CD_DAY_OF_WEEK', 'CD_BUILD_UP_ARE
A', 'CD_COLL_TYPE',
       'CD_LIGHT_COND', 'CD_ROAD_TYPE', 'CD_PROV_REFNIS', 'CD_
RGN_REFNIS',
       'MS_ACCT', 'DAY', 'MONTH', 'YEAR', 'SEVERE_ACC'],
      dtype='object')
```

In [54]:

```
# definition of features set X and target variable y
# note: DATE and YEAR have been dropped
X = df_SEVERE_sampled[['DT_HOUR', 'CD_DAY_OF_WEEK', 'CD_BUILD_UP_AREA', 'CD
_COLL_TYPE',
       'CD_LIGHT_COND', 'CD_ROAD_TYPE', 'CD_PROV_REFNIS', 'CD_RGN_REFNIS',
'DAY', 'MONTH']].values
y = df_SEVERE_sampled['SEVERE_ACC'].values
```

In [55]:

```
# Normalization of data
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
```

In [56]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, r
andom_state=4)
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (126340, 10) (126340,)
Test set: (31586, 10) (31586,)
```

K-Nearest Neighbors Algorithm

In [57]:

```
from sklearn.neighbors import KNeighborsClassifier
```

Let's start the algorithm with k = 4

In [58]:

```
k = 4

# Training the model
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)

# Predicting the test set
yhat = neigh.predict(X_test)

# Accuracy Evaluation
from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict
(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
Train set Accuracy:  0.7211255342725977
Test set Accuracy:  0.5599949344646362
```

Now, let's check what can be the best value for k between one and 9

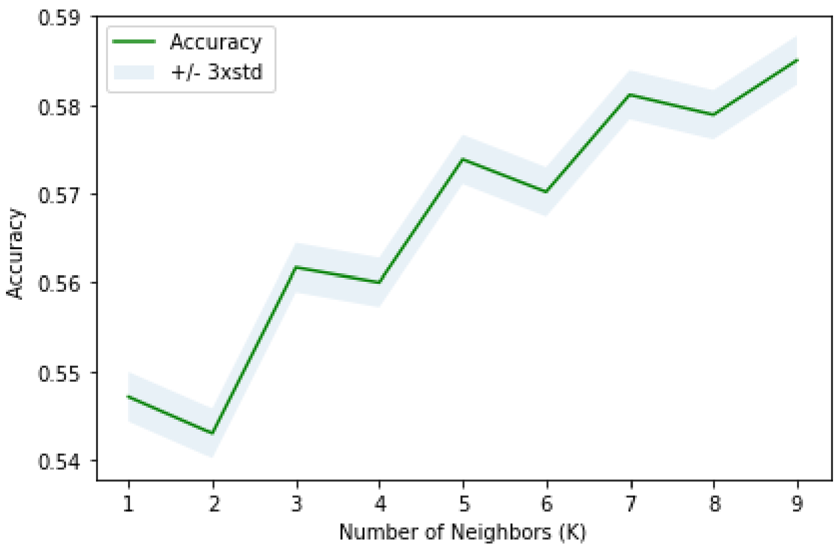
In [59]:

```
Ks = 10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
ConfustionMx = [];
for n in range(1,Ks):

    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)


    std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])

print('Mean accuracy for each k: ',mean_acc)
```

```
Mean accuracy for each k:  [0.54710948 0.54299373 0.56167289
0.55999493 0.57386184 0.57018932
 0.58114354 0.57889571 0.58503767]
```

In [60]:

```
# Plotting the accuracies for different k's
plt.plot(range(1,Ks),mean_acc,'g')
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc,
alpha=0.10)
plt.legend(('Accuracy ', '+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()
```



In [61]:

```
# Test with k = 30
k = 30

# Training the model
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)

# Predicting the test set
yhat = neigh.predict(X_test)

# Accuracy Evaluation
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict
(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
Train set Accuracy:  0.6401298084533797
Test set Accuracy:  0.6036218577850947
```

As one can see, although we increased the number of neighbors from 9 to 30, the accuracy did not improve greatly. It means that taking a number of neighbors k = 9 is good for our KNN model. The accuracy achieved is 0.585%. Let's see if we can do better with other methods.

Decision Trees Algorithm

In [63]:

```
from sklearn.tree import DecisionTreeClassifier
drugTree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
drugTree.fit(X_train,y_train)
predTree = drugTree.predict(X_test)
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_test, predTree
))
```

```
DecisionTrees's Accuracy:  0.6138162477046792
```

The decision tree algorithm is slightly better than the K-NN, with an accuracy of 61.4% on the test set. We will now check whether or not the logistic regression might be a better estimator of the risk of accidents.

Logistic Regression

In [64]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_train)
```

In [65]:

```
# Prediction
yhat = LR.predict(X_test)
```

In [67]:

```
# Computing the prediction probabilities
yhat_prob = LR.predict_proba(X_test)
yhat_prob[0:5]
```

Out[67]:

```
array([[0.60315603, 0.39684397],
       [0.34069039, 0.65930961],
       [0.44890161, 0.55109839],
       [0.63727986, 0.36272014],
       [0.52818453, 0.47181547]])
```

In [73]:

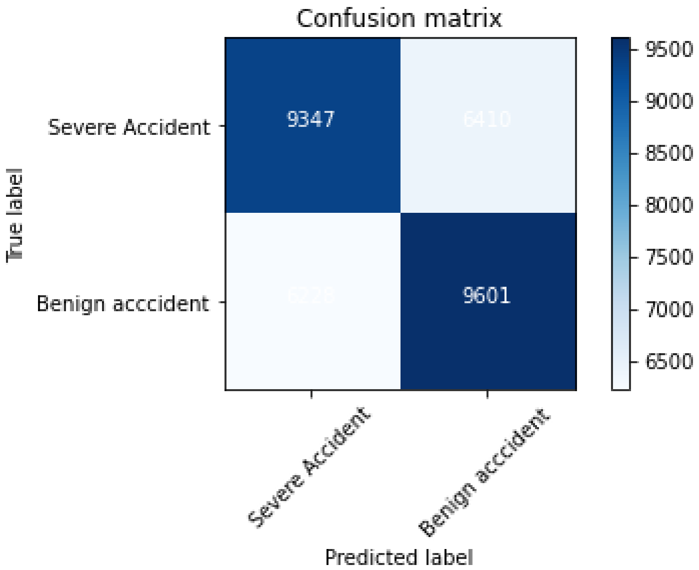
```
#Evaluation with the confusion matrix

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):

    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')
print('Confusion Matrix: ')
print(confusion_matrix(y_test, yhat, labels=[1,0]))
```

```
Confusion Matrix:
[[9347 6410]
 [6228 9601]]
```

In [74]:

```
cnf_matrix = confusion_matrix(y_test, yhat, labels=[1,0])
np.set_printoptions(precision=2)

plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['Severe Accident','Benign acccid
ent'],normalize= False,  title='Confusion matrix')
```

```
Confusion matrix, without normalization
[[9347 6410]
 [6228 9601]]
```



In [75]:

```
print (classification_report(y_test, yhat))
```

```
              precision    recall  f1-score   support

           0       0.60      0.61      0.60     15829
           1       0.60      0.59      0.60     15757

    accuracy                           0.60     31586
   macro avg       0.60      0.60      0.60     31586
weighted avg       0.60      0.60      0.60     31586
```

In [76]:

```
# Log loss: measures the performance of a classifier where the predicted ou
tput is a probability value between 0 and 1
from sklearn.metrics import log_loss
log_loss(y_test, yhat_prob)
```

Out[76]:

```
0.6675086826824437
```

Now, the accuracy of our predictor (Logistic regression) is greatly improved, in comparison to the previously presented ones: 66.8% accuracy on our balanced data set, making it our best estimator of the probability of a severe accident.

III. Results and discussion

This project allowed us to analyse the circumstances of severe accidents, i.e. accidents occuring with deads or severely injured people. As we have observed, 120 accidents are typically recorded everyday on the belgian roads, and 12% among them are severe (~ 14 per day).

We have also seen that the number of accidents is highly dependent on the time of the year; around Christmas for example, the number of accidents is low (~ 75 per day). On the contrary, during the last week of June, the number of daily accidents ranges between 138 and 148.

Another important feature for car's accidents is the hour of the day; typically more accidents occur during peak hours and in particular in the afternoon/evening (16:00-18:00), at the end of the working day. The day of the week is also (to a lesser extend) an explanatory variable of the number of accidents.

Beyond that, one can observe that the number of accidents is highly dependent on the location of your car, e.g. the "region". Flanders, for instance, counts more than the double number of accidents than the ones recorded in "Wallonia" (and even more than "Brussels").

In our Models building section, we have developed several Machine Learning algorithms for testing the likelyhood of getting a severe accident, based on the presented variables of the data set, i.e. the hour of the day, the day of the week, the day of the month, the month, the type of build-up area, the type of collision, the light conditions, the type of road, the province and the region. The first step in this section was to resample our dataset, so that the number of severe accidents was equal to the number of benign accidents. In order not to overfit the models, it has been decided to undersample our data set, i.e. randomly remove some of the benign accidents. We ended up with a balanced data set of 157,926 observations, with a 50/50 splitting of benign and severe accidents, respectively. Then, our data set was split into a test and training set (20% in the test set and 80% in the training set). Evenutally, we applied several algorithms on our training set and we tested their accuracy on the test set. The different algorithms were the K-Nearest Neighbors (K-NN), the decision trees and the logistic regression. As we demonstrated, the best algorithm was the logistic regression, which achieved an accuracy of more than 66.7% with our test set taken from the previously balanced data set. The advantage of this method is that it beyond assigning an observation to an accident's category (benign vs severe), it gives us the probability of being in this category.

IV. Conclusion

This analysis allowed us to help drivers to take the best decision whenever they want to drive their car from a point a to a point b (both of them in Belgium, even though this analysis could be enlarged and applied to all countries in the world). When a driver gives its starting and destination points, the differents roads that he is going to take, the timing at which he is going to drive etc., the algorithm is going to return to him a boolean value of the outcome likely to occur (severe accident vs benign accident), together with the probability of that outcome. Based on that the driver should take the decision whether or not to take other means of transportation, postpone his meeting, change his itinerary, or simply being even more cautious than usual. This new feature would be a big advance if used by GPS applications, in fact giving the driver an alternative safer route is (to my knowledge) a feature that has not been invented yet. This could thus be useful for anyone owning a car and a GPS.