Accident Severity Classification

Outline

1. Introduction

The UK government collects and publishes (usually on an annual basis) detailed information about traffic accidents across the country. This information includes, but is not limited to, geographical locations, weather conditions, type of vehicles, number of casualties and vehicle manoeuvres, making this a very interesting and comprehensive dataset for analysis and research.

The data come from the Open Data website of the UK government, where they have been published by the Department of Transport.

The dataset comprises of two csv files:

- Accident_Information.csv: every line in the file represents a unique traffic accident (identified by the Accident_Index column), featuring various properties related to the accident as columns. Date range: 2005-2017
- Vehicle_Information.csv: every line in the file represents the involvement of a unique vehicle in a unique traffic accident, featuring various vehicle and passenger properties as columns. Date range: 2004-2016

Our target is to predict the accident severity. The severity is devided to two catagories; severe and slight.

We had more than 2 million observations and close to 60 features. So, we sampled the data into about 600K observations and 23 features.

Two models were selected - Logistic Regression and the Random Forest Classifier.

In [2]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from datetime import datetime as dt
import time
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split as split
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn.linear_model import LogisticRegression
#from pandas.tools.plotting import scatter_matrix
import warnings
from sklearn.metrics import roc_auc_score
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.preprocessing import MinMaxScaler, FunctionTransformer, OneHotEncoder, KBinsD
iscretizer, MaxAbsScaler
from sklearn.feature_selection import VarianceThreshold

from sklearn.ensemble import RandomForestClassifier
from sklearn.impute import SimpleImputer
import seaborn as sns
sns.set()
import math

warnings.filterwarnings('ignore')
%matplotlib inline

import os
print(os.listdir(r"C:\Users\157088\Desktop\Coding and projects\Coursera\Coursera_Capstone
\Data"))
```

```
['Accident_Information.csv', 'Vehicle_Information.csv']
```

2. Data Preparation

2.1 Load Data

In [3]:

```
#Load Data and encode to latin
acc = pd.read_csv(r'C:\Users\157088\Desktop\Coding and projects\Coursera\Coursera_Capstone
\Data\Accident_Information.csv', encoding = 'latin')
veh = pd.read_csv(r'C:\Users\157088\Desktop\Coding and projects\Coursera\Coursera_Capstone
\Data\Vehicle_Information.csv', encoding = 'latin')

# Merging two data sets into one with inner join by index
df = pd.merge(veh, acc, how = 'inner', on = 'Accident_Index')

#Check data sample
print(df.shape)
df.head()
```

```
(2058408, 57)
```

Out[3]:

	Accident_Index	Age_Band_of_Driver	Age_of_Vehicle	Driver_Home_Area_Type	Driver_IMD_Dec
0	200501BS00002	36 - 45	3.0	Data missing or out of range	N.
1	200501BS00003	26 - 35	5.0	Urban area	[illegible]
2	200501BS00004	46 - 55	4.0	Urban area	[illegible]
3	200501BS00005	46 - 55	10.0	Data missing or out of range	N.
4	200501BS00006	46 - 55	1.0	Urban area	[illegible]

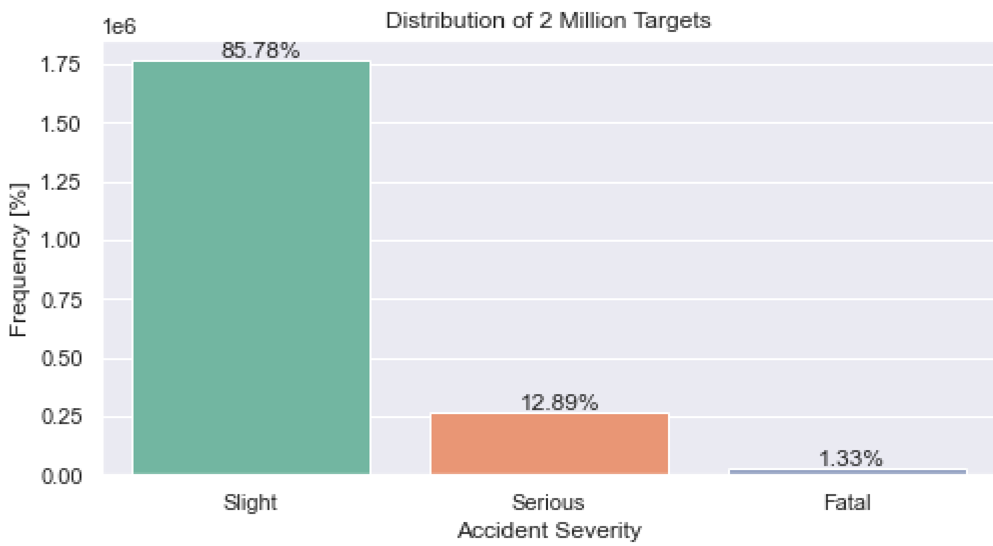
5 rows × 57 columns

2.2 Sample the data

by reducing rows with Slight Accident Severity

In [4]:

```
#Distribution of original data by targets

ax = sns.countplot(x = df.Accident_Severity ,palette="Set2")
sns.set(font_scale=1)
ax.set_xlabel(' ')
ax.set_ylabel(' ')
fig = plt.gcf()
fig.set_size_inches(8,4)
for p in ax.patches:
    ax.annotate('{:.2f}%'.format(100*p.get_height()/len(df.Accident_Severity)), (p.get_x()
+ 0.3, p.get_height()+10000))

plt.title('Distribution of 2 Million Targets',)
plt.xlabel('Accident Severity')
plt.ylabel('Frequency [%]')
plt.show()
```



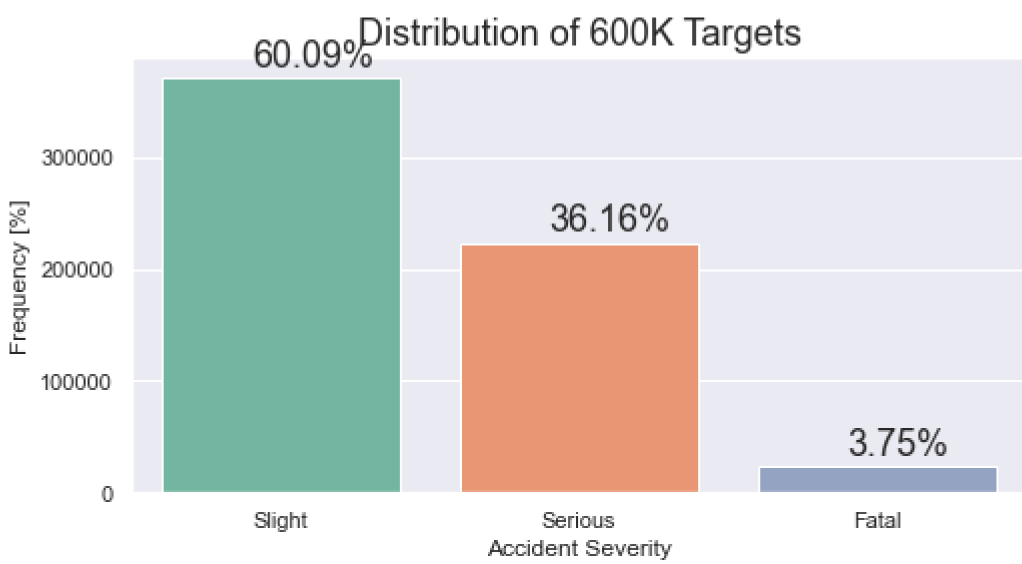
In [5]:

```
# Creating weights that are opposite to the weights of target
weights = np.where(df['Accident_Severity'] == 'Slight', .2, .8)

#Sampling only 30% of the data with new weights
df = df.sample(frac=0.3, replace=True, weights=weights)
print(df.shape)
#df.Accident_Severity.value_counts(normalize=True)
```

(617522, 57)

In [6]:

```
#Distribution of sample data by targets

ax = sns.countplot(x = df.Accident_Severity ,palette="Set2")
sns.set(font_scale=1.5)
ax.set_xlabel(' ')
ax.set_ylabel(' ')
fig = plt.gcf()
fig.set_size_inches(8,4)
for p in ax.patches:
    ax.annotate('{:.2f}%'.format(100*p.get_height()/len(df.Accident_Severity)), (p.get_x()
+ 0.3, p.get_height()+10000))

plt.title('Distribution of 600K Targets',)
plt.xlabel('Accident Severity')
plt.ylabel('Frequency [%]')
plt.show()
```



2.3 Checking for missing values

some will be filled, some will get omitted

In [7]:

```
#Missing values for each column
null_count = df.isnull().sum()
null_count[null_count>0]#.plot('bar', figsize=(30,10))
```

Out[7]:

```
Age_of_Vehicle                               102484
Driver_IMD_Decile                            206133
Engine_Capacity_.CC.                          75462
make                                          34054
model                                         96183
Propulsion_Code                               70376
Vehicle_Location.Restricted_Lane                281
2nd_Road_Class                               263537
2nd_Road_Number                                5824
Did_Police_Officer_Attend_Scene_of_Accident      49
Latitude                                         37
Location_Easting_OSGR                            37
Location_Northing_OSGR                           37
Longitude                                        37
LSOA_of_Accident_Location                     44185
Pedestrian_Crossing-Human_Control               217
Pedestrian_Crossing-Physical_Facilities         378
Speed_limit                                      16
Time                                             43
InScotland                                       15
dtype: int64
```

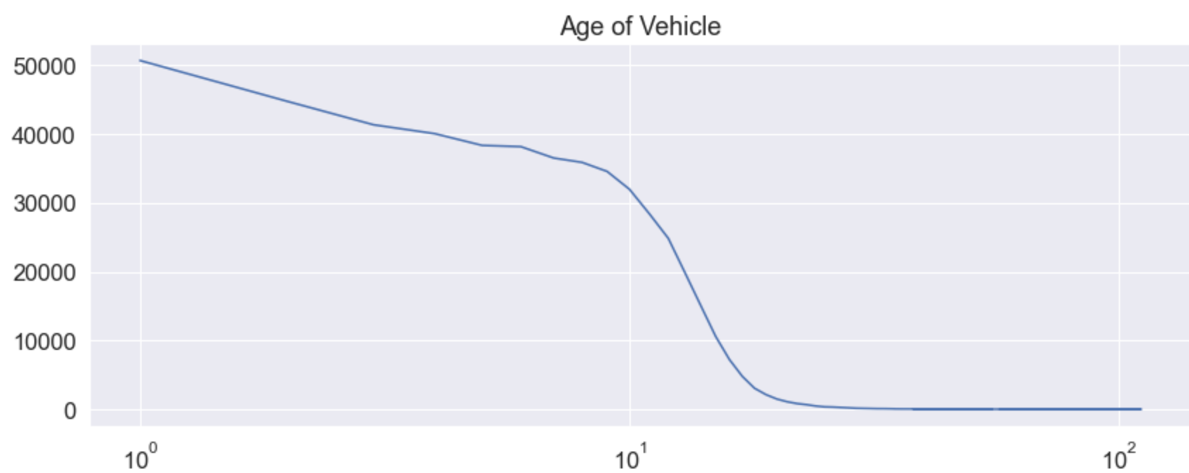
2.4 Exploratory Visualization

Age of Vehicle

In [8]:

```
(df.Age_of_Vehicle
 .value_counts()
 .plot(title = "Age of Vehicle",
       logx = True,
       figsize=(14,5)))

print('Min:',    df.Age_of_Vehicle.min(), '\n'
      'Max:',    df.Age_of_Vehicle.max(), '\n'
      'Median:', df.Age_of_Vehicle.median())
```

```
Min: 1.0
Max: 111.0
Median: 7.0
```



In [9]:

```
df.head()
```

Out[9]:

	Accident_Index	Age_Band_of_Driver	Age_of_Vehicle	Driver_Home_Area_Type	Driver_I
733987	2010950007644	36 - 45	2.0	Data missing or out of range	
759849	201001WW50033	26 - 35	6.0	Urban area	
784511	2010070204645	36 - 45	14.0	Urban area	
1885554	201613321O887	56 - 65	2.0	Urban area	
2032889	2016551601476	Data missing or out of range	4.0	Data missing or out of range	

5 rows × 57 columns

2.5 Create a new dataframe

with only the features we need and want, 25 features overall

In [10]:

```
df2 = df[['Accident_Index', '1st_Road_Class','Day_of_Week', 'Junction_Detail','Light_Condi
tions', 'Number_of_Casualties',
          'Number_of_Vehicles', 'Road_Surface_Conditions', 'Road_Type', 'Special_Condition
s_at_Site', 'Speed_limit',
          'Time', 'Urban_or_Rural_Area', 'Weather_Conditions', 'Age_Band_of_Driver', 'Age_
of_Vehicle',
          'Hit_Object_in_Carriageway', 'Hit_Object_off_Carriageway', 'make', 'Engine_Capac
ity_.CC.', 'Sex_of_Driver',
          'Skidding_and_Overturning', 'Vehicle_Manoeuvre', 'Vehicle_Type', 'Accident_Sever
ity'
        ]]
```

In [11]:

```
df2.head()
```

Out[11]:

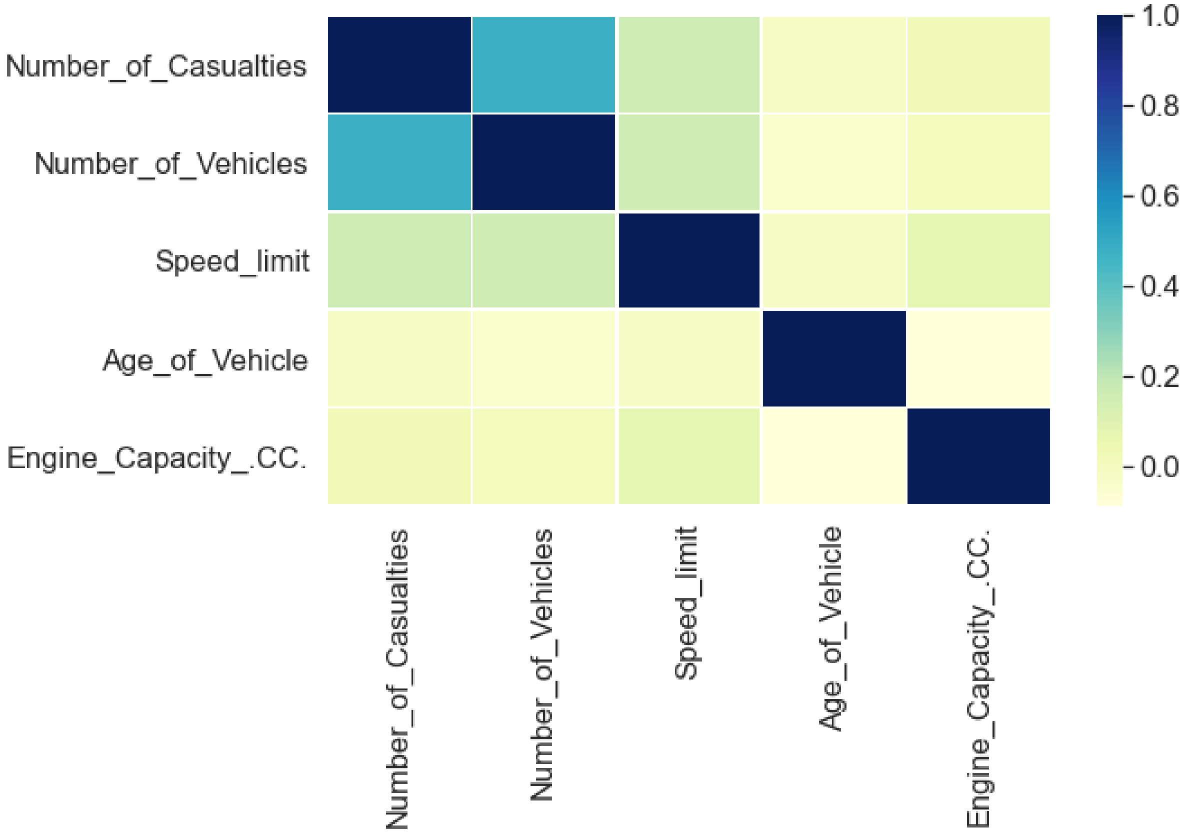
	Accident_Index	1st_Road_Class	Day_of_Week	Junction_Detail	Light_Conditions	Nun
733987	2010950007644	Unclassified	Monday	Roundabout	Daylight	
759849	201001WW50033	A	Wednesday	T or staggered junction	Daylight	
784511	2010070204645	A	Thursday	T or staggered junction	Daylight	
1885554	20161332I0887	A	Thursday	Roundabout	Daylight	
2032889	2016551601476	A	Tuesday	Not at junction or within 20 metres	Daylight	

5 rows × 25 columns

Correlation matrix

In [12]:

```
plt.figure(figsize=(9,5))
sns.heatmap(df2.corr(),linewidths=.5,cmap="YlGnBu")
plt.show()
```

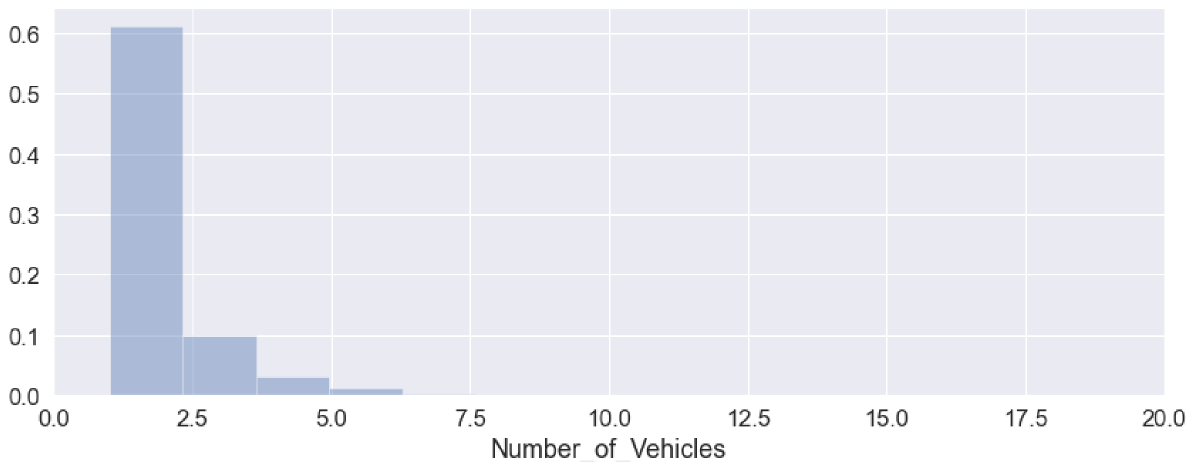


Number of Vehicles Distribution

In [13]:

```
plt.figure(figsize=(14,5))
sns.distplot(df2.Number_of_Vehicles).set_xlim(0,20)
print('Min:',    df2.Number_of_Vehicles.min(), '\n'
      'Max:',    df2.Number_of_Vehicles.max(), '\n'
      'Median:', df2.Number_of_Vehicles.median())
```

```
Min: 1
Max: 67
Median: 2.0
```

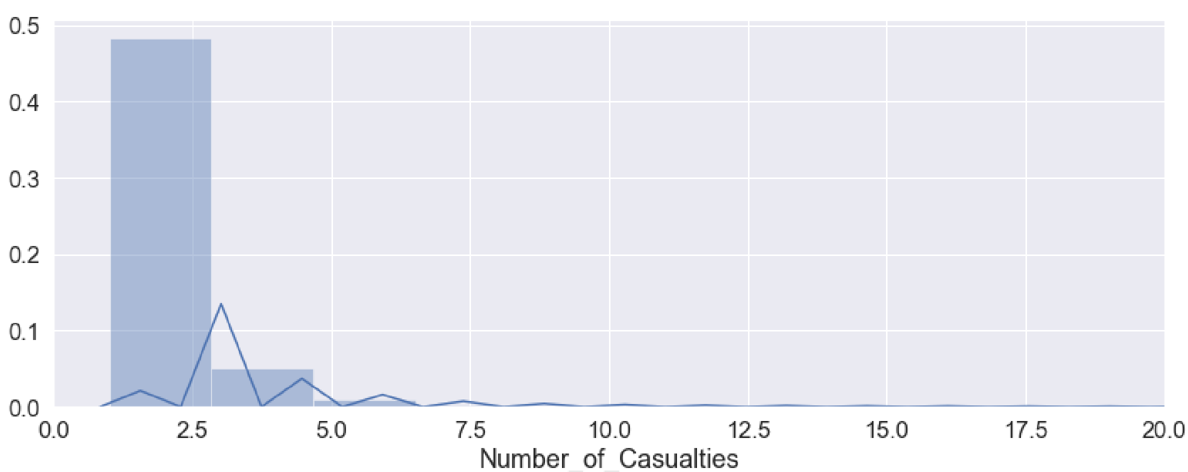


Number of casualties distribution

In [14]:

```
plt.figure(figsize=(14,5))
sns.distplot(df2.Number_of_Casualties).set_xlim(0,20)
print('Min:',    df2.Number_of_Casualties.min(), '\n'
      'Max:',    df2.Number_of_Casualties.max(), '\n'
      'Median:', df2.Number_of_Casualties.median())
```

```
Min: 1
Max: 93
Median: 1.0
```



From multiclass to two-classes

In [15]:

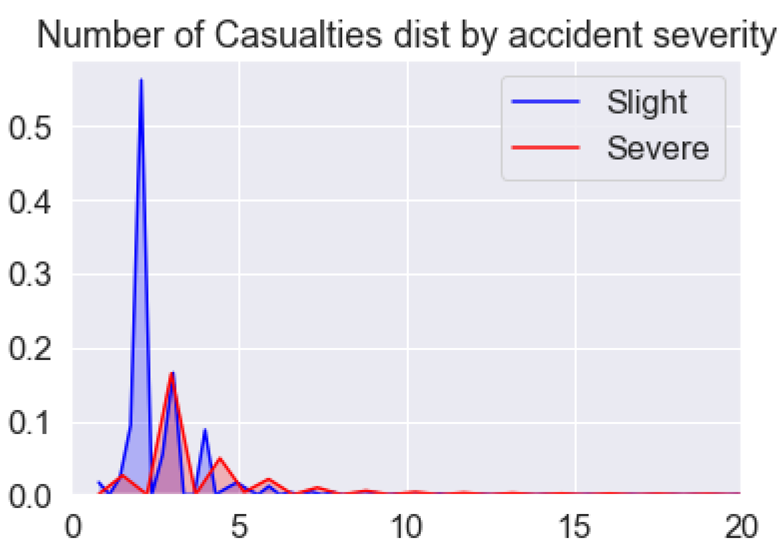
```
df2['Accident_Severity'] = df2['Accident_Severity'].replace(['Serious', 'Fatal'], 'Serious or Fatal')
df2 = pd.get_dummies(df2, columns=['Accident_Severity'])
df2 = df2.drop('Accident_Severity_Serious or Fatal', axis=1)
df2.Accident_Severity_Slight.value_counts(normalize=True)
```

Out[15]:

```
1    0.600885
0    0.399115
Name: Accident_Severity_Slight, dtype: float64
```

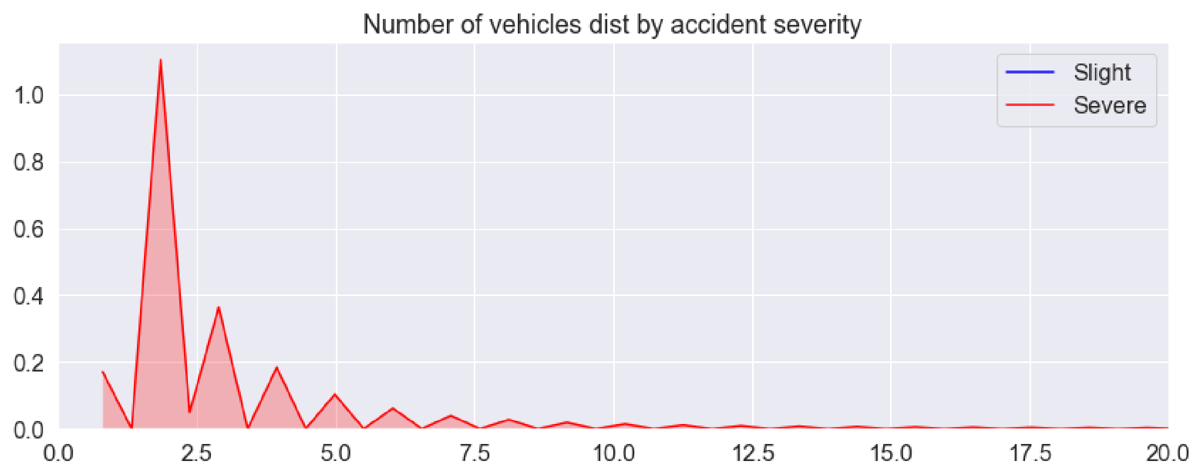
In [16]:

```
acc_slight = df2.Accident_Severity_Slight == 1
acc_severe = df2.Accident_Severity_Slight == 0

sns.kdeplot(df2.Number_of_Casualties[acc_slight],shade=True,color='Blue', label='Slight').set_xlim(0,20)
sns.kdeplot(df2.Number_of_Casualties[acc_severe],shade=True,color='Red', label='Severe').set_xlim(0,20)

plt.title('Number of Casualties dist by accident severity')
plt.show()

#print("we can see distribution between failed (under 2000), and successful (bigger the 2000)")
```



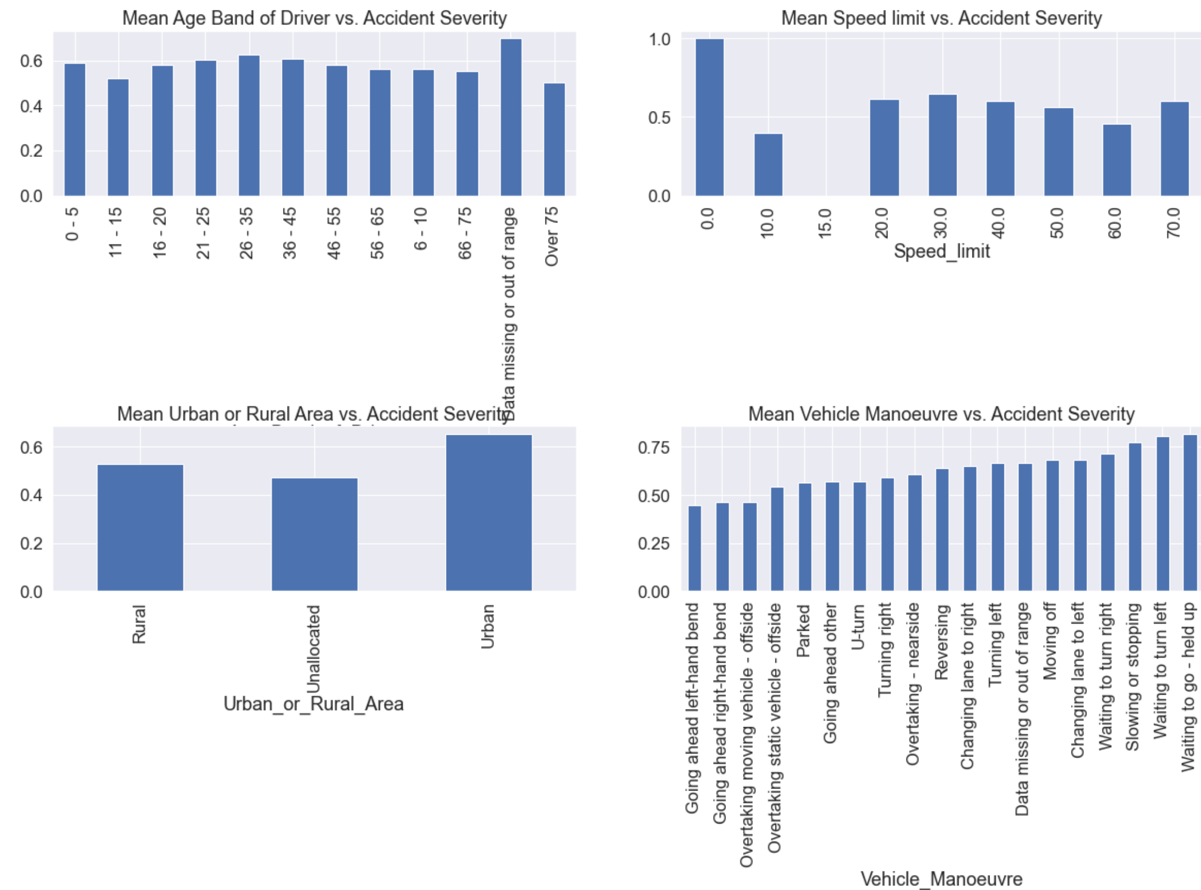
In [17]:

```
plt.figure(figsize=(14,5))

sns.kdeplot(df2.Number_of_Vehicles[acc_slight],shade=True,color='Blue', label='Slight').se
t_xlim(0,20)
sns.kdeplot(df2.Number_of_Vehicles[acc_severe],shade=True,color='Red', label='Severe').set
_xlim(0,20)

plt.title('Number of vehicles dist by accident severity')
plt.show()

#print("we can see distribution between failed (under 2000), and successful (bigger the 20
00)")
```



In [18]:

```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(20,10))
plt.subplots_adjust(hspace=1.4)

(df2.groupby(['Age_Band_of_Driver'])
 .mean()
 ['Accident_Severity_Slight']
 .sort_index()
 .plot
 .bar(title = "Mean Age Band of Driver vs. Accident Severity",
      ax = axes[0,0]))

(df2.groupby(['Speed_limit'])
 .mean()
 ['Accident_Severity_Slight']
 .sort_index()
 .plot
 .bar(title = "Mean Speed limit vs. Accident Severity",
      ax = axes[0,1]))

(df2.groupby(['Urban_or_Rural_Area'])
 .mean()
 ['Accident_Severity_Slight']
 .sort_index()
 .plot
 .bar(title = "Mean Urban or Rural Area vs. Accident Severity",
      ax = axes[1,0]))

(df2.groupby(['Vehicle_Manoeuvre'])
 .mean()
 ['Accident_Severity_Slight']
 .sort_values()
 .plot
 .bar(title = "Mean Vehicle Manoeuvre vs. Accident Severity",
      ax = axes[1,1]))

plt.show()
```



2.6 Split features and targets from the data

In [19]:

```
X = df2.drop(['Accident_Index','Accident_Severity_Slight'], axis=1)
y = df2.Accident_Severity_Slight
print(X.shape,
      y.shape)
```

```
(617522, 23) (617522,)
```

3. Training/Predicting Pipeline

Transform Speed Limit

In [20]:

```
def get_Speed_limit(df):
    return df[['Speed_limit']]

FullTransformerOnSpeedLimit = Pipeline([("Select_Speed_Limit", FunctionTransformer(func=ge
t_Speed_limit, validate=False)),
                                        ("Fill_Null",          SimpleImputer(missing_value
s=np.nan, strategy='most_frequent')),
                                        ("One_Hot_Encoder",    OneHotEncoder(sparse = Fals
e, handle_unknown='ignore'))
                                       ])

#FullTransformerOnSpeedLimit.fit_transform(X[:5000], y[:5000])
```

Transform Time

In [21]:

```
def get_Time(df):
    return pd.to_datetime(df['Time'], format='%H:%M').dt.time

def find_time_group(time_object):
    if time_object<pd.datetime.time(pd.datetime(2000,1,1,5,0)):
        return 'Night'
    elif time_object<pd.datetime.time(pd.datetime(2000,1,1,7,0)):
        return 'Early Morning'
    elif time_object<pd.datetime.time(pd.datetime(2000,1,1,10,0)):
        return 'Morning'
    elif time_object<pd.datetime.time(pd.datetime(2000,1,1,15,0)):
        return 'Midday'
    elif time_object<pd.datetime.time(pd.datetime(2000,1,1,18,0)):
        return 'Afternoon'
    elif time_object<pd.datetime.time(pd.datetime(2000,1,1,20,0)):
        return 'Evening'
    elif time_object<=pd.datetime.time(pd.datetime(2000,1,1,23,59)):
        return 'Late Evening'
    return np.nan

FullTransformerOnTime = Pipeline([("Select_Time",     FunctionTransformer(func=get_Time, v
alidate=False)),
                                  ("Group_Time",      FunctionTransformer(func=lambda x: x
.apply(find_time_group).to_frame(), validate=False)),
                                  ("Fill_Null",       SimpleImputer(missing_values=np.nan,
strategy='most_frequent')),
                                  ("One_Hot_Encoder", OneHotEncoder(sparse = False, handle
_unknown='ignore'))
                                 ])

#FullTransformerOnTime.fit_transform(X[:5000], y[:5000])
```

Transform Age of Vehicle

In [22]:

```
def get_Age_of_Vehicle(df):
    return df[['Age_of_Vehicle']]

FullTransformerOnAgeofVehicle = Pipeline([("Select_Age_of_Vehicle", FunctionTransformer(fu
nc=get_Age_of_Vehicle, validate=False)),
                                          ("Fill_Null",             SimpleImputer(missing_
values=np.nan, strategy='median'))
                                         ])

#FullTransformerOnAgeofVehicle.fit_transform(X[:5000], y[:5000])
```

Transform Make

In [23]:

```
def get_make(df):
    list_of_small_makers = list(df['make'].value_counts()[df['make'].value_counts() < 2000
].index)
    return df['make'].replace(list_of_small_makers, 'Other').to_frame()

FullTransformerOnMake = Pipeline([("Select_Make",      FunctionTransformer(func=get_make,
validate=False)),
                                   ("Fill_Null",       SimpleImputer(missing_values=np.nan
, strategy='constant', fill_value='Other')),
                                   ("One_Hot_Encoder", OneHotEncoder(sparse = False, handl
e_unknown='ignore'))])

#FullTransformerOnMake.fit_transform(X[:5000], y[:5000])
```

Transform Engine Capacity

In [24]:

```
def get_Engine_Capacity(df):
    return df[['Engine_Capacity_.CC.']]

FullTransformerOnEngineCapacity = Pipeline([("Select_Engine_Capacity",       FunctionTrans
former(func=get_Engine_Capacity, validate=False)),
                                            ("Fill_Null",                    SimpleImputer
(missing_values=np.nan, strategy='most_frequent')),
                                            ("Car_Types_by_Engine_Capacity", KBinsDiscreti
zer(n_bins=7, encode='ordinal', strategy='quantile')),
                                            ("One_Hot_Encoder",              OneHotEncoder
(sparse = False, handle_unknown='ignore'))
                                           ])

#FullTransformerOnEngineCapacity.fit_transform(X[:5000], y[:5000])
#FullTransformerOnEngineCapacity.named_steps["Car_Types_by_Engine_Capacity"].bin_edges_[0]
```

Data To OneHot Transformer On Columns

In [25]:

```
def get_columns_to_one_hot(df):
    return df[['1st_Road_Class', 'Day_of_Week', 'Junction_Detail', 'Light_Conditions', 'Nu
mber_of_Casualties',
               'Number_of_Vehicles', 'Road_Surface_Conditions', 'Road_Type', 'Special_Cond
itions_at_Site',
               'Urban_or_Rural_Area', 'Weather_Conditions', 'Age_Band_of_Driver', 'Hit_Obj
ect_in_Carriageway',
               'Hit_Object_off_Carriageway', 'Sex_of_Driver', 'Skidding_and_Overturning',
               'Vehicle_Manoeuvre', 'Vehicle_Type'
              ]]

DataToOneHotTransformerOnColumns = Pipeline([("Select_Columns",  FunctionTransformer(func=
get_columns_to_one_hot, validate=False)),
                                             ("One_Hot_Encoder", OneHotEncoder(sparse = Fa
lse, handle_unknown='ignore'))])

#DataToOneHotTransformerOnColumns.fit_transform(X[:5000], y[:5000])
```

Feature Union

In [26]:

```
FeatureUnionTransformer = FeatureUnion([
                                        ("FTAgeofVehicle",   FullTransformerOnAgeofVehicle
),
                                        ("FTEngineCapacity", FullTransformerOnEngineCapaci
ty),
                                        ("FTMake",           FullTransformerOnMake),
                                        ("FTSpeedLimit",     FullTransformerOnSpeedLimit),
                                        ("FTTime",           FullTransformerOnTime),
                                        ("OHEColumns",       DataToOneHotTransformerOnColu
mns)])

#FeatureUnionTransformer.fit_transform(X[:5000], y[:5000])
```

In [27]:

```
Full_Transformer = Pipeline([
                            ("Feature_Engineering", FeatureUnionTransformer),
                            ("Min_Max_Transformer", MaxAbsScaler())
                            ])

#Full_Transformer.fit(X[:5000], y[:5000])
```

4. Prediction and submission

In [28]:

```
X_train, X_test, y_train, y_test = split(X, y)
```

4.1 Logistic Regression

In [29]:

```
%%time

clf = LogisticRegression(class_weight = "balanced")

Full_Transformer.fit(X_train)
X_train_transformed = Full_Transformer.transform(X_train)
clf.fit(X_train_transformed, y_train)

X_test_transformed = Full_Transformer.transform(X_test)

y_pred = clf.predict(X_test_transformed)

print('Classification Report:',classification_report(y_test, y_pred))

print('Score:',roc_auc_score(y_test.values, clf.predict_proba(X_test_transformed)[:, 1]))
```

```
Classification Report:               precision    recall  f1-score   support

           0       0.56      0.65      0.60     61646
           1       0.74      0.66      0.70     92735

    accuracy                           0.65    154381
   macro avg       0.65      0.65      0.65    154381
weighted avg       0.67      0.65      0.66    154381

Score: 0.7094754539036983
Wall time: 4min 53s
```

4.2 Random Forest Classifier

In [30]:

```
%%time

clf = RandomForestClassifier(n_estimators=100, n_jobs=3)

Full_Transformer.fit(X_train)
X_train_transformed = Full_Transformer.transform(X_train)
clf.fit(X_train_transformed, y_train)

X_test_transformed = Full_Transformer.transform(X_test)

y_pred = clf.predict(X_test_transformed)

print('Classification Report:',classification_report(y_test, y_pred))

print('Score:',roc_auc_score(y_test.values, clf.predict_proba(X_test_transformed)[:, 1]))
```

```
Classification Report:               precision    recall  f1-score   support

           0       0.79      0.67      0.73     61646
           1       0.80      0.88      0.84     92735

    accuracy                           0.80    154381
   macro avg       0.79      0.78      0.78    154381
weighted avg       0.80      0.80      0.79    154381

Score: 0.8612115000344926
Wall time: 6min 24s
```

4.3 Using the Full Estimator

Logistic Regression

In [31]:

```
LogisticRegression_Full_Estimator = Pipeline([
                                              ("Feature_Engineering", FeatureUnionTransfor
mer),
                                              ("Min_Max_Transformer", MaxAbsScaler()),
                                              ("Clf",                 LogisticRegression(c
lass_weight = "balanced"))
                                             ])

#LogisticRegression_Full_Estimator.fit(X[:5000], y[:5000])
```

In [32]:

```
%%time

LogisticRegression_Full_Estimator.fit(X_train, y_train)
LogisticRegression_Full_Estimator.predict(X_train)
LogisticRegression_Full_Estimator.predict(X_test)

print('Classification Report:' '\n',
      classification_report(y_test, LogisticRegression_Full_Estimator.predict(X_test)))
print('Score:',roc_auc_score(y_test.values, LogisticRegression_Full_Estimator.predict_prob
a(X_test)[:, 1]))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.56      0.65      0.60     61646
           1       0.74      0.66      0.70     92735

    accuracy                           0.65    154381
   macro avg       0.65      0.65      0.65    154381
weighted avg       0.67      0.65      0.66    154381

Score: 0.7094754539036983
Wall time: 5min 7s
```

Random Forest Classifier

In [33]:

```
RandomForest_Full_Estimator = Pipeline([
                                        ("Feature_Engineering", FeatureUnionTransformer),
                                        ("Min_Max_Transformer", MaxAbsScaler()),
                                        ("Clf",                 RandomForestClassifier(n_e
stimators=100, n_jobs=3))
                                       ])

#RandomForest_Full_Estimator.fit(X[:5000], y[:5000])
```

In [34]:

```
%%time

RandomForest_Full_Estimator.fit(X_train, y_train)
RandomForest_Full_Estimator.predict(X_train)
RandomForest_Full_Estimator.predict(X_test)

print('Classification Report:' '\n',
      classification_report(y_test, RandomForest_Full_Estimator.predict(X_test)))
print('Score:',roc_auc_score(y_test.values, RandomForest_Full_Estimator.predict_proba(X_te
st)[:, 1]))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.79      0.67      0.73     61646
           1       0.80      0.88      0.84     92735

    accuracy                           0.80    154381
   macro avg       0.79      0.78      0.78    154381
weighted avg       0.80      0.80      0.79    154381

Score: 0.860662337048942
Wall time: 7min 6s
```

In [ ]:

```

```