In [22]:
```
import pandas as pd
import numpy as np
import seaborn as sns

import matplotlib.pyplot as plt

import datetime as dt

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

In [23]:
```
df = pd.read_csv('Data.csv', index_col=0)
df.head(10)
```

Out[23]:

	ID	time	lum	agg	int	atm	col	lat	long	dep	...	situation	school
0	201600000001	14.0	1	2	1	8.0	3.0	0.0	0.0	590	...	1.0	0.0
1	201600000002	18.0	1	2	6	1.0	6.0	0.0	0.0	590	...	1.0	0.0
2	201600000003	19.0	1	1	1	1.0	6.0	0.0	0.0	590	...	3.0	99.0
3	201600000004	19.0	2	2	1	7.0	3.0	0.0	0.0	590	...	1.0	99.0
4	201600000005	11.0	1	2	3	1.0	3.0	0.0	0.0	590	...	1.0	3.0
5	201600000006	11.0	1	2	1	7.0	6.0	0.0	0.0	590	...	1.0	99.0
6	201600000007	11.0	1	2	1	7.0	2.0	0.0	0.0	590	...	1.0	99.0
7	201600000008	19.0	2	1	1	1.0	1.0	0.0	0.0	590	...	1.0	0.0
8	201600000009	19.0	1	2	1	1.0	3.0	0.0	0.0	590	...	1.0	99.0
9	201600000010	10.0	1	1	1	9.0	6.0	0.0	0.0	590	...	1.0	0.0

10 rows × 29 columns

In [24]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 839985 entries, 0 to 839984
Data columns (total 29 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   ID         839985 non-null  int64  
 1   time       839985 non-null  float64
 2   lum        839985 non-null  int64  
 3   agg        839985 non-null  int64  
 4   int        839985 non-null  int64  
 5   atm        839930 non-null  float64
 6   col        839974 non-null  float64
 7   lat        362471 non-null  float64
 8   long       362467 non-null  object 
 9   dep        839985 non-null  int64  
 10  road_cat   839984 non-null  float64
 11  road_num   780914 non-null  object 
 12  traf_reg   839187 non-null  float64
 13  num_lanes  838195 non-null  float64
 14  res_lane   838345 non-null  float64
 15  long_prof  838924 non-null  float64
 16  shape      838909 non-null  float64
 17  surf       838968 non-null  float64
 18  infra      838707 non-null  float64
 19  situation  838983 non-null  float64
 20  school     838709 non-null  float64
 21  crit_age   839985 non-null  int64  
 22  ped        839985 non-null  int64  
 23  dead_age   839985 non-null  int64  
 24  num_us     839985 non-null  int64  
 25  sev        839985 non-null  int64  
 26  date       839985 non-null  object 
 27  weekend    839985 non-null  int64  
 28  holiday    839985 non-null  float64
dtypes: float64(15), int64(11), object(3)
memory usage: 192.3+ MB
```

In [25]:
```
df.describe()
```

Out[25]:

	ID	time	lum	agg	int	
count	8.399850e+05	839985.000000	839985.000000	839985.000000	839985.000000	8399
mean	2.010011e+11	13.559365	1.912588	1.685924	1.694066	
std	3.458009e+08	5.411096	1.517900	0.464147	1.510792	
min	2.005000e+11	0.000000	1.000000	1.000000	0.000000	
25%	2.007000e+11	10.000000	1.000000	1.000000	1.000000	
50%	2.010000e+11	14.000000	1.000000	2.000000	1.000000	
75%	2.013000e+11	18.000000	3.000000	2.000000	2.000000	
max	2.016001e+11	23.000000	5.000000	2.000000	9.000000	

8 rows × 26 columns

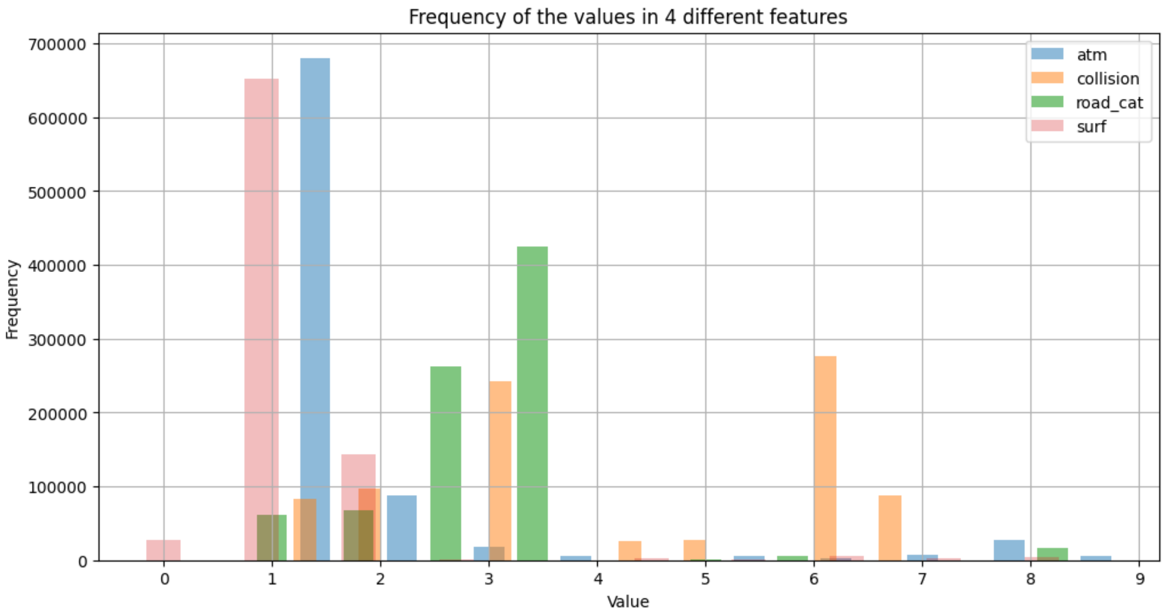
In [26]:
```
df.drop(['lat', 'long', 'road_num'], axis=1, inplace=True)
```

In [27]:
```
print('Missing values in atm:', df["atm"].isna().sum(),'\n'
    'Missing values in collision:', df["col"].isna().sum(), '\n'
    'Missing values in road_cat:', df["road_cat"].isna().sum(),'\n'
    'Missing values in surf:', df["surf"].isna().sum())
```

```
Missing values in atm: 55
Missing values in collision: 11
Missing values in road_cat: 1
Missing values in surf: 1017
```

In [28]:
```
df['atm'].hist(alpha=0.5, rwidth=0.35, align='mid', figsize=(12,6), label='atm')
df['col'].hist(alpha=0.5, rwidth=0.35, align='mid', label='collision')
df['road_cat'].hist(alpha=0.6, rwidth=0.35, align='left', label='road_cat')
df['surf'].hist(alpha=0.3,rwidth=0.35, align='left', label='surf')
plt.title('Frequency of the values in 4 different features', size=12)
plt.xticks(range(10))
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.legend()
```

Out[28]: <matplotlib.legend.Legend at 0x3ff5674dd80>



In [29]:
```
df['atm'].fillna(9, inplace=True)
df['col'].fillna(6, inplace=True)
df['road_cat'].fillna(9, inplace=True)
df['surf'].fillna(9, inplace=True)
df['surf'].replace(0,9, inplace=True)
df.surf.value_counts()
```

Out[29]:
```
1.0    652322
2.0    143254
9.0     32498
7.0      5474
5.0      2643
8.0      2159
3.0       861
6.0       466
4.0       308
Name: surf, dtype: int64
```

In [30]:
```
df[['traf_reg', 'num_lanes','res_lane', 'long_prof', 'shape', 'infra', 'situatic
```

Out[30]:

	traf_reg	num_lanes	res_lane	long_prof	shape	
count	839187.000000	838195.000000	838345.000000	838924.000000	838909.000000	838
mean	1.855246	2.039593	0.130675	1.135474	1.198732	
std	0.720949	1.550779	0.555434	0.620295	0.722200	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	2.000000	2.000000	0.000000	1.000000	1.000000	
50%	2.000000	2.000000	0.000000	1.000000	1.000000	
75%	2.000000	2.000000	0.000000	1.000000	1.000000	
max	4.000000	99.000000	3.000000	4.000000	4.000000	

In [31]:
```
df.drop(['infra', 'res_lane'], axis=1, inplace=True)
```

In [32]:
```
df['num_lanes'].value_counts()
```

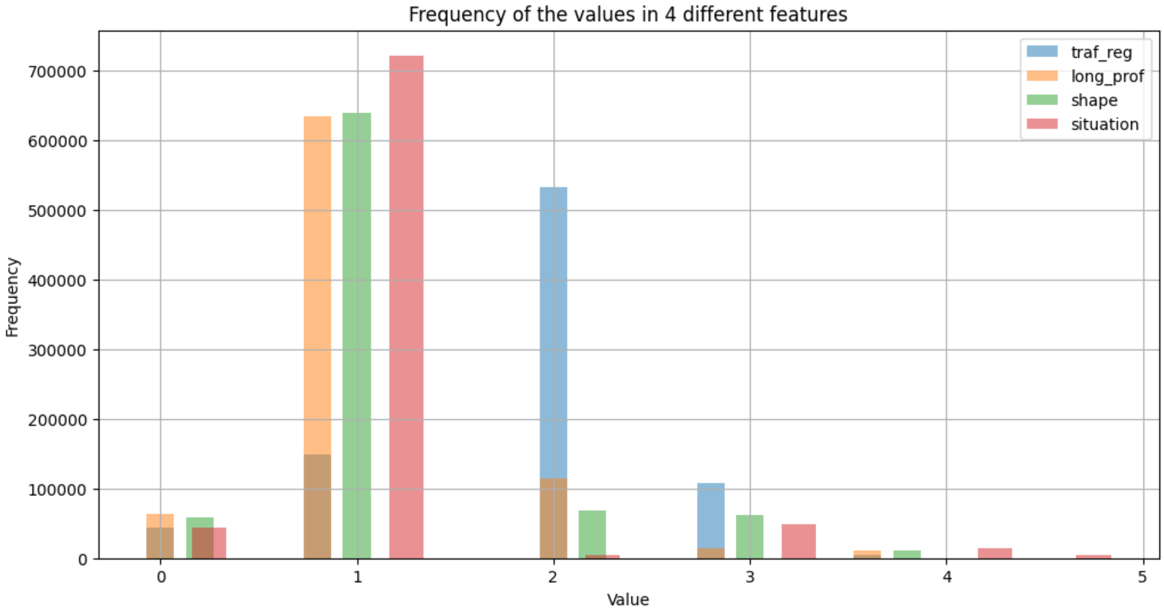
Out[32]:
```
2.0     464716
0.0     102796
1.0     101345
4.0      76934
3.0      66252
6.0      13945
5.0       7839
8.0       2332
7.0        840
10.0       407
20.0       241
50.0       158
9.0        148
11.0        32
12.0        32
40.0        30
30.0        27
13.0        15
25.0        14
21.0        11
26.0         9
15.0         8
90.0         7
24.0         6
14.0         6
22.0         5
70.0         5
60.0         3
31.0         3
16.0         2
53.0         2
45.0         2
27.0         2
17.0         2
65.0         1
84.0         1
39.0         1
54.0         1
29.0         1
62.0         1
99.0         1
42.0         1
41.0         1
36.0         1
44.0         1
33.0         1
52.0         1
28.0         1
91.0         1
86.0         1
76.0         1
23.0         1
18.0         1
Name: num_lanes, dtype: int64
```

In [33]:
```
df.num_lanes.fillna(0, inplace=True)
df['num_lanes'] = df['num_lanes'].apply(lambda x: 2 if x>6 or x==0 else x)
df.num_lanes.value_counts()
```

Out[33]:
```
2.0    573670
1.0    101345
4.0     76934
3.0     66252
6.0     13945
5.0      7839
Name: num_lanes, dtype: int64
```

In [34]:
```
df['traf_reg'].hist(alpha=0.5, rwidth=0.35, align='left', figsize=(12,6), label=
df['long_prof'].hist(alpha=0.5,rwidth=0.35, align='left', label='long_prof')
df['shape'].hist(alpha=0.5,rwidth=0.35, align='mid', label='shape')
df['situation'].hist(alpha=0.5,rwidth=0.35, align='mid', label='situation')
plt.title('Frequency of the values in 4 different features', size=12)
plt.xticks(range(6))
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.legend()
```

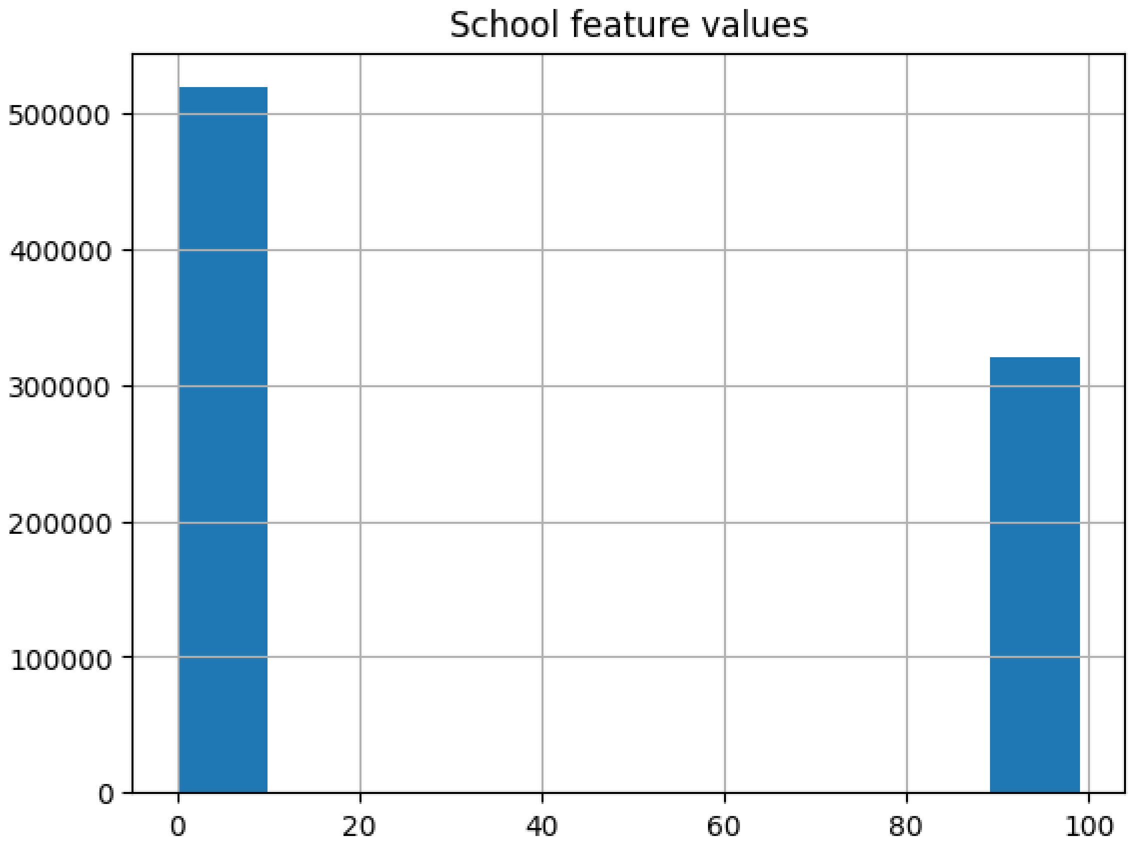
Out[34]: <matplotlib.legend.Legend at 0x3ff564ee620>



In [35]:
```
df['traf_reg'].fillna(0, inplace=True)
df['traf_reg'] = df['traf_reg'].replace(0,2)

df['long_prof'].fillna(0, inplace=True)
df['long_prof'] = df['long_prof'].replace(0,1)

df['shape'].fillna(0, inplace=True)
df['shape'] = df['shape'].replace(0,1)

df['situation'].fillna(0, inplace=True)
df['situation'] = df['situation'].replace(0,1)
```

In [36]:
```
df.school.describe(), df.school.hist()
plt.title('School feature values')
```

Out[36]: Text(0.5, 1.0, 'School feature values')



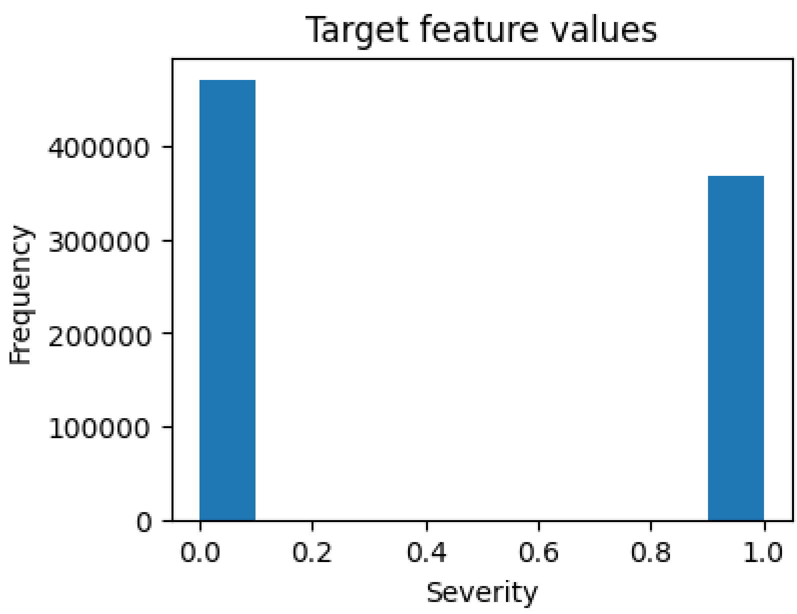
In [37]:
```
df.school.fillna(0, inplace=True)
df['school'] = df.school.apply(lambda x:1 if x>0 else 0)
```

In [38]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 839985 entries, 0 to 839984
Data columns (total 24 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   ID         839985 non-null  int64  
 1   time       839985 non-null  float64
 2   lum        839985 non-null  int64  
 3   agg        839985 non-null  int64  
 4   int        839985 non-null  int64  
 5   atm        839985 non-null  float64
 6   col        839985 non-null  float64
 7   dep        839985 non-null  int64  
 8   road_cat   839985 non-null  float64
 9   traf_reg   839985 non-null  float64
 10  num_lanes  839985 non-null  float64
 11  long_prof  839985 non-null  float64
 12  shape      839985 non-null  float64
 13  surf       839985 non-null  float64
 14  situation  839985 non-null  float64
 15  school     839985 non-null  int64  
 16  crit_age   839985 non-null  int64  
 17  ped        839985 non-null  int64  
 18  dead_age   839985 non-null  int64  
 19  num_us     839985 non-null  int64  
 20  sev        839985 non-null  int64  
 21  date       839985 non-null  object 
 22  weekend    839985 non-null  int64  
 23  holiday    839985 non-null  float64
dtypes: float64(11), int64(12), object(1)
memory usage: 160.2+ MB
```

In [39]:
```
df.sev.plot.hist(figsize=(4,3))
plt.title('Target feature values')
plt.xlabel('Severity')
plt.ylabel('Frequency')
print('Accidents classified in each level of severity:')
print(df.sev.value_counts())
```

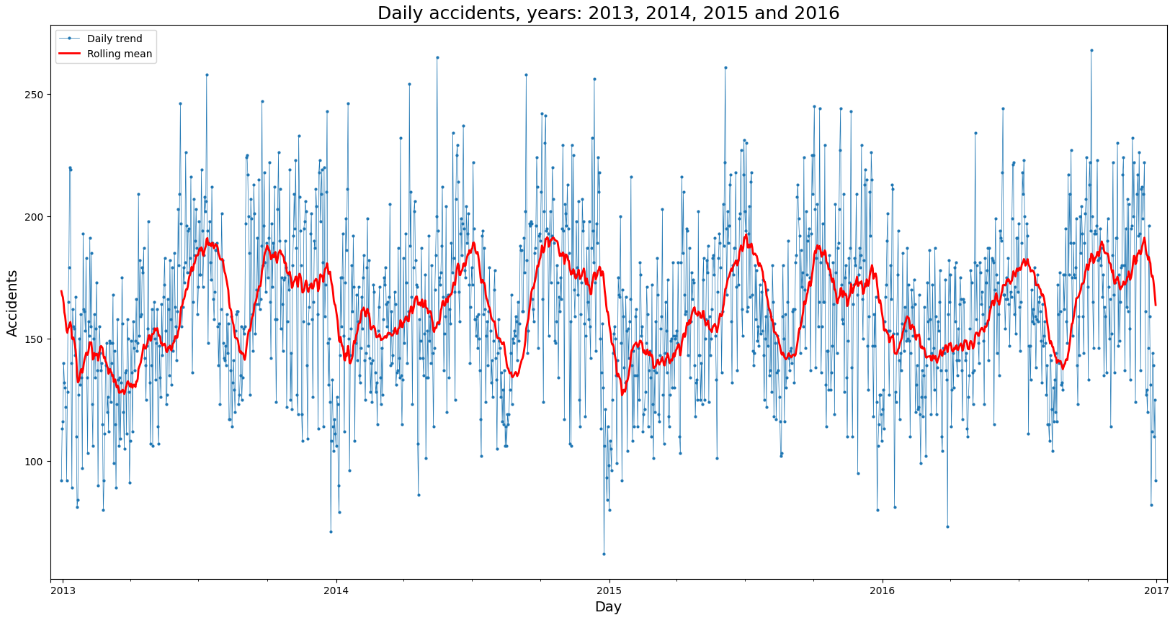
```
Accidents classified in each level of severity:
0    471695
1    368290
Name: sev, dtype: int64
```



In [40]:
```
df['date'] = pd.to_datetime(df['date'], format='%Y-%m-%d')
date = df[['ID','sev', 'date']]
date.date
```

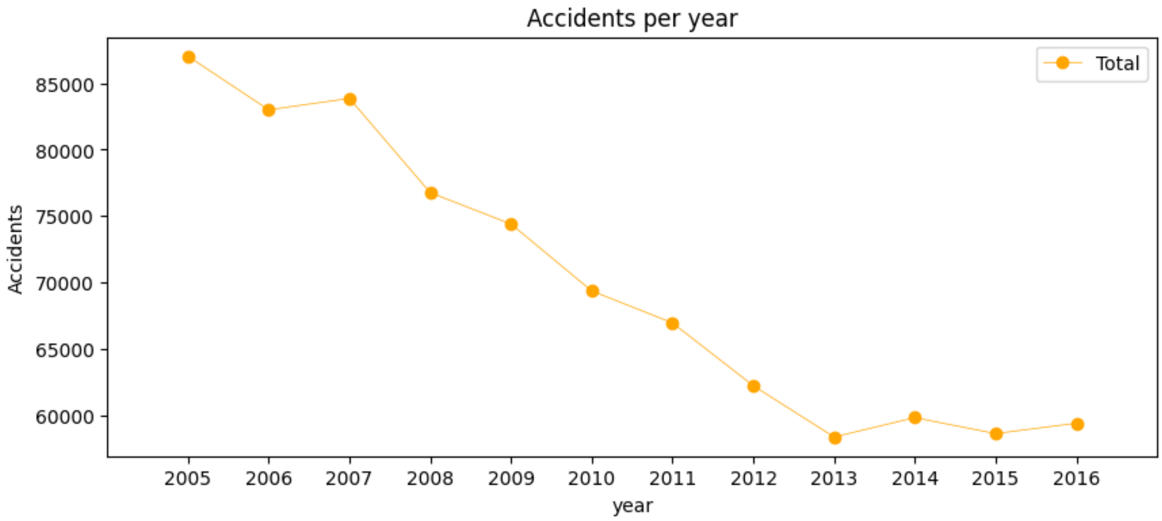
Out[40]:
```
0        2016-02-01
1        2016-03-16
2        2016-07-13
3        2016-08-15
4        2016-12-23
            ...
839980   2005-12-21
839981   2005-12-23
839982   2005-12-26
839983   2005-12-27
839984   2005-12-31
Name: date, Length: 839985, dtype: datetime64[ns]
```

In [41]:
```
date['year'] = df.date.dt.year
date['month'] = df.date.dt.month
date['weekday'] = df.date.dt.weekday
high_sev = date[date['sev']==1]

season = date[['date', 'ID']].groupby('date').count()
season['rolling'] = season.ID.rolling(window=30).mean()
season['ID'][365*8:].plot(figsize=(20,10), marker='o', markersize=2, linewidth=0
season['rolling'][365*8:].plot(color='r', linewidth=2, label='Rolling mean')
plt.title('Daily accidents, years: 2013, 2014, 2015 and 2016', size=18)
plt.xlabel('Day', size=14)
plt.ylabel('Accidents', size=14)

t0 = dt.datetime.strptime('2012-12-15', '%Y-%m-%d')
t1 = dt.datetime.strptime('2017-01-15', '%Y-%m-%d')

plt.xlim(t0,t1)
plt.legend()
plt.show()
```



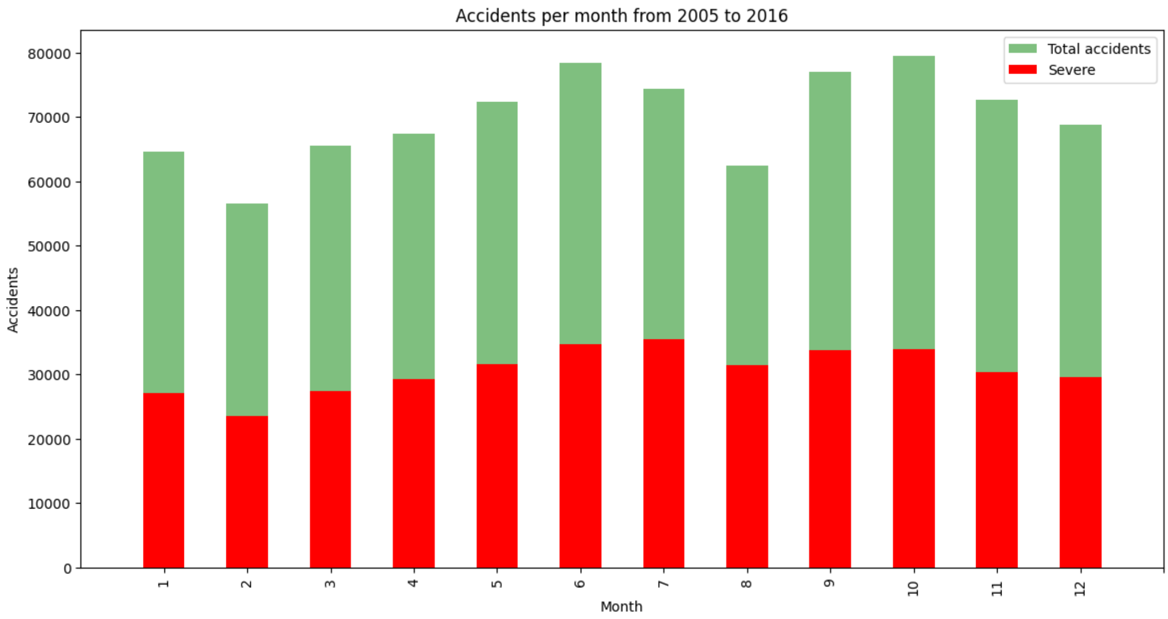
In [42]:
```
yearly = date[['year', 'ID']].groupby('year').count()
yearly['ID'].plot.line(figsize=(10,4), marker='o', linewidth=0.5, color='orange'
plt.title('Accidents per year')
plt.xticks(range(2005,2017))
plt.xlim(2004,2017)
plt.ylabel('Accidents')
plt.legend()
```

Out[42]: <matplotlib.legend.Legend at 0x3ff566b3880>



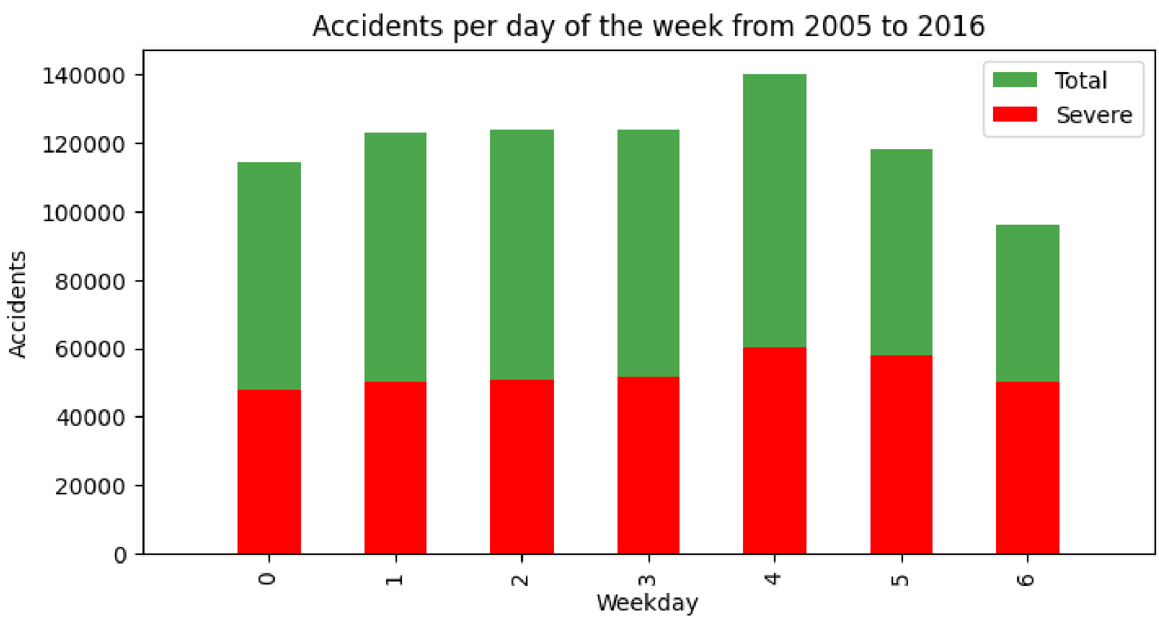
In [43]:
```
monthly = date[['month', 'ID']].groupby(['month']).count()
monthly['high_sev'] = high_sev[['month', 'ID']].groupby(['month']).count()


monthly['ID'].plot.bar(figsize=(14,7), alpha=0.5, color='g', label='Total accide
monthly['high_sev'].plot.bar(color='r', label='Severe')

plt.title('Accidents per month from 2005 to 2016')
plt.xticks(range(13))
plt.xlim(-1,12)
# plt.ylim(50000,85000)
plt.xlabel('Month')
plt.ylabel('Accidents')
plt.legend()
```

Out[43]: <matplotlib.legend.Legend at 0x3ff5635c550>



In [44]:
```
weekday = date[['weekday', 'ID']].groupby('weekday').count()
weekday['high_sev'] = high_sev[['weekday', 'ID']].groupby(['weekday']).count()
weekday['ID'].plot.bar(figsize=(8,4), alpha=0.7, color='g', label='Total')
weekday['high_sev'].plot.bar(color='r', label='Severe')

plt.title('Accidents per day of the week from 2005 to 2016')
plt.xticks(range(7))
plt.xlim(-1,7)
# plt.ylim(75000,150000)
plt.xlabel('Weekday')
plt.ylabel('Accidents')
plt.legend()
```

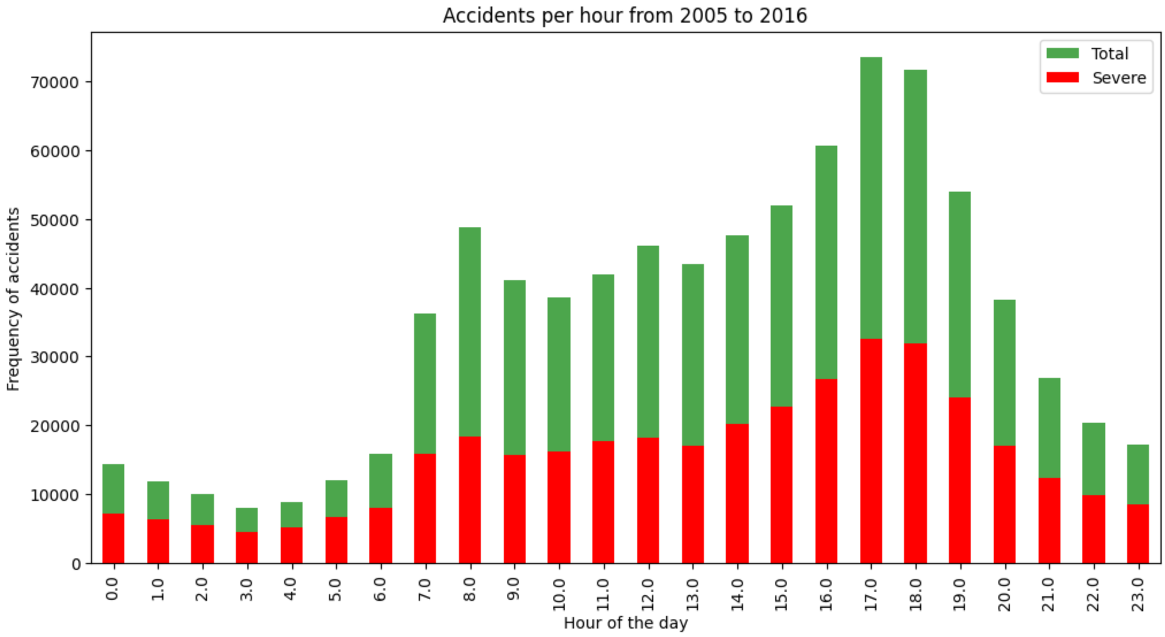
Out[44]: <matplotlib.legend.Legend at 0x3ff55b5df30>



In [45]:
```
hourly = df[['ID', 'time']].groupby('time').count()
hourly['high_sev'] = df[df.sev==1][['ID', 'time']].groupby('time').count()
hourly['ID'].plot.bar(figsize=(12,6), alpha=0.7, color='g', label='Total')
hourly['high_sev'].plot.bar(color='r', label='Severe')
```

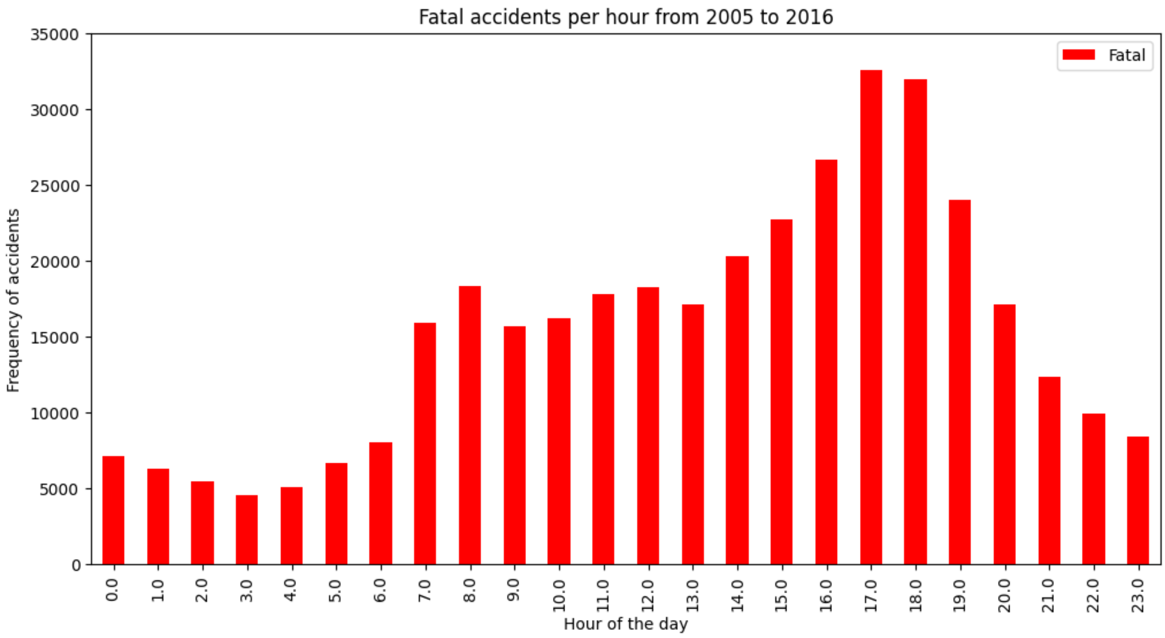
```
plt.xticks(range(24))
plt.title('Accidents per hour from 2005 to 2016')
plt.xlabel('Hour of the day')
plt.ylabel('Frequency of accidents')
plt.legend()
# df.time.value_counts()
# hourly.ID.value_counts()
hourly['ID'].sum()
```

Out[45]: 839985



In [46]:
```
hourly['high_sev'].plot.bar(figsize=(12,6),color='r', label='Fatal')
plt.xticks(range(24))
plt.ylim((0,35000))
plt.title('Fatal accidents per hour from 2005 to 2016')
plt.xlabel('Hour of the day')
plt.ylabel('Frequency of accidents')
plt.legend()
```

Out[46]: <matplotlib.legend.Legend at 0x3ff55bd25f0>



In [47]:
```
noon_morn_severe = hourly.high_sev.loc[0:6].sum()+hourly.high_sev.loc[21:23].sum
day_severe = hourly.high_sev.loc[7:20].sum()
noon_morn = hourly.ID.loc[0:6].sum()+hourly.ID.loc[21:23].sum()
day = hourly.ID.loc[7:20].sum()
noon_morn_prop = (noon_morn_severe/noon_morn)*100
day_prop = (day_severe/day)*100
print('The percentage of severe accidents from 9pm to 6am is {0:0.2f}% of the to
     while the percentage of deathly accidents from 7am to 8pm is {1:2.2f}%.'.fo
```

```
The percentage of severe accidents from 9pm to 6am is 50.67% of the total amount
of accidents ocurring between this hours,     while the percentage of deathly acc
idents from 7am to 8pm is 42.41%.
```

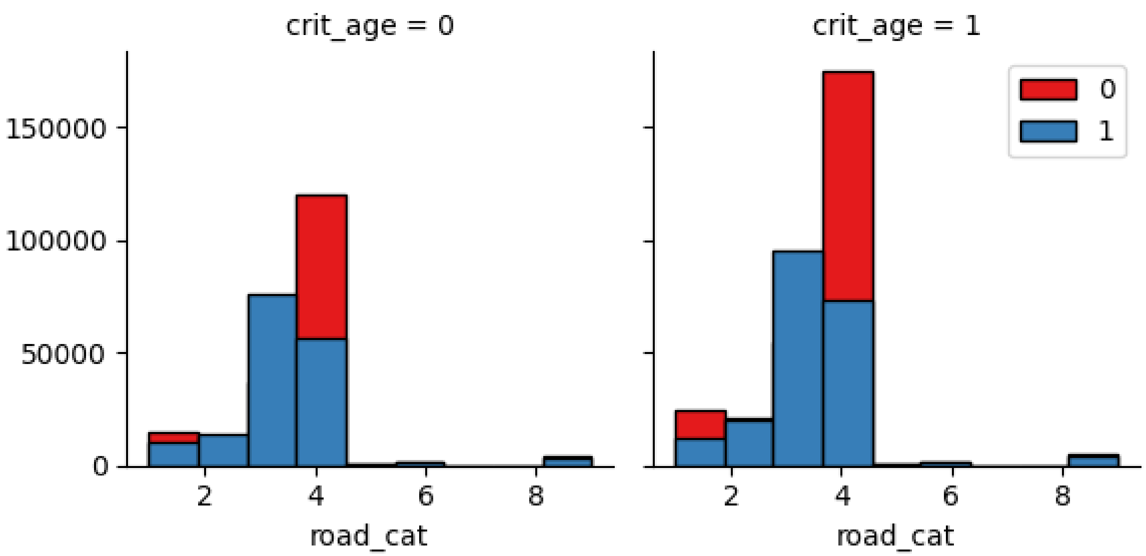
In [48]:
```
df['month'] = df.date.dt.month
df['day'] = df.date.dt.day
df.day.value_counts
```

Out[48]:
```
<bound method IndexOpsMixin.value_counts of 0          1
1          16
2          13
3          15
4          23
           ..
839980     21
839981     23
839982     26
839983     27
839984     31
Name: day, Length: 839985, dtype: int64>
```

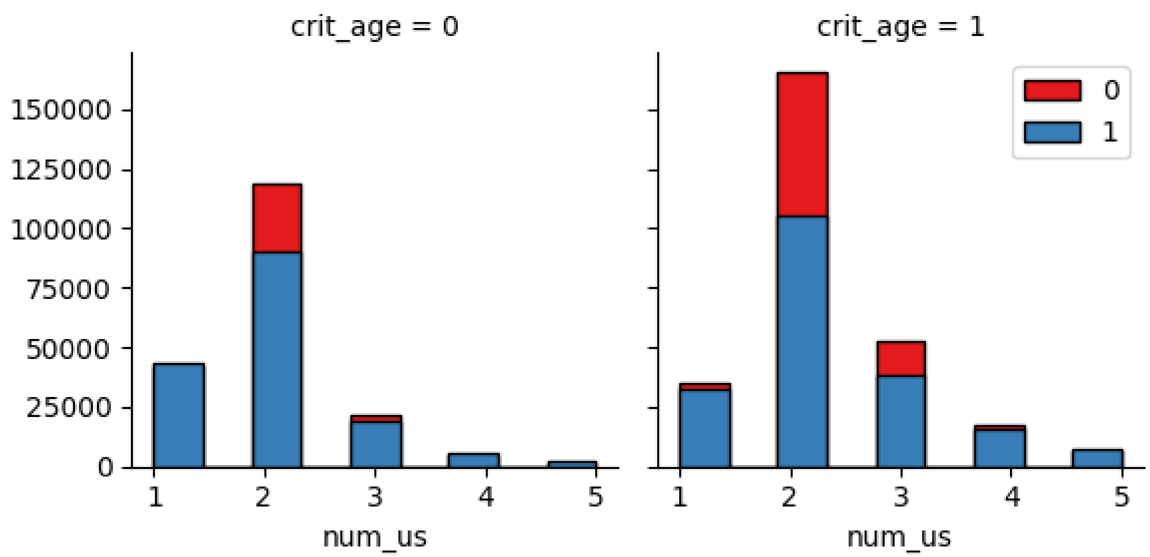
In [49]:
```
df[['sev','lum', 'agg', 'int', 'atm',
    'col', 'dep', 'road_cat', 'traf_reg',
    'num_lanes', 'long_prof', 'shape', 'surf',
    'situation', 'school', 'crit_age','dead_age', 'ped',
    'num_us', 'weekend', 'holiday', 'month', 'day']].corr()['sev'].sort_values(a
```

Out[49]:
```
sev          1.000000
shape        0.144514
situation    0.128954
weekend      0.077594
traf_reg     0.076691
long_prof    0.069781
dead_age     0.048087
atm          0.048012
num_us       0.027533
col          0.026740
holiday      0.021744
month        0.008851
lum          0.002701
day          0.002161
surf         0.000874
ped         -0.005999
school      -0.025260
crit_age    -0.038168
int         -0.062982
road_cat    -0.100728
num_lanes   -0.101300
dep         -0.105850
agg         -0.277563
Name: sev, dtype: float64
```

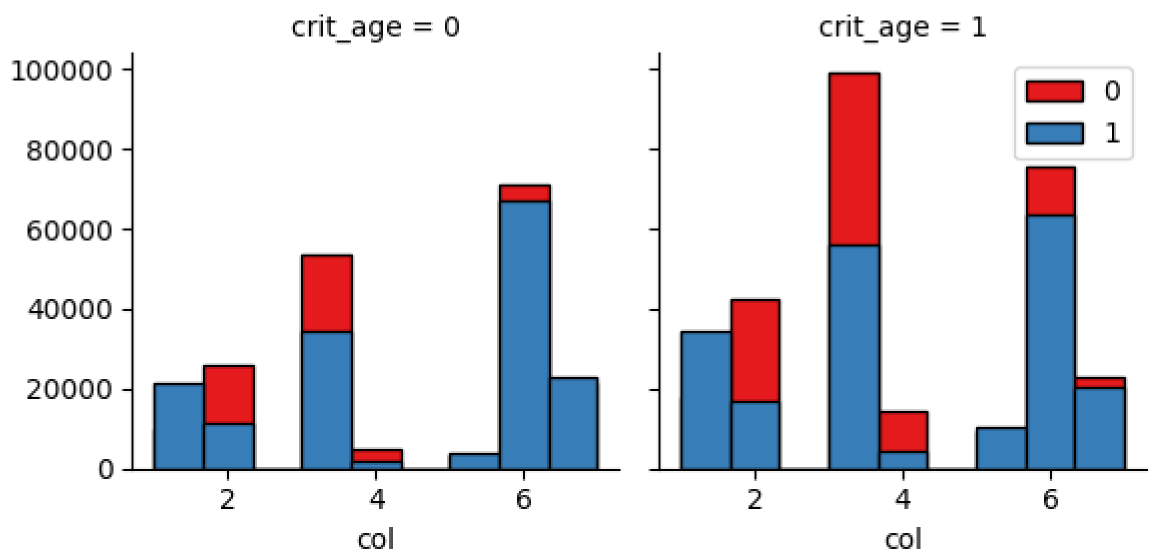
In [50]:
```
bins = np.linspace(df.atm.min(), df.atm.max(), 10)
g = sns.FacetGrid(df, col="crit_age", hue="sev", palette="Set1", col_wrap=2)
g.map(plt.hist,'road_cat', bins=bins, ec="k")
g.axes[-1].legend()
plt.show()
```



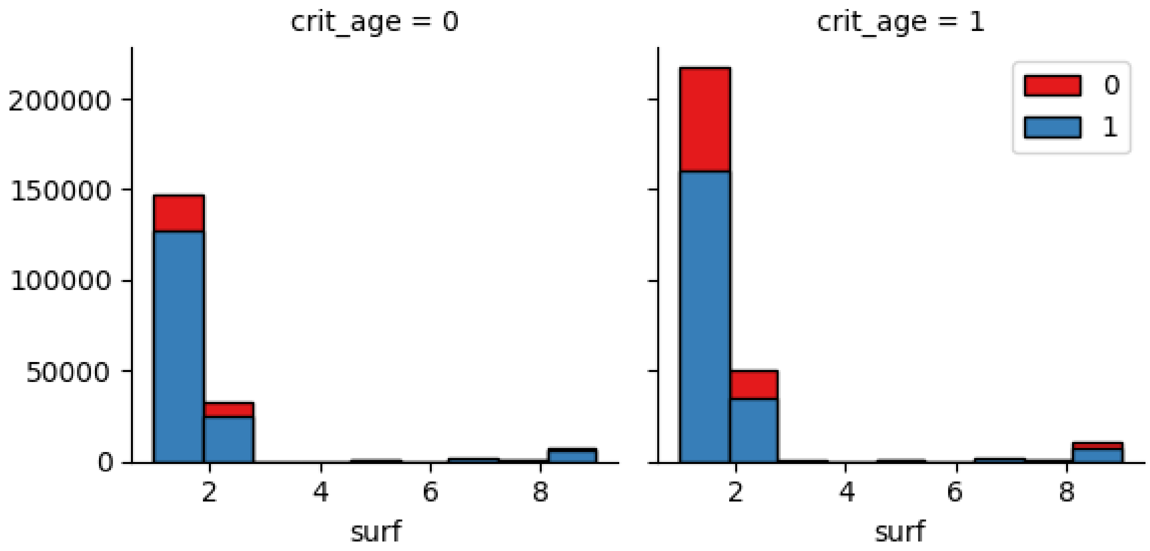
In [51]:
```
bins = np.linspace(df.lum.min(), df.lum.max(), 10)
g = sns.FacetGrid(df, col="crit_age", hue="sev", palette="Set1", col_wrap=2)
g.map(plt.hist, 'num_us', bins=bins, ec="k")
g.axes[-1].legend()
plt.show()
```



In [52]:
```
bins = np.linspace(df.col.min(), df.col.max(), 10)
g = sns.FacetGrid(df, col="crit_age", hue="sev", palette="Set1", col_wrap=2)
g.map(plt.hist, 'col', bins=bins, ec="k")
g.axes[-1].legend()
plt.show()
```

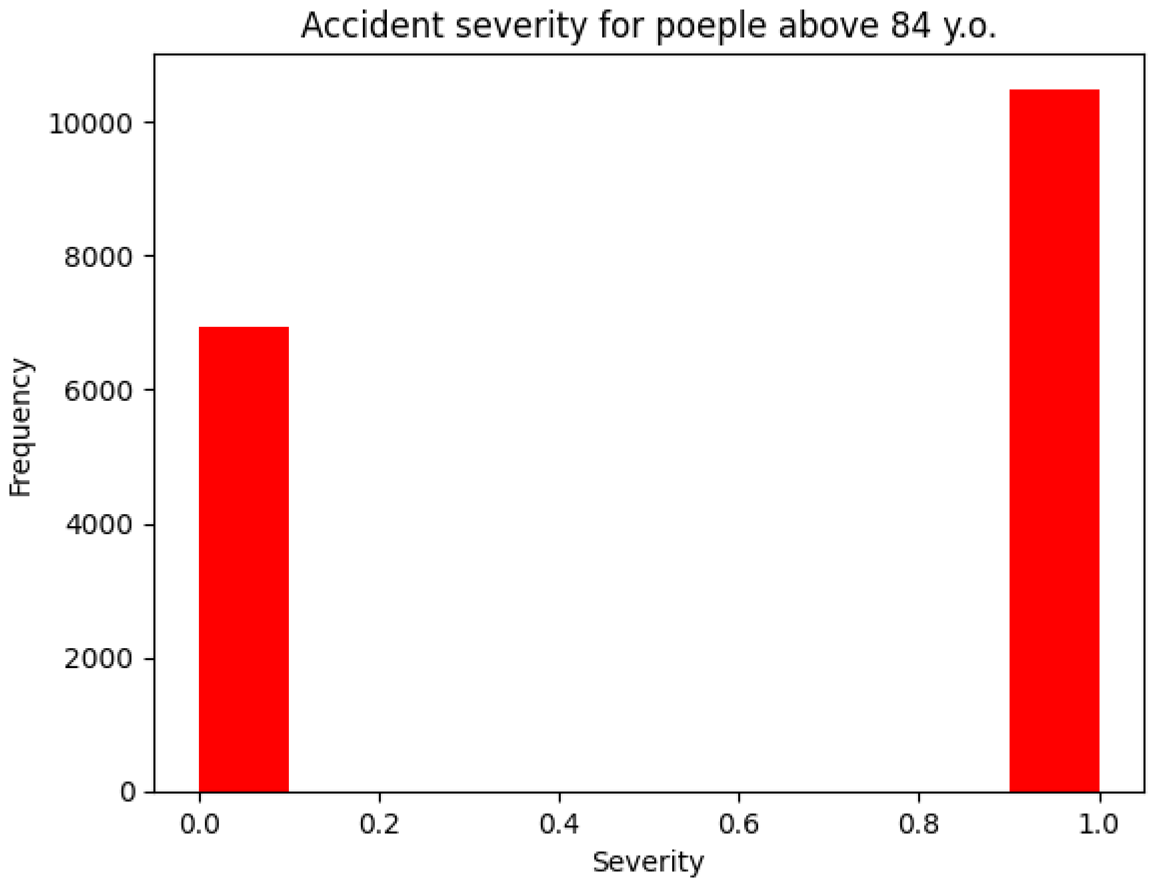


In [53]:
```
bins = np.linspace(df.surf.min(), df.surf.max(), 10)
g = sns.FacetGrid(df, col="crit_age", hue="sev", palette="Set1", col_wrap=2)
g.map(plt.hist, 'surf', bins=bins, ec="k")
g.axes[-1].legend()
plt.show()
```



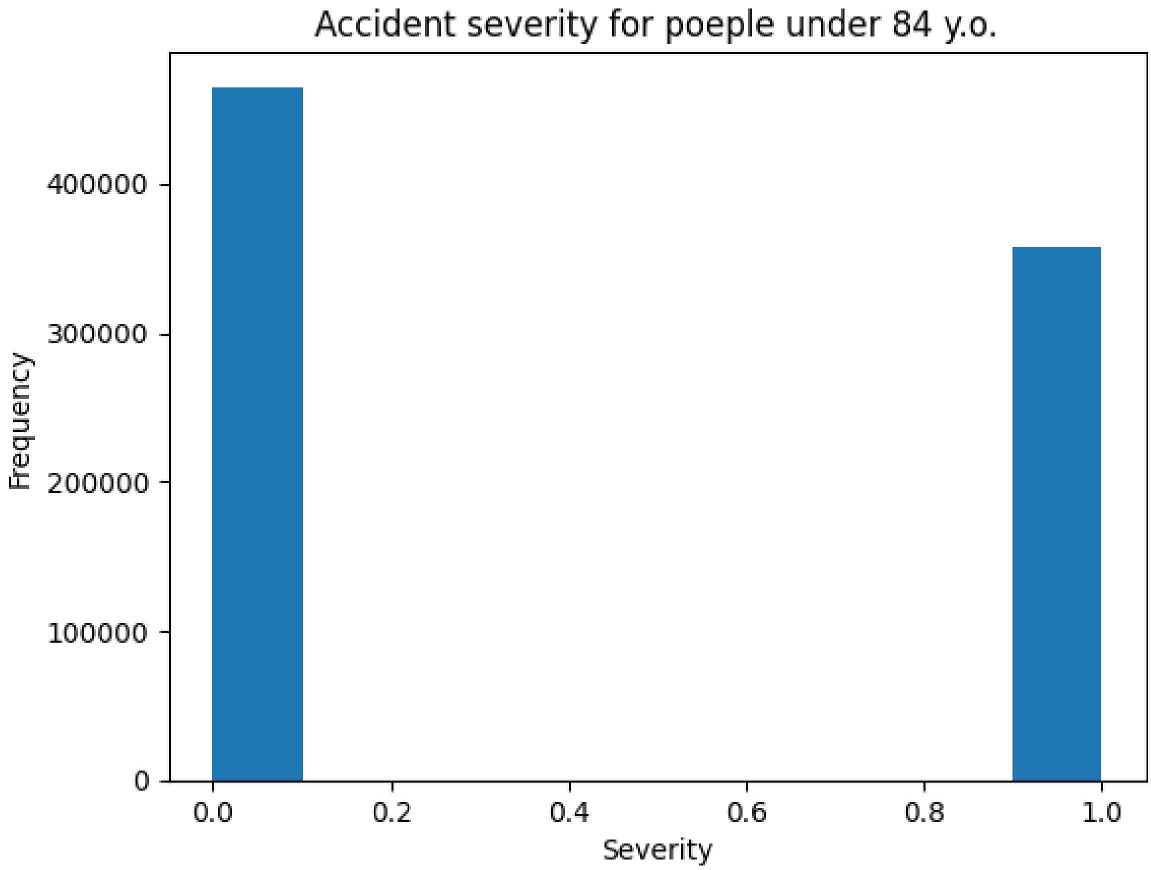
In [54]:
```
df['sev'][df['dead_age']==1].plot.hist(color='r')
# plt.xlabel('Hour of the day')
plt.title('Accident severity for poeple above 84 y.o.')
plt.xlabel('Severity')
```

Out[54]: Text(0.5, 0, 'Severity')



In [55]:
```
df['sev'][df['dead_age']==0].plot.hist()
plt.title('Accident severity for poeple under 84 y.o.')
plt.xlabel('Severity')
```

Out[55]: Text(0.5, 0, 'Severity')



In [56]:
```
df.drop(['ID', 'date'], axis=1, inplace=True)
```

In [57]:
```
#Some feature's values range from 1 to 9 while others just go either for 1 or 2,
#Normalizing the data makes that any feature has more influence in the result th

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# X = df.drop('sev', axis=1)
# X = StandardScaler().fit(X).transform(X)

xtrain, xtest, ytrain, ytest = train_test_split(df.drop('sev', axis=1), df['sev'
xtrain, xval, ytrain, yval = train_test_split(xtrain, ytrain, test_size=0.2)


print('Size of training set:', xtrain.shape[0],'\n'
      'Size of test set:',xtest.shape[0],'\n'
      'Size of evaluation set:', xval.shape[0])
```

```
Size of training set: 537590
Size of test set: 167997
Size of evaluation set: 134398
```

In [58]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

In [59]:
```
#Evaluation Metrics
import time
```

```
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import precision_score, recall_score, roc_curve
```

In [110…

```
import time
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

t0 = time.time()
tree = DecisionTreeClassifier(criterion='entropy')
tree.fit(xtrain, ytrain)
print('Time taken :', time.time() - t0)
yhat = tree.predict(xval)
score_tree = accuracy_score(yval, yhat)
print('Accuracy:', score_tree)
report = classification_report(yval, yhat)
print(report)
```

```
Time taken : 3.3008053302764893
Accuracy: 0.6339156832690963
              precision    recall  f1-score   support

           0       0.67      0.67      0.67     75454
           1       0.58      0.58      0.58     58944

    accuracy                           0.63    134398
   macro avg       0.63      0.63      0.63    134398
weighted avg       0.63      0.63      0.63    134398
```

In [61]:

```
t0=time.time()
model_rf = RandomForestClassifier(n_estimators=100,criterion='entropy',random_st
model_rf.fit(xtrain,ytrain)
print('Time taken :' , time.time()-t0)
yhat = model_rf.predict(xval)
score_rf = accuracy_score(yval,yhat)
print('Accuracy :',score_rf)
```

```
Time taken : 77.53458333015442
Accuracy : 0.7196535662733077
```

In [62]:

```
importances = pd.DataFrame({'feature':df.drop('sev', axis=1).columns,'importance
importances = importances.sort_values('importance',ascending=False).set_index('f
importances
```

Out[62]:

feature	importance
dep	0.164
day	0.145
time	0.116
month	0.104
road_cat	0.054
col	0.046
num_us	0.044
num_lanes	0.035
int	0.033
agg	0.033
traf_reg	0.029
atm	0.025
lum	0.024
long_prof	0.022
surf	0.022
crit_age	0.021
shape	0.019
school	0.018
weekend	0.015
situation	0.014
ped	0.008
holiday	0.005
dead_age	0.004

In [63]:

```
xtrain = pd.DataFrame(xtrain)
xtrain.drop(['dead_age', 'holiday', 'ped', 'situation', 'weekend', 'school', 'sh
xval.drop(['dead_age', 'holiday', 'ped', 'situation', 'weekend', 'school', 'shap
xtest.drop(['dead_age', 'holiday', 'ped', 'situation', 'weekend', 'school', 'sha
```

In [64]:

```
#RF 2:
#number of features reduced from 23 to 13

t0=time.time()
model_rf = RandomForestClassifier(n_estimators=100,criterion='entropy',random_st
model_rf.fit(xtrain,ytrain)
print('Time taken :' , time.time()-t0)
yhat = model_rf.predict(xval)
```

```
score_rf = accuracy_score(yval,yhat)
print('Accuracy :',score_rf)
```

```
Time taken : 82.73126864433289
Accuracy : 0.705910802244081
```

In [65]:
```
#RF 3:

#number of decision trees reduced from 100 to 50
#Limiting the number of features to look at when creating the next split to 5
#Limiting the max depth of the tree to 10


t0=time.time()
model_rf = RandomForestClassifier(n_estimators=50, max_features=5, max_depth =10
model_rf.fit(xtrain,ytrain)
print('Time taken :' , time.time()-t0)
yhat = model_rf.predict(xval)
score_rf = accuracy_score(yval,yhat)
print('Accuracy :',score_rf)
```

```
Time taken : 12.093673944473267
Accuracy : 0.7143930713254661
```

In [66]:
```
#RF 4:

#number of decision trees reduced from 50 to 10
#Limiting the number of features to look at when creating the next split to 8
#Limiting the max depth of the tree to 12


t0=time.time()
model_rf = RandomForestClassifier(n_estimators=10, max_features=8, max_depth =12
model_rf.fit(xtrain,ytrain)
print('Time taken :' , time.time()-t0)
yhat = model_rf.predict(xval)
score_rf = accuracy_score(yval,yhat)
print('Accuracy :',score_rf)
```

```
Time taken : 5.740341424942017
Accuracy : 0.7229423056890728
```

In [67]:
```
#Evaluation

t0=time.time()
model_rf = RandomForestClassifier(n_estimators=10, max_features=8, max_depth =12
model_rf.fit(xtrain,ytrain)
t_rf = time.time()-t0
print('Time taken :' , t_rf)
yhat_rf = model_rf.predict(xtest)
c_rf = classification_report(ytest,yhat_rf)
prec_rf = precision_score(ytest, yhat_rf)
rec_rf = recall_score(ytest, yhat_rf)
print('classification_report:' , c_rf)
print('precision_score' , prec_rf)
print('recall_score' , rec_rf)
```

```
classification_report:               precision    recall  f1-score   support

           0       0.73      0.82      0.77     94297
           1       0.72      0.60      0.66     73700

    accuracy                           0.72    167997
   macro avg       0.72      0.71      0.71    167997
weighted avg       0.72      0.72      0.72    167997

precision_score 0.7220879495206735
recall_score 0.6040162822252374
```

In [115…

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Assuming xtrain, ytrain, xval, and yval are defined elsewhere

acc = np.zeros(6)
C_values = [0.5, 0.1, 0.01, 0.001, 10, 100]

for i, c in enumerate(C_values):
    lr = LogisticRegression(C=c, solver='liblinear').fit(xtrain, ytrain)
    yhat = lr.predict(xval)
    acc[i] = accuracy_score(yval, yhat)
    print(f'C = {c}, Accuracy = {acc[i]}')

acc
```

```
C = 0.5, Accuracy = 0.6584175359752378
C = 0.1, Accuracy = 0.6585068230181996
C = 0.01, Accuracy = 0.6587895653209125
C = 0.001, Accuracy = 0.659682435750532
C = 10, Accuracy = 0.6584845012574592
C = 100, Accuracy = 0.6584845012574592
```

Out[115…

```
array([0.65841754, 0.65850682, 0.65878957, 0.65968244, 0.6584845 ,
       0.6584845 ])
```

In [70]:

```
#Evaluation

t0=time.time()
lr = LogisticRegression(C=0.001, solver='liblinear').fit(xtrain, ytrain)
t_lr = time.time()-t0
print('Time taken :' , t_lr)
yhat = lr.predict(xtest)
c_lr = classification_report(ytest,yhat)
prec_lr = precision_score(ytest, yhat)
rec_lr = recall_score(ytest, yhat)
print('classification_report:' , c_lr)
print('precision_score' , prec_lr)
print('recall_score' , rec_lr)
```

```
Time taken : 15.253852128982544
classification_report:               precision    recall  f1-score   support

           0       0.66      0.82      0.73     94297
           1       0.67      0.46      0.54     73700

    accuracy                           0.66    167997
   macro avg       0.66      0.64      0.64    167997
weighted avg       0.66      0.66      0.65    167997

precision_score 0.6678587038582271
recall_score 0.45611940298507464
```

In [71]:
```python
tt = xtrain.shape[0]
tv = xval.shape[0]
xtrain[int(tt*0.5):].shape[0], xval[int(tv*0.5):].shape[0]
```

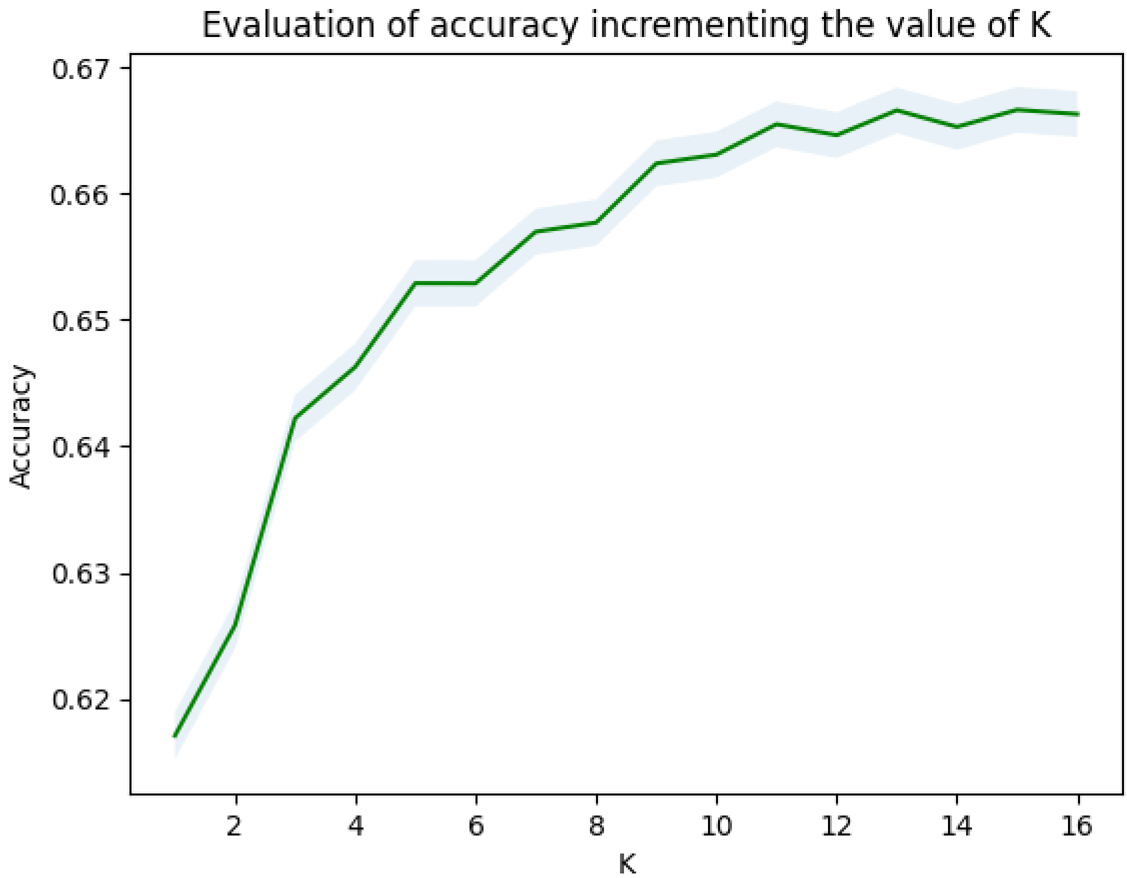
Out[71]:
```
(268795, 67199)
```

In [114…
```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report

ks = 17
mean_acc = np.zeros(ks-1)
std_acc = np.zeros(ks-1)

for n in range(1, ks):
    neigh = KNeighborsClassifier(n_neighbors=n).fit(xtrain[int(tt*0.5):], ytrain
    yhat = neigh.predict(xval[int(tv*0.5):])
    mean_acc[n-1] = accuracy_score(yval[int(tv*0.5):], yhat)
    std_acc[n-1] = np.std(yhat==yval[int(tv*0.5):])/np.sqrt(yhat.shape[0])

best_k = mean_acc.argmax() + 1
best_accuracy = mean_acc.max()
print('Best performing K is', best_k, 'with an accuracy of', best_accuracy)

neigh = KNeighborsClassifier(n_neighbors=best_k).fit(xtrain[int(tt*0.5):], ytrai
yhat_best = neigh.predict(xval[int(tv*0.5):])
report = classification_report(yval[int(tv*0.5):], yhat_best)
print(report)
```

```
Best performing K is 15 with an accuracy of 0.666646825101564
              precision    recall  f1-score   support

           0       0.69      0.73      0.71     37599
           1       0.63      0.58      0.61     29600

    accuracy                           0.67     67199
   macro avg       0.66      0.66      0.66     67199
weighted avg       0.66      0.67      0.66     67199
```

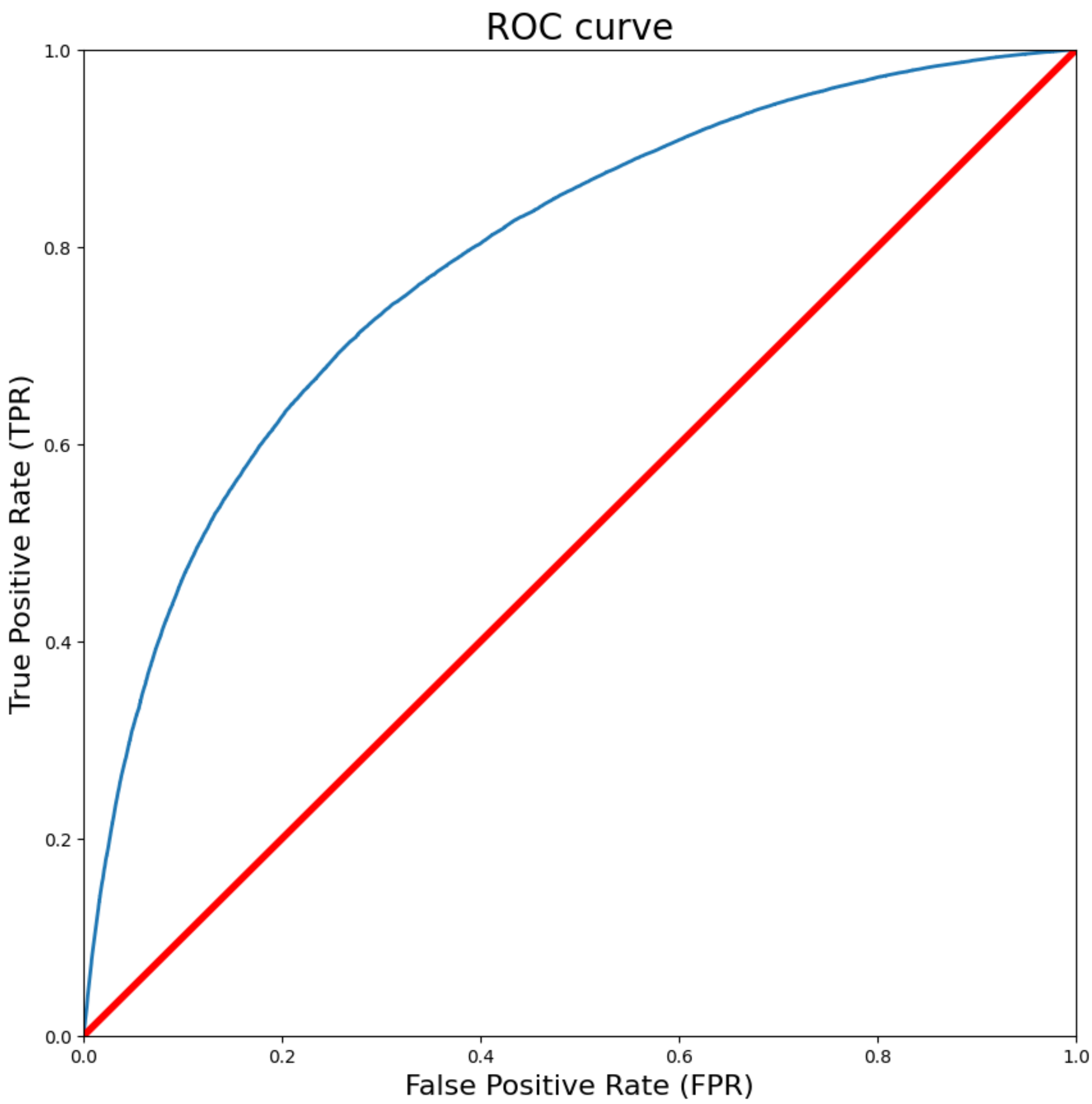
In [113…
```python
plt.plot(range(1,ks),mean_acc,'g')
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.title('Evaluation of accuracy incrementing the value of K')
plt.fill_between(range(1,ks),mean_acc-1*std_acc,mean_acc+1*std_acc, alpha=0.1)
```

Out[113…
```
<matplotlib.collections.PolyCollection at 0x3ff54b80040>
```



In [108…

```
yscores = model_rf.predict_proba(xtest)

false_positive_rate, true_positive_rate, thresholds = roc_curve(ytest.values, ys

def plot_roc_curve(false_positive_rate, true_positive_rate, label=None):
    plt.plot(false_positive_rate, true_positive_rate, linewidth=2, label='a')
    plt.plot([0, 1], [0, 1], 'r', linewidth=4)
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate (FPR)', fontsize=16)
    plt.ylabel('True Positive Rate (TPR)', fontsize=16)

plt.figure(figsize=(10, 10))
plt.title('ROC curve', fontsize=20)
plot_roc_curve(false_positive_rate, true_positive_rate)
plt.show()
```


ROC curve
True Positive Rate (TPR)
False Positive Rate (FPR)
0.0
0.2
0.4
0.6
0.8
1.0

```
In [ ]:
```