TERM PROJECT: 3253 MACHINE LEARNING COURSE, UNIVERSITY OF TORONTO CONTINUING EDUCATION

SUBMITTED by: ADNAN LANEWLA, ARJUN VERMA, ASHOK MISTRY, DAVID SIGNORETTI

ANALYSIS TO PREDICT TORONTO FIRE INJURY AND FATALITY

Dataset from the City of Toronto Open Data Catalogue at:

[https://www.toronto.ca/city-government/data-research-maps/open-data/open-data-catalogue/#e3d443bb-2593-2615-4972-20e24c0ab876 (https://www.toronto.ca/city-government/data-research-maps/open-data/open-data-catalogue/#e3d443bb-2593-2615-4972-20e24c0ab876)](https://www.toronto.ca/city-government/data-research-maps/open-data/open-data-catalogue/#e3d443bb-2593-2615-4972-20e24c0ab876)

Dataset provides information similar to the data sent to the Ontario Fire Marshal relating to all incidents to which the Toronto Fire Services (TFS) responds.

The dataset consists of 6 XML formatted files, one for each year from 2011 to 2016.

Data Gathering and Preparation

Following data gathering and preparation work done by: DAVID SIGNORETTI

Extract CSV from XML

In [1]:
```
import xml.etree.ElementTree as ET
import pandas as pd
import numpy as np
import datetime as dt
from IPython.display import display
import glob
pd.set_option('display.max_columns',200)
```

In [2]:
```
def xml2df(xml_data):
    root = ET.XML(xml_data) # element tree
    all_records = []
    for i, child in enumerate(root):
        record = {}
        for subchild in child:
            record[subchild.tag] = subchild.text
        all_records.append(record)
    df = pd.DataFrame(all_records)
    return df
```

In [3]:
```
# import the xml files into one dataframe
_d = pd.DataFrame()

filenames = sorted(glob.glob('./dataset/xml/201*.xml'))
filenames = filenames[0:6]

for f in filenames:
    print(f)
    xml_data = open(f).read()
    _x = xml2df(xml_data)
    _d = _d.append(_x)
```

```
./dataset/xml\2011.xml
./dataset/xml\2012.xml
./dataset/xml\2013.xml
./dataset/xml\2014.xml
./dataset/xml\2015.xml
./dataset/xml\2016.xml
```

In [4]:
```
# Review the shape of the Dataframe
_d.shape
```

Out[4]:
```
(720370, 103)
```

In [ ]:
```
_d.to_csv('./dataset/fire.csv')
```

In [38]:
```
_d = pd.read_csv('./dataset/fire.csv')
```

In [39]:
```
_d.info(memory_usage='deep')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 117426 entries, 0 to 117425
Columns: 104 entries, Unnamed: 0 to WATER
dtypes: float64(53), int64(26), object(25)
memory usage: 242.9 MB
```

In [40]:
```
_d.head()
```

Out[40]:

	Unnamed: 0	AGENT_APP_HOUR	AGENT_APP_MIN	AGENT_APP_SEC	AGE_OF_STRUCTURE	A
0	0	NaN	NaN	NaN	NaN	
1	1	NaN	NaN	NaN	NaN	
2	2	NaN	NaN	NaN	NaN	
3	3	NaN	NaN	NaN	NaN	
4	4	0.0	12.0	0.0	3.0	

In [41]:
```
# Remove any whitespaces from the names
_d.columns = _d.columns.str.replace(' ', '')
# Set names to lower case
_d.columns = _d.columns.str.lower()
```

In [42]:
```
_d.initial_call_hour = _d.initial_call_hour.astype(dtype=str)
_d.initial_call_min = _d.initial_call_min.astype(dtype=str)
_d.initial_call_sec = _d.initial_call_sec.astype(dtype=str)
```

In [43]:
```
_d['incident_date_time'] = pd.to_datetime(_d['incident_date'] + ' ' + _d['init
ial_call_hour']\
                                    +':'+_d['initial_call_min']+':'+_d['initia
l_call_sec'])
```

In [44]:
```
_d[['incident_date','initial_call_hour','initial_call_min',
 'initial_call_sec','incident_date_time']].dtypes
```

Out[44]:
```
incident_date                  object
initial_call_hour              object
initial_call_min               object
initial_call_sec               object
incident_date_time     datetime64[ns]
dtype: object
```

In [45]:
```
#list(_d)
```

In [17]:
```
# Deterime the pecentage of nan per column
#_d.isna().sum()/len(_d)*100
```

In [18]:
```
# remove columns that have more than 90% nan or 105683 nan rows
#df = _d.loc[:, _d.isnull().sum() < 0.9*_d.shape[0]]
```

In [46]:
```
df = _d.copy()
```

In [23]:
```
#df = _d[['incident_date_time','civilian_fire_fatality','civilian_fire_injur
y','civ_evacuation','fd_station',\
#          'ff_fatalities','ff_injuries','incident_date','incident_number','occ
_status',\
#          'occ_type','rescued_adults','rescued_children','rescued_seniors','re
scues','responding_units',\
#          'smoke_alarm_impact_on_num_evac','est_loss','response_type','respond
ingunits',\
#          'status_on_arrival','total_num_personnel','property']]

# set incident as indext
#df = df.set_index('incident_number')
```

In [20]:
```
# Review the new shape of the Dataframe
#df.head()
```

In [25]:
```
# Delete orignal Dataframe _d
del _d
```

In [48]:
```
# Fill any NAN data withthe average of the column
for key, value in df.iteritems():
    if np.issubdtype(df[key].dtype, np.number) == True:
        _v = df[key].mean()
        df[key].fillna(value=_v)
```

In [49]:
```
df.info(memory_usage='deep')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 117426 entries, 0 to 117425
Columns: 105 entries, unnamed:0 to incident_date_time
dtypes: datetime64[ns](1), float64(53), int64(23), object(28)
memory usage: 260.8 MB
```

In [28]:
```
# Change dates from object to date time in int64 unix timestamp
#@df['incident_date'] = pd.to_datetime(df['incident_date'])
```

In [50]:
```
df.to_csv('./dataset/TFSDataSet.csv')
```

Adnan's Exploratory analysis

Feature Engineering

Adding a feature column (total_inj_fatality) combining Civilian and Firefighter Injuries and Fatalities.

In [76]:
```
import pandas as pd
import numpy as np
from IPython.display import display
pd.set_option('display.max_columns',200)
import matplotlib
import matplotlib.pyplot as plt
```

In [77]:
```
def calc_total_inj_fat(df):
    df_fat_inj = df[['ff_injuries', 'ff_fatalities', 'civilian_fire_injury',
'civilian_fire_fatality']]
    df.drop(['ff_injuries', 'ff_fatalities', 'civilian_fire_injury', 'civilian
_fire_fatality'], inplace=True, axis=1)
    ff_inj = df_fat_inj['ff_injuries'].astype(int)
    ff_fat = df_fat_inj['ff_fatalities'].astype(int)
    cv_inj = df_fat_inj['civilian_fire_injury'].astype(int)
    cv_fat = df_fat_inj['civilian_fire_fatality'].astype(int)
    print('ff_injuries')
    print(df_fat_inj['ff_injuries'].value_counts())
    print('ff_fatalities')
    print(df_fat_inj['ff_fatalities'].value_counts())
    print('civilian_fire_injury')
    print(df_fat_inj['civilian_fire_injury'].value_counts())
    print('civilian_fire_fatality')
    print(df_fat_inj['civilian_fire_fatality'].value_counts())
    df['ff_injuries'] = np.where(ff_inj >= 1, 1,0)
    df['ff_fatalities'] = np.where(ff_fat>= 1, 1,0)
    df['civilian_fire_injury'] = np.where(cv_inj >= 1, 1,0)
    df['civilian_fire_fatality'] = np.where(cv_fat >= 1, 1,0)
    ff_inj = np.where(ff_inj >= 1, 1,0)
    ff_fat = np.where(ff_fat>= 1, 1,0)
    cv_inj = np.where(cv_inj >= 1, 1,0)
    cv_fat = np.where(cv_fat >= 1, 1,0)
    total_inj_fat = np.empty(720370,)
    for index, val in enumerate(ff_inj):
        total_inj_fat[index] = np.where((ff_inj[index] + ff_fat[index] + cv_in
j[index] + cv_fat[index]) >=1, 1,0)
    df['total_inj_fatality'] = total_inj_fat
    print('ff_injuries')
    print(df['ff_injuries'].value_counts())
    print('ff_fatalities')
    print(df['ff_fatalities'].value_counts())
    print('civilian_fire_injury')
    print(df['civilian_fire_injury'].value_counts())
    print('civilian_fire_fatality')
    print(df['civilian_fire_fatality'].value_counts())
    print('total_inj_fatality')
    print(df['total_inj_fatality'].value_counts())
    return df
```

In [78]:
```
Pure_df = pd.read_csv('./dataset/TFSDataSet.csv')
```

In [73]:
```
Pure_df.shape
```

Out[73]:
```
(117426, 106)
```

Data Exploration and Analysis

In [79]:
```
import pandas as pd
import numpy as np
import os
from IPython.display import display
pd.set_option('display.max_columns',200)

import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
sns.set_context('poster')
sns.set_style('white')
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib

C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\envs\mlboo
k\lib\site-packages\IPython\core\magics\pylab.py:160: UserWarning: pylab impo
rt has clobbered these variables: ['f']
`%matplotlib` prevents importing * from pylab and numpy
  "\n`%matplotlib` prevents importing * from pylab and numpy"
```

In [80]:
```
Pure_df = pd.read_csv('./dataset/TFSDataSetWithTotalFatality.csv')
```

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\envs\mlboo
k\lib\site-packages\IPython\core\interactiveshell.py:2728: DtypeWarning: Colu
mns (21,22,44) have mixed types. Specify dtype option on import or set low_me
mory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

In [81]:
```
df = Pure_df.copy()

df.head()
```

Out[81]:

	Unnamed: 0	Unnamed: 0.1	aid_to_from_other_depts	alarm_to_fd	arrive_date	bld_height	canutec
0	0	0	4	3.0	2011-01-01 00:10:02	0	
1	1	1	4	1.0	2011-01-01 00:09:02	0	
2	2	2	4	3.0	2011-01-01 00:09:34	0	
3	3	3	4	1.0	2011-01-01 00:10:46	0	
4	4	4	1	5.0	2011-01-01 00:11:03	0	

In [82]:
```
def plotbar_0(label, ax1, ax2, title, df):
    feature = df.groupby(label)
    feature.size().plot(kind='bar', color='blue', legend=True, label='No injur
ies', ax=axes[ax1,ax2], title=title)
```

In [83]:
```
def plotbar_1(label, ax1, ax2, title, df):
    feature = df.groupby(label)
    feature.size().plot(kind='bar', color='Orange', legend=True, label='Injuri
es', ax=axes[ax1,ax2], title=title)
```

In [84]:
```
def plotbar_0_3(label, title, df):
    feature = df.groupby(label)
    feature.size().plot(kind='bar', color='blue', legend=True, label=label, ti
tle=title, figsize=(16,8))
    save_fig(title)
    plt.show()
    plt.clf()
    plt.cla()
    plt.close()
```

In [85]:
```
def plotbar_1_3(label, title, df):
    feature = df.groupby(label)
    feature.size().plot(kind='bar', color='Orange', legend=True, label=label,
title=title, figsize=(16,8))
    save_fig(title)
    plt.show()
    plt.clf()
    plt.cla()
    plt.close()
```

In [86]:
```
def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join("./Images/", fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

In [87]:
```
def get_injuries(df):
    df = df[(df['total_inj_fatality'] == 1)]
    return df
```

In [88]:
```
def get_noinjuries(df):
    df = df[(df['total_inj_fatality'] == 0)]
    return df
```

In [89]:
```
def label_vs_injuries(df, label, title_0, title_1):
    df_copy = df.copy()
    if(df_copy[label].isna().sum()/len(df_copy[label]) *100 > 0.0):
        _v = df_copy[label].mean()
        print(_v)
        df_copy[label].fillna(value=_v, inplace=True)
    df_copy_0 = get_noinjuries(df_copy)
    df_copy_1 = get_injuries(df_copy)
    plotbar_0(label,title_0, df_copy_0)
    plotbar_1(label,title_1, df_copy_1)
```

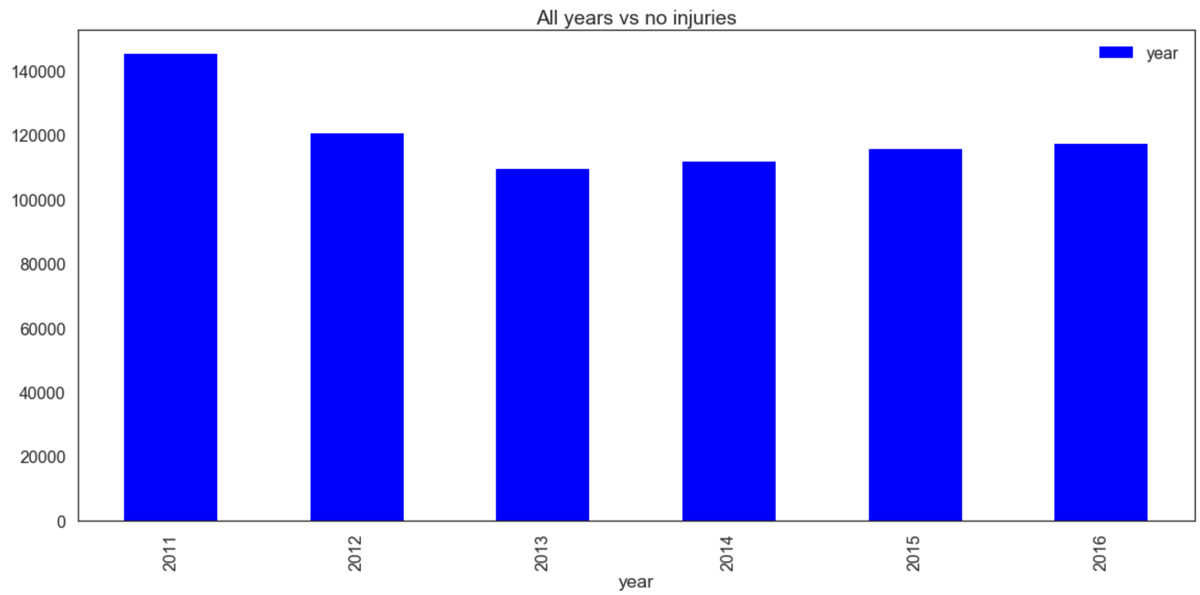
Injuries vs Year

In [90]:
```
data_allyear = df.copy()
```

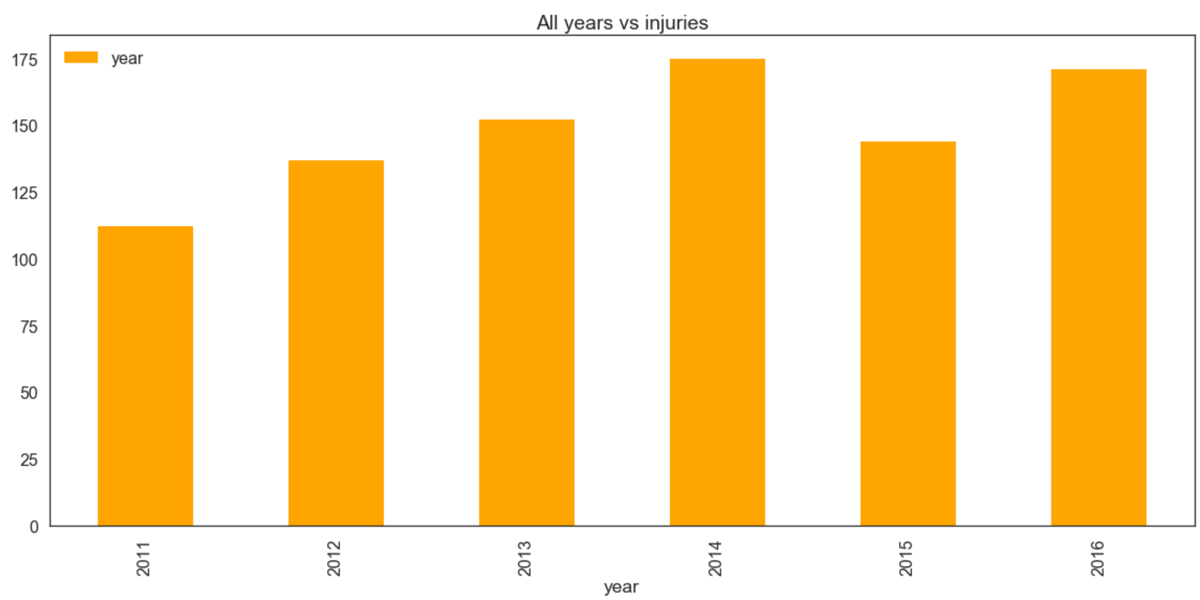
In [12]:
```
data_allyear['year'] = df['incident_date'].map(lambda x: x[0:4])
data_allyear = data_allyear[['year', 'total_inj_fatality']]
```

In [13]:
```
data_allyear_0 = get_noinjuries(data_allyear)
data_allyear_1 = get_injuries(data_allyear)

plotbar_0_3('year','All years vs no injuries', data_allyear_0)
plotbar_1_3('year','All years vs injuries', data_allyear_1)
```

```
Saving figure All years vs no injuries
```




```
Saving figure All years vs injuries
```



After looking at the plots it seems like there is no correlation with what year is it to number of injuries

- We can ignore the year in the incident data

Injuries vs Month

In [14]:
```
data2011 = df[(df['incident_date'] >= '2011-01-01') & (df['incident_date'] <= '2012-01-01')]
data2012 = df[(df['incident_date'] >= '2012-01-01') & (df['incident_date'] <= '2013-01-01')]
data2013 = df[(df['incident_date'] >= '2013-01-01') & (df['incident_date'] <= '2014-01-01')]
data2014 = df[(df['incident_date'] >= '2014-01-01') & (df['incident_date'] <= '2015-01-01')]
data2015 = df[(df['incident_date'] >= '2015-01-01') & (df['incident_date'] <= '2016-01-01')]
data2016 = df[(df['incident_date'] >= '2016-01-01') & (df['incident_date'] <= '2017-01-01')]
```

In [15]:
```
def get_month(dataframe):
    dataframe['month'] = dataframe['incident_date'].map(lambda x: x[5:7])
    dataframe = dataframe[['month', 'total_inj_fatality']]
    return dataframe
```

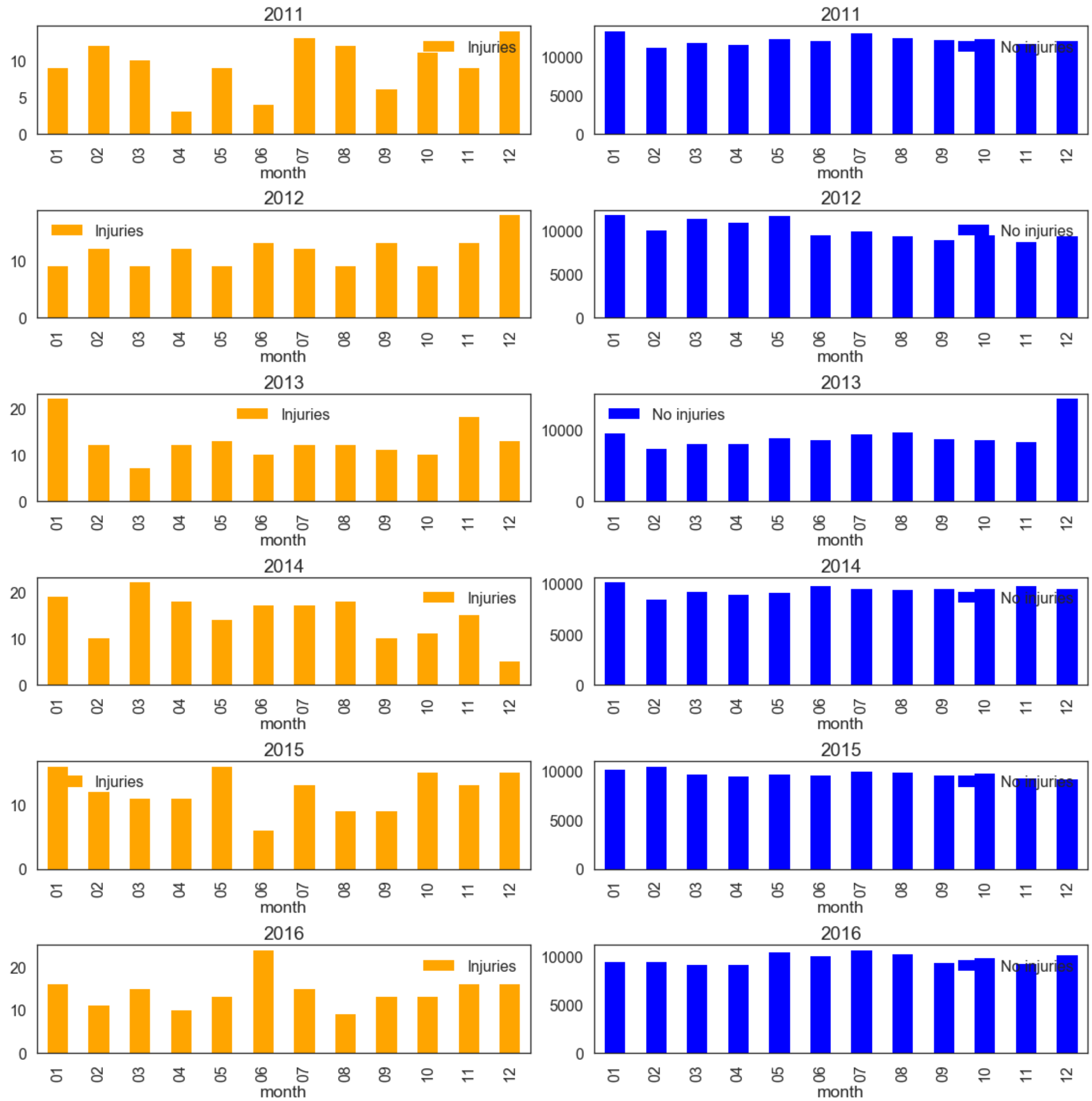
In [16]:
```
data2011 = get_month(data2011)
data2012 = get_month(data2012)
data2013 = get_month(data2013)
data2014 = get_month(data2014)
data2015 = get_month(data2015)
data2016 = get_month(data2016)
```

```
/Applications/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
```

In [17]:
```
data2011_0 = data2011[(data2011['total_inj_fatality'] == 0)]
data2011_1 = data2011[(data2011['total_inj_fatality'] == 1)]

data2012_0 = data2012[(data2012['total_inj_fatality'] == 0)]
data2012_1 = data2012[(data2012['total_inj_fatality'] == 1)]

data2013_0 = data2013[(data2013['total_inj_fatality'] == 0)]
data2013_1 = data2013[(data2013['total_inj_fatality'] == 1)]

data2014_0 = data2014[(data2014['total_inj_fatality'] == 0)]
data2014_1 = data2014[(data2014['total_inj_fatality'] == 1)]

data2015_0 = data2015[(data2015['total_inj_fatality'] == 0)]
data2015_1 = data2015[(data2015['total_inj_fatality'] == 1)]

data2016_0 = data2016[(data2016['total_inj_fatality'] == 0)]
data2016_1 = data2016[(data2016['total_inj_fatality'] == 1)]
```

In [18]:
```
fig, axes = plt.subplots(nrows=6, ncols=2, figsize=(16,16))
plt.subplots_adjust(top=2)

plotbar_1('month', 0,0, 2011, data2011_1)
plotbar_0('month', 0,1, 2011, data2011_0)

plotbar_1('month', 1,0, 2012, data2012_1)
plotbar_0('month', 1,1, 2012, data2012_0)

plotbar_1('month', 2,0, 2013, data2013_1)
plotbar_0('month', 2,1, 2013, data2013_0)

plotbar_1('month', 3,0, 2014, data2014_1)
plotbar_0('month', 3,1, 2014, data2014_0)

plotbar_1('month', 4,0, 2015, data2015_1)
plotbar_0('month', 4,1, 2015, data2015_0)

plotbar_1('month', 5,0, 2016, data2016_1)
plotbar_0('month', 5,1, 2016, data2016_0)

save_fig('month vs injuries')
```

```
Saving figure month vs injuries
```



Looking at the month for every year from incident_data column. It doesn't seem like there is any correlation with the Number of injuries with what month it is in the year.

- We can remove the column incident data

Time-to-reach vs injuries

Using the initial call min and onscene min feature we calculated min to reach feature.

Ploting this feature with repect to injuries and no injuries

In [19]:
```
df_tt_min = df.copy()
```

In [20]:
```
df_tt_min['onscene_min'].isna().sum()/len(df_tt_min['onscene_min'])*100
```

Out[20]: 2.1337645931951634

In [21]:
```
df_tt_min['total_min'] = 0
```

In [22]:
```
df_tt_min.head(10)
```

Out[22]:

	Unnamed: 0	Unnamed: 0.1	aid_to_from_other_depts	alarm_to_fd	arrive_date	bld_height	canutec
0	0	0	4	3.0	2011-01-01 00:10:02	0	
1	1	1	4	1.0	2011-01-01 00:09:02	0	
2	2	2	4	3.0	2011-01-01 00:09:34	0	
3	3	3	4	1.0	2011-01-01 00:10:46	0	
4	4	4	1	5.0	2011-01-01 00:11:03	0	
5	5	5	4	1.0	2011-01-01 00:13:46	0	
6	6	6	4	1.0	2011-01-01 00:12:54	0	
7	7	7	4	3.0	2011-01-01 00:12:43	0	
8	8	8	4	3.0	2011-01-01 00:15:44	0	
9	9	9	4	4.0	2011-01-01 00:14:28	0	

In [23]:
```
df.tail(10)
```

Out[23]:

	Unnamed: 0	Unnamed: 0.1	aid_to_from_other_depts	alarm_to_fd	arrive_date	bld_height	ca
720360	720360	117416	4	3.0	2016-12-31 23:37:06	0	
720361	720361	117417	4	3.0	2016-12-31 23:53:02	0	
720362	720362	117418	4	3.0	2016-12-31 23:57:24	0	
720363	720363	117419	4	3.0	2016-12-31 23:51:31	0	
720364	720364	117420	4	3.0	2016-12-31 23:55:31	0	
720365	720365	117421	4	1.0	2016-12-31 23:57:28	0	
720366	720366	117422	4	3.0	2017-01-01 00:01:46	0	
720367	720367	117423	4	3.0	2017-01-01 00:02:27	0	
720368	720368	117424	4	5.0	2017-01-01 00:04:56	0	
720369	720369	117425	4	3.0	2017-01-01 00:03:54	0	

In [24]:
```
_v = df_tt_min['onscene_min'].mean()
print(_v)
df_tt_min['onscene_min'].fillna(value=_v, inplace=True)
```

```
29.531010682284656
```

In [25]:
```
df_tt_min['onscene_min'].isna().sum()/len(df_tt_min['onscene_min'])*100
```

Out[25]:
```
0.0
```

In [26]:
```
for index, row in df_tt_min.iterrows():
    x = df_tt_min.iloc[index]['onscene_min']
    y = df_tt_min.iloc[index]['initial_call_min']
    if(x > y):
        df_tt_min.at[index,'total_min'] = x - y
    else:
        df_tt_min.at[index,'total_min'] = y - x
```

In [27]:
```
df_tt_min.head()
```

Out[27]:

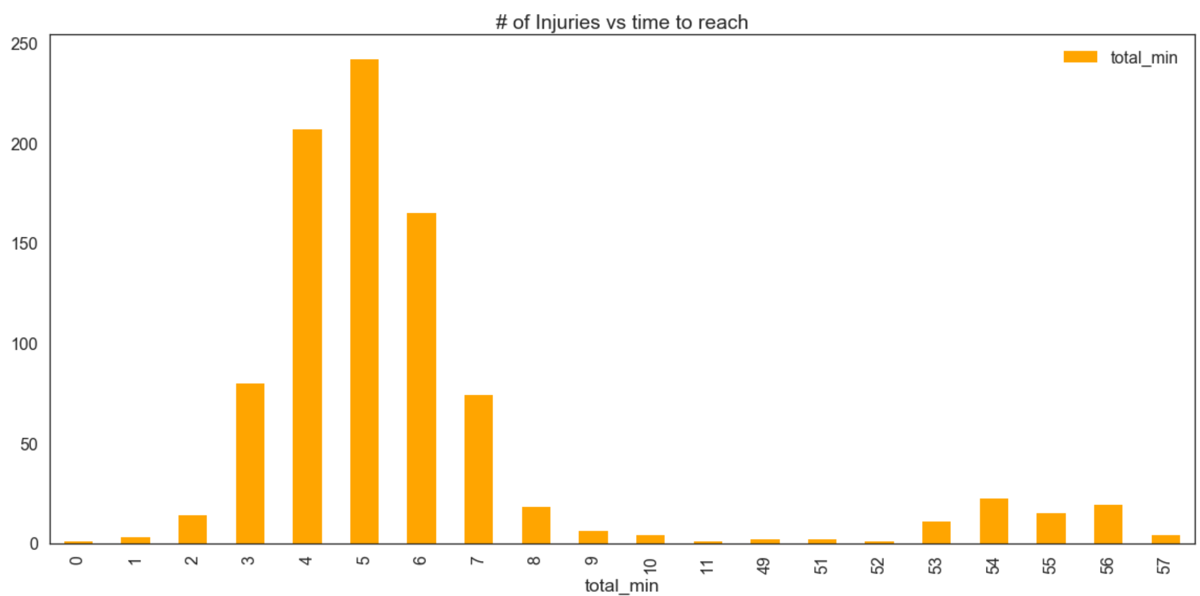
	Unnamed: 0	Unnamed: 0.1	aid_to_from_other_depts	alarm_to_fd	arrive_date	bld_height	canutec
0	0	0	4	3.0	2011-01-01 00:10:02	0	
1	1	1	4	1.0	2011-01-01 00:09:02	0	
2	2	2	4	3.0	2011-01-01 00:09:34	0	
3	3	3	4	1.0	2011-01-01 00:10:46	0	
4	4	4	1	5.0	2011-01-01 00:11:03	0	

In [28]:
```
df_tt_min.to_csv('./dataset/TFSDataSetWithTotalFatality_totalmin.csv')
```

In [29]:
```
df_tt_min = df_tt_min[['total_min', 'total_inj_fatality']]
df_tt_min_1 = get_injuries(df_tt_min)
df_tt_min_0 = get_noinjuries(df_tt_min)
```

In [30]:
```
plotbar_1_3('total_min','# of Injuries vs time to reach', df_tt_min_1)
save_fig('Number of Injuries vs time to reach')
```

Saving figure # of Injuries vs time to reach

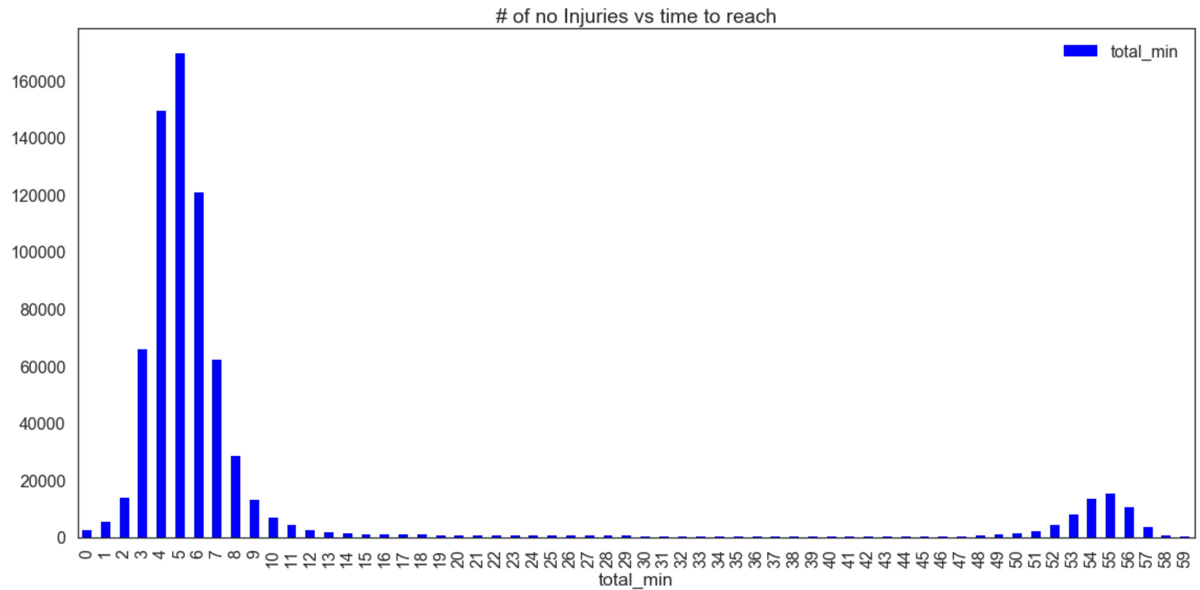


Saving figure Number of Injuries vs time to reach

<matplotlib.figure.Figure at 0x1a26320d68>

In [31]:
```
plotbar_0_3('total_min','# of no Injuries vs time to reach', df_tt_min_0)
save_fig('Number of No Injuries vs time to reach')
```

```
Saving figure # of no Injuries vs time to reach
```




```
Saving figure Number of No Injuries vs time to reach
```

```
<matplotlib.figure.Figure at 0x1a4ac4d828>
```

Looking at the time it takes to reach the location of concern. It seems like there may be some concern of injuries if the paramedics or fire department does get there under 5 mins.

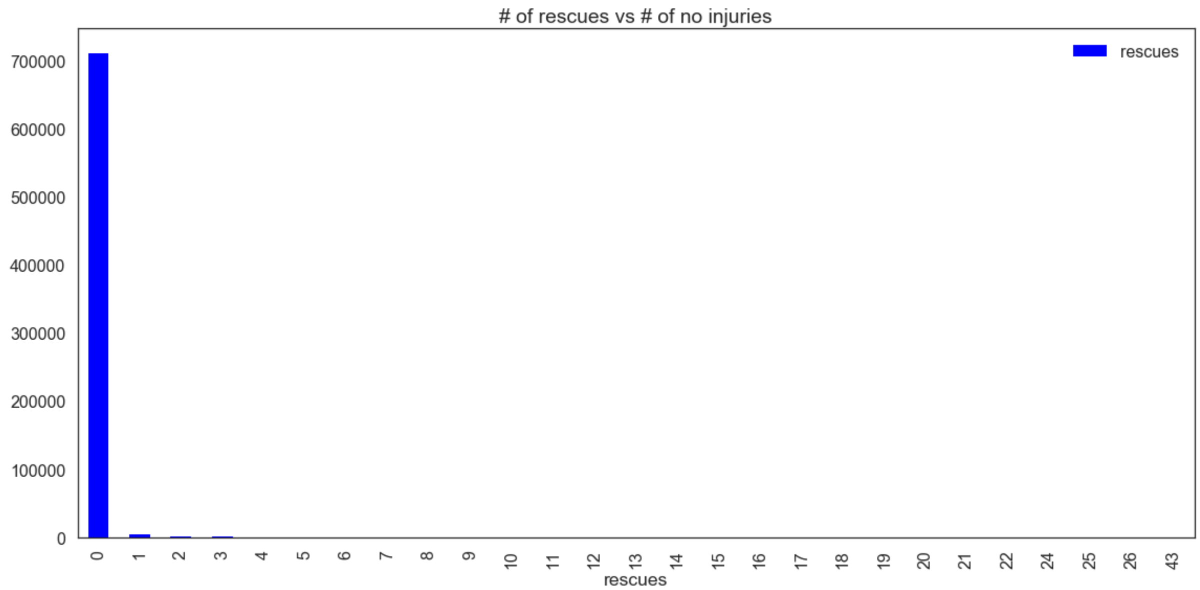
- Keep total_min and scale this column between 0 and 1

Number of Rescues vs Injuries

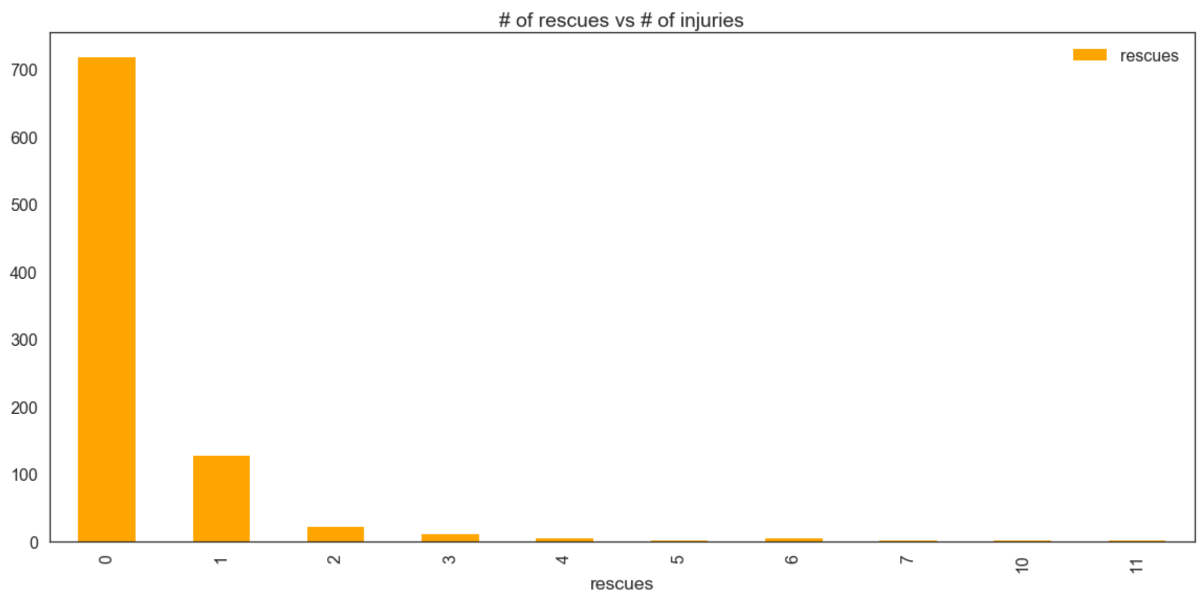
In [32]:
```
def rescues_vs_injuries(df, label, title_0, title_1):
    df_copy = df.copy()
    if(df_copy[label].isna().sum()/len(df_copy[label]) *100 > 0.0):
        _v = df_copy[label].mean()
        print(_v)
        df_copy[label].fillna(value=_v, inplace=True)
    df_copy_0 = get_noinjuries(df_copy)
    df_copy_1 = get_injuries(df_copy)
    plotbar_0_3(label,title_0, df_copy_0)
    plotbar_1_3(label,title_1, df_copy_1)
```

In [33]:
```
rescues_vs_injuries(df, 'rescues', '# of rescues vs # of no injuries', '# of r
escues vs # of injuries')
```

Saving figure # of rescues vs # of no injuries

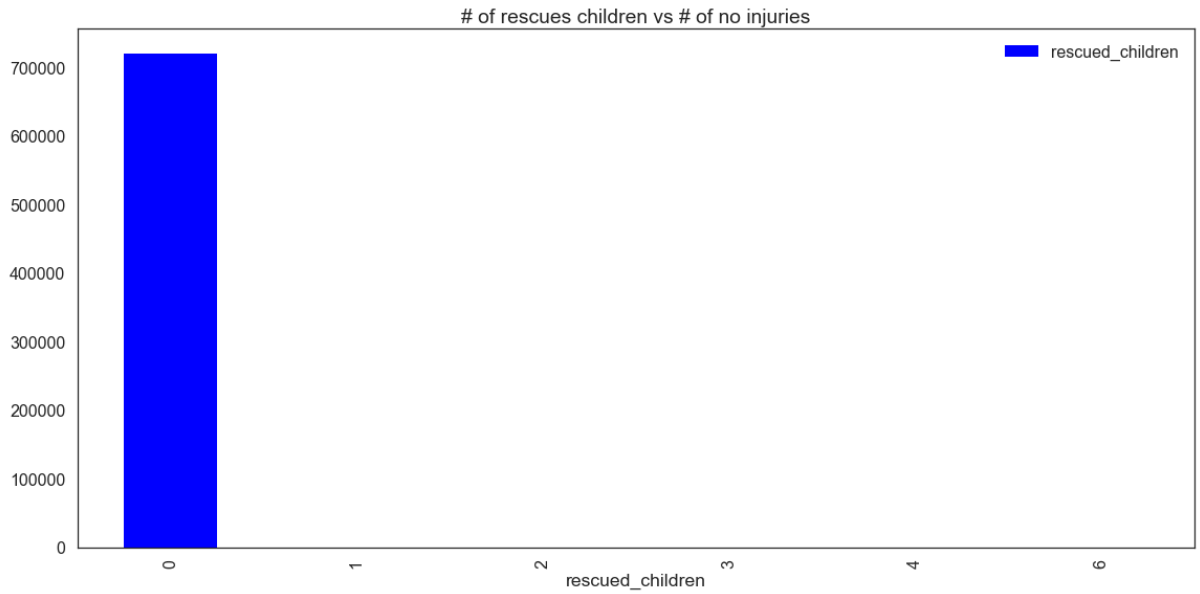


Saving figure # of rescues vs # of injuries

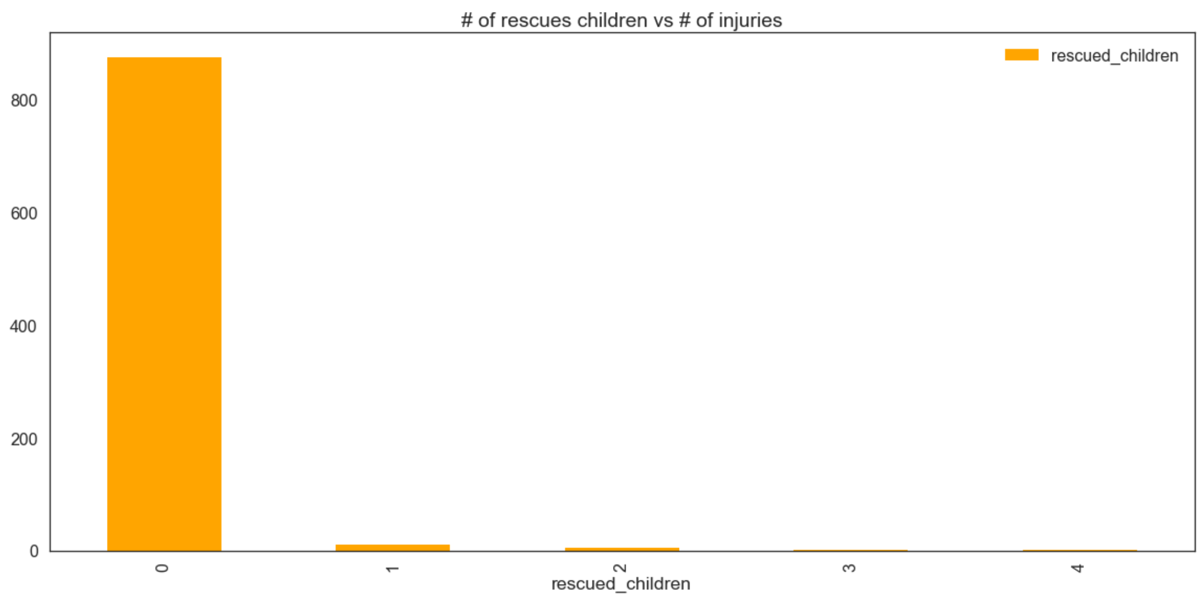


In [34]:
```
rescues_vs_injuries(df, 'rescued_children', '# of rescues children vs # of no
 injuries', '# of rescues children vs # of injuries')
```

```
Saving figure # of rescues children vs # of no injuries
```

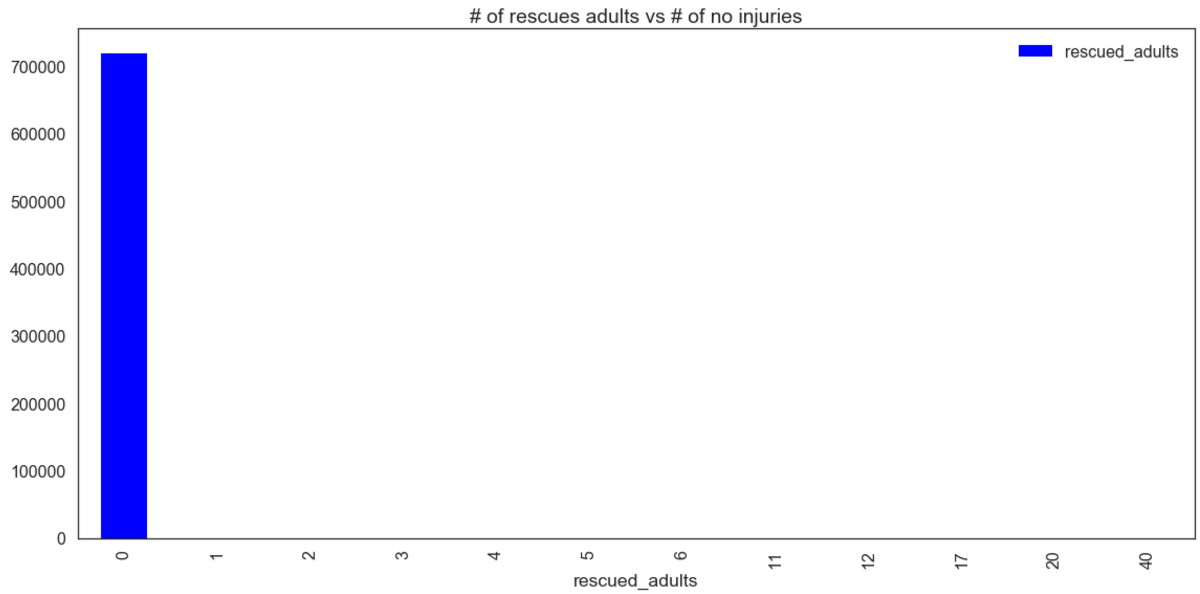



```
Saving figure # of rescues children vs # of injuries
```

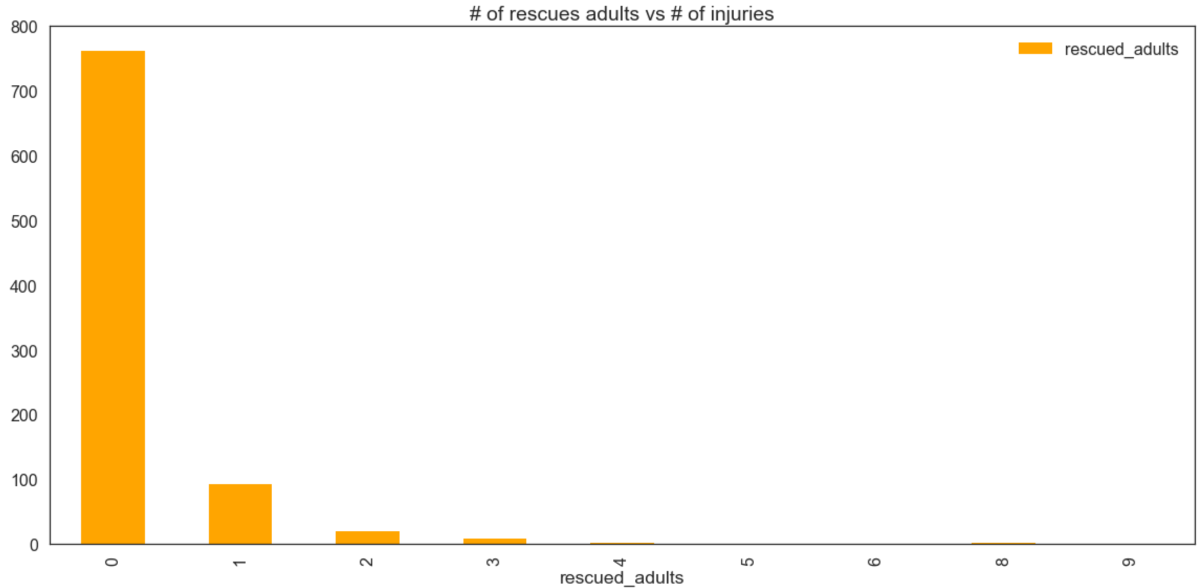


In [35]:
```
rescues_vs_injuries(df, 'rescued_adults', '# of rescues adults vs # of no inju
ries', '# of rescues adults vs # of injuries')
```

Saving figure # of rescues adults vs # of no injuries

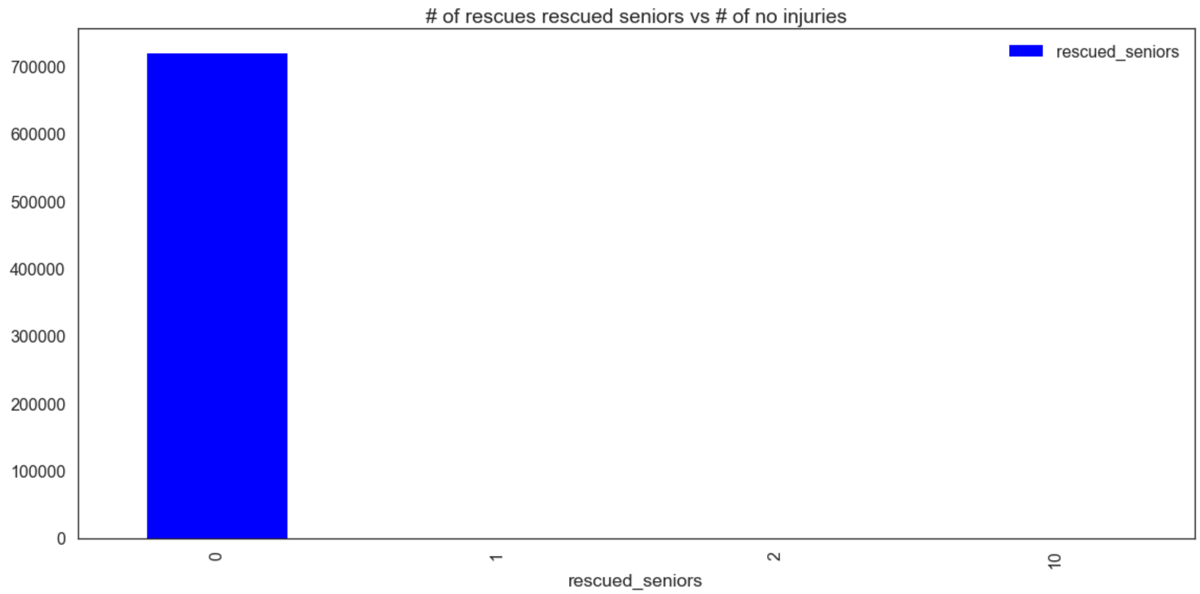


Saving figure # of rescues adults vs # of injuries

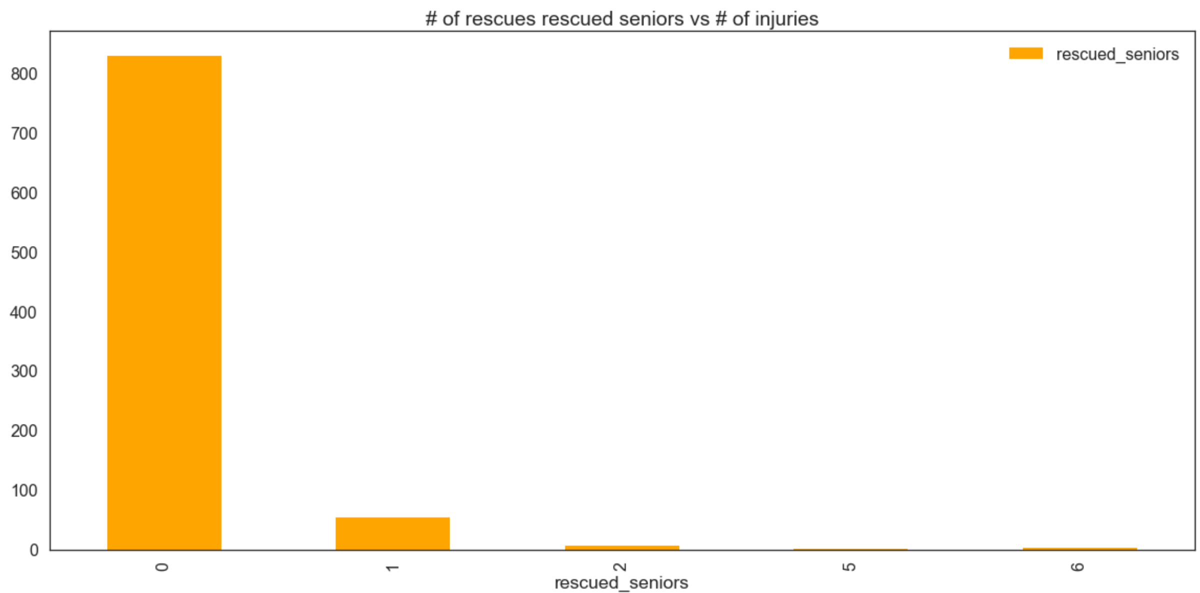


In [36]:
```
rescues_vs_injuries(df, 'rescued_seniors', '# of rescues rescued seniors vs # of no injuries', '# of rescues rescued seniors vs # of injuries')
```

```
Saving figure # of rescues rescued seniors vs # of no injuries
```




```
Saving figure # of rescues rescued seniors vs # of injuries
```

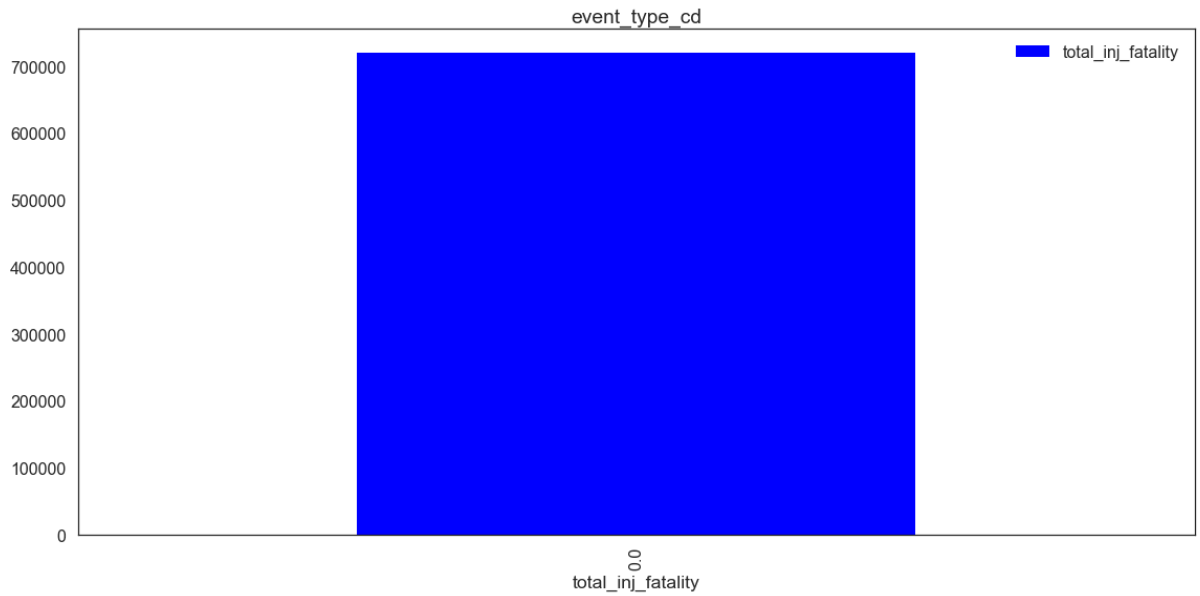


It seems like there is slight correlation of injuries with # of resuces.

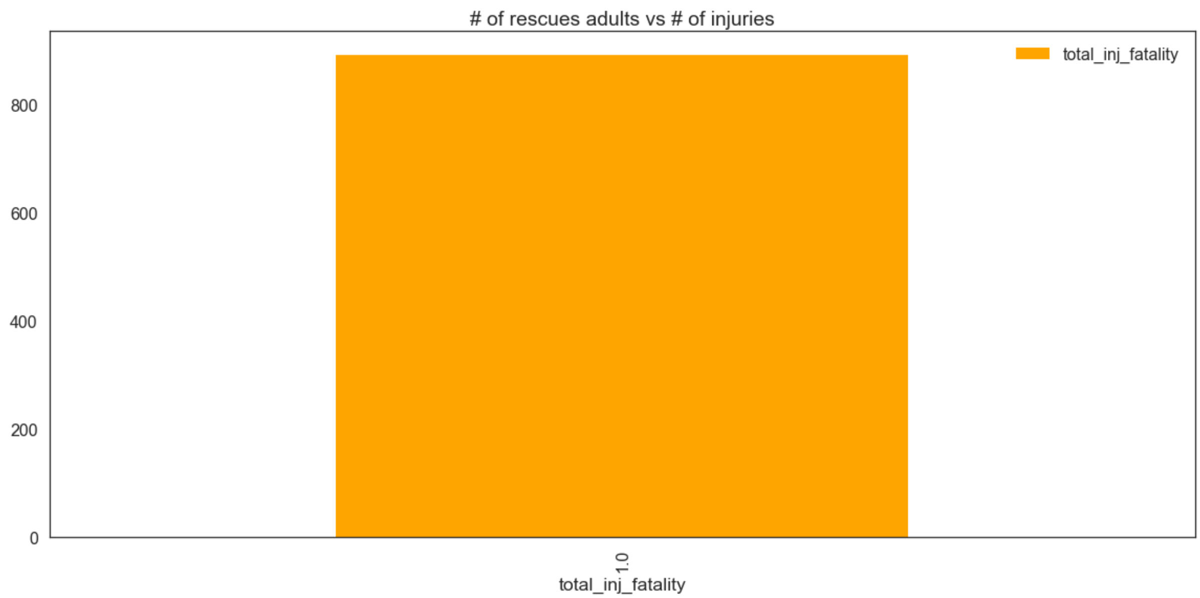
- We see that whenever there was one rescue of any type we had almost some injuries. So this columns tells us that whenever there was a rescue there was a chance of injury
- keep 'rescues' column and scale it between 0 - 1
- Discard 'rescued_adults' column
- Discard 'rescued_children' column
- Discard 'rescued_seniors' column

In [37]:
```
rescues_vs_injuries(df, 'total_inj_fatality', 'event_type_cd', '# of rescues a
dults vs # of injuries')
```

```
Saving figure event_type_cd
```



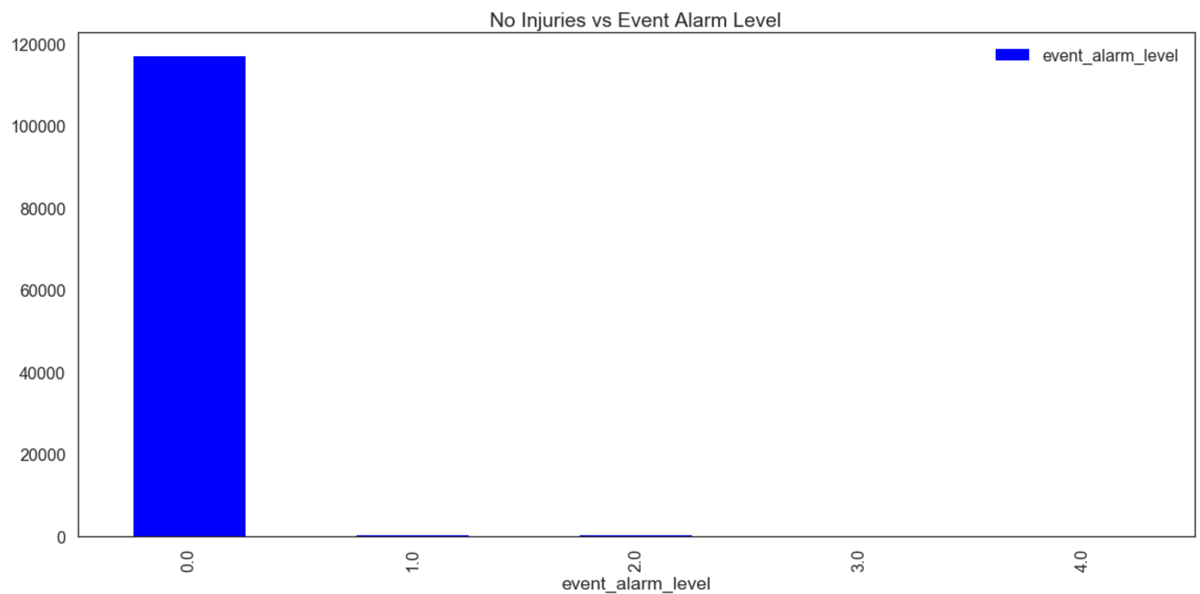

```
Saving figure # of rescues adults vs # of injuries
```



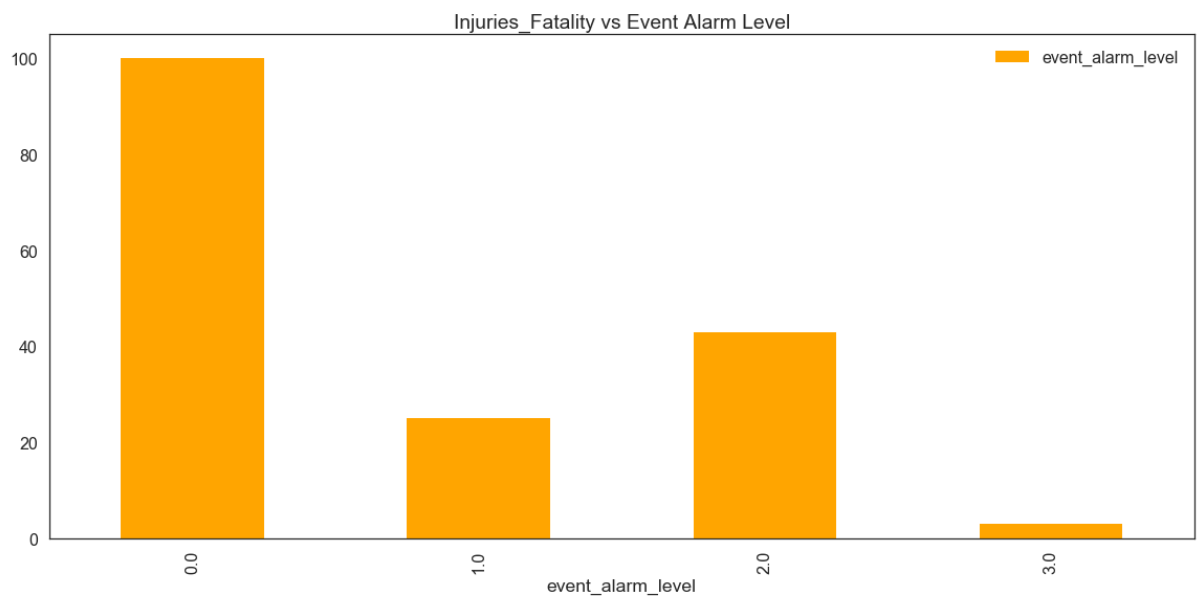
Injuries vs Event Alarm Level

In [69]:
```
data_event_alarm = df.copy()
data_event_alarm = data_event_alarm[['event_alarm_level', 'total_inj_fatality'
]]

data_event_alarm_0 = get_noinjuries(data_event_alarm)
data_event_alarm_1 = get_injuries(data_event_alarm)
plotbar_0_3('event_alarm_level','No Injuries vs Event Alarm Level', data_event
_alarm_0)
plotbar_1_3('event_alarm_level','Injuries_Fatality vs Event Alarm Level', data
_event_alarm_1)
```

```
Saving figure No Injuries vs Event Alarm Level
```




```
Saving figure Injuries_Fatality vs Event Alarm Level
```



Event_Type_CD vs Injuries

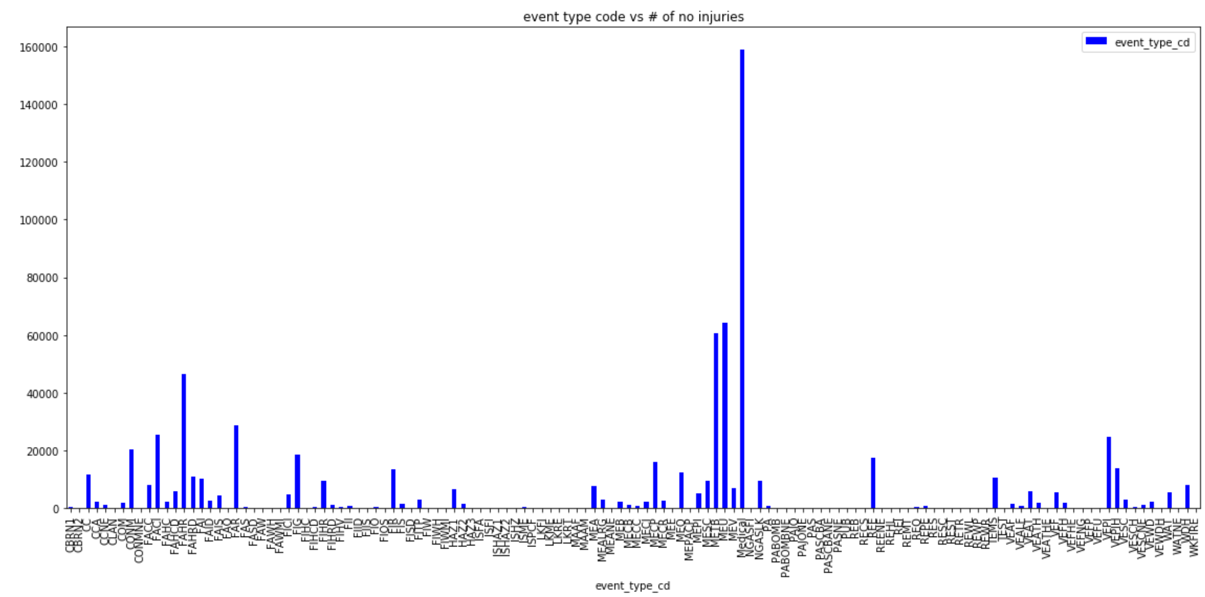
Plotting the type of event code vs injuries / no injuries

In [18]:
```
df_event = df.copy()
```

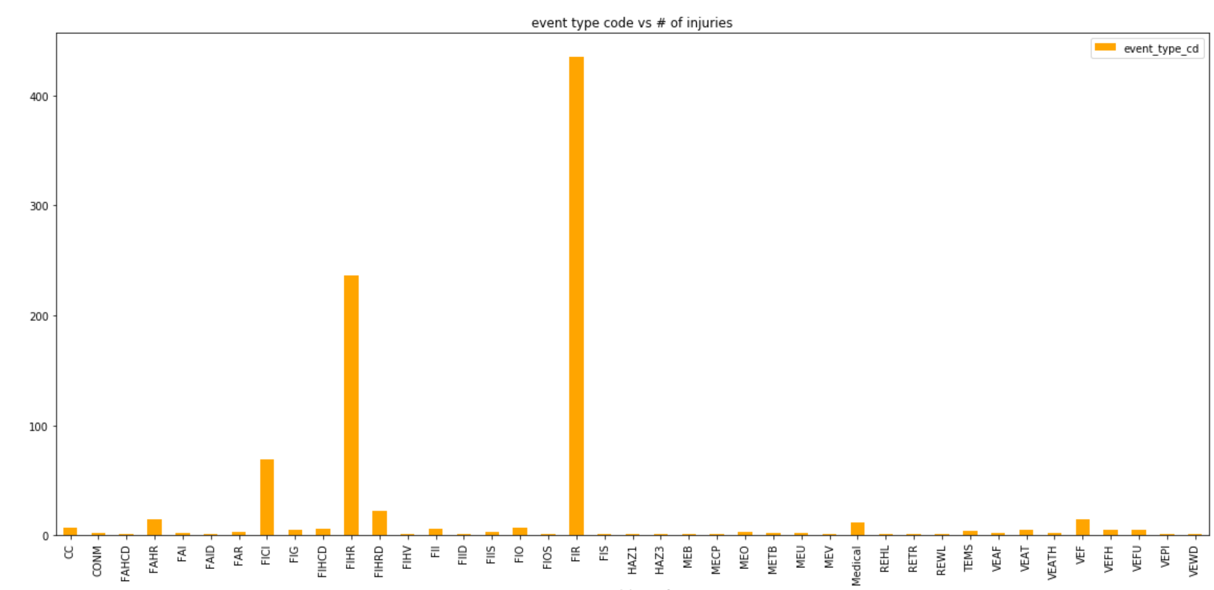
In [19]:
```
def event_type_vs_injuries(df, label, title_0, title_1):
    df_copy = df.copy()
    df_copy_0 = get_noinjuries(df_copy)
    df_copy_1 = get_injuries(df_copy)
    plotbar_0(label,title_0, df_copy_0)
    plotbar_1(label,title_1, df_copy_1)
```

In [20]:
```
event_type_vs_injuries(df_event, 'event_type_cd', 'event type code vs # of no injuries', 'event type code vs # of injuries')
```

```
Saving figure event type code vs # of no injuries
```




```
Saving figure event type code vs # of injuries
```



In [29]:
```
l = df_event['event_type_cd'].value_counts()
```

In [30]:
```
#l = df_event[['event_type_cd']]
d = dict([(y,x+1) for x,y in enumerate((set(l)))])

[d[x] for x in l]
```

Out[30]:
```
[44,
 59,
 56,
 92,
 30,
 107,
 78,
 15,
 18,
 105,
 63,
 52,
 103,
 40,
 93,
 64,
 101,
 20,
 96,
 80,
 79,
 74,
 62,
 51,
 97,
 81,
 55,
 42,
 6,
 95,
 48,
 29,
 90,
 25,
 24,
 104,
 65,
 28,
 82,
 76,
 49,
 45,
 10,
 91,
 83,
 84,
 35,
 108,
 86,
 77,
 75,
 60,
 106,
 100,
 99,
 94,
 57,
```

```
46,
109,
102,
98,
89,
88,
87,
85,
73,
72,
71,
70,
69,
68,
67,
66,
61,
58,
54,
53,
50,
47,
43,
41,
39,
38,
37,
36,
36,
34,
33,
32,
31,
31,
27,
26,
26,
23,
22,
22,
21,
19,
17,
17,
16,
13,
12,
12,
11,
14,
9,
8,
8,
7,
5,
5,
4,
```

```
     3,
     3,
     2,
     2,
     2,
     2,
     2,
     2,
     2,
     1,
     1,
     1,
     1,
     1,
     1,
     1]
```

In [24]:
```
df_event['event_type_cd'].isna().sum()/len(df_event['event_type_cd']) *100 >
0.0
```

Out[24]:
```
True
```

In [ ]:
```
_v = df_event['event_type_cd'].mean()
print(_v)
df_event['event_type_cd'].fillna(value=_v, inplace=True)
```

OFM vs Injuries

Plotting OFM vs injuries and no injuries

In [13]:
```
df['ofm_investigations_contacted'].value_counts()
```

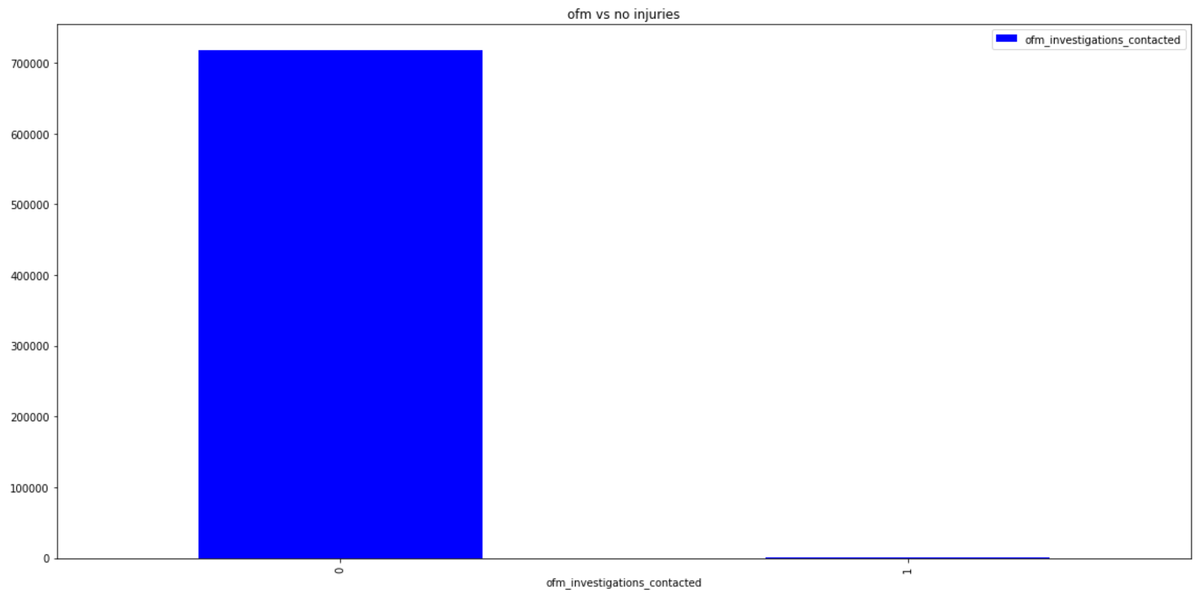
Out[13]:
```
0    719226
1      1144
Name: ofm_investigations_contacted, dtype: int64
```

In [31]:
```
test = df[(df['ofm_investigations_contacted'] == 1) & (df['total_inj_fatality'
] == 1)]
print(test.shape)
```

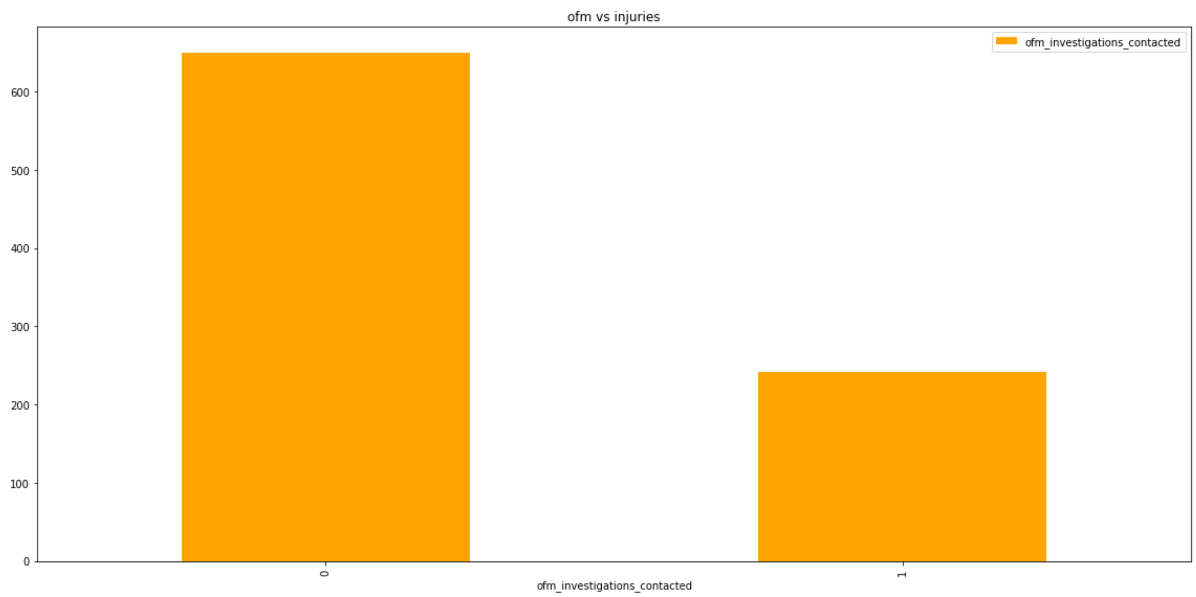
```
(241, 59)
```

In [26]:
```
label_vs_injuries(df,'ofm_investigations_contacted', 'ofm vs no injuries', 'of
m vs injuries')
```

Saving figure ofm vs no injuries



Saving figure ofm vs injuries



In [16]:
```
df['ofm_investigations_contacted'].isna().sum()/len(df['ofm_investigations_con
tacted']) *100 > 0.0
```

Out[16]: False

In [17]:
```
df_ofm = df[['ofm_investigations_contacted']]
```

In [19]:
```
df_ofm.to_csv('./dataset/TFSDataSetWithTotalFatality_Ashok.csv')
```

Aid to from other department vs Injuries

In [21]:
```
df['aid_to_from_other_depts'].value_counts()
```

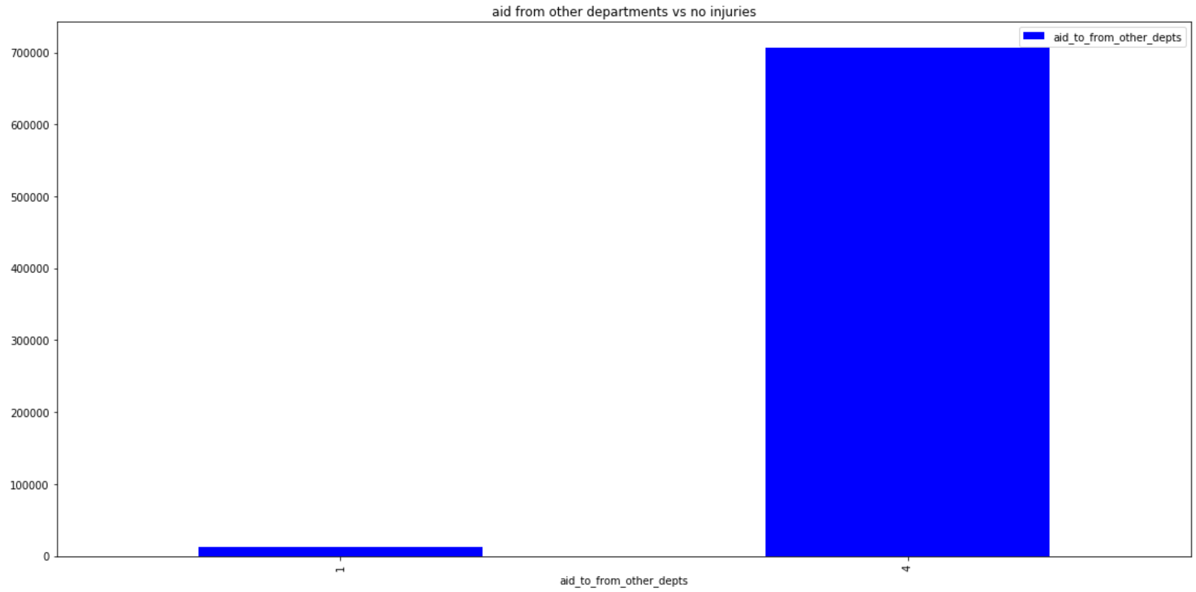
Out[21]:
```
4    707962
1     12408
Name: aid_to_from_other_depts, dtype: int64
```

In [22]:
```
df['aid_to_from_other_depts'].isna().sum()/len(df['aid_to_from_other_depts'])
* 100 > 0.0
```

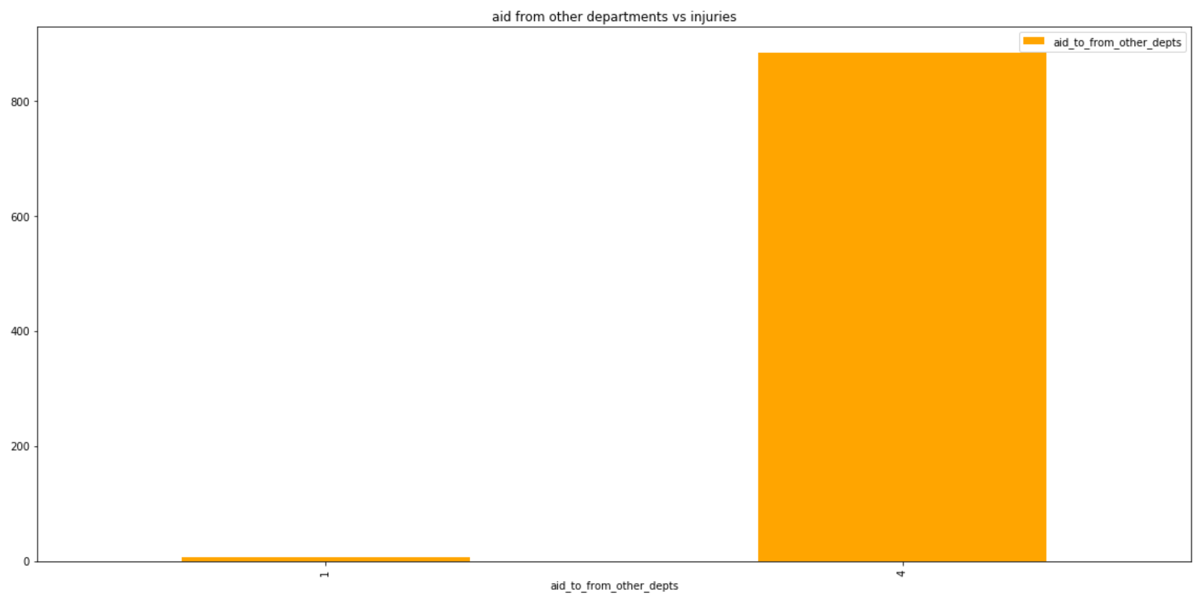
Out[22]:
```
False
```

In [27]:
```
label_vs_injuries(df, 'aid_to_from_other_depts', 'aid from other departments v
s no injuries', 'aid from other departments vs injuries')
```

```
Saving figure aid from other departments vs no injuries
```




```
Saving figure aid from other departments vs injuries
```



Response_Type vs Injuries

In [16]:
```
df['response_type'].isna().sum()/len(df['response_type']) > 100.0
```

Out[16]:
```
False
```

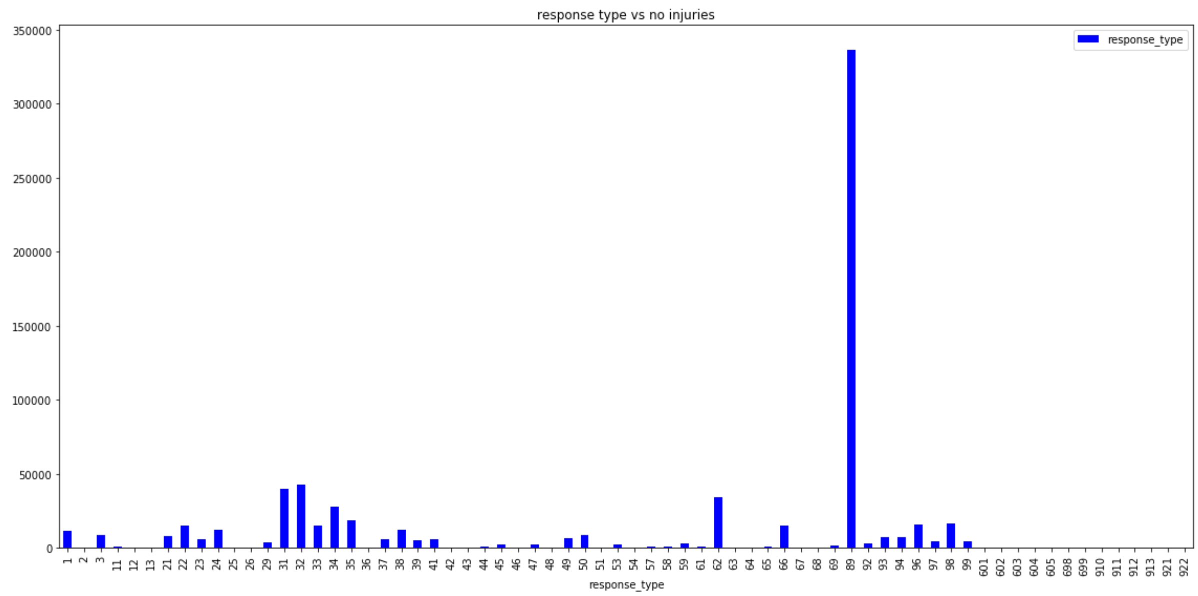
In [28]:
```
test = df[(df['response_type'] == 2) & (df['total_inj_fatality'] == 0)]
```

In [29]:
```
test.shape
```

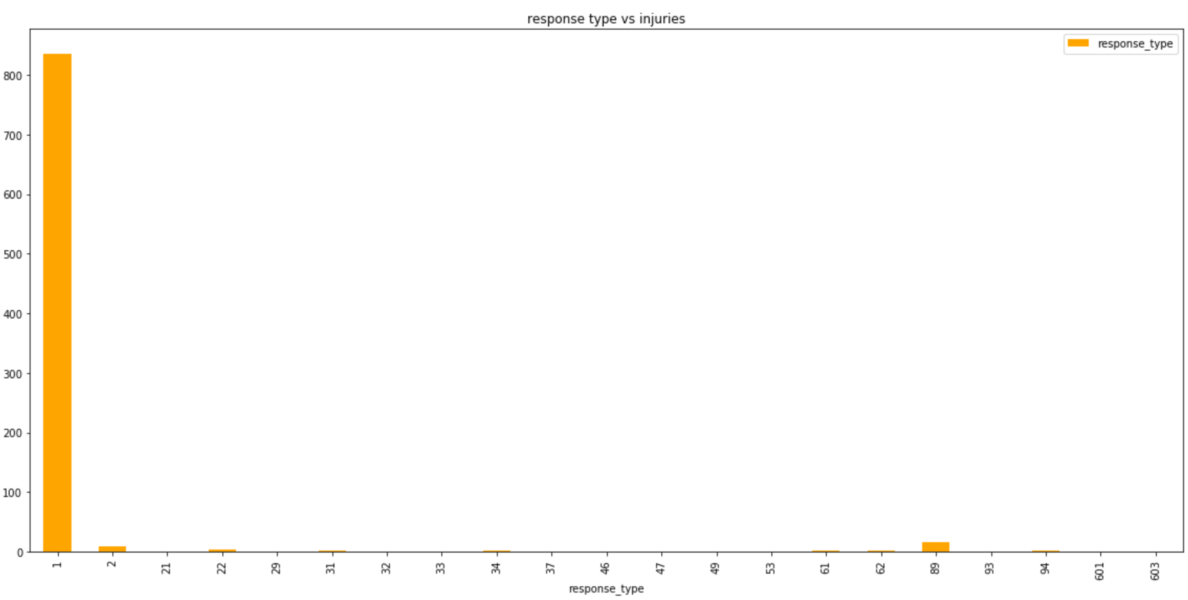
Out[29]:
```
(82, 59)
```

In [15]:
```
label_vs_injuries(df, 'response_type', 'response type vs no injuries', 'respon
se type vs injuries')
```

```
Saving figure response type vs no injuries
```




```
Saving figure response type vs injuries
```



Alarm_Level vs Injuries

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

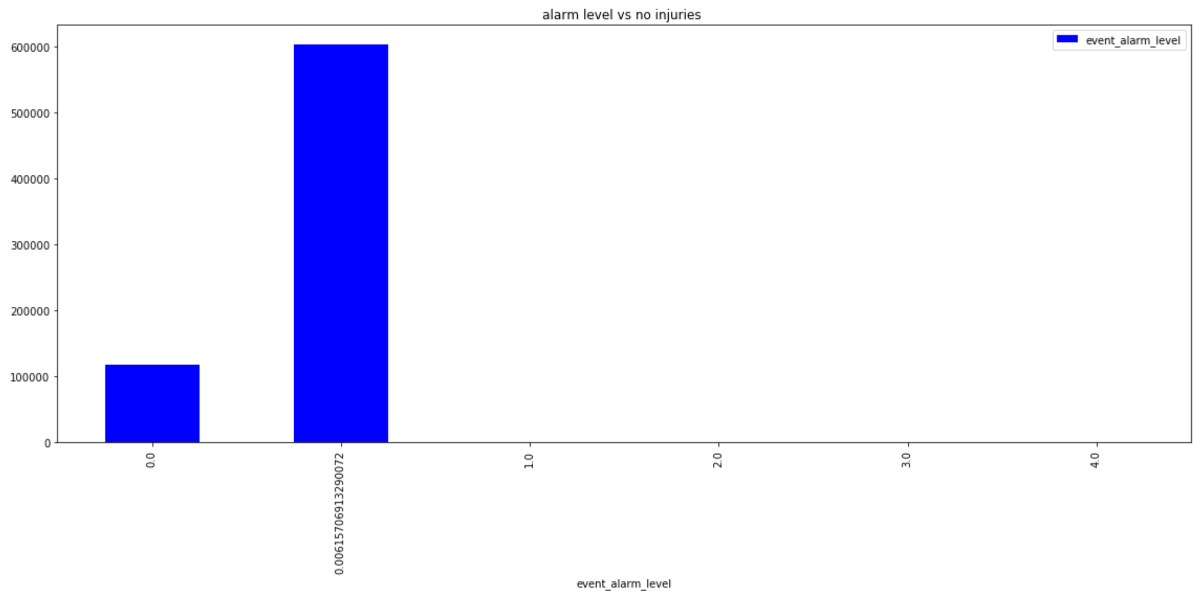
Out[32]:
```
0.0    116941
1.0       256
2.0       221
3.0         7
4.0         1
Name: event_alarm_level, dtype: int64
```

In [39]:
```
alarmdf = df[(df['event_alarm_level'] == 2.0) & (df['total_inj_fatality'] == 1
)]
print(alarmdf.shape)
```

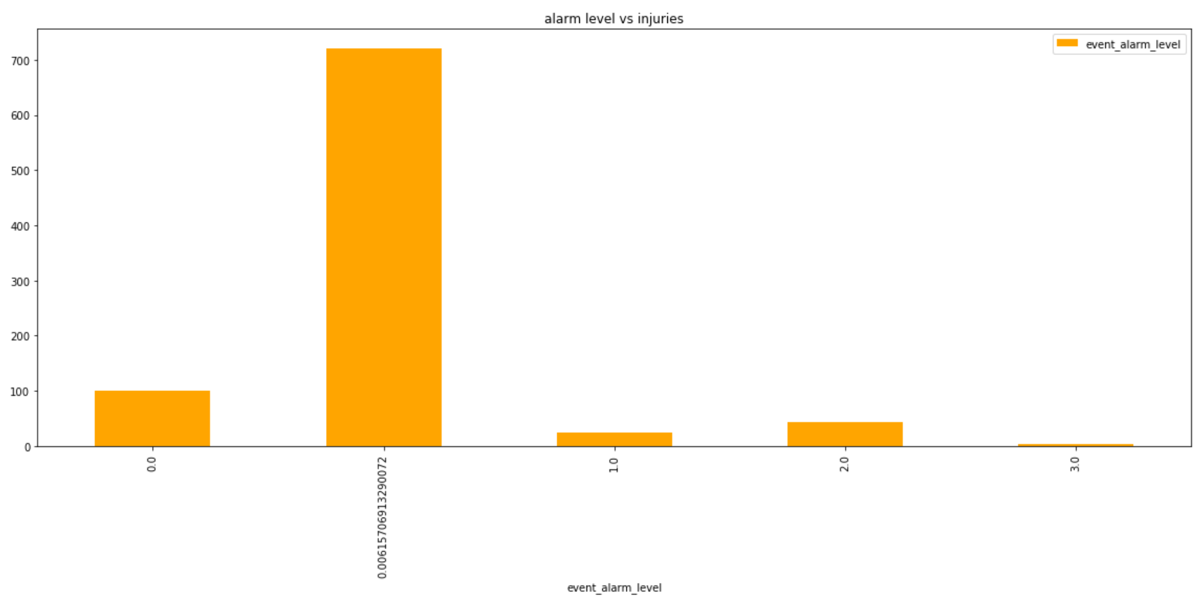
```
(43, 59)
```

In [33]:
```
label_vs_injuries(df, 'event_alarm_level', 'alarm level vs no injuries ', 'ala
rm level vs injuries')
```

```
0.00615706913290072
Saving figure alarm level vs no injuries
```




```
Saving figure alarm level vs injuries
```



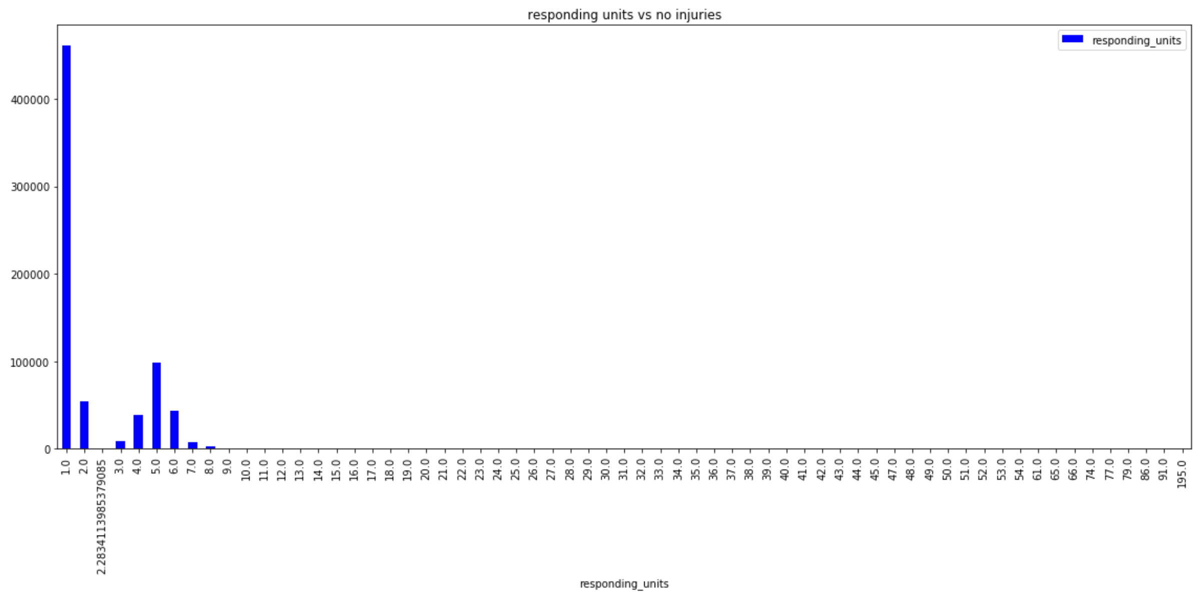
Responding Units vs Injuries

In [46]:
```
res_df = df[(df['responding_units'] == 22.0) & (df['total_inj_fatality'] == 0
)]
print(res_df.shape)
```

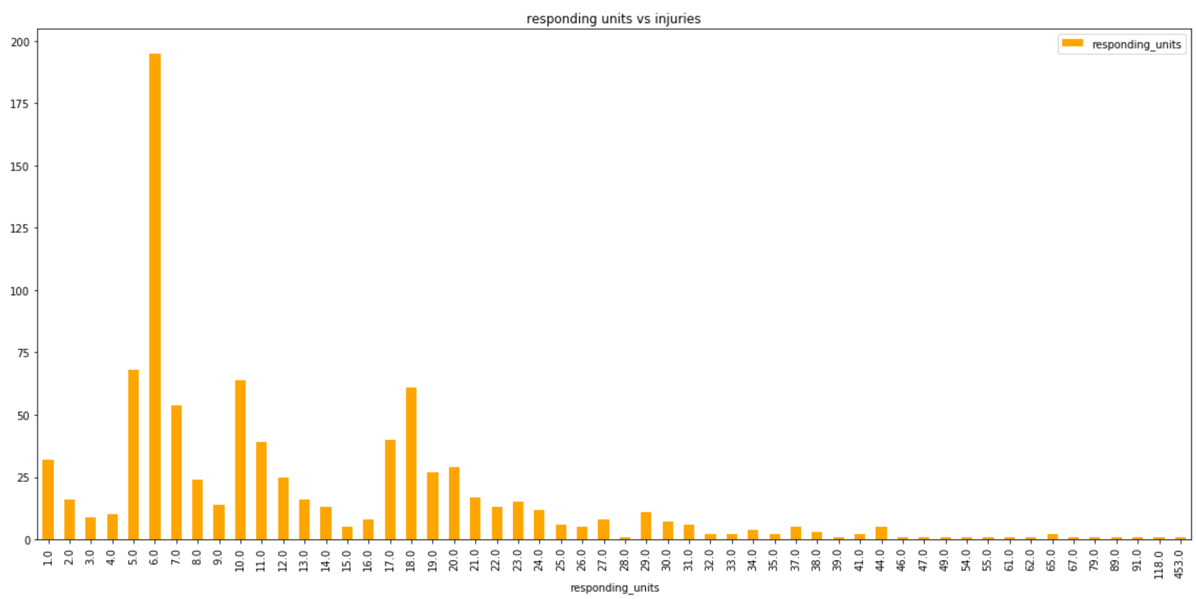
```
(40, 59)
```

In [41]:
```
label_vs_injuries(df, 'responding_units','responding units vs no injuries', 'r
esponding units vs injuries')a
```

```
2.2834113985379085
Saving figure responding units vs no injuries
```




```
Saving figure responding units vs injuries
```



Making csv

In [75]:
```
def fill_na(df, label):
    if(df[label].isna().sum()/len(df[label]) *100 > 0.0):
        _v = df[label].mean()
        print(label + ': ' + str(_v))
        df[label].fillna(value=_v, inplace=True)
```

In [52]:
```
df_tt_min['rescues'].isna().sum()/len(df_tt_min['rescues']) *100 > 0.0
```
Out[52]:
```
False
```

In [53]:
```
_v = df_tt_min['rescues'].mean()
print(_v)
df_tt_min['rescues'].fillna(value=_v, inplace=True)
```
```
0.026537751433291224
```

In [54]:
```
df_tt_min['rescues_unscaled'] = df_tt_min['rescues']
```

In [55]:
```
rescues_scaled = scaler.fit_transform(df_tt_min[['rescues_unscaled']])
```

In [56]:
```
rescues_scaled.shape
```
Out[56]:
```
(720370, 1)
```

In [57]:
```
df_tt_min['rescues_scaled'] = rescues_scaled
```

In [58]:
```
df_final = df_tt_min[['rescues_unscaled','rescues_scaled','min_to_reach_unscal
ed','min_to_reach_scaled','incident_number']]
```

In [64]:
```
df_final.to_csv('./dataset/TFSDataSetWithTotalFatality_Adnan.csv')
```

Ashok's Exploratory Analysis

Data exploration and analysis

In [38]:
```
list_of_columns = ['opp', 'moe','tssa','esa','mol','ems','canutec',
           'gas','hydro','municipal_building_office','municipal_health_office'
,
           'municipal_police','other']
```

In [39]:
```
def recode_empty_cells(dataframe, list_of_columns):

    for column in list_of_columns:
      dataframe[column] = dataframe[column].replace(r'\s+', np.nan, regex=True
)
      dataframe[column] = dataframe[column].fillna(0)

    return dataframe
```

In [40]:
```
recode_empty_cells(df, list_of_columns)
```

Out[40]:

	Unnamed: 0	Unnamed: 0.1	aid_to_from_other_depts	alarm_to_fd	arrive_date	bld_height	ca
0	0	0	4	3.0	2011-01-01 00:10:02	0	
1	1	1	4	1.0	2011-01-01 00:09:02	0	
2	2	2	4	3.0	2011-01-01 00:09:34	0	
3	3	3	4	1.0	2011-01-01 00:10:46	0	
4	4	4	1	5.0	2011-01-01 00:11:03	0	
5	5	5	4	1.0	2011-01-01 00:13:46	0	
6	6	6	4	1.0	2011-01-01 00:12:54	0	
7	7	7	4	3.0	2011-01-01 00:12:43	0	
8	8	8	4	3.0	2011-01-01 00:15:44	0	
9	9	9	4	4.0	2011-01-01 00:14:28	0	
10	10	10	4	1.0	2011-01-01 00:20:27	0	
11	11	11	4	3.0	2011-01-01 00:16:39	0	
12	12	12	4	5.0	2011-01-01 00:20:47	0	
13	13	13	4	3.0	2011-01-01 00:25:07	0	
14	14	14	4	5.0	2011-01-01 00:25:26	0	
15	15	15	4	3.0	2011-01-01 00:26:38	0	
16	16	16	4	1.0	2011-01-01 00:29:06	0	
17	17	17	4	3.0	2011-01-01 00:26:44	0	
18	18	18	4	3.0	2011-01-01 00:30:28	0	
19	19	19	1	5.0	2011-01-01 00:35:12	0	
20	20	20	4	3.0	2011-01-01 00:34:21	0	
21	21	21	4	3.0	2011-01-01 00:38:34	0	
22	22	22	4	3.0	2011-01-01 00:39:10	0	

	Unnamed: 0	Unnamed: 0.1	aid_to_from_other_depts	alarm_to_fd	arrive_date	bld_height	ca
23	23	23	4	1.0	2011-01-01 00:40:17	0	
24	24	24	4	3.0	2011-01-01 00:47:08	0	
25	25	25	4	5.0	2011-01-01 00:44:37	0	
26	26	26	4	3.0	2011-01-01 00:51:39	0	
27	27	27	4	3.0	2011-01-01 00:51:01	0	
28	28	28	4	5.0	2011-01-01 00:52:15	0	
29	29	29	4	3.0	2011-01-01 00:57:13	0	
...	...	...	...	...	...	...	
720340	720340	117396	4	1.0	2016-12-31 22:33:33	0	
720341	720341	117397	4	3.0	2016-12-31 22:36:58	0	
720342	720342	117398	4	3.0	2016-12-31 22:39:30	0	
720343	720343	117399	4	3.0	2016-12-31 22:43:29	0	
720344	720344	117400	4	5.0	2016-12-31 22:47:21	0	
720345	720345	117401	4	2.0	2016-12-31 22:47:07	0	
720346	720346	117402	4	5.0	2016-12-31 22:51:37	0	
720347	720347	117403	4	3.0	2016-12-31 22:57:00	0	
720348	720348	117404	4	4.0	2016-12-31 22:58:30	0	
720349	720349	117405	4	1.0	2016-12-31 23:04:21	0	
720350	720350	117406	4	3.0	2016-12-31 23:04:11	0	
720351	720351	117407	4	3.0	NaN	0	
720352	720352	117408	4	3.0	2016-12-31 23:13:13	0	
720353	720353	117409	4	3.0	2016-12-31 23:16:15	0	
720354	720354	117410	4	5.0	2016-12-31 23:21:18	0	

	Unnamed: 0	Unnamed: 0.1	aid_to_from_other_depts	alarm_to_fd	arrive_date	bld_height	ca
720355	720355	117411	4	1.0	2016-12-31 23:23:58	0	
720356	720356	117412	4	3.0	2016-12-31 23:29:29	0	
720357	720357	117413	4	3.0	2016-12-31 23:32:24	0	
720358	720358	117414	4	3.0	2016-12-31 23:32:30	0	
720359	720359	117415	4	3.0	2016-12-31 23:32:38	0	
720360	720360	117416	4	3.0	2016-12-31 23:37:06	0	
720361	720361	117417	4	3.0	2016-12-31 23:53:02	0	
720362	720362	117418	4	3.0	2016-12-31 23:57:24	0	
720363	720363	117419	4	3.0	2016-12-31 23:51:31	0	
720364	720364	117420	4	3.0	2016-12-31 23:55:31	0	
720365	720365	117421	4	1.0	2016-12-31 23:57:28	0	
720366	720366	117422	4	3.0	2017-01-01 00:01:46	0	
720367	720367	117423	4	3.0	2017-01-01 00:02:27	0	
720368	720368	117424	4	5.0	2017-01-01 00:04:56	0	
720369	720369	117425	4	3.0	2017-01-01 00:03:54	0	

720370 rows × 59 columns

In [41]:
```
df.drop("Unnamed: 0", axis=1, inplace=True)
```

In [42]:
```
df.drop("Unnamed: 0.1", axis=1, inplace=True)
```

In [43]:
```
#Convert the string to datetime format
df['incident_date'] = pd.to_datetime(df['incident_date'])
```

In [44]:
```
df.dtypes
```

Out[44]:
```
aid_to_from_other_depts                int64
alarm_to_fd                          float64
arrive_date                           object
bld_height                             int64
canutec                               object
control_date                          object
cross_street                          object
dispatch_date                         object
dispatch_hour                        float64
dispatch_min                         float64
dispatch_sec                         float64
ems                                   object
esa                                   object
est_km                                 int64
est_loss                               int64
est_num_persons_displaced              int64
event_alarm_level                    float64
event_type                            object
event_type_cd                         object
fd_station                            object
fire_dept_incident                    object
fsa                                   object
gas                                   object
hydro                                 object
incident_date                 datetime64[ns]
incident_number                       object
initial_call_hour                      int64
initial_call_min                       int64
initial_call_sec                       int64
initial_unit_personnel                 int64
main_street                           object
moe                                   object
mol                                   object
municipal_building_office             object
municipal_health_office               object
municipal_police                      object
ofm_investigations_contacted           int64
onscene_hour                         float64
onscene_min                          float64
onscene_sec                          float64
opp                                   object
other                                 object
property                              object
rescued_adults                         int64
rescued_children                       int64
rescued_seniors                        int64
rescues                                int64
responding_units                     float64
response_type                          int64
smoke_alarm_impact_on_num_evac         int64
total_num_personnel                    int64
tssa                                  object
ff_injuries                            int64
ff_fatalities                          int64
civilian_fire_injury                   int64
civilian_fire_fatality                 int64
```

```
total_inj_fatality                        float64
dtype: object
```

Day-of-Week vs Injuries

In [45]:
```
df['dow'] = df['incident_date'].apply(lambda x: x.date().weekday())
df['is_weekend'] = df['incident_date'].apply(lambda x: 1 if x.date().weekday()
 in (5, 6) else 0)
```

In [46]:
```
df.head()
```

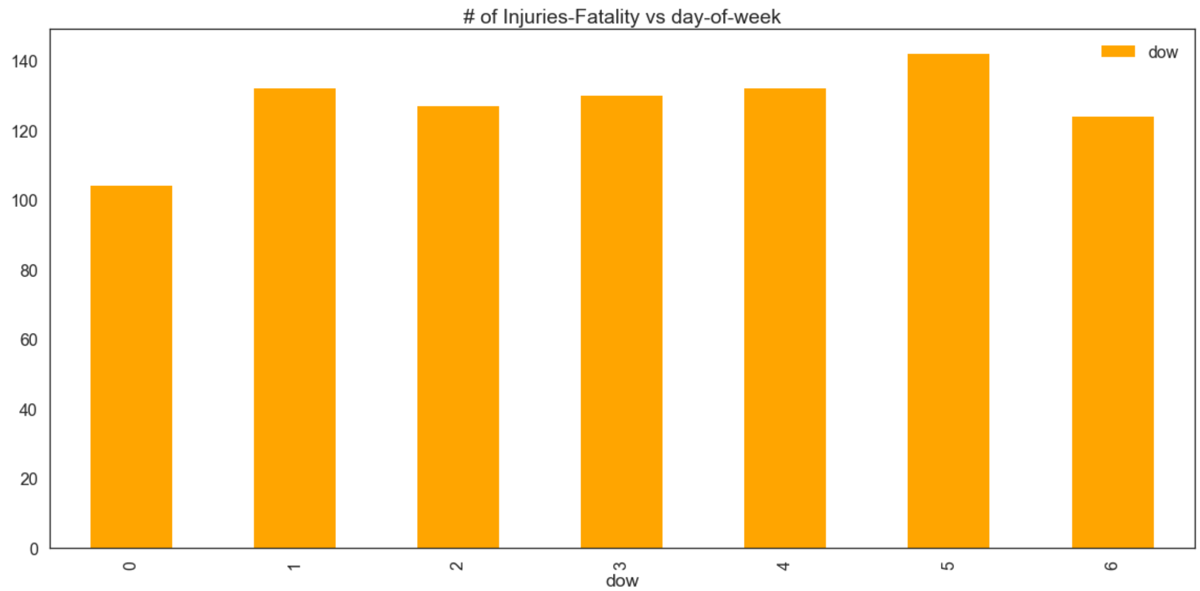
Out[46]:

	aid_to_from_other_depts	alarm_to_fd	arrive_date	bld_height	canutec	control_date	cross_st
0	4	3.0	2011-01-01 00:10:02	0	0	NaN	
1	4	1.0	2011-01-01 00:09:02	0	0	NaN	LAWREI AV
2	4	3.0	2011-01-01 00:09:34	0	0	NaN	
3	4	1.0	2011-01-01 00:10:46	0	0	NaN	SYL
4	1	5.0	2011-01-01 00:11:03	0	0	NaN	VAROV

In [47]:
```
df_dow_inj = df[['dow', 'total_inj_fatality']]
df_dow_inj_1 = get_injuries(df_dow_inj)
df_dow_inj_0 = get_noinjuries(df_dow_inj)
```

In [48]:
```
plotbar_1_3('dow','# of Injuries-Fatality vs day-of-week', df_dow_inj_1)
# Monday is 0, Sunday is 6
save_fig('Injuries_Fatality vs day of week chart1')
```

```
Saving figure # of Injuries-Fatality vs day-of-week
```




```
Saving figure Injuries_Fatality vs day of week chart1
<matplotlib.figure.Figure at 0x1a26ac8d30>
```

Day-of-Week in Year vs Injuries

In [49]:
```
df['year'] = df['incident_date'].apply(lambda x: x.date().year)
```

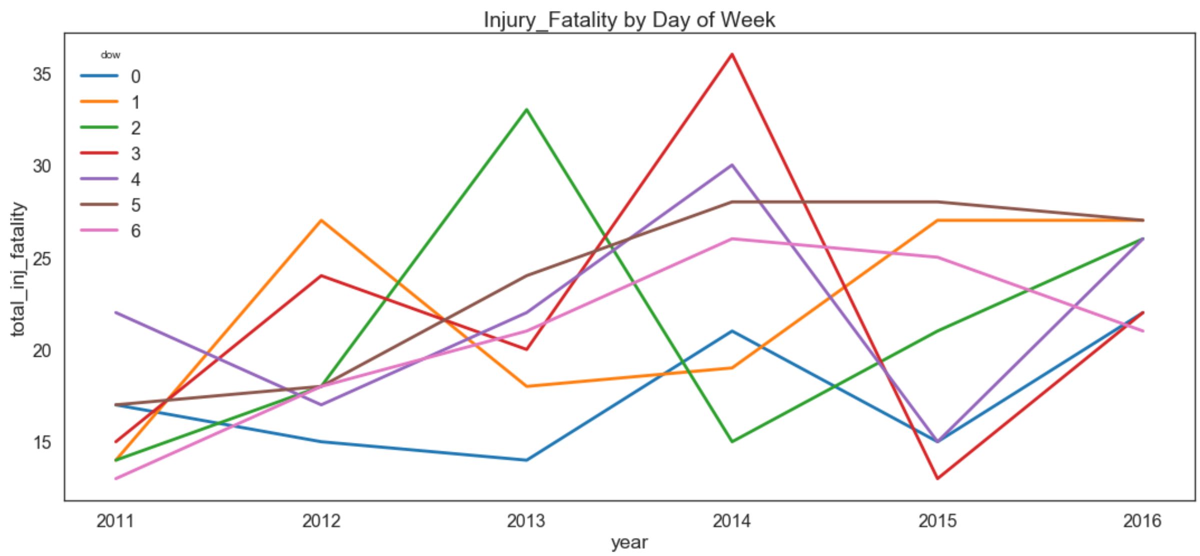
In [50]:
```
df.head()
```

Out[50]:

	aid_to_from_other_depts	alarm_to_fd	arrive_date	bld_height	canutec	control_date	cross_st
0	4	3.0	2011-01-01 00:10:02	0	0	NaN	
1	4	1.0	2011-01-01 00:09:02	0	0	NaN	LAWREI AV
2	4	3.0	2011-01-01 00:09:34	0	0	NaN	
3	4	1.0	2011-01-01 00:10:46	0	0	NaN	SYL
4	1	5.0	2011-01-01 00:11:03	0	0	NaN	VARO\

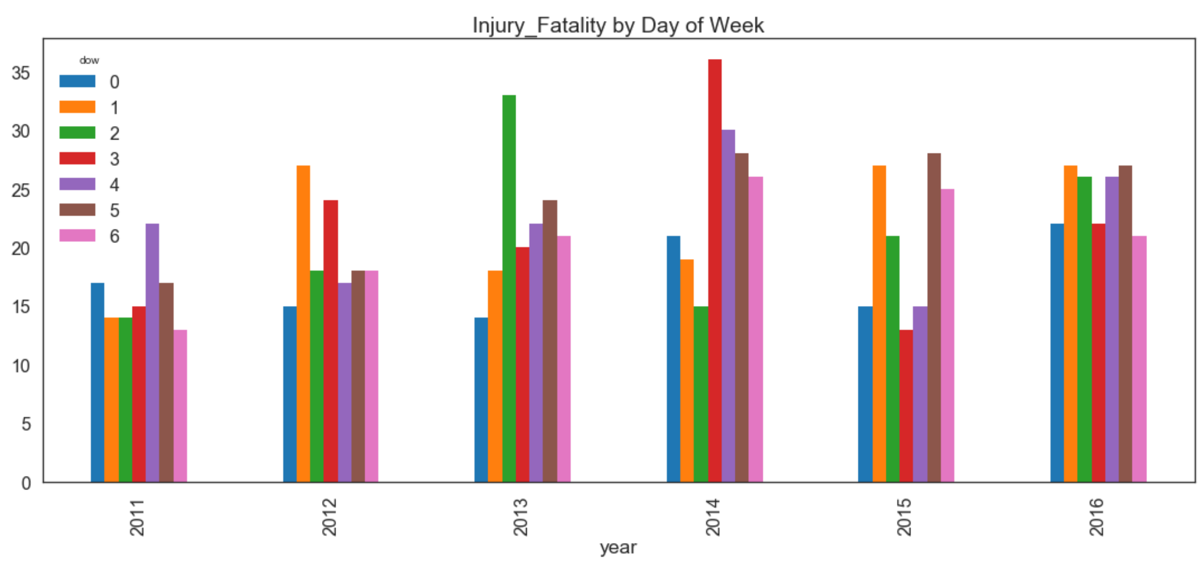
In [51]:
```
#plot data
fig, ax = plt.subplots(figsize=(15,7))
df.groupby(['year', 'dow'])['total_inj_fatality'].sum().unstack().plot(ax=ax,
title = 'Injury_Fatality by Day of Week')
plt.ylabel('total_inj_fatality')
save_fig('Number Injuries_Fatality vs Day of Week  ')
```

```
Saving figure Number Injuries_Fatality vs Day of Week
```



In [52]:
```
fig, ax = plt.subplots(figsize=(15,7))
df.groupby(['year', 'dow'])['total_inj_fatality'].sum().unstack().plot(kind =
'bar', ax=ax,
                                                                        title =
'Injury_Fatality by Day of Week')

save_fig('Injuries_Fatality vs Day of Week  ')
```

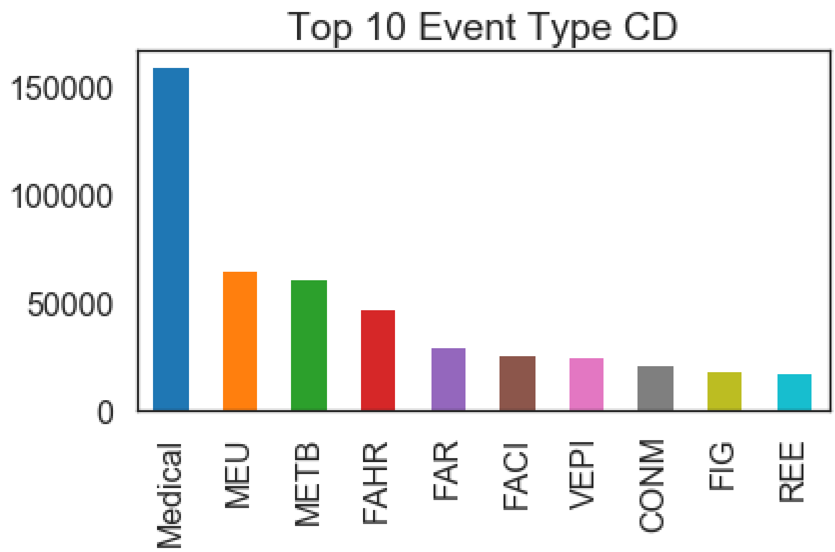
```
Saving figure Injuries_Fatality vs Day of Week
```



Top 10 Event_Type_CD and Event_Type

In [53]:
```
df['event_type_cd'].value_counts()[:10].plot(kind='bar', title = 'Top 10 Event
 Type CD')
save_fig('Top 10 Event Type CD')
```

```
Saving figure Top 10 Event Type CD
```



In [54]:
```
df['event_type_cd'].value_counts()
```

Out[54]:
```
Medical     158831
MEU          64176
METB         60585
FAHR         46474
FAR          28735
FACI         25573
VEPI         24852
CONM         20504
FIG          18462
REE          17365
MECP         16076
FIR          13976
VEPIH        13761
MEO          12386
CC           11658
FAHRD        10958
TEMS         10686
FAI          10275
FIHR          9633
NGASLK        9550
MESC          9529
WDH           7931
FACC          7878
MEA           7831
MEV           7074
HAZ1          6485
FAHCD         5795
VEAT          5735
VEF           5641
WAT           5526
             ...
VEFHE           29
REWP            27
RETR            20
FAW             19
LKFI            19
TEST            18
FIWMI           17
ISHAZ1          16
ISHAZ2          13
HAZ3            13
LKME             9
PAJO             5
FIWH             5
FIW              4
WKFIRE           3
LKRT             3
PAS              2
VEFP             2
REENE            2
ISHZ             2
MAAF             2
PASCBANE         2
ISPCF            2
VEFNG            1
REMT             1
PASCBA           1
```

```
PABOMBNE         1
PAJONE           1
MEPACP           1
MAAM             1
Name: event_type_cd, Length: 130, dtype: int64
```

In [55]:
```
eventtype_cd = df['event_type_cd'].value_counts()[:10]
```

In [56]:
```
eventtype_cd = eventtype_cd.rename_axis('evtype').reset_index(name='counts')
```

In [57]:
```
evtype = eventtype_cd['evtype'].tolist()
```

In [58]:
```
evtype
```

Out[58]:
```
['Medical', 'MEU', 'METB', 'FAHR', 'FAR', 'FACI', 'VEPI', 'CONM', 'FIG', 'RE
E']
```

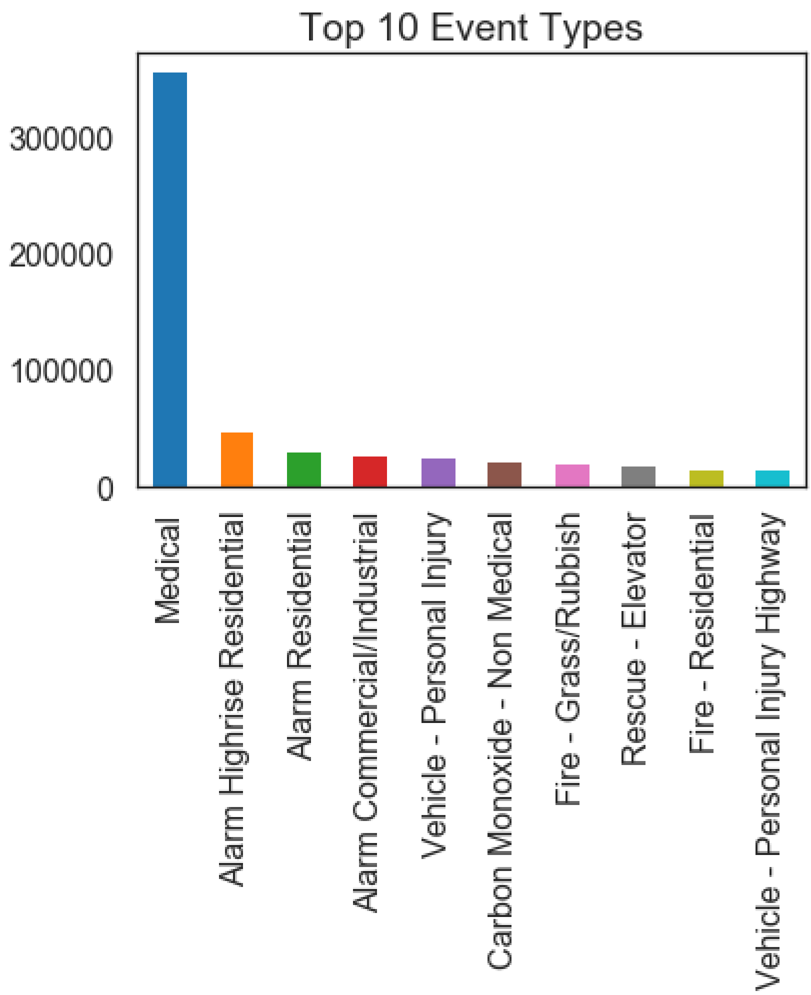
In [59]:
```
df[df.apply(lambda x: x['event_type_cd'] in eventtype_cd['evtype'], axis=1)]
```

Out[59]:

aid_to_from_other_depts	alarm_to_fd	arrive_date	bld_height	canutec	control_date	cross_str

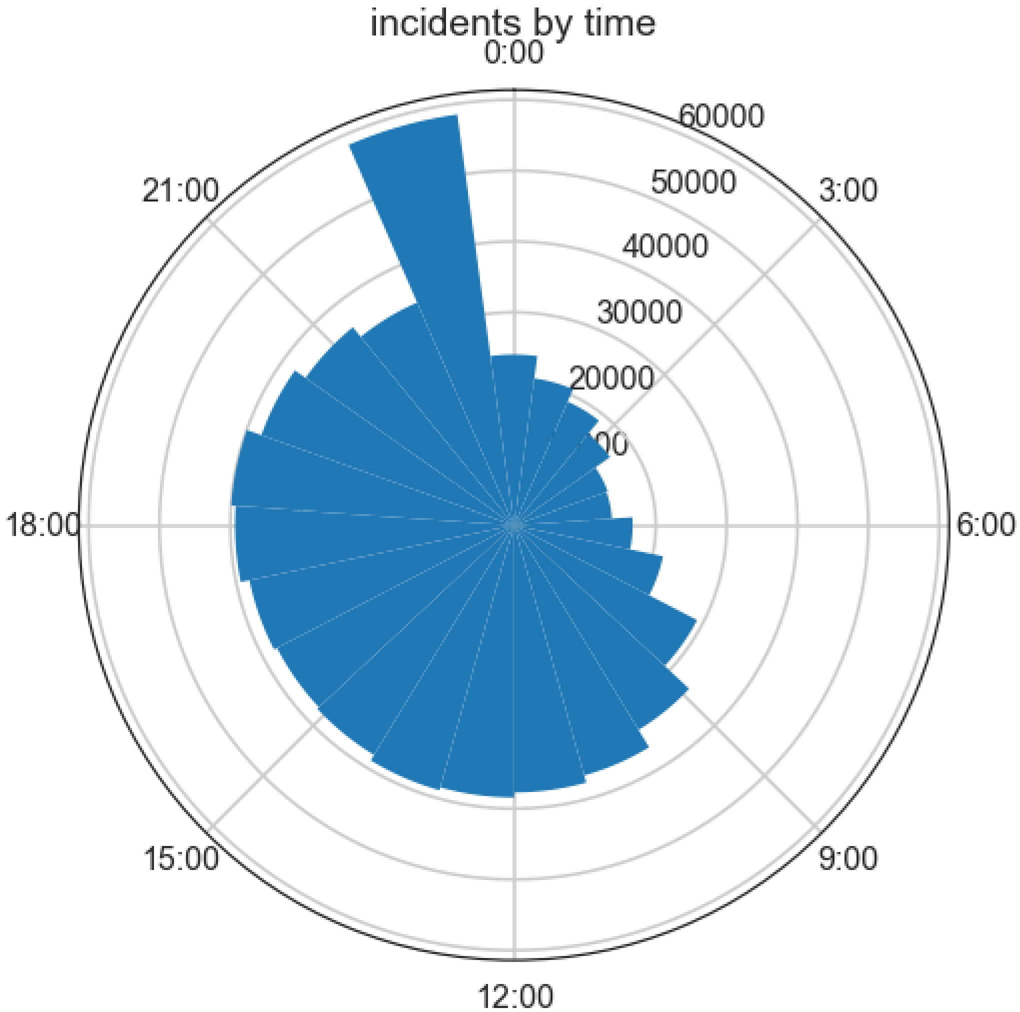
In [60]:
```
df['event_type'].value_counts()[:10].plot(kind='bar', title = 'Top 10 Event Ty
pes')
```

Out[60]: <matplotlib.axes._subplots.AxesSubplot at 0x1a27fae5f8>



Incidents by Time of Day

In [68]:
```
N = 23
bottom = 2
int_hour = df['initial_call_hour'].tolist()

# create theta for 24 hours
theta = np.linspace(0.0, 2 * np.pi, N, endpoint=False)

# make the histogram that bined on 24 hour
radii, tick = np.histogram(int_hour, bins = 23)

# width of each bin on the plot
width = (2*np.pi) / N

# make a polar plot
plt.figure(figsize = (12, 8))
ax = plt.subplot(111, polar=True)
bars = ax.bar(theta, radii, width=width, bottom=bottom)

# set the lable go clockwise and start from the top
ax.set_theta_zero_location("N")
# clockwise
ax.set_theta_direction(-1)

# set the label
ticks = ['0:00', '3:00', '6:00', '9:00', '12:00', '15:00', '18:00', '21:00']
ax.set_xticklabels(ticks)
plt.title('incidents by time')
plt.show()
save_fig('Incidents by Time')
```




```
Saving figure Incidents by Time
<matplotlib.figure.Figure at 0x1a27e7f080>
```

Making csv of the selected columns

In [ ]:
```
ashok_df = df.copy()
ashok_df = ashok_df[['event_alarm_level','responding_units','ofm_investigations_contacted','aid_to_from_other_depts']]
ashok_df.head()
```

In [14]:
```
fill_na(ashok_df, 'event_alarm_level')
fill_na(ashok_df, 'responding_units')
fill_na(ashok_df, 'ofm_investigations_contacted')
fill_na(ashok_df, 'aid_to_from_other_depts')
```

```
event_alarm_level: 0.00615706913290072
responding_units: 2.2834113985379085
```

In [15]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [16]:
```
scaler = MinMaxScaler()
scaled_units = scaler.fit_transform(ashok_df[['responding_units']])
ashok_df['responding_units_scaled'] = scaled_units
scaled_aid = scaler.fit_transform(ashok_df[['aid_to_from_other_depts']])
ashok_df['aid_to_from_other_depts_scaled'] = scaled_aid
```

In [17]:
```
event_encoded = pd.get_dummies(ashok_df['event_alarm_level'])
```

In [24]:
```
final_ashok_df = ashok_df.join(event_encoded)
```

In [25]:
```
final_ashok_df.head()
```

Out[25]:

	event_alarm_level	responding_units	ofm_investigations_contacted	aid_to_from_other_depts	re
0	0.006157	1.0	0	4	
1	0.006157	1.0	0	4	
2	0.006157	1.0	0	4	
3	0.006157	1.0	0	4	
4	0.006157	4.0	0	1	

In [26]:
```
final_ashok_df.to_csv('./dataset/TFSDataSetWithTotalFatality_Ashok.csv')
```

In [27]:
```
test = pd.read_csv('./dataset/TFSDataSetWithTotalFatality_Ashok.csv')
test = test.join(df['incident_number'])
```

In [30]:
```
test.to_csv('./dataset/TFSDataSetWithTotalFatality_Ashok.csv')
```

In [ ]:
```

```

Arjun's Exploratory Analysis

Data exploration ana analysis

In [1]:
```
#Import the libraries
import pandas as pd
import numpy as np
import os
from IPython.display import display
pd.set_option('display.max_columns',200)
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
```

In [2]:
```
#Load the file
pure_df = pd.read_csv('C:/Users/Arjun/Documents/teamproject_3253/dataset/TFSDataSetWithTotalFatality.csv', low_memory=False)
df = pure_df.copy()

def f(row):
    val = 0
    if row['opp'] == '1':
        val = 1
    elif row['moe'] == '1':
        val = 1
    elif row['tssa'] == '1':
        val = 1
    elif row['esa'] == '1':
        val = 1
    elif row['mol'] == '1':
        val = 1
    elif row['ems'] == '1':
        val = 1
    elif row['canutec'] == '1':
        val = 1
    elif row['gas'] == '1':
        val = 1
    elif row['hydro'] == '1':
        val = 1
    elif row['municipal_building_office'] == '1':
        val = 1
    elif row['municipal_health_office'] == '1':
        val = 1
    elif row['municipal_police'] == '1':
        val = 1
    return val

df['contacted'] = df.apply(f, axis=1)
df.info()
print("Done...")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 720370 entries, 0 to 720369
Data columns (total 60 columns):
Unnamed: 0                        720370 non-null int64
Unnamed: 0.1                      720370 non-null int64
aid_to_from_other_depts           720370 non-null int64
alarm_to_fd                       720369 non-null float64
arrive_date                       703568 non-null object
bld_height                        720370 non-null int64
canutec                           720370 non-null object
control_date                      241537 non-null object
cross_street                      330021 non-null object
dispatch_date                     719679 non-null object
dispatch_hour                     720193 non-null float64
dispatch_min                      720193 non-null float64
dispatch_sec                      720193 non-null float64
ems                               720370 non-null object
esa                               720370 non-null object
est_km                            720370 non-null int64
est_loss                          720370 non-null int64
est_num_persons_displaced         720370 non-null int64
event_alarm_level                 117426 non-null float64
event_type                        720290 non-null object
event_type_cd                     720338 non-null object
fd_station                        720370 non-null object
fire_dept_incident                344884 non-null object
fsa                               365309 non-null object
gas                               720370 non-null object
hydro                             720370 non-null object
incident_date                     720370 non-null object
incident_number                   720338 non-null object
initial_call_hour                 720370 non-null int64
initial_call_min                  720370 non-null int64
initial_call_sec                  720370 non-null int64
initial_unit_personnel            720370 non-null int64
main_street                       354910 non-null object
moe                               720370 non-null object
mol                               720370 non-null object
municipal_building_office         720370 non-null object
municipal_health_office           720370 non-null object
municipal_police                  720370 non-null object
ofm_investigations_contacted      720370 non-null int64
onscene_hour                      704999 non-null float64
onscene_min                       704999 non-null float64
onscene_sec                       704999 non-null float64
opp                               720370 non-null object
other                             720370 non-null object
property                          710507 non-null object
rescued_adults                    720370 non-null int64
rescued_children                  720370 non-null int64
rescued_seniors                   720370 non-null int64
rescues                           720370 non-null int64
responding_units                  720338 non-null float64
response_type                     720370 non-null int64
smoke_alarm_impact_on_num_evac    720370 non-null int64
total_num_personnel               720370 non-null int64
tssa                              720370 non-null object
```

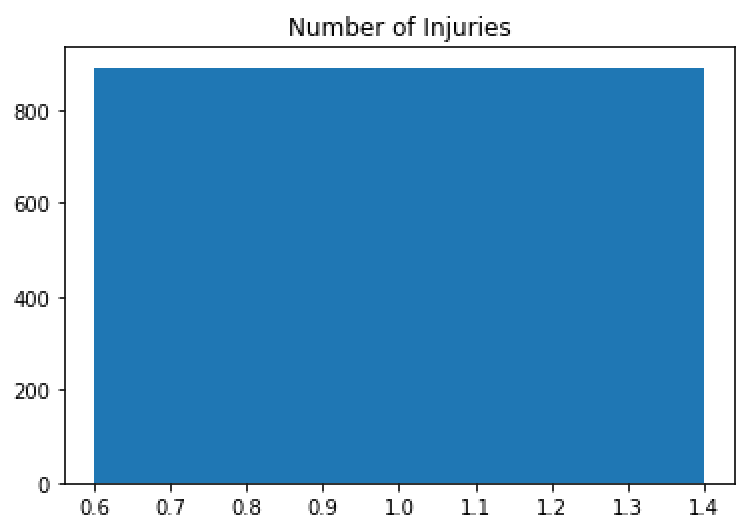
```
ff_injuries                         720370 non-null int64
ff_fatalities                       720370 non-null int64
civilian_fire_injury                720370 non-null int64
civilian_fire_fatality              720370 non-null int64
total_inj_fatality                  720370 non-null int64
contacted                           720370 non-null int64
dtypes: float64(9), int64(25), object(26)
memory usage: 329.8+ MB
Done...
```

Check for relationship to Injury - Graphs

In [107]:

```
print("Total Injuries value count:" , df['total_inj_fatality'].value_counts())
print("Total Injuries is null:", df['total_inj_fatality'].isnull().sum())

plt.title("Number of Injuries")
x = [1]
y = [df['total_inj_fatality'].sum(), ]
plt.bar(x,y)
plt.show()
```

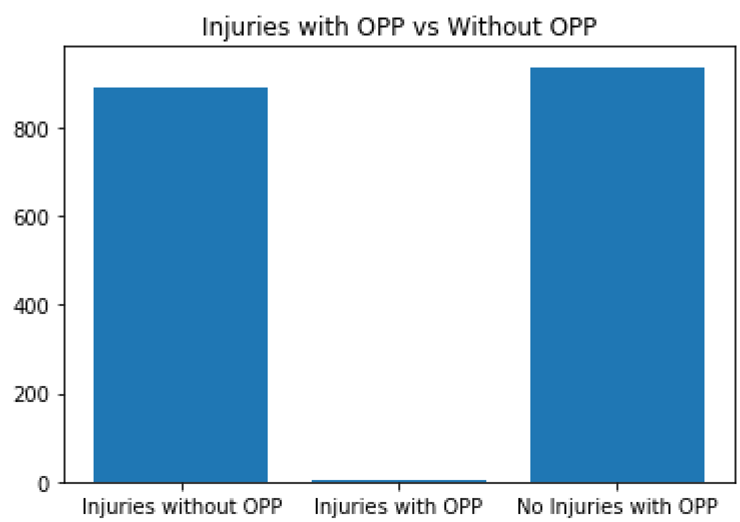
```
Total Injuries value count: 0    719479
1       891
Name: total_inj_fatality, dtype: int64
Total Injuries is null: 0
```



Plotting OPP Contacted vs injuries / no injuries

In [108]:
```
print("OPP value count:" , df['opp'].value_counts())
print("OPP Injuries is null:", df['opp'].isnull().sum())

inj_nopp = df[(df.total_inj_fatality == 1) & (df.opp != '1') ]
inj_opp =  df[(df.total_inj_fatality == 1) & (df.opp == '1') ]
#ninj_nopp = df[(df.total_inj_fatality != 1) & (df.opp != '1') ]
ninj_opp = df[(df.total_inj_fatality != 1) & (df.opp == '1') ]

x = [1,2,3]
y = [inj_nopp.total_inj_fatality.sum(), inj_opp.total_inj_fatality.sum(),
     ninj_opp.total_inj_fatality.count()]
print(y)
plt.title("Injuries with OPP vs Without OPP")
obj = ('Injuries without OPP','Injuries with OPP', 'No Injuries with OPP')
plt.bar(x,y, tick_label=obj)
plt.show()
```

```
OPP value count:      719431
1       939
Name: opp, dtype: int64
OPP Injuries is null: 0
[888, 3, 936]
```



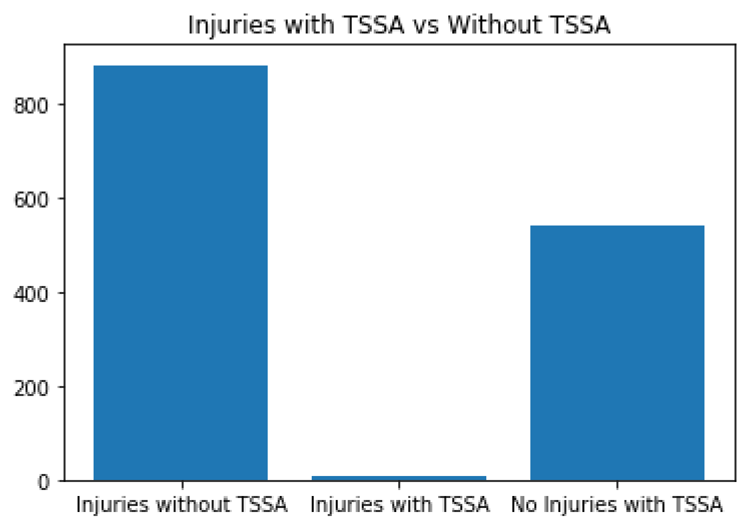
Plotting MOE Contacted vs injuries / no injuries

In [ ]:
```
print("MOE value count:" , df['moe'].value_counts())
print("MOE Injuries is null:", df['moe'].isnull().sum())

inj_nmoe = df[(df.total_inj_fatality == 1) & (df.moe != '1') ]
inj_moe =  df[(df.total_inj_fatality == 1) & (df.moe == '1') ]
#ninj_nopp = df[(df.total_inj_fatality != 1) & (df.opp != '1') ]
ninj_moe = df[(df.total_inj_fatality != 1) & (df.moe == '1') ]

x = [1,2,3]
y = [inj_nmoe.total_inj_fatality.sum(), inj_moe.total_inj_fatality.sum(),
     ninj_moe.total_inj_fatality.count()]
print(y)
plt.title("Injuries with MOE vs Without MOE")
obj = ('Injuries without MOE','Injuries with MOE', 'No Injuries with MOE')
plt.bar(x,y, tick_label=obj)
plt.show()
```

Plotting TSSA Contacted vs injuries / no injuries

In [110]:
```
print("TSSA value count:" , df['tssa'].value_counts())
print("TSSA Injuries is null:", df['tssa'].isnull().sum())

inj_ntssa = df[(df.total_inj_fatality == 1) & (df.tssa != '1') ]
inj_tssa =  df[(df.total_inj_fatality == 1) & (df.tssa == '1') ]
#ninj_nopp = df[(df.total_inj_fatality != 1) & (df.opp != '1') ]
ninj_tssa = df[(df.total_inj_fatality != 1) & (df.tssa == '1') ]

x = [1,2,3]
y = [inj_ntssa.total_inj_fatality.sum(), inj_tssa.total_inj_fatality.sum(),
     ninj_tssa.total_inj_fatality.count()]
print(y)
plt.title("Injuries with TSSA vs Without TSSA")
obj = ('Injuries without TSSA','Injuries with TSSA', 'No Injuries with TSSA')
plt.bar(x,y, tick_label=obj)
plt.show()
```

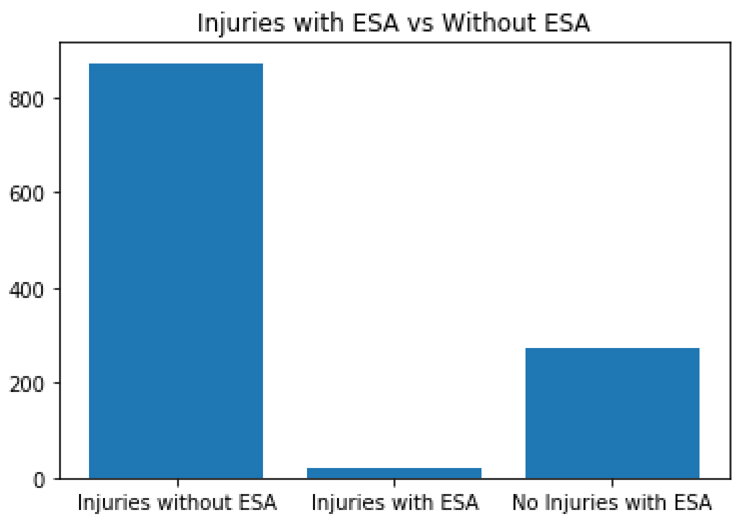
```
TSSA value count:      719822
1       548
Name: tssa, dtype: int64
TSSA Injuries is null: 0
[882, 9, 539]
```



Plotting ESA Contacted vs injuries / no injuries

In [111]:
```
print("ESA value count:" , df['esa'].value_counts())
print("ESA Injuries is null:", df['esa'].isnull().sum())

inj_nesa = df[(df.total_inj_fatality == 1) & (df.esa != '1') ]
inj_esa =  df[(df.total_inj_fatality == 1) & (df.esa == '1') ]
#ninj_nopp = df[(df.total_inj_fatality != 1) & (df.opp != '1') ]
ninj_esa = df[(df.total_inj_fatality != 1) & (df.esa == '1') ]

x = [1,2,3]
y = [inj_nesa.total_inj_fatality.sum(), inj_esa.total_inj_fatality.sum(),
     ninj_esa.total_inj_fatality.count()]
print(y)
plt.title("Injuries with ESA vs Without ESA")
obj = ('Injuries without ESA','Injuries with ESA', 'No Injuries with ESA')
plt.bar(x,y, tick_label=obj)
plt.show()
```

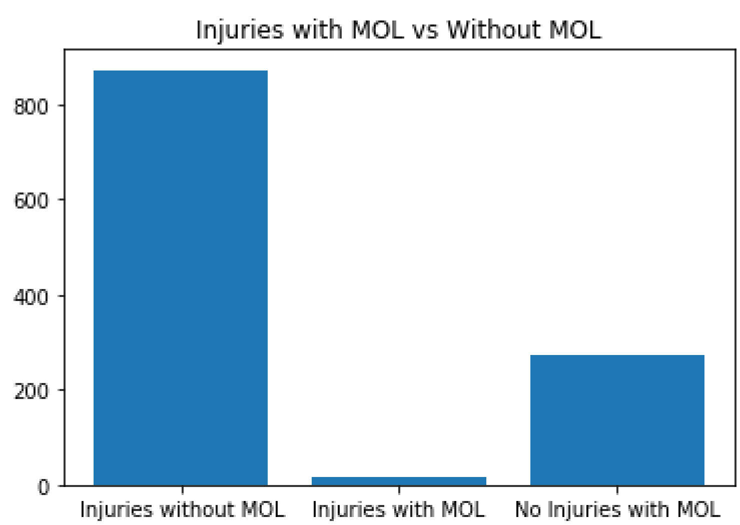
```
ESA value count:      720080
1       290
Name: esa, dtype: int64
ESA Injuries is null: 0
[872, 19, 271]
```



Plotting MOL Contacted vs injuries / no injuries

In [112]:
```
print("MOL value count:" , df['mol'].value_counts())
print("MOL Injuries is null:", df['mol'].isnull().sum())

inj_nmol = df[(df.total_inj_fatality == 1) & (df.mol != '1') ]
inj_mol =  df[(df.total_inj_fatality == 1) & (df.mol == '1') ]
#ninj_nopp = df[(df.total_inj_fatality != 1) & (df.opp != '1') ]
ninj_mol = df[(df.total_inj_fatality != 1) & (df.mol == '1') ]

x = [1,2,3]
y = [inj_nmol.total_inj_fatality.sum(), inj_mol.total_inj_fatality.sum(),
     ninj_mol.total_inj_fatality.count()]
print(y)
plt.title("Injuries with MOL vs Without MOL")
obj = ('Injuries without MOL','Injuries with MOL', 'No Injuries with MOL')
plt.bar(x,y, tick_label=obj)
plt.show()
```

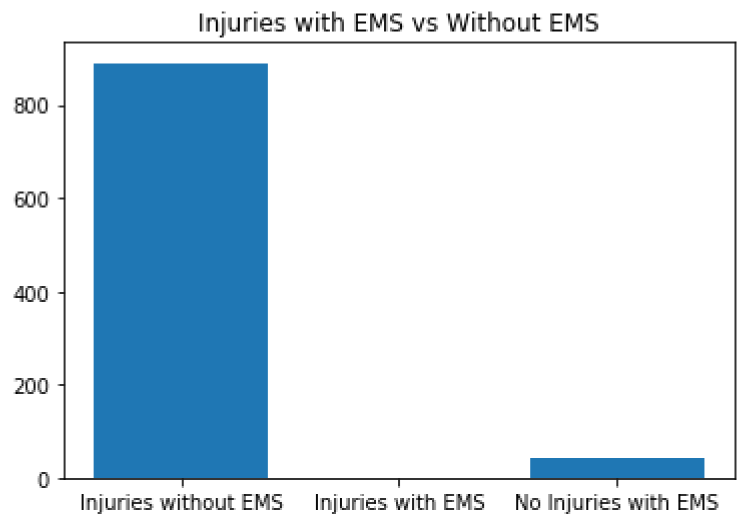
```
MOL value count:      720080
1       290
Name: mol, dtype: int64
MOL Injuries is null: 0
[873, 18, 272]
```



Plotting EMS Contacted vs injuries / no injuries

In [10]:
```
print("EMS value count:" , df['ems'].value_counts())
print("EMS Injuries is null:", df['ems'].isnull().sum())

inj_nems = df[(df.total_inj_fatality == 1) & (df.ems != '1') ]
inj_ems =  df[(df.total_inj_fatality == 1) & (df.ems == '1') ]
#ninj_nopp = df[(df.total_inj_fatality != 1) & (df.opp != '1') ]
ninj_ems = df[(df.total_inj_fatality != 1) & (df.ems == '1') ]

x = [1,2,3]
y = [inj_nems.total_inj_fatality.sum(), inj_ems.total_inj_fatality.sum(),
     ninj_ems.total_inj_fatality.count()]
print(y)
plt.title("Injuries with EMS vs Without EMS")
obj = ('Injuries without EMS','Injuries with EMS', 'No Injuries with EMS')
plt.bar(x,y, tick_label=obj)
plt.show()
```

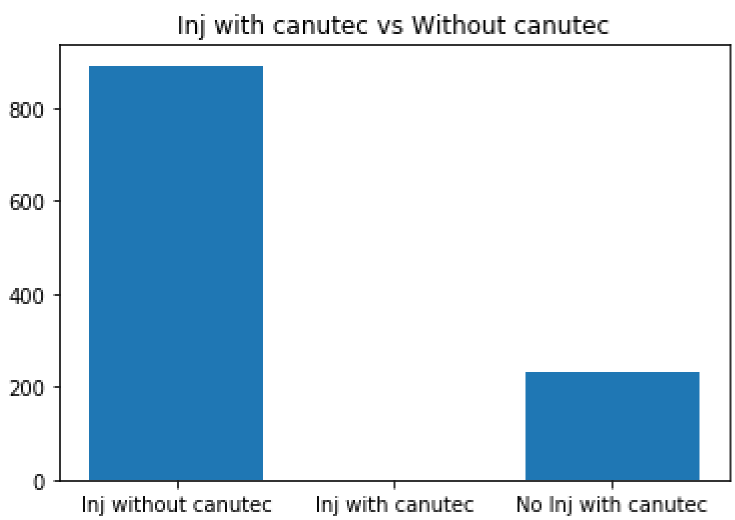
```
EMS value count:      720325
1        45
Name: ems, dtype: int64
EMS Injuries is null: 0
[890, 1, 44]
```



Plotting canutec Contacted vs injuries / no injuries

In [113]:
```
print("canutec value count:" , df['canutec'].value_counts())
print("canutec Injuries is null:", df['canutec'].isnull().sum())

inj_ncanutec = df[(df.total_inj_fatality == 1) & (df.canutec != '1') ]
inj_canutec =  df[(df.total_inj_fatality == 1) & (df.canutec == '1') ]
#ninj_nopp = df[(df.total_inj_fatality != 1) & (df.opp != '1') ]
ninj_canutec = df[(df.total_inj_fatality != 1) & (df.canutec == '1') ]

x = [1,2,3]
y = [inj_ncanutec.total_inj_fatality.sum(), inj_canutec.total_inj_fatality.sum
(),
     ninj_canutec.total_inj_fatality.count()]
print(y)
plt.title("Inj with canutec vs Without canutec")
obj = ('Inj without canutec','Inj with canutec', 'No Inj with canutec')
plt.bar(x,y, tick_label=obj)
plt.show()
```

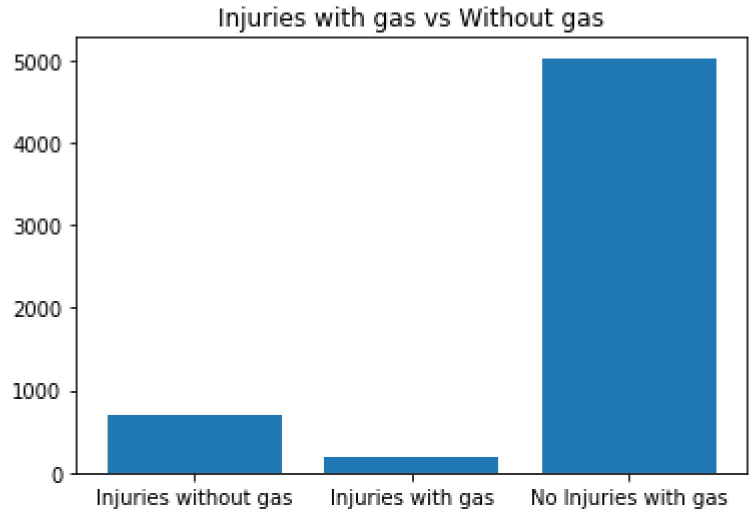
```
canutec value count:      720138
1       232
Name: canutec, dtype: int64
canutec Injuries is null: 0
[891, 0, 232]
```



Plotting gas Contacted vs injuries / no injuries

In [114]:
```
print("gas value count:" , df['gas'].value_counts())
print("gas Injuries is null:", df['gas'].isnull().sum())

inj_ngas = df[(df.total_inj_fatality == 1) & (df.gas != '1') ]
inj_gas =  df[(df.total_inj_fatality == 1) & (df.gas == '1') ]
#ninj_nopp = df[(df.total_inj_fatality != 1) & (df.opp != '1') ]
ninj_gas = df[(df.total_inj_fatality != 1) & (df.gas == '1') ]

x = [1,2,3]
y = [inj_ngas.total_inj_fatality.sum(), inj_gas.total_inj_fatality.sum(),
     ninj_gas.total_inj_fatality.count()]
print(y)
plt.title("Injuries with gas vs Without gas")
obj = ('Injuries without gas','Injuries with gas', 'No Injuries with gas')
plt.bar(x,y, tick_label=obj)
plt.show()
```

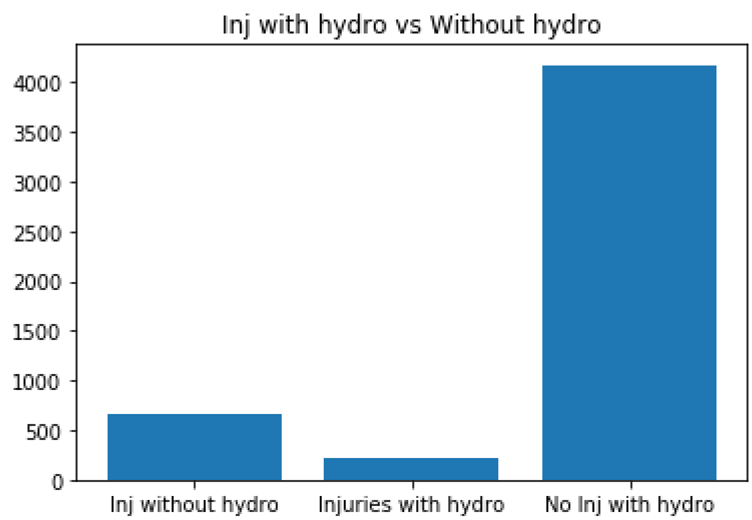
```
gas value count:      715150
1      5220
Name: gas, dtype: int64
gas Injuries is null: 0
[702, 189, 5031]
```



Plotting hydro Contacted vs injuries / no injuries

In [115]:
```
print("hydro value count:" , df['hydro'].value_counts())
print("hydro Injuries is null:", df['hydro'].isnull().sum())

inj_nhydro = df[(df.total_inj_fatality == 1) & (df.hydro != '1') ]
inj_hydro =  df[(df.total_inj_fatality == 1) & (df.hydro == '1') ]
#ninj_nopp = df[(df.total_inj_fatality != 1) & (df.opp != '1') ]
ninj_hydro = df[(df.total_inj_fatality != 1) & (df.hydro == '1') ]

x = [1,2,3]
y = [inj_nhydro.total_inj_fatality.sum(), inj_hydro.total_inj_fatality.sum(),
     ninj_hydro.total_inj_fatality.count()]
print(y)
plt.title("Inj with hydro vs Without hydro")
obj = ('Inj without hydro','Injuries with hydro', 'No Inj with hydro')
plt.bar(x,y, tick_label=obj)
plt.show()
```

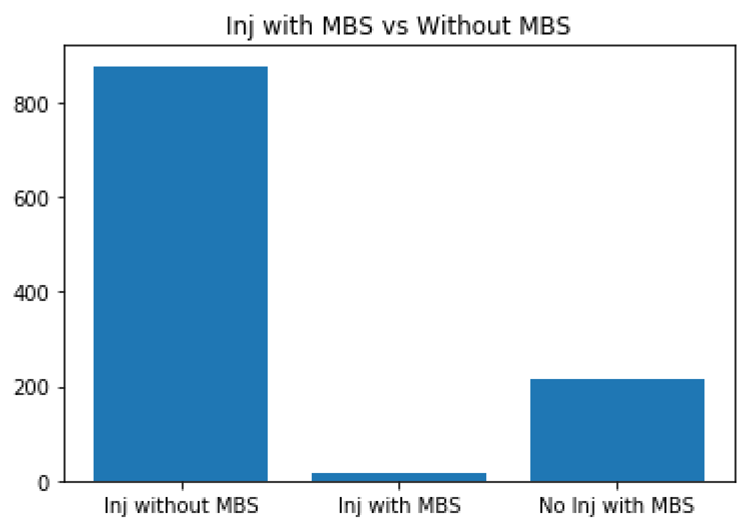
```
hydro value count:      715966
1      4404
Name: hydro, dtype: int64
hydro Injuries is null: 0
[664, 227, 4177]
```



Plotting municipal Contacted vs injuries / no injuries

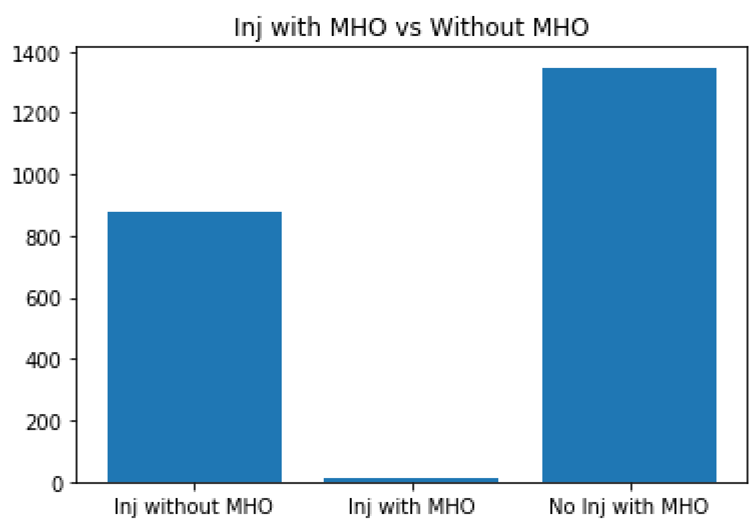
In [116]:
```
print("municipal_building_office value count:" , df['municipal_building_office'].value_counts())
print("municipal_building_office Injuries is null:", df['municipal_building_office'].isnull().sum())

inj_nmunicipal_building_office = df[(df.total_inj_fatality == 1) & (df.municipal_building_office != '1') ]
inj_municipal_building_office =  df[(df.total_inj_fatality == 1) & (df.municipal_building_office == '1') ]
#ninj_nopp = df[(df.total_inj_fatality != 1) & (df.opp != '1') ]
ninj_municipal_building_office = df[(df.total_inj_fatality != 1) & (df.municipal_building_office == '1') ]

x = [1,2,3]
y = [inj_nmunicipal_building_office.total_inj_fatality.sum(), inj_municipal_building_office.total_inj_fatality.sum(),
     ninj_municipal_building_office.total_inj_fatality.count()]
print(y)
plt.title("Inj with MBS vs Without MBS")
obj = ('Inj without MBS','Inj with MBS',
       'No Inj with MBS')
plt.bar(x,y, tick_label=obj)
plt.show()
```

```
municipal_building_office value count:      720142
1       228
Name: municipal_building_office, dtype: int64
municipal_building_office Injuries is null: 0
[876, 15, 213]
```



In [117]:
```
print("municipal_health_office value count:" , df['municipal_health_office'].v
alue_counts())
print("mmunicipal_health_office Injuries is null:", df['municipal_health_offic
e'].isnull().sum())

inj_nmunicipal_health_office = df[(df.total_inj_fatality == 1) & (df.municipal
_health_office != '1') ]
inj_municipal_health_office =  df[(df.total_inj_fatality == 1) & (df.municipal
_health_office == '1') ]
#ninj_nopp = df[(df.total_inj_fatality != 1) & (df.opp != '1') ]
ninj_municipal_health_office = df[(df.total_inj_fatality != 1) & (df.municipal
_health_office == '1') ]

x = [1,2,3]
y = [inj_nmunicipal_health_office.total_inj_fatality.sum(), inj_municipal_heal
th_office.total_inj_fatality.sum(),
     ninj_municipal_health_office.total_inj_fatality.count()]
print(y)
plt.title("Inj with MHO vs Without MHO")
obj = ('Inj without MHO','Inj with MHO',
       'No Inj with MHO')
plt.bar(x,y, tick_label=obj)
plt.show()
```

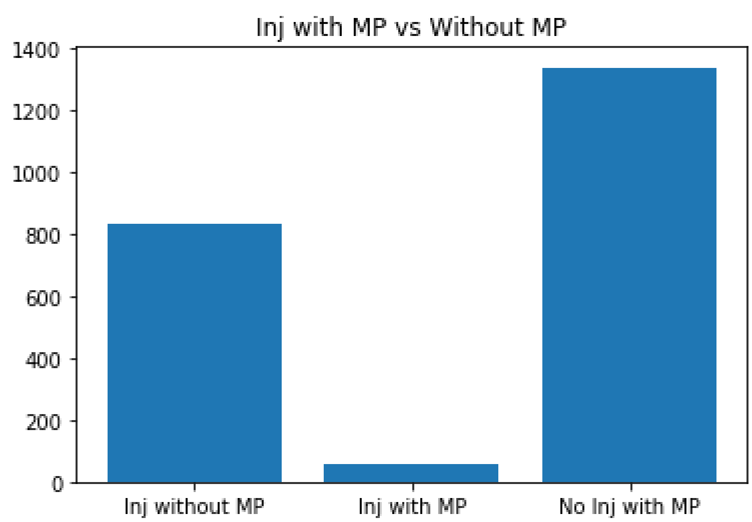
```
municipal_health_office value count:      719008
1      1362
Name: municipal_health_office, dtype: int64
mmunicipal_health_office Injuries is null: 0
[877, 14, 1348]
```



In [118]:
```
print("municipal_police value count:" , df['municipal_police'].value_counts())
print("municipal_police Injuries is null:", df['municipal_police'].isnull().sum())

inj_nmunicipal_police = df[(df.total_inj_fatality == 1) & (df.municipal_police != '1') ]
inj_municipal_police =  df[(df.total_inj_fatality == 1) & (df.municipal_police == '1') ]
#ninj_nopp = df[(df.total_inj_fatality != 1) & (df.opp != '1') ]
ninj_municipal_police = df[(df.total_inj_fatality != 1) & (df.municipal_police == '1') ]

x = [1,2,3]
y = [inj_nmunicipal_police.total_inj_fatality.sum(), inj_municipal_police.total_inj_fatality.sum(),
     ninj_municipal_police.total_inj_fatality.count()]
print(y)
plt.title("Inj with MP vs Without MP")
obj = ('Inj without MP','Inj with MP',
       'No Inj with MP')
plt.bar(x,y, tick_label=obj)
```

```
municipal_police value count:      718973
1      1397
Name: municipal_police, dtype: int64
municipal_police Injuries is null: 0
[831, 60, 1337]
```

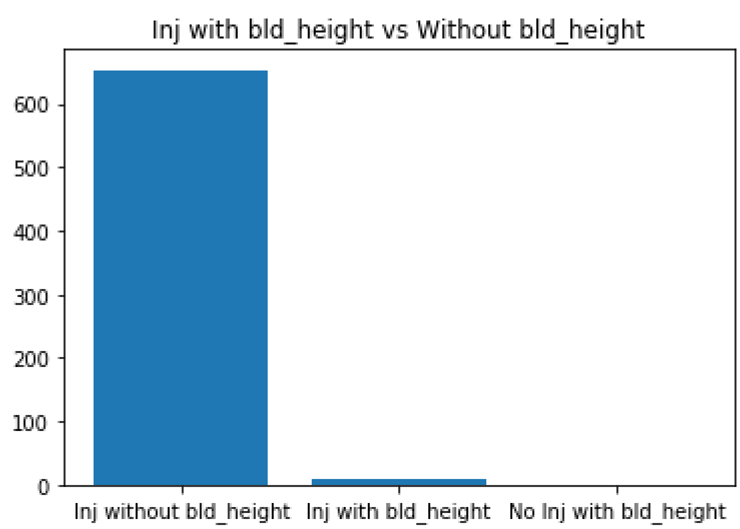
Out[118]: <BarContainer object of 3 artists>



Plotting Building height vs injuries / no injuries

In [119]:
```
group_labels = ['small', 'med', 'high', 'no_val']
bin_val = [1,33,66,101,10001]
df['bld_height_bin'] = pd.cut(df.bld_height, bins=bin_val, labels=group_labels
)


print("bld_height value count:" , df['bld_height_bin'].value_counts())
print("bld_height Injuries is null:", df['bld_height_bin'].isnull().sum())



inj_nbld_height = df[(df.total_inj_fatality == 1) & (df.bld_height_bin == 'sma
ll') ]
inj_bld_height =  df[(df.total_inj_fatality == 1) & (df.bld_height_bin == 'me
d') ]
#ninj_nopp = df[(df.total_inj_fatality != 1) & (df.opp != '1') ]
ninj_bld_height = df[(df.total_inj_fatality == 1) & (df.bld_height_bin == 'hig
h') ]

x = [1,2,3]
y = [inj_nbld_height.total_inj_fatality.sum(), inj_bld_height.total_inj_fatali
ty.sum(),
     ninj_bld_height.total_inj_fatality.count()]
print(y)
plt.title("Inj with bld_height vs Without bld_height")
obj = ('Inj without bld_height','Inj with bld_height',
       'No Inj with bld_height')
plt.bar(x,y, tick_label=obj)
plt.show()
```

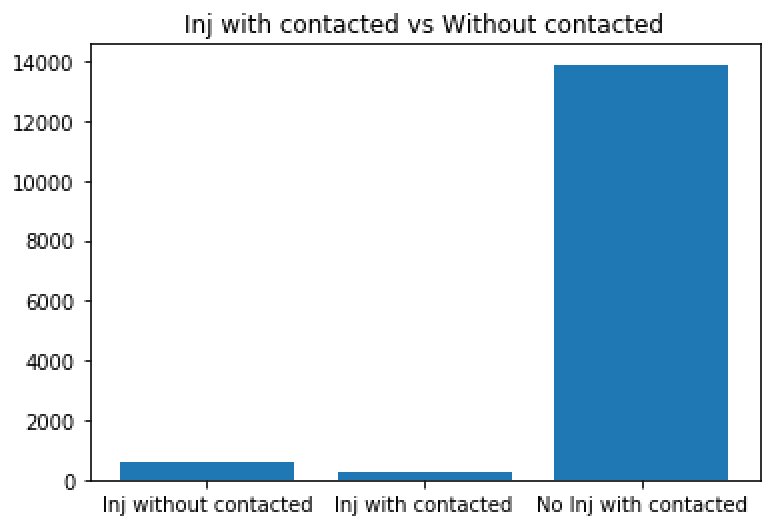
```
bld_height value count: small     6528
no_val     477
med        128
high        11
Name: bld_height_bin, dtype: int64
bld_height Injuries is null: 713226
[653, 9, 0]
```



In [120]:
```
print("contacted value count:" , df['contacted'].value_counts())
print("contacted Injuries is null:", df['contacted'].isnull().sum())

inj_ncontacted = df[(df.total_inj_fatality == 1) & (df.contacted != 1) ]
inj_contacted =  df[(df.total_inj_fatality == 1) & (df.contacted == 1) ]
#ninj_nopp = df[(df.total_inj_fatality != 1) & (df.opp != '1') ]
ninj_contacted = df[(df.total_inj_fatality != 1) & (df.contacted == 1) ]

x = [1,2,3]
y = [inj_ncontacted.total_inj_fatality.sum(), inj_contacted.total_inj_fatality
.sum(),
     ninj_contacted.total_inj_fatality.count()]
print(y)
plt.title("Inj with contacted vs Without contacted")
obj = ('Inj without contacted','Inj with contacted',
       'No Inj with contacted')
plt.bar(x,y, tick_label=obj)
```

```
contacted value count: 0    706204
1     14166
Name: contacted, dtype: int64
contacted Injuries is null: 0
[613, 278, 13888]
```

Out[120]: <BarContainer object of 3 artists>



Dave's - Exploratory analysis

Encoding with One-Hot-Encoder

In [51]:
```
df = pd.read_csv('./dataset/TFSDataSet.csv')
```

In [52]:
```
_df = df[['incident_number','incident_date_time',\
          'smoke_alarm_impact_on_num_evac','property','response_type','total_n
um_personnel']]
```

In [53]:
```
_df.property  = pd.to_numeric(_df.property, errors='coerce')
```

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\envs\mlboo
k\lib\site-packages\pandas\core\generic.py:3643: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  self[name] = value
```

In [54]:
```
_df.property.fillna(1 , inplace=True)
_df.response_type.fillna(1, inplace=True)
```

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\envs\mlboo
k\lib\site-packages\pandas\core\generic.py:4355: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  self._update_inplace(new_data)
```

In [55]:
```
from sklearn.preprocessing import MaxAbsScaler
mms_ = MaxAbsScaler()
_df['total_num_personnel_scaler'] = mms_.fit_transform(_df.total_num_personnel
.values.reshape(-1,1))
_df['smoke_alarm_impact_on_num_evac_scalar'] = mms_.fit_transform(_df.smoke_al
arm_impact_on_num_evac.values.reshape(-1,1))
_df['property_scaler'] = mms_.fit_transform(_df.property.values.reshape(-1,1))
```

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\envs\mlboo
k\lib\site-packages\ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  This is separate from the ipykernel package so we can avoid doing imports u
ntil
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\envs\mlboo
k\lib\site-packages\ipykernel_launcher.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  after removing the cwd from sys.path.
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\envs\mlboo
k\lib\site-packages\ipykernel_launcher.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  """
```

In [56]:
```
_df.head()
```

Out[56]:

	incident_number	incident_date_time	smoke_alarm_impact_on_num_evac	property	response_ty
0	F16000001	2016-01-01 00:01:14	0	896.0	
1	F16000002	2016-01-01 00:04:47	0	323.0	
2	F16000003	2016-01-01 00:05:05	0	896.0	
3	F16000004	2016-01-01 00:06:41	0	321.0	
4	F16000005	2016-01-01 00:06:50	0	302.0	

In [57]:
```
#_df.property.astype(object)
#_df.response_type.astype(object)
#_df.property.isna().sum()
```

In [58]:
```
from sklearn.preprocessing import OneHotEncoder

encoder_ = OneHotEncoder()

for column in _df.columns[4:5]:
    e_ = encoder_.fit_transform(_df[column].values.reshape(-1,1)).toarray()
    df_e = pd.DataFrame(e_, columns = [column+str(int(i)) for i in range(e_.sh
ape[1])])
    _df = pd.concat([_df, df_e], axis=1)
```

In [61]:
```
_df.to_csv('./dataset/DS_TFSDataSet.csv')
```

In [62]:
```
_df.shape
```

Out[62]:
```
(117426, 76)
```

Merging all individual's csv together

Following work done by: ADNAN LANEWLA

In [1]:
```
import pandas as pd
import numpy as np
import os
from IPython.display import display
pd.set_option('display.max_columns',200)
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
```

In [2]:
```
Pure_df = pd.read_csv('./dataset/TFSDataSetWithTotalFatality.csv')
```

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\envs\mlboo
k\lib\site-packages\IPython\core\interactiveshell.py:2728: DtypeWarning: Colu
mns (21,22,44) have mixed types. Specify dtype option on import or set low_me
mory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

In [3]:
```
df = Pure_df.copy()
```

Combining all individual's .csv's

In [4]:
```
adnan_df = pd.read_csv('./dataset/TFSDataSetWithTotalFatality_Adnan.csv')
arjun_df = pd.read_csv('./dataset/TFSDataSetWithTotalFatality_arjun.csv')
ashok_df = pd.read_csv('./dataset/TFSDataSetWithTotalFatality_Ashok.csv')
dave_df = pd.read_csv('./dataset/TFSDataSetWithTotalFatality_Dave.csv')
```

In [5]:
```
adnan_df.tail()
```

Out[5]:

	Unnamed: 0	rescues_unscaled	rescues_scaled	min_to_reach_unscaled	min_to_reach_sca
720365	720365	0	0.0	5	0.0847
720366	720366	0	0.0	54	0.9152
720367	720367	0	0.0	54	0.9152
720368	720368	0	0.0	55	0.9322
720369	720369	0	0.0	56	0.9491

In [6]:
```
dave_df.tail()
```

Out[6]:

	Unnamed: 0	incident_number	incident_date_time	smoke_alarm_impact_on_num_evac	prop
720365	720365	F16122093	2016-12-31 23:52:49	0	3
720366	720366	F16122094	2016-12-31 23:55:44	0	3
720367	720367	F16122095	2016-12-31 23:56:01	0	3
720368	720368	F16122096	2016-12-31 23:59:18	0	3
720369	720369	F16122097	2016-12-31 23:59:52	0	8

In [7]:
```
adnan_df['incident_number'].fillna(value='no_incident_number', inplace=True)
arjun_df['incident_number'].fillna(value='no_incident_number', inplace=True)
ashok_df['incident_number'].fillna(value='no_incident_number', inplace=True)
dave_df['incident_number'].fillna(value='no_incident_number', inplace=True)
```

In [8]:
```
dave_df.shape[0]
```

Out[8]: 720370

In [9]:
```
adnan_df.shape
```

Out[9]: (720370, 6)

In [24]:
```
#to check if all the rows for all the .csv's match
for index, row in adnan_df.iterrows():
    adnan_incident = adnan_df.at[index,'incident_number']
    arjun_incident = arjun_df.at[index,'incident_number']
    ashok_incident = ashok_df.at[index,'incident_number']
    if(index >= dave_df.shape[0]):
        continue
    dave_incident = dave_df.at[index,'incident_number']
    if(adnan_incident != arjun_incident != ashok_incident != dave_incident):
        raise Exception('index: ' + str(index))
```

Merging all the scaled columns

In [13]:
```
final_scaled = dave_df.copy()
```

In [14]:
```
final_scaled.drop(['Unnamed: 0','smoke_alarm_impact_on_num_evac','property','r
esponse_type','total_num_personnel'], inplace=True, axis=1)
```

In [15]:
```
final_scaled.head(1)
```

Out[15]:

	incident_number	incident_date_time	total_num_personnel_scaler	smoke_alarm_impact_on_num
0	F11000010	2011-01-01 00:03:43	0.003132	

In [16]:
```
adnan_df.head(1)
```

Out[16]:

	Unnamed: 0	rescues_unscaled	rescues_scaled	min_to_reach_unscaled	min_to_reach_scaled	i
0	0	0	0.0	7	0.118644	

In [17]:
```
final_scaled = final_scaled.join(adnan_df[['rescues_scaled', 'min_to_reach_sca
led']])
```

In [18]:
```
ashok_df.head(1)
```

Out[18]:

	Unnamed: 0	Unnamed: 0.1	event_alarm_level	responding_units	ofm_investigations_contacted	aid_
0	0	0	0.006157	1.0	0	

In [19]:
```
final_scaled = final_scaled.join(ashok_df[['ofm_investigations_contacted',
                                           'responding_units_scaled',
                                           'aid_to_from_other_depts_scaled',
                                           'event_alarm_level_scaled_0',
                                           'event_alarm_level_scaled_1',
                                           'event_alarm_level_scaled_2',
                                           'event_alarm_level_scaled_3',
                                           'event_alarm_level_scaled_4',
                                           'event_alarm_level_scaled_5']])
```

In [20]:
```
final_scaled.head(1)
```

Out[20]:

	incident_number	incident_date_time	total_num_personnel_scaler	smoke_alarm_impact_on_num
0	F11000010	2011-01-01 00:03:43	0.003132	

In [21]:
```
arjun_df.head(1)
```

Out[21]:

	Unnamed: 0	incident_number	contacted	total_inj_fatality	bld_heigh_small	bld_heigh_med	bld
0	0	F11000010	0	0	0	0	

In [22]:
```
final_scaled = final_scaled.join(arjun_df[['contacted', 'bld_heigh_small','bld
_heigh_med','bld_heigh_high','bld_heigh_no_val', 'total_inj_fatality']])
```

In [23]:
```
final_scaled.head(1)
```

Out[23]:

	incident_number	incident_date_time	total_num_personnel_scaler	smoke_alarm_impact_on_num
0	F11000010	2011-01-01 00:03:43	0.003132	

In [24]:
```
final_scaled.shape
```

Out[24]:
```
(720370, 90)
```

In [45]:
```
final_scaled.to_csv('./dataset/TFS_Final_Scaled.csv')
```

Merging all the unscaled columns

In [25]:
```
final_unscaled = adnan_df.copy()
```

In [26]:
```
final_unscaled.head(1)
```

Out[26]:

	Unnamed: 0	rescues_unscaled	rescues_scaled	min_to_reach_unscaled	min_to_reach_scaled	i
0	0	0	0.0	7	0.118644	

In [27]:
```
final_unscaled.drop(['Unnamed: 0','rescues_scaled','min_to_reach_scaled'], axi
s=1,inplace=True)
```

In [28]:
```
final_unscaled.head(1)
```

Out[28]:

	rescues_unscaled	min_to_reach_unscaled	incident_number
0	0	7	F11000010

In [29]:
```
final_unscaled = final_unscaled.join(dave_df[['incident_date_time']])
```

In [30]:
```
final_unscaled.head(1)
```

Out[30]:

	rescues_unscaled	min_to_reach_unscaled	incident_number	incident_date_time
0	0	7	F11000010	2011-01-01 00:03:43

In [31]:
```
final_unscaled = final_unscaled[['incident_number','incident_date_time','rescu
es_unscaled','min_to_reach_unscaled']]
```

In [32]:
```
final_unscaled.head(1)
```

Out[32]:

	incident_number	incident_date_time	rescues_unscaled	min_to_reach_unscaled
0	F11000010	2011-01-01 00:03:43	0	7

In [33]:
```
dave_df.head(1)
```

Out[33]:

	Unnamed: 0	incident_number	incident_date_time	smoke_alarm_impact_on_num_evac	property
0	0	F11000010	2011-01-01 00:03:43	0	301.0

In [34]:
```
final_unscaled = final_unscaled.join(dave_df[['smoke_alarm_impact_on_num_evac'
, 'property','response_type','total_num_personnel']])
```

In [35]:
```
final_unscaled.head(1)
```

Out[35]:

	incident_number	incident_date_time	rescues_unscaled	min_to_reach_unscaled	smoke_alarm_
0	F11000010	2011-01-01 00:03:43	0	7	

In [36]:
```
ashok_df.head(1)
```

Out[36]:

	Unnamed: 0	Unnamed: 0.1	event_alarm_level	responding_units	ofm_investigations_contacted	aid_
0	0	0	0.006157	1.0	0	

In [37]:
```
final_unscaled = final_unscaled.join(ashok_df[['event_alarm_level','responding
_units','ofm_investigations_contacted','aid_to_from_other_depts']])
```

In [38]:
```
final_unscaled.head(1)
```

Out[38]:

	incident_number	incident_date_time	rescues_unscaled	min_to_reach_unscaled	smoke_alarm_
0	F11000010	2011-01-01 00:03:43	0	7	

In [39]:
```
arjun_df.head(1)
```

Out[39]:

	Unnamed: 0	incident_number	contacted	total_inj_fatality	bld_heigh_small	bld_heigh_med	bld
0	0	F11000010	0	0	0	0	

In [40]:
```
final_unscaled = final_unscaled.join(arjun_df[['contacted','bld_heigh_small',
'bld_heigh_med','bld_heigh_high','bld_heigh_no_val','total_inj_fatality']])
```

In [41]:
```
final_unscaled.head(1)
```

Out[41]:

	incident_number	incident_date_time	rescues_unscaled	min_to_reach_unscaled	smoke_alarm_
0	F11000010	2011-01-01 00:03:43	0	7	

In [82]:
```
final_unscaled.shape
```

Out[82]:
```
(720370, 18)
```

In [83]:
```
final_unscaled.to_csv('./dataset/TFS_Final_Unscaled.csv')
```

Model Evaluation

Following work done by: ADNAN LANEWLA

Random Forest

In [128]:
```
import pandas as pd
import numpy as np
import os
from IPython.display import display
pd.set_option('display.max_columns',200)
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn import ensemble
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import cross_val_score
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import train_test_split
import urllib.request
import seaborn as sns
```

Training Random Forest With Unscaled Values

Reading csv file and splitting into train test

In [40]:
```
df = pd.read_csv('./dataset/TFS_Final_Unscaled.csv')
```

In [41]:
```
df.head(1)
```

Out[41]:

	Unnamed: 0	incident_number	incident_date_time	rescues_unscaled	min_to_reach_unscaled	sn
0	0	F11000010	2011-01-01 00:03:43	0	7	

In [42]:
```
X = df.drop(['Unnamed: 0','incident_number','incident_date_time','total_inj_fa
tality'],axis=1)
```

In [43]:
```
Y = df[['total_inj_fatality']]
```

In [75]:
```
X = X.values
Y = Y.values
```

In [73]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, ran
dom_state=123, stratify=Y)
```

In [74]:
```
X_train = X_train.values
X_test = X_test.values
y_train = y_train.values
y_train = y_train.ravel()
y_test = y_test.values
y_test = y_test.ravel()
```

In [51]:
```
X_train.shape
```

Out[51]:
```
(576296, 15)
```

Random Forest Unscaled Values using Randomized Grid Search

In [67]:
```
def plot_roc_curve(fpr, tpr, auc, label=None):
    plt.plot(fpr, tpr, linewidth=2, label='ROC curve (area = %0.2f)' % auc)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)
```

In [46]:
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
```

In [47]:
```
rnd_clf = RandomForestClassifier(n_estimators=100)
```

In [52]:
```
param_dist = {"max_depth": [3, None],
              "max_features": sp_randint(2, 15),
              "min_samples_split": sp_randint(4, 25),
              "bootstrap": [True, False],
              "criterion": ["gini", "entropy"],
              "min_samples_leaf": sp_randint(1, 20),
              "max_leaf_nodes": [None,2,20]}
```

In [53]:
```
n_iter_search = 50
random_search = RandomizedSearchCV(rnd_clf, param_distributions=param_dist,
                                   n_iter=n_iter_search, cv=4, random_state=12
3, scoring='roc_auc')
```

In [54]:
```
random_search.fit(X_train, y_train)
```

Out[54]:
```
RandomizedSearchCV(cv=4, error_score='raise',
          estimator=RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
          fit_params=None, iid=True, n_iter=50, n_jobs=1,
          param_distributions={'min_samples_leaf': <scipy.stats._distn_infras
tructure.rv_frozen object at 0x0000026E52EBA0B8>, 'bootstrap': [True, False],
'criterion': ['gini', 'entropy'], 'max_leaf_nodes': [None, 2, 20], 'max_featu
res': <scipy.stats._distn_infrastructure.rv_frozen object at 0x0000026E67E70E
48>, 'max_depth': [3, None], 'min_samples_split': <scipy.stats._distn_infrast
ructure.rv_frozen object at 0x0000026E67E70438>},
          pre_dispatch='2*n_jobs', random_state=123, refit=True,
          return_train_score='warn', scoring='roc_auc', verbose=0)
```

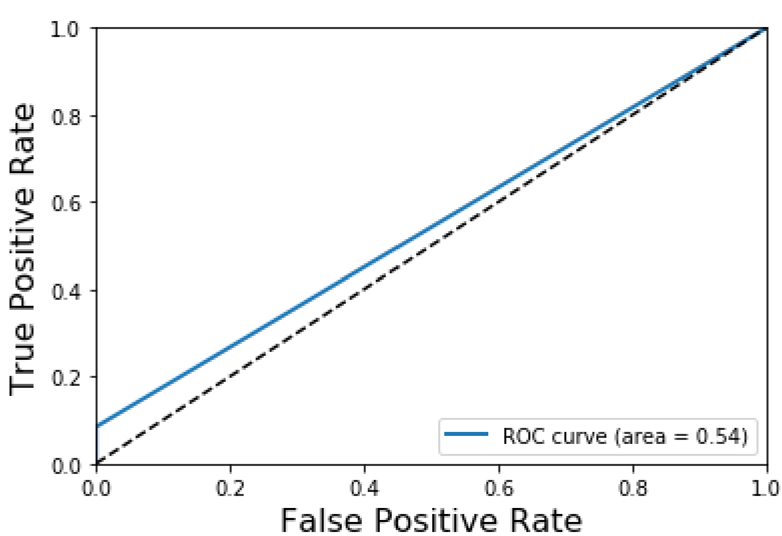
In [55]:
```
print(random_search.best_score_)
print(random_search.best_params_)
```

```
0.984121566822
{'min_samples_leaf': 8, 'bootstrap': True, 'criterion': 'entropy', 'min_sampl
es_split': 10, 'max_leaf_nodes': None, 'max_features': 3, 'max_depth': None}
```

In [57]:
```
y_pred = random_search.predict(X_test)
```

In [68]:
```
fpr, tpr, threshold = roc_curve(y_test, y_pred)
roc_auc = auc(fpr,tpr)
plt.Figure(figsize=(8,8))
plot_roc_curve(fpr, tpr, roc_auc, 'Random Forest')
plt.legend(loc='lower right')
plt.show
```

Out[68]:
```
<function matplotlib.pyplot.show>
```



- As we can see from the above observation that our AUC is around 0.54 which is almost random.
- This is expected since the data is skewed and there are no features which could be a good predictor of injuries / fatalities

Random Forest with Scaled Values

In [76]:
```
clf = RandomForestClassifier(random_state=123)
```

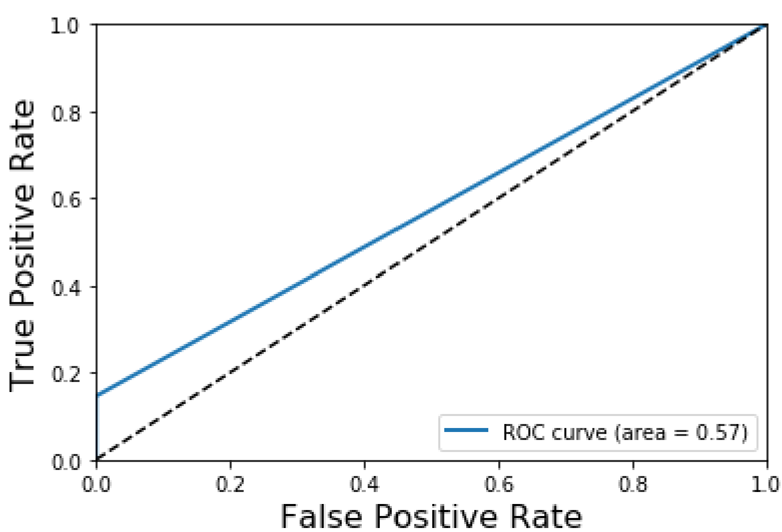
In [77]:
```
clf.fit(X_train,y_train)
```

Out[77]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=123, verbose=0, warm_start=False)
```

In [80]:
```
y_predict = clf.predict(X_test)
```

In [81]:
```
fpr, tpr, threshold = roc_curve(y_test, y_predict)
roc_auc = auc(fpr,tpr)
plt.Figure(figsize=(8,8))
plot_roc_curve(fpr, tpr, roc_auc, 'Random Forest')
plt.legend(loc='lower right')
plt.show
```

Out[81]:
```
<function matplotlib.pyplot.show>
```



- Random forest gives us the auc score slighly better than randomized grid search

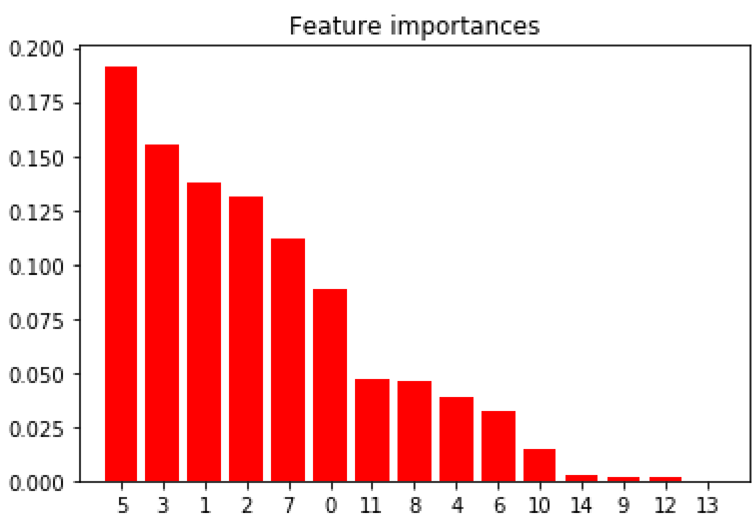
In [83]:
```
importances = clf.feature_importances_
```

In [84]:
```
std = np.std([clf.feature_importances_ for tree in clf.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]
```

In [85]:
```
for f in range(X_train.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f
]]))
```

```
1. feature 5 (0.191596)
2. feature 3 (0.155064)
3. feature 1 (0.137445)
4. feature 2 (0.130821)
5. feature 7 (0.112055)
6. feature 0 (0.088941)
7. feature 11 (0.046861)
8. feature 8 (0.045682)
9. feature 4 (0.038482)
10. feature 6 (0.032086)
11. feature 10 (0.014663)
12. feature 14 (0.002752)
13. feature 9 (0.001986)
14. feature 12 (0.001545)
15. feature 13 (0.000023)
```

In [87]:
```
plt.figure()
plt.title("Feature importances")
plt.bar(range(X_train.shape[1]), importances[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(X_train.shape[1]), indices)
plt.xlim([-1, X_train.shape[1]])
plt.show()
```



Training Random Forest with Scaled Values

Reading scaled csv and splitting data into train test set

In [89]:
```
df = pd.read_csv('./dataset/TFS_Final_Scaled.csv')
```

In [90]:
```
df.head(1)
```

Out[90]:

	Unnamed: 0	incident_number	incident_date_time	total_num_personnel_scaler	smoke_alarm_imp
0	0	F11000010	2011-01-01 00:03:43	0.003132	

In [91]:
```
X = df.drop(['Unnamed: 0','incident_number','incident_date_time','total_inj_fa
tality'],axis=1)
```

In [92]:
```
Y = df[['total_inj_fatality']]
```

In [93]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, ran
dom_state=123, stratify=Y)
```

In [94]:
```
X_train = X_train.values
X_test = X_test.values
y_train = y_train.values
y_train = y_train.ravel()
y_test = y_test.values
y_test = y_test.ravel()
```

In [95]:
```
X_train.shape
```

Out[95]: (576296, 87)

Random Forest Scaled Values using Randomized Grid Search

In [109]:
```
def plot_roc_curve(fpr, tpr, auc, label=None):
    plt.plot(fpr, tpr, linewidth=2, label='ROC curve (area = %0.2f)' % auc)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)
```

In [110]:
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
```

In [112]:
```
rnd_clf = RandomForestClassifier()
```

In [113]:
```
param_dist = {"max_depth": [3, None],
              "n_estimators" : [3, 4, 6, 7, 10, 20, 50, 100,200],
              "max_features": sp_randint(2, 15),
              "min_samples_split": sp_randint(4, 25),
              "bootstrap": [True, False],
              "criterion": ["gini", "entropy"],
              "min_samples_leaf": sp_randint(1, 20),
              "max_leaf_nodes": [None,2,20]}
```

In [114]:
```
n_iter_search = 50
random_search = RandomizedSearchCV(rnd_clf, param_distributions=param_dist,
                                   n_iter=n_iter_search, cv=4, random_state=12
3, scoring='roc_auc')
```

In [115]:
```
random_search.fit(X_train, y_train)
```

Out[115]:
```
RandomizedSearchCV(cv=4, error_score='raise',
          estimator=RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
          fit_params=None, iid=True, n_iter=50, n_jobs=1,
          param_distributions={'min_samples_leaf': <scipy.stats._distn_infras
tructure.rv_frozen object at 0x0000026E008BE9B0>, 'bootstrap': [True, False],
'criterion': ['gini', 'entropy'], 'min_samples_split': <scipy.stats._distn_in
frastructure.rv_frozen object at 0x0000026E008B44E0>, 'max_leaf_nodes': [Non
e, 2, 20], 'max_features': <scipy.stats._distn_infrastructure.rv_frozen objec
t at 0x0000026E00571B38>, 'n_estimators': [3, 4, 6, 7, 10, 20, 50, 100, 200],
'max_depth': [3, None]},
          pre_dispatch='2*n_jobs', random_state=123, refit=True,
          return_train_score='warn', scoring='roc_auc', verbose=0)
```

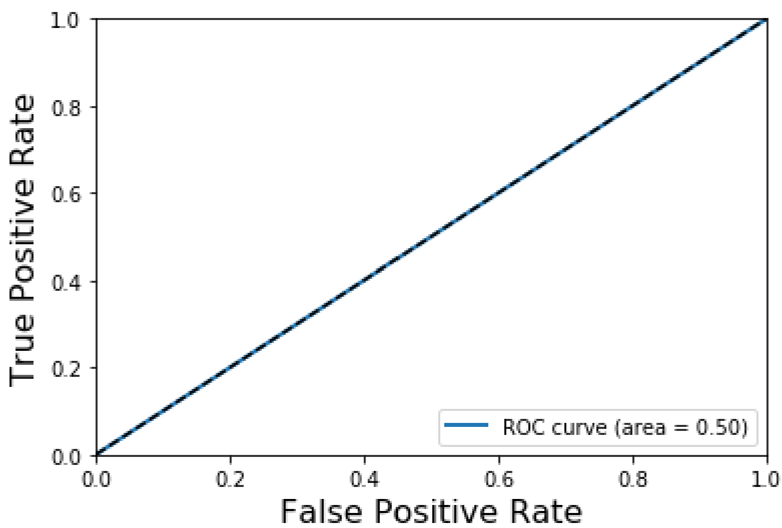
In [116]:
```
print(random_search.best_score_)
print(random_search.best_params_)
```

```
0.982221681967
{'min_samples_leaf': 14, 'bootstrap': False, 'criterion': 'entropy', 'min_sam
ples_split': 14, 'max_leaf_nodes': 20, 'max_features': 12, 'n_estimators': 20
0, 'max_depth': None}
```

In [117]:
```
y_pred = random_search.predict(X_test)
```

In [118]:
```
fpr, tpr, threshold = roc_curve(y_test, y_pred)
roc_auc = auc(fpr,tpr)
plt.Figure(figsize=(8,8))
plot_roc_curve(fpr, tpr, roc_auc, 'Random Forest')
plt.legend(loc='lower right')
plt.show
```

Out[118]: <function matplotlib.pyplot.show>



- With scaled valies our classifier performs poorly.
- auc value is 0.5 is random so our classifer is unable to predict injuries / fatalities

Random Forest Classifier

In [119]:
```
clf = RandomForestClassifier(random_state=123)
```

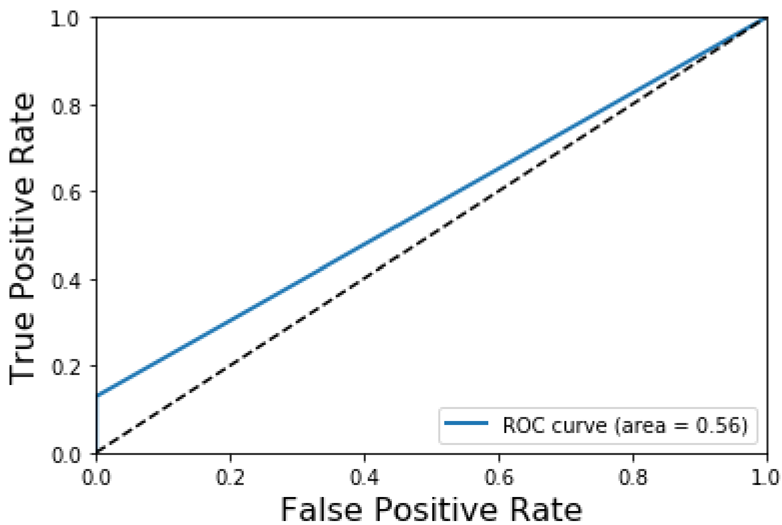
In [120]:
```
clf.fit(X_train,y_train)
```

Out[120]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=123, verbose=0, warm_start=False)
```

In [121]:
```
y_predict = clf.predict(X_test)
```

In [122]:
```
fpr, tpr, threshold = roc_curve(y_test, y_predict)
roc_auc = auc(fpr,tpr)
plt.Figure(figsize=(8,8))
plot_roc_curve(fpr, tpr, roc_auc, 'Random Forest')
plt.legend(loc='lower right')
plt.show
```

Out[122]: <function matplotlib.pyplot.show>



Feature Importances

In [123]:
```
importances = clf.feature_importances_
```

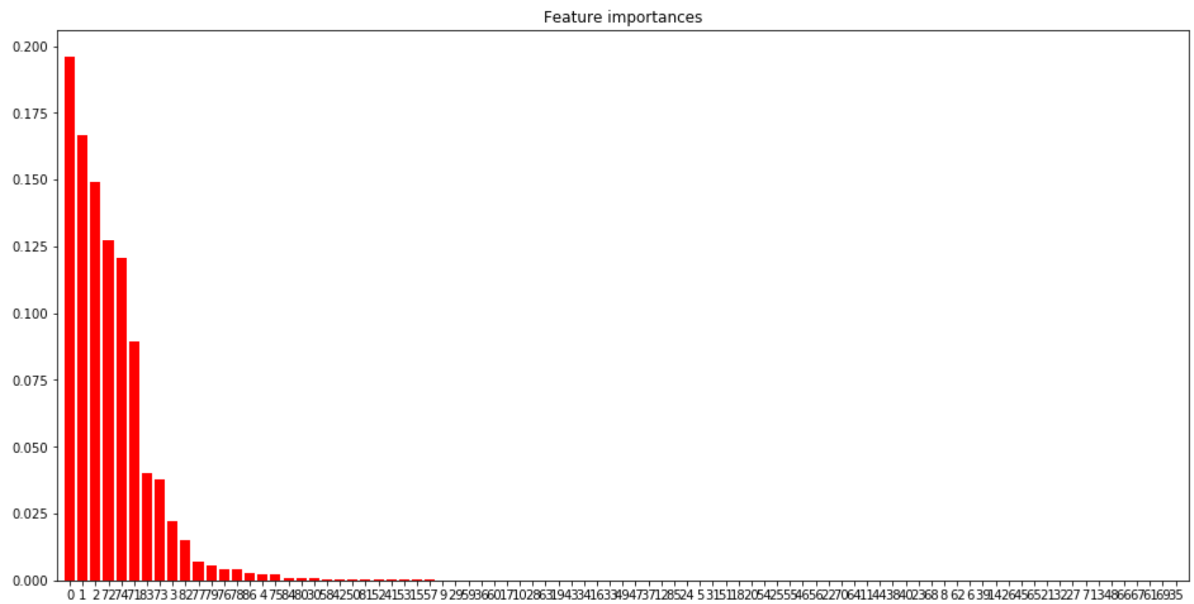
In [124]:
```
std = np.std([clf.feature_importances_ for tree in clf.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]
```

In [125]:
```
for f in range(X_train.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f
]]))
```

```
1. feature 0 (0.196196)
2. feature 1 (0.166861)
3. feature 2 (0.149041)
4. feature 72 (0.127231)
5. feature 74 (0.120678)
6. feature 71 (0.089578)
7. feature 83 (0.040353)
8. feature 73 (0.037833)
9. feature 3 (0.022289)
10. feature 82 (0.014985)
11. feature 77 (0.007221)
12. feature 79 (0.005609)
13. feature 76 (0.004243)
14. feature 78 (0.004126)
15. feature 86 (0.002884)
16. feature 4 (0.002370)
17. feature 75 (0.002270)
18. feature 84 (0.000830)
19. feature 80 (0.000817)
20. feature 30 (0.000804)
21. feature 58 (0.000405)
22. feature 42 (0.000324)
23. feature 50 (0.000316)
24. feature 81 (0.000286)
25. feature 52 (0.000258)
26. feature 41 (0.000255)
27. feature 53 (0.000231)
28. feature 15 (0.000156)
29. feature 57 (0.000148)
30. feature 9 (0.000135)
31. feature 29 (0.000129)
32. feature 59 (0.000120)
33. feature 36 (0.000112)
34. feature 60 (0.000103)
35. feature 17 (0.000091)
36. feature 10 (0.000088)
37. feature 28 (0.000084)
38. feature 63 (0.000079)
39. feature 19 (0.000073)
40. feature 43 (0.000059)
41. feature 34 (0.000056)
42. feature 16 (0.000028)
43. feature 33 (0.000028)
44. feature 49 (0.000026)
45. feature 47 (0.000026)
46. feature 37 (0.000025)
47. feature 12 (0.000023)
48. feature 85 (0.000023)
49. feature 24 (0.000016)
50. feature 5 (0.000013)
51. feature 31 (0.000012)
52. feature 51 (0.000011)
53. feature 18 (0.000007)
54. feature 20 (0.000004)
55. feature 54 (0.000004)
56. feature 25 (0.000004)
57. feature 55 (0.000003)
```

```
58. feature 46 (0.000003)
59. feature 56 (0.000003)
60. feature 22 (0.000003)
61. feature 70 (0.000002)
62. feature 64 (0.000002)
63. feature 11 (0.000001)
64. feature 44 (0.000001)
65. feature 38 (0.000001)
66. feature 40 (0.000000)
67. feature 23 (0.000000)
68. feature 68 (0.000000)
69. feature 8 (0.000000)
70. feature 62 (0.000000)
71. feature 6 (0.000000)
72. feature 39 (0.000000)
73. feature 14 (0.000000)
74. feature 26 (0.000000)
75. feature 45 (0.000000)
76. feature 65 (0.000000)
77. feature 21 (0.000000)
78. feature 32 (0.000000)
79. feature 27 (0.000000)
80. feature 7 (0.000000)
81. feature 13 (0.000000)
82. feature 48 (0.000000)
83. feature 66 (0.000000)
84. feature 67 (0.000000)
85. feature 61 (0.000000)
86. feature 69 (0.000000)
87. feature 35 (0.000000)
```

In [126]:
```
plt.figure(figsize=(16,8))
plt.title("Feature importances")
plt.bar(range(X_train.shape[1]), importances[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(X_train.shape[1]), indices)
plt.xlim([-1, X_train.shape[1]])
plt.show()
```



Cross Validation with Best Randomized Classifier

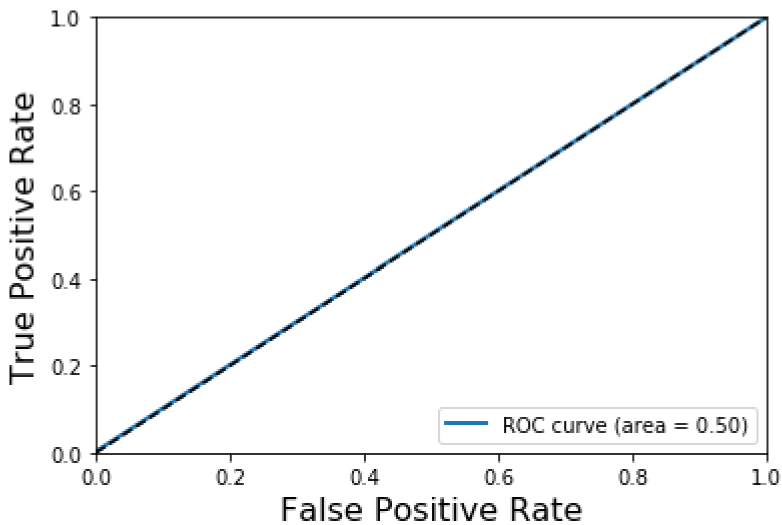
In [130]:
```
y_predict = cross_val_predict(random_search.best_estimator_, X_train, y_train,
 cv=5, method='predict')
```

In [132]:
```
y_predict.shape
```

Out[132]:
```
(576296,)
```

In [134]:
```
fpr, tpr, threshold = roc_curve(y_train, y_predict)
roc_auc = auc(fpr,tpr)
plt.Figure(figsize=(8,8))
plot_roc_curve(fpr, tpr, roc_auc, 'Random Forest')
plt.legend(loc='lower right')
plt.show
```

Out[134]: <function matplotlib.pyplot.show>



Neural Net with Scaled Data

We decided to train the Neural Net with our scaled data.

- The result we got is very consistent to the other tradional models
- We used 1 hidden layers with 8 units
- The input layer has the same number of neurons as the number of features

In [139]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.python.keras.models import Sequential
```

In [151]:
```
from tensorflow.python.keras.layers import Dense
from tensorflow.python.keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import StratifiedKFold
```

In [223]:
```
model = Sequential()
model.add(Dense(88,input_shape=(87,), kernel_initializer='normal',activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, kernel_initializer='normal', activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_train,y_train, epochs=5, batch_size=10)
```

```
Epoch 1/5
576296/576296 [==============================] - 48s - loss: 0.0051 - acc: 0.9987
Epoch 2/5
576296/576296 [==============================] - 47s - loss: 0.0047 - acc: 0.9988
Epoch 3/5
576296/576296 [==============================] - 47s - loss: 0.0047 - acc: 0.9988
Epoch 4/5
576296/576296 [==============================] - 47s - loss: 0.0047 - acc: 0.9988
Epoch 5/5
576296/576296 [==============================] - 47s - loss: 0.0046 - acc: 0.9988
```

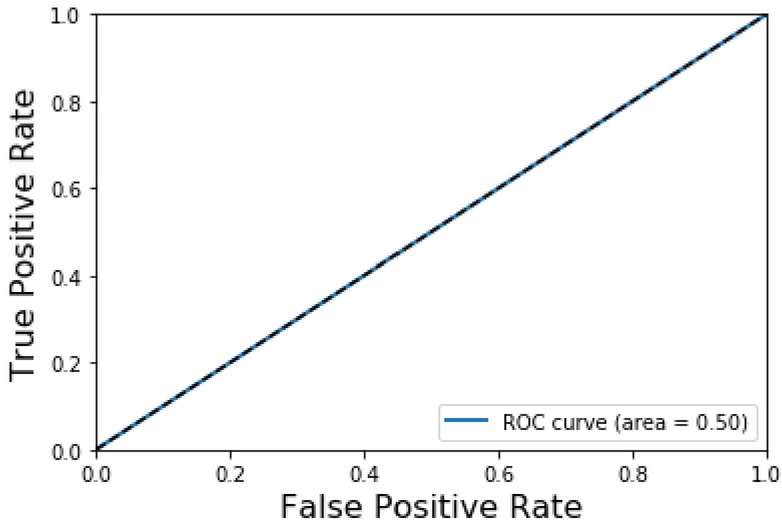
Out[223]:
```
<tensorflow.python.keras._impl.keras.callbacks.History at 0x26e1005cef0>
```

In [224]:
```
probabilities = model.predict(X_test)
```

In [225]:
```
y_pred = (probabilities >0.5)
y_test = (y_test > 0.5)
```

In [226]:
```
fpr, tpr, threshold = roc_curve(y_test, y_pred)
roc_auc = auc(fpr,tpr)
plt.Figure(figsize=(8,8))
plot_roc_curve(fpr, tpr, roc_auc, 'Neural Net')
plt.legend(loc='lower right')
plt.show
```

Out[226]:
```
<function matplotlib.pyplot.show>
```



- AUC score is same as random forest.

Confusion Matrix

In [194]:
```
from sklearn.metrics import confusion_matrix
```

In [209]:
```
confusion_matrix(y_test, y_pred, labels=[True, False])
```

Out[209]:
```
array([[      0,    178],
       [      0, 143896]], dtype=int64)
```

In [222]:
```
np.where(y_test == True)
```

Out[222]:
```
(array([    36,    380,   2666,   2906,   3312,   4393,   4435,   4631,
          4732,   5475,   6613,   7186,   7440,   9027,  10641,  10825,
         11110,  13145,  13284,  15080,  15791,  16611,  16832,  17748,
         18848,  19378,  19631,  20057,  21571,  21618,  21683,  23052,
         23787,  24871,  25309,  25576,  26500,  29234,  30640,  31004,
         33758,  33820,  35162,  35374,  36341,  36753,  36866,  38515,
         41108,  42461,  43074,  45532,  46132,  46163,  47406,  47934,
         49398,  49506,  50558,  51122,  51655,  52006,  53813,  53839,
         54104,  54118,  54786,  55239,  56702,  57549,  58798,  59178,
         59745,  60256,  61305,  61605,  63454,  63763,  63846,  64414,
         65832,  67164,  67332,  67949,  68993,  69760,  69889,  70029,
         70319,  70584,  70739,  71398,  71443,  71551,  71599,  72357,
         72491,  73825,  73858,  74905,  75348,  76779,  77017,  77558,
         78780,  79396,  80138,  82231,  82321,  82339,  82448,  83172,
         84436,  84881,  86456,  88748,  88792,  89663,  90324,  92431,
         92588,  93348,  94368,  95655,  95800,  96160,  96355,  97071,
         97727,  98329,  99533, 100298, 101061, 103383, 104839, 105319,
        105746, 106626, 107093, 107139, 107808, 108149, 108209, 109359,
        111584, 112522, 114678, 116323, 116846, 117018, 117214, 117294,
        118173, 119302, 120090, 120526, 120643, 121700, 122901, 123119,
        125381, 127043, 128543, 129965, 130906, 135330, 135716, 135826,
        136132, 136589, 136922, 136992, 137297, 138242, 139757, 139939,
        140515, 140538], dtype=int64),)
```

In [213]:
```
np.where(Y_pred == True)
```

Out[213]:
```
(array([], dtype=int64),)
```

Anomaly Detection on Scaled Values

In [1]:
```
import pandas as pd
import numpy as np
import os
from IPython.display import display
pd.set_option('display.max_columns',200)
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn import ensemble
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import cross_val_score
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import train_test_split
import urllib.request
import seaborn as sns
```

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\envs\mlboo
k\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: This
module was deprecated in version 0.18 in favor of the model_selection module
into which all the refactored classes and functions are moved. Also note that
the interface of the new CV iterators are different from that of this module.
This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:
```
from sklearn import svm
```

In [25]:
```
df = pd.read_csv('./dataset/TFS_Final_Scaled.csv')
```

In [26]:
```
df.head(1)
```

Out[26]:

	Unnamed: 0	incident_number	incident_date_time	total_num_personnel_scaler	smoke_alarm_imp
0	0	F11000010	2011-01-01 00:03:43	0.003132	

Splitting the data in to train test outliers

Its important to split the data in to three sets. For the sake of explanation let's take an example of banana and orange. Let's say we have a skewed dataset where 95% examples are of banana and 5% orange.

- First set Train set aside which doesn't contain any outliters or any oranges. So when we train an oneClassSVM we tell the algorithm this is how the data looks like which only has banana.
- Then we set aside Test set which also doesn't contain any outliers. Now we predict on test set to make sure the algorithm doesn't predict any oranges
- Then we predict on Outliers set which has only oranges. And then we check how many bananas algorithm detected and that would be error

In [27]:
```
df = df.drop(['Unnamed: 0','incident_number','incident_date_time'],axis=1)
```

In [30]:
```
X = df[(df['total_inj_fatality'] == 0)]
```

In [31]:
```
# training takes long for the whole dataset use only a fraction of it
X = X.sample(frac=0.3)
```

In [32]:
```
X.shape
```

Out[32]:
```
(215844, 88)
```

In [33]:
```
X_outliers = df[(df['total_inj_fatality'] == 1)]
```

In [34]:
```
X.drop(labels=['total_inj_fatality'],axis=1,inplace=True)
X_outliers.drop(labels=['total_inj_fatality'],axis=1,inplace=True)
```

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\envs\mlboo
k\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
```

In [35]:
```
X.shape
```

Out[35]:
```
(215844, 87)
```

In [36]:
```
X_outliers.shape
```

Out[36]:
```
(891, 87)
```

In [37]:
```
X_train, X_test = train_test_split(X, test_size = 0.2, random_state=123)
```

In [38]:
```
print(X_train.shape)
print(X_test.shape)
```

```
(172675, 87)
(43169, 87)
```

In [59]:
```
len(X_train)
```

Out[59]: 172675

In [39]:
```
X_train = X_train.values
X_test = X_test.values
X_outliers = X_outliers.values
```

Training an Anomaly Detection Algorithm (OneClassSVM)

Calculating the error for train, test , outlier prediction

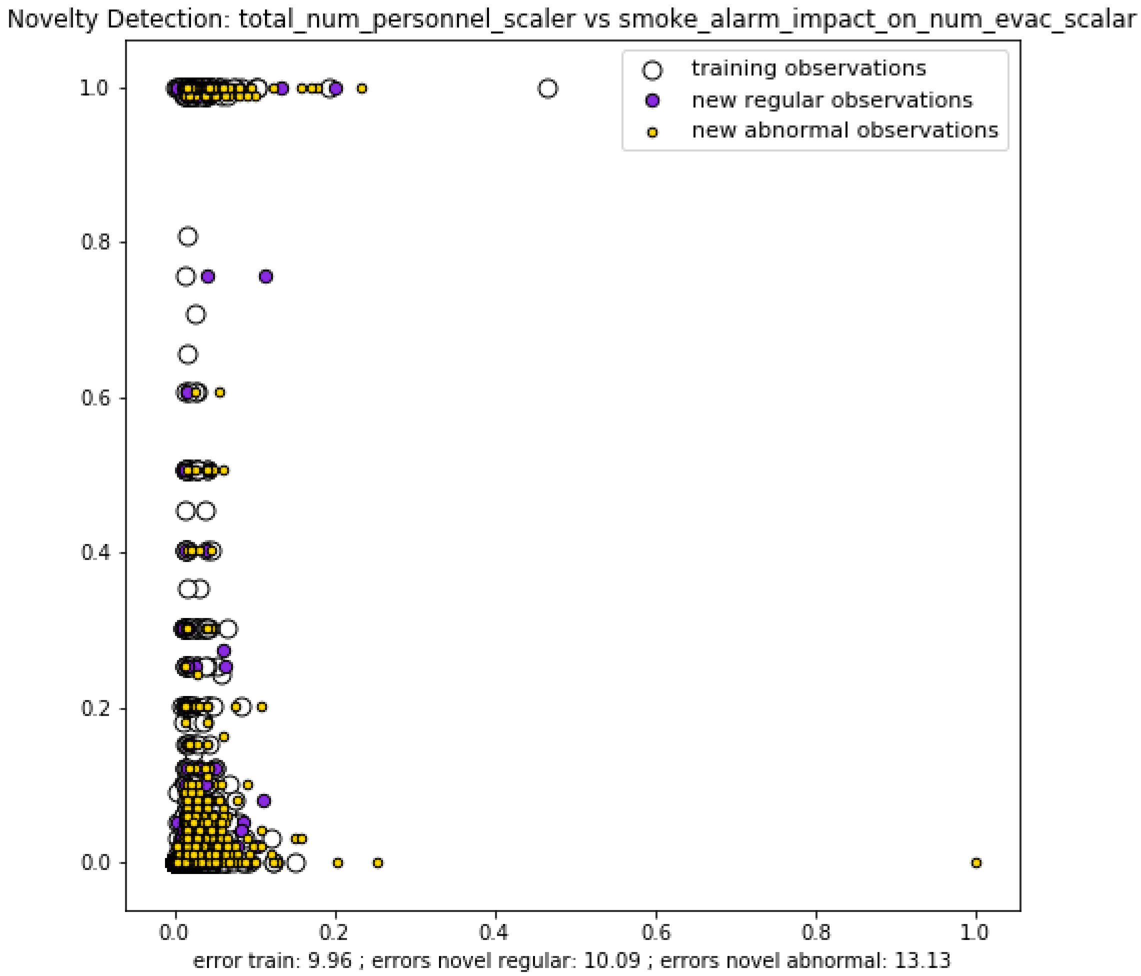
In [40]:
```
clf = svm.OneClassSVM(nu=0.1, kernel="rbf", gamma=0.1)
clf.fit(X_train)
y_pred_train = clf.predict(X_train)
y_pred_test = clf.predict(X_test)
y_pred_outliers = clf.predict(X_outliers)
n_error_train = y_pred_train[y_pred_train == -1].size
n_error_test = y_pred_test[y_pred_test == -1].size
n_error_outliers = y_pred_outliers[y_pred_outliers == 1].size
```

Plotting scatter plot

In [97]:
```
def scatter_plot(x, y):
    error_train = float((n_error_train/len(X_train)) * 100)
    error_test = float((n_error_test/len(X_test)) * 100)
    error_outliers = float((n_error_outliers/len(X_outliers)) * 100)

    plt.title("Novelty Detection: " + str(X.columns[x]) + ' vs ' + str(X.colum
ns[y]))
    s1 = 80
    s2 = 40
    s3=20
    b1 = plt.scatter(X_train[:, x], X_train[:, y], c='white', s=s1, edgecolors
='k')
    b2 = plt.scatter(X_test[:, x], X_test[:, y], c='blueviolet', s=s2,
                     edgecolors='k')
    c = plt.scatter(X_outliers[:, x], X_outliers[:, y], c='gold', s=s3,
                    edgecolors='k')
    plt.axis('tight')

    plt.legend([b1, b2, c],
               ["training observations",
                "new regular observations", "new abnormal observations"],
               loc="upper right",
               prop=matplotlib.font_manager.FontProperties(size=11))
    plt.xlabel(
        "error train: %0.2f ; errors novel regular: %0.2f ; "
        "errors novel abnormal: %0.2f"
        % (error_train,error_test, error_outliers))
    plt.rcParams["figure.figsize"] = [8,8]
    plt.show()
```

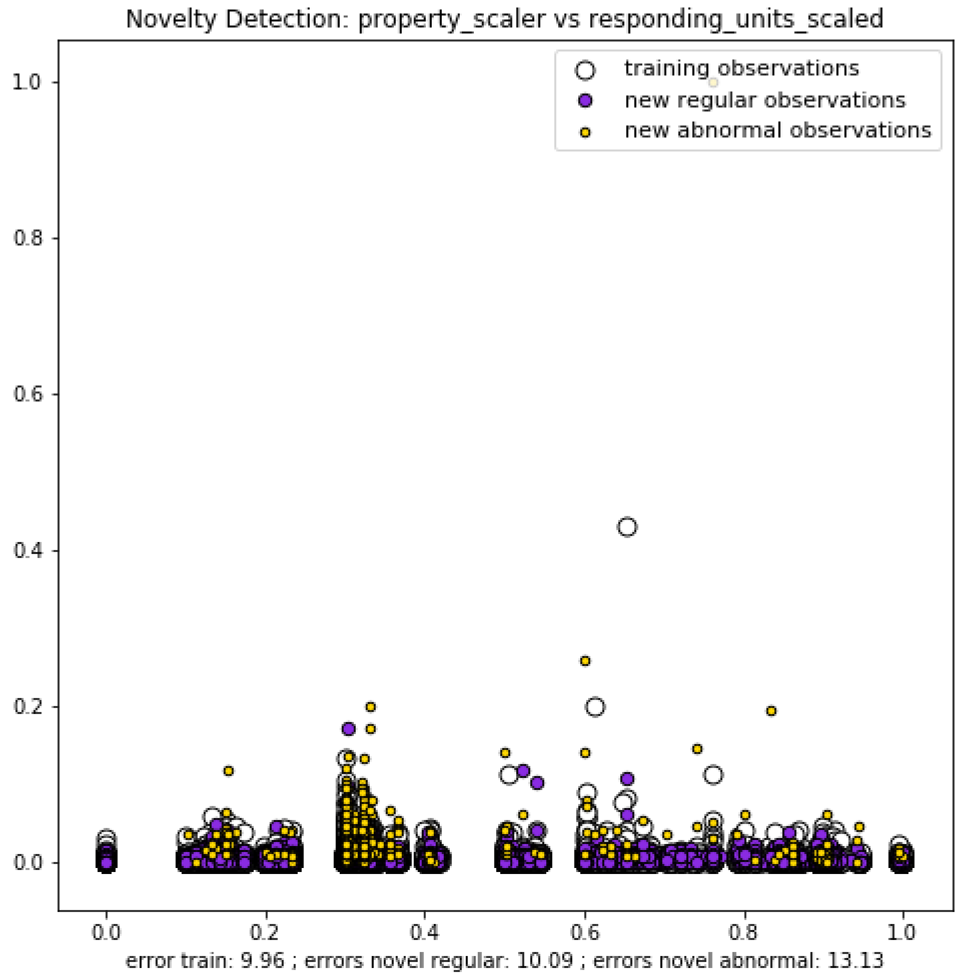
Scatter plotting between total num personnel and smoke alarm impact on num evac

In [98]:
```
scatter_plot(0,1)
```



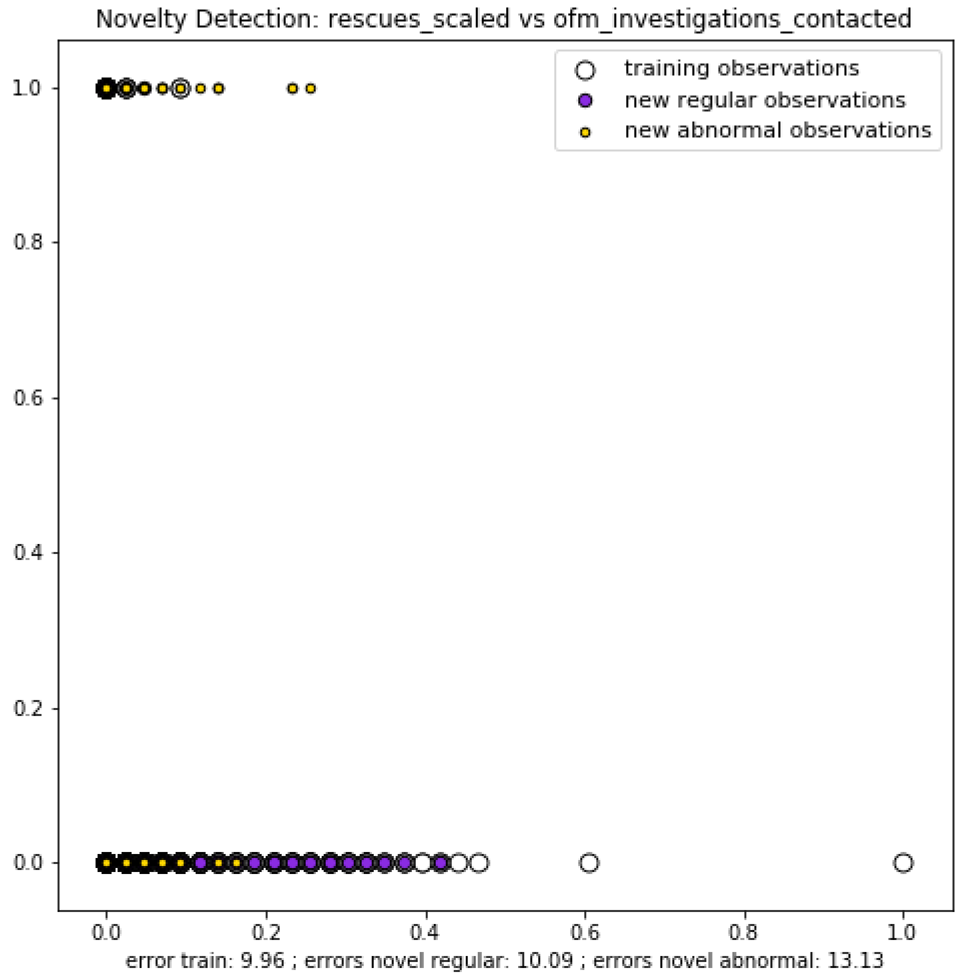
Scatter plotting between property and responding units

In [99]:
```
scatter_plot(2,74)
```



Scatter plotting between rescures and ofm contacted

In [100]:
```
scatter_plot(71,73)
```



Conclusion

Even though our accuracy metric ('AUC') is random. It does make sense why it's that bad.

- After looking at all the plots while exploring the data. It comes as no surprise that there is not a single feature which we could have used that would have been a good predictor for injuries/fatalities
- For any feature we had injuries/fatalities we also had no injuries. For that reason the classifer couldn't draw a decision boundary to separate injuries with no injuries.
- The data we have is skewed. The data we have from 2011 - 2016, only 0.12 reflect injuries/fatalities. That means that 99.88 % data has no injuries

Future Features which could be a good predictor for injuries

After analyzing the dataset. There are couple of features which comes to mind which could be a good predictor for injuries / fatalities

- Having the subscript of the 911 call could be a good features for predicting injuries / fatalities.
- If we have a features where we know that a specific ambulance on scene was also arrived at one of the hospital or it made a call to the hospital that could also be a good predictor of injuries / fatalities

Following work done by: ARJUN VERMA

DBSCAN and KMEANS

In [1]:
```
#Import the libraries
import pandas as pd
import numpy as np
import os
from IPython.display import display
pd.set_option('display.max_columns',200)
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
```

Load the scaled file for modeling

In [2]:
```
#Drop the columns not needed
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
from collections import Counter
from sklearn.decomposition import PCA
from sklearn.model_selection import StratifiedShuffleSplit
import pandas as pd
```

In [3]:
```
df_analysis = pd.read_csv('C:/TFS_Final_Scaled.csv', low_memory=False)
df_analysis = df_analysis.dropna()
df_analysis = df_analysis.drop("incident_number",axis=1)
df_analysis = df_analysis.drop("incident_date_time",axis=1)
df_analysis = df_analysis.drop("Unnamed: 0",axis=1)
print(df_analysis.shape)
#df_analysis = df_analysis.head(100000)
print("file loaded...")
```

```
(720370, 88)
file loaded...
```

Split the data in train and test sets

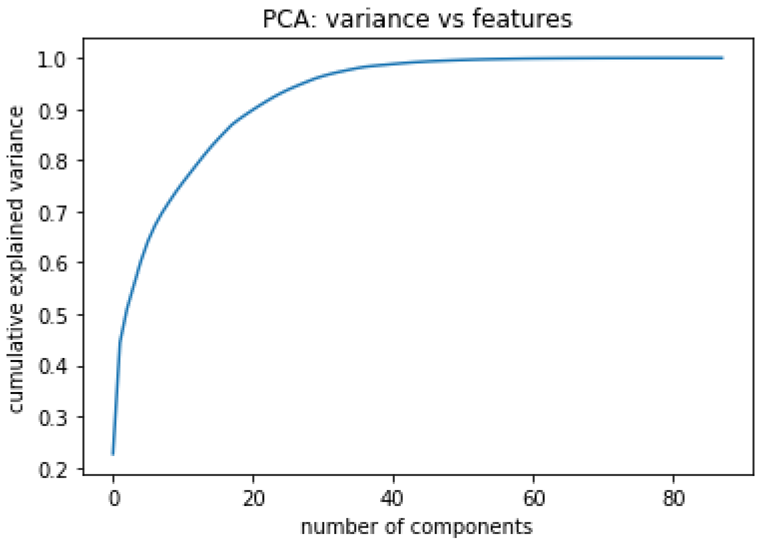
In [4]:
```
#Split to train and test set
#stratify = y
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(df_analysis, df_analysis["total_inj
_fatality"]):
    start_train_set = df_analysis.loc[train_index]
    start_test_set = df_analysis.loc[test_index]

X_train_set = start_train_set.drop("total_inj_fatality", axis = 1)
y_train_set = start_train_set["total_inj_fatality"].copy()
X_test_set = start_test_set.drop("total_inj_fatality", axis = 1)
y_test_set = start_test_set["total_inj_fatality"].copy()
X_train_set_sc = StandardScaler().fit_transform(X_train_set)
X_test_set_sc = StandardScaler().fit_transform(X_test_set)

X_train_set_inj = start_train_set
X_test_set_inj = start_test_set
X_train_set_sc = StandardScaler().fit_transform(X_train_set)
X_test_set_sc = StandardScaler().fit_transform(X_test_set)
X_train_set_inj_sc = StandardScaler().fit_transform(X_train_set_inj)
X_test_set_inj_sc = StandardScaler().fit_transform(X_test_set_inj)

print("done")
```

```
done
```

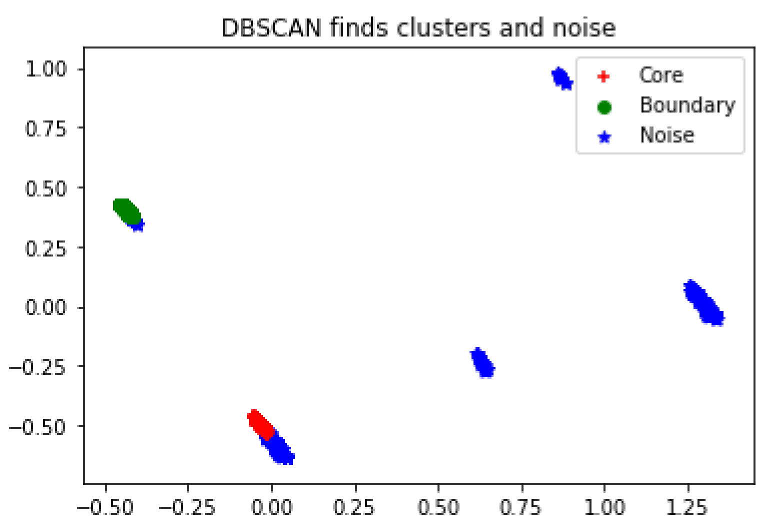
DBSCAN

In [8]:
```
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix, classification_report

#Compute DBSCAN
print("start")
print("start_test_set shape:", start_test_set.shape)

#Explained variance
pca = PCA().fit(start_train_set)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.title("PCA: variance vs features")
plt.show()

#First 13 explain about 80% of the variance, we reduce the dimension to cluste
r and improve performance
pca = PCA(n_components=13).fit(start_test_set)
print("PCA explained variance ration: ",pca.explained_variance_ratio_)

pca_2d = pca.transform(start_test_set)
print("pca 2d shape: ", pca_2d.shape)

db = DBSCAN(eps=.2, min_samples=60, n_jobs=-1, algorithm='auto').fit(pca_2d)
#core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
#core_samples_mask[db.core_sample_indices_] = True

print(db.labels_)
from collections import Counter
labels = db.labels_
print(Counter(db.labels_))
print("Outlier with injury: ", start_test_set[db.labels_==-1].total_inj_fatali
ty.sum())

# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

print('Estimated number of clusters: %d' % n_clusters_)
print('Estimated number of noise points: %d' % n_noise_)

#pca = PCA(n_components=2).fit(start_test_set)
#pca_2d = pca.transform(start_test_set)
for i in range(0, pca_2d.shape[0]):
    if db.labels_[i] == 0:
        c1 = plt.scatter(pca_2d[i,0],pca_2d[i,1],c='r', marker='+')
    elif db.labels_[i] == 1:
        c2 = plt.scatter(pca_2d[i,0],pca_2d[i,1],c='g', marker='o')
    elif db.labels_[i] == -1:
        c3 = plt.scatter(pca_2d[i,0],pca_2d[i,1],c='b', marker='*')

plt.legend([c1, c2, c3], ['Core', 'Boundary','Noise'])
plt.title('DBSCAN finds clusters and noise')
plt.show()

print("done")
```

```
start
start_test_set shape: (144074, 88)
```




```
PCA explained variance ration:  [0.22701159 0.21772559 0.06594063 0.04750739
0.04589257 0.03903732
 0.0307277  0.02470983 0.02110208 0.01972052 0.01844721 0.01797333
 0.01789213]
pca 2d shape:  (144074, 13)
[ 0  1  1 ...  0 43  0]
Counter({1: 51723, 0: 22749, 7: 9634, 2: 6438, 14: 6263, 9: 4851, 8: 4752, 1
1: 4157, 12: 4141, 3: 2856, 24: 2822, 5: 2655, 17: 2392, 4: 2242, 29: 2054, 2
2: 1224, 13: 1140, -1: 1090, 23: 1054, 39: 877, 35: 695, 25: 628, 18: 587, 1
9: 585, 21: 546, 33: 516, 27: 512, 30: 428, 41: 392, 45: 357, 6: 326, 31: 29
3, 16: 259, 43: 242, 40: 218, 37: 198, 26: 197, 20: 193, 38: 183, 28: 160, 4
2: 131, 15: 129, 34: 123, 44: 111, 32: 109, 36: 107, 52: 105, 10: 102, 46: 9
8, 50: 88, 51: 75, 48: 72, 47: 68, 53: 64, 49: 63})
Outlier with injury:  23
Estimated number of clusters: 54
Estimated number of noise points: 1090
```



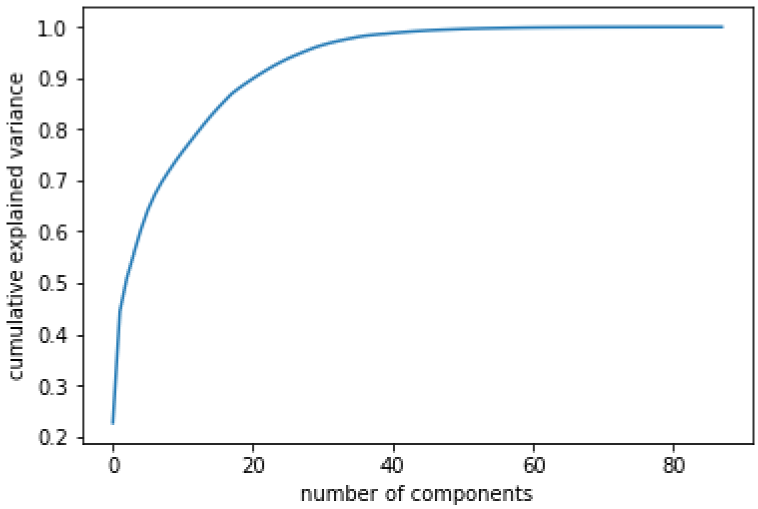

```
done
```

KMeans

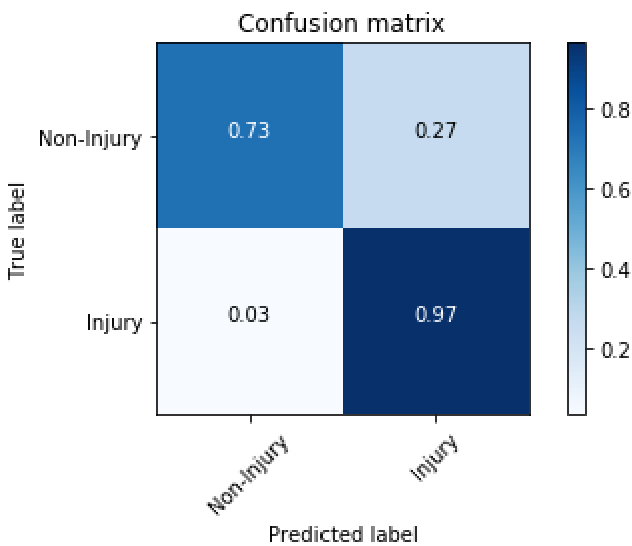
In [133]:
```
from sklearn.metrics import confusion_matrix, classification_report
import itertools

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

    print(cm)

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

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

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


print("start")


#Explained variance
pca = PCA().fit(start_train_set)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.title("PCA: variance vs features")
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.show()

#First 13 explain about 80% of the variance, we reduce the dimension to cluste
r and improve performance
pca = PCA(n_components=13).fit(X_train_set_sc)
print("PCA explained variation: ",pca.explained_variance_ratio_)

X_train_set_sc_pca = pca.transform(X_train_set_sc)
X_test_set_sc_pca = pca.transform(X_test_set_sc)
```

```
kmeans = KMeans(n_clusters=2,random_state=123,n_jobs=-1,precompute_distances=True,max_iter=1000,init='k-means++')
kmeans = kmeans.fit(X_train_set_sc_pca)
y_train_set_pred = kmeans.predict(X_train_set_sc_pca)

#Test Set
y_test_set_pred = kmeans.predict(X_test_set_sc_pca)
class_names = {'Non-Injury','Injury'}

# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test_set, y_test_set_pred)
np.set_printoptions(precision=2)
y_test_set_np = np.array(y_test_set)

# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                      title='Confusion matrix')

plt.show()
print(classification_report(y_test_set,y_test_set_pred))
print("done")
```

```
start
```




```
PCA explained variance ration:  [0.04 0.02 0.02 0.02 0.01 0.01 0.01 0.01 0.01
0.01 0.01 0.01 0.01]
Normalized confusion matrix
[[0.73 0.27]
 [0.03 0.97]]
```




```
             precision    recall  f1-score   support

          0       1.00      0.73      0.85    215844
          1       0.00      0.97      0.01       267

avg / total       1.00      0.73      0.85    216111

done
```

In [ ]:

Following work done by: DAVID SIGNORETTI

KNN and SVC on Unscaled data

In [1]:
```
import pandas as pd
import numpy as np
import datetime as dt
from IPython.display import display
import warnings
import matplotlib.pyplot as plt

%matplotlib inline
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns',200)
```

In [2]:
```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
```

In [3]:
```
_df = pd.read_csv('TFS_Final_Unscaled.csv')
```

In [4]:
```
_df.drop(['Unnamed: 0','incident_number','incident_date_time'], axis=1, inplac
e=True)
```

In [5]:
```
_df.head()
```

Out[5]:

	rescues_unscaled	min_to_reach_unscaled	smoke_alarm_impact_on_num_evac	property	respo
0	0	7	0	301.0	
1	0	6	0	301.0	
2	0	4	0	302.0	
3	0	6	0	861.0	
4	0	5	0	323.0	

In [6]:
```
# debug size
#_df = _df.iloc[0:200000,:]
```

In [7]:
```
_df.shape
```

Out[7]:
```
(200000, 16)
```

In [8]:
```
X = _df.iloc[:,0:15]
Y = _df.iloc[:,15]

# One-third of data as a part of test set
validation_size = 0.33

seed = 7
x_t, x_v, y_t, y_v = train_test_split(X, Y, test_size=validation_size, \
                                                      random_state=seed)

print('Testing Size - ', len(x_t))
print('Training Size - ', len(x_v))
```

```
Testing Size -  134000
Training Size -  66000
```

KNN

In [9]:
```
knn_ = KNeighborsClassifier(n_neighbors= 3,\
                            weights='distance',\
                            metric='euclidean',\
                            algorithm='kd_tree')

scores = cross_val_score(knn_, x_t, y_t, cv=2, scoring='accuracy')
print('Mean',scores.mean())
print('STD',scores.std())

knn_.fit(x_t,y_t)
knn_p = knn_.predict(x_v)
print(knn_p)
```

```
Mean 0.9989701474713678
STD 0.00011941835600079276
[0 0 0 ... 0 0 0]
```

SVC

In [10]:
```
svc_ = SVC(kernel='rbf')

scores = cross_val_score(svc_, x_t, y_t, cv=5, scoring='accuracy')
print('Mean',scores.mean())
print('STD',scores.std())

svc_.fit(x_t,y_t)
svc_p = svc_.predict(x_v)
print(svc_p)
```

```
Mean 0.9991567172528182
STD 5.0600609897574254e-05
[0 0 0 ... 0 0 0]
```

In [ ]:

KNN and SVC on Scaled Data

In [1]:
```
import pandas as pd
import numpy as np
import datetime as dt
from IPython.display import display
import warnings
import matplotlib.pyplot as plt

%matplotlib inline
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns',200)
```

In [2]:
```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
```

In [3]:
```
_df = pd.read_csv('TFS_Final_Scaled.csv')
```

In [4]:
```
_df.drop(['Unnamed: 0','incident_number','incident_date_time'], axis=1, inplac
e=True)
```

In [5]:
```
_df.head()
```

Out[5]:

	total_num_personnel_scaler	smoke_alarm_impact_on_num_evac_scalar	property_scaler	respon
0	0.003132	0.0	0.301301	
1	0.003132	0.0	0.301301	
2	0.003132	0.0	0.302302	
3	0.003132	0.0	0.861862	
4	0.010963	0.0	0.323323	

In [6]:
```
# debug size
#_df = _df.iloc[0:50000,:]
```

In [7]:
```
_df.shape
```

Out[7]:
```
(720370, 88)
```

In [8]:
```
X = _df.iloc[:,0:87]
Y = _df.iloc[:,87]

# One-third of data as a part of test set
validation_size = 0.33

seed = 7
x_t, x_v, y_t, y_v = train_test_split(X, Y, test_size=validation_size, \
                                                      random_state=seed)

print('Testing Size - ', len(x_t))
print('Training Size - ', len(x_v))
```

```
Testing Size -  482647
Training Size -  237723
```

KNN

In [9]:
```
knn_ = KNeighborsClassifier(n_neighbors= 3,\
                           weights='distance',\
                           metric='euclidean',\
                           algorithm='kd_tree')

scores = cross_val_score(knn_, x_t, y_t, cv=2, scoring='accuracy')
print('Mean',scores.mean())
print('STD',scores.std())

knn_.fit(x_t,y_t)
knn_p = knn_.predict(x_v)
print(knn_p)
```

```
Mean 0.9984916513254847
STD 3.729121211015762e-05
[0 0 0 ... 0 0 0]
```

SVC

In [10]:
```
svc_ = SVC(kernel='rbf')

scores = cross_val_score(svc_, x_t, y_t, cv=5, scoring='accuracy')
print('Mean',scores.mean())
print('STD',scores.std())

svc_.fit(x_t,y_t)
svc_p = svc_.predict(x_v)
print(svc_p)
```

```
Mean 0.9987796464382107
STD 4.135013539885878e-06
[0 0 0 ... 0 0 0]
```

CONCLUSION

Even though our accuracy metric ('AUC') is random. It does make sense why it's that bad.

- After looking at all the plots while exploring the data. It comes as no surprise that there is not a single feature which we could have used that would have been a good predictor for injuries/fatalities
- For any feature we had injuries/fatalities we also had no injuries. For that reason the classifer couldn't draw a decision boundary to separate injuries with no injuries.
- The data we have is skewed. The data we have from 2011 - 2016, only 0.12 reflect injuries/fatalities. That means that 99.88 % data has no injuries

Future Features which could be a good predictor for injuries

After analyzing the dataset. There are couple of features which comes to mind which could be a good predictor for injuries / fatalities

- Having the subscript of the 911 call could be a good features for predicting injuries / fatalities.
- If we have a features where we know that a specific ambulance on scene was also arrived at one of the hospital or it made a call to the hospital that could also be a good predictor of injuries / fatalities

In [ ]:

In [ ]:

In [ ]: