In [3]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as mticks
%matplotlib inline
import seaborn as sns
sns.set()
from scipy import stats
from sklearn import linear_model, metrics, model_selection, neighbors, naive_bayes, svm,
preprocessing, feature_selection

##import eveything we need for analysing data, visualing data and machine learning
```

Pilot Study

In [4]:

```
manchester_traffic = pd.read_csv('tfgm Accident Casualty Severity 2018 Balanced.csv')
manchester_traffic.head(10)
##importing and reading data
```

Out[4]:

	Accident Index	OutputDate	Day	OutputTime	Easting	Northing	LocalAuthority	Severity	VehicleType	NumberCasualties	F
0	1.094790e+11	06/09/2018	5	12:45	389003	390192	109	3	9	1	
1	1.043990e+11	20/01/2018	7	15:53	391770	404621	104	3	9	1	
2	1.144420e+11	27/05/2018	1	13:46	370945	401000	114	2	9	1	
3	1.145070e+11	21/11/2018	4	12:28	364451	403013	114	3	9	1	
4	1.005260e+11	15/11/2018	5	00:03	369333	407889	100	3	9	1	
5	1.125000e+11	02/11/2018	6	14:20	377504	387336	112	2	9	1	
6	1.144170e+11	16/03/2018	6	11:00	365444	399781	114	3	5	1	
7	1.104820e+11	25/08/2018	7	21:04	394939	398987	110	3	9	2	
8	1.145170e+11	18/12/2018	3	07:15	354096	410535	114	3	19	1	
9	1.024530e+11	05/06/2018	3	20:15	385094	395527	102	2	1	1	

In [5]:

```
manchester_traffic.shape
##check how shape of dataset. There are 400 rows and 19 columns
```

Out[5]:

```
(400, 19)
```

In [9]:

```
print(manchester_traffic.isnull().sum())
##check if there are any null values. Results shows there are not any null values in the
dataset
```

```
Accident Index      0
OutputDate          0
Day                 0
OutputTime          0
Easting             0
Northing            0
LocalAuthority      0
```

```
Severity            0
VehicleType         0
NumberCasualties    0
PedMovement         0
PedLocation         0
PoliceReported      0
Manoeuvre           0
NumberVehicles      0
CasualtyClass       0
CarPassenger        0
Skidding            0
SevereCasualty      0
dtype: int64
```

In [10]:

```
manchester_traffic.drop_duplicates(keep='first',inplace=True)
##drop duplicate rows but keep first row
```

In [13]:

```
manchester_traffic[(manchester_traffic['Accident Index'] == 1.124430e+11)]
##checking duplicate row has been deleted
```

Out[13]:

	Accident Index	OutputDate	Day	OutputTime	Easting	Northing	LocalAuthority	Severity	VehicleType	NumberCasualties
78	1.124430e+11	31/05/2018	5	21:45	379735	396267	112	1	9	6

In [14]:

```
manchester_traffic.shape
##checking duplicate row has been deleted
```

Out[14]:

```
(398, 19)
```

In [15]:

```
manchester_traffic.info()
##checking datatype make sure it is right. We can see that there is an issue OutputDate a
nd OutTime should be datetime
##datatype however it is object this needs to change.
##To make it easier to view data we will combine both columns OutputDate and OutTime and
change to datetime
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 398 entries, 0 to 399
Data columns (total 19 columns):
Accident Index      398 non-null float64
OutputDate          398 non-null object
Day                 398 non-null int64
OutputTime          398 non-null object
Easting             398 non-null int64
Northing            398 non-null int64
LocalAuthority      398 non-null int64
Severity            398 non-null int64
VehicleType         398 non-null int64
NumberCasualties    398 non-null int64
PedMovement         398 non-null int64
PedLocation         398 non-null int64
PoliceReported      398 non-null int64
Manoeuvre           398 non-null int64
NumberVehicles      398 non-null int64
CasualtyClass       398 non-null int64
CarPassenger        398 non-null int64
Skidding            398 non-null int64
```

```
SevereCasualty      398 non-null int64
dtypes: float64(1), int64(16), object(2)
memory usage: 62.2+ KB
```

In [16]:

```python
manchester_traffic['Date_time'] =  manchester_traffic['OutputDate'] +' '+ manchester_traffic['OutputTime']
##To make it easier to view data we have combine both columns OutputDate and OutTime and change to datetime
```

In [17]:

```python
manchester_traffic["Date_time"]= pd.to_datetime(manchester_traffic["Date_time"] )
##Change our new column Date_time datatype to datetime
```

In [18]:

```python
manchester_traffic.drop(['OutputDate','OutputTime'],axis =1 , inplace=True)
##We no longer need OutputDate and OutputTime. We will drop those 2 columns
```

In [19]:

```python
manchester_traffic.head()
##We Check columns have been dropped
```

Out[19]:

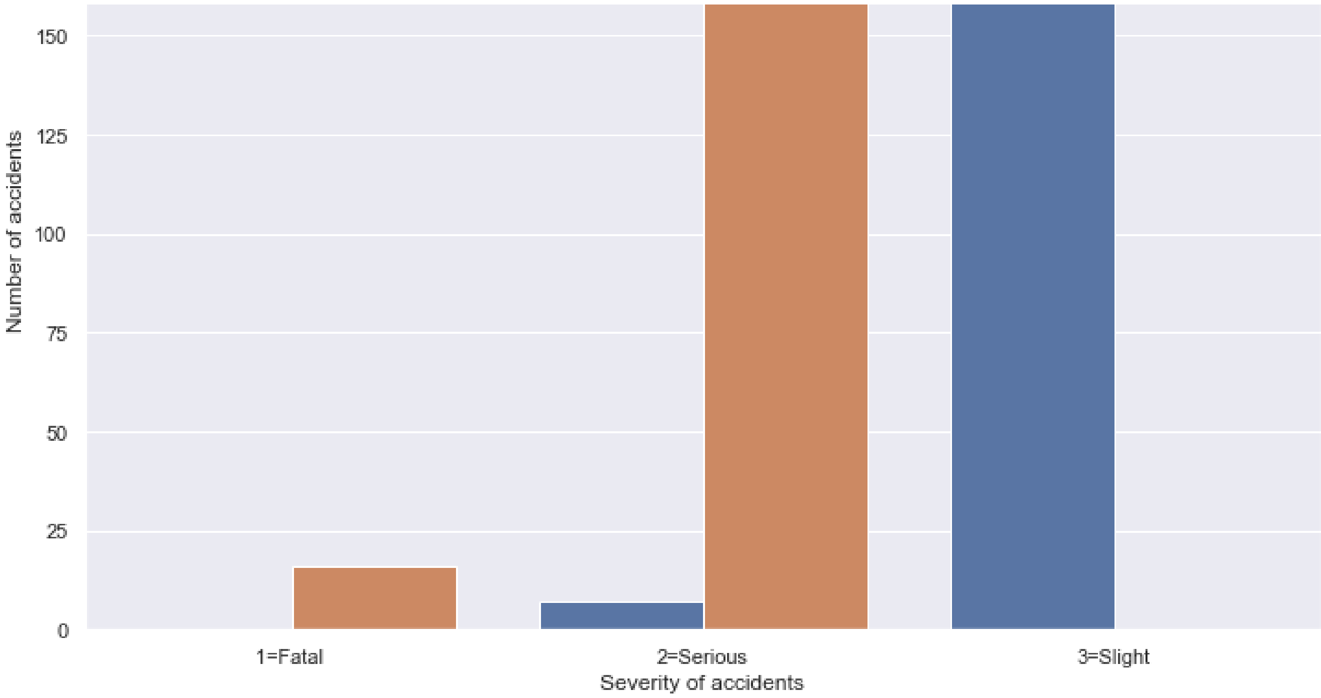
	Accident Index	Day	Easting	Northing	LocalAuthority	Severity	VehicleType	NumberCasualties	PedMovement	PedLocatio
0	1.094790e+11	5	389003	390192	109	3	9	1	0	(
1	1.043990e+11	7	391770	404621	104	3	9	1	0	(
2	1.144420e+11	1	370945	401000	114	2	9	1	0	(
3	1.145070e+11	4	364451	403013	114	3	9	1	0	(
4	1.005260e+11	5	369333	407889	100	3	9	1	0	(

In [44]:

```python
fig, ax = plt.subplots(figsize=(12,8))
sns.countplot('Severity', data=manchester_traffic, hue='SevereCasualty', ax=ax)
ax.set_ylabel('Number of accidents')
ax.set_xlabel('Severity of accidents')
ax.set_xticklabels(['1=Fatal ','2=Serious','3=Slight'])
ax.legend(['No','Yes'], title='Are casualties resulting from road traffic accidents severe?')
##Countplot to show number accidents grouped by severity of accidents  when casualites are severe and not severe
```

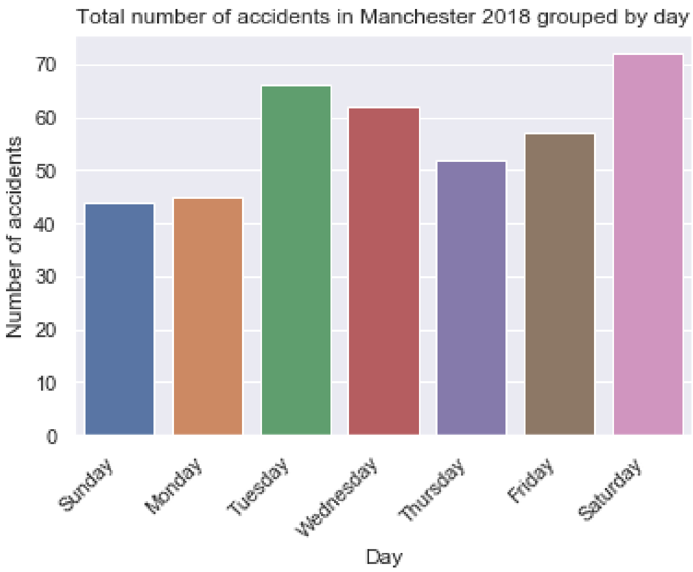
Out[44]:

<matplotlib.legend.Legend at 0x2050fb33320>



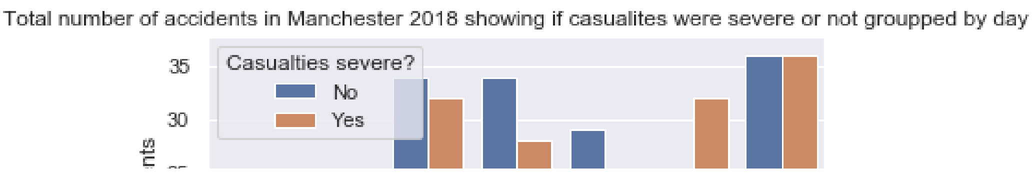
In [43]:

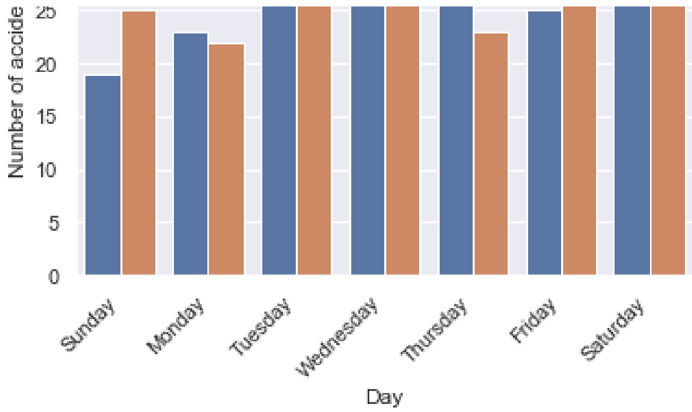
```
ax = sns.countplot(x="Day", data=manchester_traffic)
ax.set_xticklabels(['Sunday','Monday','Tuesday','Wednesday','Thursday','Friday','Saturday
'])
ax.set_title("Total number of accidents in Manchester 2018 grouped by day");
ax.set_ylabel('Number of accidents')
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right');
##Count plot showing total number of accidents in Manchester 2018 grouped by day
```



In [52]:

```
ax = sns.countplot(x="Day", hue="SevereCasualty", data=manchester_traffic)
ax.set_xticklabels(['Sunday','Monday','Tuesday','Wednesday','Thursday','Friday','Saturday
'])
ax.legend(['No','Yes'], title='Casualties severe?')
ax.set_ylabel('Number of accidents')
ax.set_title("Total number of accidents in Manchester 2018 showing if casualites were sev
ere or not groupped by day");
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right');
##Count plot showing total number of accidents in Manchester 2018 showing if casualites w
ere severe or not groupped by day
```



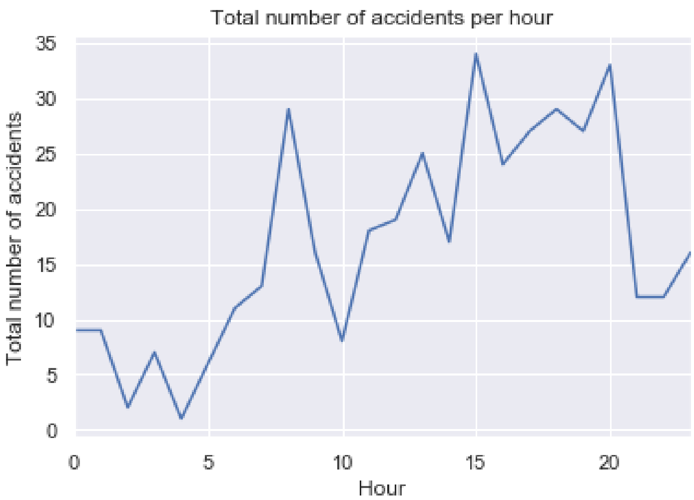


In [54]:

```
time_x = manchester_traffic['Date_time'].dt.hour
time_x.value_counts().sort_index().plot()
plt.xlabel('Hour')
plt.ylabel('Total number of accidents')
plt.title('Total number of accidents per hour')
##Time plot showing total number of accidents per hour in Manchester 2018
```

Out[54]:

```
Text(0.5, 1.0, 'Total number of accidents per hour')
```

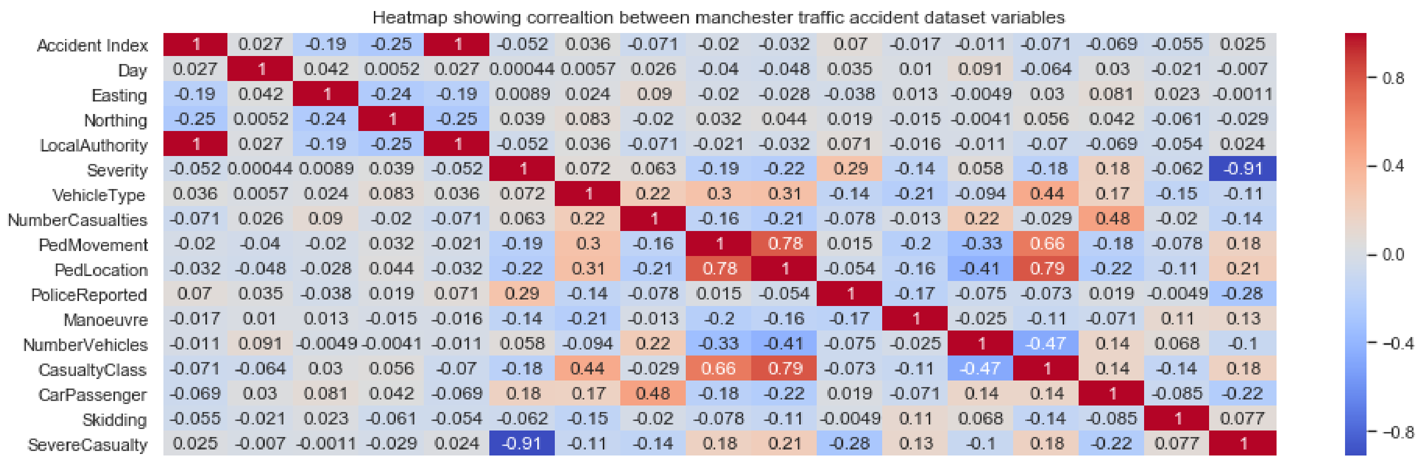


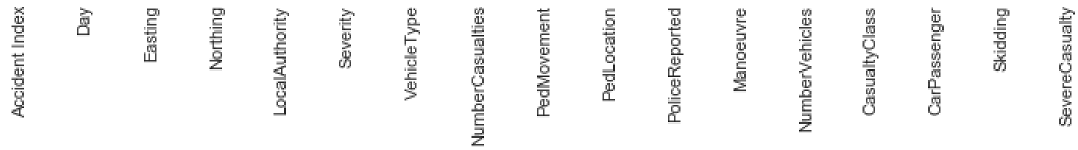
In [59]:

```
plt.figure(figsize = (16,5))
sns.heatmap(manchester_traffic.corr(),cmap='coolwarm',annot=True)
plt.title('Heatmap showing correaltion between manchester traffic accident dataset variab
les')
##Heatmap showing correaltion between manchester traffic accident dataset variables
```

Out[59]:

```
Text(0.5, 1, 'Heatmap showing correaltion between manchester traffic accident dataset var
iables')
```





Machine learning

Baseline modeling

In [67]:

```
manchester_traffic['Accident Month'] = manchester_traffic['Date_time'].dt.month
```

In [68]:

```
manchester_traffic['Accident Hour'] =  manchester_traffic['Date_time'].dt.hour
```

In [69]:

```
manchester_traffic.drop(['Date_time'],axis =1 , inplace=True)
## Machine learning models does not accept data with datetime datatype so created two new
##columns called accident month and hour, datatype int and dropped Date_time column
```

In [71]:

```
manchester_traffic.head()
##check above changes
```

Out[71]:

	Accident Index	Day	Easting	Northing	LocalAuthority	Severity	VehicleType	NumberCasualties	PedMovement	PedLocatio
0	1.094790e+11	5	389003	390192	109	3	9	1	0	(
1	1.043990e+11	7	391770	404621	104	3	9	1	0	(
2	1.144420e+11	1	370945	401000	114	2	9	1	0	(
3	1.145070e+11	4	364451	403013	114	3	9	1	0	(
4	1.005260e+11	5	369333	407889	100	3	9	1	0	(

In [72]:

```
manchester_traffic['SevereCasualty'].value_counts(normalize=True)

## This tells us that in the data around 50% of casulaties of accident is severe
```

Out[72]:

```
0    0.502513
1    0.497487
Name: SevereCasualty, dtype: float64
```

In [93]:

```
X = manchester_traffic.drop(['Accident Index','SevereCasualty'],axis =1)
y = manchester_traffic['SevereCasualty']
## We want to predict quality from everything else, so easiest thing is to drop that one
column and
##accident index because the column is only for reference sake  to get the features
```

In [94]:

```
rs = np.random.RandomState(seed=20)
```

```
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size = 0.
4, shuffle=True, random_state=rs)
X_test, X_valid, y_test, y_valid = model_selection.train_test_split(X_test, y_test, test
_size = 0.5, shuffle=True, random_state=rs)

## create our three sets: train, test and validation
## use a set seed to make this work repeatable
```

In [95]:

```
print(X_test.shape)
print(X_valid.shape)
print(X_train.shape)

## just sanity check that proportions are correct
```

```
(80, 17)
(80, 17)
(238, 17)
```

In [76]:

```
baseline_predictions = [0 for x in y_valid]
metrics.accuracy_score(y_valid, baseline_predictions)
## baseline predicition not using any model to see probaility that accident did not cause
d severe casualties
```

Out[76]:

```
0.5125
```

In [78]:

```
baseline_predictions = [1 for x in y_valid]
metrics.accuracy_score(y_valid, baseline_predictions)
## baseline predicition not using any model to see probaility that accident did  cause se
vere casualties
```

Out[78]:

```
0.4875
```

In [96]:

```
classifiers = [neighbors.KNeighborsClassifier(),
               naive_bayes.GaussianNB(),
               naive_bayes.MultinomialNB(),
               linear_model.LogisticRegression(solver='lbfgs', multi_class='ovr', max_it
er=9999),
               svm.LinearSVC(max_iter=9999),
               svm.SVC(kernel='rbf', gamma='auto', max_iter=9999)]
classifier_names = ['KNN','Gaussian NB','Multinomial NB','Logistic','Linear SVM','Non-Lin
ear SVM']
accuracies = []
for clf, name in zip(classifiers,classifier_names):
    clf.fit(X_train, y_train)
    predictions = clf.predict(X_valid)
    acc = metrics.accuracy_score(predictions, y_valid)
    accuracies.append(acc)
models = pd.DataFrame({'model':classifier_names, 'baseline':accuracies})
models

## table showing baseline/accuracy when using model KNN and Gaussian..We can see that Gau
ssian NB is more accurate than KNN
```

```
C:\Users\Chandni\Anaconda3\lib\site-packages\sklearn\svm\base.py:929: ConvergenceWarning:
Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

Out[96]:

	model	baseline

	model	baseline
0	KNN	0.6125
1	Gaussian NB	0.9125
2	Multinomial NB	0.4750
3	Logistic	0.4500
4	Linear SVM	0.4875
5	Non-Linear SVM	0.5250

Feature Scaling

In [80]:

```
scaler = preprocessing.MinMaxScaler()
X = scaler.fit_transform(X)

##Normalising the data to improve accuracy of model
```

In [81]:

```
rs = np.random.RandomState(seed=20)
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size = 0.
4, shuffle=True, random_state=rs)
X_test, X_valid, y_test, y_valid = model_selection.train_test_split(X_test, y_test, test
_size = 0.5, shuffle=True, random_state=rs)
print(X_train.shape)
print(X_valid.shape)
print(X_test.shape)
## create our three sets: train, test and validation
## use a set seed to make this work repeatable
```

```
(238, 17)
(80, 17)
(80, 17)
```

In [82]:

```
classifiers = [neighbors.KNeighborsClassifier(),
               naive_bayes.GaussianNB(),
               naive_bayes.MultinomialNB(),
               linear_model.LogisticRegression(solver='lbfgs', multi_class='ovr', max_it
er=9999),
               svm.LinearSVC(max_iter=9999),
               svm.SVC(kernel='rbf', gamma='auto', max_iter=9999)]
classifier_names = ['KNN','Gaussian NB','Multinomial NB','Logistic','Linear SVM','Non-Lin
ear SVM']
accuracies = []
for clf, name in zip(classifiers,classifier_names):
    clf.fit(X_train, y_train)
    predictions = clf.predict(X_valid)
    acc = metrics.accuracy_score(predictions, y_valid)
    accuracies.append(acc)
models = pd.DataFrame({'model':classifier_names, 'scaled':accuracies})
models
## table showing baseline/accuracy when using model after scalling
##KNN and Gaussian..We can see that Gaussian NB is more accurate than KNN
```

Out[82]:

	model	scaled
0	KNN	0.8625
1	Gaussian NB	0.9375
2	Multinomial NB	0.7375
3	Logistic	0.9750
4	Linear SVM	0.9750

	model	scaled
5	Non-Linear SVM	0.9750

Feature Selection Method - Wrapper methods

In [84]:

```
columns = manchester_traffic.drop(['Accident Index','SevereCasualty'],axis =1).columns.t
olist()
num_to_keep = 5
best_features = []
for i in range(0, num_to_keep):
    best_acc = 0.0
    feature_to_add = ''
    for column in columns:
        feature_num = columns.index(column)
        if feature_num in best_features:
            continue
        features = best_features+[feature_num]
        clf = neighbors.KNeighborsClassifier()
        clf.fit(X_train[:,features], y_train)
        predictions = clf.predict(X_valid[:,features])
        acc = metrics.accuracy_score(predictions, y_valid)
        if acc > best_acc:
            best_acc = acc
            feature_to_add = feature_num
    best_features.append(feature_to_add)
print(best_features)
##using feature selection method getting best 5 features to use with KNN
```

```
[4, 0, 1, 2, 3]
```

In [85]:

```
columns = manchester_traffic.drop(['Accident Index','SevereCasualty'],axis =1).columns.t
olist()
num_to_keep = 5
best_features = []
for i in range(0, num_to_keep):
    best_acc = 0.0
    feature_to_add = ''
    for column in columns:
        feature_num = columns.index(column)
        if feature_num in best_features:
            continue
        features = best_features+[feature_num]
        clf = naive_bayes.GaussianNB()
        clf.fit(X_train[:,features], y_train)
        predictions = clf.predict(X_valid[:,features])
        acc = metrics.accuracy_score(predictions, y_valid)
        if acc > best_acc:
            best_acc = acc
            feature_to_add = feature_num
    best_features.append(feature_to_add)
print(best_features)
##using feature selection method getting best 5 features to use with GaussianNB
```

```
[4, 0, 1, 2, 3]
```

In [86]:

```
subset = [4,0,1,2,3]
clf = neighbors.KNeighborsClassifier()
clf.fit(X_train[:,subset], y_train)
predictions = clf.predict(X_valid[:,subset])
acc = metrics.accuracy_score(predictions, y_valid)
acc
##KNN model give 0.975 accuracy when using the five subset given by wrapper method
```

Out[86]:

0.975

In [88]:

```
subset = [4,0,1,2,3]
clf = naive_bayes.GaussianNB()
clf.fit(X_train[:,subset], y_train)
predictions = clf.predict(X_valid[:,subset])
acc = metrics.accuracy_score(predictions, y_valid)
acc
##KNN GaussianNB give 0.975 accuracy when using the five subset given by wrapper method
```

Out[88]:

0.975

In [98]:

```
scaler = preprocessing.StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_train_scaled[0]

##preprocessing.StandardScaler to see it will help increase accuracy
```

Out[98]:

```
array([-1.12938237,  0.81254084, -0.62584206, -0.74342974,  0.9638678 ,
       -1.9270804 , -0.56312675, -0.41412594, -0.50214147,  1.78197604,
        0.62915887,  0.0758575 , -0.81670666, -0.42878091,  3.96258533,
       -0.41975714, -0.99985999])
```

In [100]:

```
classifiers = [neighbors.KNeighborsClassifier(),
               naive_bayes.GaussianNB(),
               naive_bayes.MultinomialNB(),
               linear_model.LogisticRegression(solver='lbfgs', multi_class='ovr', max_it
er=9999),
               svm.LinearSVC(max_iter=9999),
               svm.SVC(kernel='rbf', gamma='auto', max_iter=9999)]
classifier_names = ['KNN','Gaussian NB','Logistic','Linear SVM','Non-Linear SVM']
X_valid_scaled = scaler.transform(X_valid)
accuracies = []
for clf in classifiers:
    if isinstance(clf,naive_bayes.MultinomialNB):
        accuracies.append('N/A')
    else:
        clf.fit(X_train_scaled, y_train)
        predictions = clf.predict(X_valid_scaled)
        acc = metrics.accuracy_score(predictions, y_valid)
        accuracies.append(acc)
models['scaled'] = accuracies
models

## Because we have standardised, we have some negative values and MultinomialNB does not
work anymore, need to enter N/A
## Overall, a little improvement but not much
```

Out[100]:

	model	baseline	scaled
0	KNN	0.5125	0.9125
1	Gaussian NB	0.9125	0.9375
2	Multinomial NB	0.4750	N/A
3	Logistic	0.4500	0.975
4	Linear SVM	0.4875	0.975
5	Non-Linear SVM	0.5250	0.975

model baseline scaled

In [101]:

```python
best = 0.0
best_n = 0
best_weight = ''
best_p = 0
for n in range(3,16):
    for weight in ['uniform','distance']:
        for p in [1,2,3,4,5]:
            clf = neighbors.KNeighborsClassifier(n_neighbors=n, weights=weight, metric='
minkowski', p=p)
            clf.fit(X_train_scaled, y_train)
            predictions = clf.predict(X_valid_scaled)
            acc = metrics.accuracy_score(y_valid, predictions)
            if acc > best:
                best = acc
                best_n = n
                best_weight = weight
                best_p = p
print('Best Accuracy was '+str(best)+' using '+str(best_n)+' neighbours, '+str(weight)+'
weighting and p='+str(p))

## Try a load of different neighbours, weights and p values to find the best
## Bear in mind danger of overfitting by picking best on validation set
## Probably sensible to look at every accuracy score, see if there's a trend or if its ra
ndom fluctuations
```

```
Best Accuracy was 0.975 using 13 neighbours, distance weighting and p=5
```

In [ ]:

```python

```

In [ ]:

```python

```

Main Study

In [102]:

```python
all_manchester_traffic = pd.read_csv('tfgm Accident Casualty Severity 2018 All.csv')
all_manchester_traffic.head(20)
##importing and reading data
```

Out[102]:

	Accident Index	Severity	NumberVehicles	NumberCasualties	OutputDate	Day	OutputTime	Easting	Northing	LocalAuthor
0	1.003910e+11	3	2	1	01/01/2018	2	20:40	373324	407137	1
1	1.013890e+11	3	1	1	01/01/2018	2	23:20	381896	410649	1
2	1.013900e+11	3	1	1	01/01/2018	2	17:50	378030	407548	1
3	1.013910e+11	3	2	3	01/01/2018	2	19:03	381389	410964	1
4	1.013910e+11	3	2	3	01/01/2018	2	19:03	381389	410964	1
5	1.013910e+11	3	2	3	01/01/2018	2	19:03	381389	410964	1
6	1.013910e+11	3	1	1	01/01/2018	2	03:40	379089	411664	1
7	1.023870e+11	3	2	2	01/01/2018	2	18:25	384824	395127	1
8	1.023870e+11	3	2	2	01/01/2018	2	18:25	384824	395127	1
9	1.023920e+11	3	2	1	01/01/2018	2	19:32	384121	398252	1
10	1.024020e+11	3	2	1	01/01/2018	2	13:00	385962	401947	1
11	1.143880e+11	3	2	1	01/01/2018	2	14:00	365663	400392	1

	Accident Index	Severity	NumberVehicles	NumberCasualties	OutputDate	Day	OutputTime	Easting	Northing	LocalAutho
12	1.003870e+11	2	1	1	02/01/2018	3	16:46	374049	406848	
13	1.003900e+11	3	1	2	02/01/2018	3	20:10	372326	408901	1
14	1.003900e+11	3	1	2	02/01/2018	3	20:10	372326	408901	1
15	1.023940e+11	2	2	2	02/01/2018	3	20:28	383813	393271	1
16	1.023940e+11	2	2	2	02/01/2018	3	20:28	383813	393271	1
17	1.024210e+11	3	2	1	02/01/2018	3	17:20	384389	394533	1
18	1.064260e+11	3	2	2	02/01/2018	3	21:48	390073	412952	1
19	1.064260e+11	3	2	2	02/01/2018	3	21:48	390073	412952	1

20 rows × 49 columns

In [103]:

```
all_manchester_traffic.shape
##check how many rows and columns
```

Out[103]:

```
(5026, 49)
```

In [106]:

```
manchester_traffic.drop_duplicates(keep='first',inplace=True)
##see if any data that needs dropping
```

In [107]:

```
all_manchester_traffic.shape
##there were no duplicates
```

Out[107]:

```
(5026, 49)
```

In [108]:

```
print(all_manchester_traffic.isnull().sum())
print('Total: '  + str(all_manchester_traffic.isnull().sum().sum()))
##check if there are any null values. Results shows there are not any null values in the
dataset
```

```
Accident Index                      0
Severity                            0
NumberVehicles                      0
NumberCasualties                    0
OutputDate                          0
Day                                 0
OutputTime                          0
Easting                             0
Northing                            0
LocalAuthority                      0
Road1Class                          0
CarriagewayType                     0
SpeedLimit                          0
JunctionDetail                      0
JunctionControl                     0
Road2Class                          0
PedCrossingHumanControl             0
PedCrossingPhysicalFacilities       0
LightingCondition                   0
WeatherCondition                    0
RoadSurface                         0
SpecialConditions                   0
CarriagewayHazard                   0
PoliceReported                      0
VehicleReferenceNumber              0
```

```
VehicleType                      0
ArtTowing                        0
Manoeuvre                        0
VehicleLocationOffRoad           0
JunctionLocation                 0
Skidding                         0
HitObjectOnCWay                  0
VehicleLeaveCWay                 0
HitObjectOffCWay                 0
FirstPointImpact                 0
JourneyPurpose                   0
ForeignReg                       0
SexOfDriver                      0
AgeBandOfDriver                  0
CasualtyNumber                   0
CasualtyClass                    0
Sex                              0
AgeBandOfCasualty                0
PedLocation                      0
PedMovement                      0
CarPassenger                     0
BusPassenger                     0
PedInjWork                       0
SevereCasualty                   0
dtype: int64
Total: 0
```

In [109]:

```
all_manchester_traffic.info()
##checking datatype make sure it is right. We can see that there is an issue OutputDate a
nd OutTime should be datetime
##datatype however it is object this needs to change.
##To make it easier to view data we will combine both columns OutputDate and OutTime and
change to datetime
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5026 entries, 0 to 5025
Data columns (total 49 columns):
Accident Index                   5026 non-null float64
Severity                         5026 non-null int64
NumberVehicles                   5026 non-null int64
NumberCasualties                 5026 non-null int64
OutputDate                       5026 non-null object
Day                              5026 non-null int64
OutputTime                       5026 non-null object
Easting                          5026 non-null int64
Northing                         5026 non-null int64
LocalAuthority                   5026 non-null int64
Road1Class                       5026 non-null int64
CarriagewayType                  5026 non-null int64
SpeedLimit                       5026 non-null int64
JunctionDetail                   5026 non-null int64
JunctionControl                  5026 non-null int64
Road2Class                       5026 non-null int64
PedCrossingHumanControl          5026 non-null int64
PedCrossingPhysicalFacilities    5026 non-null int64
LightingCondition                5026 non-null int64
WeatherCondition                 5026 non-null int64
RoadSurface                      5026 non-null int64
SpecialConditions                5026 non-null int64
CarriagewayHazard                5026 non-null int64
PoliceReported                   5026 non-null int64
VehicleReferenceNumber           5026 non-null int64
VehicleType                      5026 non-null int64
ArtTowing                        5026 non-null int64
Manoeuvre                        5026 non-null int64
VehicleLocationOffRoad           5026 non-null int64
JunctionLocation                 5026 non-null int64
Skidding                         5026 non-null int64
HitObjectOnCWay                  5026 non-null int64
VehicleLeaveCWay                 5026 non-null int64
```

```
HitObjectOffCWay                5026 non-null int64
FirstPointImpact                5026 non-null int64
JourneyPurpose                  5026 non-null int64
ForeignReg                      5026 non-null int64
SexOfDriver                     5026 non-null int64
AgeBandOfDriver                 5026 non-null int64
CasualtyNumber                  5026 non-null int64
CasualtyClass                   5026 non-null int64
Sex                             5026 non-null int64
AgeBandOfCasualty               5026 non-null int64
PedLocation                     5026 non-null int64
PedMovement                     5026 non-null int64
CarPassenger                    5026 non-null int64
BusPassenger                    5026 non-null int64
PedInjWork                      5026 non-null int64
SevereCasualty                  5026 non-null int64
dtypes: float64(1), int64(46), object(2)
memory usage: 1.9+ MB
```

In [111]:

```
all_manchester_traffic['Date_time'] =  all_manchester_traffic['OutputDate'] +' '+ all_ma
nchester_traffic['OutputTime']
##To make it easier to view data we have combine both columns OutputDate and OutTime and
change to datetime
```

In [112]:

```
all_manchester_traffic["Date_time"]= pd.to_datetime(all_manchester_traffic["Date_time"]
)
##Change our new column Date_time datatype to datetime
```

In [113]:

```
all_manchester_traffic.drop(['OutputDate','OutputTime'],axis =1 , inplace=True)
##We no longer need OutputDate and OutputTime. We will drop those 2 columns
```

In [114]:

```
all_manchester_traffic.head()
##We Check columns have been dropped
```

Out[114]:

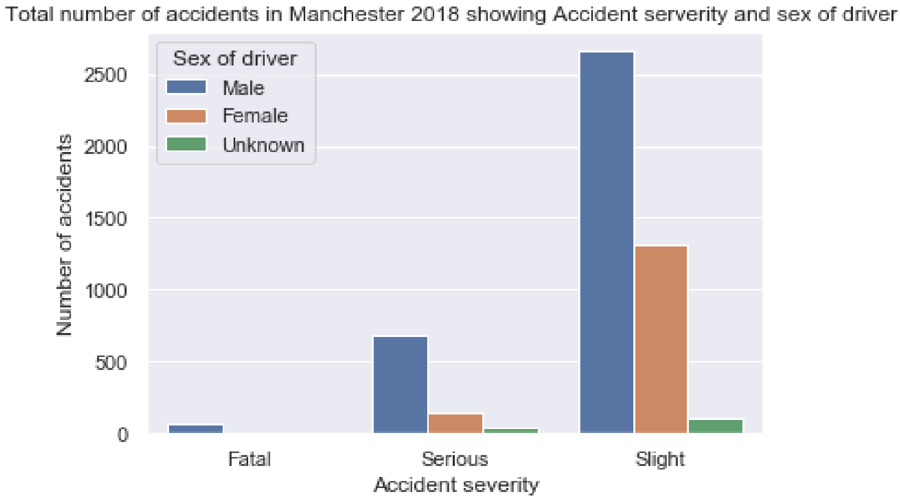
	Accident Index	Severity	NumberVehicles	NumberCasualties	Day	Easting	Northing	LocalAuthority	Road1Class	Carriagew
0	1.003910e+11	3	2	1	2	373324	407137	100	5	
1	1.013890e+11	3	1	1	2	381896	410649	101	5	
2	1.013900e+11	3	1	1	2	378030	407548	101	5	
3	1.013910e+11	3	2	3	2	381389	410964	101	5	
4	1.013910e+11	3	2	3	2	381389	410964	101	5	

5 rows × 48 columns

In [118]:

```
ax = sns.countplot(x="Severity", hue="SexOfDriver", data=all_manchester_traffic)
```

```python
ax.set_xticklabels(['Fatal','Serious','Slight'])
ax.legend(['Male','Female','Unknown'], title='Sex of driver')
ax.set_ylabel('Number of accidents')
ax.set_xlabel('Accident severity')
ax.set_title("Total number of accidents in Manchester 2018 showing Accident serverity and
sex of driver ");
ax.set_xticklabels(ax.get_xticklabels());
##Bar plot showing total number of accidents in Manchester 2018 showing Accident severity
and sex of driver
```



In [123]:

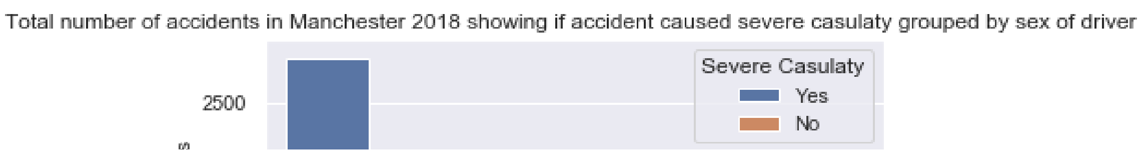
```python
all_manchester_traffic.groupby(['SexOfDriver', 'Severity']).agg({'Severity': 'count'})
##Check if graph is right
```

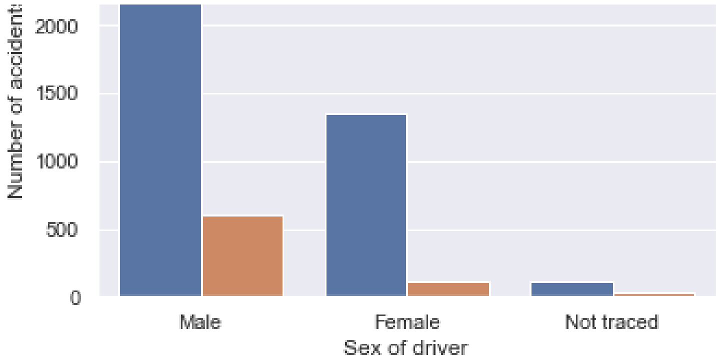
Out[123]:

		Severity
SexOfDriver	**Severity**	
1	**1**	71
	2	687
	3	2660
2	**1**	7
	2	141
	3	1313
3	**1**	4
	2	38
	3	105

In [122]:

```python
ax = sns.countplot(x="SexOfDriver", hue="SevereCasualty", data=all_manchester_traffic)
ax.set_xticklabels(['Male','Female','Not traced'])
ax.set_ylabel('Number of accidents')
ax.set_xlabel('Sex of driver')
ax.legend(['Yes','No'], title='Severe Casulaty')

ax.set_title("Total number of accidents in Manchester 2018 showing if accident caused sev
ere casulaty grouped by sex of driver ");
ax.set_xticklabels(ax.get_xticklabels());
##Bar plot showing total number of accidents in Manchester 2018 showing if accident cause
d severe casulaty grouped by sex of driver
```





In [125]:

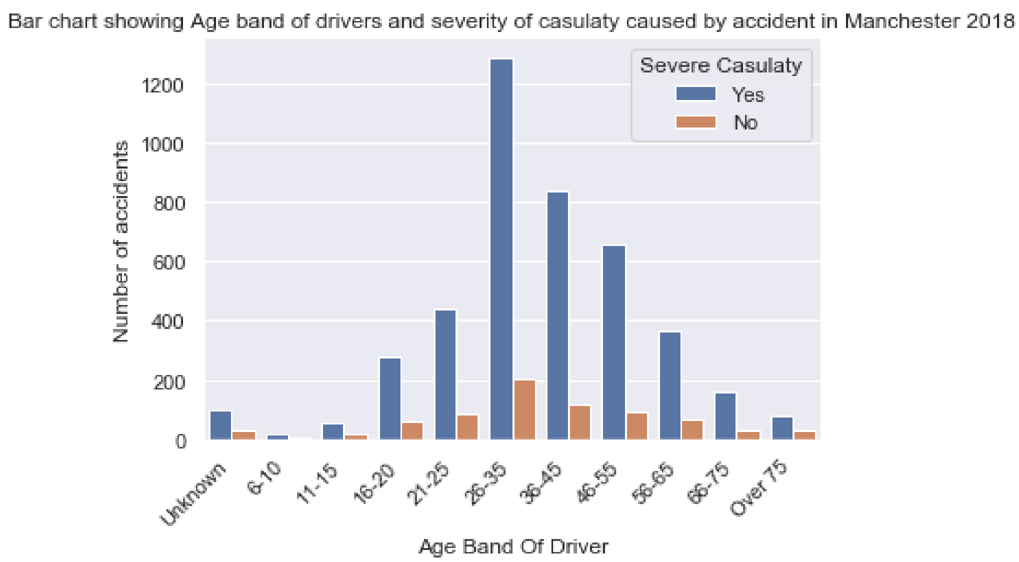
```python
all_manchester_traffic.groupby(['SexOfDriver', 'SevereCasualty']).agg({'SevereCasualty':
'count'})
##Check if graph is right
```

Out[125]:

		SevereCasualty
SexOfDriver	**SevereCasualty**	
1	0	2818
	1	600
2	0	1350
	1	111
3	0	110
	1	37

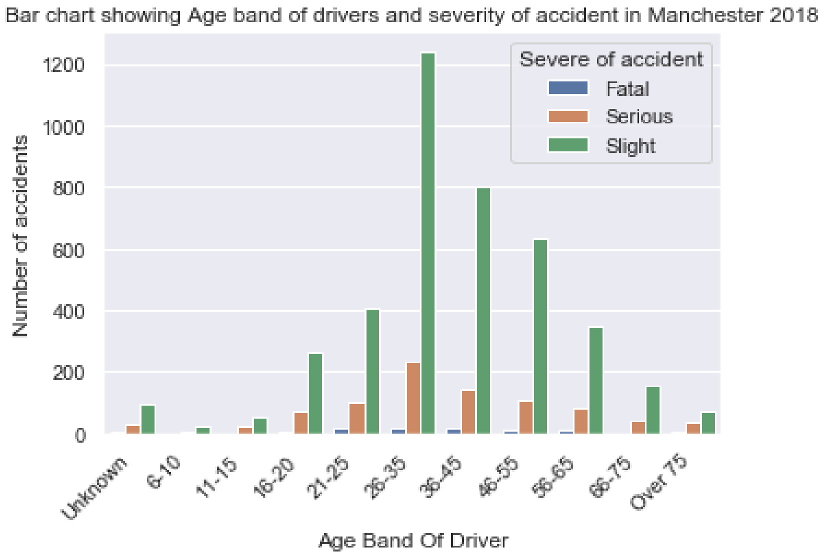
In [137]:

```python
ax = sns.countplot(x="AgeBandOfDriver", hue="SevereCasualty", data=all_manchester_traffi
c)
ax.set_xticklabels(['Unknown','6-10','11-15','16-20','21-25','26-35','36-45','46-55','56-
65','66-75','Over 75'])
ax.set_ylabel('Number of accidents')
ax.set_xlabel('Age Band Of Driver')
ax.legend(['Yes','No'], title='Severe Casulaty')
ax.set_title("Bar chart showing Age band of drivers and severity of casulaty caused by ac
cident in Manchester 2018");


ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right');
##Bar plot showing showing Age band of drivers and severity of casulaty caused by acciden
t in Manchester 2018
```



In [136]:

```
ax = sns.countplot(x="AgeBandOfDriver", hue="Severity", data=all_manchester_traffic)
ax.set_xticklabels(['Unknown','6-10','11-15','16-20','21-25','26-35','36-45','46-55','56-
65','66-75','Over 75'])
ax.set_ylabel('Number of accidents')
ax.set_xlabel('Age Band Of Driver')
ax.legend(['Fatal','Serious','Slight'], title='Severe of accident')

ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right');
ax.set_title("Bar chart showing Age band of drivers and severity of accident in Mancheste
r 2018");
##Bar plot showing Age band of drivers and severity of accident in Manchester 2018
```



In [128]:

```
all_manchester_traffic.groupby(['AgeBandOfDriver', 'Severity']).agg({'SevereCasualty': '
count'})
##check is graph is right
```

Out[128]:

		SevereCasualty
AgeBandOfDriver	**Severity**	
0	**1**	4
	2	30
	3	93
2	**2**	7
	3	20
3	**2**	22
	3	50
4	**1**	4
	2	71
	3	265
5	**1**	16
	2	101
	3	408
6	**1**	19
	2	231
	3	1241
7	**1**	15

AgeBandOfDriver	Severity	SevereCasualty
	2	144
	3	799
8	1	10
	2	107
	3	632
9	1	9
	2	80
	3	345
10	1	1
	2	40
	3	152
11	1	4
	2	33
	3	73

In [138]:

```
corr_matrix = all_manchester_traffic.corr()
corr_matrix["SevereCasualty"].sort_values(ascending=False)
##Correlaton between SevereCasualty and othe varibales
```

Out[138]:

```
SevereCasualty                    1.000000
PedLocation                       0.138026
Manoeuvre                         0.135199
PedMovement                       0.112919
CasualtyClass                     0.105654
Skidding                          0.101029
VehicleLocationOffRoad            0.053378
PedInjWork                        0.041298
CarriagewayType                   0.038581
LocalAuthority                    0.037585
Accident Index                    0.037504
CarriagewayHazard                 0.028210
ArtTowing                         0.027476
VehicleLeaveCWay                  0.025657
ForeignReg                        0.023476
AgeBandOfCasualty                 0.021274
HitObjectOffCWay                  0.017998
Road1Class                        0.016291
HitObjectOnCWay                   0.015556
Day                               0.011184
LightingCondition                 0.004018
JourneyPurpose                    0.003296
PedCrossingHumanControl           0.001542
SpecialConditions                -0.003943
RoadSurface                      -0.004112
BusPassenger                     -0.010070
Northing                         -0.011806
PedCrossingPhysicalFacilities    -0.017328
Easting                          -0.021322
JunctionLocation                 -0.025961
SpeedLimit                       -0.027066
AgeBandOfDriver                  -0.029114
Road2Class                       -0.039982
JunctionDetail                   -0.045132
JunctionControl                  -0.045696
WeatherCondition                 -0.050468
FirstPointImpact                 -0.077577
SexOfDriver                      -0.079677
CasualtyNumber                   -0.085271
NumberCasualties                 -0.103607
Sex                              -0.103830
NumberVehicles                   -0.105575
```

```
CarPassenger                     -0.120870
VehicleType                      -0.129476
VehicleReferenceNumber           -0.138030
PoliceReported                   -0.172512
Severity                         -0.838995
Name: SevereCasualty, dtype: float64
```

Machine learning

Baseline modeling

In [139]:

```
all_manchester_traffic['Accident Month'] = all_manchester_traffic['Date_time'].dt.month
```

In [140]:

```
all_manchester_traffic['Accident Hour'] =  all_manchester_traffic['Date_time'].dt.hour
```

In [141]:

```
all_manchester_traffic.drop(['Date_time'],axis =1 , inplace=True)

## Machine learning models does not accept data with datetime datatype so created two new
##columns called accident month and hour, datatype int and dropped Date_time column
```

In [142]:

```
all_manchester_traffic.head()
##check above changes
```

Out[142]:

	Accident Index	Severity	NumberVehicles	NumberCasualties	Day	Easting	Northing	LocalAuthority	Road1Class	Carriagew
0	1.003910e+11	3	2	1	2	373324	407137	100	5	
1	1.013890e+11	3	1	1	2	381896	410649	101	5	
2	1.013900e+11	3	1	1	2	378030	407548	101	5	
3	1.013910e+11	3	2	3	2	381389	410964	101	5	
4	1.013910e+11	3	2	3	2	381389	410964	101	5	

5 rows × 49 columns

In [143]:

```
manchester_traffic['SevereCasualty'].value_counts(normalize=True)

## This tells us that in the data around 14% of casulaties of accident is severe
```

Out[143]:

```
0    0.851174
1    0.148826
Name: SevereCasualty, dtype: float64
```

In [144]:

```
X = all_manchester_traffic.drop(['Accident Index','SevereCasualty'],axis =1)
y = all_manchester_traffic['SevereCasualty']
```

In [145]:

```
rs = np.random.RandomState(seed=20)
```

```
rs = np.random.RandomState(seed=20)
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size = 0.
4, shuffle=True, random_state=rs)
X_test, X_valid, y_test, y_valid = model_selection.train_test_split(X_test, y_test, test
_size = 0.5, shuffle=True, random_state=rs)

## create our three sets: train, test and validation
## use a set seed to make this work repeatable
```

In [146]:

```
print(X_test.shape)
print(X_valid.shape)
print(X_train.shape)

## just sanity check that proportions are correct
```

```
(1005, 47)
(1006, 47)
(3015, 47)
```

In [150]:

```
baseline_predictions = [0 for x in y_valid]
metrics.accuracy_score(y_valid, baseline_predictions)
## baseline predicition not using any model to see probaility that accident did not cause
severe casualties
```

Out[150]:

```
0.8558648111332008
```

In [151]:

```
baseline_predictions = [1 for x in y_valid]
metrics.accuracy_score(y_valid, baseline_predictions)
## baseline predicition not using any model to see probaility that accident caused severe
casualties
```

Out[151]:

```
0.14413518886679922
```

In [149]:

```
classifiers = [neighbors.KNeighborsClassifier(),
               naive_bayes.GaussianNB(),
               naive_bayes.MultinomialNB(),
               linear_model.LogisticRegression(solver='lbfgs', multi_class='ovr', max_it
er=9999),
               svm.LinearSVC(max_iter=9999),
               svm.SVC(kernel='rbf', gamma='auto', max_iter=9999)]
classifier_names = ['KNN','Gaussian NB','Multinomial NB','Logistic','Linear SVM','Non-Lin
ear SVM']
accuracies = []
for clf, name in zip(classifiers,classifier_names):
    clf.fit(X_train, y_train)
    predictions = clf.predict(X_valid)
    acc = metrics.accuracy_score(predictions, y_valid)
    accuracies.append(acc)
models = pd.DataFrame({'model':classifier_names, 'baseline':accuracies})
models
## table showing baseline/accuracy when using model KNN and Gaussian..We can see that Gau
ssian NB is more accurate than KNN
```

```
C:\Users\Chandni\Anaconda3\lib\site-packages\sklearn\svm\base.py:929: ConvergenceWarning:
Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

Out[149]:

	model	baseline

	model	baseline
0	KNN	0.842942
1	Gaussian NB	0.893638
2	Multinomial NB	0.510934
3	Logistic	0.852883
4	Linear SVM	0.855865
5	Non-Linear SVM	0.853877

Feature Scaling

In [152]:

```
scaler = preprocessing.MinMaxScaler()
X = scaler.fit_transform(X)

##Normalising the data to improve accuracy of model
```

In [153]:

```
rs = np.random.RandomState(seed=20)
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size = 0.
4, shuffle=True, random_state=rs)
X_test, X_valid, y_test, y_valid = model_selection.train_test_split(X_test, y_test, test
_size = 0.5, shuffle=True, random_state=rs)

## create our three sets: train, test and validation
## use a set seed to make this work repeatable
print(X_train.shape)
print(X_valid.shape)
print(X_test.shape)
```

```
(3015, 47)
(1006, 47)
(1005, 47)
```

In [155]:

```
classifiers = [neighbors.KNeighborsClassifier(),
               naive_bayes.GaussianNB(),
               naive_bayes.MultinomialNB(),
               linear_model.LogisticRegression(solver='lbfgs', multi_class='ovr', max_it
er=9999),
               svm.LinearSVC(max_iter=9999),
               svm.SVC(kernel='rbf', gamma='auto', max_iter=9999)]
classifier_names = ['KNN','Gaussian NB','Multinomial NB','Logistic','Linear SVM','Non-Lin
ear SVM']
accuracies = []
for clf, name in zip(classifiers,classifier_names):
    clf.fit(X_train, y_train)
    predictions = clf.predict(X_valid)
    acc = metrics.accuracy_score(predictions, y_valid)
    accuracies.append(acc)
models = pd.DataFrame({'model':classifier_names, 'scaled':accuracies})
models

## table showing baseline/accuracy when using model after scalling
##KNN and Gaussian..We can see that Gaussian NB is more accurate than KNN
```

Out[155]:

	model	scaled
0	KNN	0.895626
1	Gaussian NB	0.933400
2	Multinomial NB	0.856859
3	Logistic	0.979125

	model	scaled
4	Linear SVM	0.979125
5	Non-Linear SVM	0.977137

Feature Selection Method - Wrapper methods

In [156]:

```
columns = all_manchester_traffic.drop(['Accident Index','SevereCasualty'],axis =1).colum
ns.tolist()

num_to_keep = 5
best_features = []
for i in range(0, num_to_keep):
    best_acc = 0.0
    feature_to_add = ''
    for column in columns:
        feature_num = columns.index(column)
        if feature_num in best_features:
            continue
        features = best_features+[feature_num]
        clf = neighbors.KNeighborsClassifier()
        clf.fit(X_train[:,features], y_train)
        predictions = clf.predict(X_valid[:,features])
        acc = metrics.accuracy_score(predictions, y_valid)
        if acc > best_acc:
            best_acc = acc
            feature_to_add = feature_num
    best_features.append(feature_to_add)
print(best_features)
##using feature selection method getting best 5 features to use with KNN
```

```
[22, 0, 36, 12, 14]
```

In [157]:

```
subset = [22,0,36,12,14]
clf = neighbors.KNeighborsClassifier()
clf.fit(X_train[:,subset], y_train)
predictions = clf.predict(X_valid[:,subset])
acc = metrics.accuracy_score(predictions, y_valid)
acc
##KNN model give 0.984 accuracy when using the five subset given by wrapper method
```

Out[157]:

```
0.9840954274353877
```

In [158]:

```
columns = all_manchester_traffic.drop(['Accident Index','SevereCasualty'],axis =1).colum
ns.tolist()
num_to_keep = 5
best_features = []
for i in range(0, num_to_keep):
    best_acc = 0.0
    feature_to_add = ''
    for column in columns:
        feature_num = columns.index(column)
        if feature_num in best_features:
            continue
        features = best_features+[feature_num]
        clf = naive_bayes.GaussianNB()
        clf.fit(X_train[:,features], y_train)
        predictions = clf.predict(X_valid[:,features])
        acc = metrics.accuracy_score(predictions, y_valid)
        if acc > best_acc:
            best_acc = acc
            feature_to_add = feature_num
```

```
    best_features.append(feature_to_add)
print(best_features)
##using feature selection method getting best 5 features to use with GaussianNB
```

```
[0, 2, 42, 1, 24]
```

In [159]:

```
subset = [0,2,42,1,24]
clf = naive_bayes.GaussianNB()
clf.fit(X_train[:,subset], y_train)
predictions = clf.predict(X_valid[:,subset])
acc = metrics.accuracy_score(predictions, y_valid)
acc
##KNN model give 0.981 accuracy when using the five subset given by wrapper method
```

Out[159]:

```
0.9811133200795229
```

In [ ]:

```

```