Crime_Data _Analysis 2024 (python)

December 20, 2024

1 Business Case, Data Value and Narrative

The crime dataset that we chose has real data and is acquired from UCI Machine Learning repository where the title of the dataset is 'Crime and Communities'. The dataset has large number of numerical columns which was the main reason for choosing such a dataset rather than a categorical one because it makes is easy to apply algorithms and avoids the hassle of conversion of categorical data to numerical data. The crime attributes in the dataset that could be predicted by applying various machine learning algorithms as considered by the FBI are Rape, Murder, Larceny, Robbery, Assault, Burglaries, Autotheft and Arsons.

The other columns in the dataset include information about community names, county codes, community codes, percent of the population considered urban, age based population, gender based population, race based population and so on which are useful factors to predict crimes. We used some features as predictors from the dataset to train the different models and created a binary label to predict the Occurence of Crime based on the selected features. We also calculated various metrics like accuracy, f1 score, rms value, confusion matrix for the various clustering, regression and classification algorithms that we applied to the dataset.

The state column which was missing from the dataset was added to it based on the enrichment dataset viz cities.json available on the link below which contains mappings of latitude and longitude co-ordinates to their respective cities and states.

Base and Enrichment Dataset Location: https://www.kaggle.com/kkanda/analyzing-uci-crime-and-communities-dataset/data

[3]:
```
# Import necessary Libraries

import pandas as pd
import numpy as np
import math
import seaborn as sns
sns.set()
import matplotlib.pyplot as plt
import matplotlib.cm as cm
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
import scipy.cluster.hierarchy as sch

from sklearn.metrics import silhouette_score as sil, calinski_harabasz_score as␣
↪chs, silhouette_samples
from sklearn.metrics import accuracy_score,precision_score, recall_score,␣
↪confusion_matrix, silhouette_score
from sklearn.model_selection import cross_val_score
from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer
from sklearn.decomposition import PCA
```

1.1 Loading the dataset into a Pandas Dataframe

[5]:
```
df = pd.read_csv('crimedata.csv',sep= ',', encoding= "ISO-8859-1")
```

1.2 Data Cleaning

The dataset 'crimedata.csv' was first loaded into a Pandas Dataframe and then some columns were renamed following appropriate naming conventions to make the data readable. Many columns had the character '?' which was replaced by 0 as part of data cleaning. Also, checks were placed to ensure that there were no '?' values at all after cleaning the data.

[7]:
```
df=df.rename(columns = {'Êcommunityname':'Community Name'})
df = df.replace('?', '0')
df.head()
```

[7]:
```
           Community Name state  countyCode communityCode  fold  population  \
0  BerkeleyHeightstownship    NJ          39          5320     1       11980
1           Marpletownship    PA          45         47616     1       23123
2               Tigardcity    OR           0             0     1       29344
3         Gloversvillecity    NY          35         29443     1       16656
4              Bemidjicity    MN           7          5068     1       11245

   householdsize  racepctblack  racePctWhite  racePctAsian  …  burglaries  \
0           3.10          1.37         91.78          6.50  …          14
1           2.82          0.80         95.57          3.44  …          57
2           2.43          0.74         94.33          3.43  …         274
3           2.40          1.70         97.35          0.50  …         225
4           2.76          0.53         89.16          1.17  …          91

   burglPerPop  larcenies  larcPerPop  autoTheft  autoTheftPerPop  arsons  \
0       114.85        138     1132.08         16           131.26       2
1       242.37        376     1598.78         26           110.55       1
2       758.14       1797     4972.19        136            376.3      22
```

```
3      1301.78        716     4142.56          47             271.93        0
4       728.93       1060     8490.87          91             728.93        5

   arsonsPerPop  ViolentCrimesPerPop  nonViolPerPop
0         16.41                41.02        1394.59
1          4.25               127.56        1955.95
2         60.87               218.59        6167.51
3             0               306.64              0
4         40.05                    0        9988.79

[5 rows x 147 columns]
```

[9]:
```
df.loc[df['countyCode'] == '?']
df.loc[df['ViolentCrimesPerPop'] == '?']
```

[9]:
```
Empty DataFrame
Columns: [Community Name, state, countyCode, communityCode, fold, population,
householdsize, racepctblack, racePctWhite, racePctAsian, racePctHisp,
agePct12t21, agePct12t29, agePct16t24, agePct65up, numbUrban, pctUrban,
medIncome, pctWWage, pctWFarmSelf, pctWInvInc, pctWSocSec, pctWPubAsst,
pctWRetire, medFamInc, perCapInc, whitePerCap, blackPerCap, indianPerCap,
AsianPerCap, OtherPerCap, HispPerCap, NumUnderPov, PctPopUnderPov,
PctLess9thGrade, PctNotHSGrad, PctBSorMore, PctUnemployed, PctEmploy,
PctEmplManu, PctEmplProfServ, PctOccupManu, PctOccupMgmtProf, MalePctDivorce,
MalePctNevMarr, FemalePctDiv, TotalPctDiv, PersPerFam, PctFam2Par, PctKids2Par,
PctYoungKids2Par, PctTeen2Par, PctWorkMomYoungKids, PctWorkMom,
NumKidsBornNeverMar, PctKidsBornNeverMar, NumImmig, PctImmigRecent,
PctImmigRec5, PctImmigRec8, PctImmigRec10, PctRecentImmig, PctRecImmig5,
PctRecImmig8, PctRecImmig10, PctSpeakEnglOnly, PctNotSpeakEnglWell,
PctLargHouseFam, PctLargHouseOccup, PersPerOccupHous, PersPerOwnOccHous,
PersPerRentOccHous, PctPersOwnOccup, PctPersDenseHous, PctHousLess3BR, MedNumBR,
HousVacant, PctHousOccup, PctHousOwnOcc, PctVacantBoarded, PctVacMore6Mos,
MedYrHousBuilt, PctHousNoPhone, PctWOFullPlumb, OwnOccLowQuart, OwnOccMedVal,
OwnOccHiQuart, OwnOccQrange, RentLowQ, RentMedian, RentHighQ, RentQrange,
MedRent, MedRentPctHousInc, MedOwnCostPctInc, MedOwnCostPctIncNoMtg,
NumInShelters, NumStreet, PctForeignBorn, PctBornSameState, …]
Index: []

[0 rows x 147 columns]
```

1.3 Criteria Based Label Creation

After studying the dataset carefully, we found out that predicting the occurence of a crime could be a useful and valuable usecase. But, to do so we had to create a label named 'violent_crime_occurence' based on the mean value from the column Violent Crimes Per Population. After calculating the mean and comparing the mean values with the available values in the column 'ViolentCrimesPerPop', a decision 'yes' or '1' was made that a crime has occured if the value in the corresponding column was greater than the mean value or ‘no’ or ‘0’ if the value was less than the mean. Hence, a binary variable was created.

[10]:
```
violent_crimes = list(map(float, df.ViolentCrimesPerPop))
violent_crimes_mean = sum(violent_crimes)/len(violent_crimes)
violent_crimes_mean
```

[10]:
```
530.3040045146731
```

[11]:
```
df['mean_violent_crimes'] = violent_crimes_mean
df['violent_crime_occurence'] = np.
 ↪where(violent_crimes>=df['mean_violent_crimes'], '1', '0')
df.groupby('violent_crime_occurence').mean()
```

[11]:
```
                             fold    population  householdsize  racepctblack  \
violent_crime_occurence
0                        5.509979  32689.042670       2.707529      4.716284
1                        5.464567  92072.383202       2.706942     18.142375

                         racePctWhite  racePctAsian  racePctHisp  agePct12t21  \
violent_crime_occurence
0                           90.651535      2.434721     4.499188    14.372340
1                           71.258031      3.119226    14.530604    14.585984

                         agePct12t29  agePct16t24         …          \
violent_crime_occurence                                  …
0                          27.183125    13.781672         …
1                          28.525249    14.344055         …

                         PctSameHouse85  PctSameCity85  PctSameState85  \
violent_crime_occurence
0                             52.989732      77.301493       88.433861
1                             48.771535      77.620039       87.497874

                          LandArea      PopDens  PctUsePubTrans  \
violent_crime_occurence
0                        20.915279  2383.545630        2.755100
1                        39.823228  3547.116535        3.586522

                         LemasPctOfficDrugUn    murders  murdPerPop  \
violent_crime_occurence
0                                   0.433827   2.309704    2.775354
1                                   2.021929  18.166667   11.739829

                         mean_violent_crimes
violent_crime_occurence
0                                 530.304005
```

```
1                                      530.304005

[2 rows x 105 columns]
```

1.4 Data Slicing

In order to apply some clustering as well as classificatioj algorithms, the data needed to be sliced in order to better vizualise it and hence a temporary dataframe was created in order to do so which contained a slice of the actual data.

[13]:
```
df1 = df.iloc[:200]
df1.head(200)
```

[13]:
```
                  Community Name state countyCode communityCode  fold  \
0      BerkeleyHeightstownship    NJ         39          5320     1
1               Marpletownship    PA         45         47616     1
2                   Tigardcity    OR          0             0     1
3             Gloversvillecity    NY         35         29443     1
4                  Bemidjicity    MN          7          5068     1
5               Springfieldcity    MO          0             0     1
6                  Norwoodtown    MA         21         50250     1
7                 Andersoncity    IN          0             0     1
8                    Fargocity    ND         17         25700     1
9                    Wacocity    TX          0             0     1
10                Shermancity    TX          0             0     1
11               SanPablocity    CA          0             0     1
12          BowlingGreencity    KY          0             0     1
13              PineBluffcity    AR          0             0     1
14                 NewUlmcity    MN         15         46042     1
15              Maplewoodcity    MN        123         40382     1
16                Enfieldtown    CT          3         25990     1
17              Glendalecity    CA          0             0     1
18           Worthingtoncity    OH          0             0     1
19              Arlingtoncity    TX          0             0     1
20              Plymouthcity    MN         53         51730     1
21                NewYorkcity    NY          0             0     1
22                 Marinacity    CA          0             0     1
23               Lebanoncity    NH          9         41300     1
24              Rockledgecity    FL          0             0     1
25                 Rogerscity    AR          0             0     1
26              Bellairecity    TX          0             0     1
27               ElCajoncity    CA          0             0     1
28              MosesLakecity    WA          0             0     1
29           WestMemphiscity    AR          0             0     1
..                         …   …        …            …   …
170          BullheadCitycity    AZ          0             0     1
171            CulverCitycity    CA          0             0     1
```

172	Newbergcity	OR	0	0	1
173	Readingcity	PA	11	63624	1
174	Rustoncity	LA	0	0	1
175	Richmondcity	CA	0	0	1
176	Methuentown	MA	9	40675	1
177	Florissantcity	MO	0	0	1
178	WestFargocity	ND	17	84780	1
179	Douglascity	AZ	0	0	1
180	Selmacity	AL	0	0	1
181	Nortoncity	OH	153	57260	1
182	Missioncity	TX	0	0	1
183	Middletownship	NJ	9	45810	1
184	MountLaureltownship	NJ	5	49020	1
185	Greeleycity	CO	0	0	1
186	CostaMesacity	CA	0	0	1
187	LosAlamitoscity	CA	0	0	1
188	SouthOrangeVillagetownship	NJ	13	69274	1
189	RedBluffcity	CA	0	0	1
190	Plainfieldtown	IN	0	0	1
191	Fredericksburgcity	VA	630	29744	1
192	Colleyvillecity	TX	0	0	1
193	Pittsburgcity	CA	0	0	1
194	MissionViejocity	CA	0	0	1
195	LongBeachcity	CA	0	0	1
196	Duluthcity	MN	137	17000	1
197	Shelbycity	NC	0	0	1
198	Coronacity	CA	0	0	1
199	Beverlycity	MA	9	5595	1

	population	householdsize	racepctblack	racePctWhite	racePctAsian	\
0	11980	3.10	1.37	91.78	6.50	
1	23123	2.82	0.80	95.57	3.44	
2	29344	2.43	0.74	94.33	3.43	
3	16656	2.40	1.70	97.35	0.50	
4	11245	2.76	0.53	89.16	1.17	
5	140494	2.45	2.51	95.65	0.90	
6	28700	2.60	1.60	96.57	1.47	
7	59459	2.45	14.20	84.87	0.40	
8	74111	2.46	0.35	97.11	1.25	
9	103590	2.62	23.14	67.60	0.92	
10	31601	2.54	12.63	83.22	0.77	
11	25158	2.89	21.34	49.42	17.21	
12	40641	2.54	12.18	86.39	1.12	
13	57140	2.74	53.52	45.65	0.49	
14	13132	2.53	0.06	99.21	0.47	
15	30954	2.69	2.52	94.39	2.03	
16	45532	2.85	2.65	95.72	1.04	

17	180038	2.62	1.30	74.02	14.14
18	14869	2.67	2.28	94.74	2.67
19	261721	2.60	8.41	82.64	3.92
20	50889	2.77	1.61	95.66	2.04
21	7322564	2.60	28.71	52.26	7.00
22	26436	3.34	18.97	53.60	20.84
23	12183	2.36	0.41	97.55	1.55
24	16023	2.63	13.79	83.94	1.42
25	24692	2.54	0.06	97.72	0.77
26	13842	2.35	0.41	94.65	1.98
27	88693	2.70	2.92	87.36	2.82
28	11235	2.60	1.89	82.45	1.82
29	28259	2.86	42.15	56.94	0.52
..	…	…	…	…	…
170	21951	2.49	0.52	95.28	0.72
171	38793	2.40	10.38	69.22	12.04
172	13086	2.88	0.24	96.06	1.28
173	78380	2.50	9.71	78.64	1.42
174	20027	2.89	33.90	63.88	1.61
175	87425	2.67	43.76	36.18	11.83
176	39990	2.73	1.02	94.74	1.31
177	51206	2.67	4.06	95.02	0.52
178	12287	2.77	0.10	98.54	0.21
179	12822	3.20	1.07	71.46	0.47
180	23755	2.72	58.44	41.00	0.43
181	11475	2.80	1.04	98.41	0.38
182	28653	3.45	0.16	75.37	0.15
183	14771	2.76	13.09	84.93	1.09
184	30270	2.56	5.97	90.82	2.63
185	60536	2.67	0.67	89.10	1.00
186	96357	2.57	1.33	84.31	6.56
187	11676	2.84	3.00	84.87	7.18
188	16390	3.17	18.69	76.63	3.42
189	12363	2.57	0.56	91.22	1.06
190	10433	2.51	0.44	98.73	0.41
191	19027	2.55	21.63	76.04	1.08
192	12724	3.11	0.67	96.74	1.71
193	47564	3.04	17.58	58.60	12.18
194	72820	2.89	0.93	90.30	6.25
195	429433	2.70	13.68	58.38	13.57
196	85493	2.47	0.87	95.89	0.90
197	14669	2.41	42.50	57.03	0.23
198	76095	3.18	2.76	75.88	7.10
199	38195	2.58	0.86	97.63	1.02

	…	larcenies	larcPerPop	autoTheft	\
0	…	138	1132.08	16	

1	…	376	1598.78	26
2	…	1797	4972.19	136
3	…	716	4142.56	47
4	…	1060	8490.87	91
5	…	7690	5091.64	454
6	…	288	974.19	144
7	…	2250	3691.79	125
8	…	3149	3946.71	206
9	…	6121	5673.63	1070
10	…	1817	5654.8	151
11	…	1460	5404.4	430
12	…	1786	3895.99	148
13	…	2010	3424.02	517
14	…	283	2069.62	21
15	…	1618	4776.1	159
16	…	906	1987.5	232
17	…	4501	2509.23	1447
18	…	414	2739.73	18
19	…	11514	3938.78	2452
20	…	1468	2418.09	100
21	…	235132	3212.39	112464
22	…	547	3526.76	50
23	…	582	4694.68	27
24	…	721	4002.66	63
25	…	1215	3938.79	48
26	…	357	2388.92	45
27	…	4316	4634.68	889
28	…	975	7103.83	55
29	…	797	2860.32	238
..	…	…	…	…
170	…	1296	4607.34	185
171	…	1450	3672.47	469
172	…	527	3539.05	28
173	…	3362	4289.63	585
174	…	1519	7501.98	32
175	…	3786	4283.48	1460
176	…	1003	2435.23	659
177	…	1055	2034.87	108
178	…	344	2486.45	29
179	…	677	4759.23	141
180	…	1846	7430.07	131
181	…	241	2036.33	10
182	…	1562	3988.76	202
183	…	428	2830.69	15
184	…	383	1236.12	81
185	…	3352	5095.31	171
186	…	4918	4971.59	975

187	…	279	2310.94	101
188	…	498	2989.55	496
189	…	769	5759.87	75
190	…	328	2079.63	23
191	…	631	2819.1	32
192	…	229	1211.77	8
193	…	1353	2587.15	263
194	…	1539	1827.05	220
195	…	14108	3235.53	7626
196	…	3266	3852.28	268
197	…	937	5831.83	41
198	…	3099	3319.23	1334
199	…	589	1511.57	106

	autoTheftPerPop	arsons	arsonsPerPop	ViolentCrimesPerPop	\
0	131.26	2	16.41	41.02	
1	110.55	1	4.25	127.56	
2	376.3	22	60.87	218.59	
3	271.93	0	0	306.64	
4	728.93	5	40.05	0	
5	300.6	134	88.72	442.95	
6	487.1	17	57.5	226.63	
7	205.1	9	14.77	439.73	
8	258.18	8	10.03	115.31	
9	991.8	18	16.68	1544.24	
10	469.94	6	18.67	722.02	
11	1591.71	20	74.03	2605.96	
12	322.85	9	19.63	798.39	
13	880.7	46	78.36	1476.93	
14	153.58	1	7.31	0	
15	469.34	7	20.66	0	
16	508.94	7	15.36	89.94	
17	806.68	73	40.7	374.07	
18	119.12	4	26.47	112.5	
19	838.8	97	33.18	772.77	
20	164.72	62	102.13	0	
21	1536.49	4443	60.7	2097.71	
22	322.37	5	32.24	644.75	
23	217.79	4	32.27	145.2	
24	349.75	2	11.1	560.71	
25	155.61	6	19.45	204.23	
26	301.12	1	6.69	347.97	
27	954.64	31	33.29	894.51	
28	400.73	8	58.29	1143.9	
29	854.15	0	0	724.95	
..	…	…	…	…	
170	657.68	19	67.55	807	

171	1187.85	1	2.53	932.05
172	188.03	6	40.29	235.04
173	746.41	52	66.35	1195.53
174	158.04	4	19.76	1293.95
175	1651.85	112	126.72	3239.2
176	1600.02	8	19.42	364.19
177	208.31	11	21.22	179.38
178	209.61	6	43.37	43.37
179	991.21	0	0	161.69
180	527.27	0	0	3268.26
181	84.5	4	33.8	168.99
182	515.83	16	40.86	178.75
183	99.21	4	26.46	231.48
184	261.43	2	6.45	103.28
185	259.93	41	62.32	203.69
186	985.63	26	26.28	448.84
187	836.58	17	140.81	356.17
188	2977.55	0	0	672.35
189	561.76	9	67.41	981.2
190	145.83	2	12.68	190.21
191	142.97	12	53.61	446.77
192	42.33	1	5.29	31.75
193	502.9	18	34.42	868.12
194	261.18	35	41.55	225.56
195	1748.95	243	55.73	1631.98
196	316.11	14	16.51	0
197	255.18	7	43.57	1419.06
198	1428.8	23	24.63	519.47
199	272.03	4	10.27	87.26

	nonViolPerPop	mean_violent_crimes	violent_crime_occurence
0	1394.59	530.304005	0
1	1955.95	530.304005	0
2	6167.51	530.304005	0
3	0	530.304005	0
4	9988.79	530.304005	0
5	6867.42	530.304005	0
6	1890.88	530.304005	0
7	4909.26	530.304005	0
8	4747.58	530.304005	0
9	8903.93	530.304005	1
10	7599.9	530.304005	1
11	8839.53	530.304005	1
12	5508.05	530.304005	1
13	7371	530.304005	1
14	2720.49	530.304005	0
15	5933.23	530.304005	0

16	3130.42	530.304005	0
17	4246.34	530.304005	0
18	3540.47	530.304005	0
19	6171.23	530.304005	1
20	3131.33	530.304005	0
21	6164.95	530.304005	1
22	5338.49	530.304005	1
23	5461	530.304005	0
24	5618.16	530.304005	1
25	4869.19	530.304005	0
26	3693.79	530.304005	0
27	7091.62	530.304005	1
28	9143.9	530.304005	1
29	4421.48	530.304005	1
..	…	…	…
170	7341.18	530.304005	1
171	5891.14	530.304005	1
172	4378.48	530.304005	0
173	7043.06	530.304005	1
174	9116.95	530.304005	1
175	8497.95	530.304005	1
176	4681.09	530.304005	0
177	2659.8	530.304005	0
178	3259.85	530.304005	0
179	6854.13	530.304005	0
180	0	530.304005	1
181	2729.19	530.304005	0
182	5709.91	530.304005	0
183	4027.78	530.304005	0
184	2149.5	530.304005	0
185	6443.62	530.304005	0
186	7247.12	530.304005	0
187	4472.79	530.304005	0
188	7311.8	530.304005	1
189	7182.98	530.304005	1
190	2675.63	530.304005	0
191	3471.38	530.304005	0
192	1423.43	530.304005	0
193	4843.49	530.304005	1
194	2945.37	530.304005	0
195	6595.13	530.304005	1
196	5198.1	530.304005	0
197	8956.25	530.304005	1
198	6434.96	530.304005	0
199	3174.56	530.304005	0

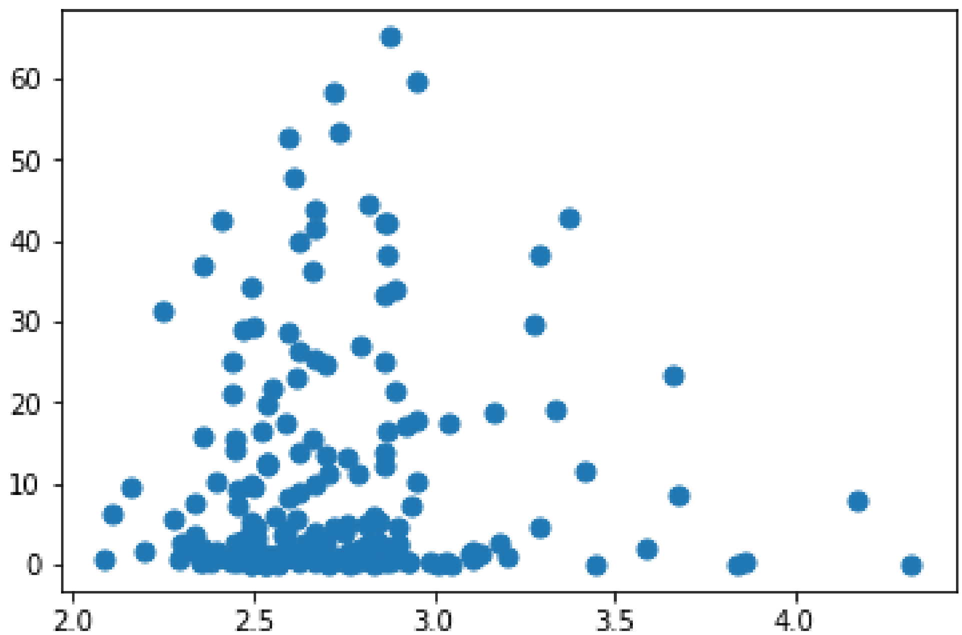
[200 rows x 149 columns]

1.5 Feature Selection for Clustering Algorithms

[15]:
```
features = ['householdsize', 'racepctblack']
X = df1[features].values
y = df1['violent_crime_occurence'].astype(float).values
```

1.6 Plotting the actual data to vizualize it

[16]:
```
plt.scatter(X[:, 0], X[:, 1], s=50);
```



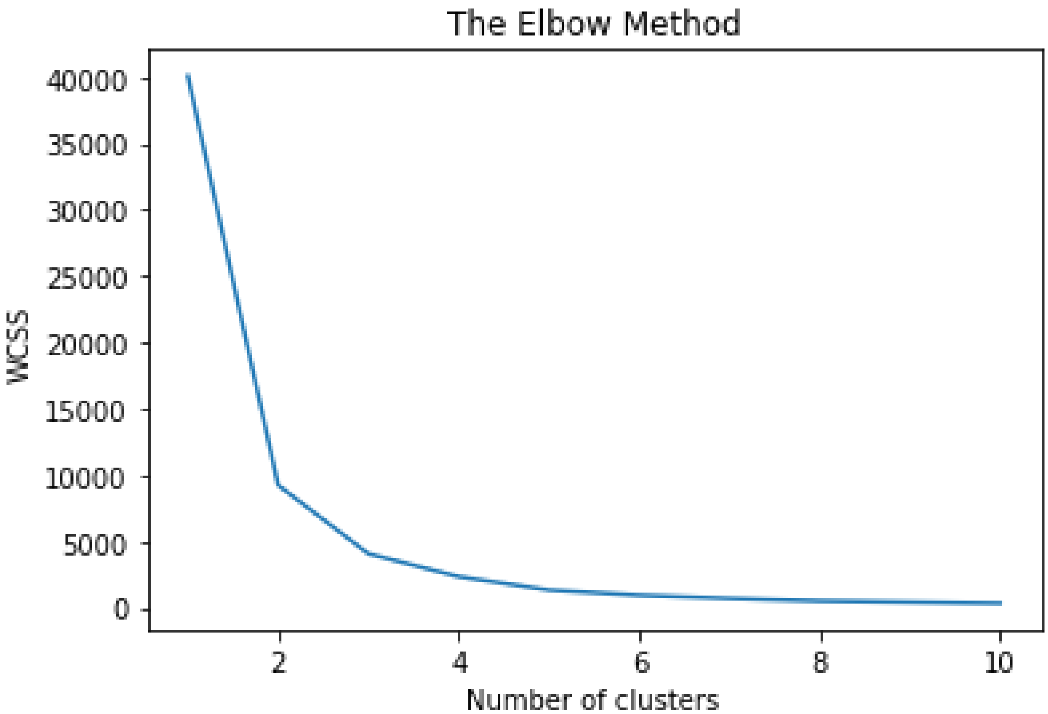
1.7 Splitting the data

[17]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)
```

1.8 Using the elbow method to find the optimal number of clusters

[19]:
```
from sklearn.cluster import KMeans
wcss = []
for i in range(1,11):
    kmeans =KMeans(n_clusters =i, init = 'k-means++', max_iter =300, n_init=10,␣
↪random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
```

```
plt.plot(range(1,11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



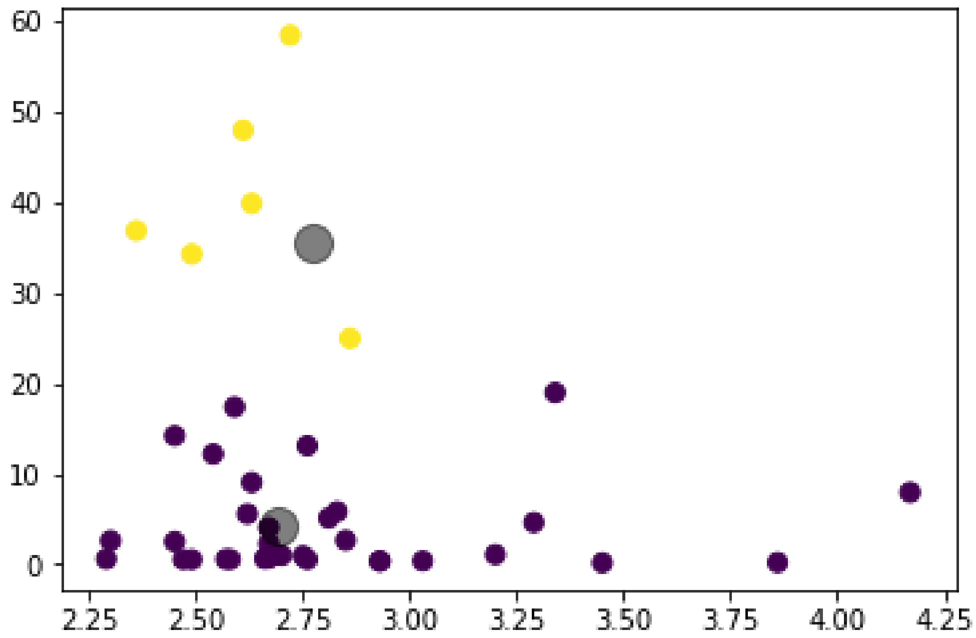
1.9 Applying kMeans Algorithm

[23]:
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=2)
kmeans.fit(X_train)
y_pred = kmeans.predict(X_test)
```

1.9.1 Vizualising the clusters

[24]:
```
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred, s=50, cmap='viridis')

centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5);
```



1.9.2 Metrics Calculation

[29]:
```
kmeans_accuracy = accuracy_score(y_test, y_pred)
kmeans_precison=precision_score(y_test,y_pred,average=None)
kmeans_recall=recall_score(y_test,y_pred,average=None)
kmeans_f1=f1_score(y_test,y_pred,average=None)
kmeans_confusion_matrix = confusion_matrix(y_test, y_pred)
```

[30]:
```
print("K-Means")
print("Scores")
print("Accuracy -->",kmeans_accuracy)
print("Precison -->",kmeans_precison)
print("Recall -->",kmeans_recall)
print("F1 -->",kmeans_f1)

print("Confusion Matrix")
print(kmeans_confusion_matrix)
```

```
K-Means
Scores
Accuracy --> 0.725
Precison --> [0.70588235 0.83333333]
Recall --> [0.96       0.33333333]
F1 --> [0.81355932 0.47619048]
Confusion Matrix
```

```
[[24  1]
 [10  5]]
```

1.10 Applying GMM

1.10.1 Data Cleaning

[31]:
```python
#converting huge ranges of data to average values
def extractSubstring(myStr):
    if "-" in myStr :
        lowVal,hiVal = myStr.split("-")

        lowVal = re.sub(r'[^\w]', '', lowVal)
        hiVal = re.sub(r'[^\w]', '', hiVal)

        lowVal = atof(lowVal)
        hiVal = atof(hiVal)
        lowV = float(lowVal)
        hiV = float(hiVal)
        average = (lowV + hiV)/2
    else:
        lowVal = myStr
        average = convert_to_float(lowVal)

    return average

def convert_to_float(input_str):
    return float(input_str.replace(",",""))

df['PolicReqPerOffic'] = df['PolicReqPerOffic'].apply(extractSubstring)
df['ViolentCrimesPerPop'] = df['ViolentCrimesPerPop'].apply(extractSubstring)
```

1.10.2 Feature selection

[33]:
```python
Features = ['PolicReqPerOffic','ViolentCrimesPerPop']
X = df[Features].values
```

1.10.3 Applying GMM

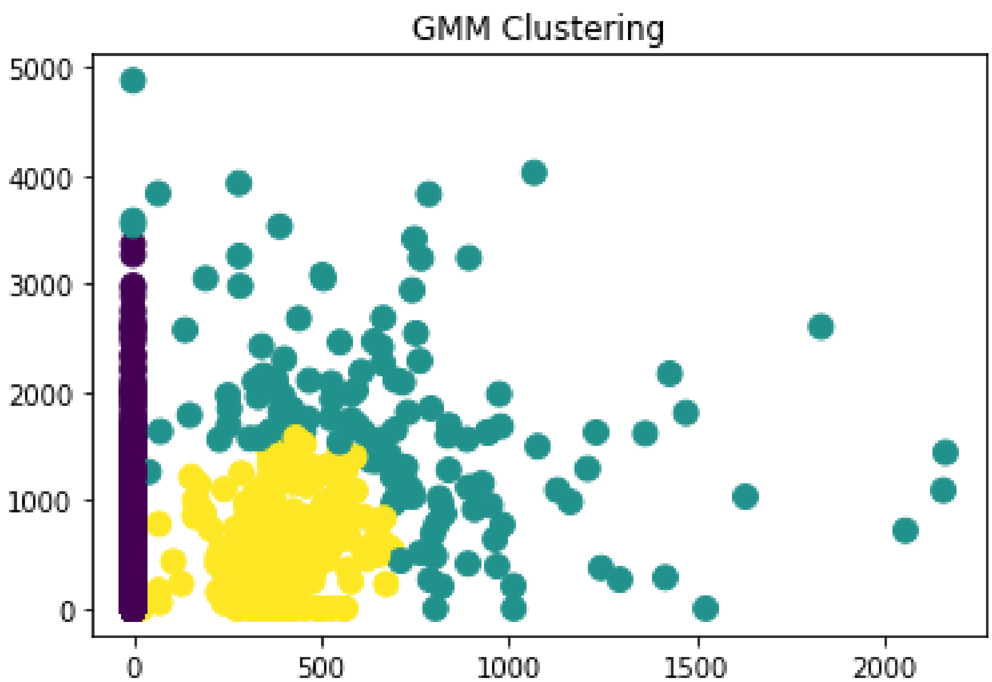
The intent is to cluster the dataset based on Violent crimes per population and for crimes occuring what number of police are required to control and handle the crime.

[36]:
```python
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3).fit(X)
labels = gmm.predict(X)
```

1.10.4 Vizualising the clusters

[37]:
```
plt.scatter(X[:, 0], X[:, 1], c=labels, s=80, cmap='viridis');
plt.title("GMM Clustering")
```

[37]: Text(0.5, 1.0, 'GMM Clustering')



1.11 Linear Regression

1.11.1 Feature Selection

[57]:
```
X1 = df[['PctUnemployed']].astype(int).values
y1 = df['ViolentCrimesPerPop'].astype(int).values
```

1.11.2 Splitting the data

[58]:
```
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.2,␣
↪random_state=0)
```

1.11.3 Fitting the model

[59]:
```
from sklearn import datasets, linear_model
regr = linear_model.LinearRegression()
regr.fit(X1_train, y1_train)
```

[59]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

1.11.4 Predicting the values

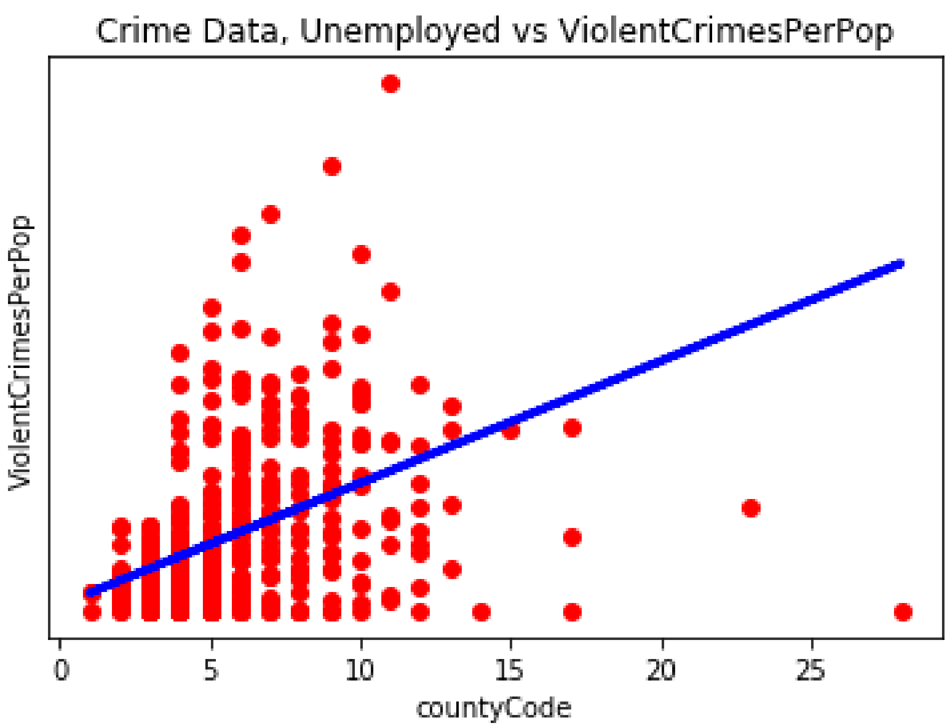
[60]:
```
y_pred = regr.predict(X1_test)
```

1.11.5 Vizualisation of plots

[61]:
```
plt.scatter(X1_test, y1_test,  color='red')
plt.plot(X1_test, y_pred, color='blue', linewidth=3)

plt.title('Crime Data, Unemployed vs ViolentCrimesPerPop')
plt.xlabel('countyCode')
plt.ylabel('ViolentCrimesPerPop')

plt.yticks(())
plt.show()
```



1.12 Logistic Regression

[63]:
```
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
from patsy import dmatrices
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import train_test_split
from sklearn.cross_validation import cross_val_score
```

```
/anaconda3/lib/python3.6/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)
```

1.12.1 Data Slicing

[209]:
```
df1 = df.iloc[:200]
```

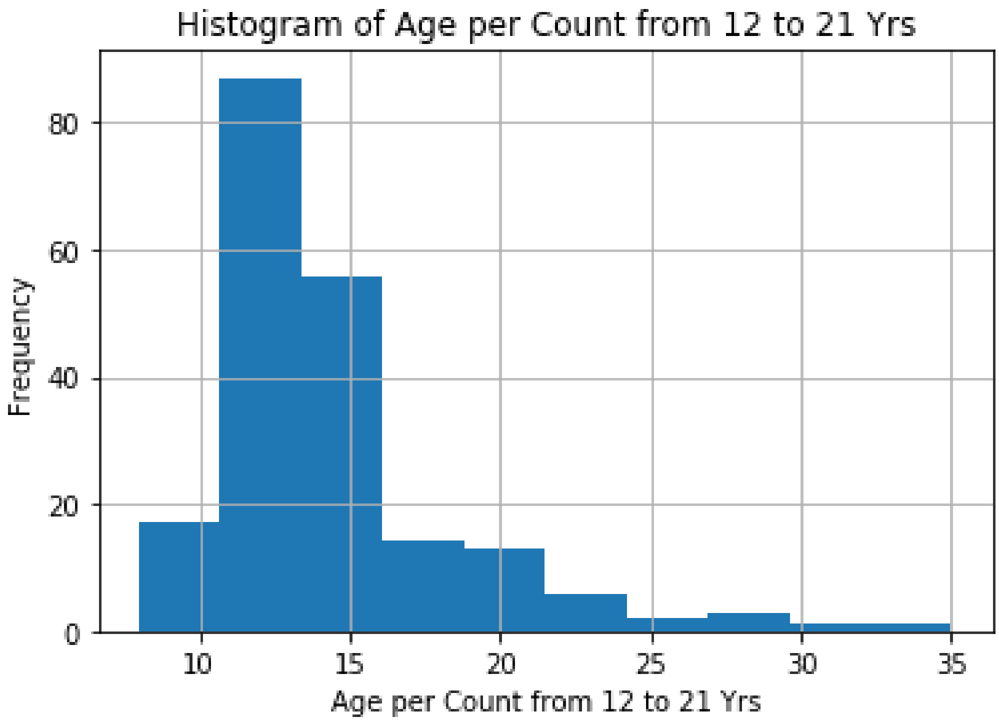
1.12.2 Visualizing Selected Features by plotting their Histograms

[64]:
```
age12t21 = df1['agePct12t21'].astype(int)
```

[65]:
```
age12t21.replace('?','0', inplace = True)
```

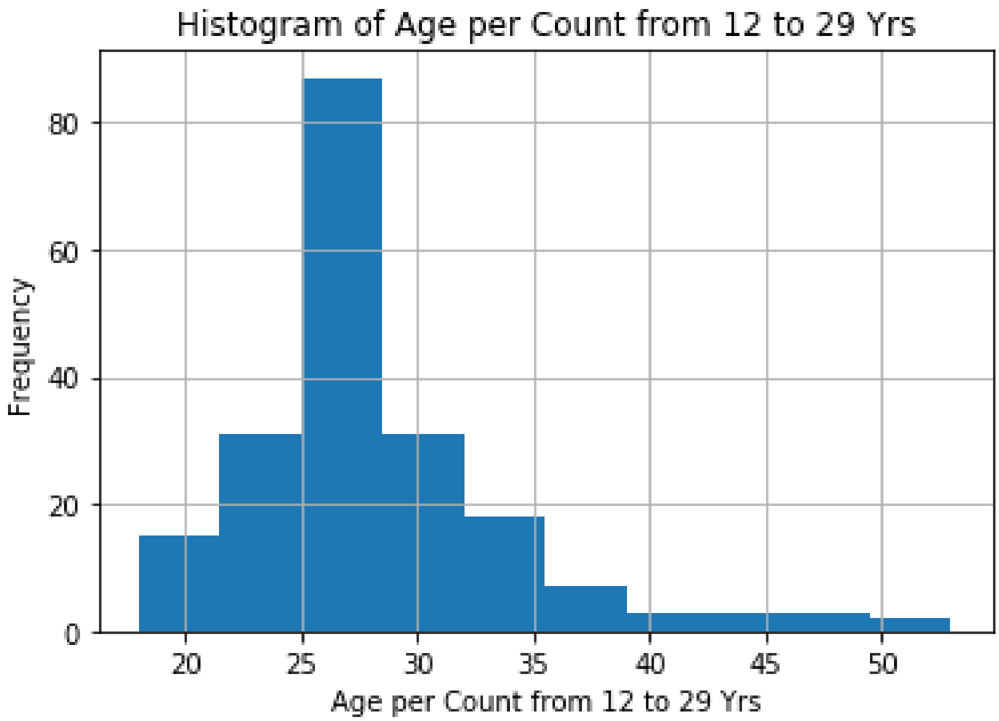
[66]:
```
%matplotlib inline
age12t21.hist()
plt.title('Histogram of Age per Count from 12 to 21 Yrs')
plt.xlabel('Age per Count from 12 to 21 Yrs')
plt.ylabel('Frequency')
plt.show
```

[66]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



[67]:
```
age12t29 = df1['agePct12t29'].astype(int)
%matplotlib inline
age12t29.hist()
plt.title('Histogram of Age per Count from 12 to 29 Yrs')
plt.xlabel('Age per Count from 12 to 29 Yrs')
plt.ylabel('Frequency')
plt.show
```

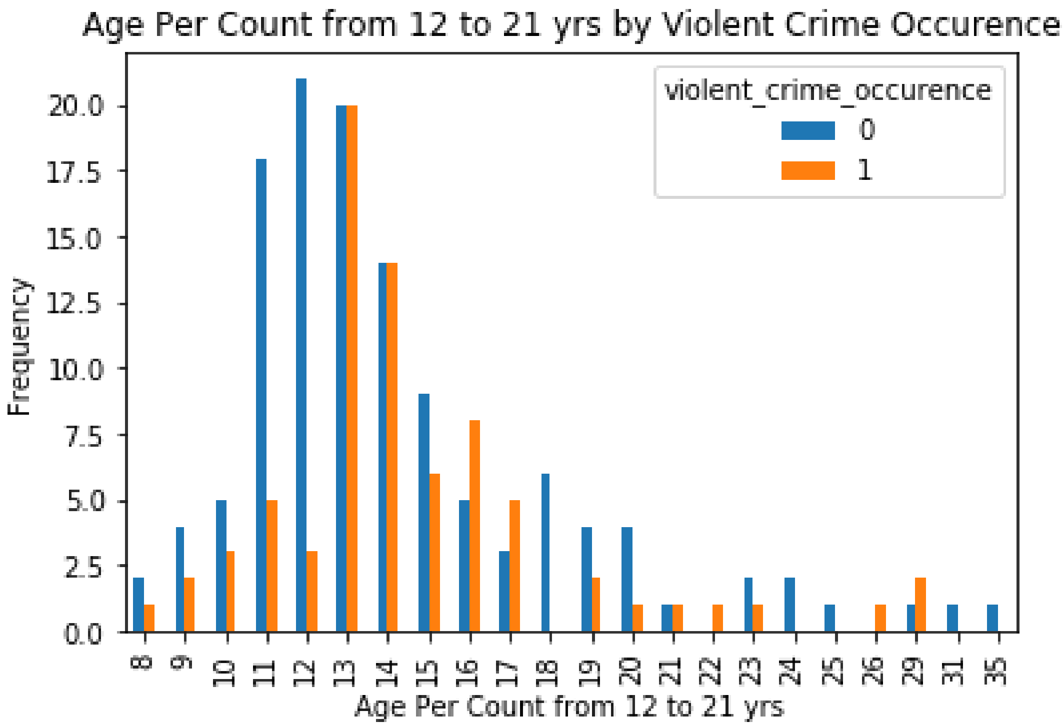
[67]: <function matplotlib.pyplot.show(*args, **kw)>



1.12.3 Plotting the features to analyze the label and its frequency

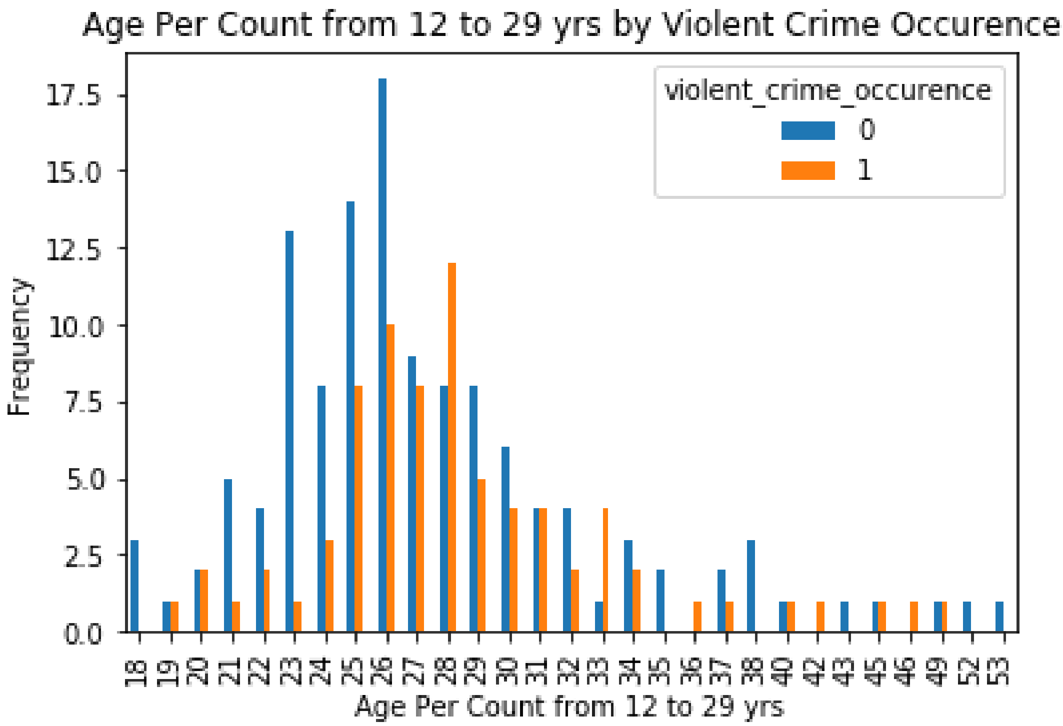
[72]:
```
pd.crosstab(age12t21, df1.violent_crime_occurence).plot(kind='bar')
plt.title('Age Per Count from 12 to 21 yrs by Violent Crime Occurence')
plt.xlabel('Age Per Count from 12 to 21 yrs')
plt.xticks(rotation='vertical')
plt.ylabel('Frequency')
plt.show
```

[72]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



[69]:
```
pd.crosstab(age12t29, df1.violent_crime_occurence).plot(kind='bar')
plt.title('Age Per Count from 12 to 29 yrs by Violent Crime Occurence')
plt.xlabel('Age Per Count from 12 to 29 yrs')
plt.xticks(rotation='vertical')
plt.ylabel('Frequency')
plt.show
```

[69]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



[70]:
```
X_LogReg= ['agePct12t21','agePct12t29','agePct16t24', 'agePct65up',␣
↪'PctUnemployed', 'murdPerPop', 'MalePctDivorce']
```

[71]:
```
y_LogReg = df1[['violent_crime_occurence']]
```

1.12.4 Training the Model

[73]:
```
X_train_LogReg, X_test_LogReg, y_train_LogReg, y_test_LogReg =␣
↪train_test_split(df1[X_LogReg], y_LogReg, test_size=0.2, random_state=0)
```

[74]:
```
logreg = LogisticRegression()
logreg.fit(X_train_LogReg, y_train_LogReg)
```

```
/anaconda3/lib/python3.6/site-packages/sklearn/utils/validation.py:578:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

[74]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

1.12.5 Metrics

[75]:
```
y_pred_LogReg = logreg.predict(X_test_LogReg)
print('Accuracy of logistic regression classifier on test set: {:.2f}'.
 ↪format(logreg.score(X_test_LogReg, y_test_LogReg)))
```

```
Accuracy of logistic regression classifier on test set: 0.78
```

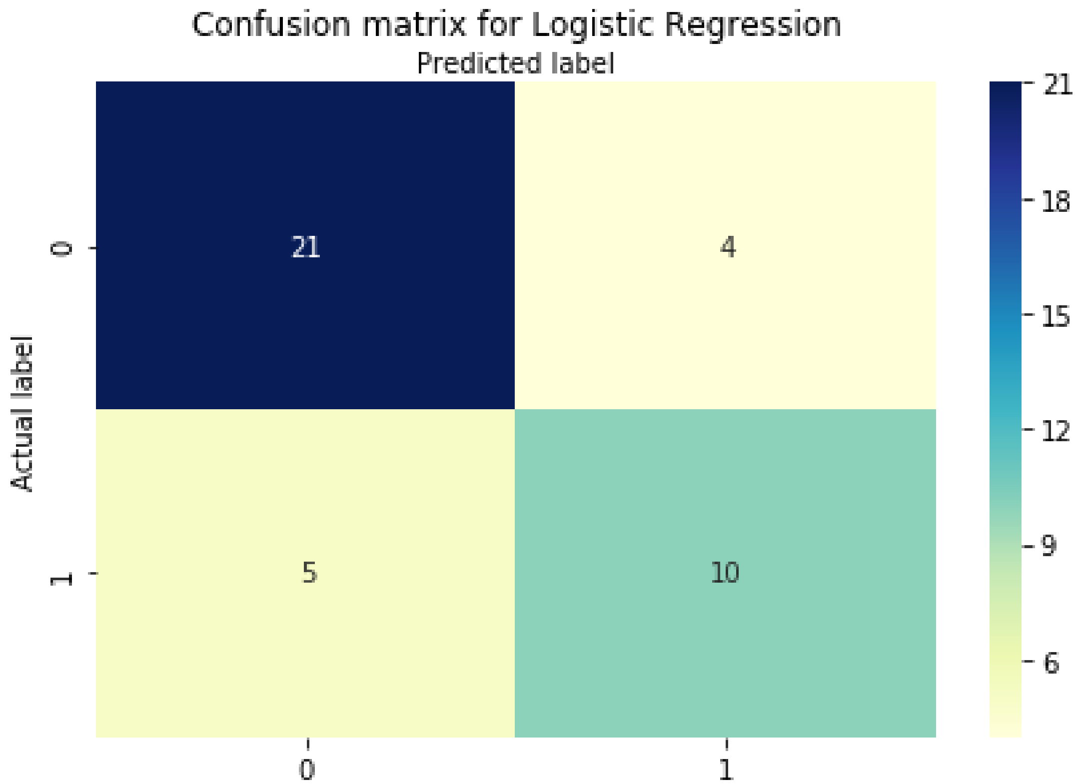
1.12.6 Creating the Confusion Matrix to make further conclusion

[76]:
```
cnf_matrix_LogitRegression = metrics.confusion_matrix(y_test_LogReg,␣
 ↪y_pred_LogReg)
cnf_matrix_LogitRegression

class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix_LogitRegression), annot=True, cmap="YlGnBu"␣
 ↪,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix for Logistic Regression', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
print("Model Accuracy for Logistic Regression:",metrics.
 ↪accuracy_score(y_test_LogReg, y_pred_LogReg))
```

```
Model Accuracy for Logistic Regression: 0.775
```



1.13 Decision Tree

Using Decision Tree Classifier from sklearn, we are trying to predict whether a crime has occured based on certain features or not and then calculating the accuracy of the decision tree classifier after training and testing the model.

[100]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
```

[102]:
```
#X = balance_data.values[:,␣
↪[5,6,17,37,47,50,56,96,129,131,133,135,137,139,141,143,145]]
df =␣
↪df[['population','householdsize','medIncome','PctUnemployed','PolicReqPerOffic','murders','
df = df
X_DecisionTree = df.drop('violent_crime_occurence', axis=1)
Y_DecisionTree = df['violent_crime_occurence']
```

[103]:

```
from sklearn.model_selection import train_test_split
X_train_DecisionTree, X_test_DecisionTree, Y_train_DecisionTree,␣
↪Y_test_DecisionTree = train_test_split(X_DecisionTree, Y_DecisionTree,␣
↪random_state=1)
```

1.13.1 Implementing Decision Tree Classifier

[104]:
```
clf_gini = DecisionTreeClassifier(criterion = "gini", random_state =␣
↪100,max_depth=20, min_samples_split=9, min_samples_leaf=6)
clf_gini
```

[104]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=20,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=6, min_samples_split=9,
            min_weight_fraction_leaf=0.0, presort=False, random_state=100,
            splitter='best')
```

[105]:
```
clf_gini.fit(X_train_DecisionTree, Y_train_DecisionTree)
```

[105]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=20,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=6, min_samples_split=9,
            min_weight_fraction_leaf=0.0, presort=False, random_state=100,
            splitter='best')
```

[106]:
```
Y_Pred_DecisionTree = clf_gini.predict(X_test_DecisionTree)
Y_Pred_DecisionTree
```

[106]:
```
array(['0', '1', '1', '0', '1', '0', '1', '0', '1', '1', '1', '1', '1',
       '0', '0', '0', '0', '0', '1', '0', '1', '0', '0', '0', '0', '0',
       '1', '0', '1', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
       '0', '1', '1', '1', '1', '0', '0', '1', '0', '0', '0', '0', '1',
       '0', '0', '1', '0', '1', '0', '0', '0', '0', '1', '0', '1', '0',
       '0', '0', '1', '1', '1', '1', '1', '0', '0', '0', '1', '1', '0',
       '1', '0', '0', '0', '1', '0', '1', '0', '1', '0', '0', '0', '0',
       '0', '0', '0', '1', '1', '0', '1', '0', '0', '0', '0', '0', '1',
       '0', '0', '0', '0', '1', '0', '1', '1', '0', '0', '0', '1', '0',
       '0', '0', '0', '0', '0', '0', '1', '1', '0', '1', '1', '0', '0',
       '1', '0', '0', '0', '0', '0', '0', '0', '0', '1', '1', '1', '1',
       '0', '1', '0', '1', '0', '1', '0', '1', '1', '0', '0', '0', '1',
       '0', '1', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '1',
       '0', '1', '0', '0', '0', '0', '0', '0', '0', '1', '0', '0', '1',
       '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '0', '0', '0',
       '0', '0', '1', '0', '0', '0', '0', '1', '0', '0', '1', '0', '1',
       '0', '0', '0', '1', '1', '0', '1', '0', '1', '0', '0', '0', '1',
```

```
'1', '0', '1', '1', '0', '0', '0', '0', '0', '0', '1', '0', '1',
'0', '0', '0', '1', '0', '0', '0', '0', '1', '0', '0', '1', '0',
'1', '0', '0', '0', '0', '0', '0', '0', '1', '0', '0', '0', '0',
'1', '1', '0', '0', '1', '0', '1', '1', '1', '0', '0', '0', '1',
'0', '0', '1', '0', '0', '0', '0', '0', '1', '1', '0', '0', '0',
'1', '1', '0', '0', '0', '1', '1', '0', '0', '0', '0', '0', '0',
'0', '0', '0', '0', '1', '1', '0', '0', '0', '0', '0', '0', '1',
'1', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
'1', '1', '0', '0', '0', '0', '1', '1', '0', '1', '0', '1', '0',
'0', '1', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '0',
'1', '1', '1', '1', '0', '0', '1', '0', '0', '1', '0', '0', '1',
'1', '0', '0', '0', '1', '0', '1', '1', '0', '0', '0', '0', '1',
'0', '0', '1', '1', '0', '1', '1', '0', '0', '1', '0', '0', '0',
'0', '0', '1', '0', '0', '0', '1', '0', '0', '0', '0', '0', '1',
'1', '1', '1', '0', '0', '0', '0', '1', '0', '0', '0', '1', '0',
'0', '0', '0', '0', '1', '0', '1', '0', '1', '0', '0', '1', '1',
'0', '0', '1', '0', '0', '0', '1', '1', '0', '0', '0', '0', '0',
'1', '1', '0', '0', '0', '0', '0', '1', '0', '0', '1', '1', '0',
'0', '1', '0', '0', '0', '0', '1', '1', '0', '0', '0', '0', '0',
'0', '1', '1', '1', '0', '0', '0', '1', '1', '0', '1', '1', '1',
'0', '0', '0', '1', '0', '0', '0', '0', '1', '0', '0', '0', '0',
'1', '0', '0', '1', '1', '0', '0', '1', '0', '1', '0', '1', '0',
'0', '0', '1', '0', '0', '0', '1', '0', '1', '0', '1', '0', '0',
'0', '0', '1', '0', '1', '0', '1', '0', '0', '0', '1', '0', '0',
'0', '1', '0', '0', '0', '1', '1', '1', '0', '1', '0', '0', '0',
'0', '1', '0', '0', '1', '1', '0', '0'], dtype=object)
```

1.13.2 Metrics

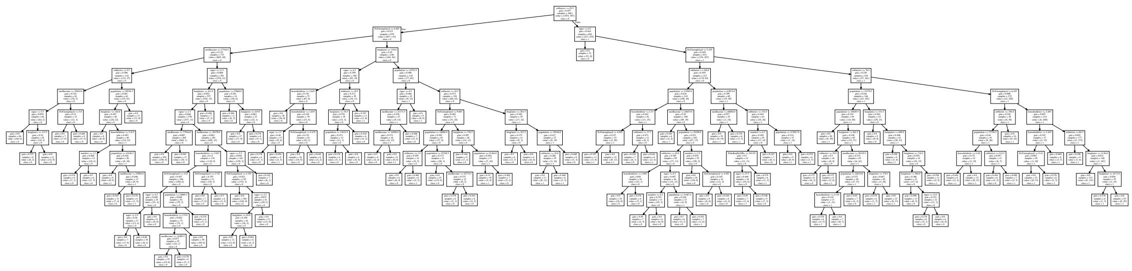
[17]:
```
ac=accuracy_score(Y_test_DecisionTree, Y_Pred_DecisionTree)*100
ac
```

[17]:
```
85.5595667870036
```

1.13.3 Plotting the tree

[107]:
```
import graphviz
dot_data = tree.export_graphviz(clf_gini, out_file=None,␣
 ↪feature_names=X_DecisionTree.columns, class_names=['0','1'])
graph = graphviz.Source(dot_data)
graph.render("crime")
graph
```

[107]:

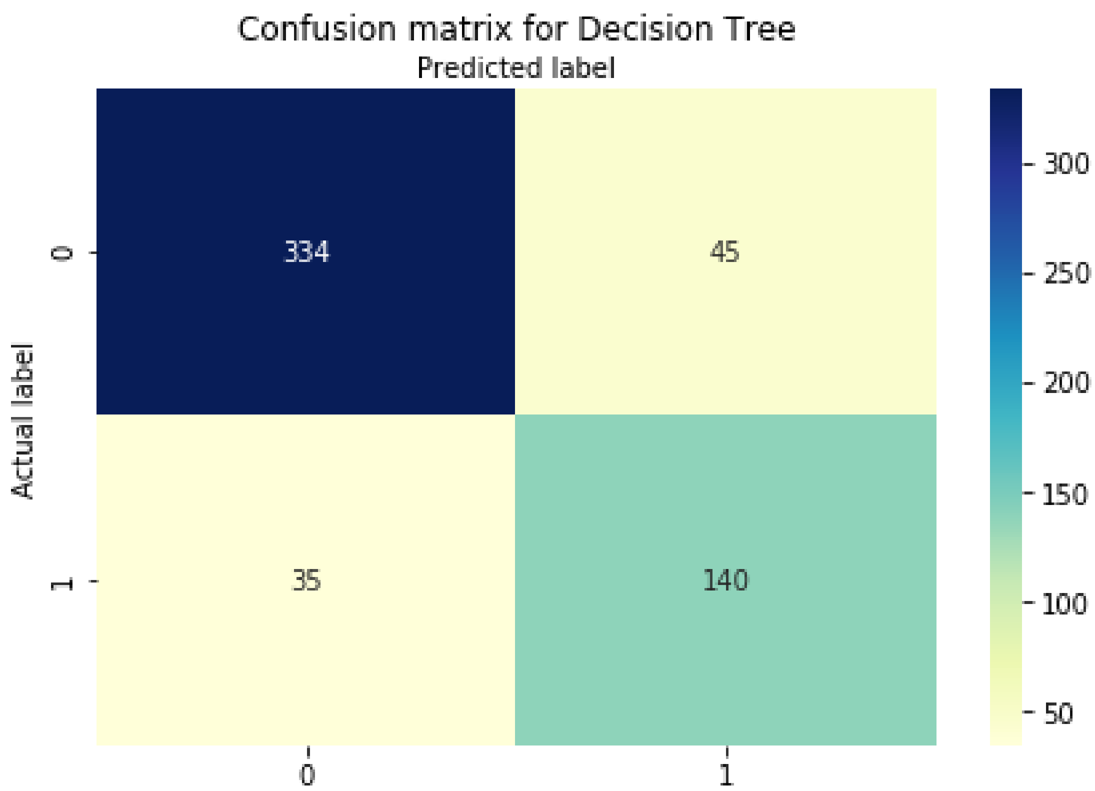


1.13.4 Confusion Matrix

[108]:
```
# For Decision Tree
cnf_matrix_DecisionTree = metrics.confusion_matrix(Y_test_DecisionTree,␣
 ↪Y_Pred_DecisionTree)
cnf_matrix_DecisionTree
# name  of classes
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix_DecisionTree), annot=True, cmap="YlGnBu"␣
 ↪,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix for Decision Tree', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
print("Model Accuracy for Random Forest:",metrics.
 ↪accuracy_score(Y_test_DecisionTree, Y_Pred_DecisionTree))
```

```
Model Accuracy for Random Forest: 0.855595667870036
```



1.14 Gaussian Naive Bayes Classifier

1.14.1 Label Creation

[113]:
```
murder = list(map(float, df.murdPerPop))
murders_mean = sum(murder)/len(murder)
murders_mean
```

[113]: 5.8592957110609545

[114]:
```
df['mean_murder'] = murders_mean
df['murder_occurence'] = np.where(murder>=df['mean_murder'], 'yes', 'no')
df.groupby('murder_occurence').mean()
```

[114]:
```
                      fold    population  householdsize  racepctblack  \
murder_occurence
no                5.511692  29966.262724       2.708425      4.346499
yes               5.461235  97352.679369       2.705230     18.866544

                  racePctWhite  racePctAsian  racePctHisp  agePct12t21  \
murder_occurence
no                   90.388755      2.568191     5.626314    14.318920
```

```
yes                    71.734625      2.865112    12.390250    14.688331

                    agePct12t29  agePct16t24      …        PctSameHouse85  \
murder_occurence                                  …
no                    27.257166    13.797407      …             52.455076
yes                   28.385545    14.314731      …             49.787530

                    PctSameCity85  PctSameState85   LandArea      PopDens  \
murder_occurence
no                      76.972270       88.175915  20.060523  2497.271045
yes                     78.249488       87.989488  41.481209  3331.356767

                    PctUsePubTrans  LemasPctOfficDrugUn    murders  murdPerPop  \
murder_occurence
no                        2.755475             0.460784   0.545392    1.055365
yes                       3.586899             1.972510  21.558476   15.037898

                    mean_murder
murder_occurence
no                     5.859296
yes                    5.859296

[2 rows x 105 columns]
```

1.14.2 Data Slicing

[115]:
```python
df1 = df.iloc[:700]
```

1.14.3 Applying Gaussian NB classifier

[116]:
```python
X_NaiveBayes= ['agePct12t21','agePct12t29','agePct16t24',␣
↪'agePct65up','PctUnemployed']
Y_NaiveBayes = df1[['murder_occurence']]
```

[117]:
```python
X_train_NaiveBayes, X_test_NaiveBayes, Y_train_NaiveBayes, Y_test_NaiveBayes =␣
↪train_test_split(df1[X_NaiveBayes], Y_NaiveBayes, test_size=0.2,␣
↪random_state=0)
```

[118]:
```python
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
```

[119]:
```python
model.fit(X_train_NaiveBayes, Y_train_NaiveBayes)
```

```
/anaconda3/lib/python3.6/site-packages/sklearn/utils/validation.py:578:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
```

```
ravel().
  y = column_or_1d(y, warn=True)
```

[119]:
```
GaussianNB(priors=None)
```

1.14.4 Model Accuracy

[120]:
```
Y_Pred_NaiveBayes = model.predict(X_test_NaiveBayes)
print('Accuracy of Gaussian Naive Bayes classifier on test set: {:.2f}'.
 ↪format(model.score(X_test_NaiveBayes, Y_test_NaiveBayes)))
```

```
Accuracy of Gaussian Naive Bayes classifier on test set: 0.79
```

1.14.5 Corelation matrix showing the features

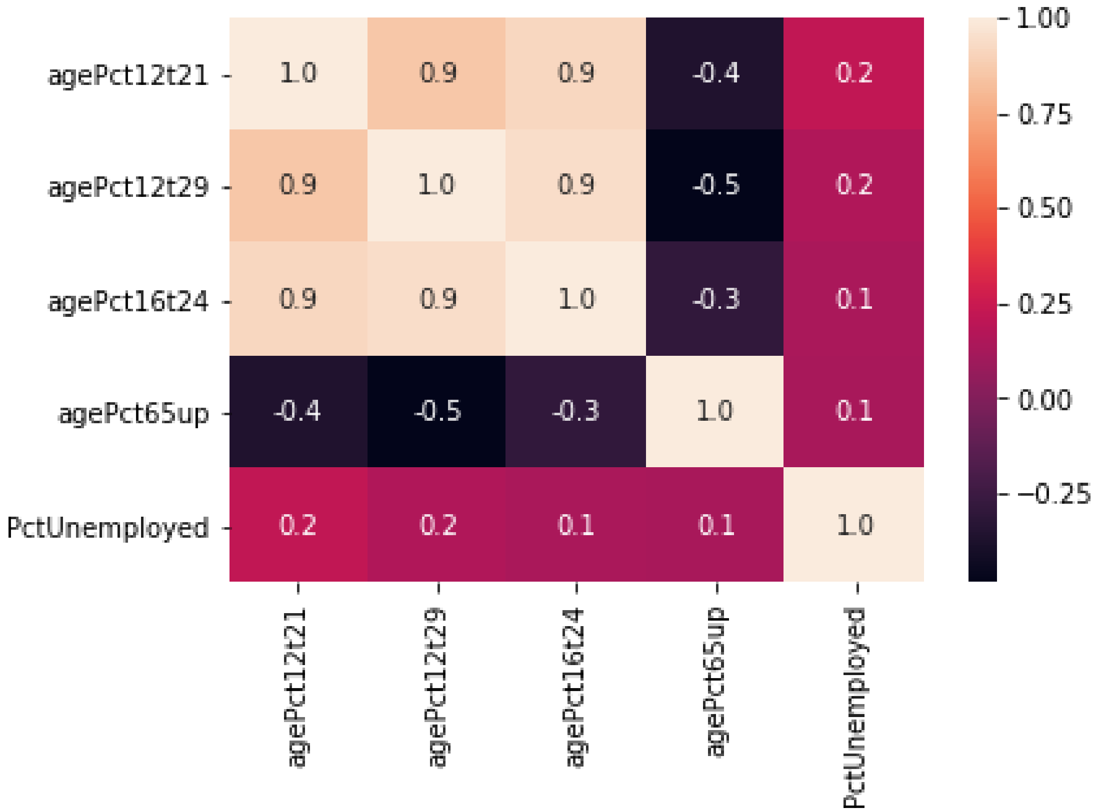
[121]:
```
df1[X_NaiveBayes].corr()
```

[121]:
```
              agePct12t21  agePct12t29  agePct16t24  agePct65up  \
agePct12t21      1.000000     0.857899     0.923780   -0.362218
agePct12t29      0.857899     1.000000     0.946554   -0.484518
agePct16t24      0.923780     0.946554     1.000000   -0.288004
agePct65up      -0.362218    -0.484518    -0.288004    1.000000
PctUnemployed    0.219374     0.159712     0.141656    0.131281

              PctUnemployed
agePct12t21        0.219374
agePct12t29        0.159712
agePct16t24        0.141656
agePct65up         0.131281
PctUnemployed      1.000000
```

1.14.6 Corelation Heatmap for better visualization

[122]:
```
import seaborn as sns
sns.heatmap(df1[X_NaiveBayes].corr(), annot=True, fmt=".1f")
plt.show()
```



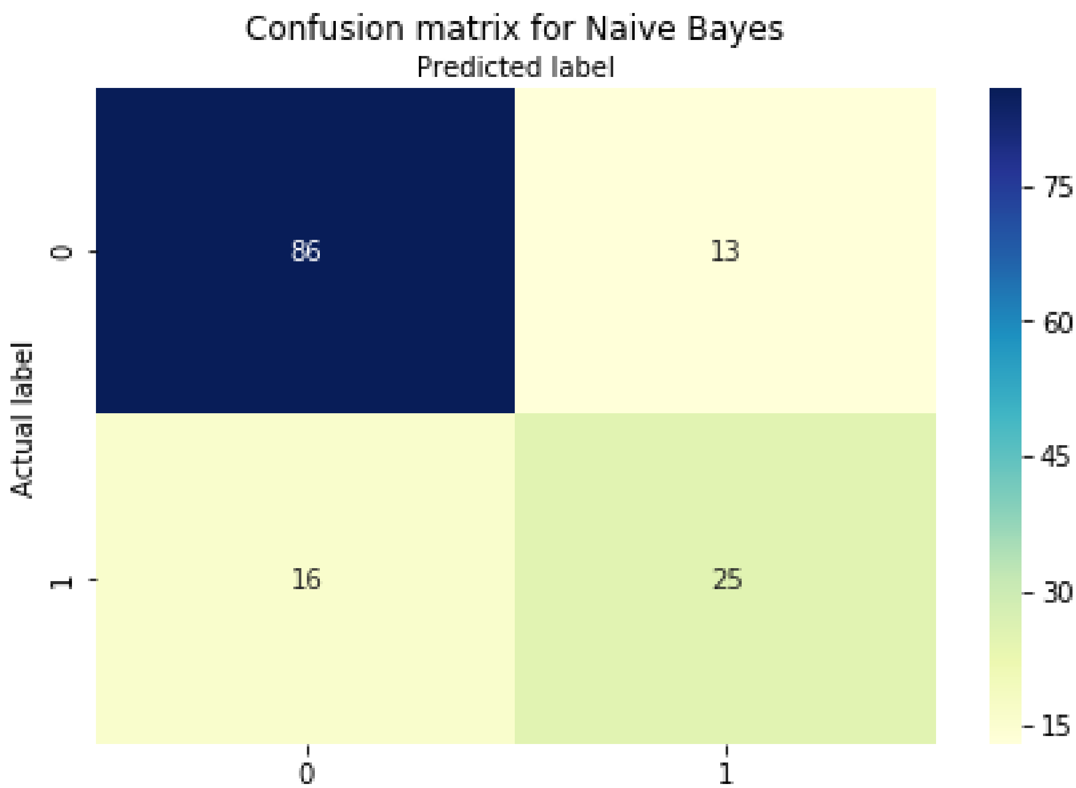
1.15 Confusion Matrix

[123]:
```
cnf_matrix_NaiveBayes = metrics.confusion_matrix(Y_test_NaiveBayes,␣
↪Y_Pred_NaiveBayes)
cnf_matrix_NaiveBayes
# name  of classes
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix_NaiveBayes), annot=True, cmap="YlGnBu"␣
↪,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix for Naive Bayes', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

```
print("Model Accuracy for Random Forest:",metrics.
 ↪accuracy_score(Y_test_NaiveBayes, Y_Pred_NaiveBayes))
```

```
Model Accuracy for Random Forest: 0.7928571428571428
```



1.16 Random Forest Classifier

1.16.1 Label Creation

[124]:
```
df['mean_violent_crimes'] = violent_crimes_mean
df['violent_crime_occurence'] = np.
 ↪where(violent_crimes>=df['mean_violent_crimes'], '1', '0')
df.groupby('violent_crime_occurence').mean()
```

[124]:
```
                             fold    population  householdsize  racepctblack  \
violent_crime_occurence
0                        5.509979  32689.042670       2.707529      4.716284
1                        5.464567  92072.383202       2.706942     18.142375

                         racePctWhite  racePctAsian  racePctHisp  agePct12t21  \
violent_crime_occurence
0                           90.651535      2.434721     4.499188    14.372340
```

```
1                            71.258031      3.119226    14.530604    14.585984

                         agePct12t29  agePct16t24          …           \
violent_crime_occurence                                    …
0                          27.183125    13.781672          …
1                          28.525249    14.344055          …

                         PctSameCity85  PctSameState85   LandArea  \
violent_crime_occurence
0                            77.301493       88.433861  20.915279
1                            77.620039       87.497874  39.823228

                             PopDens  PctUsePubTrans  LemasPctOfficDrugUn  \
violent_crime_occurence
0                        2383.545630        2.755100             0.433827
1                        3547.116535        3.586522             2.021929

                           murders  murdPerPop  mean_murder  \
violent_crime_occurence
0                         2.309704    2.775354     5.859296
1                        18.166667   11.739829     5.859296

                         mean_violent_crimes
violent_crime_occurence
0                                 530.304005
1                                 530.304005

[2 rows x 106 columns]
```

1.16.2 Feature Selection

[125]:
```
df =␣
 ↪df[['population','householdsize','medIncome','PctUnemployed','PolicReqPerOffic','murders','
df = df
X = df.drop('violent_crime_occurence', axis=1)
y = df['violent_crime_occurence']
```

[126]:
```
X_train_RandomForest, X_test_RandomForest, Y_train_RandomForest,␣
 ↪Y_test_RandomForest = train_test_split(X, y, random_state=1)
```

1.16.3 Calculating gini index for Random Forest Classifier

[127]:
```
from sklearn.ensemble import RandomForestClassifier
clf_gini = RandomForestClassifier(criterion = "gini",random_state =␣
 ↪200,max_depth=30, min_samples_split=9, min_samples_leaf=6)
clf_gini
```

[127]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=30, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=6, min_samples_split=9,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=200, verbose=0, warm_start=False)
```

1.16.4 Fitting and predicitng the model

[128]:
```
clf_gini.fit(X_train_RandomForest, Y_train_RandomForest)
```

[128]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=30, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=6, min_samples_split=9,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=200, verbose=0, warm_start=False)
```

[129]:
```
Y_Pred_RandomForest = clf_gini.predict(X_test_RandomForest)
```

1.16.5 Metrics

[130]:
```
ac=accuracy_score(Y_test_RandomForest,Y_Pred_RandomForest)*100
ac
```

[130]:
```
86.64259927797833
```

1.16.6 Confusion Matrix

[131]:
```
cnf_matrix_RandomForest = metrics.confusion_matrix(Y_test_RandomForest,␣
 ↪Y_Pred_RandomForest)
cnf_matrix_RandomForest
```

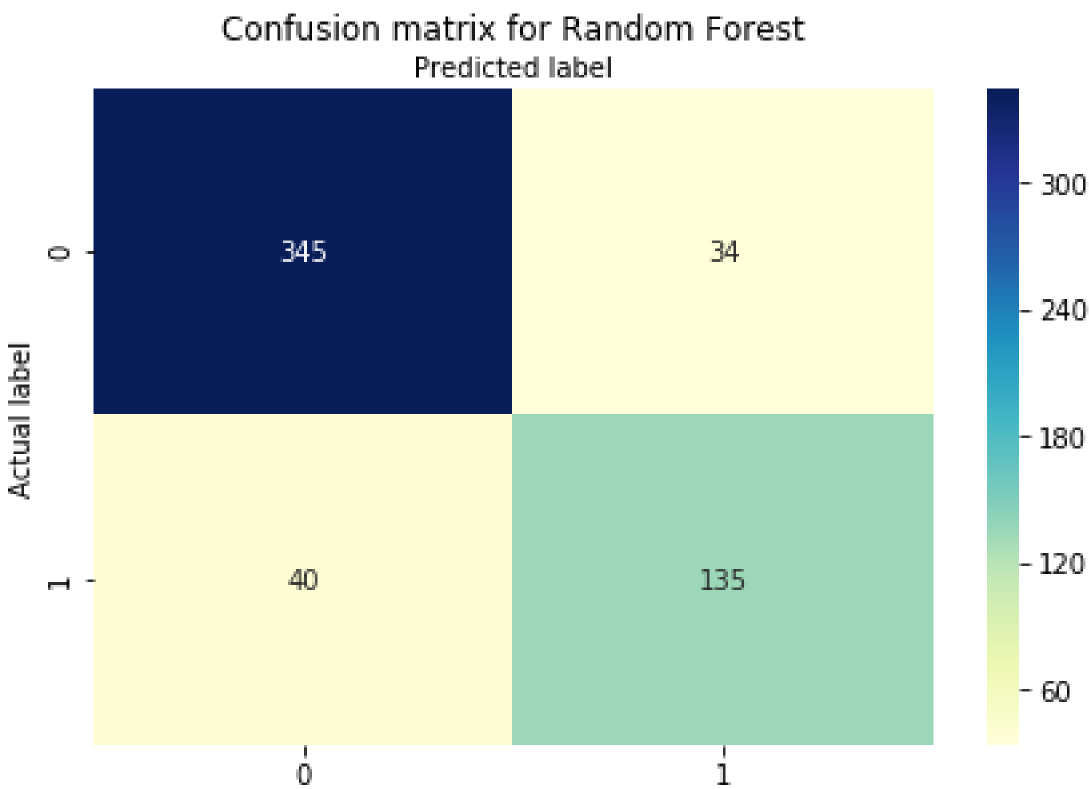
[131]:
```
array([[345,  34],
       [ 40, 135]])
```

[132]:
```
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix_RandomForest), annot=True, cmap="YlGnBu"␣
 ↪,fmt='g')
ax.xaxis.set_label_position("top")
```

```
plt.tight_layout()
plt.title('Confusion matrix for Random Forest', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
print("Model Accuracy for Random Forest:",metrics.
 ↪accuracy_score(Y_test_RandomForest, Y_Pred_RandomForest))
```

```
Model Accuracy for Random Forest: 0.8664259927797834
```



1.17 SVM

[144]:
```
#X = balance_data.values[:,␣
 ↪[5,6,17,37,47,50,56,96,129,131,133,135,137,139,141,143,145]]
df2 =␣
 ↪df2[['population','householdsize','racePctWhite','racepctblack','racePctHisp','medIncome','
df2 = df2
```

[145]:
```
X_SVM = df2.iloc[:, [3, 4]].values
Y_SVM = df2.iloc[:, 12].values
```

1.17.1 Splitting the dataset into the Training set and Test set

[146]:
```
from sklearn.cross_validation import train_test_split
X_train_SVM, X_test_SVM, Y_train_SVM, Y_test_SVM = train_test_split(X_SVM,␣
↪Y_SVM, test_size = 0.30, random_state = 0)
```

1.17.2 Feature Scaling

[147]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_SVM = sc.fit_transform(X_train_SVM)
X_test_SVM = sc.transform(X_test_SVM)
print(X_train_SVM)
```

```
[[ 0.557996   -0.49362092]
 [-0.58847309 -0.28112885]
 [-0.62648141  3.73638802]
 …
 [-0.27887807  1.55338608]
 [-0.55737538 -0.35266784]
 [ 0.01551364  0.09498213]]
```

1.17.3 Training & fitting the model

[148]:
```
# Fitting SVM to the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear',random_state = 0)
classifier.fit(X_train_SVM, Y_train_SVM)
```

[148]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=0, shrinking=True,
  tol=0.001, verbose=False)
```

[149]:
```
Y_Pred_SVM = classifier.predict(X_test_SVM)
```

The support vectors for the model are as follows

[150]:
```
print(classifier.support_vectors_)
```

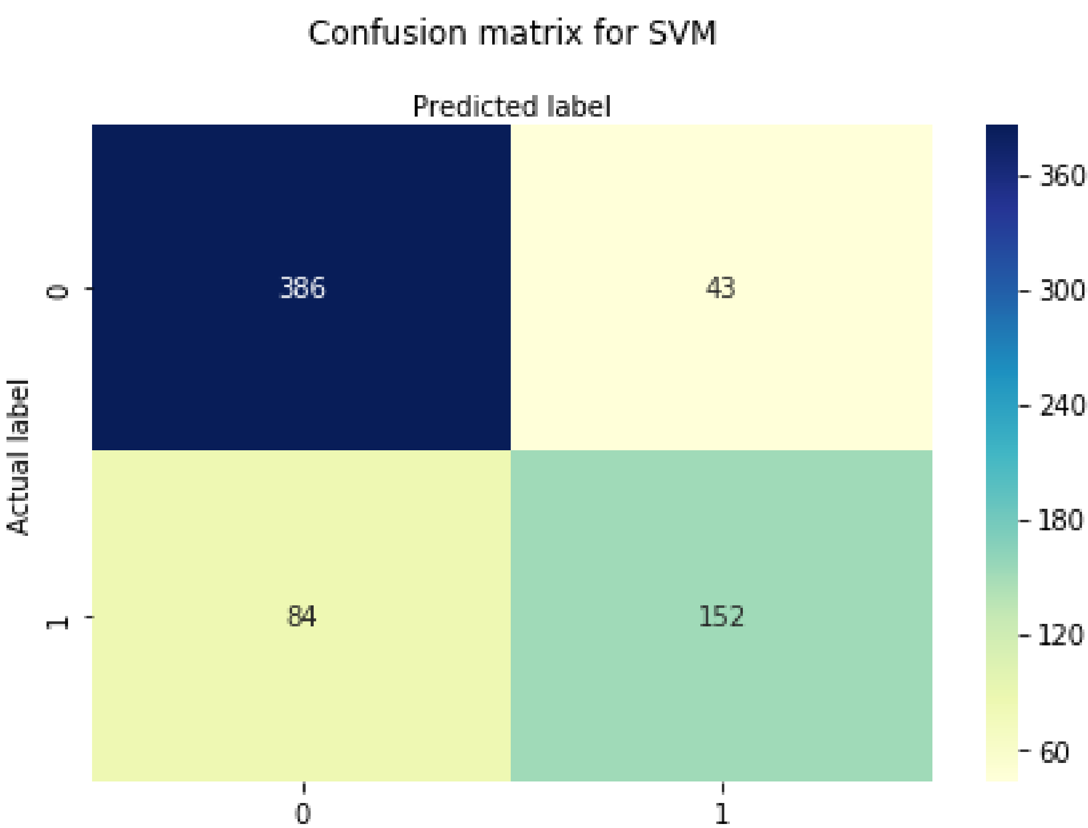
```
[[ 0.5282804  -0.43128991]
 [-0.24847141  0.92865938]
 [-0.21530052 -0.43483145]
 …
 [ 0.69275276 -0.09059428]
 [-0.27887807  1.55338608]
 [ 0.01551364  0.09498213]]
```

1.17.4 Confusion Matrix

[113]:
```
cnf_matrix_RandomForest = metrics.confusion_matrix(Y_test_SVM, Y_Pred_SVM)
cnf_matrix_RandomForest
# name  of classes
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix_RandomForest), annot=True, cmap="YlGnBu"␣
 ↪,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix for SVM', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
print("Model Accuracy for SVM:",metrics.accuracy_score(Y_test_SVM, Y_Pred_SVM))
```

```
Model Accuracy for SVM: 0.8090225563909774
```



1.17.5 Accuracy

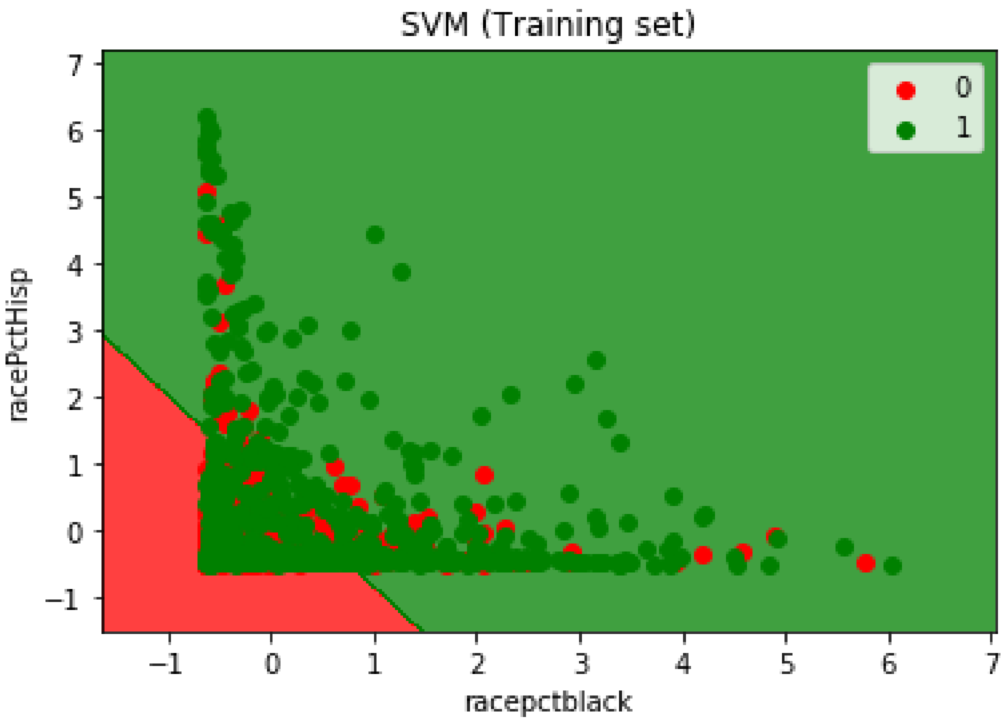
[151]:
```python
ac=accuracy_score(Y_test_SVM,Y_Pred_SVM)*100
ac
```

[151]: 80.90225563909775

1.17.6 Vizualising model results

[152]:
```python
# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train_SVM, Y_train_SVM
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:,␣
↪0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:,␣
↪1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).
↪reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('SVM (Training set)')
plt.xlabel('racepctblack')
plt.ylabel('racePctHisp')
plt.legend()
plt.show()
```

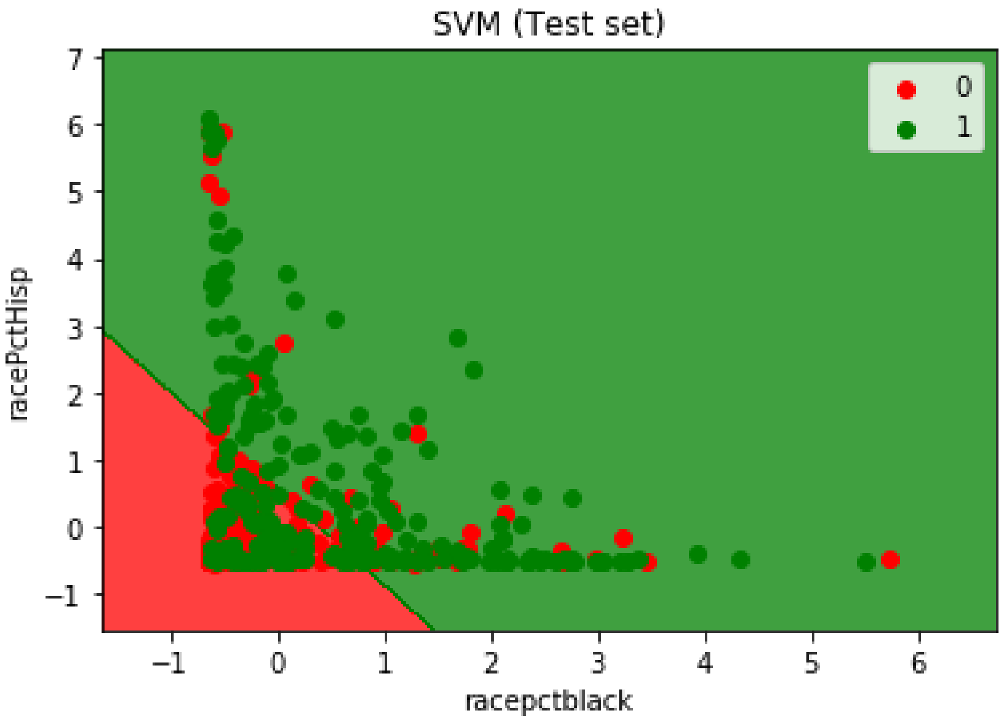
```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row if you really want to
specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row if you really want to
specify the same RGB or RGBA value for all points.
```



[153]:
```
# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test_SVM, Y_test_SVM
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:,␣
↪0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:,␣
↪1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).
↪reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('SVM (Test set)')
plt.xlabel('racepctblack')
plt.ylabel('racePctHisp')
plt.legend()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row if you really want to
```

```
specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row if you really want to
specify the same RGB or RGBA value for all points.
```



1.17.7 Fitting SVM to the Training set

[154]:
```python
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf',random_state = 0)
classifier.fit(X_train_SVM, Y_train_SVM)
```

[154]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=0, shrinking=True,
  tol=0.001, verbose=False)
```

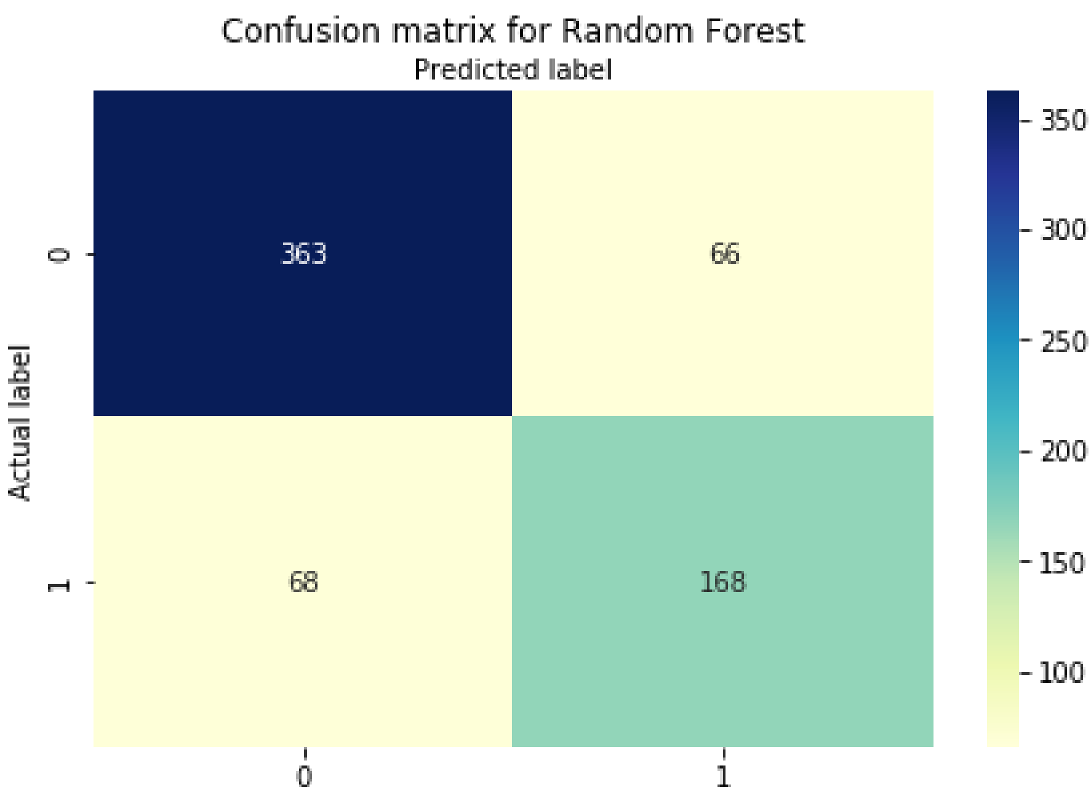
[155]:
```python
Y_Pred_SVMrbf = classifier.predict(X_test_SVM)
```

[156]:
```python
cnf_matrix_RandomForest = metrics.confusion_matrix(Y_test_SVM, Y_Pred_SVMrbf)
cnf_matrix_RandomForest
# name  of classes
class_names=[0,1]
```

```
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix_RandomForest), annot=True, cmap="YlGnBu"␣
↪,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix for Random Forest', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
print("Model Accuracy for Random Forest:",metrics.accuracy_score(Y_test_SVM,␣
↪Y_Pred_SVMrbf))
```

```
Model Accuracy for Random Forest: 0.7984962406015037
```



[157]:
```
ac=accuracy_score(Y_test_SVM,Y_Pred_SVMrbf)*100
ac
```

[157]: 79.84962406015038

1.18 PCA

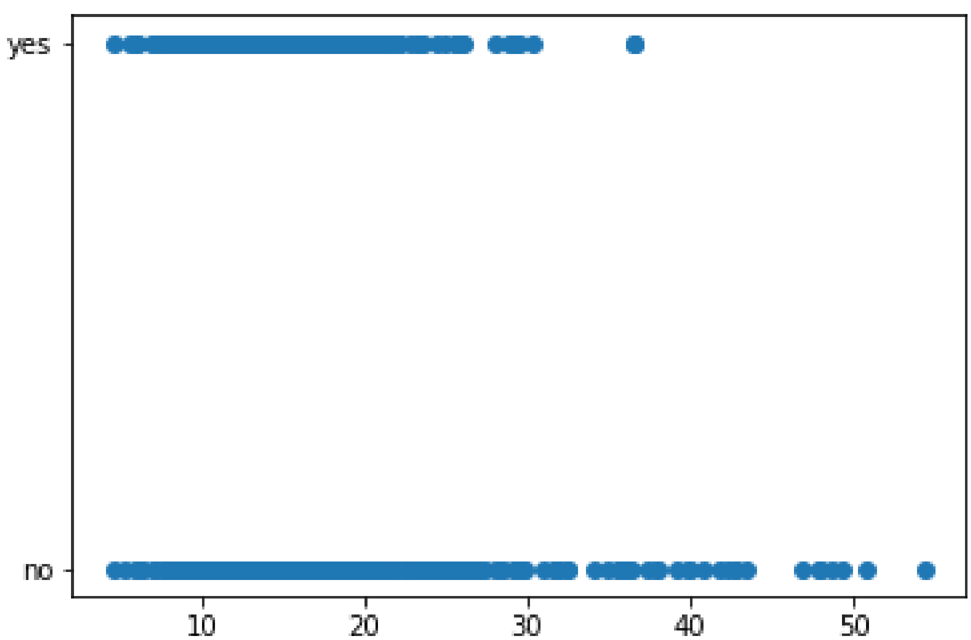
Principal Component Analysis(PCA) PCA is one of the ways to speed up a Machine Learning algorithm so that it fits faster to the training data. There might be a case where the input data or features might be in a higher dimension resulting in slow learning algorithm which takes a long time. To reduce the dimensionality without affecting or loosing any information which can be seen by the variance ratio. One of the aim of PCA is to maximise variance that is, after PCA is applied and if we want to reconstruct the original data back from the principal components, variance or information gained should be maximised or the information lost while doing so minimised. On our crime dataset, we applied PCA over the Age columns with certain age ranges in order to reduce the feature dimensionality into a 2 dimensional space and plot the features against the label 'violent_crime_occurence' to see the result of applying PCA. The feature set first needs to be standardized and scaled well to give us accurate results. The feature set consists of the following columns from the dataset: 'agePct12t21','’agePct12t29,'agePct16t24'and 'agePct65up'.

[160]:
```
from sklearn.cross_validation import cross_val_score
```

1.18.1 Individual feature vs label plots to vizualize data before applying PCA

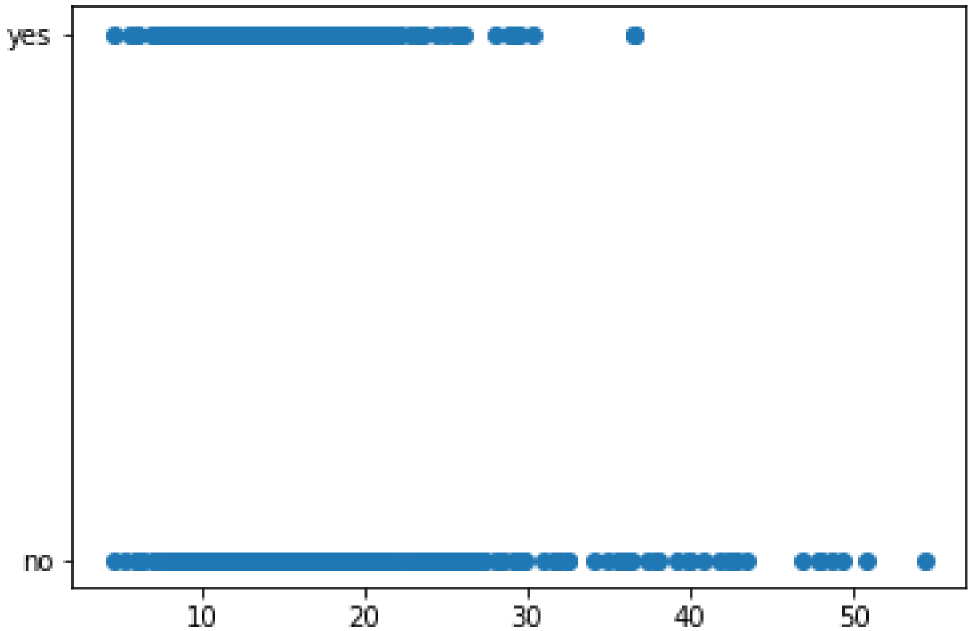
[161]:
```
X3 = df['agePct12t21'].values
y3 = df['violent_crime_occurence'].values
plt.scatter(X3, y3)
plt.show
```

[161]: <function matplotlib.pyplot.show(*args, **kw)>



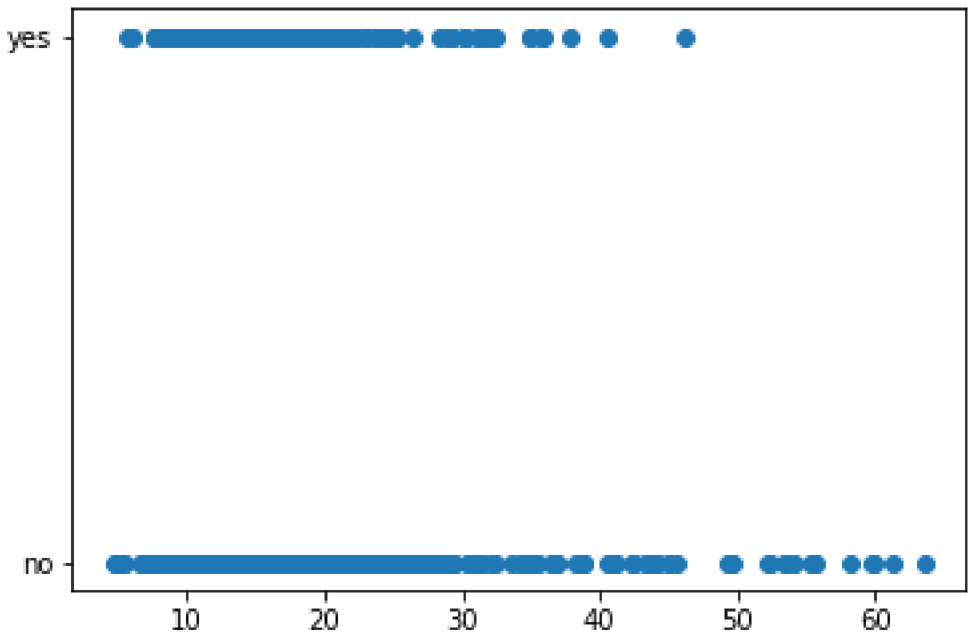
[162]:
```
X4 = df['agePct12t21'].values
y4 = df['violent_crime_occurence'].values
plt.scatter(X4, y4)
plt.show
```

[162]: `<function matplotlib.pyplot.show(*args, **kw)>`



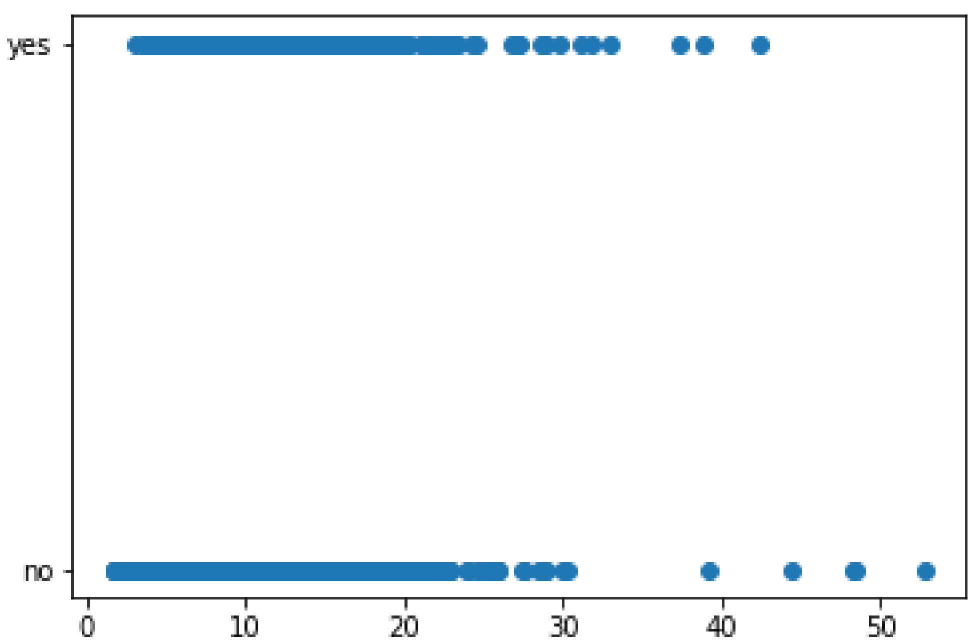
[163]:
```
X5 = df['agePct16t24'].values
y5 = df['violent_crime_occurence'].values
plt.scatter(X5, y5)
plt.show
```

[163]: `<function matplotlib.pyplot.show(*args, **kw)>`



[164]:
```
X6 = df['agePct65up'].values
y6 = df['violent_crime_occurence'].values
plt.scatter(X6, y6)
plt.show
```

[164]: <function matplotlib.pyplot.show(*args, **kw)>



1.18.2 Vizualization Inference

It can be inferred that all the Age range features have similar kind of a relationship with the label and hence could be combined in order to reduce the dimensionality and hence speed up the learning process of a model.

1.18.3 Feature Selection

[165]:
```
features = ['agePct12t21','agePct12t29','agePct16t24', 'agePct65up']
X= df.loc[:, features].values
y = df.loc[:, ['violent_crime_occurence']].values
```

1.18.4 Splitting Dataset into Test and Training Data

[166]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)
```

1.18.5 Scaling and Standardizing Data

[167]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

1.18.6 Applying PCA from sklearn for 2 Principal Components

[168]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
principalComponents = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
explained_variance = pca.explained_variance_ratio_
```

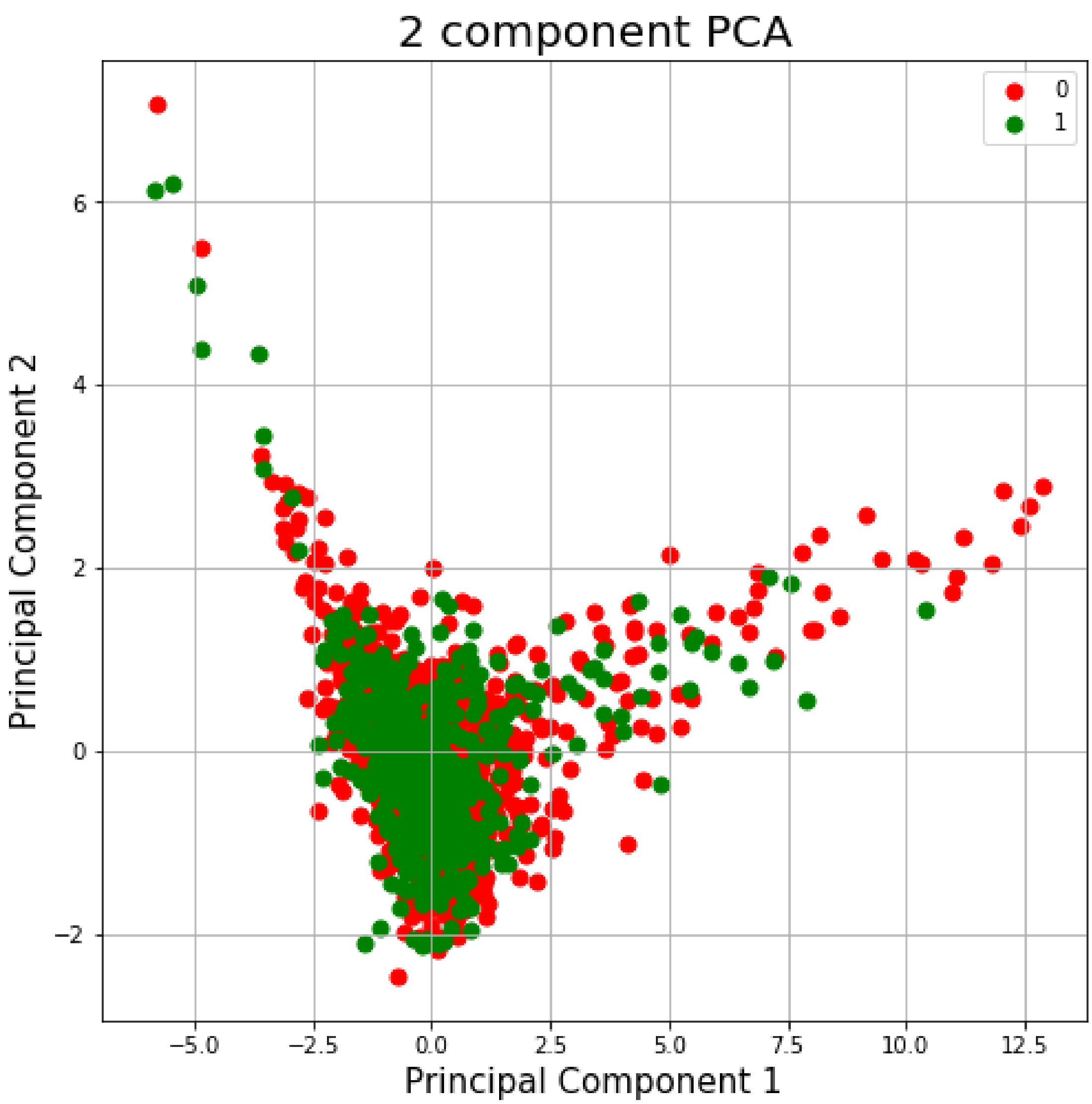
1.18.7 Final Dataframe with label concatenated with features

[169]:
```
finalDf = pd.concat([principalDf, df[['violent_crime_occurence']]], axis = 1)
```

1.18.8 Plot to observe the 2 Principal Components as a result of PCA

[117]:
```
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
```

```
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
targets = ['0', '1']
colors = ['r', 'g']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['violent_crime_occurence'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```



1.18.9 Variance Ratio

The variance ratio values are 75.94% and 20.71% meaning that approximately 96% of the information can recontructed from the model and hence the Principal Components are as per model conventions.

[171]:
```
print(explained_variance)
```

```
[0.75942017 0.20717271]
```

1.19 KNN

[195]:
```
X_KNN = balance_data.iloc[:, [2,6]].values
Y_KNN = balance_data.iloc[:, 12].values
```

1.19.1 Splitting the dataset into the Training set and Test set

[196]:
```
from sklearn.cross_validation import train_test_split
X_train_KNN, X_test_KNN, Y_train_KNN, Y_test_KNN = train_test_split(X_KNN,␣
 ↪Y_KNN, test_size = 0.30, random_state = 0)
```

1.19.2 Feature Scaling

[197]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_KNN = sc.fit_transform(X_train_KNN)
X_test_KNN = sc.transform(X_test_KNN)
```

1.19.3 Training and testing the model

[198]:
```
# Fitting K-NN to the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train_KNN, Y_train_KNN)
```

[198]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
```

[199]:
```
# Predicting the Test set results
Y_Pred_KNN = classifier.predict(X_test_KNN)
```

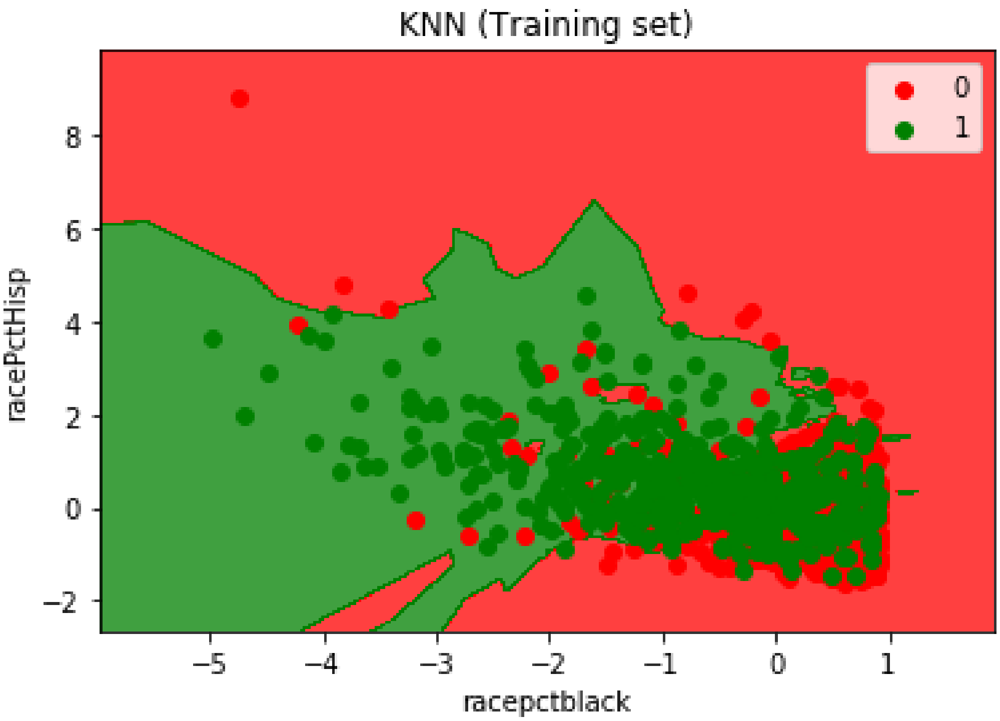
1.19.4 Accuracy, Confusion Matrix & Heatmap

[200]:
```
ac=accuracy_score(Y_test_KNN,Y_Pred_KNN)*100
ac
```

[200]: 79.84962406015038

[201]:
```
# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train_KNN, Y_train_KNN
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:,␣
↪0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:,␣
↪1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).
↪reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('KNN (Training set)')
plt.xlabel('racepctblack')
plt.ylabel('racePctHisp')
plt.legend()
plt.show()
```

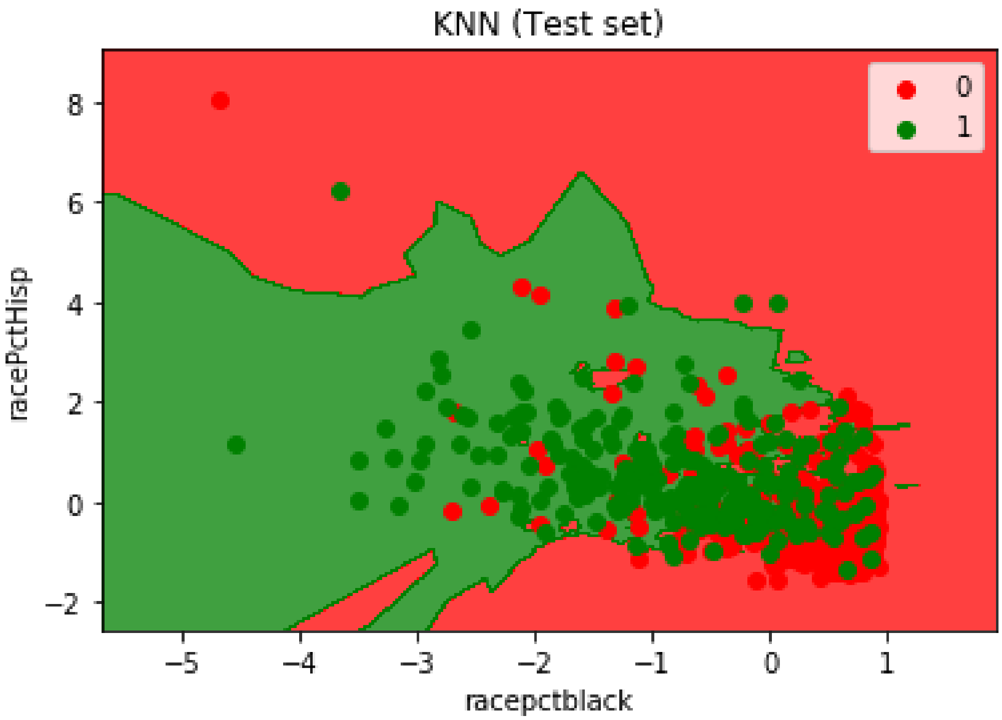
```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row if you really want to
specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row if you really want to
specify the same RGB or RGBA value for all points.
```



[202]:
```python
# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test_KNN, Y_test_KNN
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:,␣
↪0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:,␣
↪1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).
↪reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('KNN (Test set)')
plt.xlabel('racepctblack')
plt.ylabel('racePctHisp')
plt.legend()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row if you really want to
```

```
specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row if you really want to
specify the same RGB or RGBA value for all points.
```



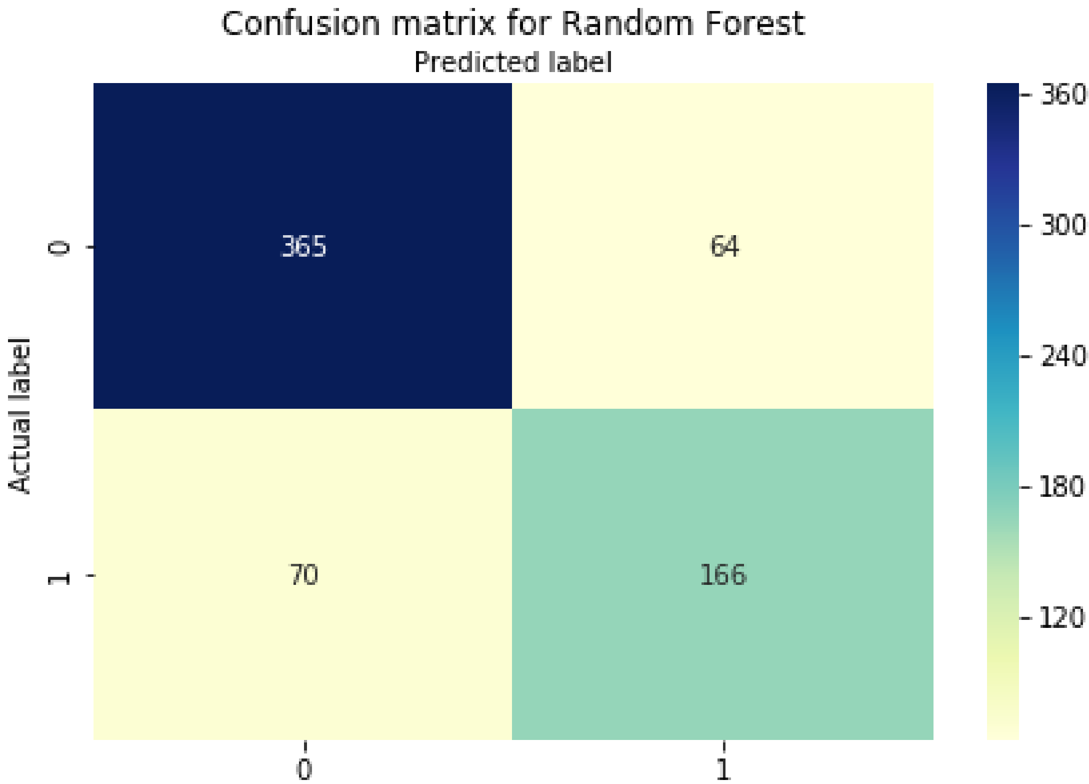
[203]:

```python
cnf_matrix_RandomForest = metrics.confusion_matrix(Y_test_KNN, Y_Pred_KNN)
cnf_matrix_RandomForest
# name  of classes
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix_RandomForest), annot=True, cmap="YlGnBu"␣
 ↪,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix for Random Forest', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

```
print("Model Accuracy for Random Forest:",metrics.accuracy_score(Y_test_KNN,␣
↪Y_Pred_KNN))
```

```
Model Accuracy for Random Forest: 0.7984962406015037
```



2 Conclusion

The predictions made by various classification algorithms show the occurence possibility of a crime whether a crime will occur or not, if a crime occurs, will it be a violent or a non-violent crime or if a crime occurs, is the cause of the crime murder or not. These predictions might help the local police departments as well as the FBI solve many cases with esfficiency and accuracy.

Among the classification algorithms, Random Forest Classifier performs the best making a decision based on majority vote and constructing a decision tree for each feature. The highest accuracy achieve with Random Forest Classifier is 86.86%. Also, we observed that the dataset performs well with non-linear data as compared to linear data hence not so good results were achieved with Linear Regression.