BITS F464 - Semester 1 - MACHINE LEARNING

ASSIGNMENT 2 – DECISION TREES AND SUPPORT VECTOR MACHINES

Team number: 24

Full names of all students in the team:

Pavas Garg, Tushar Raghani, Rohan Pothireddy, Kolasani Amit Vishnu

Id number of all students in the team:

2021A7PS2587H, 2021A7PS1404H, 2021A7PS0365H, 2021A7PS0151H

1. Preprocess and perform exploratory data analysis of the dataset obtained

Importing The Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
import random
```

Importing the Dataset

In [2]:
```
df = pd.read_csv("communities.data",header=None)
df.head()
```

Out[2]:

	0	1	2	3	4	5	6	7	8	9	...	118	119	120	1
0	8	?	?	Lakewoodcity	1	0.19	0.33	0.02	0.90	0.12	...	0.12	0.26	0.20	0.(
1	53	?	?	Tukwilacity	1	0.00	0.16	0.12	0.74	0.45	...	0.02	0.12	0.45	
2	24	?	?	Aberdeentown	1	0.00	0.42	0.49	0.56	0.17	...	0.01	0.21	0.02	
3	34	5	81440	Willingborotownship	1	0.04	0.77	1.00	0.08	0.12	...	0.02	0.39	0.28	
4	42	95	6096	Bethlehemtownship	1	0.01	0.55	0.02	0.95	0.09	...	0.04	0.09	0.02	

5 rows × 128 columns

1. 🔬Preprocessing of Dataset

In [3]:

```
# regular expression model
import re

# Read the .names file
with open('communities.names', 'r') as file:
    names_content = file.read()

# Find the start and end positions of the attribute names
start_pos = names_content.find('@attribute') + len('@attribute')
end_pos = names_content.find('@data')

# Extract the attribute names section
attributes_section = names_content[start_pos:end_pos]

# Split the attribute names into a list
attributes_list = [line.split()[1] for line in attributes_section.split('\n')

# Special handling for the first attribute to ensure correct extraction
first_attribute_line = names_content[names_content.find('@attribute'):names_c
first_attribute_name = re.search('@attribute\s+(\S+)\s', first_attribute_line
attributes_list[0] = first_attribute_name

# Print the extracted attribute names
print("Attribute Names:")
print(attributes_list)
```

```
Attribute Names:
['state', 'county', 'community', 'communityname', 'fold', 'population', 'hous
eholdsize', 'racepctblack', 'racePctWhite', 'racePctAsian', 'racePctHisp', 'a
gePct12t21', 'agePct12t29', 'agePct16t24', 'agePct65up', 'numbUrban', 'pctUrb
an', 'medIncome', 'pctWWage', 'pctWFarmSelf', 'pctWInvInc', 'pctWSocSec', 'pc
tWPubAsst', 'pctWRetire', 'medFamInc', 'perCapInc', 'whitePerCap', 'blackPerC
ap', 'indianPerCap', 'AsianPerCap', 'OtherPerCap', 'HispPerCap', 'NumUnderPo
v', 'PctPopUnderPov', 'PctLess9thGrade', 'PctNotHSGrad', 'PctBSorMore', 'PctU
nemployed', 'PctEmploy', 'PctEmplManu', 'PctEmplProfServ', 'PctOccupManu', 'P
ctOccupMgmtProf', 'MalePctDivorce', 'MalePctNevMarr', 'FemalePctDiv', 'TotalP
ctDiv', 'PersPerFam', 'PctFam2Par', 'PctKids2Par', 'PctYoungKids2Par', 'PctTe
en2Par', 'PctWorkMomYoungKids', 'PctWorkMom', 'NumIlleg', 'PctIlleg', 'NumImm
ig', 'PctImmigRecent', 'PctImmigRec5', 'PctImmigRec8', 'PctImmigRec10', 'PctR
ecentImmig', 'PctRecImmig5', 'PctRecImmig8', 'PctRecImmig10', 'PctSpeakEnglOn
ly', 'PctNotSpeakEnglWell', 'PctLargHouseFam', 'PctLargHouseOccup', 'PersPerO
ccupHous', 'PersPerOwnOccHous', 'PersPerRentOccHous', 'PctPersOwnOccup', 'Pct
PersDenseHous', 'PctHousLess3BR', 'MedNumBR', 'HousVacant', 'PctHousOccup',
'PctHousOwnOcc', 'PctVacantBoarded', 'PctVacMore6Mos', 'MedYrHousBuilt', 'Pct
HousNoPhone', 'PctWOFullPlumb', 'OwnOccLowQuart', 'OwnOccMedVal', 'OwnOccHiQu
art', 'RentLowQ', 'RentMedian', 'RentHighQ', 'MedRent', 'MedRentPctHousInc',
'MedOwnCostPctInc', 'MedOwnCostPctIncNoMtg', 'NumInShelters', 'NumStreet', 'P
ctForeignBorn', 'PctBornSameState', 'PctSameHouse85', 'PctSameCity85', 'PctSa
meState85', 'LemasSwornFT', 'LemasSwFTPerPop', 'LemasSwFTFieldOps', 'LemasSwF
TFieldPerPop', 'LemasTotalReq', 'LemasTotReqPerPop', 'PolicReqPerOffic', 'Pol
icPerPop', 'RacialMatchCommPol', 'PctPolicWhite', 'PctPolicBlack', 'PctPolicH
isp', 'PctPolicAsian', 'PctPolicMinor', 'OfficAssgnDrugUnits', 'NumKindsDrugs
Seiz', 'PolicAveOTWorked', 'LandArea', 'PopDens', 'PctUsePubTrans', 'PolicCar
s', 'PolicOperBudg', 'LemasPctPolicOnPatr', 'LemasGangUnitDeploy', 'LemasPctO
fficDrugUn', 'PolicBudgPerPop', 'ViolentCrimesPerPop']
```

In [4]:

```
# adding attribute names for the dataset
attribute_names = attributes_list
```

```
df.columns = attribute_names

print("Number of observations in this dataset are: ", len(df))
print("Number of features in this dataset are: ", len(df.columns)-1)
df.head()
```

```
Number of observations in this dataset are:  1994
Number of features in this dataset are:  127
```

Out[4]:

	state	county	community	communityname	fold	population	householdsize	racepctblac
0	8	?	?	Lakewoodcity	1	0.19	0.33	0.0
1	53	?	?	Tukwilacity	1	0.00	0.16	0.1
2	24	?	?	Aberdeentown	1	0.00	0.42	0.4
3	34	5	81440	Willingborotownship	1	0.04	0.77	1.0
4	42	95	6096	Bethlehemtownship	1	0.01	0.55	0.0

5 rows × 128 columns

In [5]:

```
# understanding the given data
df.describe()
```

Out[5]:

	state	fold	population	householdsize	racepctblack	racePctWhite	ra
count	1994.000000	1994.000000	1994.000000	1994.000000	1994.000000	1994.000000	19
mean	28.683551	5.493982	0.057593	0.463395	0.179629	0.753716	
std	16.397553	2.873694	0.126906	0.163717	0.253442	0.244039	
min	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	
25%	12.000000	3.000000	0.010000	0.350000	0.020000	0.630000	
50%	34.000000	5.000000	0.020000	0.440000	0.060000	0.850000	
75%	42.000000	8.000000	0.050000	0.540000	0.230000	0.940000	
max	56.000000	10.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 102 columns

In [6]:

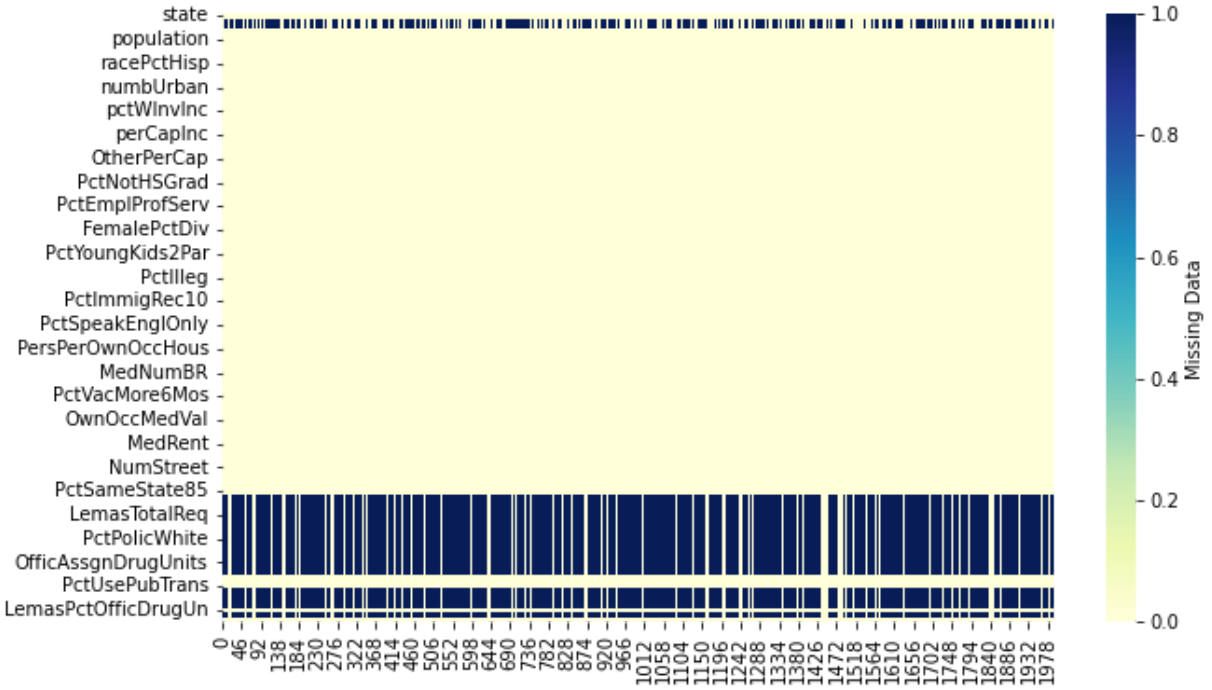
```
# to suppress specific warning
import warnings
with warnings.catch_warnings():
    warnings.simplefilter("ignore", category=pd.errors.PerformanceWarning)
```

Missing Values

In [7]:

```
# replacing missing values represented by ? by NA
df.replace('?', pd.NA, inplace=True)
```

In [8]:
```
# heatmap for missing data visualization
plt.figure(figsize=(10,6))
sns.heatmap(df.isna().transpose(),cmap="YlGnBu",cbar_kws={'label': 'Missing D
plt.show()
```



In [9]:
```
# printing columns which have null values
[col for col in df.columns if df[col].isnull().any()]
```

Out[9]:
```
['county',
 'community',
 'OtherPerCap',
 'LemasSwornFT',
 'LemasSwFTPerPop',
 'LemasSwFTFieldOps',
 'LemasSwFTFieldPerPop',
 'LemasTotalReq',
 'LemasTotReqPerPop',
 'PolicReqPerOffic',
 'PolicPerPop',
 'RacialMatchCommPol',
 'PctPolicWhite',
 'PctPolicBlack',
 'PctPolicHisp',
 'PctPolicAsian',
 'PctPolicMinor',
 'OfficAssgnDrugUnits',
 'NumKindsDrugsSeiz',
 'PolicAveOTWorked',
 'PolicCars',
 'PolicOperBudg',
 'LemasPctPolicOnPatr',
 'LemasGangUnitDeploy',
 'PolicBudgPerPop']
```

In [10]:
```
# Set the threshold for null values
threshold = 1000

# Count null values in each column
missing_values = df.isna().sum()
```

```
print("Count of missing values:")
print(missing_values[missing_values > 0])

# Filter columns that exceed the threshold
columns_to_drop = missing_values[missing_values > threshold].index

# Display columns with null values exceeding the threshold
print("\nColumns with null values exceeding 1000:")
print(columns_to_drop)

# Drop columns exceeding the threshold
df = df.drop(columns=columns_to_drop)
```

```
Count of missing values:
county                  1174
community               1177
OtherPerCap                1
LemasSwornFT            1675
LemasSwFTPerPop         1675
LemasSwFTFieldOps       1675
LemasSwFTFieldPerPop    1675
LemasTotalReq           1675
LemasTotReqPerPop       1675
PolicReqPerOffic        1675
PolicPerPop             1675
RacialMatchCommPol      1675
PctPolicWhite           1675
PctPolicBlack           1675
PctPolicHisp            1675
PctPolicAsian           1675
PctPolicMinor           1675
OfficAssgnDrugUnits     1675
NumKindsDrugsSeiz       1675
PolicAveOTWorked        1675
PolicCars               1675
PolicOperBudg           1675
LemasPctPolicOnPatr     1675
LemasGangUnitDeploy     1675
PolicBudgPerPop         1675
dtype: int64

Columns with null values exceeding 1000:
Index(['county', 'community', 'LemasSwornFT', 'LemasSwFTPerPop',
       'LemasSwFTFieldOps', 'LemasSwFTFieldPerPop', 'LemasTotalReq',
       'LemasTotReqPerPop', 'PolicReqPerOffic', 'PolicPerPop',
       'RacialMatchCommPol', 'PctPolicWhite', 'PctPolicBlack', 'PctPolicHis
p',
       'PctPolicAsian', 'PctPolicMinor', 'OfficAssgnDrugUnits',
       'NumKindsDrugsSeiz', 'PolicAveOTWorked', 'PolicCars', 'PolicOperBudg',
       'LemasPctPolicOnPatr', 'LemasGangUnitDeploy', 'PolicBudgPerPop'],
      dtype='object')
```

In [11]:

```
# function to replace NULL values with mean
def replace_with_mean(df,feature,mean):
   df[feature].fillna(mean,inplace=True)
```

In [12]:

```
# checking null count
null_count = df.isnull().sum().sum()
print("NULL count: ",null_count)
```

```
NULL count:  1
```

In [13]:
```
# printing features with missing values
missing_values = df.isna().sum()
print(missing_values[missing_values>0])
```

```
OtherPerCap    1
dtype: int64
```

In [14]:
```
# We see that the column OtherPerCap has only one missing value, so we replac
mean_value = pd.to_numeric(df['OtherPerCap'], errors='coerce').mean()
replace_with_mean(df,'OtherPerCap',mean_value)
```

In [15]:
```
null_count = df.isnull().sum().sum()
print("NULL count: ",null_count)
```

```
NULL count:  0
```

Categorical Variables

In [16]:
```
# checking for categorical features
categorical_columns = df.select_dtypes(include=['category', 'object']).column
print(len(categorical_columns))
```

```
2
```

In [17]:
```
categorical_columns
```

Out[17]:
```
Index(['communityname', 'OtherPerCap'], dtype='object')
```

In [18]:
```
# checking community name feature
df['communityname'].value_counts()
```

Out[18]:
```
Greenvillecity      5
Jacksonvillecity    5
Auburncity          5
Athenscity          4
Springfieldcity     4
                   ..
RedondoBeachcity    1
GrantsPasscity      1
FortWorthcity       1
Raleighcity         1
Ontariocity         1
Name: communityname, Length: 1828, dtype: int64
```

In [19]:
```
# as we can see that community name doesn't play important role in predicting
df = df.drop('communityname',axis=1)
```

In [20]:
```
type_of_values = [type(val) for val in df["OtherPerCap"]]
print(type_of_values[:5])
```

```
[<class 'str'>, <class 'str'>, <class 'str'>, <class 'str'>, <class 'str'>]
```

In [21]:
```
# converting string type to float
df["OtherPerCap"] = [float(val) for val in df["OtherPerCap"]]
df["OtherPerCap"]
```

Out[21]:
```
0       0.36
1       0.22
2       0.28
3       0.36
4       0.51
        ...
1989    0.36
1990    0.23
1991    0.22
1992    0.27
1993    0.25
Name: OtherPerCap, Length: 1994, dtype: float64
```

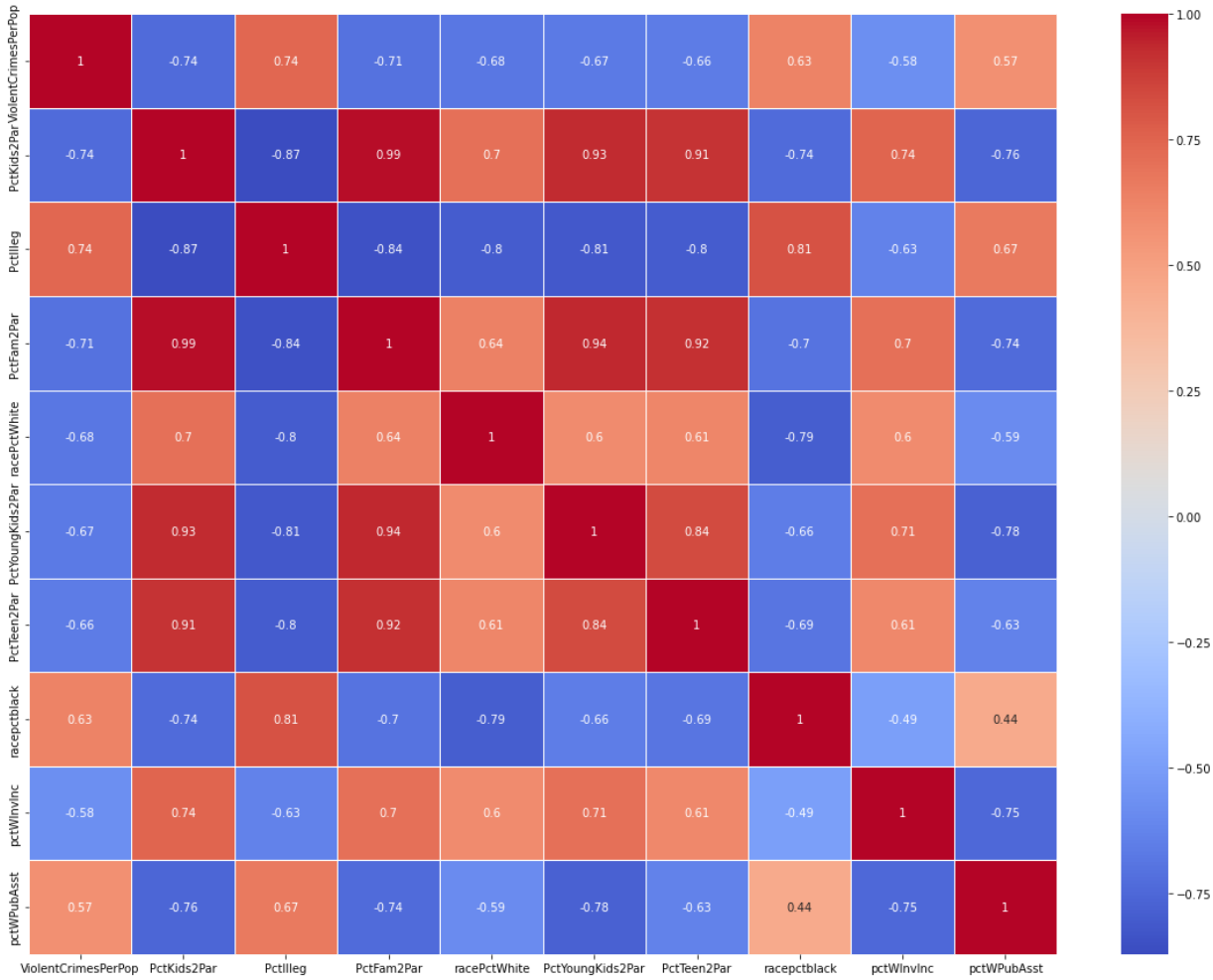
In [22]:
```
df.shape
```

Out[22]:
```
(1994, 103)
```

Correlation Matrix

- This correlation matrix shows top 10 features which are highly correlated with target feature.

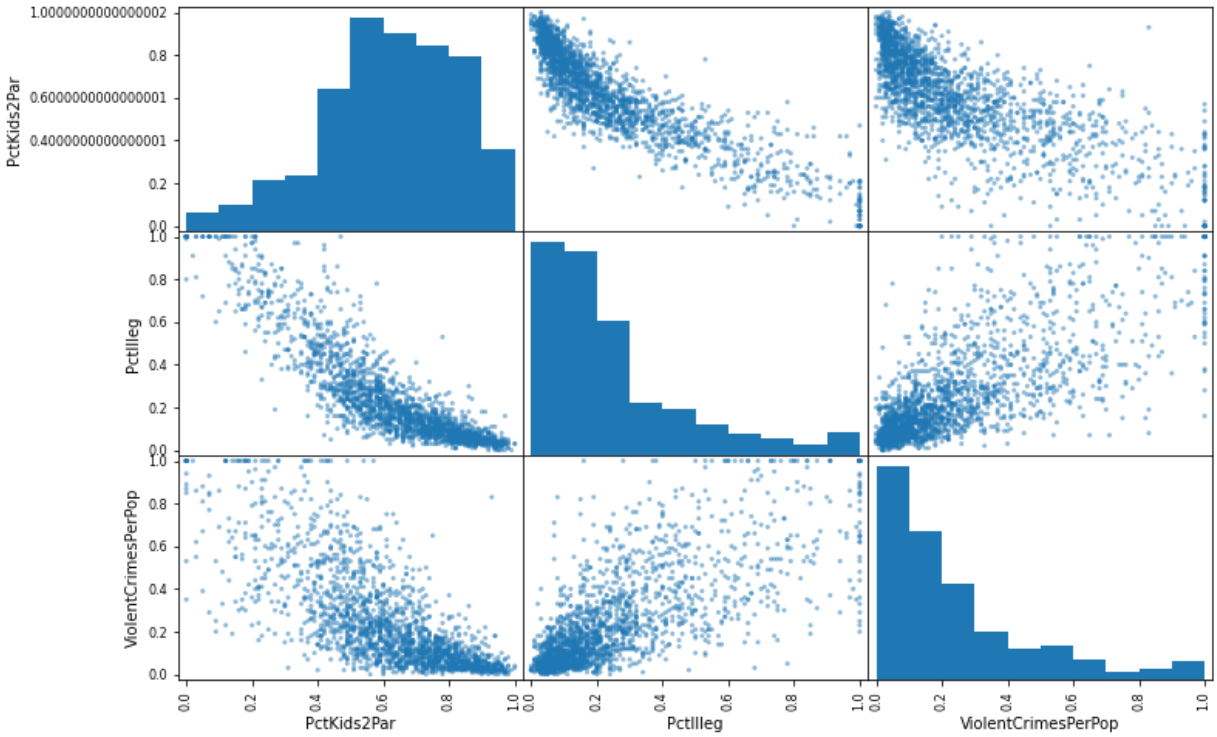
In [23]:
```
# correlation matrix
correlation_matrix = df.select_dtypes(include='number').corr()
plt.subplots(figsize=(20,15))

# finding correlation with target attribute
correlation_with_target = correlation_matrix["ViolentCrimesPerPop"]

# sort features according to correlation
sorted_features = correlation_with_target.abs().sort_values(ascending=False)
top_n_features = sorted_features.index[:10]
subset_df = df[top_n_features]

sns.heatmap(subset_df.corr(), cmap="coolwarm", annot=True, linewidths=.5)
plt.show()
```



Plotting Correlation Graphs for Strongly Related Features

In [24]:

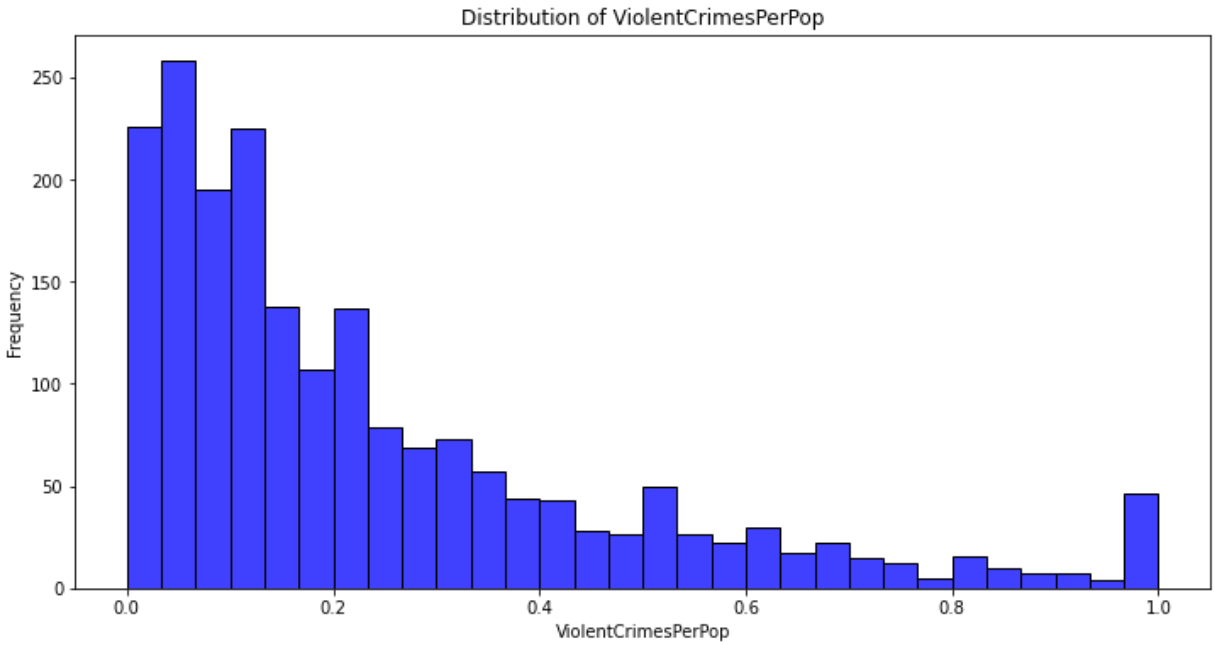
```
from pandas.plotting import scatter_matrix
attributes = ["PctKids2Par","PctIlleg","ViolentCrimesPerPop"]
scatter_matrix(df[attributes],figsize=(12,8))
plt.show()
```



Discretizing continuous target variable

In [25]:

```
target_variable = df.columns[-1]

# Plot the distribution of the target variable
plt.figure(figsize=(12, 6))
sns.histplot(df[target_variable], bins=30, kde=False, color='blue', edgecolor

# Set labels and title
plt.xlabel(target_variable)
plt.ylabel('Frequency')
plt.title('Distribution of {}'.format(target_variable))

# Show the plot
plt.show()
```



Converting target label "ViolentCrimesPerPop" to discrete class labels

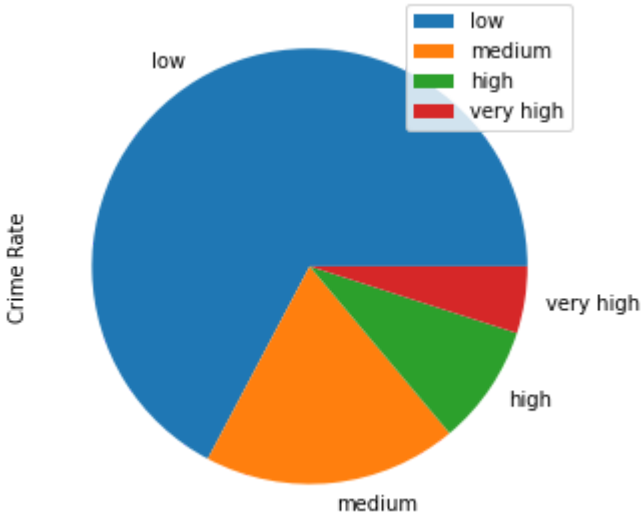
We will make 4 classes

- ViolentCrimesPerPop less than or equal to 0.25 will belong to class label "low"
- ViolentCrimesPerPop greater than 0.25 and less than or equal to 0.5 will belong to class label "medium"
- ViolentCrimesPerPop greater than 0.5 and less than or equal to 0.75 will belong to class label "high"
- ViolentCrimesPerPop greater than 0.75 and less than or equal to 1 will belong to class label "very high"

In [26]:
```
# we will do label encoding for target variable
crime_rate_values = np.array(df["ViolentCrimesPerPop"])

for ind in range(len(crime_rate_values)):
    if crime_rate_values[ind] <= 0.25:
        crime_rate_values[ind] = 1
    elif crime_rate_values[ind] > 0.25 and crime_rate_values[ind] <= 0.5:
        crime_rate_values[ind] = 2
    elif crime_rate_values[ind] > 0.5 and crime_rate_values[ind] <= 0.75:
        crime_rate_values[ind] = 3
    else:
        crime_rate_values[ind] = 4

df["ViolentCrimesPerPop"] = crime_rate_values
```

In [27]:
```
# plotting pie chart for target attribute
crime_rate = df["ViolentCrimesPerPop"]

categories = ["low","medium","high","very high"]
frequency_crime_rate = [0,0,0,0]

for val in crime_rate:
    if val == 1:
        frequency_crime_rate[0] += 1
    elif val == 2:
        frequency_crime_rate[1] += 1
```

```
    elif val == 3:
        frequency_crime_rate[2] += 1
    elif val == 4:
        frequency_crime_rate[3] += 1

dictionary = dict(zip(categories, frequency_crime_rate))

# plotting pie chart
df_crime_rate = pd.DataFrame({'Crime Rate': frequency_crime_rate,},index=cate
plot = df_crime_rate.plot.pie(y='Crime Rate', figsize=(5, 5))
```

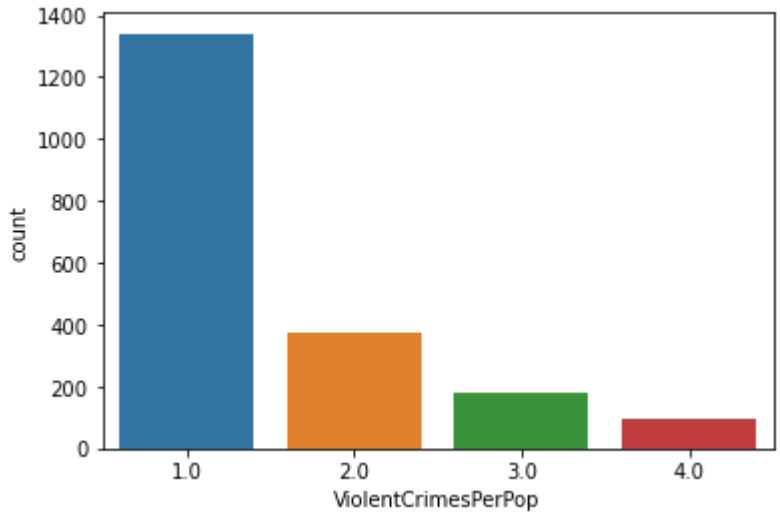


Visualizing Class Imbalance

In [28]:
```
df["ViolentCrimesPerPop"].value_counts()
```

Out[28]:
```
1.0    1341
2.0     376
3.0     178
4.0      99
Name: ViolentCrimesPerPop, dtype: int64
```

In [29]:
```
sns.countplot(x = 'ViolentCrimesPerPop',data = df)
plt.show()
```



Here we can observe that number of samples in dataset for very high crime rate is much less than number of samples which correspond to low crime rate, hence there is class imbalance.

Train-Test Split

In [30]:
```
def split_train_test(data,test_ratio):
    # np.random.seed() is very important as whenever we call the function it
    # it might happen after many calls our model sees all the data and it lea
    # seed function will randomly divide data only once and once the function
    # permuatation of indices whenever called again,hence no overfitting
    np.random.seed(45)
    # it will give random permutation of indices from 0 to len(data)-1
    # now shuffled array will contain random number for eg [0,4,1,99,12,3...]
    shuffled = np.random.permutation(len(data))
    test_set_size = int(len(data)*test_ratio)
    # it will give array of indices from index 0 to test_set_size-1
    test_indices = shuffled[:test_set_size]
    # it will give array of indices from index test_set_size till last
    train_indices = shuffled[test_set_size:]
    # it will return rows from data df corresponding to indices given in trai
    # so it is returning the train and test data respectively
    return data.iloc[train_indices], data.iloc[test_indices]
```

In [31]:
```
train_set, test_set = split_train_test(df,0.2)
```

In [32]:
```
train_set.head()
```

Out[32]:

	state	fold	population	householdsize	racepctblack	racePctWhite	racePctAsian	raceP
619	6	4	0.05	0.51	0.04	0.64	1.00	
1808	29	10	0.05	0.32	0.94	0.22	0.13	
1644	21	9	0.02	0.49	0.38	0.60	0.25	
881	45	5	0.00	0.51	0.20	0.83	0.04	
1510	41	8	0.16	0.43	0.03	0.87	0.15	

5 rows × 103 columns

In [33]:
```
len(train_set)
```

Out[33]:
1596

In [34]:
```
test_set.head()
```

Out[34]:

	state	fold	population	householdsize	racepctblack	racePctWhite	racePctAsian	raceP
710	21	4	0.01	0.37	0.01	0.99	0.02	
1091	34	6	0.07	0.58	0.22	0.59	0.05	

	state	fold	population	householdsize	racepctblack	racePctWhite	racePctAsian	raceP
39	34	1	0.01	0.14	0.21	0.67	0.61	
9	29	1	0.01	0.40	0.06	0.87	0.30	
1607	6	9	0.04	1.00	0.07	0.45	0.48	

5 rows × 103 columns

In [35]:
```
len(test_set)
```

Out[35]: 398

In [36]:
```
# changing indexing to 0 based indexing
train_set.index = range(len(train_set))
train_set
```

Out[36]:

	state	fold	population	householdsize	racepctblack	racePctWhite	racePctAsian	raceP
0	6	4	0.05	0.51	0.04	0.64	1.00	
1	29	10	0.05	0.32	0.94	0.22	0.13	
2	21	9	0.02	0.49	0.38	0.60	0.25	
3	45	5	0.00	0.51	0.20	0.83	0.04	
4	41	8	0.16	0.43	0.03	0.87	0.15	
...	...	...	...	...	...	...	...	
1591	56	8	0.01	0.29	0.00	0.97	0.03	
1592	6	10	0.02	1.00	0.01	0.48	0.32	
1593	42	9	0.00	0.53	0.01	0.95	0.20	
1594	34	3	0.02	0.59	0.14	0.83	0.16	
1595	25	5	0.00	0.59	0.01	0.93	0.24	

1596 rows × 103 columns

In [37]:
```
# changing indexing to 0 based indexing
test_set.index = range(len(test_set))
test_set
```

Out[37]:

	state	fold	population	householdsize	racepctblack	racePctWhite	racePctAsian	racePc
0	21	4	0.01	0.37	0.01	0.99	0.02	
1	34	6	0.07	0.58	0.22	0.59	0.05	
2	34	1	0.01	0.14	0.21	0.67	0.61	
3	29	1	0.01	0.40	0.06	0.87	0.30	
4	6	9	0.04	1.00	0.07	0.45	0.48	
...	...	...	...	...	...	...	...	

	state	fold	population	householdsize	racepctblack	racePctWhite	racePctAsian	racePc
393	23	9	0.00	1.00	0.01	0.95	0.11	
394	55	8	0.00	0.27	0.00	1.00	0.02	
395	8	1	0.34	0.35	0.22	0.73	0.23	
396	48	3	0.00	0.62	0.30	0.42	0.02	
397	25	10	0.08	0.51	0.06	0.87	0.22	

398 rows × 103 columns

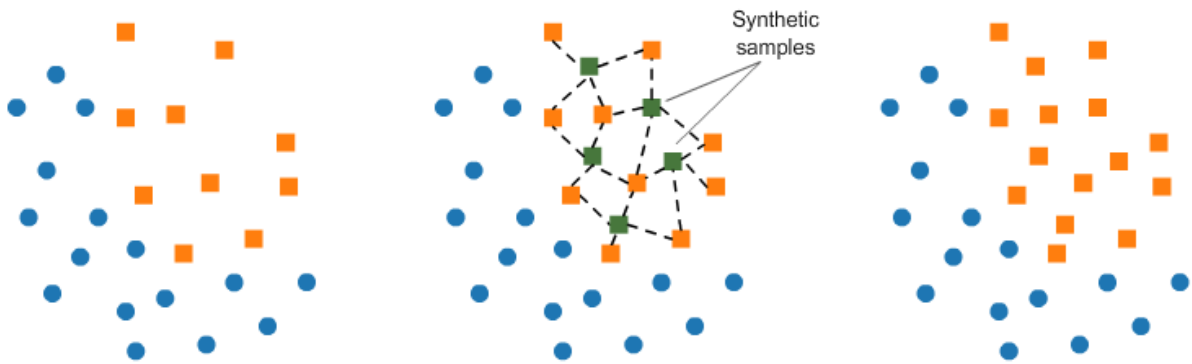
Handling Data Imbalance

Methods to handle data imbalance are:

- Undersampling the majority class
- Oversampling the minority class (by duplicating items)
- Oversampling minority class (using SMOTE - Synthetic Minority Oversamping Technique, by K Nearest Neighbors algorithm)
- Ensemble Method
- Focal Loss

SMOTE (Synthetic Minority Oversampling Technique)

- It aims to balance class distribution by randomly increasing minority class examples by replicating them.
- SMOTE synthesises new minority instances between existing minority instances. It generates the virtual training records by linear interpolation for the minority class.
- These synthetic training records are generated by randomly selecting one or more of the k-nearest neighbors for each example in the minority class.
- SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line.



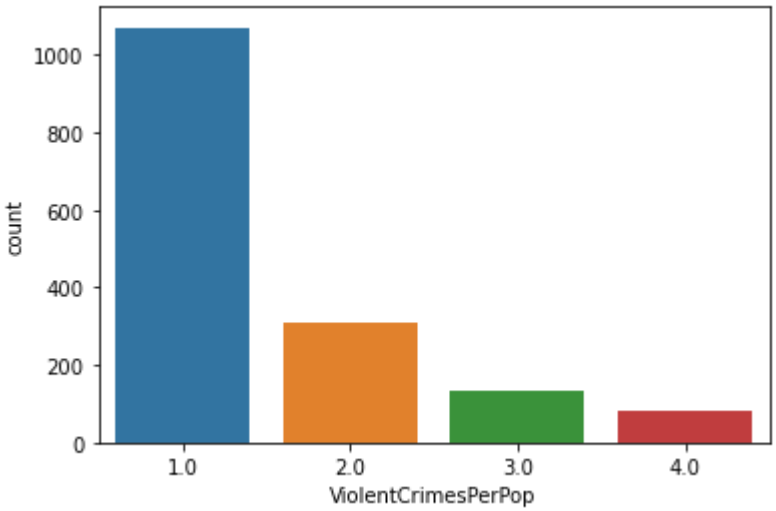
In [38]:
```
train_set["ViolentCrimesPerPop"].value_counts()
```

Out[38]:
```
1.0    1071
2.0     307
```

```
3.0     136
4.0      82
Name: ViolentCrimesPerPop, dtype: int64
```

Imbalanced Train Data

In [39]:
```
sns.countplot(x='ViolentCrimesPerPop',data = train_set)
plt.show()
```



In [40]:
```
pip install imblearn
```

```
Requirement already satisfied: imblearn in /opt/anaconda3/lib/python3.9/site-
packages (0.0)
Requirement already satisfied: imbalanced-learn in /opt/anaconda3/lib/python
3.9/site-packages (from imblearn) (0.11.0)
Requirement already satisfied: numpy>=1.17.3 in /opt/anaconda3/lib/python3.9/
site-packages (from imbalanced-learn->imblearn) (1.20.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/anaconda3/lib/pyt
hon3.9/site-packages (from imbalanced-learn->imblearn) (2.2.0)
Requirement already satisfied: scikit-learn>=1.0.2 in /opt/anaconda3/lib/pyth
on3.9/site-packages (from imbalanced-learn->imblearn) (1.3.2)
Requirement already satisfied: joblib>=1.1.1 in /opt/anaconda3/lib/python3.9/
site-packages (from imbalanced-learn->imblearn) (1.3.2)
Requirement already satisfied: scipy>=1.5.0 in /opt/anaconda3/lib/python3.9/s
ite-packages (from imbalanced-learn->imblearn) (1.7.1)

[notice] A new release of pip available: 22.2.2 -> 23.3.1
[notice] To update, run: pip3.9 install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

In [41]:
```
import collections
x = train_set.drop(columns='ViolentCrimesPerPop')
y = train_set["ViolentCrimesPerPop"]
counter = collections.Counter(y)
print(counter.items())
```

```
dict_items([(1.0, 1071), (2.0, 307), (3.0, 136), (4.0, 82)])
```

In [42]:
```
from imblearn.over_sampling import SMOTE
smote = SMOTE()
x_smt, y_smt = smote.fit_resample(x,y)
y_smt = pd.DataFrame(y_smt)
```

In [43]:
```
# updating the train_set, to a balanced train_set after using smote
train_set_smote = pd.concat([x_smt,y_smt],axis=1)
train_set = train_set_smote
print(len(train_set))
```

4284

In [44]:
```
train_set
```

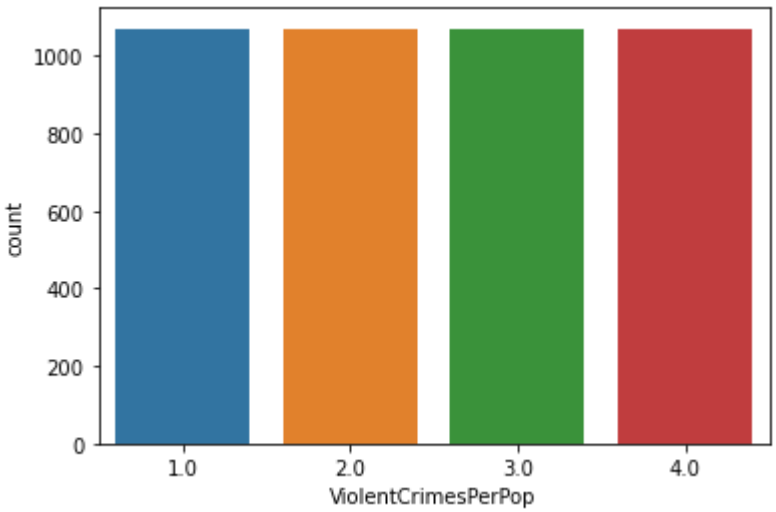
Out[44]:

	state	fold	population	householdsize	racepctblack	racePctWhite	racePctAsian	raceP
0	6	4	0.050000	0.510000	0.040000	0.640000	1.000000	0.
1	29	10	0.050000	0.320000	0.940000	0.220000	0.130000	0.
2	21	9	0.020000	0.490000	0.380000	0.600000	0.250000	0.
3	45	5	0.000000	0.510000	0.200000	0.830000	0.040000	0.
4	41	8	0.160000	0.430000	0.030000	0.870000	0.150000	0.
...	...	...	...	...	...	...	...	
4279	12	3	0.013445	0.377895	0.538293	0.495518	0.038963	0.
4280	12	4	0.025954	0.235861	0.626326	0.479069	0.030837	0.
4281	11	6	0.551534	0.277254	0.791017	0.256479	0.072003	0.
4282	1	4	0.014564	0.450000	0.825235	0.276711	0.060973	0.
4283	37	3	0.011485	0.301485	0.837623	0.329654	0.020495	0.

4284 rows × 103 columns

Balanced Train Data

In [45]:
```
# now we have balanced train_set
sns.countplot(x='ViolentCrimesPerPop',data = train_set)
plt.show()
```



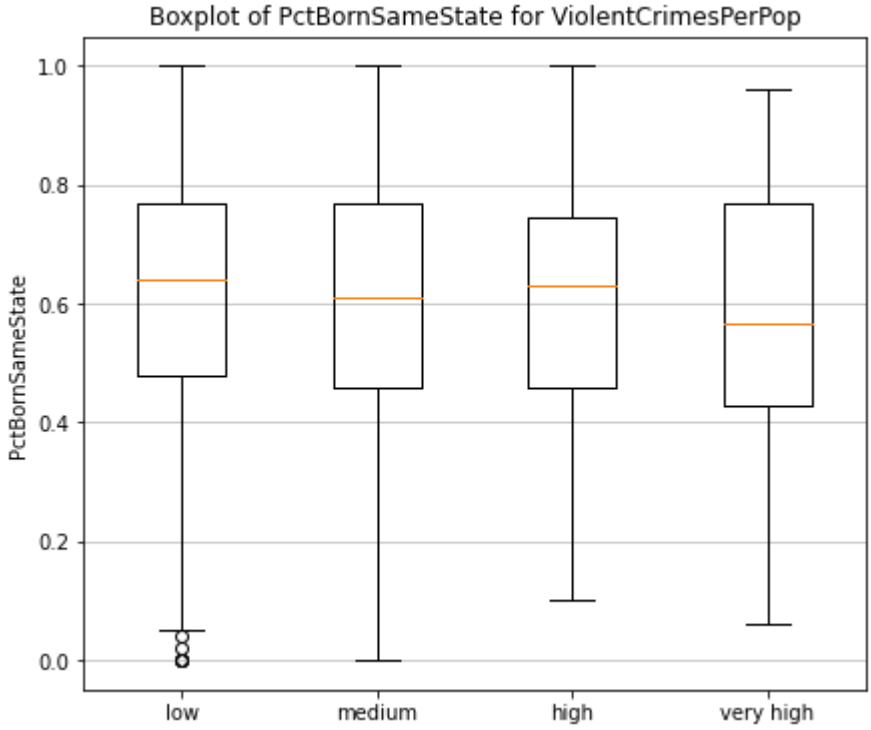
Box Plots for checking correlation between numerical features and target

- Box-plot interpretation, these plots gives us idea about distribution of numerical features for each of the classes in target variable, if the distribution looks similar that means the numerical feature has no effect on target variable, hence numerical feature is not correlated to target feature.
- We can see that for feature "PctBornSameState" rectangular boxes are in similar lines. It means whether the crime rate was low or high is not effected by this feature.
- Whereas for features "PctKids2Par" and "PctIlleg" we can see that rectangular boxes are in different lines, which affect the crime rate.
- We will find which all features to drop after performing ANOVA Test.

In [46]:
```
def make_box_plot(dataset,feature):
    low_crime_rate = dataset[dataset['ViolentCrimesPerPop'] == 1][feature]
    medium_crime_rate = dataset[dataset['ViolentCrimesPerPop'] == 2][feature]
    high_crime_rate = dataset[dataset['ViolentCrimesPerPop'] == 3][feature]
    very_high_crime_rate = dataset[dataset['ViolentCrimesPerPop'] == 4][featu

    fig = plt.figure(figsize =(7,6))
    fig = plt.figure(figsize =(7,6))
    fig, ax1 = plt.subplots(figsize=(7, 6))
    ax1.yaxis.grid(True, linestyle='-', which='major', color='grey',alpha=0.5

    plt.boxplot([low_crime_rate, medium_crime_rate, high_crime_rate, very_hig
    plt.ylabel(feature)
    plt.title(f"Boxplot of {feature} for ViolentCrimesPerPop")
    plt.show()
```

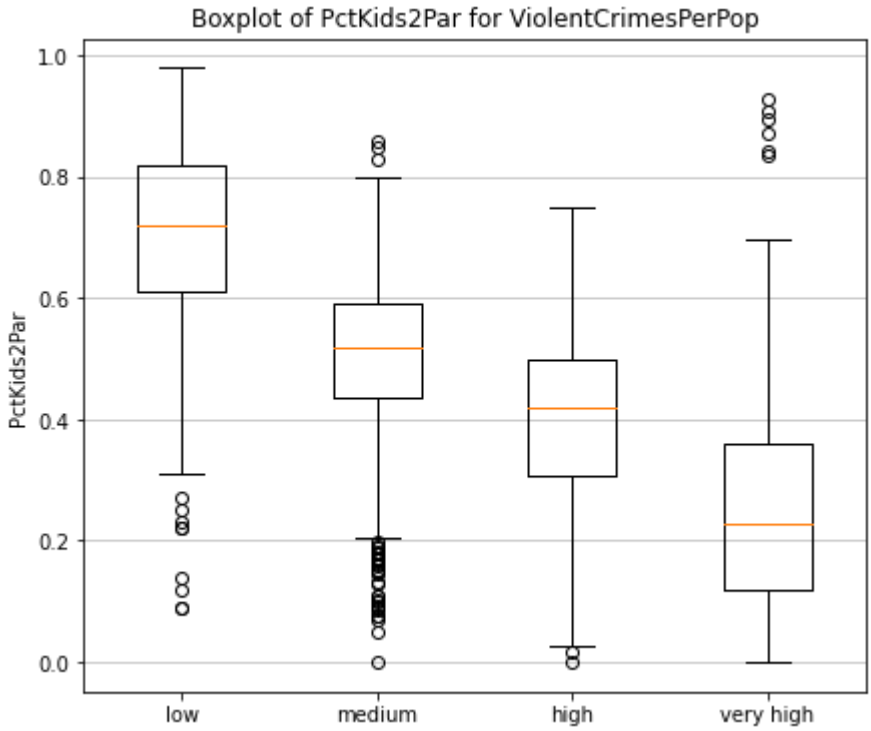
In [47]:
```
make_box_plot(train_set,"PctBornSameState")
```

```
<Figure size 504x432 with 0 Axes>
<Figure size 504x432 with 0 Axes>
```



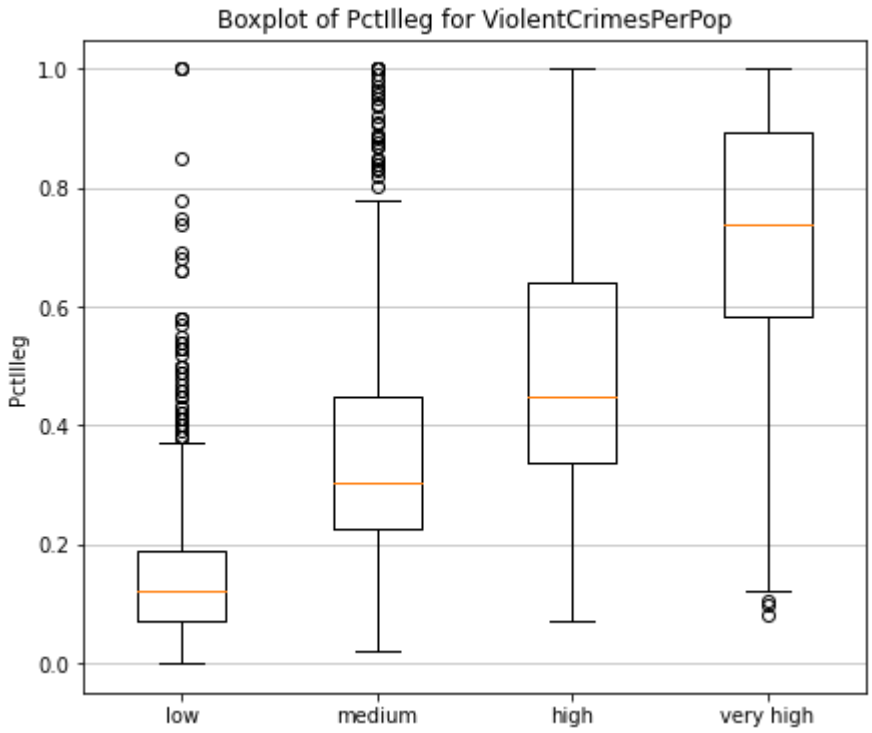
In [48]:
```
make_box_plot(train_set,"PctKids2Par")
```

```
<Figure size 504x432 with 0 Axes>
<Figure size 504x432 with 0 Axes>
```



In [49]:
```
make_box_plot(train_set,"PctIlleg")
```

```
<Figure size 504x432 with 0 Axes>
<Figure size 504x432 with 0 Axes>
```



Feature Selection

ANOVA Test

ANOVA (Analysis of Variance) Test is used for feature selection when input variables are numerical and output variable is categorical.

The variance of a feature determines how much it is impacting the response variable. If the variance is low, it implies there is no impact of this feature on response and vice-versa.

F-Distribution is a probability distribution generally used for the analysis of variance.

H0: Two variances are equal

H1: Two variances are not equal

F-value is the ratio of two Chi-distributions divided by its degrees of Freedom.

$$F = (\chi_1^2/n_1 - 1)/(\chi_2^2/n_2 - 1) \tag{1}$$

where χ_1^2 and χ_2^2 are chi distributions and n_1, n_2 are its respective degrees of freedom

Analysis of Variance is a statistical method, used to check the means of two or more groups that are significantly different from each other. It assumes Hypothesis as

H0: Means of all groups are equal (There is no relation between the given variables as the mean values of the numeric Predictor variable is same for all the groups in the categorical target variable)

H1: At least one mean of the groups are different (There is some relation between given variables)

SST represents the total variation in the data. It is calculated as the sum of the squared differences between each data point and the overall mean of the data.

$$\text{SST} = \sum(y_i - \bar{y})^2$$

SSB represents the variation between different groups or categories in your data. It is calculated as the sum of the squared differences between the group means and the overall mean, weighted by the number of data points in each group.

$$\text{SSB} = \sum(n_j(\bar{y}_j - \bar{y})^2)$$

SSE represents the variation within each group or category. It is calculated as the sum of the squared differences between individual data points and their respective group means.

$$\text{SSE} = \sum\sum((y_{ij} - \bar{y}_j)^2)$$

In [50]:

```
# defining anova function
def anova_feature_selection(X,y):
    n_features = X.shape[1]
    f_scores = [] # intializing array to store f value for each feature

    unique_classes = np.unique(y)
    n_classes = len(unique_classes)

    # to find value of f score for each feature
    for i in range(n_features):
        # feature_values is a column vector
        feature_values = [X.iloc[j][i] for j in range(X.shape[0])]

        ssb = 0
        sse = 0

        for c in unique_classes:
            # it will take datapoints belonging to class c, lets say n_j poin
            class_values = []
            for j in range(len(y)):
                if y[j] == c:
                    class_values.append(feature_values[j])

            # calculating class mean for class c
            class_mean = np.mean(class_values)

            # class size as n_j
            class_size = len(class_values)

            sse += np.sum([(x-class_mean)**2 for x in class_values])
            ssb += class_size * (class_mean - np.mean(feature_values))**2

        # for handling the case when division is undefined 0/0 form
        if sse == 0:
            epsilon = 1e-6
            sse += epsilon

        f_scores.append((ssb/(n_classes-1))/(sse/(X.shape[0]-n_classes)))

    return f_scores
```

In [51]:
```
x = train_set.drop(columns='ViolentCrimesPerPop')
y = train_set["ViolentCrimesPerPop"]
print("Number of numerical features before applying ANOVA test is: ",len(x.co
```

```
Number of numerical features before applying ANOVA test is:  102
```

In [52]:
```
# this will give f-score values for all features
f_scores = anova_feature_selection(x,y)
f_scores[:10]
```

Out[52]:
```
[76.58118946860183,
 23.36058581072567,
 132.81366019612457,
 4.350315701538942,
 1117.506807059188,
 1428.9402263284135,
 3.3633478962063603,
 92.47510547333113,
 16.40125476728194,
 46.7918024053763]
```

In [53]:
```
import json
f_scores = pd.Series(f_scores,index=x.columns)
f_scores_dict = {}
for ind in range(len(f_scores)):
    f_scores_dict[train_set.columns[ind]] = f_scores[ind]
f_scores_dict = dict(sorted(f_scores_dict.items(), key = lambda x: x[1], reve
print(json.dumps(f_scores_dict, indent = 4))
```

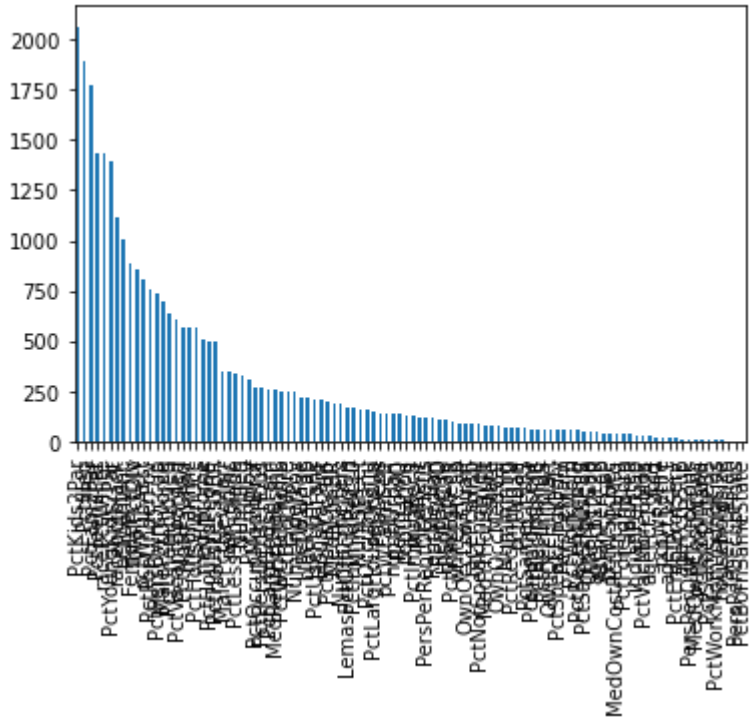
```
{
    "PctKids2Par": 2062.6734998552074,
    "PctIlleg": 1889.802749004127,
    "PctFam2Par": 1776.5628939077963,
    "PctTeen2Par": 1430.4483492699953,
    "racePctWhite": 1428.9402263284135,
    "PctYoungKids2Par": 1392.2221322583669,
    "racepctblack": 1117.506807059188,
    "pctWInvInc": 1001.7729522959864,
    "FemalePctDiv": 886.9795748438911,
    "TotalPctDiv": 851.7572031101333,
    "pctWPubAsst": 803.1168632008253,
    "PctPopUnderPov": 758.7820816022087,
    "PctPersOwnOccup": 737.6645872597893,
    "MalePctDivorce": 700.8895663451862,
    "PctUnemployed": 636.019835560002,
    "PctVacantBoarded": 611.738703586488,
    "PctNotHSGrad": 571.0478392979596,
    "medFamInc": 568.5812276901389,
    "PctHousOwnOcc": 565.1810009027955,
    "medIncome": 509.9706000846388,
    "PctHousNoPhone": 499.6131433027497,
    "PctHousLess3BR": 498.76561988545416,
    "MalePctNevMarr": 346.72694964814536,
    "perCapInc": 344.2742656596153,
    "PctLess9thGrade": 341.3734908638974,
    "NumIlleg": 330.51802249746606,
    "PctEmploy": 310.8542382610545,
    "PctOccupMgmtProf": 266.03278313901023,
    "PctPersDenseHous": 264.8855771952574,
    "PctLargHouseFam": 262.5804357125864,
    "MedRentPctHousInc": 257.6530440099757,
```

```
"PctWOFullPlumb": 253.0488372468116,
"PctBSorMore": 244.7831814984691,
"NumUnderPov": 244.57559163483282,
"pctWWage": 220.94176155124714,
"blackPerCap": 216.0341360523966,
"PctUsePubTrans": 213.00046753846112,
"HousVacant": 211.63957010243496,
"PctHousOccup": 195.924580216968,
"MedNumBR": 192.3003558227281,
"NumInShelters": 188.13690185081424,
"LemasPctOfficDrugUn": 169.99454424372837,
"PctImmigRec10": 165.1825011498099,
"NumStreet": 162.78131684990413,
"PctOccupManu": 158.36336156106447,
"PctLargHouseOccup": 149.45542857726508,
"PopDens": 143.50602387516247,
"pctWFarmSelf": 141.33886882995972,
"numbUrban": 137.5965955224334,
"RentLowQ": 135.28130405861216,
"population": 132.81366019612457,
"PctImmigRec8": 125.5795499355094,
"MedRent": 118.24931989062098,
"PersPerRentOccHous": 116.49962249603668,
"RentMedian": 115.89489471440871,
"RentHighQ": 106.9699334771906,
"HispPerCap": 106.02256794032122,
"PctImmigRec5": 99.76006542296791,
"whitePerCap": 93.39039966098518,
"OwnOccLowQuart": 93.30812378903428,
"racePctHisp": 92.47510547333113,
"PctNotSpeakEnglWell": 84.65575068339399,
"MedYrHousBuilt": 79.0316231749009,
"state": 76.58118946860183,
"OwnOccMedVal": 75.85162152186267,
"NumImmig": 69.92226681216125,
"PctRecImmig10": 69.59942212395589,
"pctUrban": 64.96315271133643,
"PctSameCity85": 64.14148953049627,
"PctImmigRecent": 63.548529143608505,
"PctRecImmig8": 63.53562968806284,
"PctRecImmig5": 61.562223978390556,
"OwnOccHiQuart": 60.49674293129975,
"PctSpeakEnglOnly": 59.869387298168974,
"PersPerFam": 56.61995853769951,
"PctWorkMom": 55.29151946934031,
"PctRecentImmig": 54.04731750251996,
"PctSameHouse85": 51.975231613783166,
"OtherPerCap": 47.76673549154781,
"agePct12t29": 46.7918024053763,
"AsianPerCap": 42.49287244557111,
"pctWSocSec": 39.710514221759,
"MedOwnCostPctIncNoMtg": 38.80183618840073,
"PctForeignBorn": 35.06631317343438,
"LandArea": 35.03841887702485,
"indianPerCap": 30.087831770105435,
"PctVacMore6Mos": 29.945443523534266,
"agePct16t24": 24.52305171867993,
"fold": 23.36058581072567,
"pctWRetire": 23.3277033569342,
"agePct12t21": 16.40125476728194,
"PctEmplProfServ": 15.08537526172965,
"agePct65up": 11.002025362780552,
"PersPerOwnOccHous": 9.108092996697955,
"MedOwnCostPctInc": 6.468640881551109,
```

```
    "PctEmplManu": 5.8741326834065735,
    "PctSameState85": 4.797541662351046,
    "PctWorkMomYoungKids": 4.678936285303488,
    "householdsize": 4.350315701538942,
    "racePctAsian": 3.3633478962063603,
    "PersPerOccupHous": 3.3124580906908556,
    "PctBornSameState": 2.220854764788875
}
```

In [54]:

```
# plot showing f values for numerical features
f_scores.sort_values(ascending=False,inplace =True)
f_scores.plot.bar()
plt.show()
```



According to ANOVA test we reject H0 if

$$F_{\text{observed}} \geq F_{\text{critical}} \tag{2}$$

where,

F_{critical} is calculated by taking degree of freedom as k-1,n-k where k is the number

$$F_{\text{critical}} \text{ comes out to be } 2.6069 \tag{4}$$

so for features where,

F_{observed} is greater than or equal to 2.6069 are included, rest of the features are dr

In [55]:

```
# so features with high f values are included,
# and we drop features which have f values less than 2.6069
count_of_drop_features = 0
for f_val in f_scores:
    if f_val < 2.6069:
        count_of_drop_features += 1
```

```
print("Number of features that should be dropped are:",count_of_drop_features
```

```
Number of features that should be dropped are: 1
```

In [56]:

```
num_features_included = len(f_scores) - count_of_drop_features
f_scores = f_scores[:num_features_included]
print("Number of numerical features remaining after applying f test are:",len
```

```
Number of numerical features remaining after applying f test are: 101
```

In [57]:

```
train_set_new = pd.DataFrame()

columns_to_copy = []
# including remaining numerical features
count = 0
for feature in f_scores_dict:
    if count == num_features_included:
        break
    columns_to_copy.append(feature)
    count += 1

train_set_new = pd.concat([train_set[columns_to_copy].copy(),train_set["Viole
train_set = train_set_new
print(train_set.shape)
features_after_f_test = train_set.columns
features_after_f_test
```

Out[57]:

```
(4284, 102)
Index(['PctKids2Par', 'PctIlleg', 'PctFam2Par', 'PctTeen2Par', 'racePctWhit
e',
       'PctYoungKids2Par', 'racepctblack', 'pctWInvInc', 'FemalePctDiv',
       'TotalPctDiv',
       ...
       'agePct65up', 'PersPerOwnOccHous', 'MedOwnCostPctInc', 'PctEmplManu',
       'PctSameState85', 'PctWorkMomYoungKids', 'householdsize',
       'racePctAsian', 'PersPerOccupHous', 'ViolentCrimesPerPop'],
      dtype='object', length=102)
```

In [58]:

```
train_set
```

Out[58]:

	PctKids2Par	PctIlleg	PctFam2Par	PctTeen2Par	racePctWhite	PctYoungKids2Par	rac
0	0.890000	0.060000	0.900000	0.830000	0.640000	0.920000	
1	0.380000	0.630000	0.380000	0.350000	0.220000	0.660000	
2	0.610000	0.130000	0.630000	0.470000	0.600000	0.650000	
3	0.750000	0.130000	0.720000	0.700000	0.830000	0.850000	
4	0.550000	0.250000	0.510000	0.490000	0.870000	0.510000	
...	...	...	...	...	...	...	
4279	0.160335	0.828932	0.175853	0.049298	0.495518	0.144482	
4280	0.114977	0.702977	0.134139	0.000000	0.479069	0.134139	
4281	0.080743	0.838513	0.118740	0.132989	0.256479	0.257739	
4282	0.353691	0.535637	0.372819	0.363792	0.276711	0.350873	
4283	0.234951	0.662822	0.224951	0.187426	0.329654	0.206436	

4284 rows × 102 columns

In [59]:
```
features_after_f_test
```

Out[59]:
```
Index(['PctKids2Par', 'PctIlleg', 'PctFam2Par', 'PctTeen2Par', 'racePctWhit
e',
       'PctYoungKids2Par', 'racepctblack', 'pctWInvInc', 'FemalePctDiv',
       'TotalPctDiv',
       ...
       'agePct65up', 'PersPerOwnOccHous', 'MedOwnCostPctInc', 'PctEmplManu',
       'PctSameState85', 'PctWorkMomYoungKids', 'householdsize',
       'racePctAsian', 'PersPerOccupHous', 'ViolentCrimesPerPop'],
      dtype='object', length=102)
```

Outlier Detection

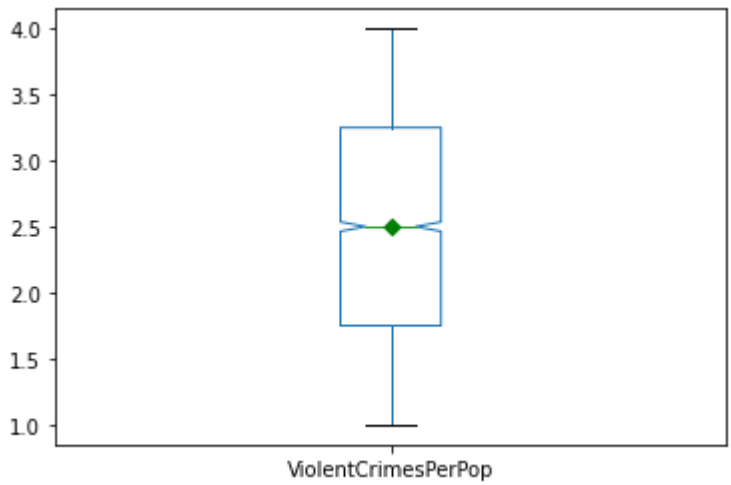
What is an Outlier?

- A data point which is significantly far from other data points
- Inter-Quartile Range Method to remove Outliers (IQR)
- IQR = Q3 - Q1
- Upper_Limit = Q3 + 1.5*IQR
- Lower_Limit = Q1 - 1.5*IQR

In [60]:
```
def plot_boxplot(dataframe,feature):
    red_circle = dict(markerfacecolor='red', marker='o')
    mean_shape = dict(markerfacecolor='green',marker='D',markeredgecolor='gre
    dataframe.boxplot(column=[feature],flierprops = red_circle,showmeans=True
    plt.grid(False)
    plt.show()
```

Plotting Individual Box Plots

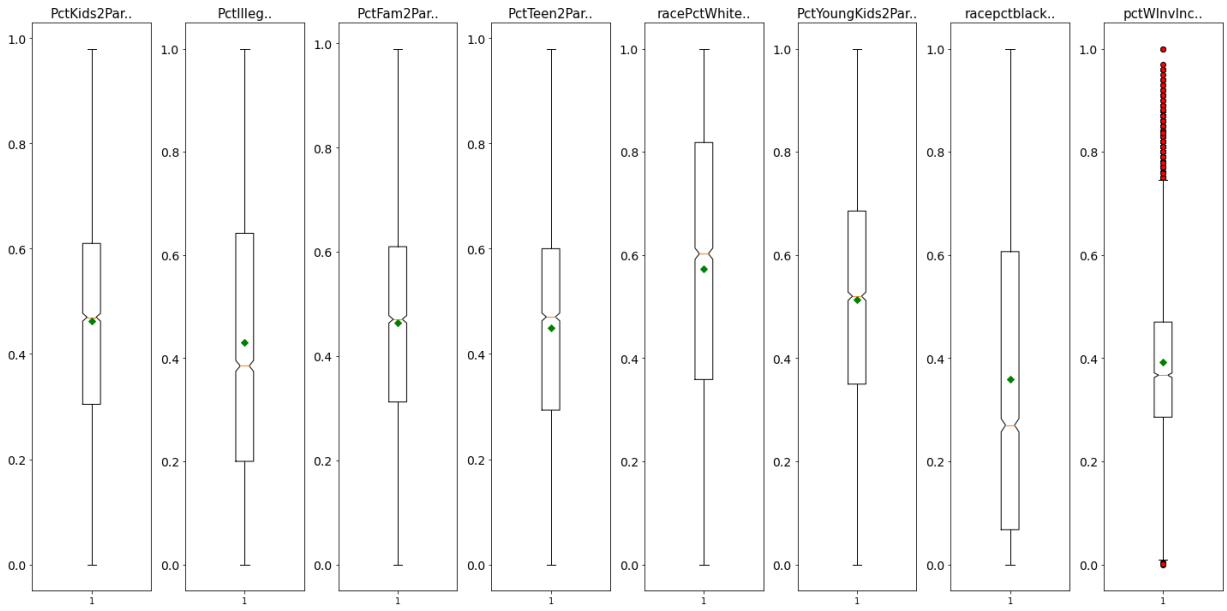
In [61]:
```
plot_boxplot(train_set,"ViolentCrimesPerPop")
```



Plotting Box Plot for multiple features (before outlier removal)

In [62]:
```
# plotting box plots for first 8 features
def plot_boxplot_multiple_features():
    red_circle = dict(markerfacecolor='red', marker='o')
    mean_shape = dict(markerfacecolor='green',marker='D',markeredgecolor='gre
    fig,axis = plt.subplots(1,len(train_set.iloc[:,:8].select_dtypes(include=
    for i,ax in enumerate(axis.flat):
        ax.boxplot(train_set.iloc[:,:8].select_dtypes(include='number').iloc[
        ax.set_title(train_set.iloc[:,:8].select_dtypes(include='number').col
        ax.tick_params(axis='y',labelsize=14)

    plt.tight_layout()
```

In [63]:
```
# red circles are the outliers
plot_boxplot_multiple_features()
```



In [64]:
```
train_set
```

Out[64]:

	PctKids2Par	PctIlleg	PctFam2Par	PctTeen2Par	racePctWhite	PctYoungKids2Par	rac
0	0.890000	0.060000	0.900000	0.830000	0.640000	0.920000	
1	0.380000	0.630000	0.380000	0.350000	0.220000	0.660000	
2	0.610000	0.130000	0.630000	0.470000	0.600000	0.650000	
3	0.750000	0.130000	0.720000	0.700000	0.830000	0.850000	
4	0.550000	0.250000	0.510000	0.490000	0.870000	0.510000	
...	...	...	...	...	...	...	
4279	0.160335	0.828932	0.175853	0.049298	0.495518	0.144482	
4280	0.114977	0.702977	0.134139	0.000000	0.479069	0.134139	
4281	0.080743	0.838513	0.118740	0.132989	0.256479	0.257739	
4282	0.353691	0.535637	0.372819	0.363792	0.276711	0.350873	

	PctKids2Par	PctIlleg	PctFam2Par	PctTeen2Par	racePctWhite	PctYoungKids2Par	rac
4283	0.234951	0.662822	0.224951	0.187426	0.329654	0.206436	

4284 rows × 102 columns

In [65]:
```
# function to return list of indices which are outliers for that feature
def find_outlier_IQR(dataframe,feature):
    q1 = dataframe[feature].quantile(0.25)
    q3 = dataframe[feature].quantile(0.75)
    iqr = q3 - q1
    lower_limit = q1 - 1.5*iqr
    upper_limit = q3 + 1.5*iqr
    outlier_indices = dataframe.index[(dataframe[feature] < lower_limit) | (d
    return outlier_indices
```

In [66]:
```
# creating a list to store indices of outliers, for all features
outlier_index_list = []
for feature in train_set.select_dtypes(include='number').columns:
    # skipping the target variable
    if feature == "ViolentCrimesPerPop":
        continue
    outlier_index_list.extend(find_outlier_IQR(train_set.select_dtypes(includ
```

In [67]:
```
# checking the outlier list
print(len(set(outlier_index_list)))
print(outlier_index_list[:10])
```

```
3238
[0, 12, 29, 53, 73, 74, 80, 82, 83, 102]
```

In [68]:
```
# function to remove outliers and which will return a clean datafram without
def remove_outliers(dataframe,outlier_index_list):
    outlier_index_list = sorted(set(outlier_index_list)) # use a set to remov
    dataframe = dataframe.drop(outlier_index_list)
    return dataframe
```

In [69]:
```
print(len(train_set))
```

```
4284
```

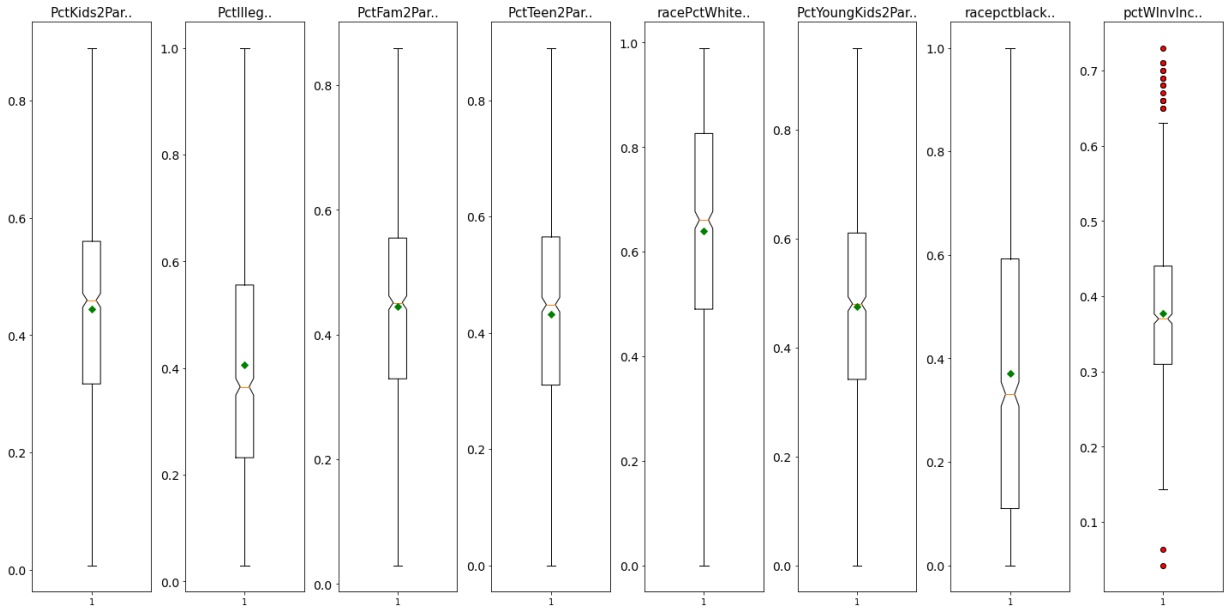
In [70]:
```
train_set = remove_outliers(train_set,outlier_index_list)
```

In [71]:
```
# checking the len after outlier removal
print(len(train_set))
```

```
1046
```

Plotting Box Plot for multiple features (after outlier removal)

In [72]:
```
plot_boxplot_multiple_features() # we can observe the difference now
```



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

Out[73]:
```
(1046, 102)
```

In [74]:
```
train_set.head()
```

Out[74]:

	PctKids2Par	PctIlleg	PctFam2Par	PctTeen2Par	racePctWhite	PctYoungKids2Par	racepct
22	0.52	0.24	0.46	0.59	0.96	0.57	
27	0.69	0.16	0.64	0.70	0.96	0.68	
28	0.49	0.09	0.47	0.42	0.92	0.50	
34	0.67	0.17	0.69	0.67	0.82	0.84	
38	0.38	0.48	0.42	0.43	0.61	0.38	

5 rows × 102 columns

Feature Scaling

Standardization Method

- Standardization is performed to transform the data to have a mean of 0 and standard deviation of 1
- Standardization is also known as Z-Score Normalization

$$z = \frac{(x - \mu)}{\sigma} \tag{6}$$

In [75]:
```
# function for finding mean of a feature in a given dataset
def find_mean(dataset,feature):
    n = len(dataset[feature])
    sum = 0
    for val in dataset[feature]:
        sum += val
    return sum/n
```

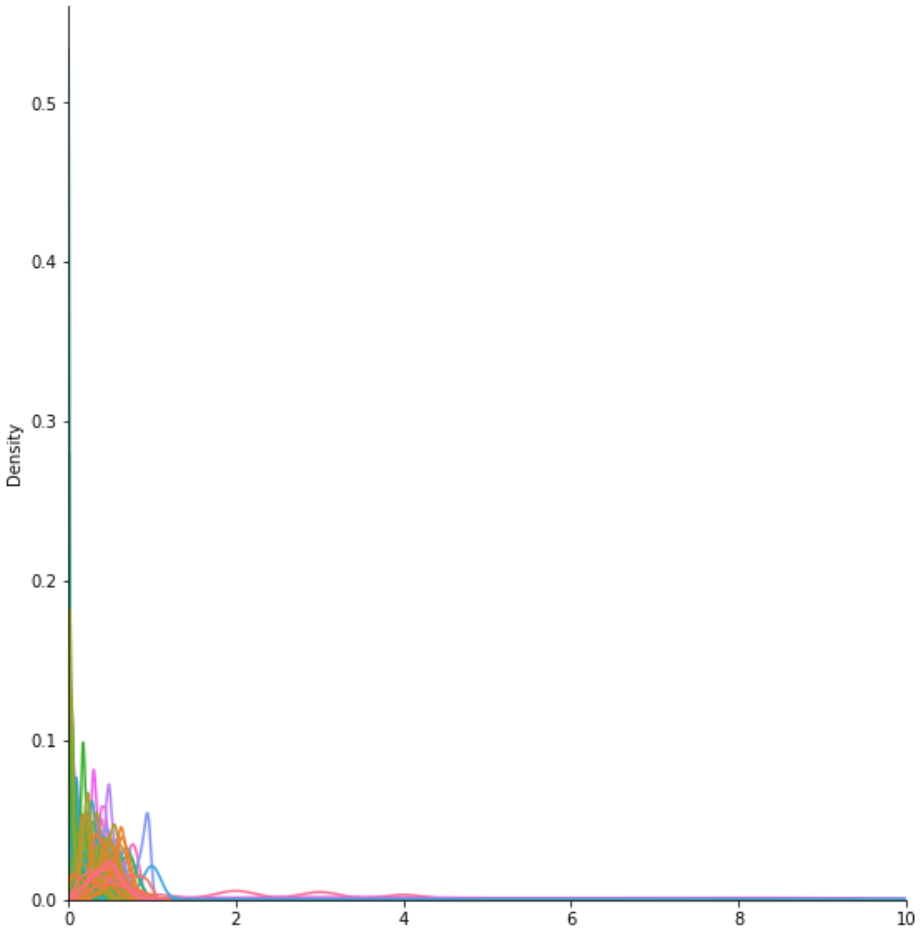
In [76]:
```
# function for finding standard deviation of a feature in a given dataset
def find_standard_deviation(dataset,feature):
    variance, squared_sum = 0,0
    n = len(dataset[feature])
    mean = find_mean(dataset,feature)
    for val in dataset[feature]:
        squared_sum += (val-mean)**2
    variance = squared_sum/n
    return math.sqrt(variance)
```

In [77]:
```
# function for scaling a feature in given dataset
def standardize_feature(dataset,feature,mean_value,standard_deviation_value):
    # to check if mean and standard deviation values are given incase of test
    if mean_value == -1:
        mean_value = find_mean(dataset,feature)
    if standard_deviation_value == -1:
        standard_deviation_value = find_standard_deviation(dataset,feature)

    standardized_feature = []
    for val in dataset[feature]:
        if standard_deviation_value == 0:
            standard_deviation_value = 1e6
        standardized_feature.append((val-mean_value)/standard_deviation_value
    return standardized_feature, mean_value, standard_deviation_value
```

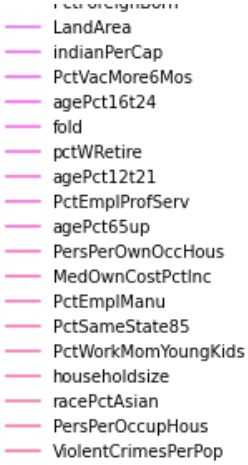
In [78]:
```
# function for scaling (standardizing) the whole dataset
def standardize_dataset(dataset,mean_array,standard_deviation_array):
    standardized_df = pd.DataFrame()
    mean_calculated = []
    standard_deviation_calculated = []
    for ind in range(len(dataset.columns)):
        standardized_result, m, sd = standardize_feature(dataset,dataset.colu
        standardized_df[dataset.columns[ind]] = standardized_result
        mean_calculated.append(m)
        standard_deviation_calculated.append(sd)
    return standardized_df, mean_calculated, standard_deviation_calculated
```

Plot showing distribution of features before standardization

In [79]:
```
x_axis_limits = (0,10)
sns.displot(train_set.select_dtypes(include='number'), kind='kde',aspect=1,he
plt.show()
```



PctKids2Par
PctIlleg
PctFam2Par
PctTeen2Par
racePctWhite
PctYoungKids2Par
racepctblack
pctWInvInc
FemalePctDiv
TotalPctDiv
pctWPubAsst
PctPopUnderPov
PctPersOwnOccup
MalePctDivorce
PctUnemployed
PctVacantBoarded
PctNotHSGrad
medFamInc
PctHousOwnOcc
medIncome
PctHousNoPhone
PctHousLess3BR
MalePctNevMarr
perCapInc
PctLess9thGrade
NumIlleg
PctEmploy
PctOccupMgmtProf
PctPersDenseHous
PctLargHouseFam
MedRentPctHousInc
PctWOFullPlumb
PctBSorMore
NumUnderPov
pctWWage
blackPerCap
PctUsePubTrans
HousVacant
PctHousOccup
MedNumBR
NumInShelters
LemasPctOfficDrugUn
PctImmigRec10
NumStreet
PctOccupManu
PctLargHouseOccup
PopDens
pctWFarmSelf
numbUrban
RentLowQ
population
PctImmigRec8
MedRent
PersPerRentOccHous
RentMedian
RentHighQ
HispPerCap
PctImmigRec5
whitePerCap
OwnOccLowQuart
racePctHisp
PctNotSpeakEnglWell
MedYrHousBuilt
state
OwnOccMedVal
NumImmig
PctRecImmig10
pctUrban
PctSameCity85
PctImmigRecent
PctRecImmig8
PctRecImmig5
OwnOccHiQuart
PctSpeakEnglOnly
PersPerFam
PctWorkMom
PctRecentImmig
PctSameHouse85
OtherPerCap
agePct12t29
AsianPerCap
pctWSocSec
MedOwnCostPctIncNoMtg
PctForeignBorn



Standardizing the train dataset

In [80]:
```
import warnings
with warnings.catch_warnings():
    warnings.filterwarnings("ignore", category=pd.errors.PerformanceWarning)
with warnings.catch_warnings():
    warnings.filterwarnings("ignore", category=np.VisibleDeprecationWarning)
```

In [81]:
```
# standardizing the train dataset
train_set_new = train_set.select_dtypes(include='number')
train_set_new = train_set_new.drop(columns = ['ViolentCrimesPerPop'])
train_mean_array = [-1 for i in range(len(train_set_new.columns))]
train_standard_deviation_array = [-1 for i in range(len(train_set_new.columns
train_set_new, mean_calculated, standard_deviation_calculated = standardize_d
train_set_new
```

```
/var/folders/69/1b088435637d6qzmvv7zpycm0000gn/T/ipykernel_29795/2610170990.p
y:8: PerformanceWarning: DataFrame is highly fragmented.  This is usually the
result of calling `frame.insert` many times, which has poor performance.  Con
sider joining all columns at once using pd.concat(axis=1) instead.  To get a
de-fragmented frame, use `newframe = frame.copy()`
  standardized_df[dataset.columns[ind]] = standardized_result
```

Out[81]:

	PctKids2Par	PctIlleg	PctFam2Par	PctTeen2Par	racePctWhite	PctYoungKids2Par	rac
0	0.438410	-0.748501	0.091621	0.917361	1.428624	0.490924	
1	1.428034	-1.109890	1.187933	1.556774	1.428624	1.061555	
2	0.263770	-1.426106	0.152527	-0.070822	1.249895	0.127794	
3	1.311607	-1.064717	1.492464	1.382389	0.803070	1.891565	
4	-0.376574	0.335666	-0.152004	-0.012694	-0.135263	-0.494713	
...	...	...	...	...	...	...	
1041	-1.788567	0.826012	-1.815525	-1.774731	-0.546857	-1.621810	
1042	-1.371703	1.017621	-1.326396	-1.449515	-0.531849	-1.635976	
1043	-1.655313	1.911917	-1.639009	-2.225653	-0.646795	-1.716479	
1044	-0.529726	0.586999	-0.439368	-0.397553	-1.624482	-0.645813	
1045	-1.220952	1.161537	-1.339976	-1.422739	-1.387919	-1.395088	

1046 rows × 101 columns

In [82]:
```
# checking mean and variance of each feature after standardizing the train da
for feature in train_set_new:
    print("Mean of",feature,"is",round(find_mean(train_set_new,feature)))
    print("Standard Deviation of",feature,"is",round(find_standard_deviation(
```

```
Mean of PctKids2Par is 0
Standard Deviation of PctKids2Par is 1
Mean of PctIlleg is 0
Standard Deviation of PctIlleg is 1
Mean of PctFam2Par is 0
Standard Deviation of PctFam2Par is 1
Mean of PctTeen2Par is 0
Standard Deviation of PctTeen2Par is 1
Mean of racePctWhite is 0
Standard Deviation of racePctWhite is 1
Mean of PctYoungKids2Par is 0
Standard Deviation of PctYoungKids2Par is 1
Mean of racepctblack is 0
Standard Deviation of racepctblack is 1
Mean of pctWInvInc is 0
Standard Deviation of pctWInvInc is 1
Mean of FemalePctDiv is 0
Standard Deviation of FemalePctDiv is 1
Mean of TotalPctDiv is 0
Standard Deviation of TotalPctDiv is 1
Mean of pctWPubAsst is 0
Standard Deviation of pctWPubAsst is 1
Mean of PctPopUnderPov is 0
Standard Deviation of PctPopUnderPov is 1
Mean of PctPersOwnOccup is 0
Standard Deviation of PctPersOwnOccup is 1
Mean of MalePctDivorce is 0
Standard Deviation of MalePctDivorce is 1
Mean of PctUnemployed is 0
Standard Deviation of PctUnemployed is 1
Mean of PctVacantBoarded is 0
Standard Deviation of PctVacantBoarded is 1
Mean of PctNotHSGrad is 0
Standard Deviation of PctNotHSGrad is 1
Mean of medFamInc is 0
Standard Deviation of medFamInc is 1
Mean of PctHousOwnOcc is 0
Standard Deviation of PctHousOwnOcc is 1
Mean of medIncome is 0
Standard Deviation of medIncome is 1
Mean of PctHousNoPhone is 0
Standard Deviation of PctHousNoPhone is 1
Mean of PctHousLess3BR is 0
Standard Deviation of PctHousLess3BR is 1
Mean of MalePctNevMarr is 0
Standard Deviation of MalePctNevMarr is 1
Mean of perCapInc is 0
Standard Deviation of perCapInc is 1
Mean of PctLess9thGrade is 0
Standard Deviation of PctLess9thGrade is 1
Mean of NumIlleg is 0
Standard Deviation of NumIlleg is 1
Mean of PctEmploy is 0
Standard Deviation of PctEmploy is 1
```

```
Mean of PctOccupMgmtProf is 0
Standard Deviation of PctOccupMgmtProf is 1
Mean of PctPersDenseHous is 0
Standard Deviation of PctPersDenseHous is 1
Mean of PctLargHouseFam is 0
Standard Deviation of PctLargHouseFam is 1
Mean of MedRentPctHousInc is 0
Standard Deviation of MedRentPctHousInc is 1
Mean of PctWOFullPlumb is 0
Standard Deviation of PctWOFullPlumb is 1
Mean of PctBSorMore is 0
Standard Deviation of PctBSorMore is 1
Mean of NumUnderPov is 0
Standard Deviation of NumUnderPov is 1
Mean of pctWWage is 0
Standard Deviation of pctWWage is 1
Mean of blackPerCap is 0
Standard Deviation of blackPerCap is 1
Mean of PctUsePubTrans is 0
Standard Deviation of PctUsePubTrans is 1
Mean of HousVacant is 0
Standard Deviation of HousVacant is 1
Mean of PctHousOccup is 0
Standard Deviation of PctHousOccup is 1
Mean of MedNumBR is 0
Standard Deviation of MedNumBR is 1
Mean of NumInShelters is 0
Standard Deviation of NumInShelters is 1
Mean of LemasPctOfficDrugUn is 0
Standard Deviation of LemasPctOfficDrugUn is 1
Mean of PctImmigRec10 is 0
Standard Deviation of PctImmigRec10 is 1
Mean of NumStreet is 0
Standard Deviation of NumStreet is 1
Mean of PctOccupManu is 0
Standard Deviation of PctOccupManu is 1
Mean of PctLargHouseOccup is 0
Standard Deviation of PctLargHouseOccup is 1
Mean of PopDens is 0
Standard Deviation of PopDens is 1
Mean of pctWFarmSelf is 0
Standard Deviation of pctWFarmSelf is 1
Mean of numbUrban is 0
Standard Deviation of numbUrban is 1
Mean of RentLowQ is 0
Standard Deviation of RentLowQ is 1
Mean of population is 0
Standard Deviation of population is 1
Mean of PctImmigRec8 is 0
Standard Deviation of PctImmigRec8 is 1
Mean of MedRent is 0
Standard Deviation of MedRent is 1
Mean of PersPerRentOccHous is 0
Standard Deviation of PersPerRentOccHous is 1
Mean of RentMedian is 0
Standard Deviation of RentMedian is 1
Mean of RentHighQ is 0
Standard Deviation of RentHighQ is 1
Mean of HispPerCap is 0
Standard Deviation of HispPerCap is 1
Mean of PctImmigRec5 is 0
Standard Deviation of PctImmigRec5 is 1
Mean of whitePerCap is 0
Standard Deviation of whitePerCap is 1
```

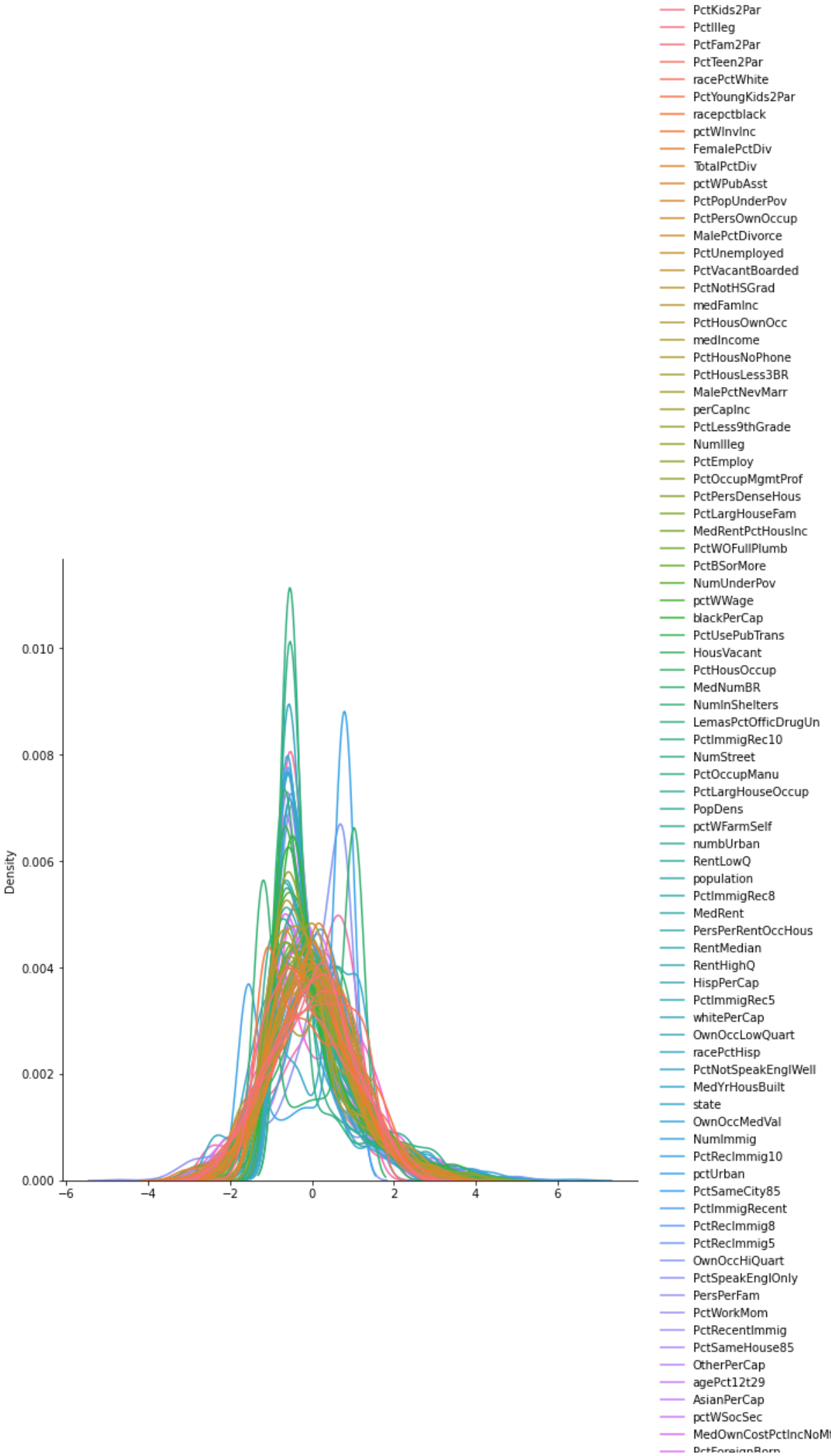
```
Mean of OwnOccLowQuart is 0
Standard Deviation of OwnOccLowQuart is 1
Mean of racePctHisp is 0
Standard Deviation of racePctHisp is 1
Mean of PctNotSpeakEnglWell is 0
Standard Deviation of PctNotSpeakEnglWell is 1
Mean of MedYrHousBuilt is 0
Standard Deviation of MedYrHousBuilt is 1
Mean of state is 0
Standard Deviation of state is 1
Mean of OwnOccMedVal is 0
Standard Deviation of OwnOccMedVal is 1
Mean of NumImmig is 0
Standard Deviation of NumImmig is 1
Mean of PctRecImmig10 is 0
Standard Deviation of PctRecImmig10 is 1
Mean of pctUrban is 0
Standard Deviation of pctUrban is 1
Mean of PctSameCity85 is 0
Standard Deviation of PctSameCity85 is 1
Mean of PctImmigRecent is 0
Standard Deviation of PctImmigRecent is 1
Mean of PctRecImmig8 is 0
Standard Deviation of PctRecImmig8 is 1
Mean of PctRecImmig5 is 0
Standard Deviation of PctRecImmig5 is 1
Mean of OwnOccHiQuart is 0
Standard Deviation of OwnOccHiQuart is 1
Mean of PctSpeakEnglOnly is 0
Standard Deviation of PctSpeakEnglOnly is 1
Mean of PersPerFam is 0
Standard Deviation of PersPerFam is 1
Mean of PctWorkMom is 0
Standard Deviation of PctWorkMom is 1
Mean of PctRecentImmig is 0
Standard Deviation of PctRecentImmig is 1
Mean of PctSameHouse85 is 0
Standard Deviation of PctSameHouse85 is 1
Mean of OtherPerCap is 0
Standard Deviation of OtherPerCap is 1
Mean of agePct12t29 is 0
Standard Deviation of agePct12t29 is 1
Mean of AsianPerCap is 0
Standard Deviation of AsianPerCap is 1
Mean of pctWSocSec is 0
Standard Deviation of pctWSocSec is 1
Mean of MedOwnCostPctIncNoMtg is 0
Standard Deviation of MedOwnCostPctIncNoMtg is 1
Mean of PctForeignBorn is 0
Standard Deviation of PctForeignBorn is 1
Mean of LandArea is 0
Standard Deviation of LandArea is 1
Mean of indianPerCap is 0
Standard Deviation of indianPerCap is 1
Mean of PctVacMore6Mos is 0
Standard Deviation of PctVacMore6Mos is 1
Mean of agePct16t24 is 0
Standard Deviation of agePct16t24 is 1
Mean of fold is 0
Standard Deviation of fold is 1
Mean of pctWRetire is 0
Standard Deviation of pctWRetire is 1
Mean of agePct12t21 is 0
Standard Deviation of agePct12t21 is 1
```

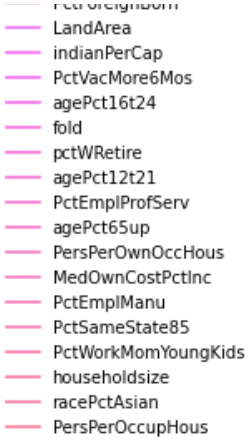
```
Mean of PctEmplProfServ is 0
Standard Deviation of PctEmplProfServ is 1
Mean of agePct65up is 0
Standard Deviation of agePct65up is 1
Mean of PersPerOwnOccHous is 0
Standard Deviation of PersPerOwnOccHous is 1
Mean of MedOwnCostPctInc is 0
Standard Deviation of MedOwnCostPctInc is 1
Mean of PctEmplManu is 0
Standard Deviation of PctEmplManu is 1
Mean of PctSameState85 is 0
Standard Deviation of PctSameState85 is 1
Mean of PctWorkMomYoungKids is 0
Standard Deviation of PctWorkMomYoungKids is 1
Mean of householdsize is 0
Standard Deviation of householdsize is 1
Mean of racePctAsian is 0
Standard Deviation of racePctAsian is 1
Mean of PersPerOccupHous is 0
Standard Deviation of PersPerOccupHous is 1
```

Plot showing distribution of features after standardization

In [83]:

```
# all features following a normal distribution with mean 0 and standard devia
sns.displot(train_set_new, kind='kde',aspect=1,height=8)
plt.show()
```


Density
0.010
0.008
0.006
0.004
0.002
0.000
−6
−4
−2
0
2
4
6
PctKids2Par
PctIlleg
PctFam2Par
PctTeen2Par
racePctWhite
PctYoungKids2Par
racepctblack
pctWInvInc
FemalePctDiv
TotalPctDiv
pctWPubAsst
PctPopUnderPov
PctPersOwnOccup
MalePctDivorce
PctUnemployed
PctVacantBoarded
PctNotHSGrad
medFamInc
PctHousOwnOcc
medIncome
PctHousNoPhone
PctHousLess3BR
MalePctNevMarr
perCapInc
PctLess9thGrade
NumIlleg
PctEmploy
PctOccupMgmtProf
PctPersDenseHous
PctLargHouseFam
MedRentPctHousInc
PctWOFullPlumb
PctBSorMore
NumUnderPov
pctWWage
blackPerCap
PctUsePubTrans
HousVacant
PctHousOccup
MedNumBR
NumInShelters
LemasPctOfficDrugUn
PctImmigRec10
NumStreet
PctOccupManu
PctLargHouseOccup
PopDens
pctWFarmSelf
numbUrban
RentLowQ
population
PctImmigRec8
MedRent
PersPerRentOccHous
RentMedian
RentHighQ
HispPerCap
PctImmigRec5
whitePerCap
OwnOccLowQuart
racePctHisp
PctNotSpeakEnglWell
MedYrHousBuilt
state
OwnOccMedVal
NumImmig
PctRecImmig10
pctUrban
PctSameCity85
PctImmigRecent
PctRecImmig8
PctRecImmig5
OwnOccHiQuart
PctSpeakEnglOnly
PersPerFam
PctWorkMom
PctRecentImmig
PctSameHouse85
OtherPerCap
agePct12t29
AsianPerCap
pctWSocSec
MedOwnCostPctIncNoMtg



In [84]:
```
# replacing standardized features with original features in train_set datafra
train_set_new.index = train_set.index
train_set_new["ViolentCrimesPerPop"] = train_set["ViolentCrimesPerPop"]
train_set = train_set_new
train_set
```

Out[84]:

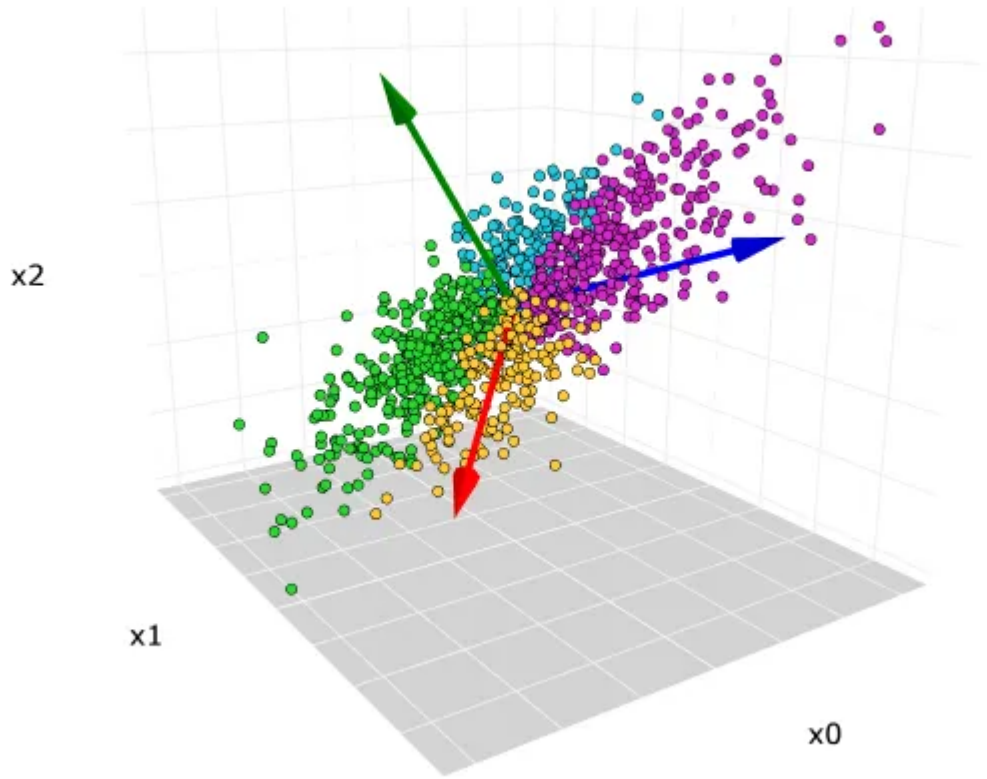
	PctKids2Par	PctIlleg	PctFam2Par	PctTeen2Par	racePctWhite	PctYoungKids2Par	rac
22	0.438410	-0.748501	0.091621	0.917361	1.428624	0.490924	
27	1.428034	-1.109890	1.187933	1.556774	1.428624	1.061555	
28	0.263770	-1.426106	0.152527	-0.070822	1.249895	0.127794	
34	1.311607	-1.064717	1.492464	1.382389	0.803070	1.891565	
38	-0.376574	0.335666	-0.152004	-0.012694	-0.135263	-0.494713	
...	...	...	...	...	...	...	
4263	-1.788567	0.826012	-1.815525	-1.774731	-0.546857	-1.621810	
4275	-1.371703	1.017621	-1.326396	-1.449515	-0.531849	-1.635976	
4279	-1.655313	1.911917	-1.639009	-2.225653	-0.646795	-1.716479	
4282	-0.529726	0.586999	-0.439368	-0.397553	-1.624482	-0.645813	
4283	-1.220952	1.161537	-1.339976	-1.422739	-1.387919	-1.395088	

1046 rows × 102 columns

Dimensionality Reduction using PCA (Principal Component Analysis)

- It is used to reduce the dimensionality of dataset by transforming a large set into a lower dimensional set that still contains most of the information of the large dataset
- Principal component analysis (PCA) is a technique that transforms a dataset of many features into principal components that "summarize" the variance that underlies the data

- PCA finds a new set of dimensions such that all dimensions are orthogonal and hence linearly independent and ranked according to variance of data along them
- Eigen vectors point in direction of maximum variance among data, and eigen value gives the importance of that eigen vector



In [85]:

```
# implementing PCA from scratch

# it will take dataset X and k components needed after PCA
def PCA(X,k):
    k_principal_components = [] # it will store first k eigen vectors
    mean = np.mean(X,axis=0)  # this will find mean for each row
    X = X - mean  # mean centering the data

    # finding the covariance matrix, will give a n*n matrix containing covari
    cov = np.cov(X.T)

    # finding eigenvalues and eigenvectors
    eigenvalues, eigenvectors = np.linalg.eig(cov)

    # transpose eigenvector
    eigenvectors = eigenvectors.T

    # will give indexes according to eigen values, sorted in decreasing order
    idx = np.argsort(eigenvalues)[::-1]
    eigenvalues = eigenvalues[idx]
    eigenvectors = eigenvectors[idx]

    # for finding how much variance does each principal component capture
    explained_variance = eigenvalues / np.sum(eigenvalues)

    # slicing first k eigenvectors
    k_principal_components = eigenvectors[:k]
```

```
    # returning the transformed features
    # multiplyinh n*d matrix with d*k matrix to get transformed feature matri
    return np.dot(X,k_principal_components.T), explained_variance, k_principa

k = 36

# this will return the new dataset
train_set_pca, explained_variance, k_principal_components = PCA(train_set.dro
train_set_pca = pd.DataFrame(train_set_pca)
print("Shape of train_set is:", train_set.shape)
print("Shape of train_set_pca is:",train_set_pca.shape)
```

```
Shape of train_set is: (1046, 102)
Shape of train_set_pca is: (1046, 36)
```

In [86]:

```
train_set_pca
```

Out[86]:

	0	1	2	3	4	5	6	
0	-2.360998	1.846873	4.125858	0.507645	-2.278640	-0.518220	0.150925	0.14495
1	-1.447141	7.959853	0.922674	-0.871256	0.030290	-1.429054	-2.064173	1.79410
2	1.492029	2.875179	2.259540	-3.062838	2.815922	-1.848797	0.599792	-0.50455
3	-7.443684	4.671614	-3.591262	3.252589	0.760099	1.552172	0.823751	-1.89725
4	1.447987	1.602728	-0.167608	0.227237	-1.506626	0.242809	2.333397	1.51630
...	...	...	...	...	...	...	...	.
1041	5.172223	-1.497303	4.050828	-0.934905	2.305401	-0.274013	-0.246995	-2.57287
1042	3.909755	-2.546775	4.609521	-2.184349	1.296757	-1.159601	-0.181275	-2.53174
1043	1.732450	-10.730062	2.054287	-4.327002	5.929301	0.688839	-1.070643	0.51765
1044	2.798917	1.118305	-1.722834	-1.539517	-3.226729	2.049230	-1.029076	-0.02501
1045	4.924808	0.748787	2.177290	-1.908641	0.149234	1.712256	0.664954	-1.79109

1046 rows × 36 columns

In [87]:

```
# plotting first two principal components
PC1 = train_set_pca[0]
PC2 = train_set_pca[1]

target = train_set["ViolentCrimesPerPop"]
label = ["low","medium","high","very high"]

labels = []
for points in target:
    labels.append(label[int(points)-1])

zipped = list(zip(PC1,PC2,target,labels))
pc_df = pd.DataFrame(zipped, columns=['PC1','PC2','Target','Label'])
pc_df
```

Out[87]:

	PC1	PC2	Target	Label
0	-2.360998	1.846873	1.0	low

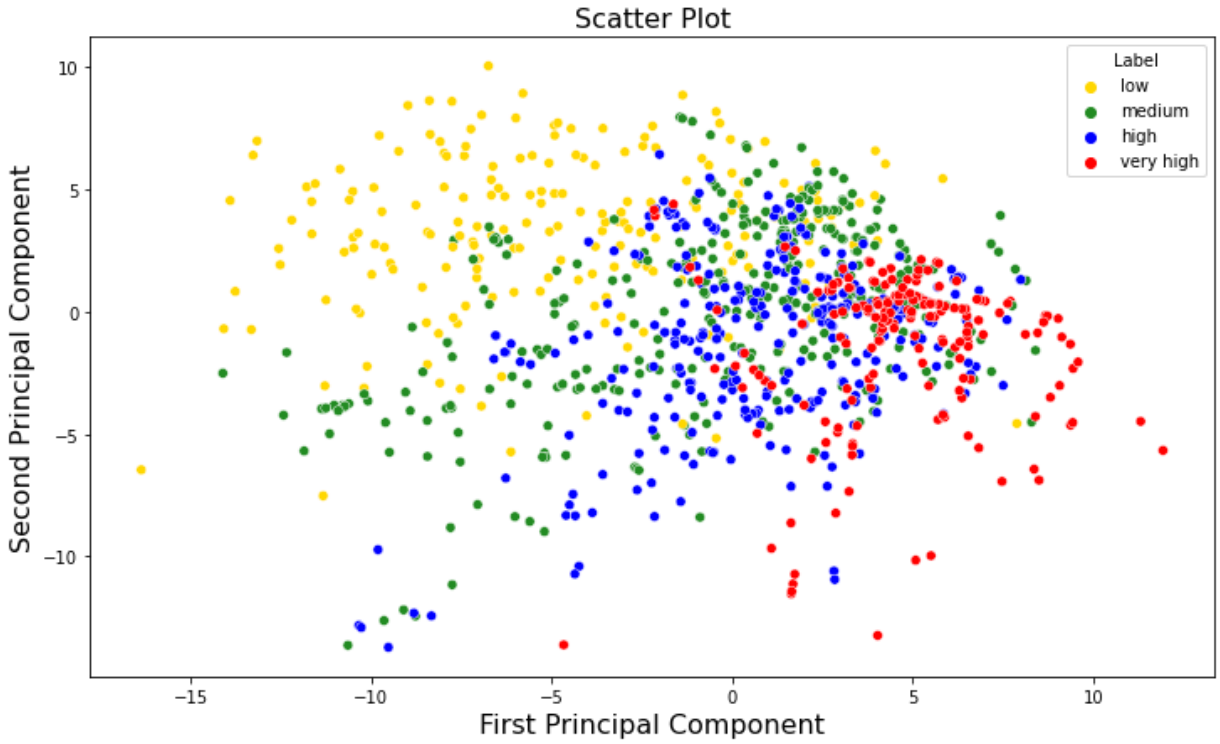
	PC1	PC2	Target	Label
1	-1.447141	7.959853	2.0	medium
2	1.492029	2.875179	1.0	low
3	-7.443684	4.671614	1.0	low
4	1.447987	1.602728	3.0	high
...	...	...	...	...
1041	5.172223	-1.497303	4.0	very high
1042	3.909755	-2.546775	4.0	very high
1043	1.732450	-10.730062	4.0	very high
1044	2.798917	1.118305	4.0	very high
1045	4.924808	0.748787	4.0	very high

1046 rows × 4 columns

Plot showing spread of data along first 2 Principal Components

In [88]:
```
plt.figure(figsize=(12,7))
forest_green = (34/255, 139/255, 34/255)
gold = (255/255,215/255,0/255)
colors = [forest_green if label == 0 else gold for label in target]
sns.scatterplot(data=pc_df,x="PC1",y="PC2",hue="Label",palette={'low':gold,'m
plt.title("Scatter Plot",fontsize=16)
plt.xlabel('First Principal Component',fontsize=16)
plt.ylabel('Second Principal Component',fontsize=16)
plt.show()
```

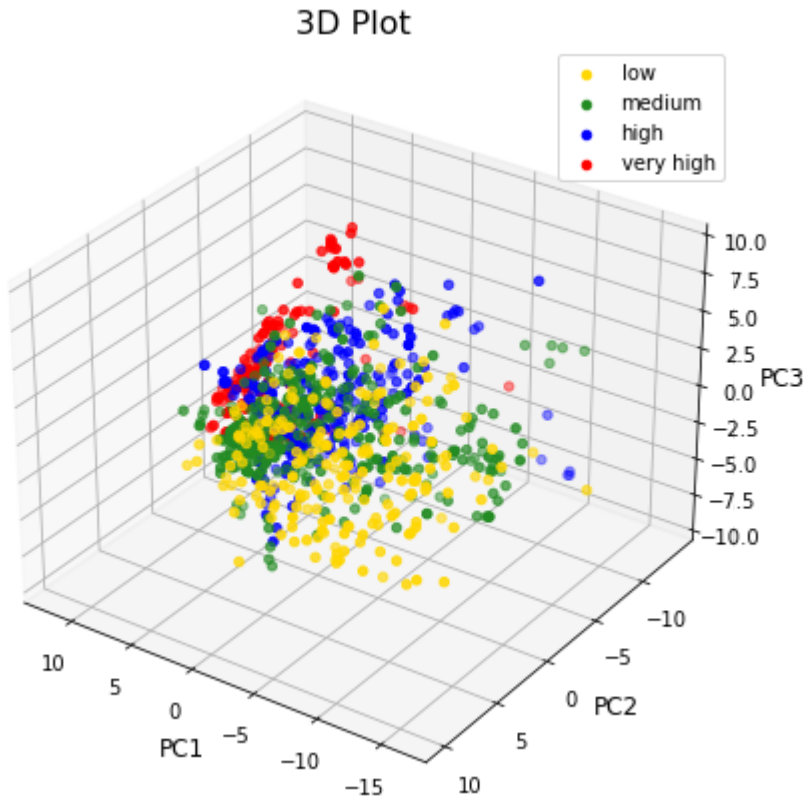
```
/opt/anaconda3/lib/python3.9/site-packages/numpy/core/_asarray.py:102: Visibl
eDeprecationWarning: Creating an ndarray from ragged nested sequences (which
is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or s
hapes) is deprecated. If you meant to do this, you must specify 'dtype=objec
t' when creating the ndarray.
  return array(a, dtype, copy=False, order=order)
```



Plot showing spread of data along first 3 Principal Components

In [89]:

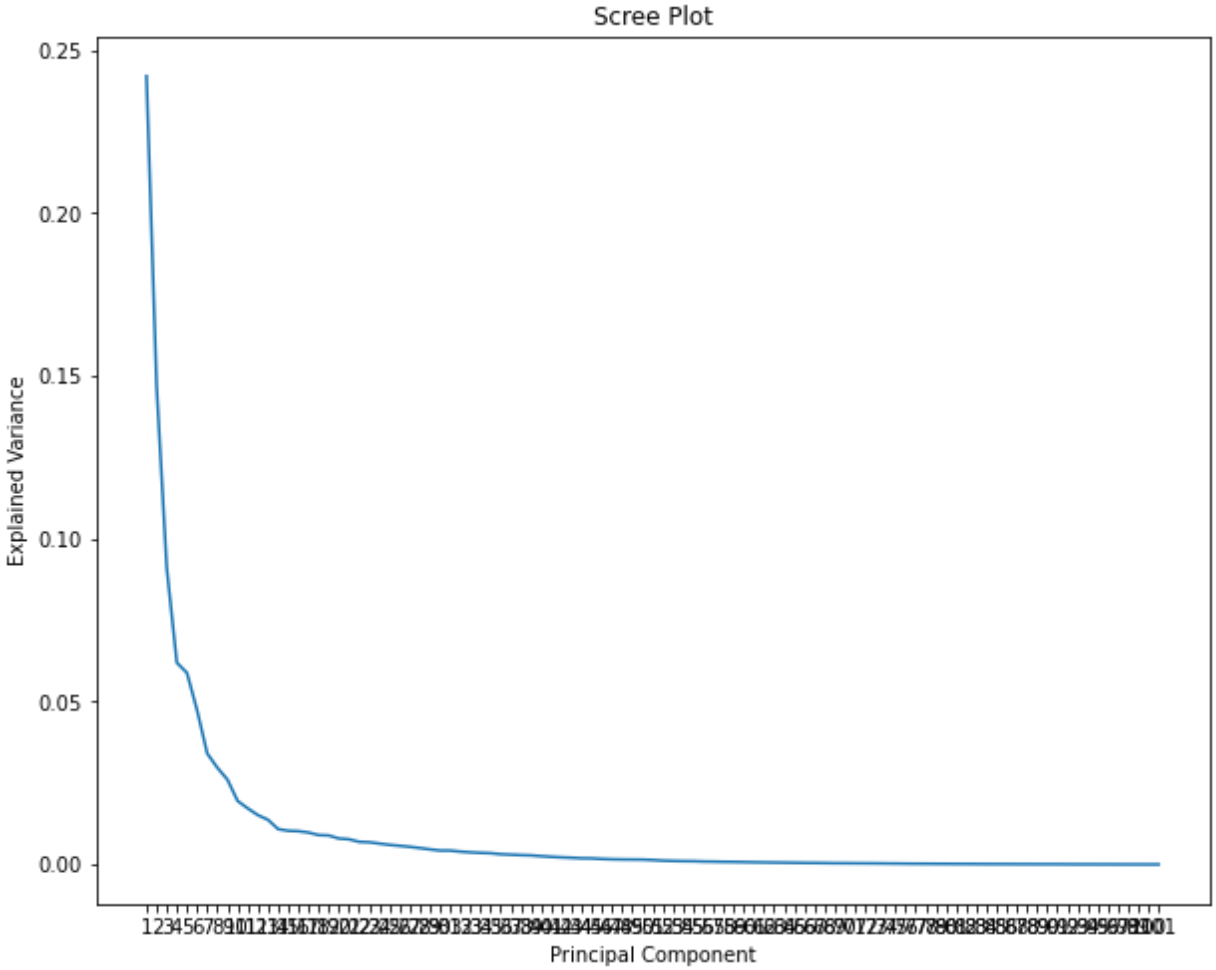
```
PC3 = train_set_pca[2]

fig = plt.figure(figsize=(12,7))
ax = fig.add_subplot(111,projection='3d')

for l in np.unique(target):
    ix = np.where(target==l)
    ix = np.asarray(ix)[0]
    ax.scatter(PC1[ix],PC2[ix],PC3[ix],label=label[int(l)-1],c=[gold if int(l

ax.set_xlabel("PC1",fontsize=12)
ax.set_ylabel("PC2",fontsize=12)
ax.set_zlabel("PC3",fontsize=12)

ax.view_init(30, 125)
ax.legend()
plt.title("3D Plot",fontsize=16)
plt.show()
```



Plot showing variance captured by each Principal Component

In [90]:

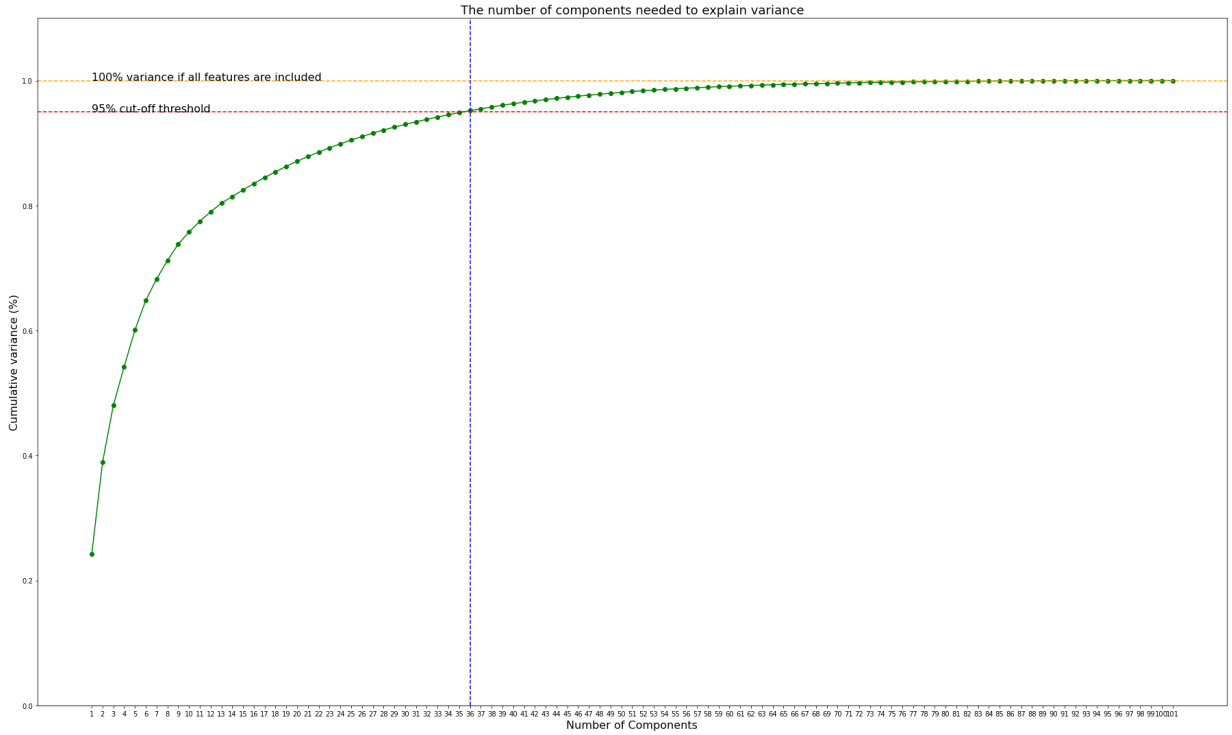
```
num_components = len(explained_variance)
components = np.arange(1, num_components + 1)
plt.figure(figsize=(10, 8))
plt.plot(components, explained_variance)
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance')
plt.title('Scree Plot')
plt.xticks(components)
plt.show()
```



Plot to find out number of Principal Components needed inorder to capture 95% variance in data

In [91]:

```
# finding cumulative variance captured by principal components
y_var = np.cumsum(explained_variance)

plt.figure(figsize=(25,15))
plt.ylim(0.0,1.1)
plt.plot(components, y_var, marker='o', linestyle='-', color='green')

plt.xlabel('Number of Components',fontsize=16)
plt.ylabel('Cumulative variance (%)',fontsize=16)
plt.title('The number of components needed to explain variance',fontsize=18)
plt.xticks(components)

plt.axhline(y=0.95, color='red', linestyle='--')
plt.axhline(y=1.00, color='orange', linestyle='--')
plt.axvline(x=36.00, color='blue', linestyle='--')
plt.text(1, 0.95, '95% cut-off threshold', color = 'black', fontsize=16)
plt.text(1, 1, '100% variance if all features are included', color = 'black',

plt.tight_layout()
plt.show()
```



In [92]:
```
# therefore for capturing 95% of variance in dataset we should take first 30
# adding the output or target attribute to train_set
train_set_pca.index = train_set.index
train_set_pca["ViolentCrimesPerPop"] = train_set["ViolentCrimesPerPop"]
train_set_pca
train_set = train_set_pca
train_set.shape
```

Out[92]: (1046, 37)

In [93]:
```
train_set
```

Out[93]:

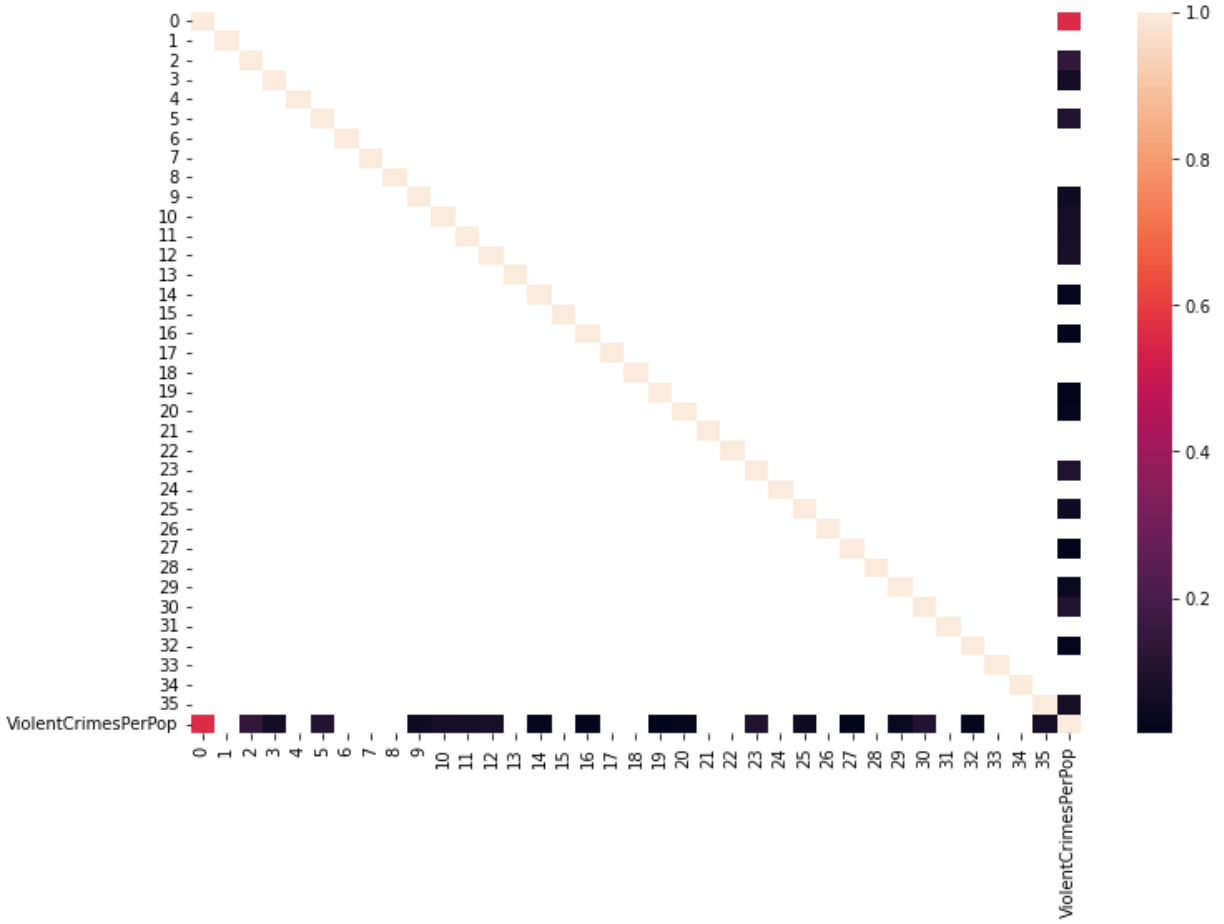
	0	1	2	3	4	5	6	
22	-2.360998	1.846873	4.125858	0.507645	-2.278640	-0.518220	0.150925	0.14495
27	-1.447141	7.959853	0.922674	-0.871256	0.030290	-1.429054	-2.064173	1.79410
28	1.492029	2.875179	2.259540	-3.062838	2.815922	-1.848797	0.599792	-0.50455
34	-7.443684	4.671614	-3.591262	3.252589	0.760099	1.552172	0.823751	-1.89725
38	1.447987	1.602728	-0.167608	0.227237	-1.506626	0.242809	2.333397	1.51630
...	...	...	...	...	...	...	...	
4263	5.172223	-1.497303	4.050828	-0.934905	2.305401	-0.274013	-0.246995	-2.57287
4275	3.909755	-2.546775	4.609521	-2.184349	1.296757	-1.159601	-0.181275	-2.53174
4279	1.732450	-10.730062	2.054287	-4.327002	5.929301	0.688839	-1.070643	0.51765
4282	2.798917	1.118305	-1.722834	-1.539517	-3.226729	2.049230	-1.029076	-0.02501
4283	4.924808	0.748787	2.177290	-1.908641	0.149234	1.712256	0.664954	-1.79109

1046 rows × 37 columns

Correlation Matrix for Train Dataset

- We can see that we have transformed our train dataset into unrelated components by using PCA, as correlation between new features is almost 0

In [94]:
```
corr = train_set.corr()
plt.subplots(figsize=(12,8))
corr_new = corr[corr>=0.00001]
sns.heatmap(corr_new)
plt.show()
```



Modifying Test Data

In [95]:
```
# original test dataset
test_set
```

Out[95]:

	state	fold	population	householdsize	racepctblack	racePctWhite	racePctAsian	racePc
0	21	4	0.01	0.37	0.01	0.99	0.02	
1	34	6	0.07	0.58	0.22	0.59	0.05	
2	34	1	0.01	0.14	0.21	0.67	0.61	
3	29	1	0.01	0.40	0.06	0.87	0.30	
4	6	9	0.04	1.00	0.07	0.45	0.48	
...	...	...	...	...	...	...	...	

	state	fold	population	householdsize	racepctblack	racePctWhite	racePctAsian	racePc
393	23	9	0.00	1.00	0.01	0.95	0.11	
394	55	8	0.00	0.27	0.00	1.00	0.02	
395	8	1	0.34	0.35	0.22	0.73	0.23	
396	48	3	0.00	0.62	0.30	0.42	0.02	
397	25	10	0.08	0.51	0.06	0.87	0.22	

398 rows × 103 columns

Transforming test data into same feature space as train data

In [96]:
```
# removing features, which were eliminated after ANOVA test
print(features_after_f_test)
```

```
Index(['PctKids2Par', 'PctIlleg', 'PctFam2Par', 'PctTeen2Par', 'racePctWhit
e',
       'PctYoungKids2Par', 'racepctblack', 'pctWInvInc', 'FemalePctDiv',
       'TotalPctDiv',
       ...
       'agePct65up', 'PersPerOwnOccHous', 'MedOwnCostPctInc', 'PctEmplManu',
       'PctSameState85', 'PctWorkMomYoungKids', 'householdsize',
       'racePctAsian', 'PersPerOccupHous', 'ViolentCrimesPerPop'],
      dtype='object', length=102)
```

In [97]:
```
# remove features other than those left after ANOVA test
test_set_new = pd.DataFrame()
for feature in features_after_f_test:
    test_set_new[feature] = test_set[feature]
test_set = test_set_new
test_set
```

```
/var/folders/69/1b088435637d6qzmvv7zpycm0000gn/T/ipykernel_29795/1667621105.p
y:4: PerformanceWarning: DataFrame is highly fragmented.  This is usually the
result of calling `frame.insert` many times, which has poor performance.  Con
sider joining all columns at once using pd.concat(axis=1) instead.  To get a
de-fragmented frame, use `newframe = frame.copy()`
  test_set_new[feature] = test_set[feature]
```

Out[97]:

	PctKids2Par	PctIlleg	PctFam2Par	PctTeen2Par	racePctWhite	PctYoungKids2Par	racep
0	0.86	0.04	0.83	0.84	0.99	0.89	
1	0.44	0.50	0.48	0.46	0.59	0.51	
2	0.81	0.22	0.75	0.69	0.67	0.82	
3	0.73	0.07	0.65	0.67	0.87	0.78	
4	0.44	0.43	0.58	0.60	0.45	0.68	
...	...	...	...	...	...	...	
393	0.60	0.34	0.57	0.63	0.95	0.71	
394	0.72	0.09	0.64	0.65	1.00	0.70	

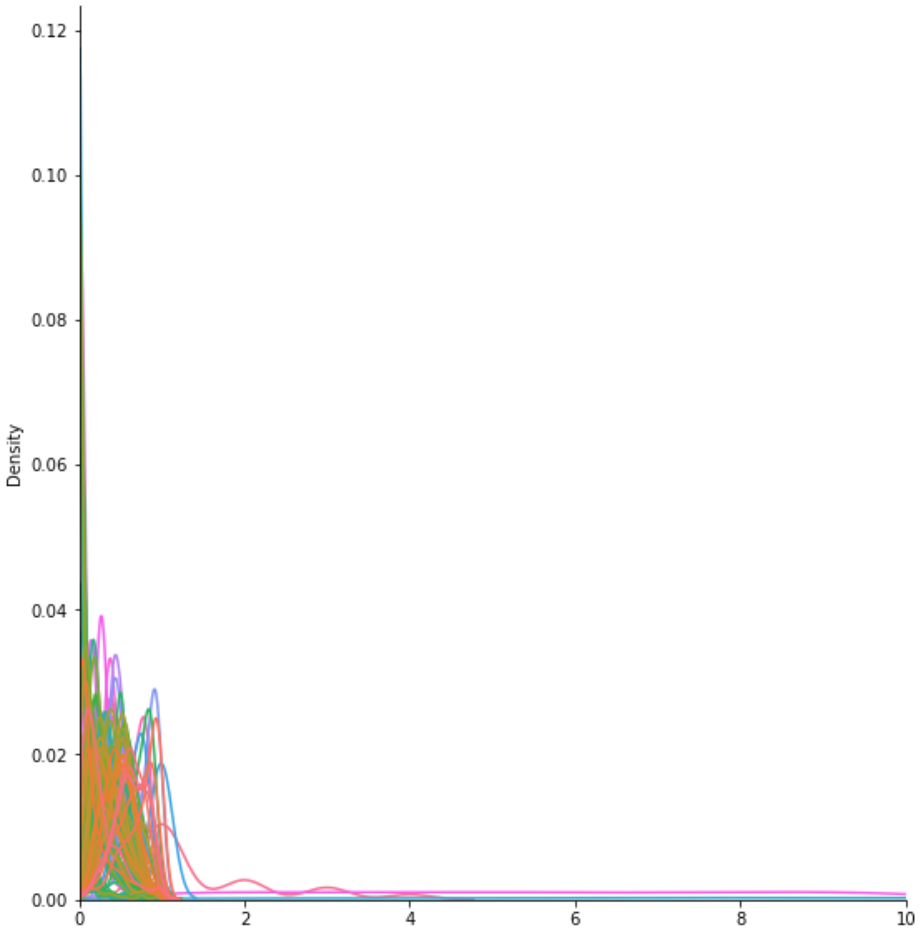
	PctKids2Par	PctIlleg	PctFam2Par	PctTeen2Par	racePctWhite	PctYoungKids2Par	racep
395	0.61	0.25	0.56	0.55	0.73	0.64	
396	0.56	0.28	0.60	0.59	0.42	0.65	
397	0.65	0.17	0.64	0.71	0.87	0.75	

398 rows × 102 columns

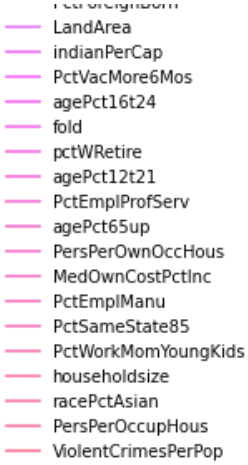
Feature Scaling for Test Data

Plot showing distribution of features before standardization

In [98]:
```
x_axis_limits = (0,10)
sns.displot(test_set.select_dtypes(include='number'), kind='kde',aspect=1,hei
plt.show()
```



PctKids2Par
PctIlleg
PctFam2Par
PctTeen2Par
racePctWhite
PctYoungKids2Par
racepctblack
pctWInvInc
FemalePctDiv
TotalPctDiv
pctWPubAsst
PctPopUnderPov
PctPersOwnOccup
MalePctDivorce
PctUnemployed
PctVacantBoarded
PctNotHSGrad
medFamInc
PctHousOwnOcc
medIncome
PctHousNoPhone
PctHousLess3BR
MalePctNevMarr
perCapInc
PctLess9thGrade
NumIlleg
PctEmploy
PctOccupMgmtProf
PctPersDenseHous
PctLargHouseFam
MedRentPctHousInc
PctWOFullPlumb
PctBSorMore
NumUnderPov
pctWWage
blackPerCap
PctUsePubTrans
HousVacant
PctHousOccup
MedNumBR
NumInShelters
LemasPctOfficDrugUn
PctImmigRec10
NumStreet
PctOccupManu
PctLargHouseOccup
PopDens
pctWFarmSelf
numbUrban
RentLowQ
population
PctImmigRec8
MedRent
PersPerRentOccHous
RentMedian
RentHighQ
HispPerCap
PctImmigRec5
whitePerCap
OwnOccLowQuart
racePctHisp
PctNotSpeakEnglWell
MedYrHousBuilt
state
OwnOccMedVal
NumImmig
PctRecImmig10
pctUrban
PctSameCity85
PctImmigRecent
PctRecImmig8
PctRecImmig5
OwnOccHiQuart
PctSpeakEnglOnly
PersPerFam
PctWorkMom
PctRecentImmig
PctSameHouse85
OtherPerCap
agePct12t29
AsianPerCap
pctWSocSec
MedOwnCostPctIncNoMtg
PctForeignBorn



Standardizing the test dataset

- We are standardizing test data set with mean and standard deviation values obtained from trained dataset

In [99]:
```
# standardizing the test dataset
test_set_new = test_set.select_dtypes(include='number')
test_set_new = test_set_new.drop(columns = ['ViolentCrimesPerPop'])
test_set_new, mean_calculated, standard_deviation_calculated = standardize_da
test_set_new
```

```
/var/folders/69/1b088435637d6qzmvv7zpycm0000gn/T/ipykernel_29795/2610170990.p
y:8: PerformanceWarning: DataFrame is highly fragmented.  This is usually the
result of calling `frame.insert` many times, which has poor performance.  Con
sider joining all columns at once using pd.concat(axis=1) instead.  To get a
de-fragmented frame, use `newframe = frame.copy()`
  standardized_df[dataset.columns[ind]] = standardized_result
```

Out[99]:

	PctKids2Par	PctIlleg	PctFam2Par	PctTeen2Par	racePctWhite	PctYoungKids2Par	rac
0	2.417657	-1.651974	2.345151	2.370573	1.562672	2.150943	
1	-0.027295	0.426013	0.213433	0.161692	-0.224628	0.179670	
2	2.126591	-0.838848	1.857901	1.498646	0.132832	1.787814	
3	1.660886	-1.516453	1.248839	1.382389	1.026482	1.580311	
4	-0.027295	0.109798	0.822495	0.975490	-0.850182	1.061555	
...	...	...	...	...	...	...	
393	0.904115	-0.296765	0.761589	1.149875	1.383942	1.217182	
394	1.602673	-1.426106	1.187933	1.266132	1.607354	1.165306	
395	0.962328	-0.703328	0.700683	0.684848	0.400927	0.854053	
396	0.671263	-0.567807	0.944308	0.917361	-0.984230	0.905928	
397	1.195181	-1.064717	1.187933	1.614903	1.026482	1.424684	

398 rows × 101 columns

In [100…

```
# checking mean and variance of each feature after standardizing the test dat
# here mean of all features might not be 0 because we have standardized using
for feature in test_set_new:
    print("Mean of",feature,"is",round(find_mean(test_set_new,feature)))
    print("Standard Deviation of",feature,"is",round(find_standard_deviation(
```

```
Mean of PctKids2Par is 1
Standard Deviation of PctKids2Par is 1
Mean of PctIlleg is -1
Standard Deviation of PctIlleg is 1
Mean of PctFam2Par is 1
Standard Deviation of PctFam2Par is 1
Mean of PctTeen2Par is 1
Standard Deviation of PctTeen2Par is 1
Mean of racePctWhite is 1
Standard Deviation of racePctWhite is 1
Mean of PctYoungKids2Par is 1
Standard Deviation of PctYoungKids2Par is 1
Mean of racepctblack is -1
Standard Deviation of racepctblack is 1
Mean of pctWInvInc is 1
Standard Deviation of pctWInvInc is 2
Mean of FemalePctDiv is -1
Standard Deviation of FemalePctDiv is 2
Mean of TotalPctDiv is -1
Standard Deviation of TotalPctDiv is 2
Mean of pctWPubAsst is -1
Standard Deviation of pctWPubAsst is 1
Mean of PctPopUnderPov is -1
Standard Deviation of PctPopUnderPov is 1
Mean of PctPersOwnOccup is 1
Standard Deviation of PctPersOwnOccup is 2
Mean of MalePctDivorce is -1
Standard Deviation of MalePctDivorce is 2
Mean of PctUnemployed is -1
Standard Deviation of PctUnemployed is 1
Mean of PctVacantBoarded is 0
Standard Deviation of PctVacantBoarded is 1
Mean of PctNotHSGrad is -1
Standard Deviation of PctNotHSGrad is 2
Mean of medFamInc is 2
Standard Deviation of medFamInc is 2
Mean of PctHousOwnOcc is 1
Standard Deviation of PctHousOwnOcc is 2
Mean of medIncome is 1
Standard Deviation of medIncome is 2
Mean of PctHousNoPhone is -1
Standard Deviation of PctHousNoPhone is 1
Mean of PctHousLess3BR is -1
Standard Deviation of PctHousLess3BR is 2
Mean of MalePctNevMarr is 0
Standard Deviation of MalePctNevMarr is 2
Mean of perCapInc is 2
Standard Deviation of perCapInc is 3
Mean of PctLess9thGrade is -1
Standard Deviation of PctLess9thGrade is 2
Mean of NumIlleg is 0
Standard Deviation of NumIlleg is 4
Mean of PctEmploy is 1
Standard Deviation of PctEmploy is 1
Mean of PctOccupMgmtProf is 1
Standard Deviation of PctOccupMgmtProf is 2
Mean of PctPersDenseHous is 0
```

```
Standard Deviation of PctPersDenseHous is 2
Mean of PctLargHouseFam is 0
Standard Deviation of PctLargHouseFam is 2
Mean of MedRentPctHousInc is 0
Standard Deviation of MedRentPctHousInc is 1
Mean of PctWOFullPlumb is 0
Standard Deviation of PctWOFullPlumb is 1
Mean of PctBSorMore is 2
Standard Deviation of PctBSorMore is 3
Mean of NumUnderPov is 0
Standard Deviation of NumUnderPov is 3
Mean of pctWWage is 1
Standard Deviation of pctWWage is 1
Mean of blackPerCap is 1
Standard Deviation of blackPerCap is 3
Mean of PctUsePubTrans is 1
Standard Deviation of PctUsePubTrans is 3
Mean of HousVacant is 0
Standard Deviation of HousVacant is 3
Mean of PctHousOccup is 0
Standard Deviation of PctHousOccup is 1
Mean of MedNumBR is 0
Standard Deviation of MedNumBR is 1
Mean of NumInShelters is 0
Standard Deviation of NumInShelters is 4
Mean of LemasPctOfficDrugUn is 0
Standard Deviation of LemasPctOfficDrugUn is 1
Mean of PctImmigRec10 is 0
Standard Deviation of PctImmigRec10 is 1
Mean of NumStreet is 1
Standard Deviation of NumStreet is 6
Mean of PctOccupManu is -1
Standard Deviation of PctOccupManu is 1
Mean of PctLargHouseOccup is 0
Standard Deviation of PctLargHouseOccup is 2
Mean of PopDens is 1
Standard Deviation of PopDens is 2
Mean of pctWFarmSelf is 0
Standard Deviation of pctWFarmSelf is 2
Mean of numbUrban is 0
Standard Deviation of numbUrban is 3
Mean of RentLowQ is 1
Standard Deviation of RentLowQ is 2
Mean of population is 0
Standard Deviation of population is 3
Mean of PctImmigRec8 is 0
Standard Deviation of PctImmigRec8 is 1
Mean of MedRent is 1
Standard Deviation of MedRent is 2
Mean of PersPerRentOccHous is 0
Standard Deviation of PersPerRentOccHous is 2
Mean of RentMedian is 1
Standard Deviation of RentMedian is 2
Mean of RentHighQ is 1
Standard Deviation of RentHighQ is 2
Mean of HispPerCap is 1
Standard Deviation of HispPerCap is 2
Mean of PctImmigRec5 is 0
Standard Deviation of PctImmigRec5 is 1
Mean of whitePerCap is 1
Standard Deviation of whitePerCap is 3
Mean of OwnOccLowQuart is 1
Standard Deviation of OwnOccLowQuart is 2
Mean of racePctHisp is 0
```

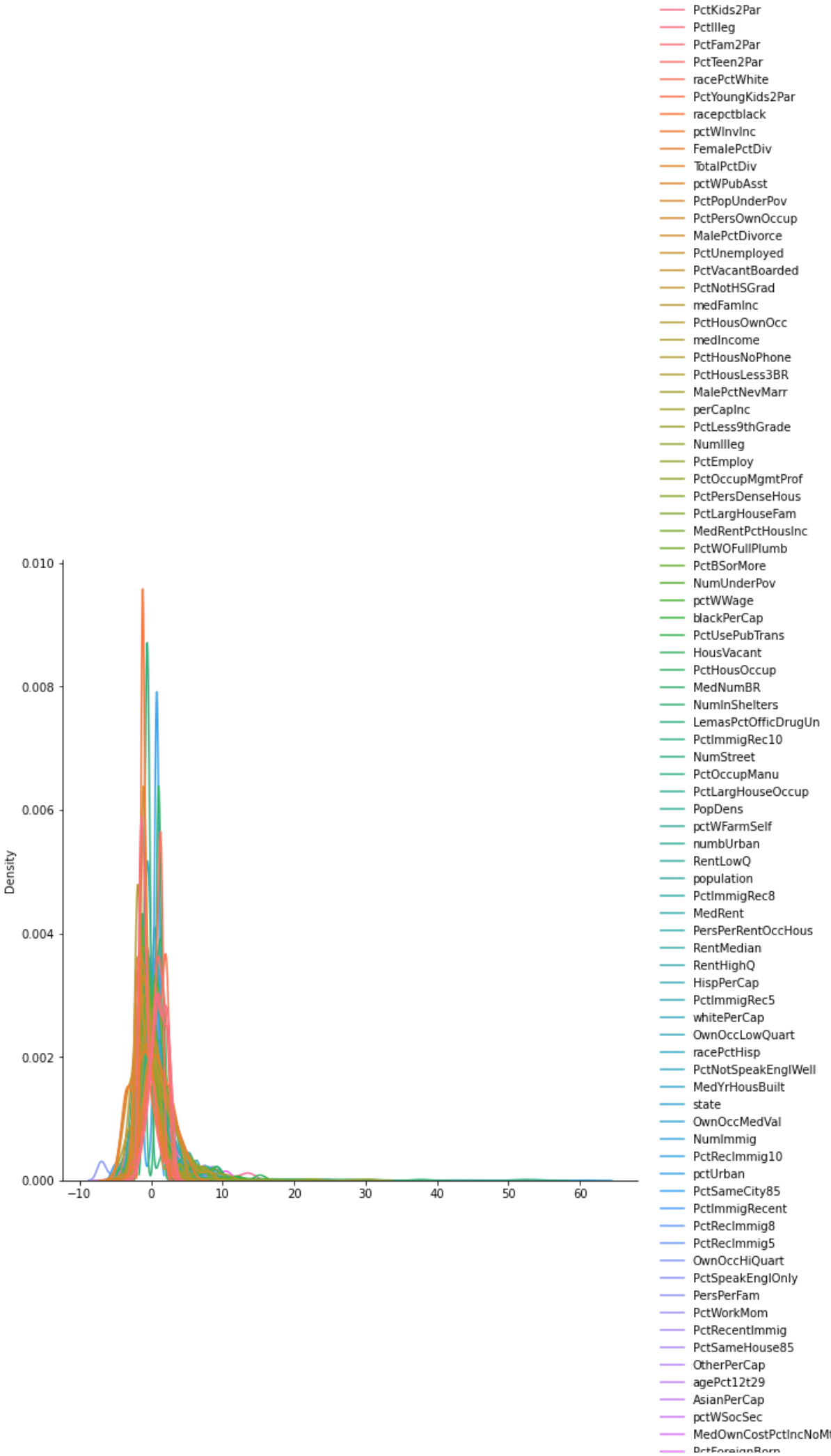
```
Standard Deviation of racePctHisp is 2
Mean of PctNotSpeakEnglWell is 1
Standard Deviation of PctNotSpeakEnglWell is 2
Mean of MedYrHousBuilt is 0
Standard Deviation of MedYrHousBuilt is 1
Mean of state is 0
Standard Deviation of state is 1
Mean of OwnOccMedVal is 1
Standard Deviation of OwnOccMedVal is 2
Mean of NumImmig is 1
Standard Deviation of NumImmig is 5
Mean of PctRecImmig10 is 1
Standard Deviation of PctRecImmig10 is 2
Mean of pctUrban is 0
Standard Deviation of pctUrban is 1
Mean of PctSameCity85 is 0
Standard Deviation of PctSameCity85 is 2
Mean of PctImmigRecent is 0
Standard Deviation of PctImmigRecent is 1
Mean of PctRecImmig8 is 1
Standard Deviation of PctRecImmig8 is 2
Mean of PctRecImmig5 is 1
Standard Deviation of PctRecImmig5 is 2
Mean of OwnOccHiQuart is 2
Standard Deviation of OwnOccHiQuart is 3
Mean of PctSpeakEnglOnly is -1
Standard Deviation of PctSpeakEnglOnly is 2
Mean of PersPerFam is 0
Standard Deviation of PersPerFam is 2
Mean of PctWorkMom is 0
Standard Deviation of PctWorkMom is 1
Mean of PctRecentImmig is 1
Standard Deviation of PctRecentImmig is 2
Mean of PctSameHouse85 is 0
Standard Deviation of PctSameHouse85 is 2
Mean of OtherPerCap is 0
Standard Deviation of OtherPerCap is 2
Mean of agePct12t29 is 0
Standard Deviation of agePct12t29 is 2
Mean of AsianPerCap is 1
Standard Deviation of AsianPerCap is 2
Mean of pctWSocSec is -1
Standard Deviation of pctWSocSec is 1
Mean of MedOwnCostPctIncNoMtg is 0
Standard Deviation of MedOwnCostPctIncNoMtg is 1
Mean of PctForeignBorn is 1
Standard Deviation of PctForeignBorn is 2
Mean of LandArea is 0
Standard Deviation of LandArea is 2
Mean of indianPerCap is 0
Standard Deviation of indianPerCap is 2
Mean of PctVacMore6Mos is 0
Standard Deviation of PctVacMore6Mos is 1
Mean of agePct16t24 is 0
Standard Deviation of agePct16t24 is 3
Mean of fold is 0
Standard Deviation of fold is 1
Mean of pctWRetire is 0
Standard Deviation of pctWRetire is 1
Mean of agePct12t21 is 0
Standard Deviation of agePct12t21 is 2
Mean of PctEmplProfServ is 0
Standard Deviation of PctEmplProfServ is 2
Mean of agePct65up is 0
```

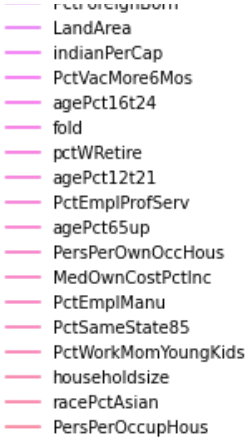
```
Standard Deviation of agePct65up is 1
Mean of PersPerOwnOccHous is 1
Standard Deviation of PersPerOwnOccHous is 2
Mean of MedOwnCostPctInc is 0
Standard Deviation of MedOwnCostPctInc is 1
Mean of PctEmplManu is 0
Standard Deviation of PctEmplManu is 1
Mean of PctSameState85 is 0
Standard Deviation of PctSameState85 is 1
Mean of PctWorkMomYoungKids is 0
Standard Deviation of PctWorkMomYoungKids is 1
Mean of householdsize is 1
Standard Deviation of householdsize is 2
Mean of racePctAsian is 1
Standard Deviation of racePctAsian is 3
Mean of PersPerOccupHous is 0
Standard Deviation of PersPerOccupHous is 2
```

Plot showing distribution of features after standardization

In [101…

```
# all features following a normal distribution with mean 0 and standard devia
sns.displot(test_set_new, kind='kde',aspect=1,height=8)
plt.show()
```


Density
0.010
0.008
0.006
0.004
0.002
0.000
−10
0
10
20
30
40
50
60
PctKids2Par
PctIlleg
PctFam2Par
PctTeen2Par
racePctWhite
PctYoungKids2Par
racepctblack
pctWInvInc
FemalePctDiv
TotalPctDiv
pctWPubAsst
PctPopUnderPov
PctPersOwnOccup
MalePctDivorce
PctUnemployed
PctVacantBoarded
PctNotHSGrad
medFamInc
PctHousOwnOcc
medIncome
PctHousNoPhone
PctHousLess3BR
MalePctNevMarr
perCapInc
PctLess9thGrade
NumIlleg
PctEmploy
PctOccupMgmtProf
PctPersDenseHous
PctLargHouseFam
MedRentPctHousInc
PctWOFullPlumb
PctBSorMore
NumUnderPov
pctWWage
blackPerCap
PctUsePubTrans
HousVacant
PctHousOccup
MedNumBR
NumInShelters
LemasPctOfficDrugUn
PctImmigRec10
NumStreet
PctOccupManu
PctLargHouseOccup
PopDens
pctWFarmSelf
numbUrban
RentLowQ
population
PctImmigRec8
MedRent
PersPerRentOccHous
RentMedian
RentHighQ
HispPerCap
PctImmigRec5
whitePerCap
OwnOccLowQuart
racePctHisp
PctNotSpeakEnglWell
MedYrHousBuilt
state
OwnOccMedVal
NumImmig
PctRecImmig10
pctUrban
PctSameCity85
PctImmigRecent
PctRecImmig8
PctRecImmig5
OwnOccHiQuart
PctSpeakEnglOnly
PersPerFam
PctWorkMom
PctRecentImmig
PctSameHouse85
OtherPerCap
agePct12t29
AsianPerCap
pctWSocSec
MedOwnCostPctIncNoMtg



In [102…

```
# replacing standardized features with original features in test_set datafram
test_set_new.index = test_set.index
test_set_new["ViolentCrimesPerPop"] = test_set["ViolentCrimesPerPop"]
test_set = test_set_new
test_set
```

Out[102…

	PctKids2Par	PctIlleg	PctFam2Par	PctTeen2Par	racePctWhite	PctYoungKids2Par	rac
0	2.417657	-1.651974	2.345151	2.370573	1.562672	2.150943	
1	-0.027295	0.426013	0.213433	0.161692	-0.224628	0.179670	
2	2.126591	-0.838848	1.857901	1.498646	0.132832	1.787814	
3	1.660886	-1.516453	1.248839	1.382389	1.026482	1.580311	
4	-0.027295	0.109798	0.822495	0.975490	-0.850182	1.061555	
…	…	…	…	…	…	…	
393	0.904115	-0.296765	0.761589	1.149875	1.383942	1.217182	
394	1.602673	-1.426106	1.187933	1.266132	1.607354	1.165306	
395	0.962328	-0.703328	0.700683	0.684848	0.400927	0.854053	
396	0.671263	-0.567807	0.944308	0.917361	-0.984230	0.905928	
397	1.195181	-1.064717	1.187933	1.614903	1.026482	1.424684	

398 rows × 102 columns

PCA for test data

In [103…

```
# taking projection of test data on principal component vectors,
# to get same number of features in test data as of train data
test_set_pca = np.dot(test_set.drop(columns="ViolentCrimesPerPop"),k_principa
test_set_pca = pd.DataFrame(test_set_pca)
test_set_pca.index = test_set.index
test_set_pca["ViolentCrimesPerPop"] = test_set["ViolentCrimesPerPop"]
test_set_pca
```

Out[103…

	0	1	2	3	4	5	6	
0	-8.381388	8.886566	1.532797	3.218392	0.246438	0.007568	-3.682067	-0.945

	0	1	2	3	4	5	6	
1	-2.147304	-1.790273	-4.256463	4.707015	0.457863	-0.279839	-1.522824	0.643
2	-21.475811	-6.583620	11.670088	-4.362050	-5.688029	6.248615	-1.821663	0.248
3	-16.186497	0.158672	10.238806	1.292196	-11.684169	6.742646	-7.338530	1.618
4	-10.602371	-22.642634	-16.262279	0.846165	8.219615	4.450340	0.015121	8.120
...	...	...	...	...	...	...	...	
393	-6.482000	-3.185092	0.137684	-3.111196	-13.191133	1.755999	-6.111537	4.253
394	-2.603001	9.247413	1.229680	-0.965769	1.719885	-0.435603	-1.019826	1.447
395	-7.409865	-4.747490	5.619794	4.717426	-7.346563	-9.696860	2.812491	-0.659
396	2.890404	-6.157752	-10.676276	-2.387439	4.066325	-1.740989	0.570418	2.333
397	-12.220624	-5.324790	2.977662	3.601110	-2.512336	4.468524	-2.442074	5.774

398 rows × 37 columns

In [104…
```
# finally we have test_set in same dimension as our train_set
test_set = test_set_pca
test_set.shape
```

Out[104…
```
(398, 37)
```

Implementing Machine Learning Models

- Decision tree model with entropy implementation
- Adaboost
- Multiclass SVM

Classification Report

- True Positive (TP): Predictions which are predicted by classifier as `positive` and are actually `positive`
- False Positive (FP): Predictions which are predicted by classifier as `positive` and are actually `negative`
- True Negative (TN): Predictions which are predicted by classifier as `negative` and are actually `negative`
- False Negative (FN): Predictions which are predicted by classifier as `negative` and are actually `positive`
- `Accuracy`

  It describes the number of correct predictions over all predictions.

$$\frac{TP+TN}{TP+FP+TN+FN} = \frac{\text{Number of correct predictions}}{\text{Number of total predictions}}$$

- Precision

  Precision is a measure of how many of the positive predictions made are correct.

$$\frac{TP}{TP+FP} = \frac{\text{Number of correctly predicted positive instances}}{\text{Number of total positive predictions made}}$$

- Recall

  Recall is a measure of how many of the positive cases the classifier correctly predicted, over all the positive cases in the data.

$$\frac{TP}{TP+FN} = \frac{\text{Number of correctly predicted positive instances}}{\text{Number of total positive instances in dataset}}$$

- F1 Score

  F1-Score is a measure combining both precision and recall. It is harmonic mean of both.

$$\frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

- Support

  Support may be defined as the number of samples of the true response that lies in each class of target values

- Confusion Matrix

  It is a table with 4 different combinations of predicted and actual values.

```
                    |                 | Predicted Positive |
Predicted Negative  |
                    |-----------------|--------------------|----
-----------------|
                    | Actual Positive | True Positives     |
False Negatives     |
                    | Actual Negative | False Positives    |
True Negatives      |
```

- ROC Curve

  The ROC curve is produced by calculating and plotting the true positive rate against the false positive rate for a single classifier at a variety of thresholds.

  The Area Under the Curve (AUC) is the measure of the ability of a binary classifier to distinguish between classes and is used as a summary of the ROC curve.The higher the AUC, the better the model's performance at distinguishing between the positive and negative classes.

$$\text{True Positive Rate (TPR)} = \frac{TP}{TP + FN}$$

$$\text{False Positive Rate (FPR)} = \frac{FP}{FP + TN}$$

In [105…

```
# Machine Learning Algorithms
MLA = {} # dictionary for comparing models
MLA_predictions = {} # dictionary for comparing ROC curve of models
```

In [106…

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1
from sklearn.metrics import classification_report, roc_curve, roc_auc_score,
from sklearn.preprocessing import label_binarize

# function to show performance metrics for classification models
def performance_metrics(name,y_test,y_pred):
    # finding accuracy score
    accuracy = accuracy_score(y_test, y_pred)

    # calculating classification report
    clf_report = pd.DataFrame(classification_report(y_test,y_pred,output_dict
    print("Test Result:\n================================================")
    print(f"Accuracy Score: {accuracy_score(y_test, y_pred) * 100:.2f}%")
    print("_______________________________________________")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")
    print("_______________________________________________")
    print(f"Confusion Matrix: \n {confusion_matrix(y_test, y_pred)}\n")

    # confusion matrix
    sns.heatmap(confusion_matrix(y_test,y_pred),fmt='',cmap = 'YlGnBu')
    plt.xlabel("Predicted Values")
    plt.ylabel("Actual Values")

    MLA[name] = [name,accuracy_score(y_test, y_pred)*100,precision_score(y_te
    MLA_predictions[name] = y_pred

    y_test_bin = label_binarize(y_test, classes=np.unique(y))
    y_pred_bin = label_binarize(y_pred, classes=np.unique(y))

    # Compute ROC curve and ROC area for each class
    fpr = dict()
    tpr = dict()
    roc_auc = dict()

    n_classes = y_test_bin.shape[1]

    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_pred_bin[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Compute micro-average ROC curve and ROC area
    fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), y_pred_bin.
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

    # Plot ROC curves
    plt.figure(figsize=(8, 6))

    for i in range(n_classes):
        plt.plot(fpr[i], tpr[i], lw=2,label='ROC curve of class {0} (area = {
```

```
    plt.plot([0, 1], [0, 1], 'k--', lw=2)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve for Multiple Cla
    plt.legend(loc="lower right")
    plt.show()
```

Splitting train and test data into x and y components

In [107…

```
x_train = train_set.drop(columns='ViolentCrimesPerPop')
y_train = train_set['ViolentCrimesPerPop']
x_test = test_set.drop(columns='ViolentCrimesPerPop')
y_test = test_set['ViolentCrimesPerPop']
```

In [108…

```
# converting column names in train and test dataset to strings
```

In [109…

```
new_column_names = []
for val in train_set.columns:
    new_column_names.append(str(val))
train_set.columns = new_column_names
test_set.columns = new_column_names
```

2. Decision tree model with entropy implementation

- Decision Tree is a Supervised learning technique that can be used for classification problems.
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.
- Decision trees can model nonlinear relationships in the data, capturing complex patterns that linear models may miss.
- Decision trees are relatively robust to irrelevant features, as they are less likely to be selected for splitting.
- Decision trees are prone to overfitting, especially when the tree is deep. Overfitting occurs when the model captures noise in the training data, leading to poor generalization on new, unseen data.

2.1 Implementation of the Model

In [110…
```
# creating validation data set, which will be used in pruning of decision tre
train_set_new, val_set = split_train_test(train_set,0.2)
```

In [111…
```
train_set_new.shape
```

Out[111…
```
(837, 37)
```

In [112…
```
val_set.shape
```

Out[112…
```
(209, 37)
```

In [113…
```
x_train_new = train_set_new.drop(columns='ViolentCrimesPerPop')
y_train_new = train_set_new['ViolentCrimesPerPop']
```

Helper functions for Decision Tree

In [114…
```
import random
from pprint import pprint
```

In [115…
```
# function to check if y_values are pure i.e all belong to a single class or
def check_purity(y_values):
    y_values = np.array(y_values)
    unique_classes = np.unique(y_values)
    if(len(unique_classes) == 1):
        return True
    return False
```

In [116…
```
# function to classify data based on majority count in y_values
def classify_data(y_values):
    y_values = np.array(y_values)
    unique_classes, counts_unique_classes = np.unique(y_values, return_counts
    index = counts_unique_classes.argmax()
    classification = unique_classes[index]
    return classification
```

In [117…
```
# function to find all possible potential splits for all features
def get_potential_splits(x_train,random_subspace=None):
    # random_subspace is number of features that we will randomly select from
    potential_splits = {} # dictionary that will contain splits for each feat
    num_columns = x_train.shape[1]

    # for creating a list from 0 to num_columns
    column_indices = list(range(num_columns))

    if random_subspace and random_subspace <= len(column_indices):
        column_indices = random.sample(column_indices,random_subspace)

    # for each feature we will have multiple splits
    for column_index in column_indices:
        potential_splits[column_index] = []
        values = x_train.iloc[:,column_index]
        unique_values = np.unique(values)
```

```
        potential_splits[column_index] = unique_values

    return potential_splits
```

In [118…

```
# split data function
def split_data(train_dataset,split_column,split_value):
    # this will take all values for that particular column
    split_column_values = train_dataset.iloc[:,split_column]

    # splitting data into binary split
    left_child_data = train_dataset[split_column_values <= split_value]
    right_child_data = train_dataset[split_column_values > split_value]

    return left_child_data, right_child_data
```

Entropy

Entropy is a measurement of uncertainty (sometimes called impurity) has the following formula

$$H(X) = -\sum_j p_j log_2(p_j)$$

$$\text{where } p_j \text{ is the probability of class j} \tag{7}$$

For two class problem, it takes the following form

$$H(X) = -plog_2(p) - (1-p)log_2(1-p)$$

When we are looking for a value to split our decision tree node, we want to minimize entropy.

In [119…

```
# entropy function, to find impurity measure for parent node
def calculate_entropy(train_dataset):
    # to find number of samples belonging to class 1 and class 2
    y_train = np.array(train_dataset.iloc[:,-1])
    _, counts = np.unique(y_train,return_counts=True)
    total_samples = counts.sum()

    # this will contain p_j i.e probability of class j, where j=1,2
    probability_of_class = counts / total_samples
    entropy = -sum((probability_of_class * np.log2(probability_of_class)))

    return entropy
```

Overall Entropy

Overall Entropy for children nodes, can be calculated using the formula

$$H(children) = \sum_j^2 (\frac{n_j}{N} \cdot H_j)$$

$$\text{where } n_j \text{ is number of training samples associated with node j such that}$$

$$N = \sum_j^2 n_j$$

and H_j is entropy of child node j

In [120…

```
# overall entropy function
def calculate_overall_entropy(left_child_data,right_child_data):
    n1 = len(left_child_data) # number of data points belonging to left child
    n2 = len(right_child_data) # number of data points belonging to right chi
    N = n1 + n2 # total number of data points
    overall_entropy = ((n1/N)*calculate_entropy(left_child_data)) + ((n2/N)*c
    return overall_entropy
```

Information Gain

Information gain is a measure which tells how much information we have gained after splitting a node at a particular value.

$$\text{Information Gain} = H(Parent) - H(Children)$$

where $H(Parent)$ is entropy of Parent node or entropy before split and $H(Children)$ is

We need to find the split, which results in highest information gain.

In [121…

```
# function for finding information gain
def calculate_information_gain(train_dataset,left_child_data,right_child_data
    return calculate_entropy(train_dataset) - calculate_overall_entropy(left_
```

In [122…

```
# function for finding best split, by taking highest information gain
def find_best_split(train_dataset):
    information_gain = -1e9

    if(len(train_dataset.columns) == 31):
        x_train = train_dataset.drop(columns="ViolentCrimesPerPop")
    else:
        x_train = train_dataset

    # find all possible splits
    potential_splits = get_potential_splits(x_train)

    # iterating over all splits to find best split, which gives minimum overa
    for column_index in potential_splits:
        for value in potential_splits[column_index]:

            # this will give data points for left and right child
            left_child_data, right_child_data = split_data(x_train,column_ind
            information_gain_current = calculate_information_gain(train_datas

            if information_gain_current > information_gain:
                information_gain = information_gain_current
                best_split_column = column_index
                best_split_value = value

    return best_split_column, best_split_value
```

Decision Tree Algorithm

- We will represent our decision tree in form of a dictionary {}, where {key:[yes,no]} represents a node, in which key will be a question and yes corresponds to left child node and no corresponds to right child node
- Also yes and no can contain dictionaries, which will represent the left and right subtrees

Pre Pruning of Decision Tree

Pre pruning is nothing but stopping the growth of decision tree on an early stage before it perfectly fits the entire training data. For that we can limit the growth of trees by setting constrains. We can limit parameters like max_depth, min_samples etc.

In [123…

```
def decision_tree_algorithm(train_dataset,counter=0,min_samples=2,max_depth=5

    # converting train_dataset to numpy 2d array
    if counter == 0:
        global column_headers
        column_headers = train_dataset.columns
        # first call of this function, so convert train_dataset to numpy 2d a
        data = train_dataset.values
    else:
        data = train_dataset

    x_train = train_dataset.drop(columns="ViolentCrimesPerPop")
    y_train = train_dataset.iloc[:,-1]

    # base case, if node is pure, or contains less than min_samples or depth
    if (check_purity(y_train)) or (len(train_dataset) < min_samples) or (coun
        classification = classify_data(y_train)
        return classification

    else:
        # recursive part of decision tree algorithm
        counter += 1 # increment the depth of decision tree

        # helper functions
        potential_splits = get_potential_splits(x_train,random_subspace)
        split_column, split_value = find_best_split(x_train)
        left_child_data, right_child_data = split_data(train_dataset,split_co

        if len(left_child_data) == 0 or len(right_child_data) == 0:
            classification = classify_data(y_train)
            return classification

        # instantiate sub tree
        feature_name = column_headers[split_column]
        question = "{} <= {}".format(feature_name,split_value)
        sub_tree = {question : []}

        # find answers
        yes_answer = decision_tree_algorithm(left_child_data,counter,min_samp
        no_answer = decision_tree_algorithm(right_child_data,counter,min_samp

        # if both answers are same then, assign sub_tree as yes_answer instea
        if yes_answer == no_answer:
            sub_tree = yes_answer
        else:
            sub_tree[question].append(yes_answer)
            sub_tree[question].append(no_answer)
```

```
    return sub_tree
```

In [124…

```
tree = decision_tree_algorithm(train_set_new,counter=0,min_samples=2,max_dept
tree
```

Out[124…

```
{'0 <= 0.7280063836287614': [{'0 <= -3.261947911874109': [{'0 <= -7.052976662
483581': [{'0 <= -9.495292969870643': [{'0 <= -10.800313303159285': [{'0 <= -
11.793150969937663': [{'0 <= -13.161891236352506': [1.0,
              {'0 <= -12.428216743809624': [{'0 <= -12.560854679077366': [1.
0,
                  {'0 <= -12.521555953092221': [1.0, 2.0]}]},
                {'0 <= -12.201598728739986': [{'0 <= -12.342009027908947':
[2.0,
                    1.0]},
                  {'0 <= -11.858100017615454': [2.0, 1.0]}]}]}]},
            {'0 <= -11.254842726339641': [{'0 <= -11.37628603637561': [{'0 <=
-11.650739822378933': [1.0,
                  {'0 <= -11.553027433257745': [1.0, 2.0]}]},
                {'0 <= -11.331454171776372': [1.0, 2.0]}]},
              {'0 <= -11.027055309898248': [{'0 <= -11.252506745086137': [1.
0,
                  2.0]},
                {'0 <= -10.912270523114513': [2.0,
                  {'0 <= -10.864369423492938': [1.0, 2.0]}]}]}]}]},
          {'0 <= -10.276282738034201': [{'0 <= -10.604581236966876': [{'0 <=
-10.735049409651822': [2.0,
                {'0 <= -10.645622948200955': [2.0,
                  {'0 <= -10.619924034969824': [2.0, 1.0]}]}]},
              {'0 <= -10.486013675044589': [1.0,
                {'0 <= -10.341837431260402': [{'0 <= -10.413951740642782':
[1.0,
                    3.0]},
                  {'0 <= -10.312302524527945': [1.0, 3.0]}]}]}]},
            {'0 <= -9.927842884988932': [{'0 <= -10.116454762549912': [{'0 <=
-10.19762926786366': [1.0,
                  {'0 <= -10.19345093645953': [2.0, 1.0]}]},
                {'0 <= -10.091616706559401': [2.0, 1.0]}]},
              {'0 <= -9.776476120320842': [{'0 <= -9.895776874322188': [1.0,
                  {'0 <= -9.810981472669866': [3.0, 1.0]}]},
                {'0 <= -9.633055829302286': [{'0 <= -9.646451363851646': [2.
0,
                    1.0]},
                  {'0 <= -9.524885688843499': [3.0, 2.0]}]}]}]}]}]},
        {'0 <= -7.9097459396435905': [{'0 <= -8.448289793500045': [{'0 <= -8.
919204245693862': [{'0 <= -9.103600678503078': [{'0 <= -9.469469525326724':
[1.0,
                  {'0 <= -9.394272281115622': [1.0, 2.0]}]},
                {'0 <= -9.050101043695019': [2.0,
                  {'0 <= -8.98320757586313': [1.0, 2.0]}]}]},
              {'0 <= -8.773642179761062': [{'0 <= -8.864540049355135': [2.0,
                  {'0 <= -8.817009524987892': [3.0, 2.0]}]},
                {'0 <= -8.585374369187083': [1.0, 2.0]}]}]},
            {'0 <= -8.133505697231275': [{'0 <= -8.365873688532659': [1.0,
                {'0 <= -8.361416371084573': [1.0,
                  {'0 <= -8.332703963867763': [3.0, 1.0]}]}]},
              {'0 <= -7.981172357951601': [1.0,
                {'0 <= -7.920532541755457': [{'0 <= -7.928386422953897': [1.
0,
                    2.0]},
                  {'0 <= -7.916281751241617': [2.0, 1.0]}]}]}]}]},
          {'0 <= -7.53388994228319': [{'0 <= -7.754919675454553': [2.0,
              {'0 <= -7.609331364061289': [{'0 <= -7.737465861649922': [1.0,
```

```
                    {'0 <= -7.71026584025775': [2.0, 1.0]}]},
                  {'0 <= -7.589337179265498': [{'0 <= -7.594285026902986': [1.
0,
                     2.0]},
                    {'0 <= -7.545205238078293': [1.0, 2.0]}]}]}]},
              {'0 <= -7.183008744808971': [1.0,
                {'0 <= -7.131378003245525': [{'0 <= -7.1747148207458356': [2.0,
                    1.0]},
                  {'0 <= -7.067287749810122': [1.0,
                    {'0 <= -7.063584801088995': [1.0, 2.0]}]}]}]}]}]}]},
      {'0 <= -4.9306413441681824': [{'0 <= -6.044588889216328': [{'0 <= -6.60
2963967443026': [{'0 <= -6.719253407218171': [{'0 <= -6.753787531842467':
[{'0 <= -7.0065925985194575': [1.0,
                    {'0 <= -6.879623402777509': [2.0, 1.0]}]},
                  2.0]},
                {'0 <= -6.664875506564018': [1.0,
                  {'0 <= -6.63022205021766': [1.0,
                    {'0 <= -6.604602028010028': [2.0, 3.0]}]}]}]},
              {'0 <= -6.301085287062221': [{'0 <= -6.3882419207592065': [{'0 <=
-6.554003791303949': [2.0,
                    {'0 <= -6.469454367604097': [2.0, 1.0]}]},
                  {'0 <= -6.328821204060464': [1.0,
                    {'0 <= -6.304780521506788': [3.0, 1.0]}]}]},
                {'0 <= -6.131347709837168': [{'0 <= -6.20557664305727': [2.0,
                    {'0 <= -6.131893279891773': [2.0, 1.0]}]},
                  {'0 <= -6.092968402111993': [{'0 <= -6.1278075165133465': [3.
0,
                     2.0]},
                    1.0]}]}]}]},
            {'0 <= -5.400839178523503': [{'0 <= -5.820960258996232': [{'0 <= -
5.8900327326663495': [{'0 <= -6.019145050085513': [2.0,
                    1.0]},
                  {'0 <= -5.871399473855346': [1.0,
                    {'0 <= -5.868614136237443': [3.0, 2.0]}]}]},
                {'0 <= -5.677615923706555': [{'0 <= -5.798978326824968': [1.0,
                    {'0 <= -5.696674707098078 5]': [1.0, 2.0]}]},
                  {'0 <= -5.5964316222616': [{'0 <= -5.6085825496204755': [2.0,
                     1.0]},
                    {'0 <= -5.584617368091665': [3.0, 2.0]}]}]}]},
              {'0 <= -5.254707533318251': [{'0 <= -5.293650419571811': [1.0,
                  {'0 <= -5.2922763100057075': [2.0,
                    {'0 <= -5.286615321684558': [1.0, 2.0]}]}]},
                {'0 <= -5.161247732039175': [2.0,
                  {'0 <= -5.091761781029507': [{'0 <= -5.121037689984868': [2.
0,
                     1.0]},
                    1.0]}]}]}]}]},
          {'0 <= -4.293304018053063': [{'0 <= -4.659533331409093': [{'0 <= -4.8
283418385905135': [{'0 <= -4.90314217952335': [{'0 <= -4.924318059926598':
[1.0,
                    {'0 <= -4.917167846535146': [1.0, 3.0]}]},
                  {'0 <= -4.839614096680089': [2.0, 1.0]}]},
                {'0 <= -4.711974558541758': [{'0 <= -4.791372393248962': [2.0,
                    {'0 <= -4.747499957554111': [1.0, 2.0]}]},
                  {'0 <= -4.697650196642075': [{'0 <= -4.700603686869406': [1.
0,
                     2.0]},
                    {'0 <= -4.6676405890279264': [4.0, 2.0]}]}]}]},
              {'0 <= -4.410601556272679': [{'0 <= -4.515595597031647': [{'0 <=
-4.607480856539886': [2.0,
                     3.0]},
                  {'0 <= -4.463133141355229': [1.0,
                    {'0 <= -4.4445712545032': [2.0, 3.0]}]}]},
                {'0 <= -4.348195137496499': [{'0 <= -4.391103604241543': [3.0,
```

```
                    {'0 <= -4.371373010213127': [2.0, 3.0]}]},
                  {'0 <= -4.325544927258225': [{'0 <= -4.341336751649347': [2.
0,
                     1.0]},
                    2.0]}]}]}]},
           {'0 <= -3.77104680704797': [{'0 <= -3.9914491583044733': [{'0 <= -
4.212850511213136': [{'0 <= -4.261373443958728': [1.0,
                     {'0 <= -4.244756116985941': [3.0, 2.0]}]},
                  {'0 <= -4.207464002666511': [1.0,
                     {'0 <= -4.035005107698246': [1.0, 3.0]}]}]},
               {'0 <= -3.8752323225329404': [{'0 <= -3.979517456687396': [3.0,
                     {'0 <= -3.875670643706253': [3.0, 1.0]}]},
                  {'0 <= -3.7994198875201928': [1.0,
                     {'0 <= -3.7912405871046837': [1.0, 2.0]}]}]}]},
             {'0 <= -3.52314222739126': [{'0 <= -3.5884011251768286': [{'0 <=
-3.6030892625298128': [1.0,
                     {'0 <= -3.6009039763399913': [1.0, 2.0]}]},
                  {'0 <= -3.585866079053335': [3.0, 1.0]}]},
               {'0 <= -3.3146702025138257': [{'0 <= -3.516768818318209': [1.0,
                     {'0 <= -3.4898713111184767': [1.0, 2.0]}]},
                  {'0 <= -3.26958842548215': [{'0 <= -3.273470166166101': [3.0,
                     2.0]},
                     {'0 <= -3.266616351092663': [1.0, 2.0]}]}]}]}]}]}]}]},
    {'0 <= -0.9176182976712918': [{'0 <= -2.004983943070221': [{'0 <= -2.5656
0007790081': [{'0 <= -2.898180615338752': [{'0 <= -3.075997341699039': [{'0 <
= -3.155393388411214': [{'0 <= -3.1854931371095785': [2.0,
                     {'0 <= -3.164257488220636': [3.0, 1.0]}]},
                  {'0 <= -3.140580653255009': [3.0, 1.0]}]},
               {'0 <= -2.972836298361113': [{'0 <= -3.0732419594505678': [1.0,
                     {'0 <= -3.006123270432864': [3.0, 1.0]}]},
                  {'0 <= -2.903766726893113': [{'0 <= -2.924736394954724': [2.
0,
                     1.0]},
                     {'0 <= -2.9033116946646595': [3.0, 1.0]}]}]}]},
             {'0 <= -2.6449228695722677': [{'0 <= -2.700769282849995': [2.0,
                  {'0 <= -2.6944924259183236': [3.0,
                     {'0 <= -2.6489393116221898': [2.0, 3.0]}]}]},
               {'0 <= -2.583261659083073': [{'0 <= -2.6449155853697217': [3.0,
                     {'0 <= -2.633987242203806': [3.0, 2.0]}]},
                  {'0 <= -2.581841279323064': [3.0, 2.0]}]}]}]},
            {'0 <= -2.2075426718777584': [{'0 <= -2.420954640966093': [{'0 <= -
2.4732030245820296': [{'0 <= -2.5351694830592': [1.0,
                     {'0 <= -2.530954922059226': [2.0, 1.0]}]},
                  {'0 <= -2.456234780182493': [1.0,
                     {'0 <= -2.4366225713987517': [2.0, 1.0]}]}]},
               {'0 <= -2.2911211047086972': [{'0 <= -2.342478557711846': [2.0,
                     3.0]},
                  {'0 <= -2.2344887368631072': [{'0 <= -2.2594230634176036':
[1.0,
                     3.0]},
                     {'0 <= -2.2095471816753536': [3.0, 1.0]}]}]}]},
             {'0 <= -2.1268180549117495': [{'0 <= -2.157471018694598': [{'0 <=
-2.187970332393074': [1.0,
                     {'0 <= -2.1660422839698183': [4.0, 3.0]}]},
                  {'0 <= -2.1529719352813457': [3.0,
                     {'0 <= -2.1482999697670127': [3.0, 2.0]}]}]},
               {'0 <= -2.1099074507248647': [{'0 <= -2.124836185212672': [1.0,
                     {'0 <= -2.118171657366007': [1.0, 3.0]}]},
                  {'0 <= -2.028969235228675': [2.0,
                     {'0 <= -2.013931764913449': [3.0, 2.0]}]}]}]}]}]},
        {'0 <= -1.3696059927761268': [{'0 <= -1.5046429969184312': [{'0 <= -
1.6786922901932955': [{'0 <= -1.8705168889146082': [{'0 <= -1.888223020097721
2': [2.0,
                     3.0]},
```

```
                {'0 <= -1.8679160234337353': [3.0,
                  {'0 <= -1.784963135026026': [3.0, 2.0]}]}]},
               {'0 <= -1.5629180790261363': [{'0 <= -1.6631476072156337': [3.
0,
                  {'0 <= -1.5765898887300933': [3.0, 2.0]}]},
                {'0 <= -1.5412154958552842': [3.0,
                  {'0 <= -1.5225793385816115': [3.0, 2.0]}]}]}]},
             {'0 <= -1.4158784305179881': [{'0 <= -1.4471406463612937': [{'0 <
= -1.4699753053260511': [3.0,
                  2.0]},
                3.0]},
               {'0 <= -1.3784157767111866': [{'0 <= -1.4144104290714843': [1.
0,
                  {'0 <= -1.4016546532179155': [1.0, 2.0]}]},
                {'0 <= -1.3756333612333598': [{'0 <= -1.3781340055727893':
[1.0,
                  2.0]},
                  {'0 <= -1.372026607460815': [1.0, 3.0]}]}]}]}]},
           {'0 <= -1.1349573576874443': [{'0 <= -1.2610276495918047': [{'0 <=
-1.292095892724028': [{'0 <= -1.3285114489981702': [2.0,
                  3.0]},
                {'0 <= -1.2745806438643457': [3.0,
                  {'0 <= -1.2647770795805384': [2.0, 1.0]}]}]},
               {'0 <= -1.1801972036817983': [{'0 <= -1.2411282099861016': [3.
0,
                  {'0 <= -1.188414861736113': [3.0, 2.0]}]},
                {'0 <= -1.1674441752956355': [{'0 <= -1.1761748198560287':
[3.0,
                  4.0]},
                  3.0]}]}]},
             {'0 <= -1.1036797188477303': [{'0 <= -1.1098271148583456': [{'0 <
= -1.1101971606736 99': [2.0,
                  {'0 <= -1.1098704772498977': [2.0, 3.0]}]},
                {'0 <= -1.1083873380710019': [2.0,
                  {'0 <= -1.1074771444644278': [1.0, 2.0]}]}]},
               {'0 <= -1.0092805711096913': [{'0 <= -1.0720584796785444': [3.
0,
                  2.0]},
                {'0 <= -0.941227502237781': [3.0,
                  {'0 <= -0.9370393492431657': [4.0, 3.0]}]}]}]}]}]}]},
      {'0 <= -0.10782598922476021': [{'0 <= -0.502356118586919': [{'0 <= -0.7
339140023153793': [{'0 <= -0.8384882167769127': [{'0 <= -0.8551494846016792':
[{'0 <= -0.893446407064499': [2.0,
                  3.0]},
                {'0 <= -0.8538268836847547': [3.0,
                  {'0 <= -0.8416075262568696': [2.0, 1.0]}]}]},
               {'0 <= -0.7864618624779887': [{'0 <= -0.8123953434935318': [3.
0,
                  {'0 <= -0.8113740724018066': [3.0, 2.0]}]},
                2.0]}]},
             {'0 <= -0.6008070261676359': [{'0 <= -0.6472824217832219': [{'0 <
= -0.6851550745943348': [3.0,
                  1.0]},
                {'0 <= -0.643846295511347': [2.0,
                  {'0 <= -0.6213102238465882': [3.0, 2.0]}]}]},
               {'0 <= -0.5541589507122966': [2.0,
                  {'0 <= -0.5420455796895325': [{'0 <= -0.5507013410792535':
[3.0,
                  2.0]},
                  3.0]}]}]}]},
           {'0 <= -0.33054441083869446': [{'0 <= -0.43589113324326323': [{'0 <
= -0.4595320863231015': [3.0,
                  {'0 <= -0.44442512795437106': [1.0, 3.0]}]},
                {'0 <= -0.4091158640442715': [{'0 <= -0.4293235047269294': [3.
```

```
0,
                  {'0 <= -0.4156298992778529': [4.0, 3.0]}]},
                {'0 <= -0.36029810111814475': [2.0,
                  {'0 <= -0.3456568780629919': [1.0, 3.0]}]}]}]},
              {'0 <= -0.2352751369879682': [{'0 <= -0.2986778674106076': [{'0 <
= -0.3296151425484334': [3.0,
                  {'0 <= -0.3176573653504632': [3.0, 2.0]}]},
                {'0 <= -0.28026514936733393': [1.0, 2.0]}]},
              {'0 <= -0.19700313751435305': [{'0 <= -0.23078876774096868':
[3.0,
                  {'0 <= -0.2001961610332711': [3.0, 2.0]}]},
                {'0 <= -0.14222335759056465': [2.0,
                  {'0 <= -0.10955800390604006': [1.0, 2.0]}]}]}]}]}]},
            {'0 <= 0.3508370021005178': [{'0 <= 0.1795722053540358': [{'0 <= 0.00
5299447064019917': [{'0 <= -0.023287656155142514': [{'0 <= -0.046944233194426
77': [1.0,
                  {'0 <= -0.03789913853667304': [3.0, 1.0]}]},
                {'0 <= -0.007634635110780774': [1.0,
                  {'0 <= -0.007544547902225154': [2.0, 3.0]}]}]},
              {'0 <= 0.05653346069025748': [{'0 <= 0.013212835634813472': [1.
0,
                  2.0]},
                3.0]}]},
              {'0 <= 0.28459915170117167': [{'0 <= 0.23688092959319113': [{'0 <
= 0.2027211652707221': [1.0,
                  2.0]},
                {'0 <= 0.24464310282273474': [3.0,
                  {'0 <= 0.2842369648131089': [4.0, 3.0]}]}]},
              {'0 <= 0.3237733199289029': [{'0 <= 0.29497672616728504': [3.0,
                  {'0 <= 0.3184052230797785': [3.0, 1.0]}]},
                {'0 <= 0.3331642570803236': [2.0,
                  {'0 <= 0.3382161767917537': [4.0, 1.0]}]}]}]}]},
            {'0 <= 0.5357280318724685': [{'0 <= 0.4417045500362445': [{'0 <= 0.
3739402864661403': [{'0 <= 0.35630682071788716': [3.0,
                  {'0 <= 0.3581197169110429': [3.0, 2.0]}]},
                {'0 <= 0.38111773171985813': [2.0,
                  {'0 <= 0.42905868764373606': [3.0, 2.0]}]}]},
              {'0 <= 0.46220999205237406': [{'0 <= 0.4435610276897869': [1.0,
                  {'0 <= 0.4539167464824569': [1.0, 2.0]}]},
                2.0]}]},
              {'0 <= 0.6803057565219721': [{'0 <= 0.5988135022419036': [{'0 <=
0.5609509980191311': [3.0,
                  {'0 <= 0.5922693167176061': [4.0, 2.0]}]},
                {'0 <= 0.6529755087530219': [{'0 <= 0.6519725602650897': [2.
0,
                    3.0]},
                  3.0]}]},
              {'0 <= 0.687978365344083': [{'0 <= 0.681727409357389': [2.0,
                  {'0 <= 0.6878895437799228': [2.0, 1.0]}]},
                {'0 <= 0.7010634595703897': [2.0, 3.0]}]}]}]}]}]}]},
  {'0 <= 3.433728477469406': [{'0 <= 2.191996513722518': [{'0 <= 1.5655304813
09002': [{'0 <= 1.2341775808035202': [{'0 <= 0.952943019577741': [{'0 <= 0.80
82727399630734': [{'0 <= 0.7837750565556105': [{'0 <= 0.7455368723108038':
[2.0,
                  {'0 <= 0.7476965274444918': [4.0, 3.0]}]},
                {'0 <= 0.7844516708284052': [2.0,
                  {'0 <= 0.8060196369233643': [2.0, 3.0]}]}]},
              {'0 <= 0.9073527772953891': [{'0 <= 0.842279043218569': [2.0,
                  {'0 <= 0.9027897255881718': [4.0, 1.0]}]},
                {'0 <= 0.9157945249570263': [{'0 <= 0.9089605427459422': [3.
0,
                    2.0]},
                  3.0]}]}]},
              {'0 <= 1.0509461346662252': [{'0 <= 0.9810808376818619': [{'0 <=
```

```
0.9754452889371454': [3.0,
                  {'0 <= 0.9807188846005152': [2.0, 3.0]}]},
                {'0 <= 1.0116667986496515': [2.0,
                  {'0 <= 1.017554903380439': [2.0, 3.0]}]}]},
              {'0 <= 1.1606222302232128': [{'0 <= 1.0856234841552408': [4.0,
                  {'0 <= 1.0934497904812814': [4.0, 2.0]}]},
                {'0 <= 1.182836331347373': [{'0 <= 1.161836008769545': [3.0,
                    1.0]},
                  {'0 <= 1.1943845392811743': [1.0, 2.0]}]}]}]},
           {'0 <= 1.4407220726948646': [{'0 <= 1.3339592885092084': [{'0 <= 1.
3048573628446292': [{'0 <= 1.2841496948169335': [3.0,
                  {'0 <= 1.299243131026492': [3.0, 1.0]}]},
                {'0 <= 1.3072450223462622': [3.0,
                  {'0 <= 1.3308576456915528': [2.0, 3.0]}]}]},
              {'0 <= 1.3912370817534285': [2.0,
                {'0 <= 1.4280829540255937': [{'0 <= 1.4279745275674756': [3.
0,
                    2.0]},
                  {'0 <= 1.4327149539603132': [3.0, 1.0]}]}]}]},
            {'0 <= 1.5026939925032532': [{'0 <= 1.4920285161992761': [{'0 <=
1.4583592864240251': [2.0,
                  {'0 <= 1.4627872432857716': [4.0, 1.0]}]},
                {'0 <= 1.4938401335918978': [3.0, 2.0]}]},
              {'0 <= 1.5230422749734944': [3.0,
                {'0 <= 1.5359762837578084': [2.0,
                  {'0 <= 1.560346664986711': [3.0, 2.0]}]}]}]}]}]},
        {'0 <= 1.816103608920021': [{'0 <= 1.6868773857375179': [{'0 <= 1.643
4857651076322': [{'0 <= 1.6278182288584504': [{'0 <= 1.6066447772982901': [3.
0,
                  {'0 <= 1.618362779449698': [2.0, 4.0]}]},
                {'0 <= 1.6297043346207005': [3.0,
                  {'0 <= 1.64171353301983': [2.0, 3.0]}]}]},
              {'0 <= 1.6661582673249649': [{'0 <= 1.6467773290508132': [4.0,
                  3.0]},
                {'0 <= 1.6814807808042973': [3.0,
                  {'0 <= 1.683899403082679': [4.0, 3.0]}]}]}]},
            {'0 <= 1.7428187946836877': [{'0 <= 1.6996698736298295': [2.0,
                {'0 <= 1.731236865744616': [3.0,
                  {'0 <= 1.7324498290033437': [4.0, 3.0]}]}]},
              {'0 <= 1.7723700103145203': [{'0 <= 1.751844752952528': [4.0,
                  {'0 <= 1.7681651832689027': [3.0, 2.0]}]},
                {'0 <= 1.7860150715413288': [{'0 <= 1.7768109582023592': [2.
0,
                    3.0]},
                  2.0]}]}]}]},
           {'0 <= 1.9920326797665648': [{'0 <= 1.922748540242347': [{'0 <= 1.8
99621811779232': [{'0 <= 1.8246466195096351': [2.0,
                    3.0]},
                  3.0]},
                {'0 <= 1.943645873872717': [{'0 <= 1.9249635326852383': [1.0,
                    {'0 <= 1.937314288056698': [4.0, 1.0]}]},
                  {'0 <= 1.9777825257466977': [{'0 <= 1.960019646373171': [2.0,
                      3.0]},
                    {'0 <= 1.9902929924908122': [2.0, 4.0]}]}]}]},
              {'0 <= 2.081607292333174': [{'0 <= 2.013669558788897': [{'0 <= 1.
9987845848114105': [2.0,
                    {'0 <= 2.0079331496625357': [3.0, 1.0]}]},
                  {'0 <= 2.0225171210964286': [1.0,
                    {'0 <= 2.0534970730234168': [2.0, 3.0]}]}]},
              {'0 <= 2.1256449277198963': [{'0 <= 2.121097166154558': [1.0,
                    {'0 <= 2.124037078193798': [2.0, 3.0]}]},
                  {'0 <= 2.1598226296002143': [{'0 <= 2.1571354259915037': [1.
0,
                      2.0]},
```

```
                    {'0 <= 2.1891841473723543': [2.0, 4.0]}]}]}]}]}]}]}]}]}]},
      {'0 <= 2.818268822990638': [{'0 <= 2.5613372937738017': [{'0 <= 2.37105
0186957797': [{'0 <= 2.290310589422022': [{'0 <= 2.2183524330200686': [{'0 <=
2.2000609238932967': [1.0,
                    2.0]},
                  {'0 <= 2.2351012621648123': [2.0,
                    {'0 <= 2.2812323688042766': [3.0, 2.0]}]}]},
                {'0 <= 2.323413791760041': [{'0 <= 2.298976249553604': [1.0,
                    2.0]},
                  {'0 <= 2.3641798464823625': [{'0 <= 2.3635173260472118': [2.
0,
                      1.0]},
                    {'0 <= 2.369973704403873': [2.0, 3.0]}]}]}]},
              {'0 <= 2.4841744323538935': [{'0 <= 2.47377135096124': [{'0 <= 2.
3720614825580992': [2.0,
                    {'0 <= 2.376045003565785': [3.0, 2.0]}]},
                  3.0]},
                {'0 <= 2.532354425850875': [{'0 <= 2.4857729369339046': [3.0,
                    {'0 <= 2.526943329156035': [2.0, 3.0]}]},
                  {'0 <= 2.543046897303006': [{'0 <= 2.5428884327230166': [2.0,
                      1.0]},
                    {'0 <= 2.5554355991089404': [2.0, 3.0]}]}]}]}]},
            {'0 <= 2.7403194310653634': [{'0 <= 2.6334863807423745': [{'0 <= 2.
5797768995257297': [{'0 <= 2.568453109396364': [3.0,
                    {'0 <= 2.5767832743757038': [4.0, 3.0]}]},
                  {'0 <= 2.6184057786072175': [2.0,
                    {'0 <= 2.625418532777493': [2.0, 3.0]}]}]},
                {'0 <= 2.6774993996252046': [{'0 <= 2.635304887530785': [2.0,
                    3.0]},
                  {'0 <= 2.6879337733978437': [{'0 <= 2.687484210378507': [1.0,
                      3.0]},
                    {'0 <= 2.73219764809725': [2.0, 1.0]}]}]}]},
              {'0 <= 2.7769256206721837': [{'0 <= 2.757395748286095': [{'0 <=
2.743988122587314': [3.0,
                    {'0 <= 2.7546314567252455': [3.0, 4.0]}]},
                  {'0 <= 2.758424272993125': [1.0, 3.0]}]},
                {'0 <= 2.7858335873707887': [3.0,
                  {'0 <= 2.8095237855620616': [{'0 <= 2.798916548952106': [4.0,
                      2.0]},
                    {'0 <= 2.816146870765243': [2.0, 3.0]}]}]}]}]}]},
          {'0 <= 3.089077085484604': [{'0 <= 3.0013013524422147': [{'0 <= 2.913
9250056268335': [{'0 <= 2.8459823700475044': [{'0 <= 2.8290610837294574': [4.
0,
                    {'0 <= 2.838280014279995': [3.0, 2.0]}]},
                  {'0 <= 2.8467287076083023': [2.0,
                    {'0 <= 2.8674992692574754': [4.0, 2.0]}]}]},
                {'0 <= 2.9326763321910496': [{'0 <= 2.9186357172453294': [4.0,
                    {'0 <= 2.9294384066107884': [4.0, 2.0]}]},
                  {'0 <= 2.9419668205042493': [{'0 <= 2.9374404399075473': [4.
0,
                      2.0]},
                    {'0 <= 2.9661054708370536': [2.0, 4.0]}]}]}]},
              {'0 <= 3.054585904253115': [{'0 <= 3.0337441099372424': [{'0 <=
3.015336387043709': [2.0,
                    {'0 <= 3.0298772191368792': [3.0, 2.0]}]},
                  {'0 <= 3.0368819137888137': [3.0,
                    {'0 <= 3.041954976324466': [2.0, 4.0]}]}]},
                {'0 <= 3.0741728531932786': [{'0 <= 3.05465013042857': [3.0,
                    {'0 <= 3.0680461203147544': [4.0, 2.0]}]},
                  {'0 <= 3.07859640222823': [{'0 <= 3.0757493410512713': [3.0,
                      2.0]},
                    {'0 <= 3.084283812636863': [3.0, 2.0]}]}]}]}]},
            {'0 <= 3.313788959385474': [{'0 <= 3.1897919631294025': [{'0 <= 3.1
54615810193867': [{'0 <= 3.1043166118347396': [3.0,
```

```
                    {'0 <= 3.109739535502683': [2.0, 4.0]}]},
                  {'0 <= 3.179409692646944': [4.0,
                    {'0 <= 3.1804128321457825': [2.0, 3.0]}]}]},
                {'0 <= 3.2362551601663565': [4.0,
                  {'0 <= 3.268890449831338': [2.0, 4.0]}]}]},
              {'0 <= 3.3648049775653766': [{'0 <= 3.3233009404260767': [{'0 <=
3.319137311732222': [4.0,
                    {'0 <= 3.3201972517530085': [4.0, 2.0]}]},
                  {'0 <= 3.332798093443041': [{'0 <= 3.331461752391881': [4.0,
                      3.0]},
                    {'0 <= 3.334934314281045': [4.0, 3.0]}]}]},
                {'0 <= 3.4098014776188745': [{'0 <= 3.367536603150854': [1.0,
                    3.0]},
                  {'0 <= 3.4134411254202544': [{'0 <= 3.411346643153476': [2.0,
                      3.0]},
                    {'0 <= 3.4327368981609254': [3.0, 4.0]}]}]}]}]}]}]}]},
    {'0 <= 5.0589940742738095': [{'0 <= 4.120312680822826': [{'0 <= 3.8337719
699365627': [{'0 <= 3.6134278344431587': [{'0 <= 3.4703393896649355': [{'0 <=
3.4586308363829894': [{'0 <= 3.449460858404483': [2.0,
                    {'0 <= 3.45805901547986': [4.0, 2.0]}]},
                  {'0 <= 3.4635735847384446': [3.0,
                    {'0 <= 3.467265038625177': [2.0, 3.0]}]}]},
                {'0 <= 3.529688363923447': [3.0,
                  {'0 <= 3.5779228223066433': [2.0,
                    {'0 <= 3.5869442941086986': [3.0, 2.0]}]}]}]},
              {'0 <= 3.7678468343669103': [{'0 <= 3.630652605563729': [{'0 <=
3.6269352337490544': [1.0,
                    {'0 <= 3.6293123263912603': [4.0, 3.0]}]},
                  {'0 <= 3.721345683781023': [2.0,
                    {'0 <= 3.740137659094317': [3.0, 4.0]}]}]},
                {'0 <= 3.7834888228016212': [{'0 <= 3.7705554893227267': [3.0,
                    4.0]},
                  {'0 <= 3.807705106456328': [{'0 <= 3.7939759125646986': [2.0,
                      3.0]},
                    4.0]}]}]}]},
            {'0 <= 3.973166074936135': [{'0 <= 3.9386340477538644': [{'0 <= 3.8
984963131082413': [{'0 <= 3.866599717666341': [4.0,
                    {'0 <= 3.888049900940029': [3.0, 4.0]}]},
                  {'0 <= 3.898691747519289': [3.0,
                    {'0 <= 3.9097546380297157': [4.0, 2.0]}]}]},
                {'0 <= 3.9501524173093925': [{'0 <= 3.9479522924938566': [2.0,
                    {'0 <= 3.949220557106969': [4.0, 1.0]}]},
                  {'0 <= 3.95368765883589': [{'0 <= 3.950210966859584': [2.0,
                      3.0]},
                    {'0 <= 3.9657093062763242': [1.0, 4.0]}]}]}]},
              {'0 <= 4.025690596509022': [{'0 <= 4.004456317270502': [{'0 <= 3.
9743453317990145': [3.0,
                    {'0 <= 3.9758095281754886': [2.0, 3.0]}]},
                  {'0 <= 4.009852469006732': [4.0,
                    {'0 <= 4.017057509534999': [2.0, 1.0]}]}]},
                {'0 <= 4.051630816017176': [{'0 <= 4.028064227176627': [4.0,
                    2.0]},
                  {'0 <= 4.075106479928334': [{'0 <= 4.062399655627791': [4.0,
                      2.0]},
                    4.0]}]}]}]},
          {'0 <= 4.616581387694879': [{'0 <= 4.327783222874218': [{'0 <= 4.2358
55204286302': [{'0 <= 4.162661636631958': [{'0 <= 4.13595939424371': [2.0,
                    4.0]},
                  {'0 <= 4.168000888580112': [4.0,
                    {'0 <= 4.185975361798905': [3.0, 4.0]}]}]},
                {'0 <= 4.242993125217393': [{'0 <= 4.240954511526099': [4.0,
                    {'0 <= 4.24282892099713': [4.0, 1.0]}]},
                  4.0]}]},
              {'0 <= 4.47311721505236': [{'0 <= 4.390399548302685': [{'0 <= 4.3
```

```
80242819863171': [4.0,
                  {'0 <= 4.381735615153468': [2.0, 4.0]}]},
                {'0 <= 4.42710491046772': [4.0,
                  {'0 <= 4.4495337952685725': [2.0, 4.0]}]}]},
              {'0 <= 4.539109095659612': [{'0 <= 4.481496721929724': [4.0,
                  3.0]},
                {'0 <= 4.596228603000508': [3.0, 4.0]}]}]}]},
          {'0 <= 4.78323843160215': [{'0 <= 4.682738946735586': [{'0 <= 4.649
235406196463': [{'0 <= 4.618677836370331': [2.0,
                  {'0 <= 4.640730617050116': [3.0, 2.0]}]},
                {'0 <= 4.66740861887883': [3.0,
                  {'0 <= 4.671906207766275': [2.0, 3.0]}]}]},
              {'0 <= 4.737499052971041': [{'0 <= 4.694243474898202': [3.0,
                  {'0 <= 4.718535287825099': [4.0, 2.0]}]},
                {'0 <= 4.780831071706645': [{'0 <= 4.7439646266533435': [4.0,
                    2.0]},
                  2.0]}]}]},
            {'0 <= 4.940539748072307': [{'0 <= 4.849648090149425': [{'0 <= 4.
811315097918657': [4.0,
                  {'0 <= 4.840766294853304': [2.0, 4.0]}]},
                {'0 <= 4.873548465507914': [{'0 <= 4.867851037920914': [4.0,
                    2.0]},
                  4.0]}]},
              {'0 <= 4.9914452250837495': [{'0 <= 4.967432621402401': [4.0,
                  {'0 <= 4.980773818279222': [3.0, 2.0]}]},
                {'0 <= 5.048475147780056': [{'0 <= 5.015143639084466': [4.0,
                    3.0]},
                  {'0 <= 5.052720919924545': [3.0, 4.0]}]}]}]}]}]}]},
      {'0 <= 6.36062556601386': [{'0 <= 5.5464466934968': [{'0 <= 5.176786621
418472': [{'0 <= 5.110379829777831': [{'0 <= 5.084959344074541': [4.0,
                {'0 <= 5.095114674748325': [4.0,
                  {'0 <= 5.109882363905693': [4.0, 2.0]}]}]},
              {'0 <= 5.1425361082832834': [{'0 <= 5.124269422341857': [4.0,
                  {'0 <= 5.140129893236866': [2.0, 4.0]}]},
                {'0 <= 5.172223163112585': [4.0, 2.0]}]}]},
            {'0 <= 5.419176765122098': [{'0 <= 5.3568388885703815': [{'0 <=
5.2302595270007455': [4.0,
                  3.0]},
                {'0 <= 5.36693157841115': [2.0,
                  {'0 <= 5.379312251543437': [2.0, 4.0]}]}]},
              {'0 <= 5.484752098018906': [{'0 <= 5.443452291254723': [4.0,
                  {'0 <= 5.453325328620955': [3.0, 4.0]}]},
                {'0 <= 5.504997778709259': [4.0,
                  {'0 <= 5.542988306545491': [2.0, 4.0]}]}]}]}]},
          {'0 <= 5.831413520812852': [{'0 <= 5.700106293293937': [{'0 <= 5.65
9873580660256': [{'0 <= 5.573706192355984': [2.0,
                  {'0 <= 5.605539614128851': [4.0, 3.0]}]},
                {'0 <= 5.660789413000242': [4.0,
                  {'0 <= 5.691205121835595': [3.0, 4.0]}]}]},
              {'0 <= 5.733740383935471': [{'0 <= 5.708373306729579': [3.0,
                  4.0]},
                {'0 <= 5.786750066671211': [4.0,
                  {'0 <= 5.816410934990887': [4.0, 1.0]}]}]}]},
            {'0 <= 6.079502603047383': [{'0 <= 5.884665984023705': [{'0 <= 5.
84173320504501': [4.0,
                  {'0 <= 5.854097080964094': [3.0, 4.0]}]},
                2.0]},
              {'0 <= 6.157842990065853': [{'0 <= 6.145937865839883': [3.0,
                  {'0 <= 6.152559692204737': [4.0, 3.0]}]},
                {'0 <= 6.291226325812763': [4.0,
                  {'0 <= 6.355381559635606': [3.0, 4.0]}]}]}]}]}]},
        {'0 <= 7.169626886754911': [{'0 <= 6.542522878072192': [{'0 <= 6.4995
92665916538': [{'0 <= 6.453128401439568': [{'0 <= 6.360952644209667': [4.0,
                  {'0 <= 6.406145417334407': [4.0, 3.0]}]},
```

```
                {'0 <= 6.455495384308176': [4.0,
                  {'0 <= 6.487085345243445': [3.0, 4.0]}]}]},
              {'0 <= 6.525480977995621': [4.0,
                {'0 <= 6.537743808390138': [4.0,
                  {'0 <= 6.540014721056761': [3.0, 4.0]}]}]}]},
            {'0 <= 6.8233743123138515': [{'0 <= 6.696232421343953': [{'0 <=
6.599152586971217': [4.0,
                  2.0]},
                {'0 <= 6.750904716840599': [3.0, 4.0]}]},
              {'0 <= 6.886755799730519': [{'0 <= 6.830017827131931': [4.0,
                  {'0 <= 6.8769287678750235': [4.0, 2.0]}]},
                {'0 <= 7.004426750451161': [4.0, 2.0]}]}]}]},
          {'0 <= 8.120054775348411': [{'0 <= 7.611891625727754': [{'0 <= 7.42
9279960624658': [2.0,
                {'0 <= 7.472052718355377': [4.0, 3.0]}]},
              {'0 <= 7.755180231075456': [{'0 <= 7.628893668264452': [4.0,
                  {'0 <= 7.704243368615966': [4.0, 2.0]}]},
                {'0 <= 7.8841958226386355': [{'0 <= 7.840895690848917': [2.0,
                    1.0]},
                  {'0 <= 7.994246682839079': [3.0, 4.0]}]}]}]},
            {'0 <= 8.733035139716744': [{'0 <= 8.462692349273986': [{'0 <= 8.
29003468356028': [2.0,
                  4.0]},
                4.0]},
              4.0]}]}]}]}]}]}]}]}
```

In [125…

```
# classify example
def classify_example(example,tree):
    question = list(tree.keys())[0]
    feature_name, comparision_operator, value = question.split()

    # ask a question
    if example[int(feature_name)] <= float(value):
        answer = tree[question][0]
    else:
        answer = tree[question][1]

    # base case, answer is a leaf node
    if not isinstance(answer,dict):
        return answer
    else:
        # recursive part
        residual_tree = answer
        return classify_example(example,residual_tree)
```

In [126…

```
def make_predictions(x_test,tree):
    # find predicted values for each test data point
    predicted_output = []
    for ind in range(len(x_test)):
        predicted_output.append(classify_example(x_test.iloc[ind],tree))

    return predicted_output
```

In [127…

```
def calculate_accuracy(y_predicted,y_test):
    # finding accuracy
    correct_predictions = 0
    for ind in range(len(y_test)):
        if y_test.iloc[ind] == y_predicted[ind]:
            correct_predictions += 1
```

```
    accuracy = correct_predictions / len(y_test)
    return accuracy
```

In [128…

```
y_predicted = make_predictions(x_test,tree)
calculate_accuracy(y_predicted,y_test)*100
```

Out[128…

```
49.246231155778894
```

Post Pruning in Decision Tree

- This is done after construction of decision tree, pruning is done in a bottom-up fashion.
- It is done by replacing a subtree with a new leaf node with a class label which is majority class of records.
- We are using validation dataset for post pruning of decision tree

In [129…

```
def filter_df(df,question):
    feature, _ , value = question.split()
    df_yes =  df[df[feature] <= float(value)]
    df_no =  df[df[feature] > float(value)]

    return df_yes, df_no
```

In [130…

```
def pruning_result(tree,x_train,y_train,x_val,y_val):

    # setting leaf with value as majority label in train set
    leaf = y_train.value_counts().index[0]
    errors_leaf = sum(y_val != leaf)
    errors_decision_node = sum(y_val != make_predictions(x_val,tree))

    # returning leaf node or subtree whichever has less error
    if errors_leaf <= errors_decision_node:
        return leaf
    else:
        return tree
```

In [131…

```
def post_pruning(tree,train_set,val_set):

    # find question for root node of tree
    question = list(tree.keys())[0]

    # this will give left and right subtrees
    yes_answer, no_answer = tree[question]

    x_train = train_set.drop(columns='ViolentCrimesPerPop')
    y_train = train_set['ViolentCrimesPerPop']
    x_val = val_set.drop(columns='ViolentCrimesPerPop')
    y_val = val_set['ViolentCrimesPerPop']

    # base case
    # both left and right subtrees are leaf nodes, so either it will return t
    if not isinstance(yes_answer,dict) and not isinstance(no_answer,dict):
        return pruning_result(tree,x_train,y_train,x_val,y_val)

    # recursive part
```

```
    else:
        df_train_yes, df_train_no = filter_df(train_set,question)
        df_val_yes, df_val_no = filter_df(val_set,question)

        # left subtree, this will do post pruning on left subtree recursively
        if isinstance(yes_answer,dict):
            yes_answer =  post_pruning(yes_answer,df_train_yes,df_val_yes)

        # right subtree, this will do post pruning on right subtree recursive
        if isinstance(no_answer,dict):
            no_answer = post_pruning(no_answer,df_train_no,df_val_no)

        # new tree, so that it doesn't replace the existing tree
        tree = {question:[yes_answer,no_answer]}

    # we are calling pruning_result here, because the tree after post_pruning
    return pruning_result(tree,x_train,y_train,x_val,y_val)
```

In [132…

```
pruned_tree = post_pruning(tree,train_set_new,val_set)
pruned_tree
```

Out[132…

```
{'0 <= 0.7280063836287614': [{'0 <= -3.261947911874109': [{'0 <= -7.052976662
483581': [1.0,
      {'0 <= -4.9306413441681824': [{'0 <= -6.044588889216328': [{'0 <= -6.60
2963967443026': [1.0,
            {'0 <= -6.301085287062221': [1.0,
              {'0 <= -6.131347709837168': [2.0, 1.0]}]}]},
          {'0 <= -5.400839178523503': [1.0,
            {'0 <= -5.254707533318251': [1.0,
              {'0 <= -5.161247732039175': [2.0,
                {'0 <= -5.091761781029507': [2.0, 1.0]}]}]}]}]},
        {'0 <= -4.293304018053063': [2.0, 1.0]}]}]},
    {'0 <= -0.9176182976712918': [{'0 <= -2.004983943070221': [{'0 <= -2.5656
0007790081': [{'0 <= -2.898180615338752': [1.0,
            2.0]},
          {'0 <= -2.2075426718777584': [1.0, 3.0]}]},
        {'0 <= -1.3696059927761268': [{'0 <= -1.5046429969184312': [3.0,
            {'0 <= -1.4158784305179881': [{'0 <= -1.4471406463612937': [2.0,
                3.0]},
              1.0]}]},
          3.0]}]},
      {'0 <= -0.10782598922476021': [{'0 <= -0.502356118586919': [{'0 <= -0.7
339140023153793': [2.0,
            {'0 <= -0.6008070261676359': [3.0, 2.0]}]},
          {'0 <= -0.33054441083869446': [{'0 <= -0.43589113324326323': [{'0 <
= -0.4595320863231015': [3.0,
                {'0 <= -0.44442512795437106': [1.0, 3.0]}]},
              3.0]},
            {'0 <= -0.2352751369879682': [2.0,
              {'0 <= -0.19700313751435305': [3.0,
                {'0 <= -0.14222335759056465': [2.0, 1.0]}]}]}]}]},
        {'0 <= 0.3508370021005178': [{'0 <= 0.1795722053540358': [3.0,
            {'0 <= 0.28459915170117167': [3.0, 2.0]}]},
          {'0 <= 0.5357280318724685': [2.0,
            {'0 <= 0.6803057565219721': [3.0, 2.0]}]}]}]}]}]},
  {'0 <= 3.433728477469406': [{'0 <= 2.191996513722518': [{'0 <= 1.5655304813
09002': [2.0,
        {'0 <= 1.816103608920021': [3.0,
          {'0 <= 1.9920326797665648': [{'0 <= 1.922748540242347': [3.0,
              {'0 <= 1.943645873872717': [1.0, 2.0]}]},
            {'0 <= 2.081607292333174': [{'0 <= 2.013669558788897': [3.0,
                {'0 <= 2.0225171210964286': [1.0, 3.0]}]},
              2.0]}]}]}]},
```

```
      {'0 <= 2.818268822990638': [{'0 <= 2.5613372937738017': [{'0 <= 2.37105
0186957797': [{'0 <= 2.290310589422022': [2.0,
              {'0 <= 2.323413791760041': [{'0 <= 2.298976249553604': [1.0,
                  2.0]},
                2.0]}]},
            3.0]},
          3.0]},
        {'0 <= 3.089077085484604': [{'0 <= 3.0013013524422147': [4.0,
            {'0 <= 3.054585904253115': [{'0 <= 3.0337441099372424': [{'0 <=
3.015336387043709': [2.0,
                  3.0]},
                3.0]},
              {'0 <= 3.0741728531932786': [3.0,
                {'0 <= 3.07859640222823': [2.0,
                  {'0 <= 3.084283812636863': [3.0, 2.0]}]}]}]}]},
          {'0 <= 3.313788959385474': [{'0 <= 3.1897919631294025': [{'0 <= 3.1
54615810193867': [{'0 <= 3.1043166118347396': [3.0,
                  4.0]},
                2.0]},
              {'0 <= 3.2362551601663565': [4.0, 2.0]}]},
            3.0]}]}]}]},
    {'0 <= 5.0589940742738095': [{'0 <= 4.120312680822826': [{'0 <= 3.8337719
699365627': [{'0 <= 3.6134278344431587': [{'0 <= 3.4703393896649355': [2.0,
              {'0 <= 3.529688363923447': [3.0, 2.0]}]},
            {'0 <= 3.7678468343669103': [3.0,
              {'0 <= 3.7834888228016212': [4.0,
                {'0 <= 3.807705106456328': [3.0, 4.0]}]}]}]},
          {'0 <= 3.973166074936135': [{'0 <= 3.9386340477538644': [{'0 <= 3.8
984963131082413': [{'0 <= 3.866599717666341': [4.0,
                  {'0 <= 3.888049900940029': [3.0, 4.0]}]},
                {'0 <= 3.898691747519289': [3.0,
                  {'0 <= 3.9097546380297157': [4.0, 2.0]}]}]},
              1.0]},
            2.0]}]},
        {'0 <= 4.616581387694879': [{'0 <= 4.327783222874218': [4.0,
            {'0 <= 4.47311721505236': [4.0, 3.0]}]},
          {'0 <= 4.78323843160215': [{'0 <= 4.682738946735586': [2.0,
              {'0 <= 4.737499052971041': [3.0, 2.0]}]},
            4.0]}]}]},
      {'0 <= 6.36062556601386': [{'0 <= 5.5464466934968': [{'0 <= 5.176786621
418472': [{'0 <= 5.110379829777831': [4.0,
              {'0 <= 5.1425361082832834': [4.0, 2.0]}]},
            4.0]},
          {'0 <= 5.831413520812852': [4.0,
            {'0 <= 6.079502603047383': [{'0 <= 5.884665984023705': [4.0, 2.
0]},
              4.0]}]}]},
        {'0 <= 7.169626886754911': [4.0,
          {'0 <= 8.120054775348411': [{'0 <= 7.611891625727754': [2.0,
              {'0 <= 7.755180231075456': [4.0, 3.0]}]},
            {'0 <= 8.733035139716744': [{'0 <= 8.462692349273986': [{'0 <= 8.
29003468356028': [2.0,
                  4.0]},
                4.0]},
              4.0]}]}]}]}]}]}
```


Visualizing Decision Boundaries after Post Pruning

- We can see that unpruned tree has overfitting, it overfits the train dataset, but pruned tree prevents overfitting

In [133…

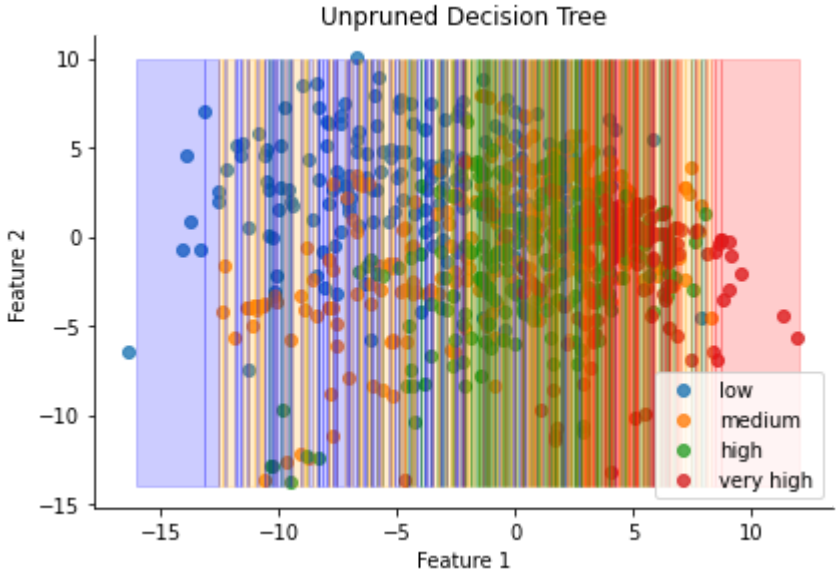
```
def plot_decision_boundaries(tree, x_min, x_max, y_min, y_max):
    forest_green = (34/255, 139/255, 34/255)
    gold = (255/255,215/255,0/255)
    color_keys = {1:"blue",2:"orange",3:"green",4:"red"}

    # recursive part
    if isinstance(tree, dict):
        question = list(tree.keys())[0]
        yes_answer, no_answer = tree[question]
        feature, _, value = question.split()

        if feature == "0":
            plot_decision_boundaries(yes_answer, x_min, float(value), y_min,
            plot_decision_boundaries(no_answer, float(value), x_max, y_min, y
        else:
            plot_decision_boundaries(yes_answer, x_min, x_max, y_min, float(v
            plot_decision_boundaries(no_answer, x_min, x_max, float(value), y

    # "tree" is a leaf
    else:
        plt.fill_between(x=[x_min, x_max], y1=y_min, y2=y_max, alpha=0.2, col

    return
```

In [134…

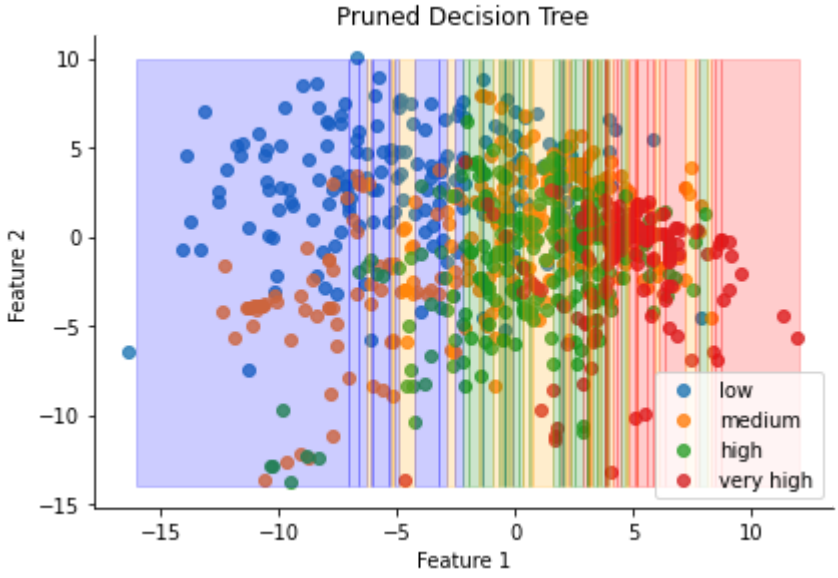
```
def create_plot(df,tree=None,title=None):
    sns.lmplot(data=df,x=df.columns[0], y=df.columns[1], hue="ViolentCrimesPe
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.legend(["low","medium","high","very high"])
    plt.title(title)

    if tree or tree == False: # root of the tree might just be a leave with "
        x_min, x_max = round(df["0"].min()), round(df["0"].max())
        y_min, y_max = round(df["1"].min()), round(df["1"].max())

        plot_decision_boundaries(tree, x_min, x_max, y_min, y_max)

    return
```

In [135…

```
# plotting decision boundaries for train_set using unpruned tree
create_plot(train_set_new,tree,"Unpruned Decision Tree")
```



In [136…
```
# plotting decision boundaries for train_set using pruned decision tree
create_plot(train_set_new,pruned_tree,"Pruned Decision Tree")
```



Accuracy difference before and after pruning decision tree

In [137…
```
# before pruning decision tree
y_predicted = make_predictions(x_test,tree)
calculate_accuracy(y_predicted,y_test)*100
```

Out[137…
```
49.246231155778894
```

In [138…
```
# after pruning decision tree
y_predicted = make_predictions(x_test,pruned_tree)
calculate_accuracy(y_predicted,y_test)*100
```

Out[138…
```
55.527638190954775
```

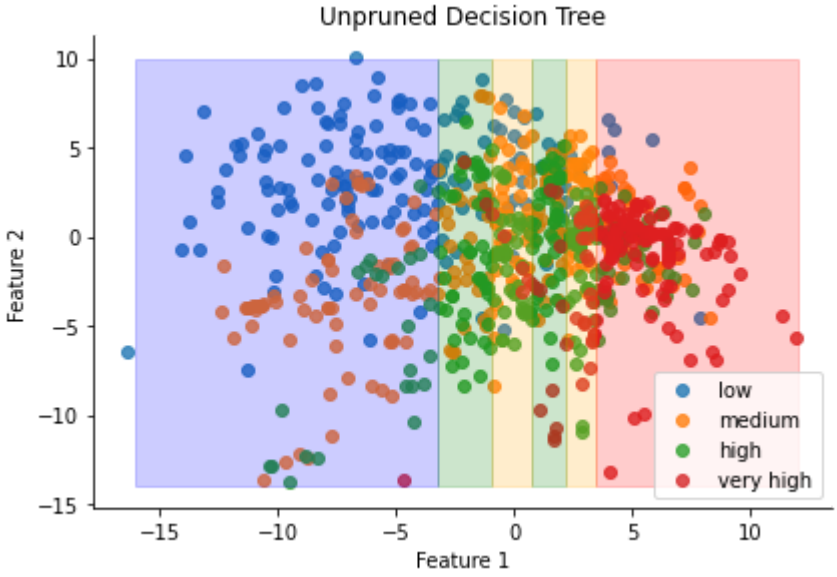
2.2 Insights drawn (plots, markdown explanations)

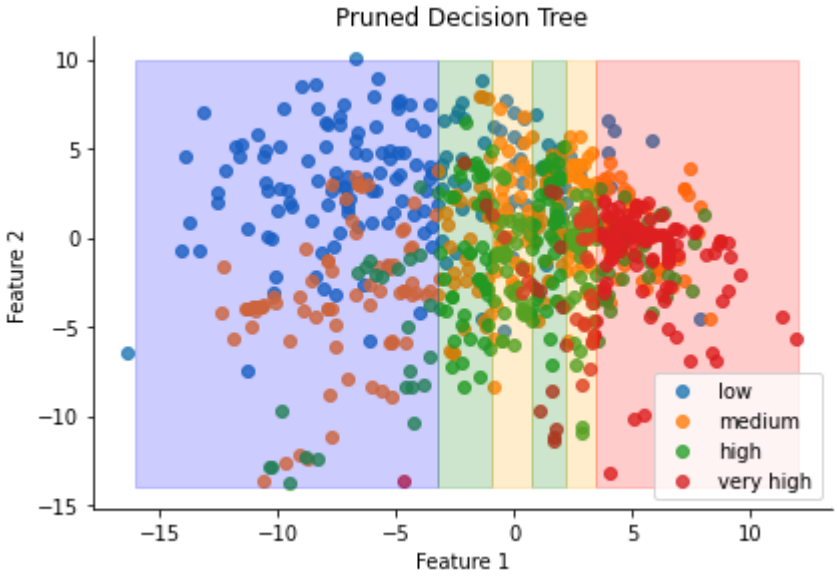
Comparing Accuracy of Decision Tree for different depths

In [139…

```
depth_array = [3,5,8,10,15,20,30]
accuracy_array = []
predictions_array = []

for depth_val in depth_array:
    plt.close('all')

    # generate the decision tree
    tree = decision_tree_algorithm(train_set_new,counter=0,min_samples=2,max_
    # do pruning of decision tree obtained
    pruned_tree = post_pruning(tree,train_set_new,val_set)

    # plot decision boundaries
    print("\n==============================================\n")
    print("Decision boundary on the training set before and after pruning\n\n

    create_plot(train_set_new,tree,"Unpruned Decision Tree")
    plt.pause(1)
    create_plot(train_set_new,pruned_tree,"Pruned Decision Tree")
    plt.pause(1)

    # find accuracy on test dataset after pruning
    y_predicted = make_predictions(x_test,pruned_tree)
    predictions_array.append(y_predicted)
    acc_val = calculate_accuracy(y_predicted,y_test)*100

    print(f"Accuracy for decision tree with max depth as {depth_val} is {acc_
    accuracy_array.append(acc_val)
```

```
==============================================

Decision boundary on the training set before and after pruning
```

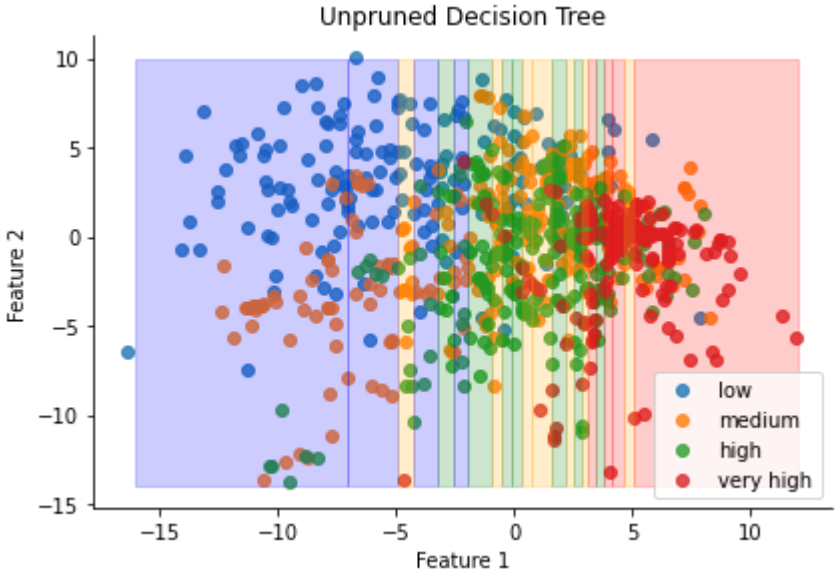


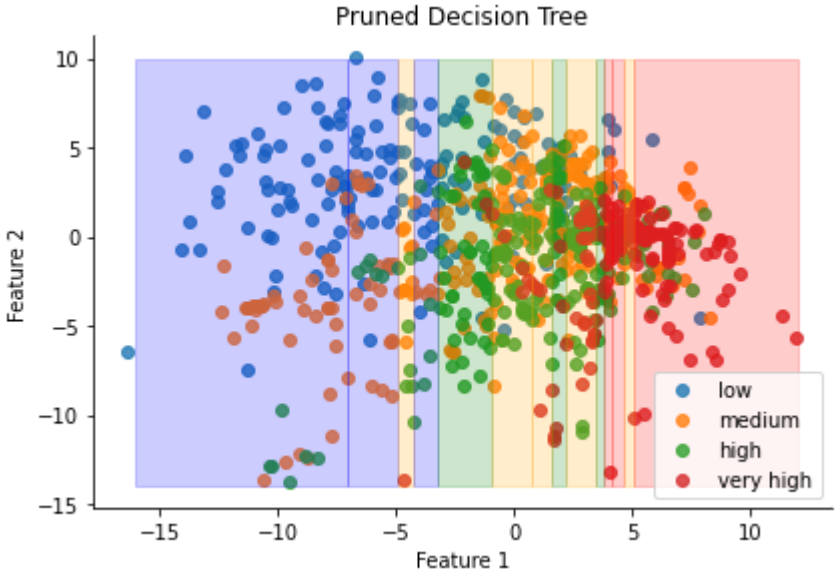


Accuracy for decision tree with max depth as 3 is 56.03015075376885

==========

Decision boundary on the training set before and after pruning

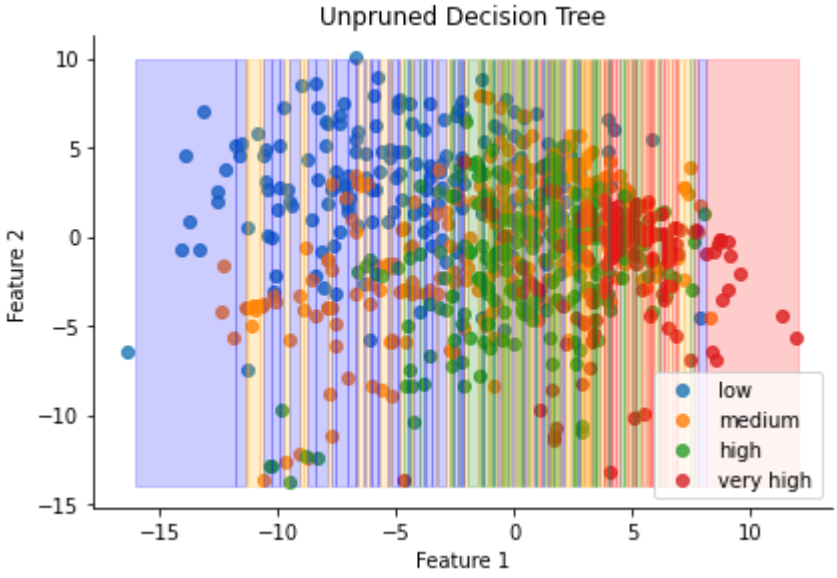


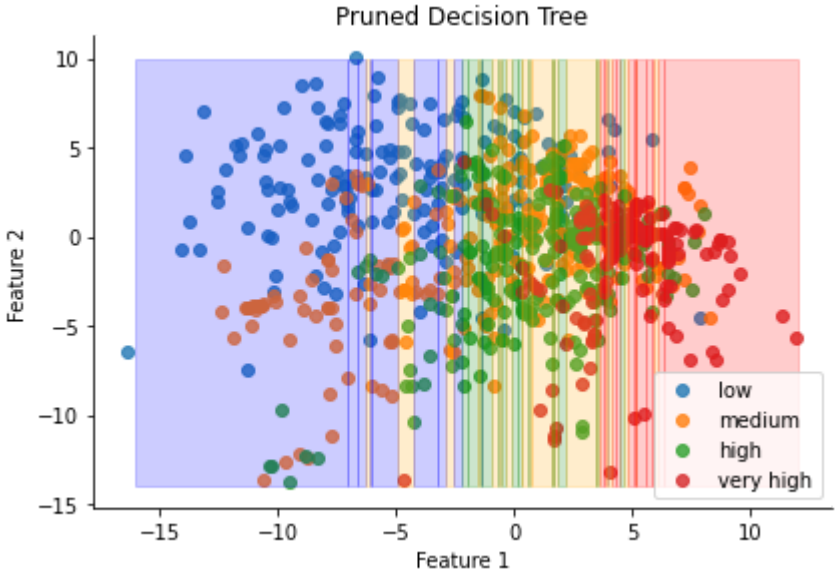


Accuracy for decision tree with max depth as 5 is 56.53266331658291

==========

Decision boundary on the training set before and after pruning

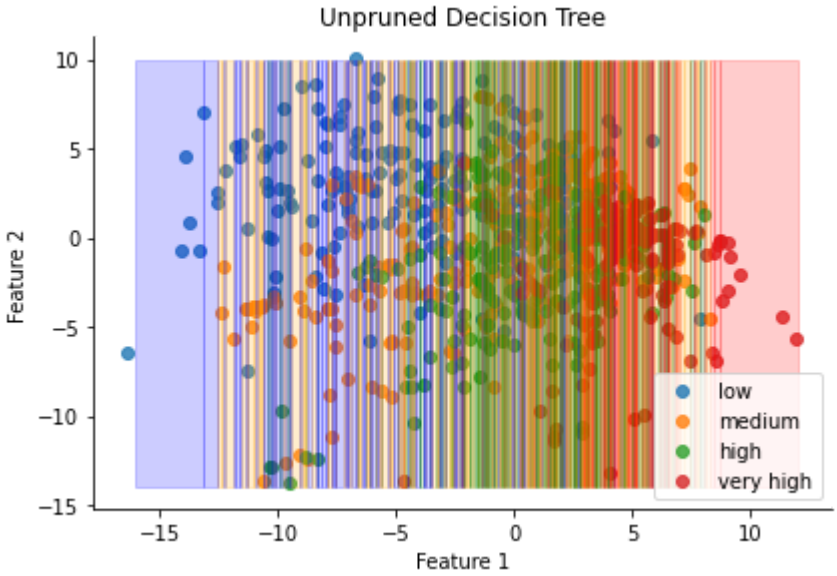


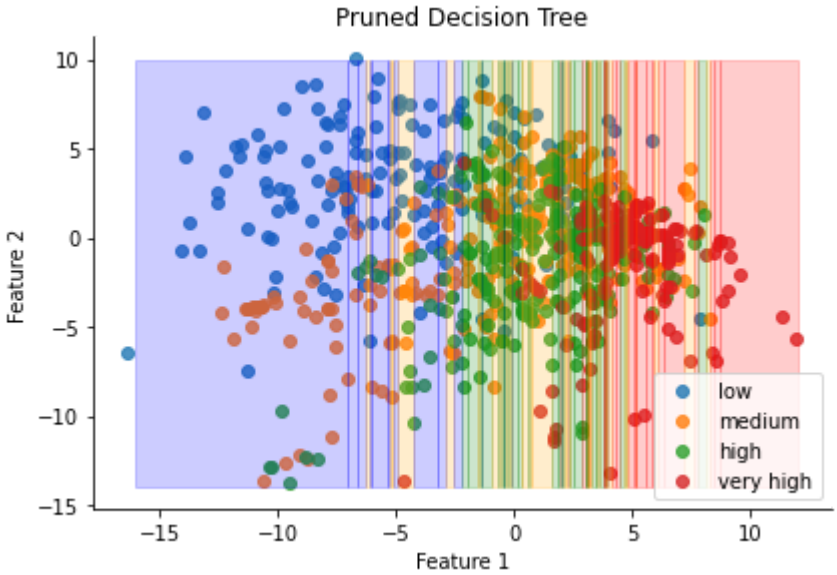


Accuracy for decision tree with max depth as 8 is 58.040201005025125

==========

Decision boundary on the training set before and after pruning

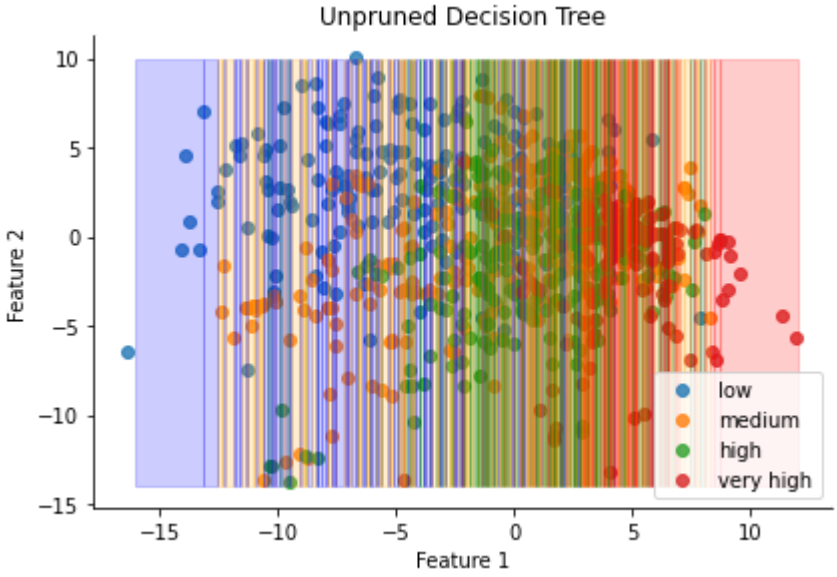


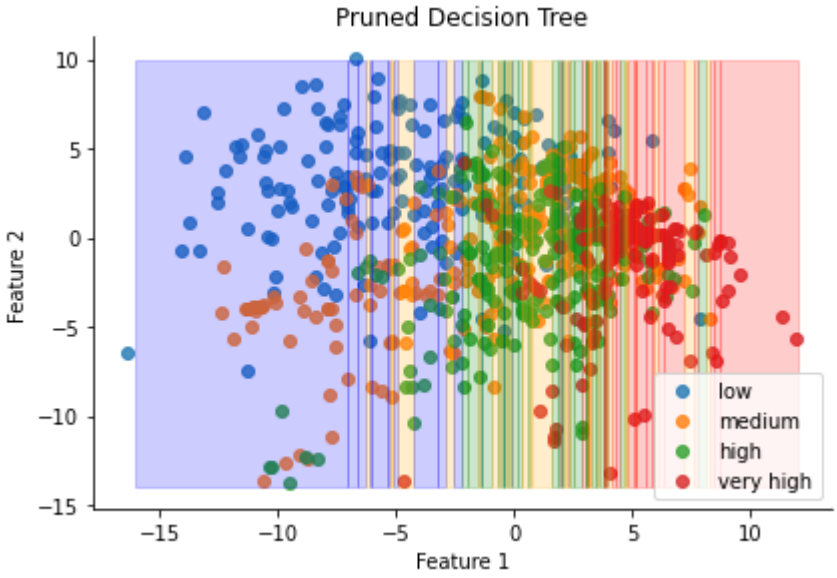


Accuracy for decision tree with max depth as 10 is 55.527638190954775

==========

Decision boundary on the training set before and after pruning

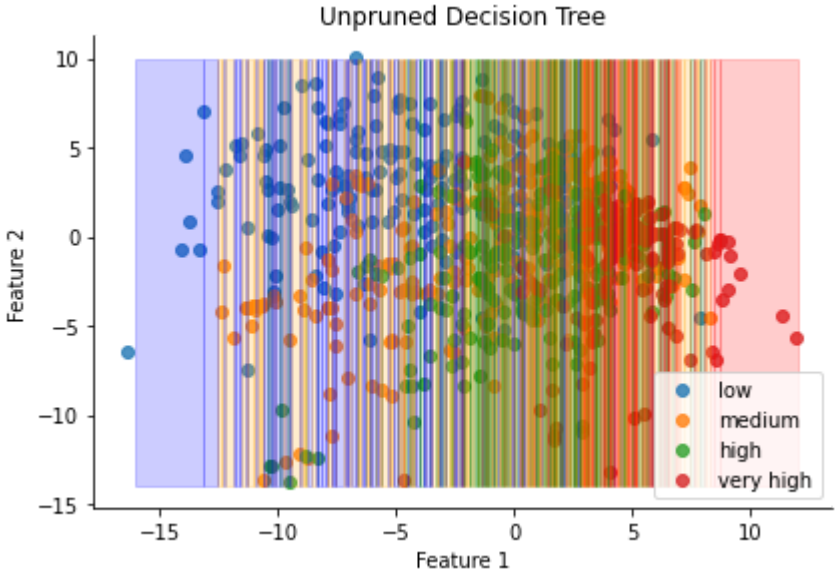


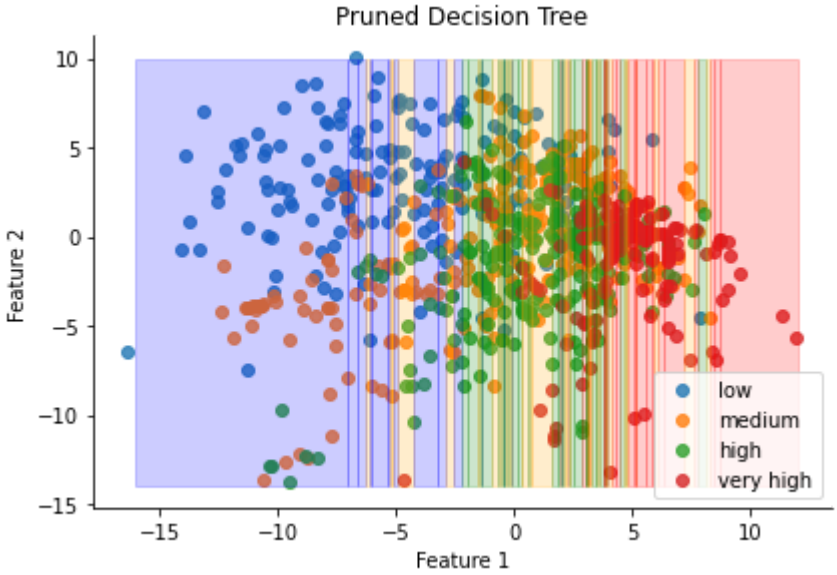


Accuracy for decision tree with max depth as 15 is 55.527638190954775

==========

Decision boundary on the training set before and after pruning

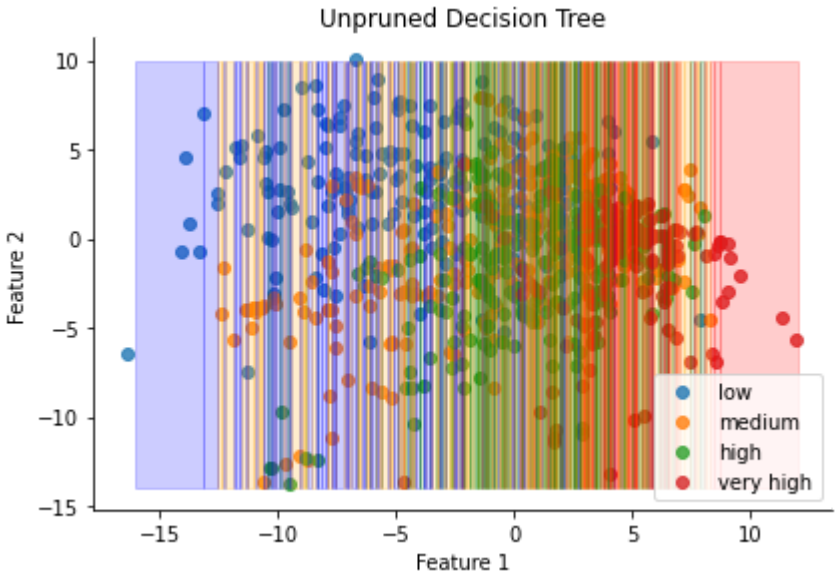


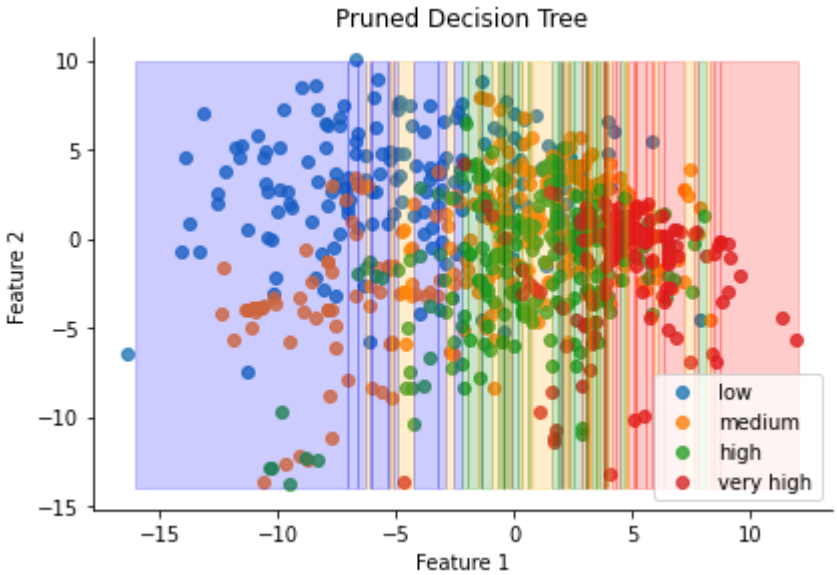


Accuracy for decision tree with max depth as 20 is 55.527638190954775

==========

Decision boundary on the training set before and after pruning

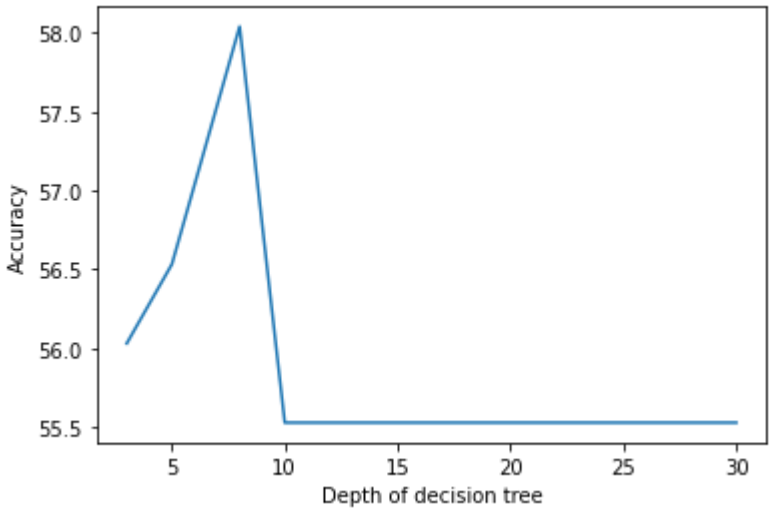




Accuracy for decision tree with max depth as 30 is 55.527638190954775

In [140…

```
# to find highest accuracy
plt.plot(depth_array,accuracy_array)
plt.xlabel("Depth of decision tree")
plt.ylabel("Accuracy")
plt.show()
```



In [141…

```
# finding highest accuracy
max_index = 0
highest_accuracy = accuracy_array[0]

for i in range(len(accuracy_array)):
    if accuracy_array[i] > highest_accuracy:
        highest_accuracy = accuracy_array[i]
        max_index = i

best_predictions = predictions_array[max_index]
```

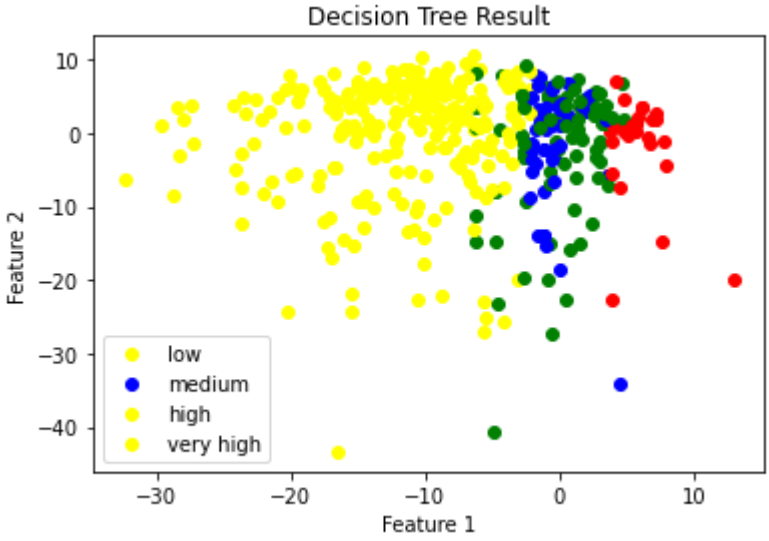
Visualizing results predicted by Decision Tree with actual values

In [142…

```
def plot_scatter_plot(values,title):
    for ind in range(len(best_predictions)):
        if(np.array(values)[ind] == 1):
            plt.scatter(x_test.iloc[ind][0],x_test.iloc[ind][1],color='yellow
        elif(np.array(values)[ind] == 2):
            plt.scatter(x_test.iloc[ind][0],x_test.iloc[ind][1],color='green'
        elif(np.array(values)[ind] == 3):
            plt.scatter(x_test.iloc[ind][0],x_test.iloc[ind][1],color='blue')
        else:
            plt.scatter(x_test.iloc[ind][0],x_test.iloc[ind][1],color='red')
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.legend(["low","medium","high","very high"])
    plt.title(title)
    plt.show()
```

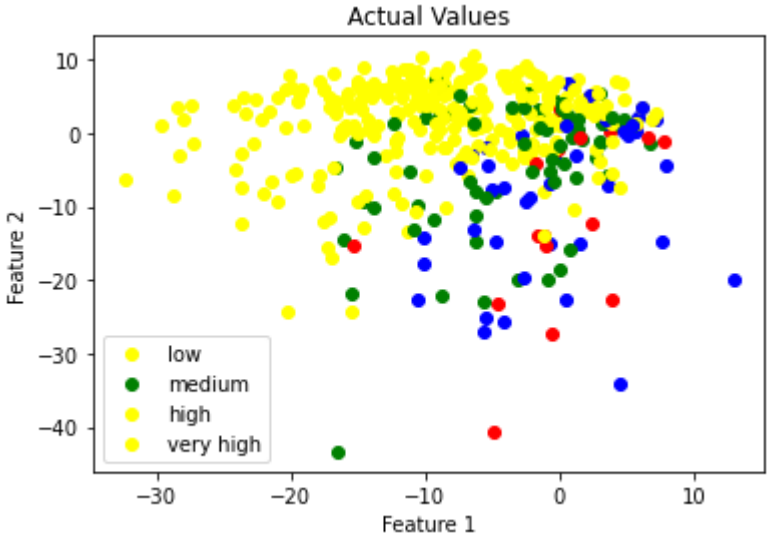
In [143…

```
# plotting scatter plot for prediction made by Decision Tree using first 2 fe
plot_scatter_plot(best_predictions,"Decision Tree Result")
```



In [144…

```
# plotting scatter plot for actual values of data points using first 2 featur
plot_scatter_plot(y_test,"Actual Values")
```



Classification Report for Decision Tree Model

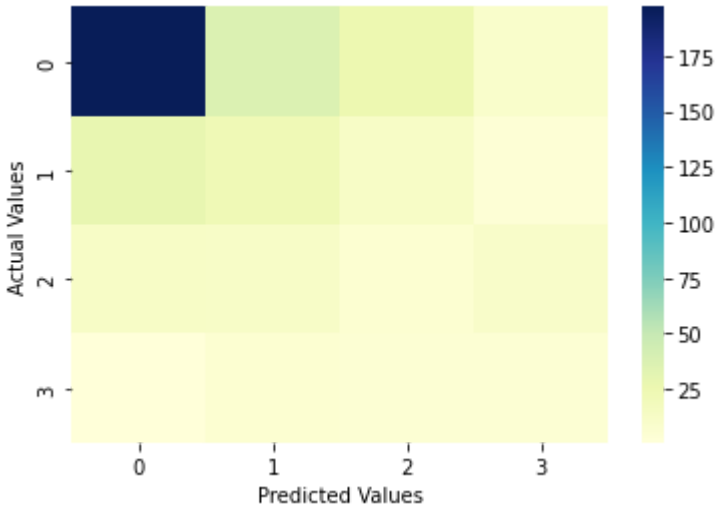
In [145…

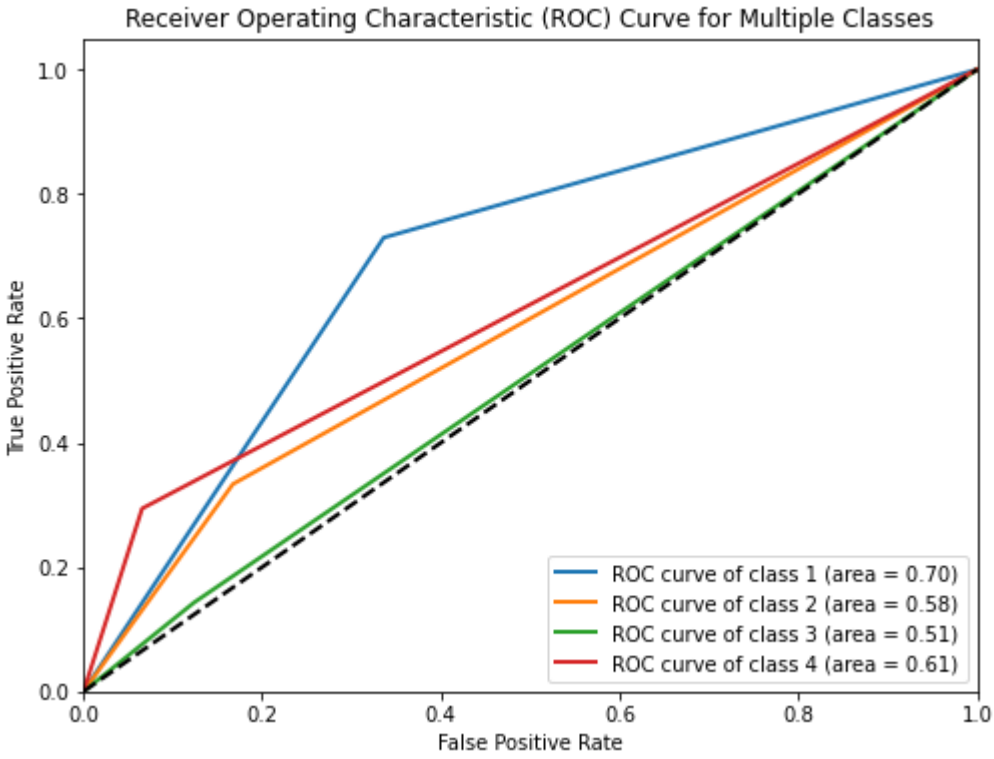
```
performance_metrics("Decision Tree",y_test,best_predictions)
```

```
Test Result:
================================================
Accuracy Score: 58.04%
________________________________________________
CLASSIFICATION REPORT:
                   1.0        2.0        3.0        4.0  accuracy    macro avg
\
precision     0.820833   0.294872   0.120000   0.166667  0.580402     0.350593
recall        0.729630   0.333333   0.142857   0.294118  0.580402     0.374984
f1-score      0.772549   0.312925   0.130435   0.212766  0.580402     0.357169
support     270.000000  69.000000  42.000000  17.000000  0.580402   398.000000

            weighted avg
precision       0.627750
recall          0.580402
```

```
f1-score       0.601194
support      398.000000
```

```
Confusion Matrix:
 [[197  37  26  10]
 [ 29  23  13   4]
 [ 13  12   6  11]
 [  1   6   5   5]]
```





3. Adaboost

Adaboost is one of the early successful algorithm wihtin the Boosting branch of Machine Learning.

- AdaBoost for classification is a supervised Machine Learning algo.
- It consists of iteratively training multiple stumps using feature data x and target label y
- After training, the model's ovearll prefictions come from a weighted sum of the stumps predictions

- For instances, consider n stumps, P_i be the prediction of i_{th} stump. whose corresponding weight is w_i then:

$$\text{Final Adaboost Prediction} = w_{stump1} * P1 + w_{stump2} * P2 + \ldots + w_{stumpn} * Pn$$

Algorithm of Adaboost

- AdaBoost trains a sequence of models with augmented sample weights generating 'confidence' coefficients α for each individual classifiers based on errors.

1. For a datset with N number of samples, we initialize weight of each data point with w_i = $\frac{1}{N}$. The weight of a datapoint represents the probability of selecting it during sampling.

1. Training Weak Clasifiers: After sampling the dataset, we train a weak classifier K_m using the training dataset. Here K_m is just a decision tree stump

- For m = 1 to M:
- sample the dataset using the weights $w_i^{(m)}$ to obtain the training samples x_i
- Fit a classifier K_m using all the training samples x_i
- Here K_m is the prediction of m_{th} stump

1. Calcuating Update Parameters from Error: Stumps in AdaBoost progressively learn from previous stump's mistakes through adjusting the weights of the dataset to accomodate the mistakes made by stumps at each iteration.

- The weights of the misclassified data will be increased, and the weights of correctly classified data will be decreased.
- As a result, as we go into further iterations, the training data will mostly comprise of data that is often misclassified by our stumps.
- When data is misclassified, $y * K_m(x) = -1$, then the weight update will be positive and the weights will exponentially increase for that data. When the data is correctly satisfied, $y * K_m(x) = 1$, the opposite happens.

- Let ϵ be the ratio of sum of weights for misclassified samples to that of the total sum of weights for samples. Compute the value of ϵ as follows:

$$\epsilon = \frac{\sum_{y_i \neq K_m(x_i)} w_i^{(m)}}{\sum_{y_i} w_i^{(m)}}$$

- In the above equation, y_i is the ground truth value of the target varibale and $w_i^{(m)}$ is the weight of the sample i at iteration m
- Larger ϵ means largers misclassification percentage, makes the confidence α_m decrease exponentially.
- Let confidence that the AdaBoost places on m_{th} stump's ability to classify the data be α_m. Compute α_m value as follows:

$$\alpha_m = \frac{1}{2}\log\left(\frac{1-\epsilon}{\epsilon}\right)$$

- Update all the weights as follows:

$$w_i^{(m+1)} = w_i^{(m)} . e^{-\alpha_m y K_m(x)}$$

1. New predictions can be computed by

$$K(x) = sign[\sum_{m=1}^{M} \alpha_m . K_m(x)]$$

- Low error means the α value is larger, and thus has higher importance in voting.
- AdaBoost predicts on -1 or 1. As long as the weighted sum is above 0, it predicts 1, else, it predicts -1.

3.1 Implementation of the Model

In [146…

```
# class for decision stump
class DecisionStump:
    # constructor will initialize value, feature_ind to split at, and thresho
    def __init__(self):
        self.value = 1
        self.feature_ind = None
        self.threshold = None
        self.alpha = None

    # function to predict outcome for this particular stump
    def predict(self,x_train):
        n_samples = x_train.shape[0]
        x_train_column = x_train[:, self.feature_ind]
        predictions = np.ones(n_samples)

        if self.value == 1:
            predictions[x_train_column < self.threshold] = -1
        else:
            predictions[x_train_column > self.threshold] = -1


        return predictions
```

In [147…

```
# class for implementing binary adaboost classifier
class Adaboost:

    # n_clf are the number of decision stumps, clfs will store each of the cl
    def __init__(self, n_clf=5):
        self.n_clf = n_clf
        self.clfs = []

    def train_model(self,x_train,y_train):
        n_samples, n_features = x_train.shape

        # initialize weights to 1/N
        w = []
        for i in range(n_samples):
            w.append(1/n_samples)
        w = np.array(w)

        # to store each decision stump
        self.clfs = []
```

```
        # iterate through classifiers
        for i in range(self.n_clf):

            # initializing a new decision stump
            clf = DecisionStump()
            min_error = float("inf")

            # finding best feature and threshold value
            for feature_i in range(n_features):

                x_train_column = x_train[:,feature_i]
                thresholds = np.unique(x_train_column)

                for threshold in thresholds:
                    # predict with value 1
                    p = 1

                    # initializing predictions to 1
                    predictions = np.ones(n_samples)
                    predictions[x_train_column < threshold] = -1

                    # calculating error
                    misclassified = w[y_train != predictions]
                    error = sum(misclassified)/sum(w)

                    if error > 0.5:
                        error = 1 - error
                        p = -1

                    # store the best configuration
                    if error < min_error:
                        clf.value = p
                        clf.threshold = threshold
                        clf.feature_ind = feature_i
                        min_error = error

            # calculate alpha
            epsilon = 1e-10 # to prevent divide by 0
            clf.alpha = 0.5 * np.log((1.0 - (min_error + epsilon)) / (min_err

            # calculate predictions and update weights
            predictions = clf.predict(x_train)
            w *= np.exp(-clf.alpha * y_train * predictions)

            # Save classifier
            self.clfs.append(clf)

    # function to find output
    def decision_function(self,x_train):
        clf_preds = [clf.alpha * clf.predict(x_train) for clf in self.clfs]
        y_pred = np.sum(clf_preds, axis=0)
        return y_pred

    # function to classify
    def predict(self, x_train):
        return np.sign(self.decision_function(x_train))
```

In [148…

```
class MulticlassAdaboost:

    # initializing classifiers array
    def __init__(self,n_stumps):
        self.classifiers = []
```

```
        self.n_stumps = n_stumps

    def train_model(self, x_train, y_train):
        self.classes = np.unique(y_train)
        self.classifiers = []

        for i, class_label in enumerate(self.classes):
            # treating the current class as positive and the rest as negative
            binary_labels = np.where(y_train == class_label, 1, -1)

            # create a binary Adaboost classifier
            # n_clf is number of stumps in that classifier
            classifier = Adaboost(n_clf=self.n_stumps)

            # training the ith classifier
            classifier.train_model(x_train, binary_labels)
            self.classifiers.append(classifier)

    # to predict the output
    def predict(self,x_test):

        # for each binary classifier, get the raw decision scores
        decision_scores = np.array([clf.decision_function(x_test) for clf in

        # Choose the class with the highest decision score as the final predi
        predictions = np.argmax(decision_scores, axis=0)+1 # to adjust the 0
        return predictions
```

In [149…

```
def find_accuracy(y_true,y_pred):
    count = 0
    for ind in range(len(y_true)):
        if y_pred[ind] == y_true[ind]:
            count += 1

    return count/len(y_true)
```

Training Multiclass Adaboost Model

In [150…

```
# creating instance of that class
multiclass_adaboost = MulticlassAdaboost(n_stumps = 5)
multiclass_adaboost.train_model(np.array(x_train), np.array(y_train))
```

In [151…

```
# finding predictions
y_pred = multiclass_adaboost.predict(np.array(x_test))
```

In [152…

```
# finding accuracy for this machine learning model
print(find_accuracy(y_test,y_pred))
```

```
0.48743718592964824
```

3.2 Insights drawn (plots, markdown explanations)

Comparing Accuracy of Adaboost for different number of decision stumps

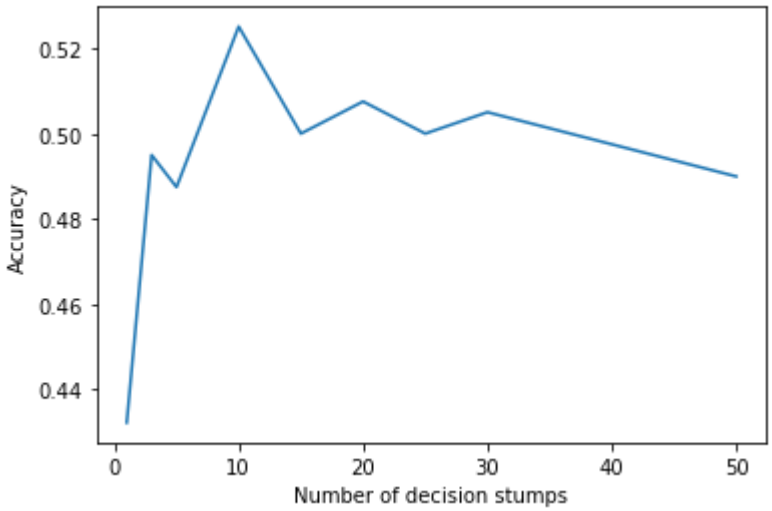
In [153…

```
num_decision_stumps = [1,3,5,7,8,10,15,20,25,30,50]
accuracy_array = []
predictions_array = []

for val in num_decision_stumps:
    # creating object for this class
    multiclass_adaboost = MulticlassAdaboost(n_stumps = val)
    multiclass_adaboost.train_model(np.array(x_train), np.array(y_train))

    y_pred = multiclass_adaboost.predict(np.array(x_test))
    predictions_array.append(y_pred)

    acc_val = find_accuracy(y_test,y_pred)
    accuracy_array.append(acc_val)

    print(f"Accuracy for Adaboost with {val} decision stumps is: {acc_val}\n"
```

```
Accuracy for Adaboost with 1 decision stumps is: 0.4321608040201005

Accuracy for Adaboost with 3 decision stumps is: 0.4949748743718593

Accuracy for Adaboost with 5 decision stumps is: 0.48743718592964824

Accuracy for Adaboost with 7 decision stumps is: 0.5025125628140703

Accuracy for Adaboost with 8 decision stumps is: 0.5100502512562815

Accuracy for Adaboost with 10 decision stumps is: 0.5251256281407035

Accuracy for Adaboost with 15 decision stumps is: 0.5

Accuracy for Adaboost with 20 decision stumps is: 0.507537688442211

Accuracy for Adaboost with 25 decision stumps is: 0.5

Accuracy for Adaboost with 30 decision stumps is: 0.5050251256281407

Accuracy for Adaboost with 50 decision stumps is: 0.4899497487437186
```

In [154…

```
plt.plot(num_decision_stumps,accuracy_array)
plt.xlabel("Number of decision stumps")
plt.ylabel("Accuracy")
plt.show()
```



In [155…

```
# finding highest accuracy
max_index = 0
highest_accuracy = accuracy_array[0]

for i in range(len(accuracy_array)):
    if accuracy_array[i] > highest_accuracy:
        highest_accuracy = accuracy_array[i]
        max_index = i

best_predictions = predictions_array[max_index]
```

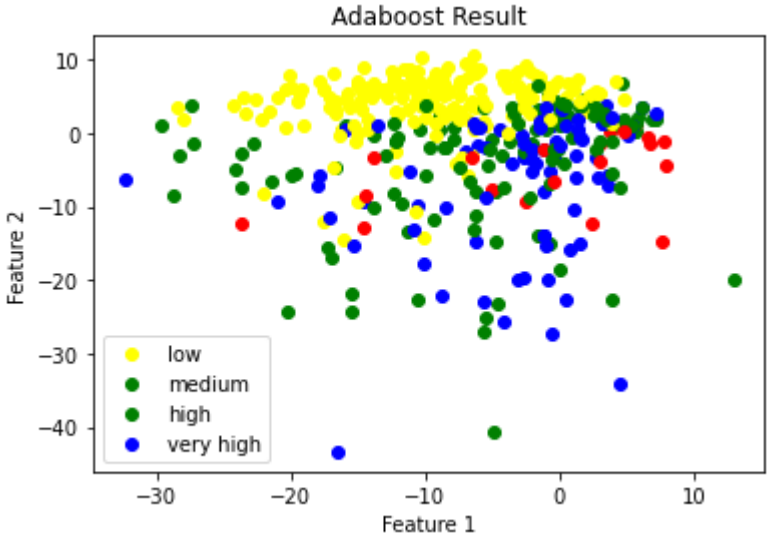
Visualizing results predicted by Adaboost with actual values

In [156…

```
def plot_scatter_plot(values,title):
    for ind in range(len(best_predictions)):
        if(np.array(values)[ind] == 1):
            plt.scatter(x_test.iloc[ind][0],x_test.iloc[ind][1],color='yellow
        elif(np.array(values)[ind] == 2):
            plt.scatter(x_test.iloc[ind][0],x_test.iloc[ind][1],color='green'
        elif(np.array(values)[ind] == 3):
            plt.scatter(x_test.iloc[ind][0],x_test.iloc[ind][1],color='blue')
        else:
            plt.scatter(x_test.iloc[ind][0],x_test.iloc[ind][1],color='red')
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.legend(["low","medium","high","very high"])
    plt.title(title)
    plt.show()
```

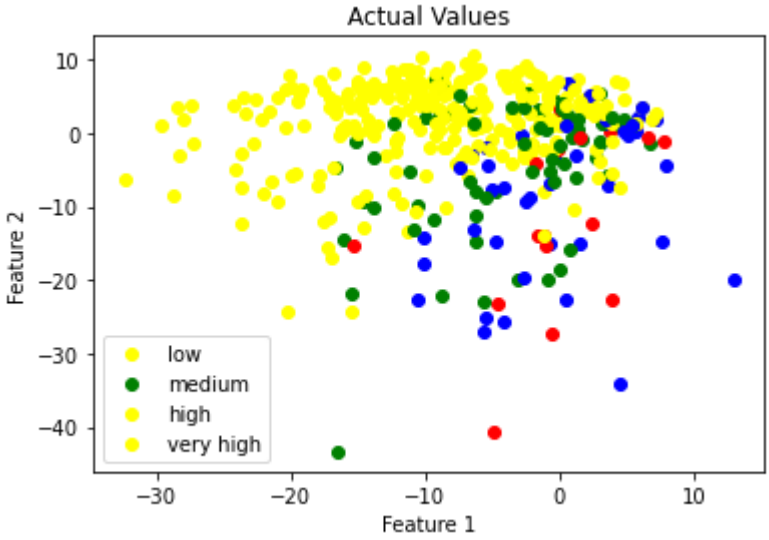
In [157…

```
# plotting scatter plot for prediction made by Adaboost using first 2 feature
plot_scatter_plot(best_predictions,"Adaboost Result")
```



In [158…

```
# plotting scatter plot for actual values of data points using first 2 featur
plot_scatter_plot(y_test,"Actual Values")
```



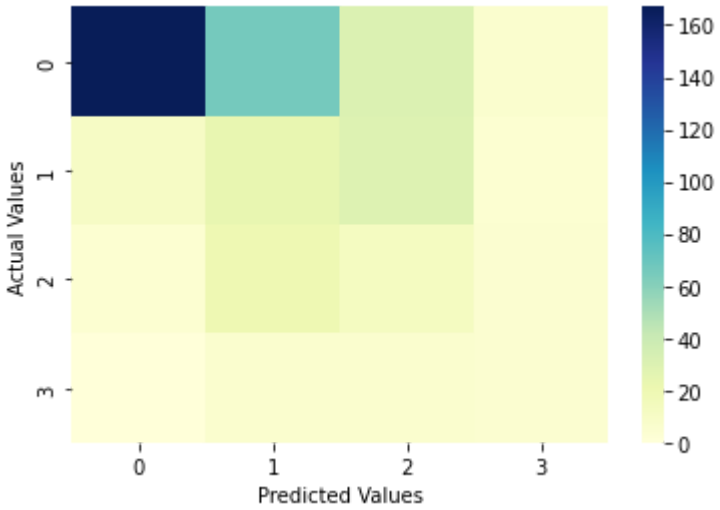
Classification Report for Adaboost Model

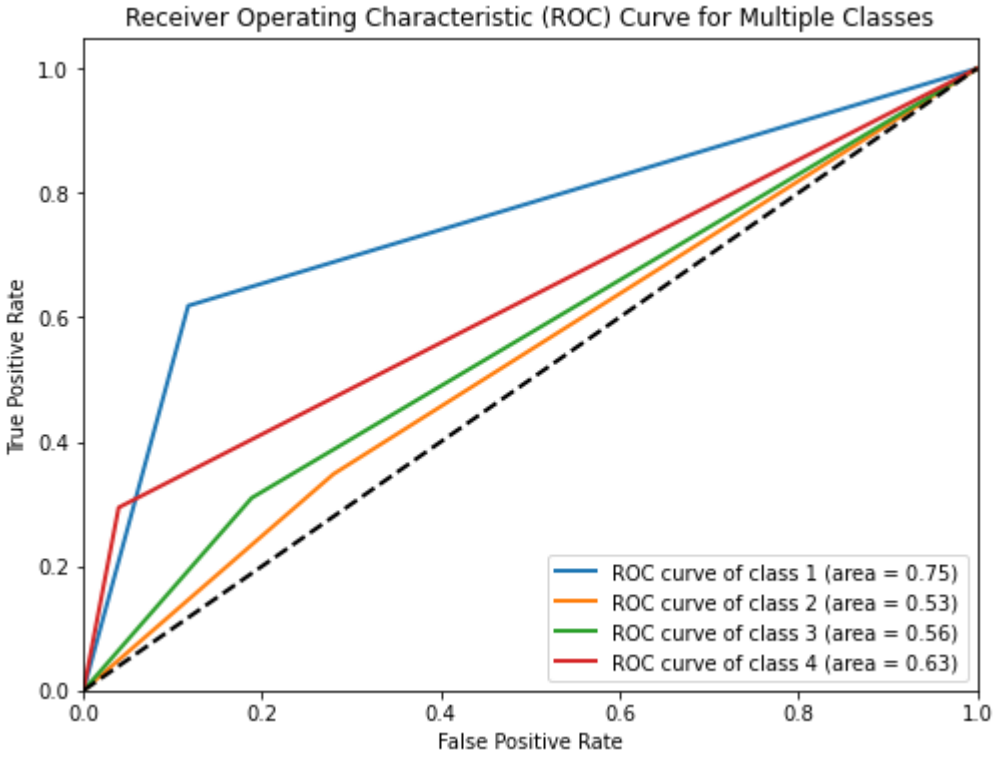
In [159…

```
performance_metrics("Adaboost",y_test,best_predictions)
```

```
Test Result:
================================================
Accuracy Score: 52.51%

_________________________________________________
CLASSIFICATION REPORT:
                    1.0         2.0         3.0         4.0  accuracy    macro avg
\
precision      0.917582    0.206897    0.162500    0.250000  0.525126     0.384245
recall         0.618519    0.347826    0.309524    0.294118  0.525126     0.392497
f1-score       0.738938    0.259459    0.213115    0.270270  0.525126     0.370446
support      270.000000   69.000000   42.000000   17.000000  0.525126   398.000000

             weighted avg
precision        0.686176
recall           0.525126
f1-score         0.580305
support        398.000000

_________________________________________________
```

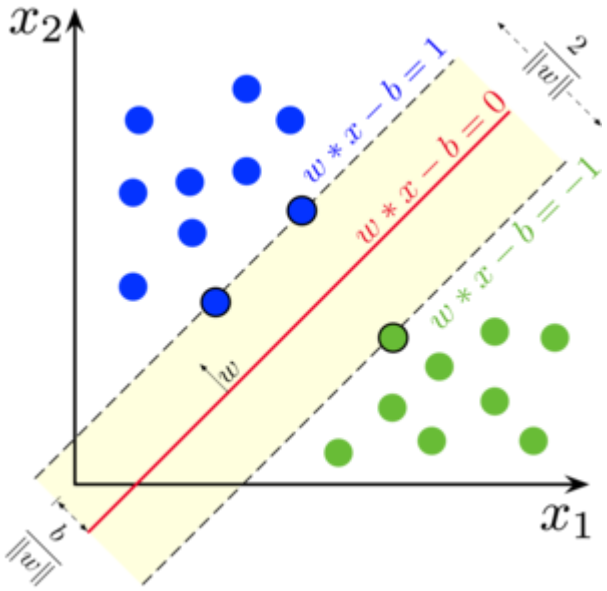
```
Confusion Matrix:
 [[167  66  31   6]
 [ 11  24  30   4]
 [  4  20  13   5]
 [  0   6   6   5]]
```





4. Multiclass SVM

- Aim is to find optimal hyperplane for data points that maximizes the margin.
- We will get output as set of weights, one for each feature, whose linear combination predicts the value of target attribute.
- To maximize the margin, we reduce the number of weights that are non-zero to a few.



- We solve the following constraint optimization problem

$$\max_{w,b} \frac{2}{||w||} \quad \text{such that} \quad \forall i : y_i(w \cdot x_i + b) \geq 1$$

- which can be written as,

$$\min_{w,b} \frac{||w||^2}{2} \quad \text{such that} \quad \forall i : y_i(w \cdot x_i + b) \geq 1$$

- Corresponding Lagrangian is

$$\mathcal{L}(w, b, \alpha_i) = \frac{1}{2}||w||^2 - \sum_{i=1}^{N} \alpha_i [y_i(w \cdot x_i + b) - 1] \quad \forall i : \alpha_i \geq 0$$

- The above indicates primal form of optimization problem, for which we can write equivalent Lagrangian Dual Problem

$$\max_{\alpha} \left[\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) \right] \quad \text{such that} \quad \forall i : \alpha_i \geq 0 \text{ and } \sum_{i=1}^{N} \alpha_i y_i = 0$$

- The dual optimization problem for the SVM is a quadratic programming problem. The solution to this quadratic programming problem yields the Lagrange multipliers for each point in the dataset.

- The weight vector is given by

$$w = \sum_{i=1}^{N} \alpha_i y_i x_i$$

- And bias term is

$$b = y_i - \sum_{j=1}^{N} \alpha_j y_j (x_j \cdot x_i)$$

- Only points with α > 0 define the hyperplane (contribute to the sum). These are called support vectors.

$$y = \text{sign}\left(\sum_{i \in \text{SV}} \alpha_i y_i (x_i \cdot x) + b\right)$$

- Here, α are the Lagrange multipliers, y are the class labels of the support vectors, x are the support vectors, x is the new example, and b is the bias term.

Kernel Transformation

- When the decision boundary between classes is not linear in the input feature space, we use non-linear Support Vector Machines (SVM) in which we use Kernel functions to map non-linear model in input space to a linear model in higher dimensional feature space.
- The kernel trick allows the SVM to operate in this higher-dimensional space without explicitly computing the transformation.
- The general form of the decision function in a non-linear SVM is

$$y = \text{sign}\left(\sum_{i \in \text{SV}}^{N} \alpha_i y_i K(x_i, x) + b\right)$$

- We will use the following Kernel functions

  - Linear Kernel

$$K(x_i, x_j) = x_i \cdot x_j$$

  - Polynomial Kernel

$$K(x_i, x_j) = (1 + x_i \cdot x_j)^p$$

  - Radial Basis Function (RBF) or Gaussian Kernel

$$K(x_i, x_j) = \exp\left(-\frac{1}{2\sigma^2}\|x_i - x_j\|^2\right)$$

4.1 Implementation of the Model

In [160…

```
# defining kernel functions

# linear kernel
```

```
def linear_kernel(xi,xj,dummy1=1,dummy2=1):
    return xi.dot(xj.T)

# polynomial kernel function
def polynomial_kernel(xi,xj,p=2,dummy=1):
    return (1 + xi.dot(xj.T))**p

# gaussian kernel function
def gaussian_kernel(xi,xj,dummy=1,sigma=0.1):
    return np.exp(-0.5*(1 /sigma ** 2) * np.linalg.norm(xi[:, np.newaxis] - x
```

In [161…

```
class Support_Vector_Machine:

    # initial constructor for SVM class
    def __init__(self, kernel='polynomial', degree=2, sigma=0.1, epoches=5000
        self.alpha = None
        self.b = 0
        self.degree = degree
        self.C = 1
        self.sigma = sigma
        self.epoches = epoches
        self.learning_rate = learning_rate
        self.kernel = kernel

        if self.kernel == 'linear':
            self.kernel = linear_kernel
        elif self.kernel == 'polynomial':
            self.kernel = polynomial_kernel
        elif self.kernel == 'gaussian':
            self.kernel =  gaussian_kernel

    # function for training binary SVM Model
    def train(self,x_train,y_train):
        self.x_train = x_train
        self.y_train = y_train
        self.alpha = np.random.random(x_train.shape[0])
        self.b = 0
        self.ones = np.ones(x_train.shape[0])

        # calculating yi*yj*K(xi, xj)
        y_mul_kernal = np.outer(y_train,y_train) * self.kernel(x_train,x_trai

        # performing gradient descent
        for i in range(self.epoches):

            # 1 - yk ∑αj*yj*K(xj, xk)
            gradient = self.ones - y_mul_kernal.dot(self.alpha)

            # α = α + η*(1 - yk ∑αj*yj*K(xj, xk)) to maximize
            self.alpha += self.learning_rate * gradient

            # checking if 0<α<C
            self.alpha[self.alpha < 0] = 0
            self.alpha[self.alpha > self.C] = self.C

            # ∑αi - (1/2) ∑i ∑j αi αj yi yj K(xi, xj)
            loss = np.sum(self.alpha) - 0.5 * np.sum(np.outer(self.alpha, sel

            alpha_index = np.where((self.alpha) > 0 & (self.alpha < self.C))[

        # for intercept b, we will only consider α which are 0<α<C
        b_values = []
        for index in alpha_index:
```

```
            b_values.append(y_train[index] - (self.alpha * y_train).dot(self.

        # finding b value
        # take mean or average of yi - ∑αj*yj*K(xi,xj)
        self.b = np.mean(b_values)

    # function to predict values
    def decision_function(self,x_test):
        return (self.alpha * self.y_train).dot(self.kernel(self.x_train, x_te
```

Training Multiclass SVM Model

In [162…

```
def find_predictions(kernel='linear',degree=2):

    # finding unique classes
    unique_classes = np.unique(y_train)
    # to store predictions made by each of the model
    dictionary_of_models = {}
    predictions = []

    # we will iterate over each class and train one vs rest model for that cl
    # points belonging to that class will be labeled as 1 and rest as -1
    for index in unique_classes:
        class_label = []  # Reset class_label for each class
        for label in y_train:
            if label == index:
                class_label.append(1)
            else:
                class_label.append(-1)

        # creating instance of model
        svm_model = Support_Vector_Machine(kernel=kernel,degree=degree)

        # training svm model for class index vs rest
        svm_model.train(np.array(x_train),class_label)

        # adding predictions
        predictions.append(svm_model.decision_function(np.array(x_test)))

    return predictions
```

In [163…

```
def find_final_predictions(kernel='linear',degree=2):
    predictions = find_predictions(kernel,degree=degree)

    # finding final predictions
    predictions = np.array(predictions).T
    final_predictions = np.argmax(predictions, axis=1) + 1  # Add 1 to align

    return final_predictions
```

In [164…

```
# function to find accuracy
def give_accuracy(y_test,y_pred):
    count = 0
    for i in range(len(y_test)):
        if(y_test[i]==y_pred[i]):
            count+=1

    accuracy = count/len(y_test)
    return accuracy
```

4.2 Insights drawn (plots, markdown explanations)

Comparing Accuracy with different Kernel functions

In [165…
```
accuracy_overall = []
predictions_array_overall = []
```

Linear Kernel Function

In [166…
```
final_predictions = find_final_predictions('linear')
predictions_array_overall.append(final_predictions)
accuracy = give_accuracy(y_test,final_predictions)
accuracy_overall.append(accuracy)
print("Accuracy for SVM with linear kernel is: ",accuracy)
```

```
Accuracy for SVM with linear kernel is:  0.6030150753768844
```

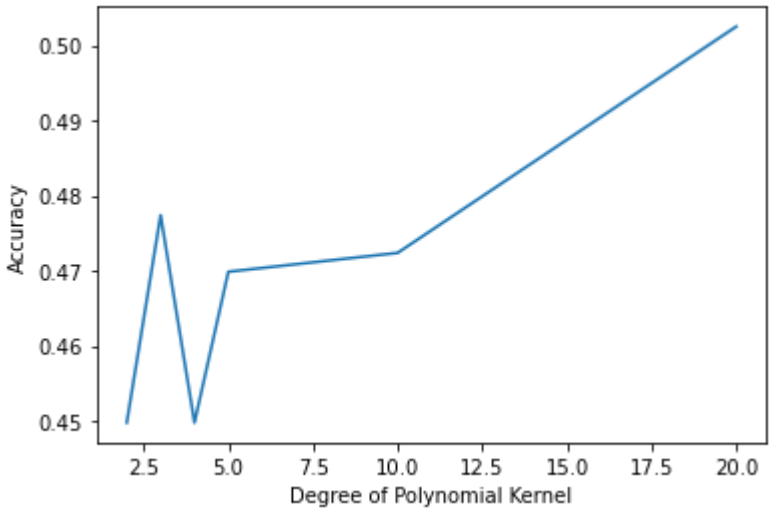
Polynomial Kernel Function

In [167…
```
accuracy_array = []
predictions_array = []
degree_array = [2,3,4,5,10,20]

for degree_val in degree_array:
    final_predictions = find_final_predictions('polynomial',degree=degree_val
    predictions_array.append(final_predictions)
    accuracy = give_accuracy(y_test,final_predictions)
    accuracy_array.append(accuracy)
    print(f"Accuracy for SVM with polynomial kernel (degree={degree_val}) is:
```

```
Accuracy for SVM with polynomial kernel (degree=2) is: 0.44974874371859297
Accuracy for SVM with polynomial kernel (degree=3) is: 0.47738693467336685
Accuracy for SVM with polynomial kernel (degree=4) is: 0.44974874371859297
Accuracy for SVM with polynomial kernel (degree=5) is: 0.46984924623115576
Accuracy for SVM with polynomial kernel (degree=10) is: 0.4723618090452261
Accuracy for SVM with polynomial kernel (degree=20) is: 0.5025125628140703
```

In [168…
```
plt.plot(degree_array,accuracy_array)
plt.xlabel("Degree of Polynomial Kernel")
plt.ylabel("Accuracy")
plt.plot()
```

Out[168…
```
[]
```



In [169…

```
# finding highest accuracy
max_index = 0
highest_accuracy = accuracy_array[0]

for i in range(len(accuracy_array)):
    if accuracy_array[i] > highest_accuracy:
        highest_accuracy = accuracy_array[i]
        max_index = i

best_predictions_polynomial = predictions_array[max_index]
```

In [170…

```
accuracy_overall.append(accuracy_array[max_index])
predictions_array_overall.append(best_predictions_polynomial)
```

Gaussian Kernel Function

In [171…

```
final_predictions = find_final_predictions('gaussian')
predictions_array_overall.append(final_predictions)
accuracy = give_accuracy(y_test,final_predictions)
accuracy_overall.append(accuracy)
print("Accuracy for SVM with gaussian kernel is: ",accuracy)
```

```
Accuracy for SVM with gaussian kernel is:  0.17336683417085427
```

In [172…

```
# finding best prediction
max_index = 0
highest_accuracy = accuracy_overall[0]

for i in range(len(accuracy_overall)):
    if accuracy_overall[i] > highest_accuracy:
        highest_accuracy = accuracy_overall[i]
        max_index = i

best_predictions = predictions_array_overall[max_index]
```

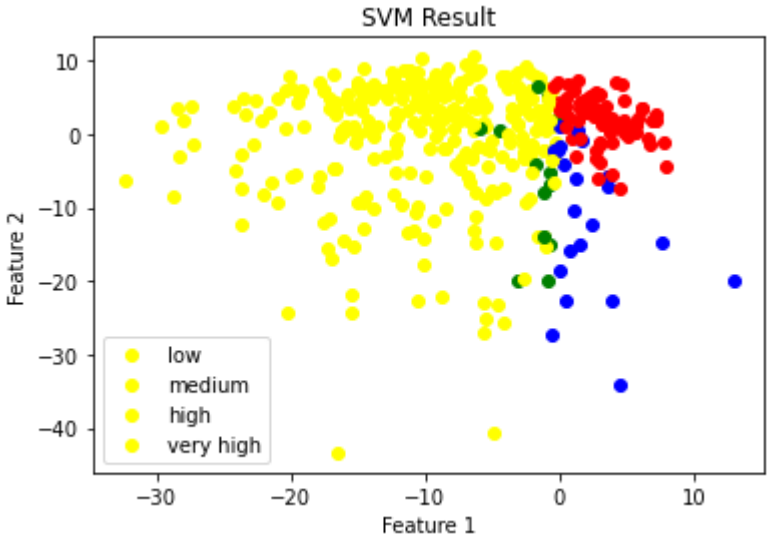
Visualizing results predicted by Multiclass SVM with actual values

In [173…

```
def plot_scatter_plot(values,title):
    for ind in range(len(best_predictions)):
        if(np.array(values)[ind] == 1):
            plt.scatter(x_test.iloc[ind][0],x_test.iloc[ind][1],color='yellow
        elif(np.array(values)[ind] == 2):
            plt.scatter(x_test.iloc[ind][0],x_test.iloc[ind][1],color='green'
        elif(np.array(values)[ind] == 3):
            plt.scatter(x_test.iloc[ind][0],x_test.iloc[ind][1],color='blue')
        else:
            plt.scatter(x_test.iloc[ind][0],x_test.iloc[ind][1],color='red')
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.legend(["low","medium","high","very high"])
    plt.title(title)
    plt.show()
```

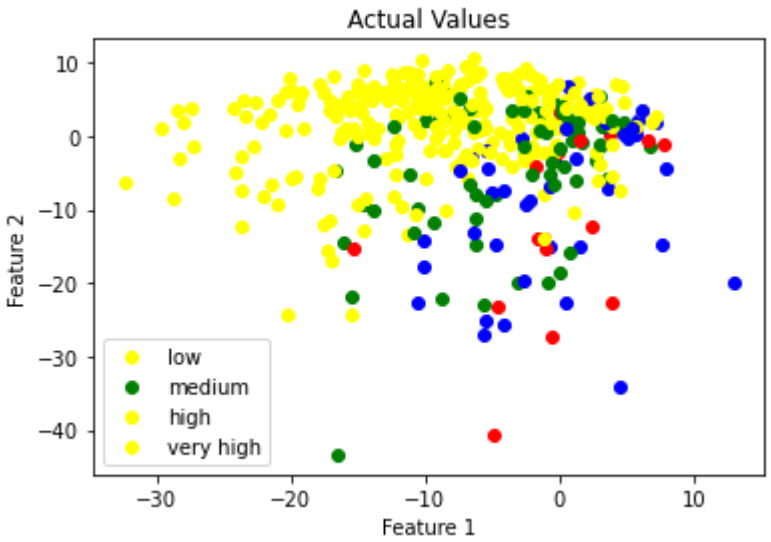
In [174…

```
# plotting scatter plot for prediction made by SVM using first 2 features of
plot_scatter_plot(best_predictions,"SVM Result")
```



In [175…

```
# plotting scatter plot for actual values of data points using first 2 featur
plot_scatter_plot(y_test,"Actual Values")
```

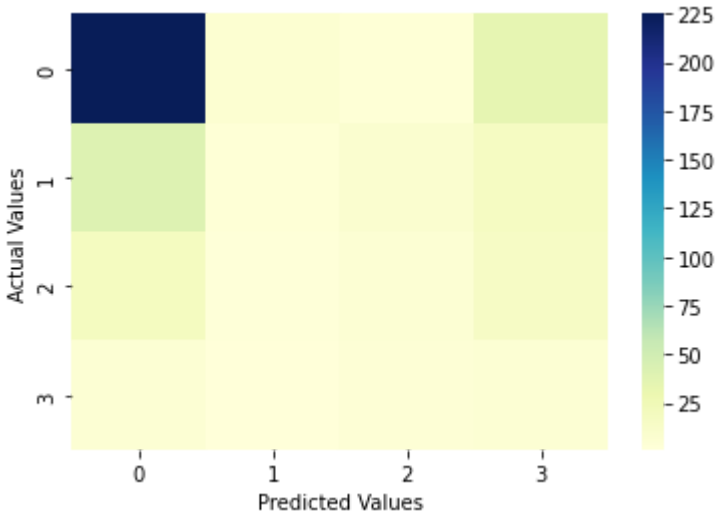


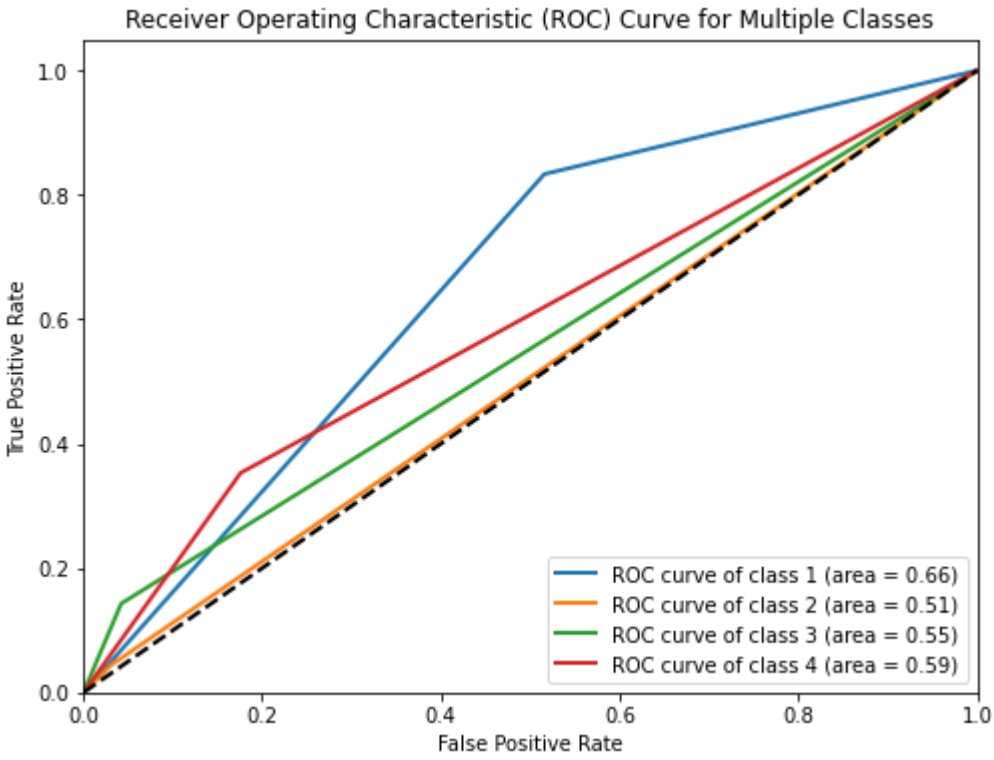
Classification Report for SVM Model

In [176…

```
performance_metrics("SVM",y_test,best_predictions)
```

```
Test Result:
================================================
Accuracy Score: 60.30%
_________________________________________________
CLASSIFICATION REPORT:
                    1.0        2.0        3.0        4.0  accuracy    macro avg
\
precision      0.773196   0.230769   0.285714   0.082192  0.603015     0.342968
recall         0.833333   0.043478   0.142857   0.352941  0.603015     0.343152
f1-score       0.802139   0.073171   0.190476   0.133333  0.603015     0.299780
support      270.000000  69.000000  42.000000  17.000000  0.603015   398.000000

             weighted avg
precision        0.598199
recall           0.603015
f1-score         0.582646
support        398.000000
_________________________________________________
Confusion Matrix:
 [[225   7   3  35]
 [ 41   3   8  17]
 [ 19   2   6  15]
 [  6   1   4   6]]
```





Comparison of insights drawn from the models

In [177…

```
MLA_columns = ['Name', 'Test Accuracy', 'Precision', 'Recall']
MLA_compare = pd.DataFrame(columns=MLA_columns)
index = 0

# MLA_compare
count = 0
for val in MLA:
    MLA_compare.loc[count] = MLA[val]
    count += 1
MLA_compare.sort_values(by='Test Accuracy', ascending=False, inplace=True)
MLA_compare
```

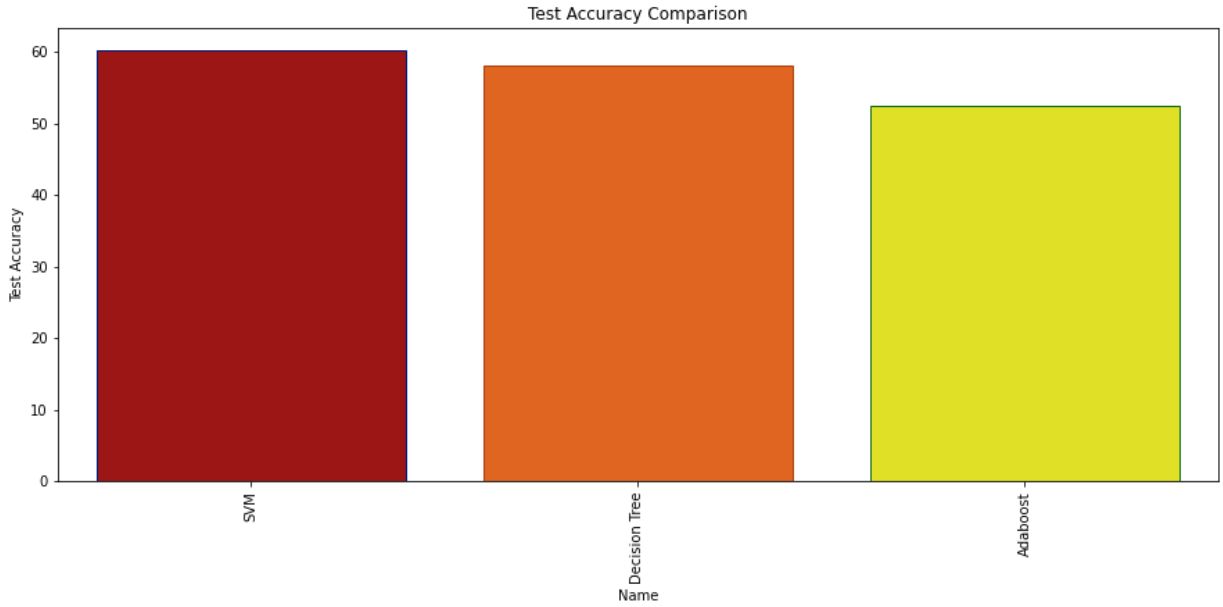
Out[177…

	Name	Test Accuracy	Precision	Recall
2	SVM	60.301508	0.598199	0.603015
0	Decision Tree	58.040201	0.627750	0.580402
1	Adaboost	52.512563	0.686176	0.525126

Test Accuracy Comparison

In [178…

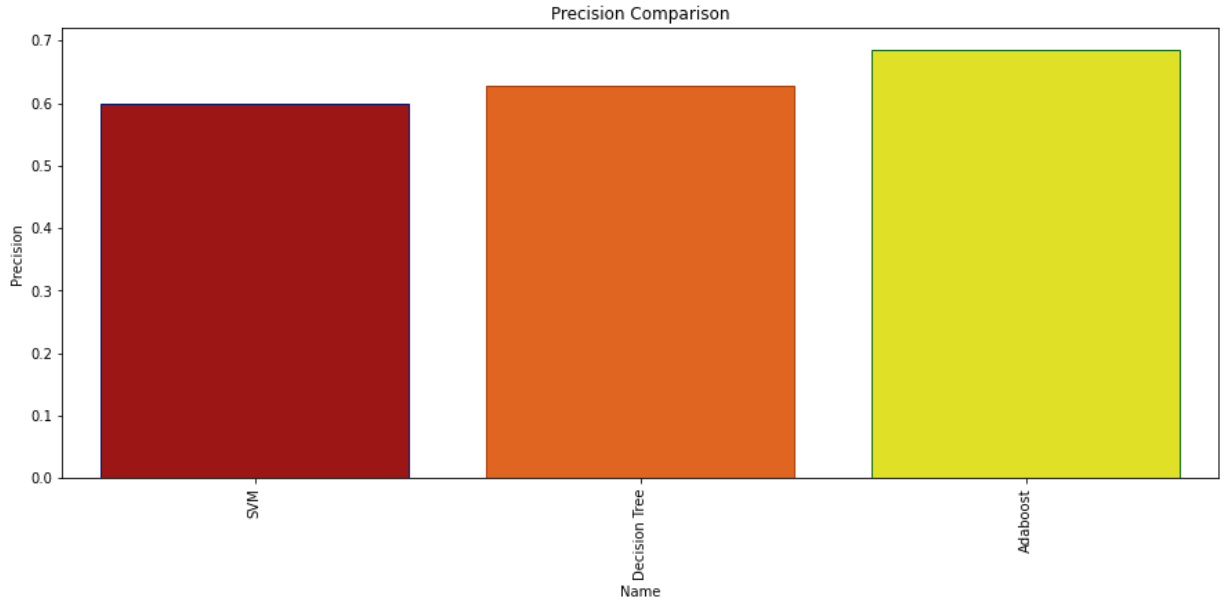
```
plt.subplots(figsize=(15,6))
sns.barplot(x="Name", y="Test Accuracy",data=MLA_compare,palette='hot',edgeco
plt.xticks(rotation=90)
plt.title('Test Accuracy Comparison')
plt.show()
```



Precision Comparison

In [179…

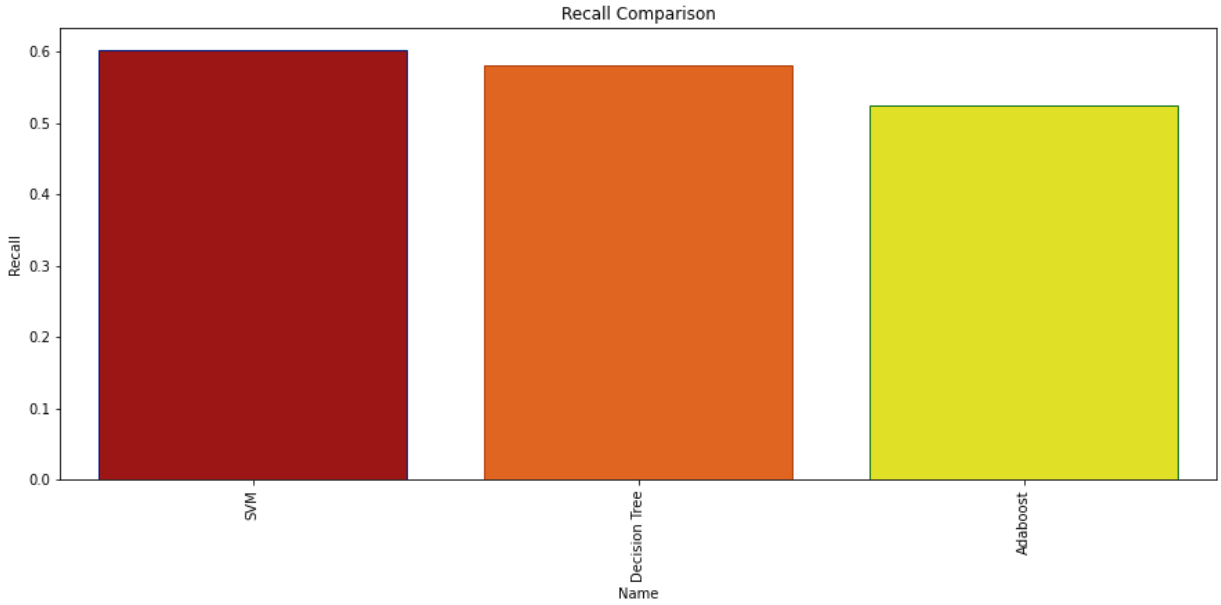
```
plt.subplots(figsize=(15,6))
sns.barplot(x="Name", y="Precision",data=MLA_compare,palette='hot',edgecolor=
plt.xticks(rotation=90)
plt.title('Precision Comparison')
plt.show()
```



Recall Comparison

In [180…

```
plt.subplots(figsize=(15,6))
sns.barplot(x="Name", y="Recall",data=MLA_compare,palette='hot',edgecolor=sns
plt.xticks(rotation=90)
plt.title('Recall Comparison')
plt.show()
```



Best Model

As we can observe SVM has highest accuracy and recall score among other machine learning models, so SVM performs better than other models on this given dataset.

5. References

- https://numpy.org/doc/stable/reference/generated/numpy.dot.html
- https://pandas.pydata.org/
- https://matplotlib.org/
- https://www.geeksforgeeks.org/machine-learning/
- https://seaborn.pydata.org/