Final Project

May 20, 2020

1 By Julian Murillo

2 CPSC 392-02

3 Chapman University

Boston Housing Data Source: https://www.kaggle.com/kyasar/boston-housing#boston_housing.csv

General Dataset Information: This Dataset contains 14 features (13 continuous and 1 binary and 1 target variable:'medv') and 506 observations/records. The reason I chose this dataset is because I have always been interested in housing and housing prices. The dataset does not contain any missing values and is ideal for regression analysis as well as classification type methods.

NOTE: I am deviating very slightly from the "Analysis Plan" but the questions to be answered and most of the analysis are the same.

[108]:
```
# Attribute Information:

# 1.) Crim: per capita crime rate by town
# 2.) Zn: proportion of residential land zoned for lots over 25,000 sq.ft.
# 3.) indus: proportion of non-retail business acres per town
# 4.) Chas : Charles River dummy variable (= 1 if tract bounds river; 0␣
 ↪otherwise)
# 5.) Nox: nitric oxides concentration (parts per 10 million)
# 6.) Rm: average number of rooms per housing
# 7.) Age: proportion of owner-occupied units built prior to 1940
# 8.) Dis: weighted distances to five Boston employment centres
# 9.) Rad: index of accessibility to radial highways
# 10.) Tax:full-value property-tax rate per $10,000
# 11.) Ptratio:pupil-teacherratiobytown
# 12.) Black:1000(Bk-0.63)^2 whereBk is the proportion of blacks bytown
# 13.) Lstat:% lower status of the population
# 14.) medv:Medianvalueofowner-occupiedhomesin$1000's

import os
os.getcwd()
```

[108]: '/Users/julianjr.'

[109]:
```
# Import libraries and read-in data

# General
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
from plotnine import *
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.preprocessing import StandardScaler #Z-score variables

# Cross-Validation
from sklearn.model_selection import KFold # k-fold cv
from sklearn.model_selection import LeaveOneOut #LOO cv
from sklearn.model_selection import cross_val_score # cross validation metrics␣
↪from sklearn.model_selection import cross_val_predict # cross validation␣
↪metrics

# Correlation Matrix
import seaborn as sn

# LOGISTIC Regression
import statsmodels.api as sm
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot

# Decision Tree
from sklearn.tree import DecisionTreeClassifier

# Classification Model Performance
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import plot_confusion_matrix

# Set Seed: So output is reproducable
import random
random.seed(1968)


%precision %.7g


# Linear Regression, Ridge and Lasso Models
```

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.linear_model import RidgeCV, LassoCV

# Error metrics/ model performance
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

# Clustering/unserpervised learning
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score

# More Plots
import joypy
%matplotlib inline

import matplotlib.pyplot as plt

boston = pd.read_csv("/Users/julianjr./Desktop/DataScience/FinalProject/
 ↪boston_housing.csv")
```

3.1 Methods:

For ALL models used to answer each question, I will be utilizing 12/13 potential predictor variables to start and I will be standardizing all of them (all continuous on different scales). I chose to exclude 'chas' from the modeling processes because when analyzing the data visually, it did not seem to have any bearing on whether or not a home was pricey/ any affect on the median value of homes or any of the other variables I was trying to predict. For the Logistic Regression and Decision Tree models, I will be using K-fold cross validation as to reduce the likelyhood of overfitting. In later revisions I will compare the testing accuracy with the training accuracy to evaluate the fit of the Logistic Regression and Decision Tree Models, but for now I will just rely on K-fold and hope it did the trick. Lastly, for the Lasso/Ridge Regression Models, I will be comparing the testing and training error to evaluate model performance and fit of the model instead of relying on K-fold. In later revisions on this section, I will impliment K-fold if Lasso and Ridge are overfit.

3.2 Part I: Explore and Visualize Data

[110]:
```
boston.head()
```

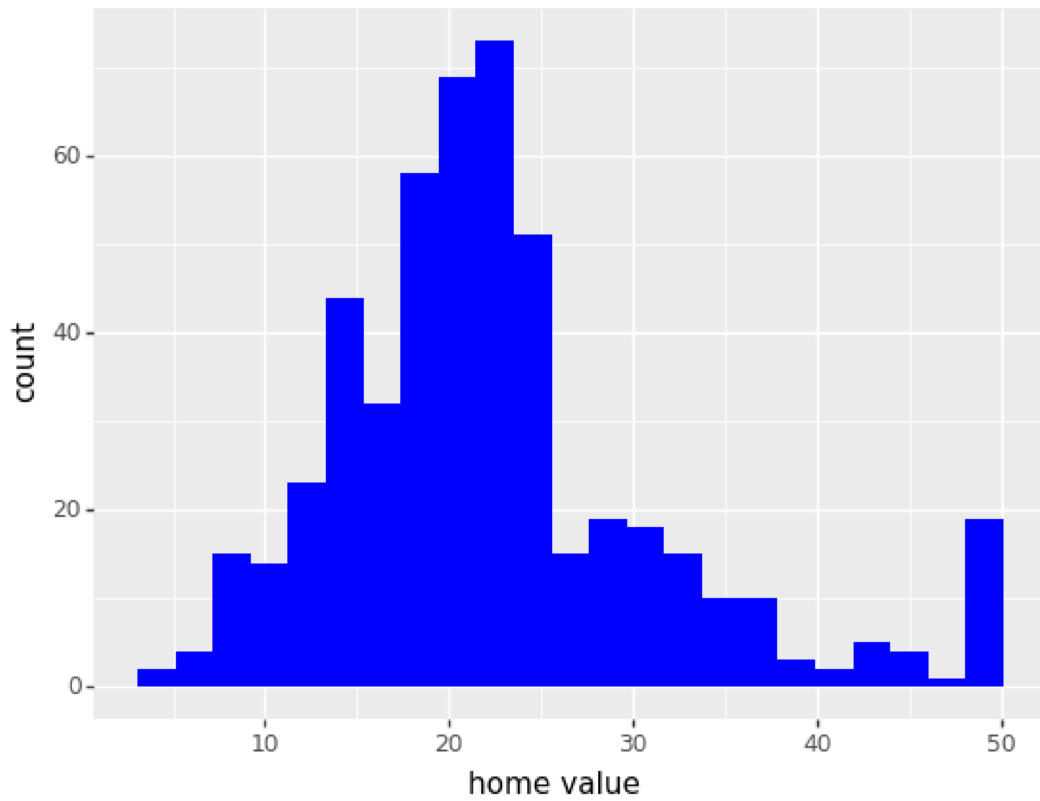
[110]:
```
      crim    zn  indus  chas    nox     rm   age     dis  rad  tax  ptratio  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222     18.7
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222     18.7

    black  lstat  medv
0  396.90   4.98  24.0
```

```
1  396.90   9.14  21.6
2  392.83   4.03  34.7
3  394.63   2.94  33.4
4  396.90   5.33  36.2
```

[111]:
```
boston.isnull().sum()

# No null values
```

[111]:
```
crim       0
zn         0
indus      0
chas       0
nox        0
rm         0
age        0
dis        0
rad        0
tax        0
ptratio    0
black      0
lstat      0
medv       0
dtype: int64
```

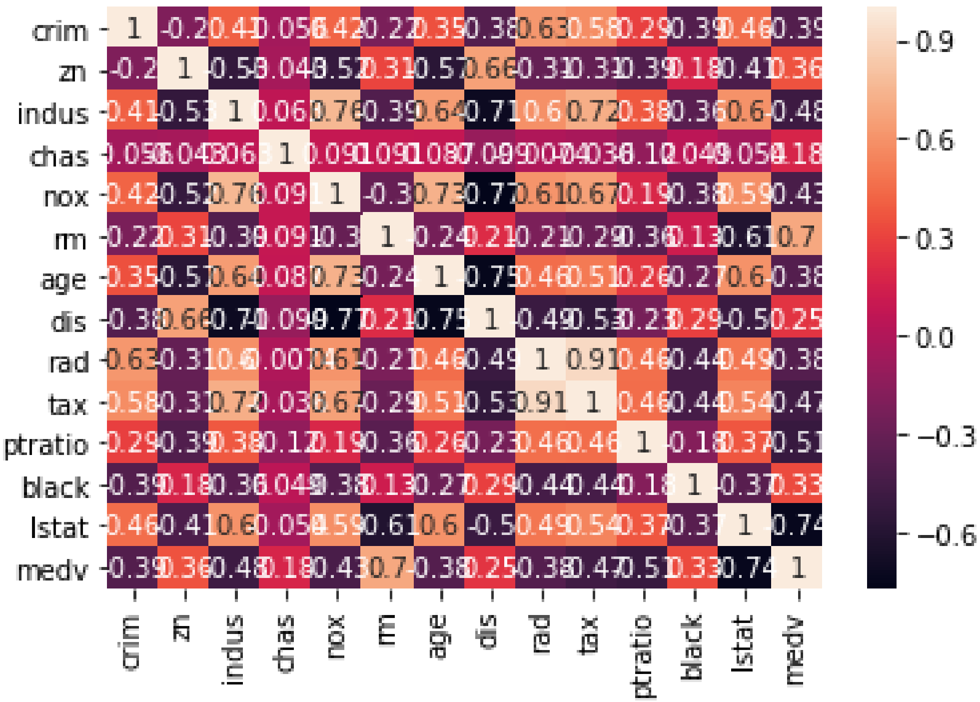
[112]:
```
# Target for first model looks pretty normally distributed
(ggplot(boston, aes(x = 'medv')) + geom_histogram(fill = "blue") + theme_grey()␣
 ↪+ xlab('home value'))
```



[112]: <ggplot: (7557272265)>

[113]:
```
corrMatrix = boston.corr()
sn.heatmap(corrMatrix, annot=True)

# Variables don't seem super highly correlated (.9 or above)
```

[113]: <matplotlib.axes._subplots.AxesSubplot at 0x1c273c6f90>



[114]:
```
# Create Binary Dummy Variable for Classification Models

# Coerce a boolean to an int by just multiplying it by one
boston['PriceyHome'] = (boston["medv"] >= boston["medv"].median()) * 1

# Factor chas variable
# boston['chas'] = boston['chas'].astype('category')... did not end up using␣
↪chas variable
boston = boston.drop(['chas'], axis = 1)
```

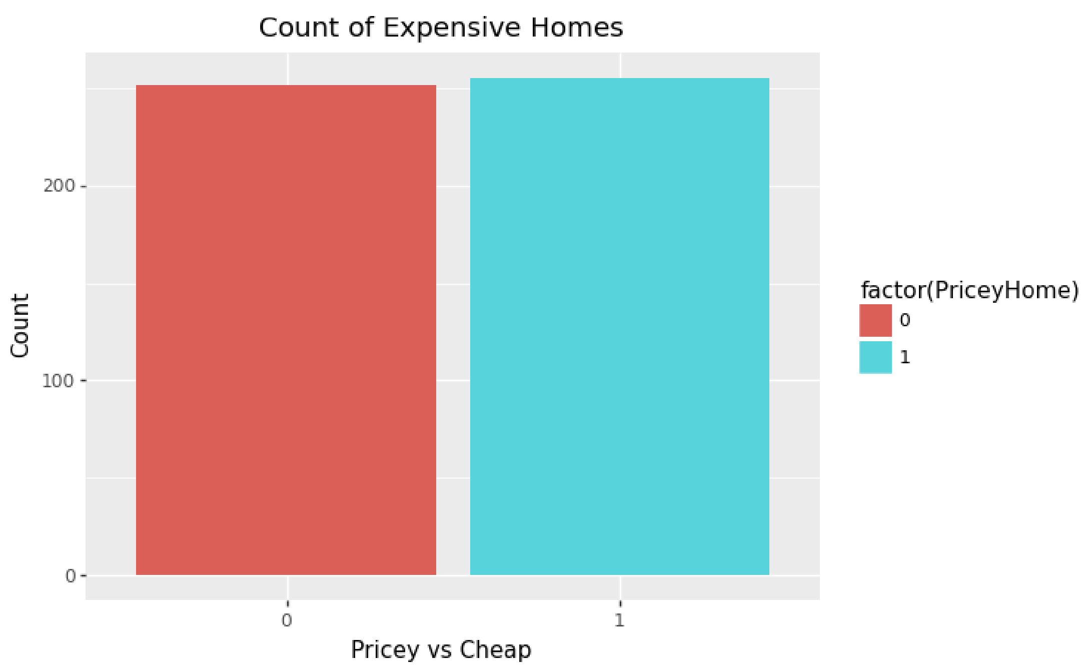
[115]:
```
boston.head(10)
```

[115]:

	crim	zn	indus	nox	rm	age	dis	rad	tax	ptratio	\
0	0.00632	18.0	2.31	0.538	6.575	65.2	4.0900	1	296	15.3	
1	0.02731	0.0	7.07	0.469	6.421	78.9	4.9671	2	242	17.8	
2	0.02729	0.0	7.07	0.469	7.185	61.1	4.9671	2	242	17.8	
3	0.03237	0.0	2.18	0.458	6.998	45.8	6.0622	3	222	18.7	
4	0.06905	0.0	2.18	0.458	7.147	54.2	6.0622	3	222	18.7	
5	0.02985	0.0	2.18	0.458	6.430	58.7	6.0622	3	222	18.7	
6	0.08829	12.5	7.87	0.524	6.012	66.6	5.5605	5	311	15.2	
7	0.14455	12.5	7.87	0.524	6.172	96.1	5.9505	5	311	15.2	
8	0.21124	12.5	7.87	0.524	5.631	100.0	6.0821	5	311	15.2	
9	0.17004	12.5	7.87	0.524	6.004	85.9	6.5921	5	311	15.2	

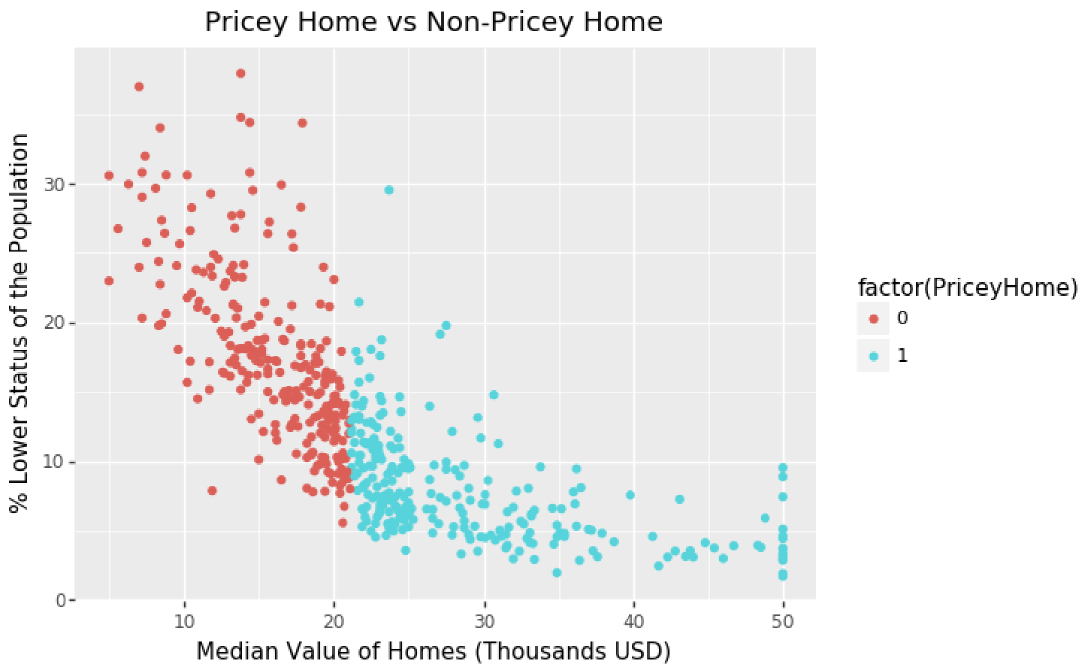
```
    black  lstat  medv  PriceyHome
0  396.90   4.98  24.0           1
1  396.90   9.14  21.6           1
2  392.83   4.03  34.7           1
3  394.63   2.94  33.4           1
4  396.90   5.33  36.2           1
5  394.12   5.21  28.7           1
6  395.60  12.43  22.9           1
7  396.90  19.15  27.1           1
8  386.63  29.93  16.5           0
9  386.71  17.10  18.9           0
```

[116]:
```
# Awesome... doesn't look like we'll run into any class-imbalance problems here.
(ggplot(boston , aes(x = 'factor(PriceyHome)')) + geom_bar(aes(fill =␣
↪'factor(PriceyHome)')) +
labs(x = 'Pricey vs Cheap', y = 'Count', title = 'Count of Expensive Homes'))
```



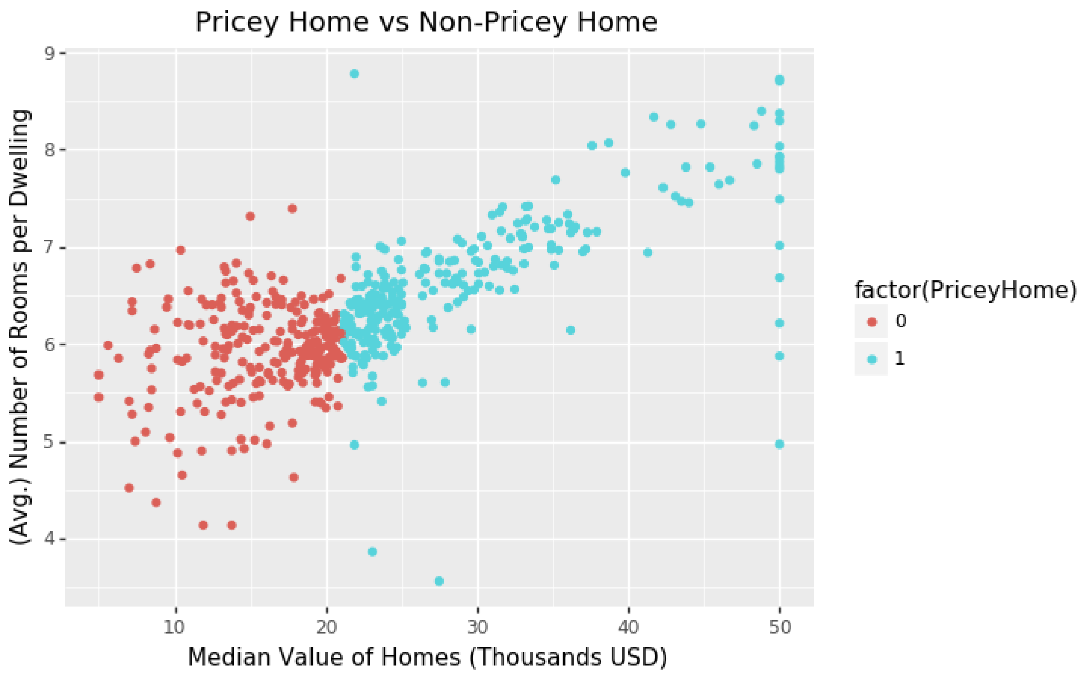
[116]: <ggplot: (7555537121)>

[117]:
```
(ggplot(boston, aes(x = 'medv', y = 'lstat')) + geom_point(mapping = aes(color␣
↪= "factor(PriceyHome)")) +
 labs(x = "Median Value of Homes (Thousands USD)", y = "% Lower Status of the␣
↪Population", title = "Pricey Home vs Non-Pricey Home"))
```



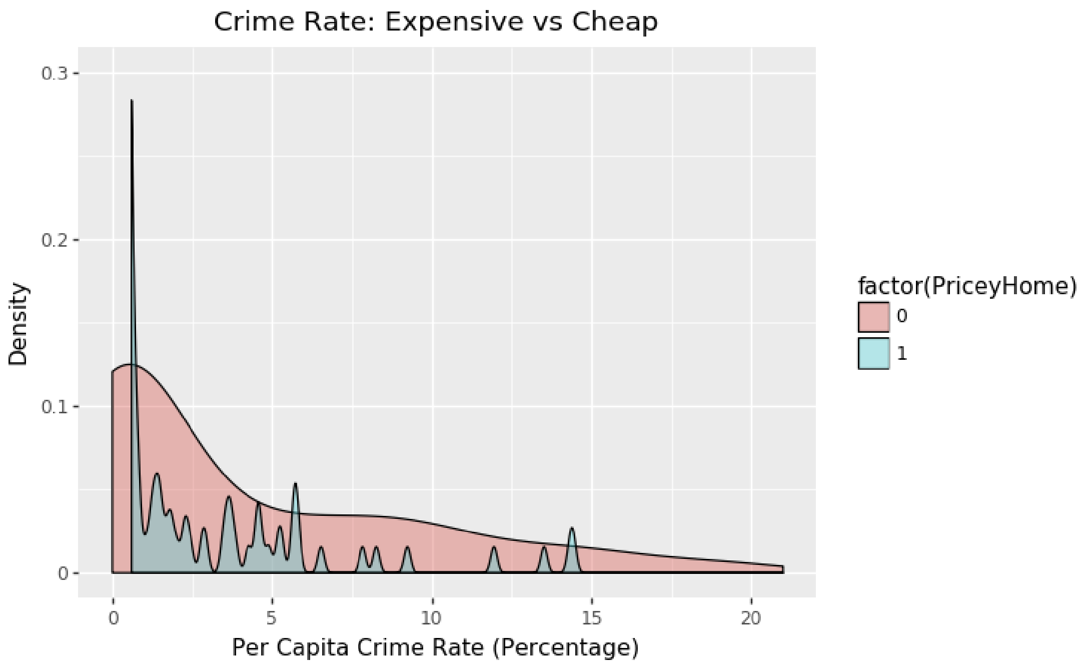
[117]: <ggplot: (7556686605)>

[118]:
```
(ggplot(boston, aes(x = 'medv', y = 'rm')) + geom_point(mapping = aes(color =␣
↪'factor(PriceyHome)')) +
 labs(x = "Median Value of Homes (Thousands USD)", y = "(Avg.) Number of Rooms␣
↪per Dwelling", title = "Pricey Home vs Non-Pricey Home"))
```



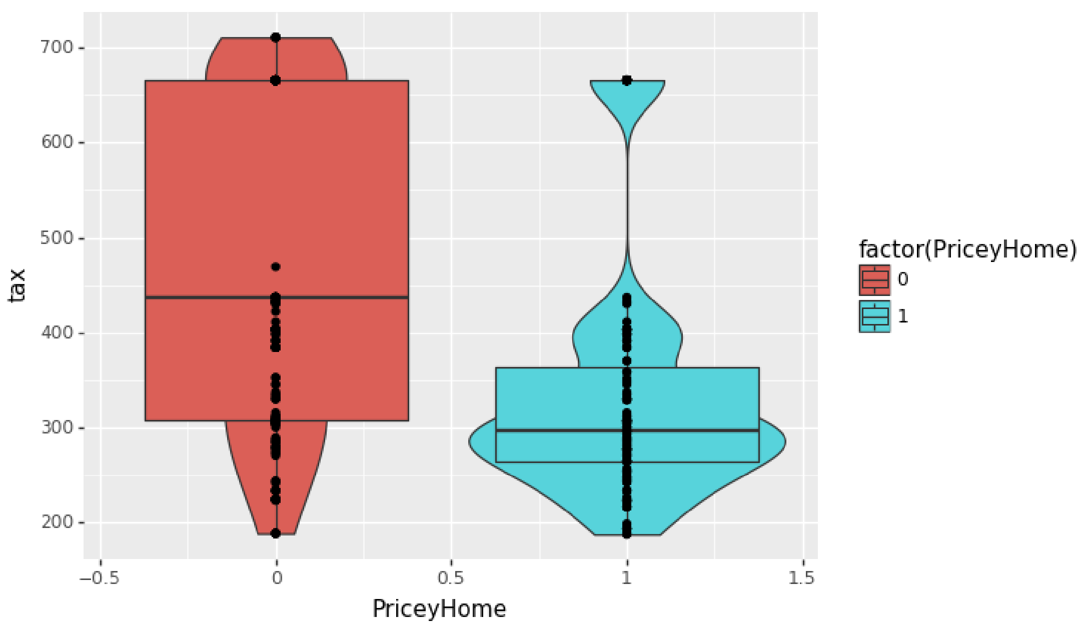
[118]: <ggplot: (7555537125)>

[119]:
```
(ggplot(boston, aes(x = 'crim', fill = 'factor(PriceyHome)')) +
geom_density(alpha = .40) +
labs(x = "Per Capita Crime Rate (Percentage)",
         y = "Density",
         title = "Crime Rate: Expensive vs Cheap") +
scale_x_continuous(limits = (0,21)) +
scale_y_continuous(limits = (0,.3)))
```



[119]: <ggplot: (7555309573)>

[120]:
```
(ggplot(boston, aes(x = "PriceyHome", y = "tax")) +
geom_violin(aes(fill = "factor(PriceyHome)")) +
geom_boxplot(aes(fill = "factor(PriceyHome)")) +
geom_point())
```



[120]: <ggplot: (7555553445)>

3.3 Part II: Question 1 (Logistic Regression and Decision Tree Models)

3.3.1 Question 1: How might we go about determining whether or not a Boston home is expensive or not with the data we have available?

3.3.2 Logistic Regression:

[125]:
```
# Select Variables
# We must exclude the medv variable... because our classification 0 or 1 (what␣
↪we are predicting)
#... is derived from the medv value
pred_vars = boston.columns[0:12]


X = boston[pred_vars]
y = boston['PriceyHome']
```

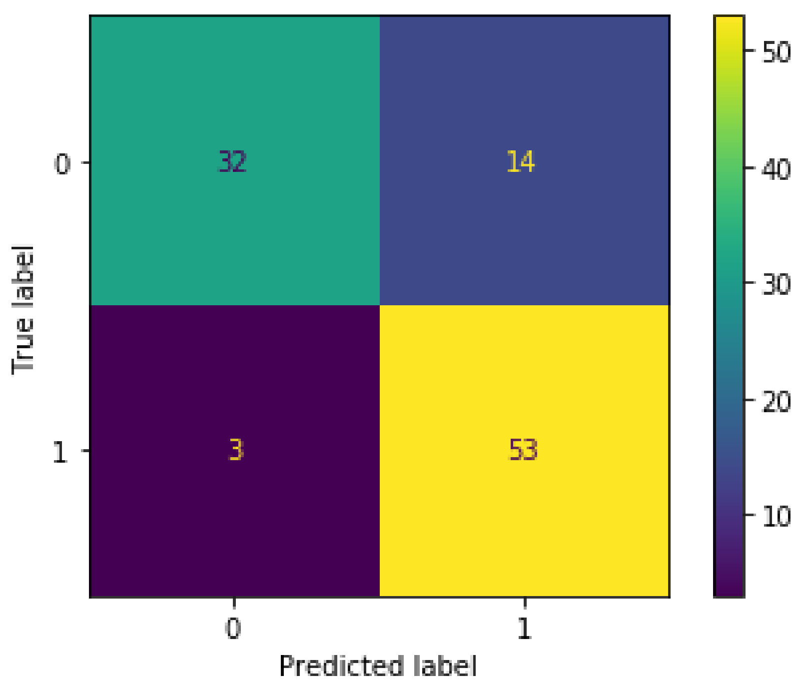
[133]:
```
# LOGISTIC REGRESSION
# K-FOLD CROSS VALIDATION


# create k-fold object
kf = KFold(n_splits = 5)

lr = LogisticRegression() #create model

acc = [] #create empty list to store accuracy for each fold
```

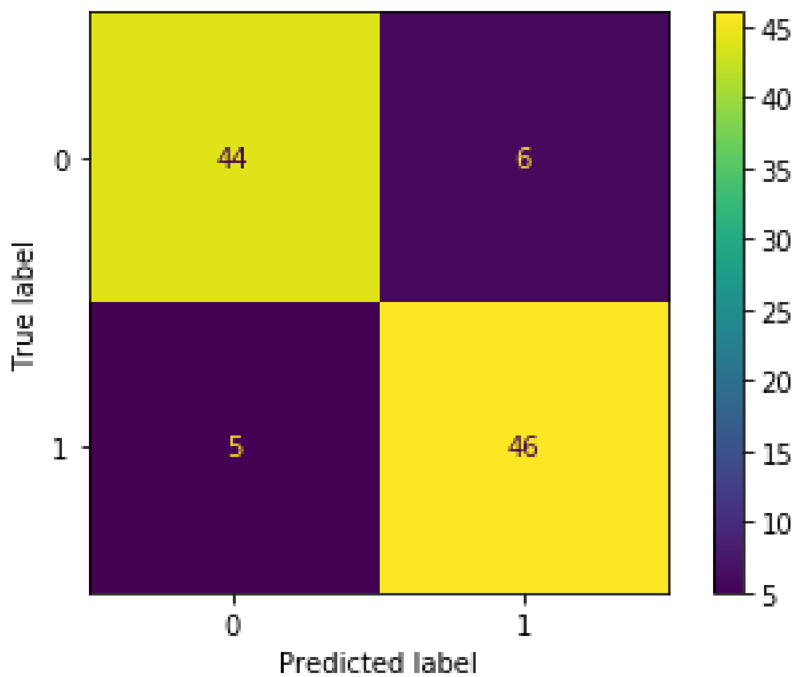
[134]:
```
#For loop to iterate through each fold and train a model, then add the accuracy␣
↪to acc.

for train_indices, test_indices in kf.split(X):
    # Get your train/test for this fold
    X_train = X.iloc[train_indices]
    X_test  = X.iloc[test_indices]
    y_train = y.iloc[train_indices]
    y_test  = y.iloc[test_indices]


    # Standardizing the training and test sets
    zscore = StandardScaler()
    zscore.fit(X_train) # ONLY FIT ON TRAINING
    Xz_train = zscore.transform(X_train)
```

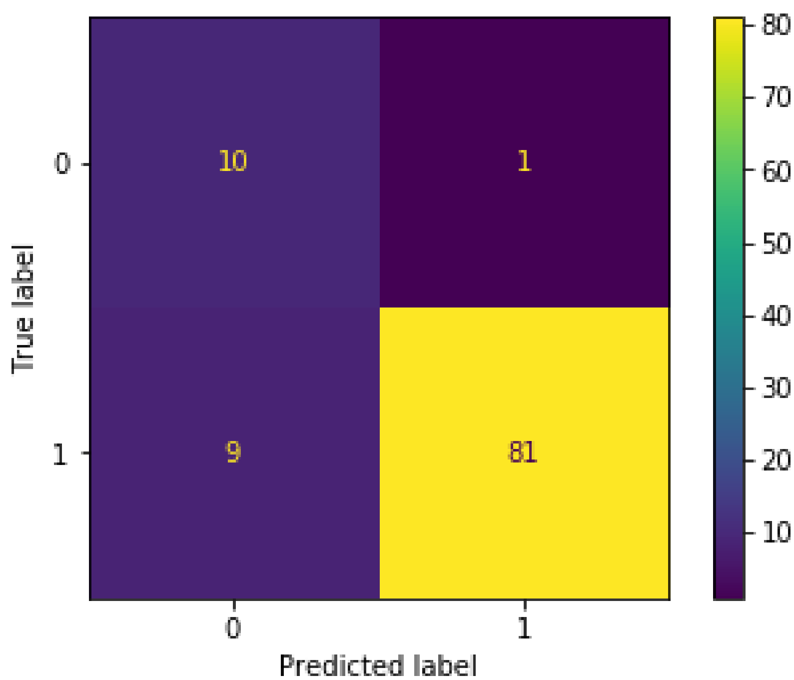
```
    Xz_test = zscore.transform(X_test) # Transform both though

    # model...fit to standardized training set otherwise whats the point of␣
↪even standardizing
    model = lr.fit(Xz_train, y_train)

    # Predict into standardized test set, get accuracy score, and store␣
↪accuracy in empty list
    acc.append(accuracy_score(y_test, model.predict(Xz_test)))

    # Plot Confusion Matrix for Each Model
    plot_confusion_matrix(lr, Xz_test, y_test)

#print overall acc
print(acc)
np.mean(acc)
```

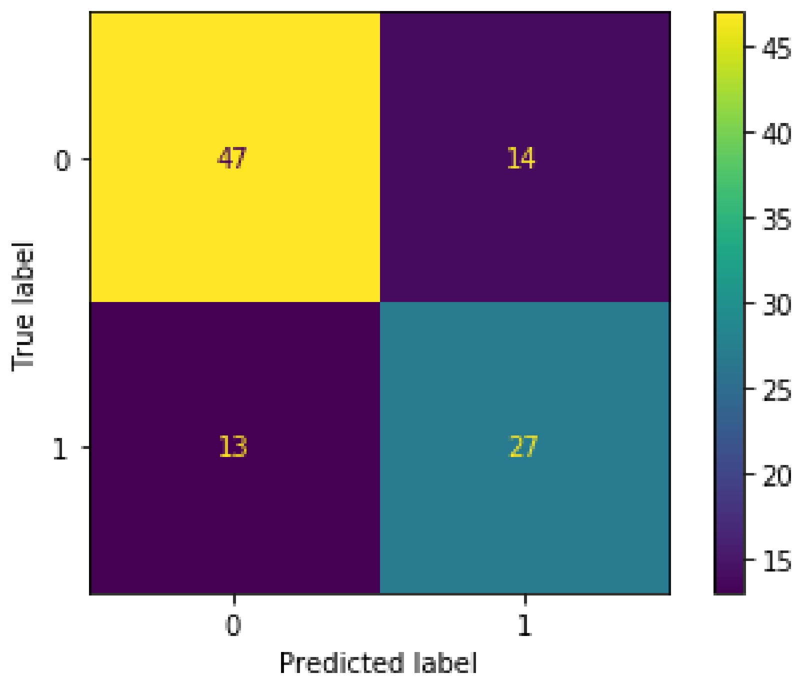
```
[0.8333333333333334, 0.8910891089108911, 0.900990099009901, 0.7326732673267327,
0.7425742574257426]
```

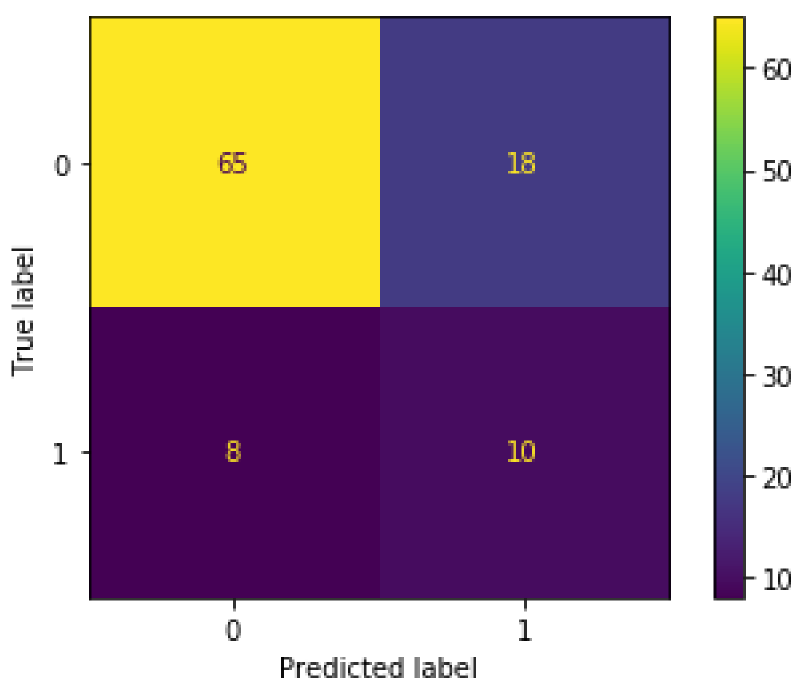
[134]: 0.8201320132013201




44
6
5
46
0
1
True label
Predicted label
45
40
35
30
25
20
15
10
5


10
1
9
81
0
1
True label
Predicted label
80
70
60
50
40
30
20
10





[135]:
```
# This should be predicting the first K-fold model we generated in our for-loop
Xz_Test_Preds = model.predict(Xz_test)
```

```
print(Xz_Test_Preds)

# accuracy_score(y_test, model.predict(Xz_test))...same as the last K-fold␣
↪model accuracy score in our list of scores
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 1 0 0
 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0]
```

[136]:
```
# Probability Scores
Ypred_prob = model.predict_proba(Xz_test)
# Ypred_prob[0:10]
# Ypred_prob.shape = (101, 2)... the first column is p(obs = 0) the second␣
↪column is p(obs = 1)
```

[137]:
```
# Create DataFrame of probability scores and combine them with the different␣
↪threshold predictions

# Probability Scores
Ypred_prob1 = pd.DataFrame(Ypred_prob[:, 1])
Ypred_prob1.columns = ['ProbabilityScore']
Ypred_prob1.round(3)

# Different Threshold value
thresh = 0.4

Ypred_prob1_thresh = pd.DataFrame(Ypred_prob1 > thresh) * 1
Ypred_prob1_thresh.columns = ['.4 Cut-off']

# Combine prediction scores with threshold value cut-offs
pd.concat([Ypred_prob1, Ypred_prob1_thresh], axis=1)
```

[137]:
```
     ProbabilityScore  .4 Cut-off
0            0.000001           0
1            0.000538           0
2            0.305402           0
3            0.039679           0
4            0.329481           0
..                …           …
96           0.445563           1
97           0.280353           0
98           0.737645           1
99           0.611700           1
100          0.269932           0

[101 rows x 2 columns]
```

[139]:
```python
# Area Under Curve for first K-fold model.

# calculate scores using actual and standardized predictions
auc = roc_auc_score(y_test, Xz_Test_Preds)

# summarize scores...rounding to 3 decimal places
print('Logistic Regression: ROC AUC for last K-fold Model = %.3f' % (auc))
```

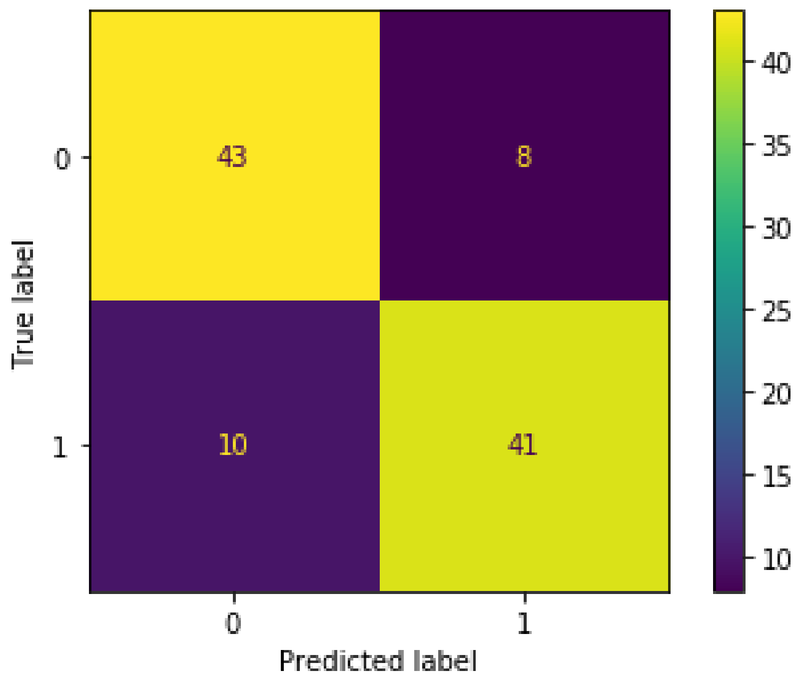
```
Logistic Regression: ROC AUC for last K-fold Model = 0.669
```

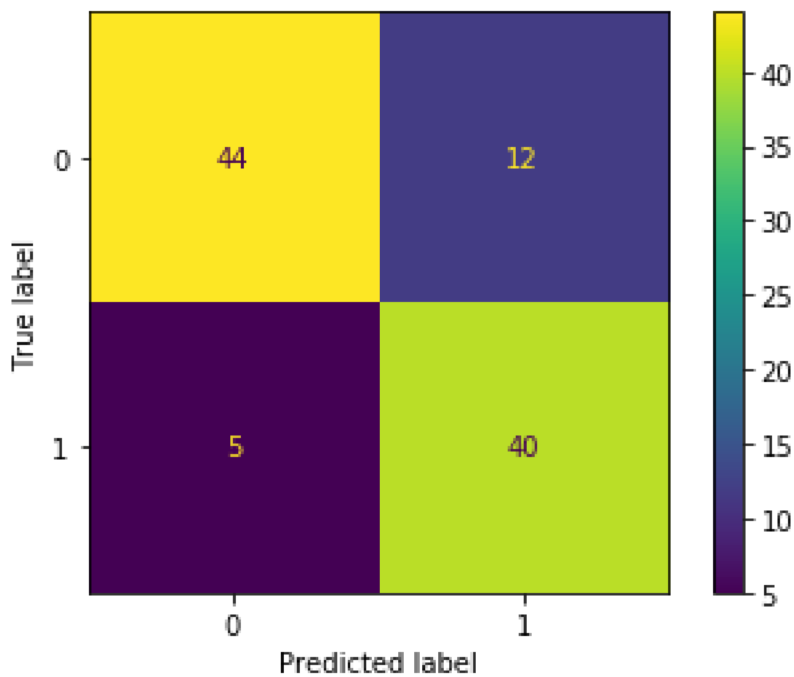
3.3.3 Decision Tree Model:

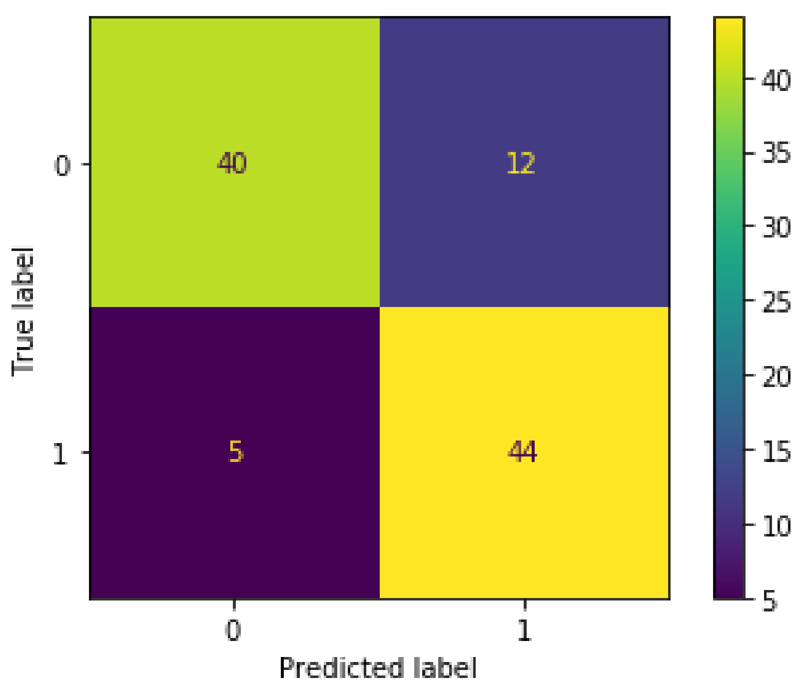
[144]:
```python
# K-Fold Decision Tree Model
# Select Variables again... just for coherence
pred_vars = boston.columns[0:12]

X = boston[pred_vars]
y = boston['PriceyHome']

# Same parameters as the first two models
kf = KFold(5, shuffle = True)

# Empty Lists
acc = []
depth = []

# Using down-sampled X and Y variables
for train, test in kf.split(X):
    X_train = X.iloc[train]
    X_test = X.iloc[test]
    y_train = y.iloc[train]
    y_test = y.iloc[test]

    # Standardize Continuous Variables
    zscore = StandardScaler()
    zscore.fit(X_train)

    Xz_train = zscore.transform(X_train)

    Xz_test = zscore.transform(X_test)

    tree = DecisionTreeClassifier()
    tree.fit(Xz_train,y_train)
    tree_model = tree.fit(Xz_train,y_train)

    acc.append(tree.score(Xz_test,y_test))
    depth.append(tree.get_depth())
```

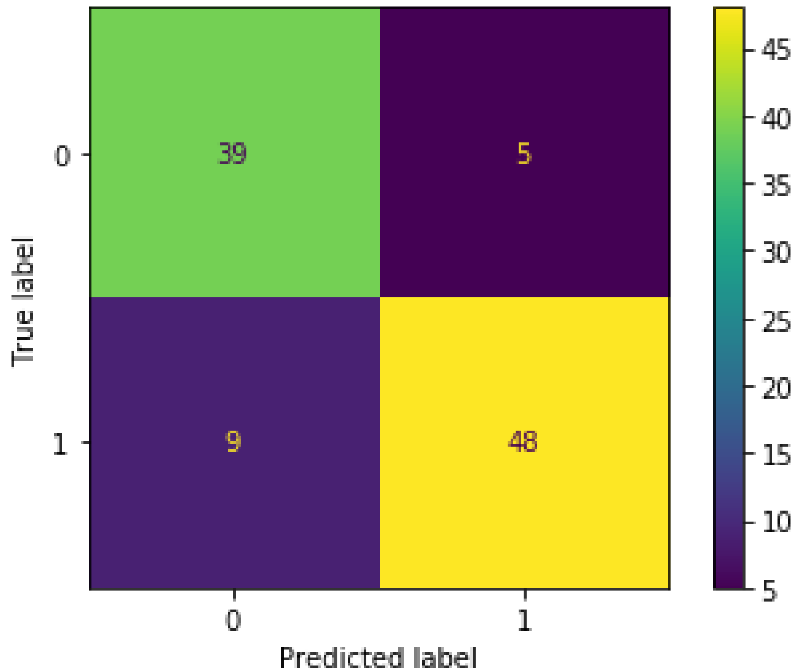
```
    plot_confusion_matrix(tree,Xz_test,y_test)

print(acc)
print(np.mean(acc))
print(depth)
```

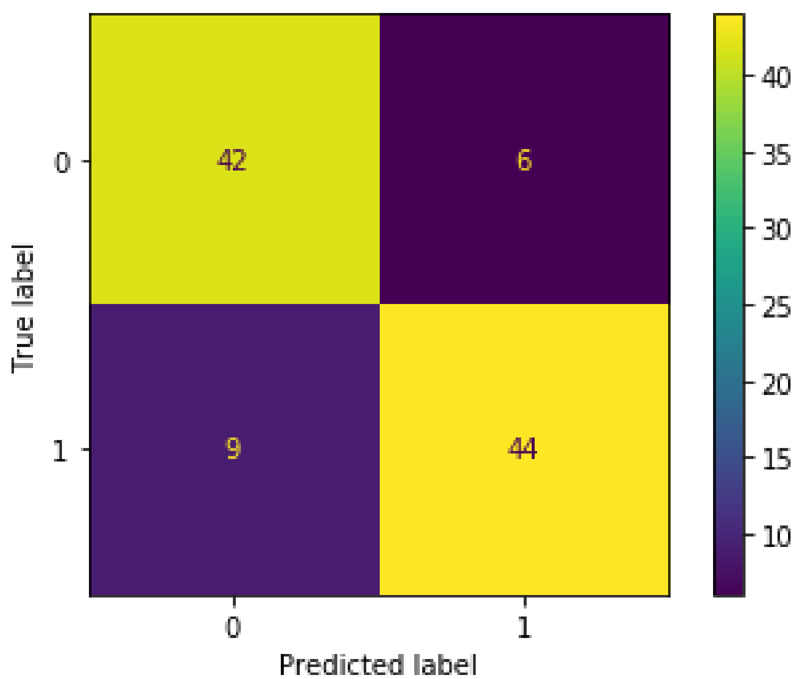
```
[0.8235294117647058, 0.8316831683168316, 0.8316831683168316, 0.8613861386138614,
0.8514851485148515]
0.8399534071054164
[11, 10, 8, 9, 9]
```




44
12
5
40
0
1
0
1
True label
Predicted label
40
35
30
25
20
15
10
5


40
12
5
44
0
1
0
1
True label
Predicted label
40
35
30
25
20
15
10
5





[145]:
```
# Generate Predictions for Tree... default cut-off = .5
Xz_Test_Preds_Tree = tree_model.predict(Xz_test)
```

```
print(Xz_Test_Preds)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 1 0 0
 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0]
```

[146]:
```
# Use Tree predictions to generate AUC

# Calculate scores using actual and standardized predictions
auc_tree = roc_auc_score(y_test, Xz_Test_Preds_Tree)

# Summarize scores...rounding to 3 decimal places
print('Logistic Regression: ROC AUC for last K-fold Model = %.3f' % (auc_tree))
```

```
Logistic Regression: ROC AUC for last K-fold Model = 0.853
```

3.3.4 Conclusion: Logistic Regression and Decision Tree Model

Logistic Regresion:

Overall I am somewhat pleased with the results from the Logit model. The lowest accuracy of all five K-fold models was 73%, and the average across all five was 82%. Further, the Area Under the ROC for the last K-fold model came out to a decent ~.7, meaning that it is 20% better than chance (rule of thumb: AUC of .7 is acceptable). Of course this was using all 13 predictor variables, but luckily sklearn penalizes variables, so we won't need to run ridge or lasso as shrinkage methods. In any future revisions of the logistic model, we could narrow down which homes we chose to predict. For instance we could try to generate the model on the top 5% or 10% of home values, to really guage what makes a super PriceyHome.

Decision Tree Model:

I am also somewhat pleased with th Decision Tree Model. The lowest accuracy of all five K-fold decision tree models was ~79%, and the average across all five was ~83.2% which is pretty good. Further, the Area Under the ROC for the last K-fold model was .853, meaning that this model was ~35.3% better than chance. Again, I elected to use all 12 potential (continuous) predictor variables. I could use a shrinkage method in a later revision to narrow down the variables and make the model more parsimonious, and also narrow down which homes to predict. I could try to generate a model using only the top 5% or 10% of medv (home values) to see what predicts the most expensive homes.

3.4 Part III: Question 2 (Ridge and Lasso Models)

3.4.1 Question 2: In the city of Boston, is there a higher crime rate when the median values of homes are lower? How might we go about predicting crime rate?

3.4.2 Ridge Regression:

[152]:
```
from sklearn.linear_model import RidgeCV, LassoCV
boston.head()
```

[152]:

```
       crim    zn  indus    nox     rm   age     dis  rad  tax  ptratio  \
0  0.00632  18.0   2.31  0.538  6.575  65.2  4.0900    1  296     15.3
1  0.02731   0.0   7.07  0.469  6.421  78.9  4.9671    2  242     17.8
2  0.02729   0.0   7.07  0.469  7.185  61.1  4.9671    2  242     17.8
3  0.03237   0.0   2.18  0.458  6.998  45.8  6.0622    3  222     18.7
4  0.06905   0.0   2.18  0.458  7.147  54.2  6.0622    3  222     18.7

    black  lstat  medv  PriceyHome
0  396.90   4.98  24.0           1
1  396.90   9.14  21.6           1
2  392.83   4.03  34.7           1
3  394.63   2.94  33.4           1
4  396.90   5.33  36.2           1
```

[164]:

```
# RIDGE
# Tune smoothing parameter to get smaller MAE
# MAE = Sum(all absolute value errors)/number of errors
feat = boston.columns[1:12]
X = boston[feat]
y = boston["crim"]

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2)

z = StandardScaler()

X_train[feat] = z.fit_transform(X_train[feat])
X_test[feat] = z.transform(X_test[feat])

X_train.head()

rr_tune = RidgeCV(cv = 5).fit(X_train,y_train)

print("TRAIN MAE: ", mean_absolute_error(y_train, rr_tune.predict(X_train)))
print("TEST MAE: ", mean_absolute_error(y_test, rr_tune.predict(X_test)),'\n')

print("TRAIN R2: ", r2_score(y_train, rr_tune.predict(X_train)))
print("TEST R2: ", r2_score(y_test, rr_tune.predict(X_test)))


print("\n Alpha = " + str(rr_tune.alpha_))

if  mean_absolute_error(y_train, rr_tune.predict(X_train)) <␣
 ↪mean_absolute_error(y_test, rr_tune.predict(X_test)):
    print("Our Model is Overfit")
elif mean_absolute_error(y_train, rr_tune.predict(X_train)) >␣
 ↪mean_absolute_error(y_test, rr_tune.predict(X_test)):
        print("Our Model is Underfit")
```

```
TRAIN MAE:  2.923988878738928
TEST MAE:  2.322512797946774

TRAIN R2:  0.4384582782576133
TEST R2:  0.4039410999038191

 Alpha = 10.0
Our Model is Underfit
```

3.4.3 LASSO Model:

[165]:
```
# LASSO
feat = boston.columns[1:12]
X = boston[feat]
y = boston["crim"]

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2)

z = StandardScaler()

X_train[feat] = z.fit_transform(X_train[feat])
X_test[feat] = z.transform(X_test[feat])

X_train.head()

lsr_tune = LassoCV(cv = 5).fit(X_train,y_train)

###########################
print("TRAIN MAE: ", mean_absolute_error(y_train, lsr_tune.predict(X_train)))
print("TEST MAE: ", mean_absolute_error(y_test, lsr_tune.predict(X_test)),'\n')

print("TRAIN R2: ", r2_score(y_train, lsr_tune.predict(X_train)))
print("TEST R2: ", r2_score(y_test, lsr_tune.predict(X_test)))


print("\n Alpha = " + str(lsr_tune.alpha_))

if  mean_absolute_error(y_train, lsr_tune.predict(X_train)) <␣
 ↪mean_absolute_error(y_test, lsr_tune.predict(X_test)):
    print("Our Model is Overfit")
elif mean_absolute_error(y_train, lsr_tune.predict(X_train)) >␣
 ↪mean_absolute_error(y_test, lsr_tune.predict(X_test)):
        print("Our Model is Underfit")
```

```
TRAIN MAE:  2.883268714779122
TEST MAE:  2.1230343749597784

TRAIN R2:  0.4152944168019205
```

```
TEST R2:  0.6899058347438429

 Alpha = 0.051952642702813225
Our Model is Underfit
```

3.4.4 Coefficients For Ridge and Lasso:

[176]:
```python
# Ridge Coefficients: Standardized:


# Coefficients into dataframe:
ridge_coefficients = pd.DataFrame({"Coef":rr_tune.coef_,
              "Name": feat})
ridge_coefficients = ridge_coefficients.append({"Coef": rr_tune.intercept_,
                    "Name": "intercept"}, ignore_index = True)

ridge_coefficients
```

[176]:

	Coef	Name
0	0.764239	zn
1	-0.669253	indus
2	-0.578999	nox
3	0.110325	rm
4	-0.202880	age
5	-1.391240	dis
6	4.165649	rad
7	0.460953	tax
8	-0.066871	ptratio
9	-0.649449	black
10	2.298568	lstat
11	3.922211	intercept

Interpretation of coefficients: Increasing 'lstat' (X10) by one standard deviation (7.141062) increases the crime rate (crim), on average by 2.298568 (Beta10).

[179]:
```python
# LASSO Coefficients: Standardized

# Coefficients into dataframe:
lasso_coefficients = pd.DataFrame({"Coef":lsr_tune.coef_,
              "Name": feat})
lasso_coefficients = lasso_coefficients.append({"Coef": lsr_tune.intercept_,
                    "Name": "intercept"}, ignore_index = True)

lasso_coefficients
```

[179]:

	Coef	Name
0	0.741308	zn
1	-0.281694	indus

```
2   -0.701652        nox
3   -0.000000         rm
4    0.000000        age
5   -1.230479        dis
6    4.453919        rad
7   -0.000000        tax
8   -0.101011    ptratio
9   -1.090720      black
10   1.677738      lstat
11   3.754046  intercept
```

Interpretation of coefficients: Increasing 'lstat' (x10) by one standard deviation (7.141062) increases the crime rate (crim), on average by 1.677738 (beta10). We can also see that Lasso reduced age, rm and tax to 0 because there is greater severity on the penalty term compared to Ridge Regression. These variables were not important.

3.4.5 Conclusion: Ridge Regression and Lasso Model

Ridge:

The Ridge regression model to predict crime rate was underfit given the MAE's for the test and training sets. It also had a mediocre R2 (coefficient of determination) of .44 for the training set and .40 for the test set. Meaning that all 12 continuous predictor varibales used only accounted for ~40% of the variation in the crime rate. However, considering that there were only 12 variables and that there are many other factors one should take into account when trying to predict the crime rate (like demographics data), I would say that an R2 of .40 is somewhat decent.

Lasso:

The Lasso model to predict the crime rate was also underfit given the MAE's for the test and training sets. However, it did have a much better R2 in the test set (approx .70), but much worse in the training set (approx .42). The lasso model only used 8/12 variables, so in a sense I guess it achieved some level of parsimony, but still performed pretty poorly. Again, there are many more factors one should take into consideration when attempting to predict crime rate.

3.5 Part IV: Question 3 (K-Means and Gaussian Mixture Model)

3.5.1 Question 3: Does the number of rooms per house increase or decrease with an increase in the parent-teacher ratio in Boston? What factors determine the parent-teacher ratio?

3.5.2 K-Means

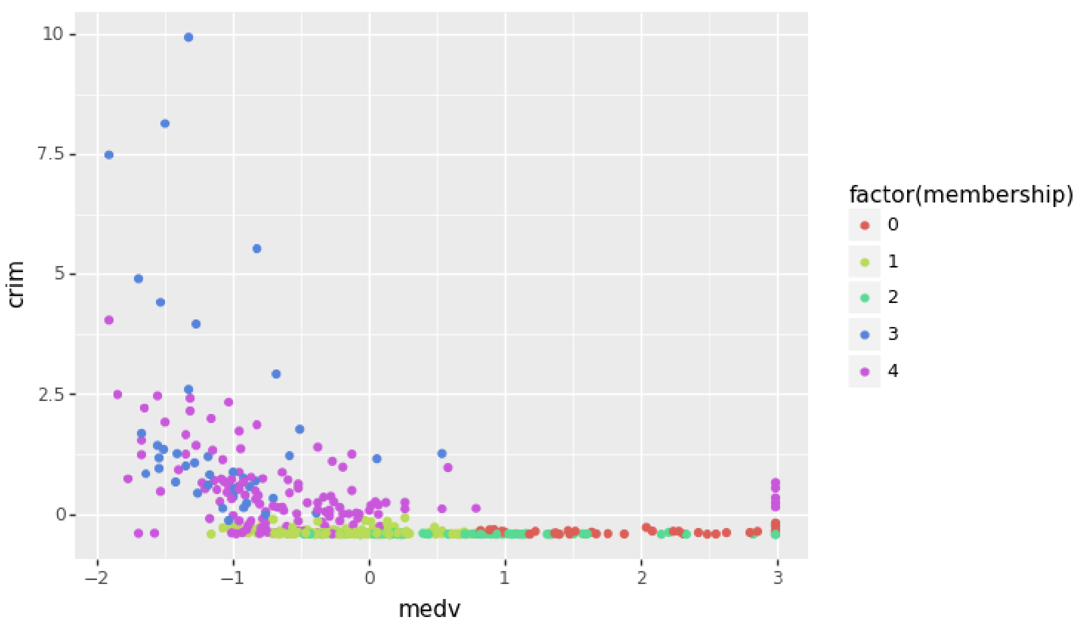
[229]:
```
# Build Model

features = boston.columns[0:13]
X = boston[features]



z = StandardScaler()
```

```
X[features] = z.fit_transform(X)
```

[230]:
```
print(features)
```

```
Index(['crim', 'zn', 'indus', 'nox', 'rm', 'age', 'dis', 'rad', 'tax',
       'ptratio', 'black', 'lstat', 'medv'],
      dtype='object')
```

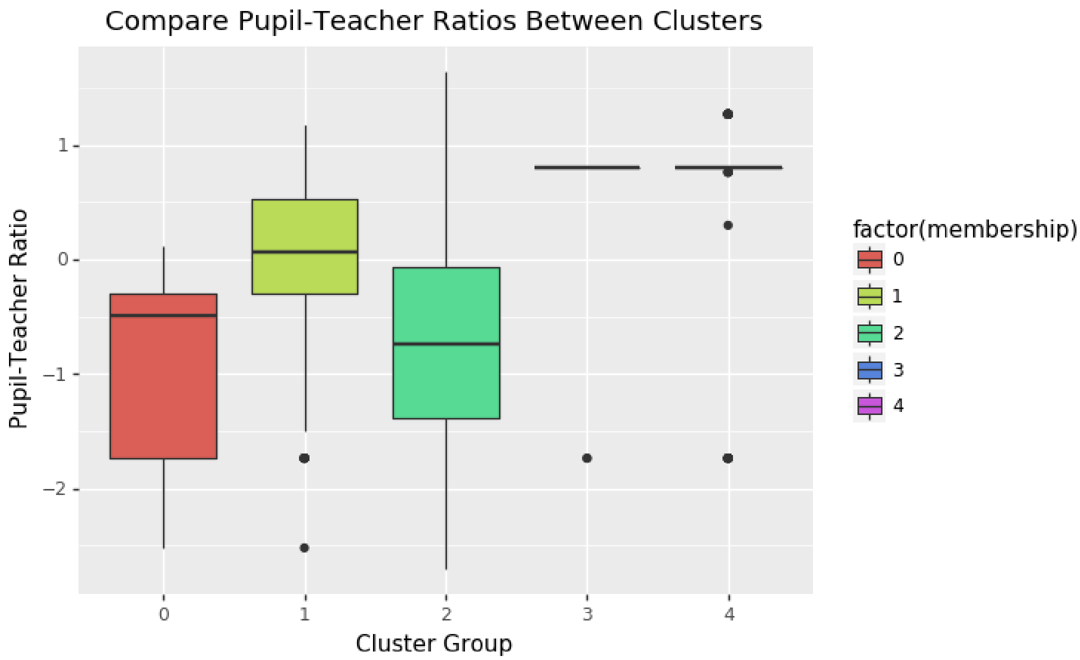
[234]:
```
# Choose K and Evaluate

km = KMeans(n_clusters = 5)
km.fit(X)

membership = km.predict(X)

X["cluster"] = membership

(ggplot(X, aes("medv", "crim", color = "factor(membership)")) + geom_point())

# We get some moderately distinct looking clusters... which is good
```



[234]: <ggplot: (7569825881)>

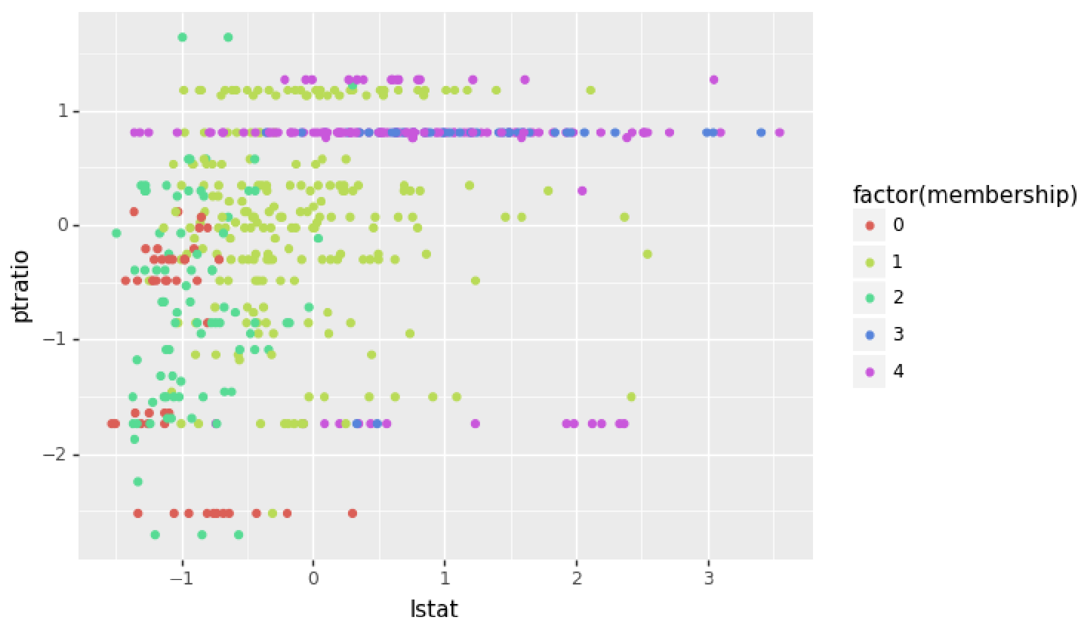
[235]:
```
(ggplot(X, aes(x = "factor(membership)", y = "ptratio", fill =␣
↪"factor(membership)")) + geom_boxplot() +
```

```
labs(x = "Cluster Group", y = "Pupil-Teacher Ratio", title = "Compare␣
↪Pupil-Teacher Ratios Between Clusters"))
```



[235]:
```
<ggplot: (7570306397)>
```

[236]:
```
(ggplot(X, aes("lstat", "ptratio", color = "factor(membership)")) +␣
↪geom_point())
```



[236]: <ggplot: (7570306525)>

[237]:
```
# Evaluation: Cohesion and Separation
silhouette_score(X, membership)
```

[237]: 0.3345478991201004

3.5.3 K-Means Model Conclusion:

After experimenting a bit with K… I decided to go with K = 5 for the final model. I wanted to keep the number of clusters small (going for parsimony), while choosing the highest silhouette score. Overall the model was okay at best, based off of the silhouette score which is based off of cohesion and separation for unserpervised learning. I was hoping to see a score nearer to 1 considering how distinct groups (.7 would have been nice). But oh well :/

Interestingly, the clusters were just what we might've expected to see. The groups (clusters red and green) with higher property values had lower crime rates and smaller pupil-teacher ratios. This is because you would expect lower rates of overflowing classrooms in wealthier areas/schools with more funding. Also, the groups with smaller pupil-teacher ratios (red and green) have less lower status individuals in them (lstat).

K = 6 yielded a score of: .2545

`K = 5 yielded a score of: .3345`

K = 4 yielded a score of: .3075

K = 3 yielded a score of: .2903

K = 2 yielded a score of: .2812

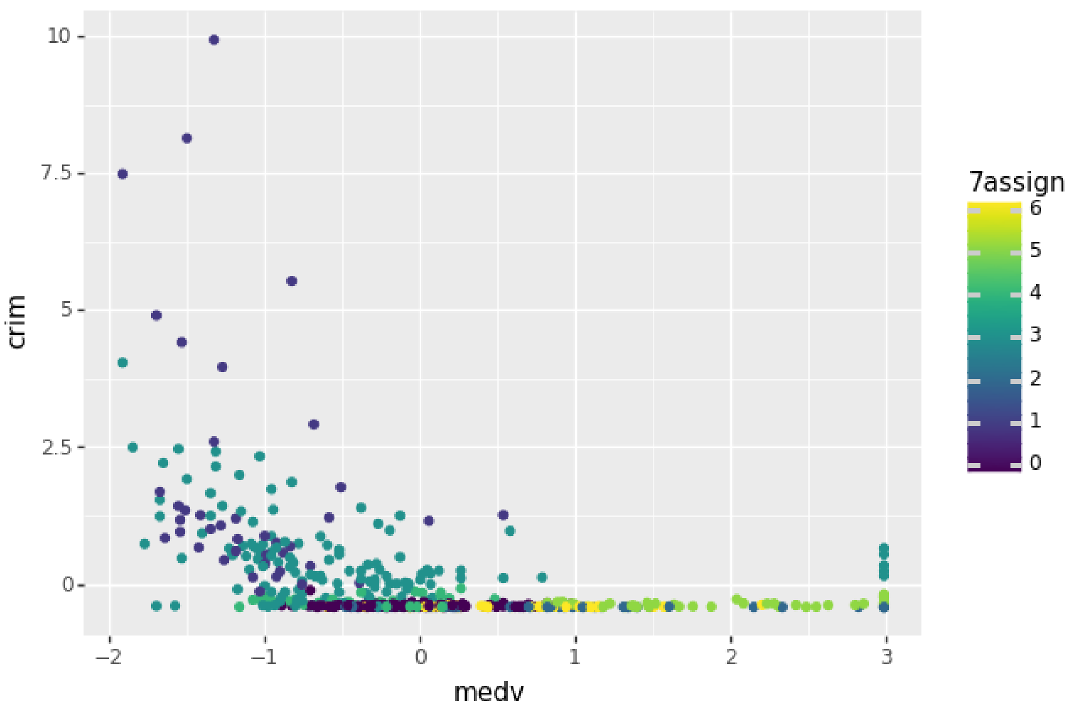
3.5.4 Gaussian Mixture Model

[239]:
```
# Build Model

Xdf = X

n_components = [2,3,4,5,6,7]

sils = []
for n in n_components:
    gmm = GaussianMixture(n_components = n)
    gmm.fit(X)
    colName = str(n) + "assign"
    clusters = gmm.predict(X)

    Xdf[colName] = clusters

```

```
    sils.append(silhouette_score(X, clusters))

print(sils)
```

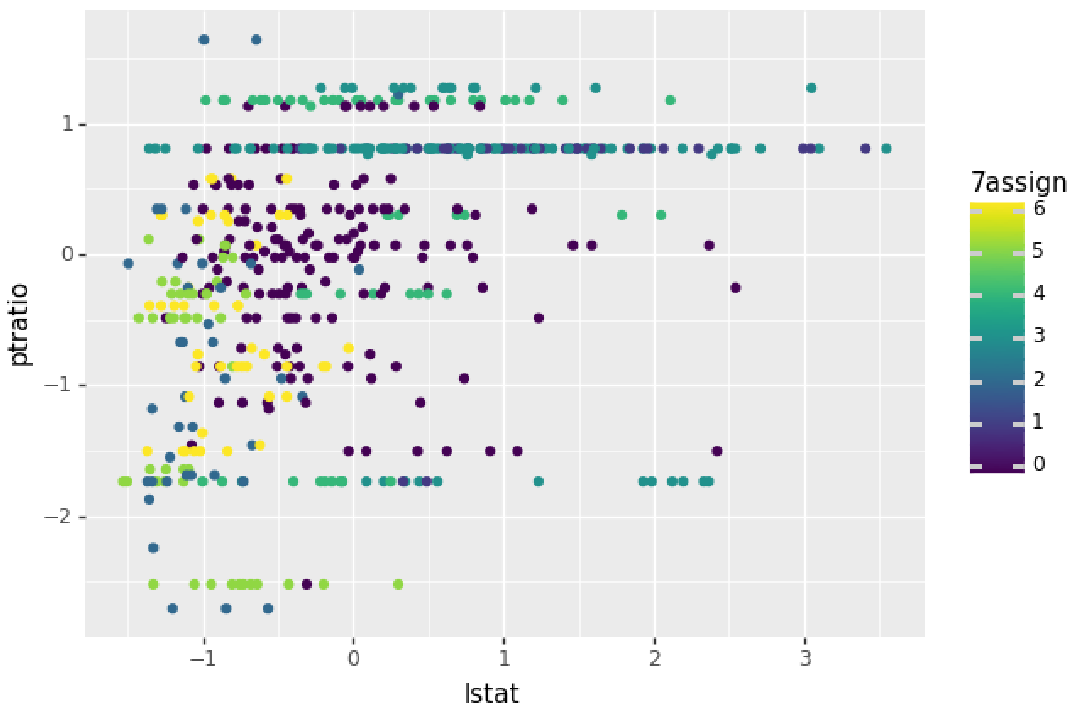
```
[0.4351476018326517, 0.3484041386144893, 0.38652448285606467,
0.43850060560309195, 0.4564911676410142, 0.5186720761445247]
```

[241]:
```
(ggplot(Xdf, aes(x = "medv", y = "crim", color = "7assign")) + geom_point())
```



[241]:
```
<ggplot: (7570629113)>
```

[242]:
```
(ggplot(Xdf, aes(x = "lstat", y = "ptratio", color = "7assign")) + geom_point())
```



[242]: <ggplot: (7565933361)>

3.5.5 GMM/EM Conclusion:

Based off of the silhouette scores for each number of components (2-7), the best was n_components = 7 which had a silhouette score of 0.5187. This model performed decently but, as alway, I would have hoped for better. Based off of the plots, it doesn't look like we have super good cohesion/separation for any group in most of the prime variables (ptratio, lstat, medv). Again I used the same rational as for all other models and decided to standardize all my variables because many of them are on different scales. The clusters are somewhat hard to differentiate visually, but it appears as though clusters were similar between K-Means and GMM models. Groups 6 and 5 from GMM had a lot of similarities with groups 0 and 2 (red and green) in regard to the several key indicators for pupil-teacher ratio.

3.6 PART V: Final Conclusion

After attempting to answer a few major questions/concerns about Boston housing, I can say with confidence that the model that performed the best overall was the K-Fold cross-validated Decision Tree model which addressed probably the most important question of them all: what determines the value of Boston Housing? Although I left sub-conclusions at the end of all analyses, I will go into more depth when developing a presentation. In addition to the specifics of the modeling process, I will add to the discussion: potential areas of improvement, inevitable limitations, and who would want to know this information (consumers of the analysis). The presentation will be much more concise and easy to digest.