DTSA 5509 Final

June 21, 2022

0.1 Section 1: Project Topic

Boston House Prices

My general goal for this project is to explore the different variables that go into determining median house values,and see if I can use different machine learning models to predict these values.

0.2 Section 2: Data Source

https://www.kaggle.com/datasets/fedesoriano/the-boston-houseprice-data

The original source of the dataset is StatLib - Carnegie Mellon University

This dataset is derived from The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978.

0.3 Section 3: Data Description

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import scipy as sp
import scipy.stats as stats
import statsmodels.formula.api as smf
import statsmodels.api as sm
from sklearn import metrics
from sklearn.model_selection import cross_val_score
import sklearn
from sklearn.metrics import mean_squared_error
from sklearn.svm import SVR
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestRegressor
```

```python
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.base import clone
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import warnings
import datetime
warnings.filterwarnings("ignore")
%matplotlib inline
```

[2]:
```python
data = pd.read_csv("boston.csv")
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    int64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```

Description of all variables:

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) [parts/10M]
6) RM: average number of rooms per dwelling

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[/10k]
11) PTRATIO: pupil-teacher ratio by town
12) B: The result of the equation B=1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
13) LSTAT: % lower status of the population
14) MEDV: Median value of owner-occupied homes in $1000s[k]

[3]:
```
print(data.shape)

print('Number of rows: ', data.shape[0])
print('Number of independent Variables: ', data.shape[1])
print('Target variable: MEDV')
```

```
(506, 14)
Number of rows:  506
Number of independent Variables:  14
Target variable: MEDV
```

[4]:
```
data.describe()
```

[4]:
```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284634
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.677082   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            LSTAT        MEDV
```

```
count  506.000000  506.000000
mean    12.653063   22.532806
std      7.141062    9.197104
min      1.730000    5.000000
25%      6.950000   17.025000
50%     11.360000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
```

[ ]:
```
## Section 4: Data Cleaning
```

[5]:
```
data.isnull().sum()
```

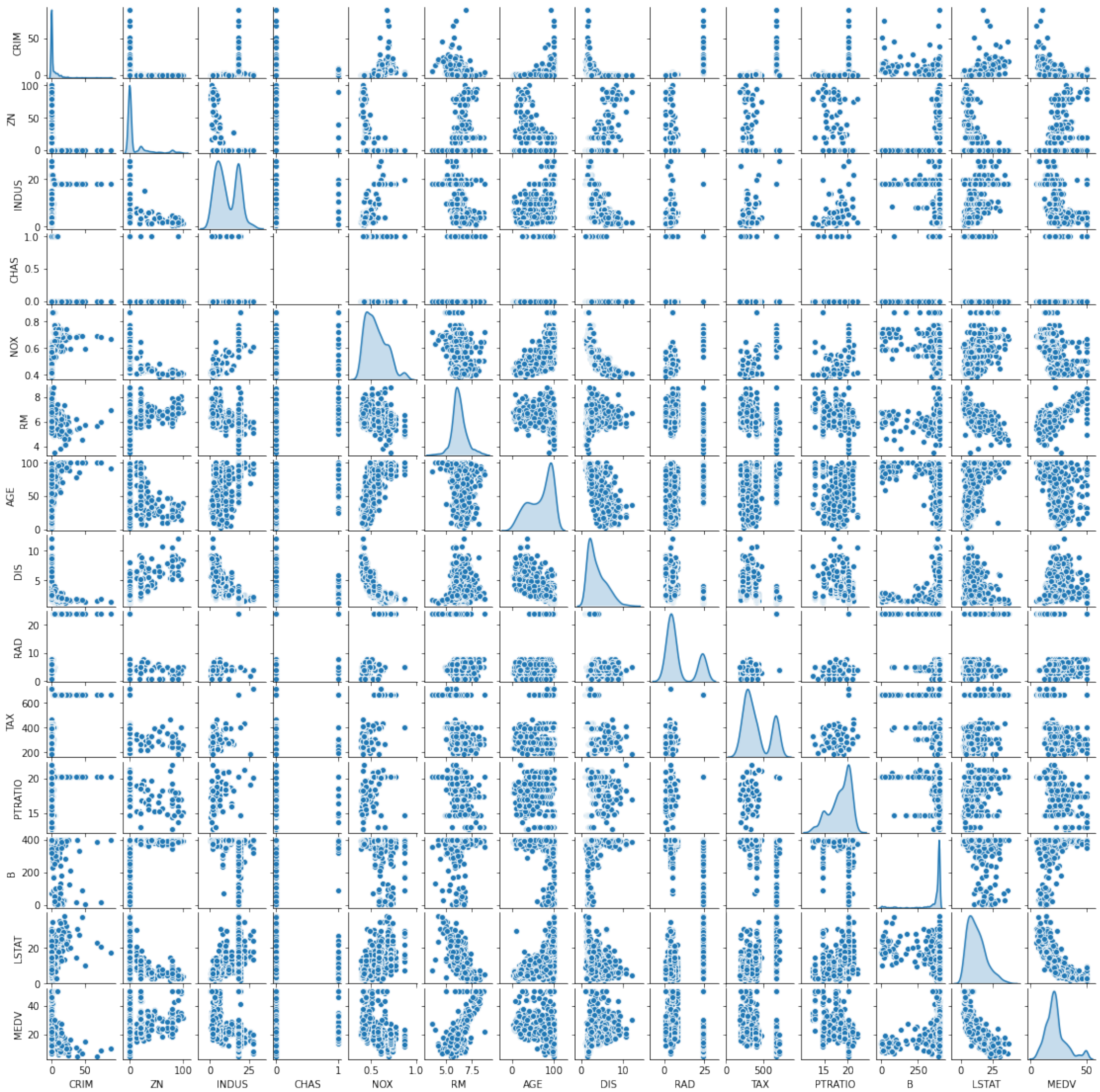
[5]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

Since there are no null values present, and I decided not to drop any of the columns from the dataset, I went ahead and proceeded straight away with my exploratory data analysis.

0.4 Section 5: Exploratory Data Analysis(EDA)

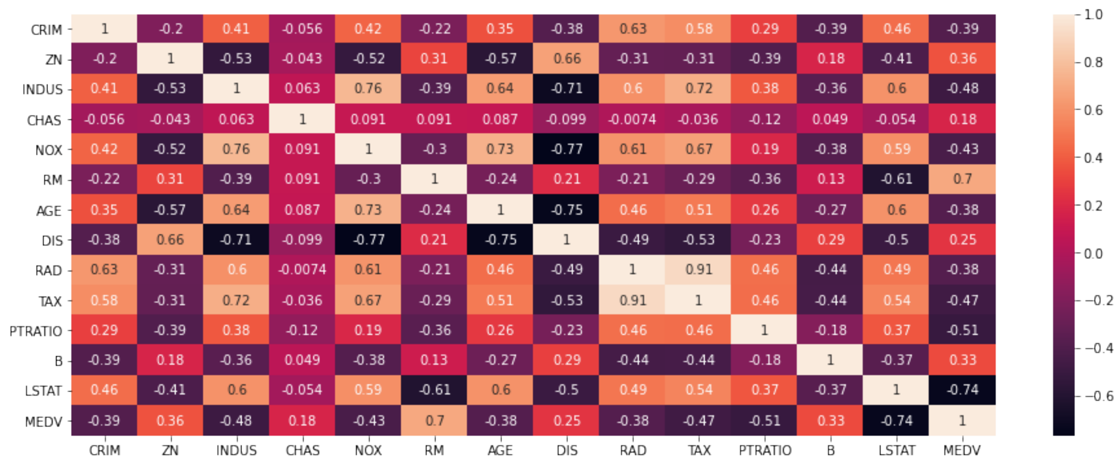
First, in order to see the relationships between all the variables, I decided to make a pair plot:

[6]:
```
x=sns.pairplot(data, diag_kind="kde")
x.fig.set_size_inches(15,15)
```



[7]:
```
plt.figure(figsize=(16,6))
sns.heatmap(data.corr(), annot = True)
```

[7]: <matplotlib.axes._subplots.AxesSubplot at 0x7f52f2d8b6d0>

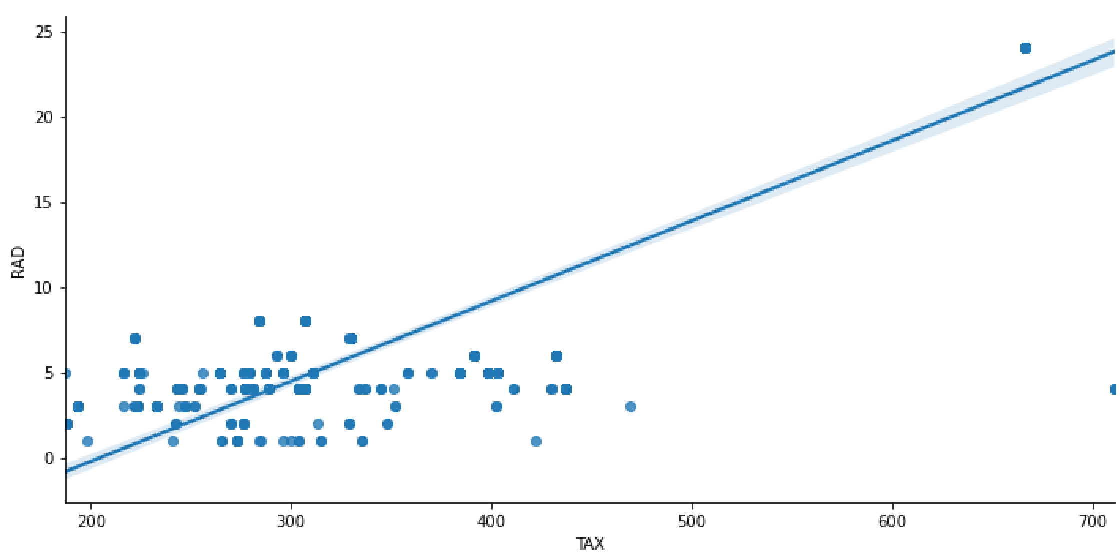


From the matrix above, we can see that there is a strong correlation between the "RAD" and "TAX" variables.

Based off of this, I wanted to explore the relationship between these two variables, and create different visualizations to see the relationship between other variables as well.

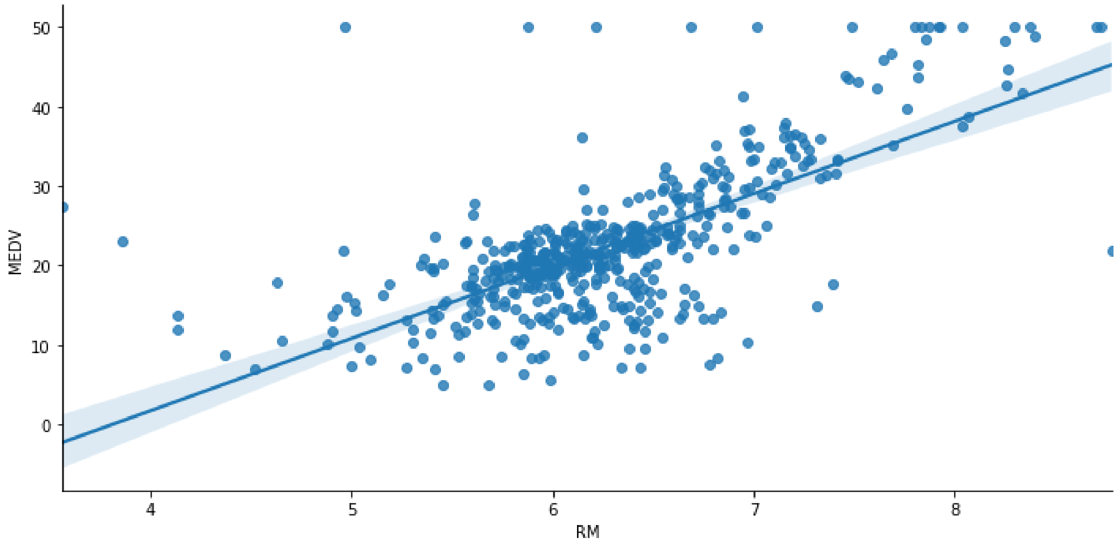
[8]:
```
sns.lmplot(x='TAX', y='RAD', data=data, aspect=2)
```

[8]: <seaborn.axisgrid.FacetGrid at 0x7f52ea3743d0>



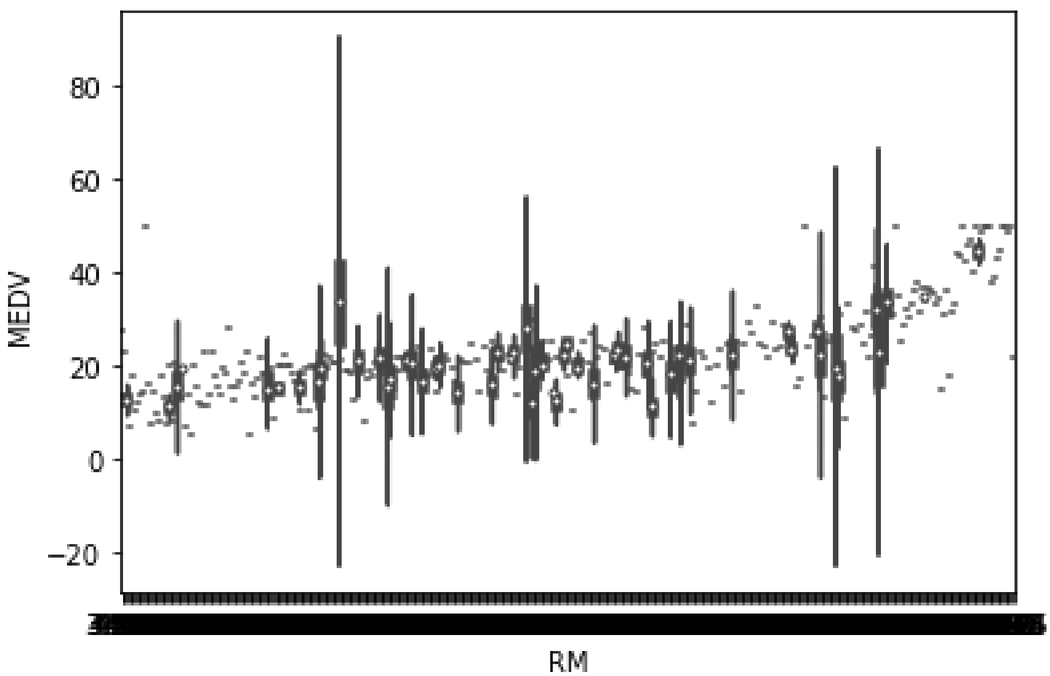
[9]:
```
sns.lmplot(x='RM', y='MEDV', data=data, aspect=2)
```

[9]: <seaborn.axisgrid.FacetGrid at 0x7f52ea8a4b50>



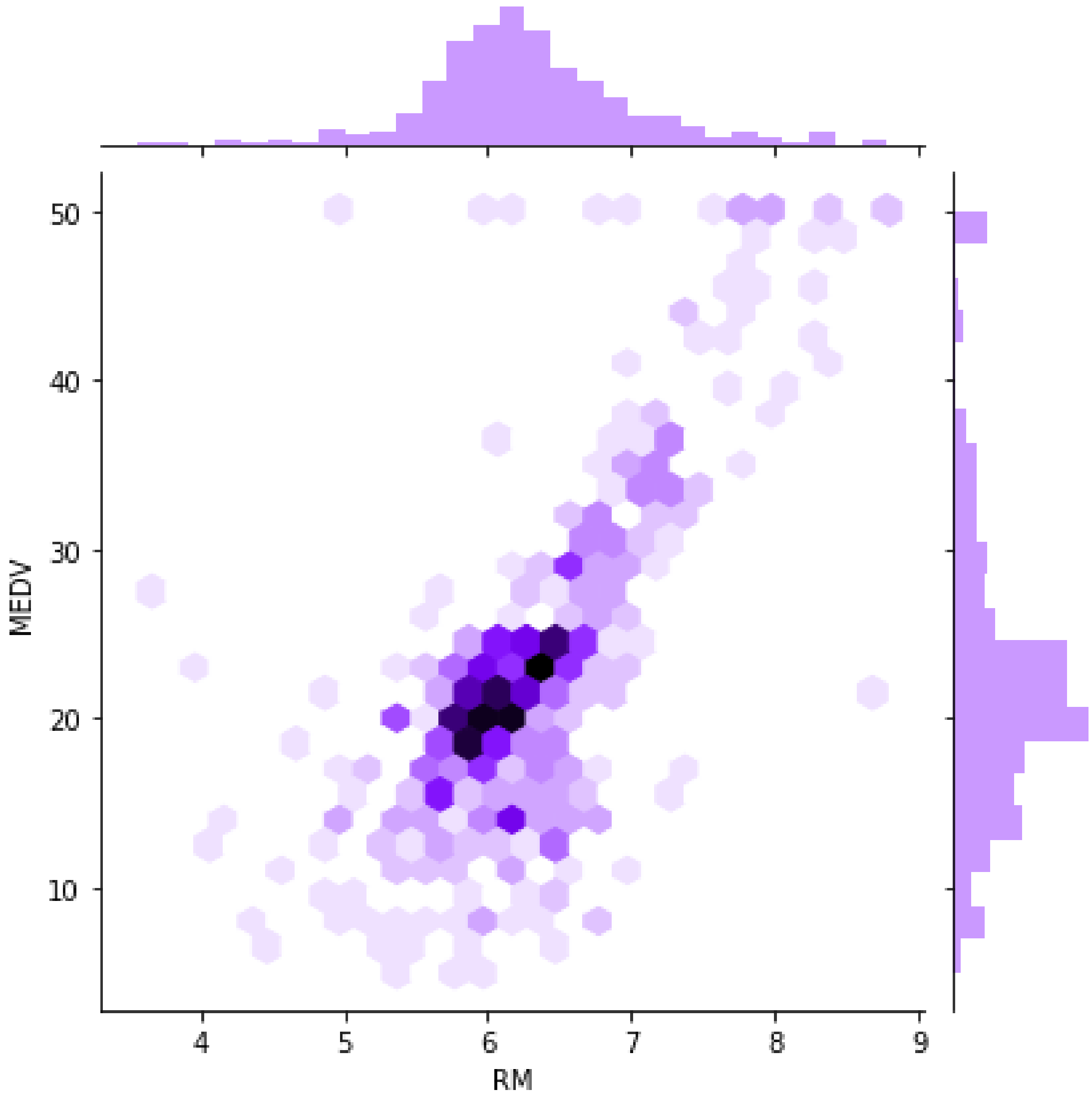

```
[10]: sns.violinplot(x='RM', y='MEDV', data=data, aspect=2)
      plt.figure(figsize=(16,6))
```

[10]: <Figure size 1152x432 with 0 Axes>



<Figure size 1152x432 with 0 Axes>

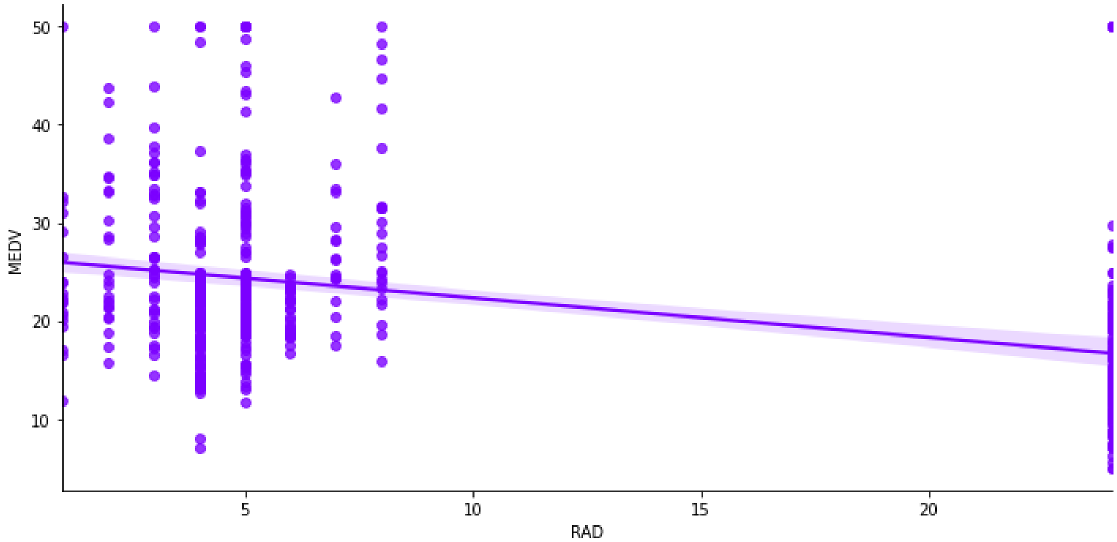
[11]:
```
sns.set_palette("gist_rainbow_r")
sns.jointplot(x="RM", y="MEDV", kind="hex",data=data )
plt.show()
```



I used several different visualizations to show the positive correlation between avg # of rooms per dwelling and median home value, a relationship which is more obvious to predict. I thought this would be the same case for the next relationship I explored. In this case, I was surprised to see the negative correlation between accessibility to radial highways and median house-value. I would have assumed that immediate access to highways would mean easier accesibility, and would increase the value of a home:

[12]:
```
sns.lmplot(x='RAD', y='MEDV', data=data, aspect=2)
```

[12]: <seaborn.axisgrid.FacetGrid at 0x7f52df80ab10>



[ ]:
```
## Section 6: Build a Model
```

Given that MEDV is the output variable, I am going to split the dataset into training and test data sets based off of this variable:

[13]:
```
X = data.drop('MEDV', axis=1)
y = data['MEDV']
```

[14]:
```
#Create test and train data
X_train, X_test, y_train, y_test = train_test_split(X, y,train_size=0.
↪8,random_state=42)
```

After splitting the data, I needed to decide which models I wanted to use.

The first model I decided to use is a Linear Regression Model. I chose this model, because I thought it would be the simplest model to start with. Before using that model, however, I wrote a small function that would calculate some important outputs:

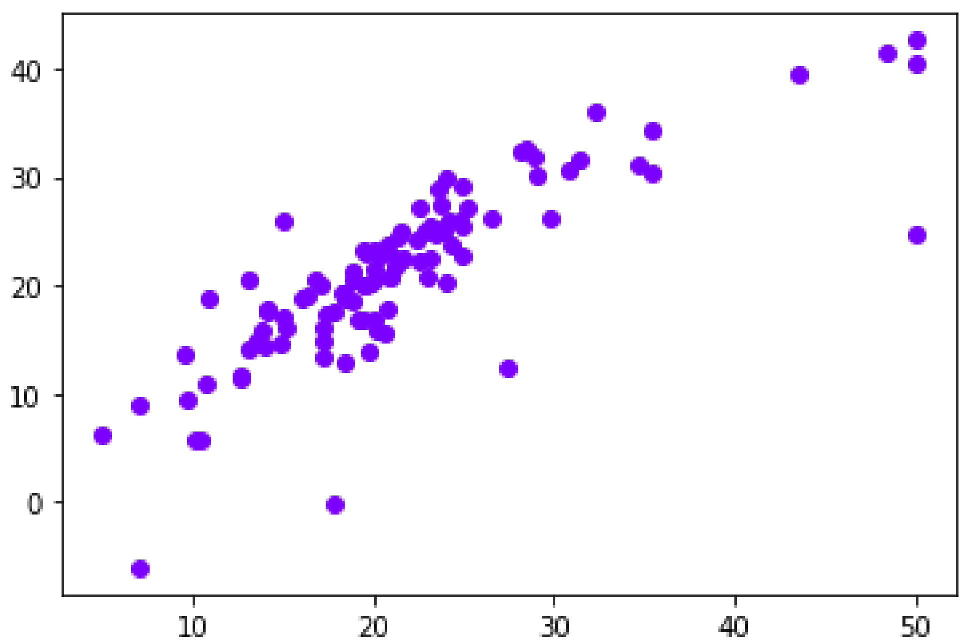
[44]:
```
def calculations(true, predicted):
    mae = metrics.mean_absolute_error(true, predicted)
    mse = metrics.mean_squared_error(true, predicted)
    rmse = np.sqrt(metrics.mean_squared_error(true, predicted))
    r2_sq = metrics.r2_score(true, predicted)
    print('MAE:', mae)
    print('MSE:', mse)
    print('RMSE:', rmse)
    print('R2 Square', r2_sq)
```

[45]:
```
lin = LinearRegression(normalize=True)
lin.fit(X_train,y_train)
test_pred = lin.predict(X_test)
train_pred = lin.predict(X_train)

print('Test set')
calculations(y_test, test_pred)
print('Train set')
calculations(y_train, train_pred)
```

```
Test set
MAE: 3.1890919658878474
MSE: 24.29111947497351
RMSE: 4.928602182665336
R2 Square 0.6687594935356321
Train set
MAE: 3.31477162678323
MSE: 21.641412753226312
RMSE: 4.6520331848801675
R2 Square 0.7508856358979673
```

[47]:
```
#Plotting prediction
pred = lin.predict(X_test)
plt.scatter(y_test, pred)
plt.show()
```



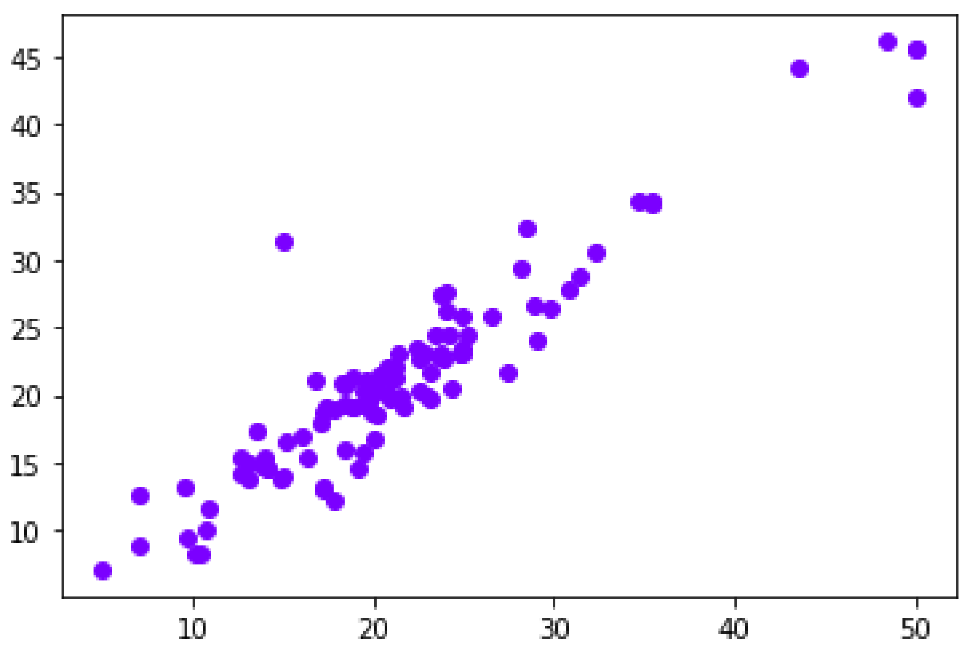
Originally, I was going to choose a DecisionTreeClassifier as my second model. However, I thought that this would work better on a binary variable, and it might take a long time to load. The second model I decided to use is a Random Forest Regressor:

[41]:
```
randfor = RandomForestRegressor()
randfor.fit(X_train,y_train)
test_pred = randfor.predict(X_test)
train_pred = randfor.predict(X_train)

print('Test set')
calculations(y_test, test_pred)
print('Train set')
calculations(y_train, train_pred)
```

```
Test set
MAE: 2.0455
MSE: 8.528578186274512
RMSE: 2.9203729532843083
R2 Square 0.8837019199237375
Train set
MAE: 0.8702549504950492
MSE: 1.8557954480198025
RMSE: 1.3622758340438264
R2 Square 0.9786379332898228
```

[46]:
```
#Plotting prediction
pred_1 = randfor.predict(X_test)
plt.scatter(y_test, pred_1)
plt.show()
```



The third model I chose is the Support Vector Algorithm:

[48]:
```
svm= SVR(kernel='rbf', C=1000000, epsilon=0.001)
svm.fit(X_train, y_train)

test_pred = svm.predict(X_test)
train_pred = svm.predict(X_train)


print('Testing Values:')
one=calculations(y_test, test_pred)
print('--------------------------------------------')
print('Training values')
two=calculations(y_train, train_pred)
```

```
Testing Values:
MAE: 2.406948541994415
MSE: 13.177320758418691
RMSE: 3.6300579552424077
R2 Square 0.8203103646022145
--------------------------------------------
Training values
MAE: 1.6575729516899593
MSE: 7.504002027415164
RMSE: 2.739343356977209
R2 Square 0.9136214111991737
```

[29]:
```
## Section 7: Discussion/Conclusion
```

1) All three models, although had high R2 values, were all examples of overfitting: testing MSE much higher than training data MSE
2) If I had to choose the best model, based off of R2 alone, I would choose the Random Forest Regressor
3) Had I done this project over, I definitely would have chosen to explore the CRIM- crime variable instead
4) Second, I would have chosen to split up the dataset into training and test data based off of a different variable, such as Crime, or even nitric-oxide concentration, just to see any change in results
5) Third, I would have chosen to focus my analysis more on the Charles River dummy variable, which would have allowed me to use different ML models
6) I was not fully happy with all of my choices, but I think I found some interesting insights