Visualizations and Random Forest

Prior to this task, you should have watched a video on random forest on Canvas.

Advantages of Random Forest:

- Random forest can solve both type of problems that is classification and regression and does a decent estimation at both fronts.
- Random forest can be used on both categorical and continuous variables.
- You do not have to scale features.
- Fairly robust to missing data and outliars.

Disadvantages of Random Forest

- It is complex, e.g., look at the tree at the end of this exercise! This makes it feel like a black box, and we have very little control over what the model does.
- It can take a long time to train.

In [1]:
```
# Here are some alternative ways to load packages in python as aliases
# This can be useful if you call them often
```

The Boston Housing Dataset consists of price of houses in various places in Boston. Alongside with price, the dataset also provide information such as Crime (CRIM), areas of non-retail business in the town (INDUS), the age of people who own the house (AGE), and there are many other attributes that available here.

In [2]:
```
import numpy as np
import sklearn as sk
import sklearn.datasets as skd
import sklearn.ensemble as ske
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
```

In [3]:
```
data = skd.load_boston()
df = pd.DataFrame(data.data, columns = data.feature_names)
df.head()
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LST
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.

In [4]:
```
df.shape
```

Out[4]: (506, 13)

In [5]:
```
print(data.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (a
ttribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000
sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0
otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by
town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carneg
ie Mellon University.

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.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers tha
t address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential
Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In
```

```
Proceedings on the Tenth International Conference of Machine Learning, 236-24
3, University of Massachusetts, Amherst. Morgan Kaufmann.
```

In [6]:
```
pd.isnull(df).sum()
```

Out[6]:
```
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
dtype: int64
```

We should check to see if there are any null values. There are several ways we've learned to do this.

In [7]:
```
df.describe()
```

Out[7]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12

We shoud check the data first to see if there are any weird anomalies.

What we should look for are:

- There are not any data points that immediately appear as anomalous
- No zeros in any of the measurement columns.

Another method to verify the quality of the data is make basic plots. Often it is easier to spot anomalies in a graph than in numbers.

In [ ]:

It is useful to know whether some pairs of attributes are correlated and how much. For many ML algorithms correlated features that are not independent should be treated with caution. Here is a good [blog (https://towardsdatascience.com/data-correlation-can-make-or-break-your-machine-learning-project-82ee11039cc9)](https://towardsdatascience.com/data-correlation-can-make-or-break-your-machine-learning-project-82ee11039cc9) on explaining why.

To prevent this, there are methods for deriving features that are as uncorrelated as possible (CA, ICA, autoencoder, dimensionality reduction, manifold learning, etc.), which we'll learn about in coming classes.

We can explore coreelation with Pandas pretty easily...

In [8]:
```
corr = df.corr(method ="pearson")
corr
```

Out[8]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.494588
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.291512
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996

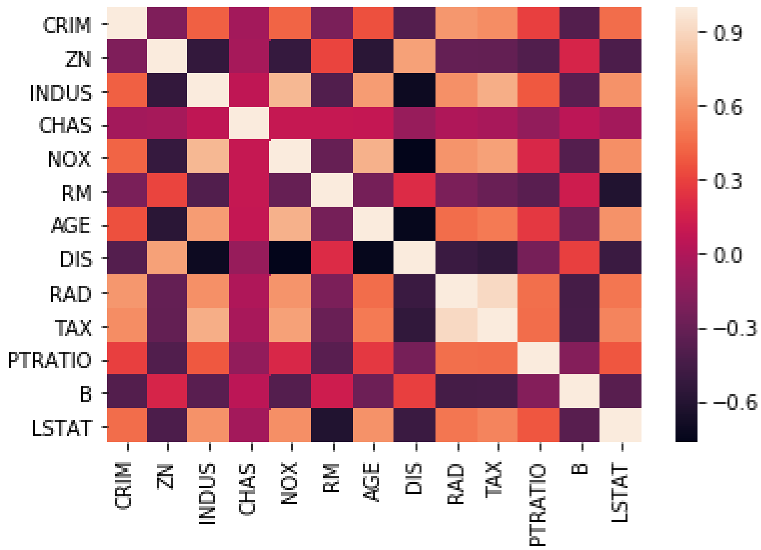
In [ ]:

Let's explore/review some visualization approaches

A good way to look at correlations quickly is a visualization called a heatmap. Let's take a look at correlations betewen features in our dataset.

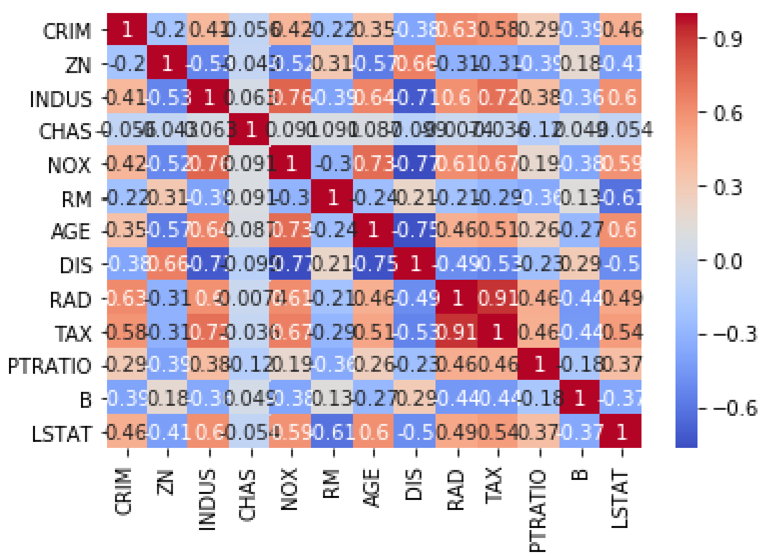
In [9]:
```
import seaborn as sns
sns.heatmap(corr)
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x177a1c17320>



You can also save the plots you make in these notebooks locally.

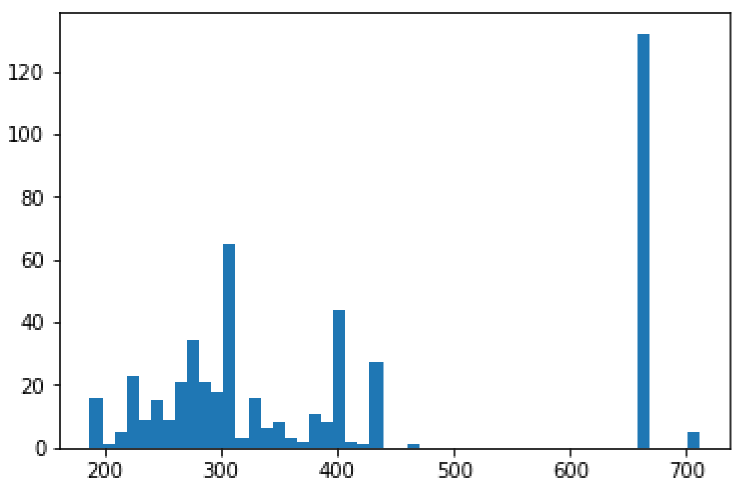
In [10]:
```
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.savefig('heatmap.png', tight_layout=True)
```



Let's take a look how we can explore the distributions of values within a specific feature. Specifically, let's look at the distribution of property tax in Boston. We can do this either in matplotlib or sns. There are so many tools available to you in Python!

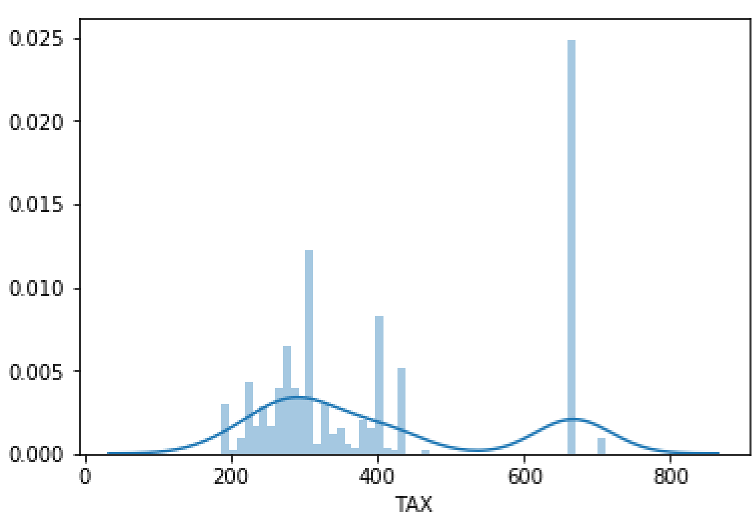
In [11]:
```
attr = df['TAX']
plt.hist(attr, bins=50)
```

Out[11]:
```
(array([ 16.,   1.,   5.,  23.,   9.,  15.,   9.,  21.,  34.,  21.,  18.,
         65.,   3.,  16.,   6.,   8.,   3.,   2.,  11.,   8.,  44.,   2.,
          1.,  27.,   0.,   0.,   1.,   0.,   0.,   0.,   0.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          0., 132.,   0.,   0.,   0.,   5.]),
 array([187.  , 197.48, 207.96, 218.44, 228.92, 239.4 , 249.88, 260.36,
        270.84, 281.32, 291.8 , 302.28, 312.76, 323.24, 333.72, 344.2 ,
        354.68, 365.16, 375.64, 386.12, 396.6 , 407.08, 417.56, 428.04,
        438.52, 449.  , 459.48, 469.96, 480.44, 490.92, 501.4 , 511.88,
        522.36, 532.84, 543.32, 553.8 , 564.28, 574.76, 585.24, 595.72,
        606.2 , 616.68, 627.16, 637.64, 648.12, 658.6 , 669.08, 679.56,
        690.04, 700.52, 711.  ]),
 <a list of 50 Patch objects>)
```



In [12]:
```
sns.distplot(attr, bins=50)
```

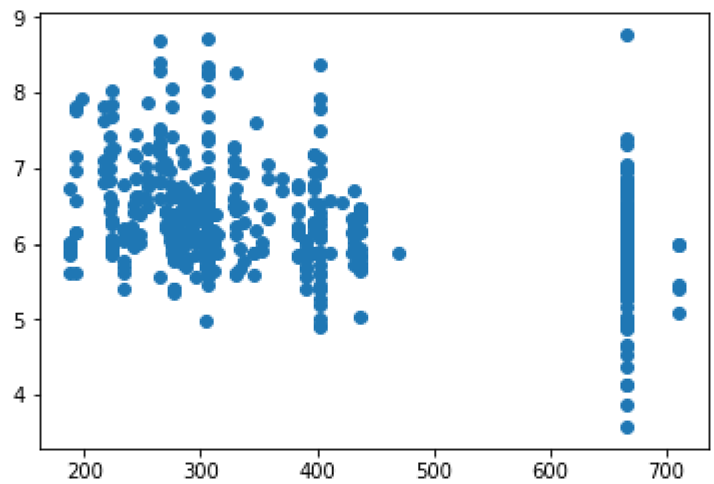
Out[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x177a2461358>
```



What's the correlation between property taxes and the number of rooms in a house?

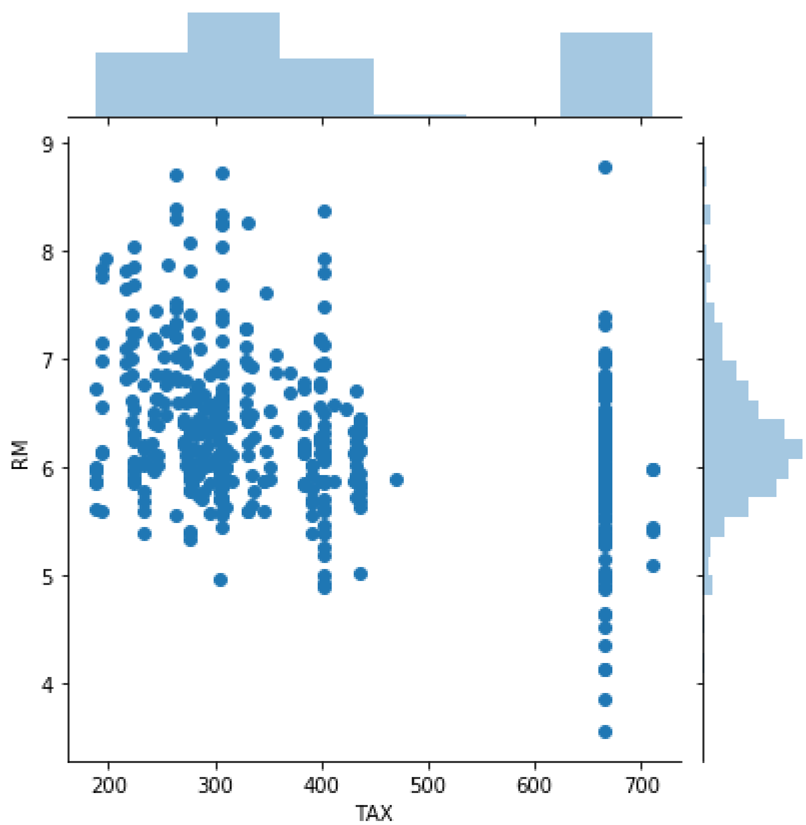
In [13]:
```
plt.scatter(df['TAX'], df['RM'])
```

Out[13]: <matplotlib.collections.PathCollection at 0x177a25615f8>



In [14]:
```
sns.jointplot(df['TAX'], df['RM'], kind='scatter')
```

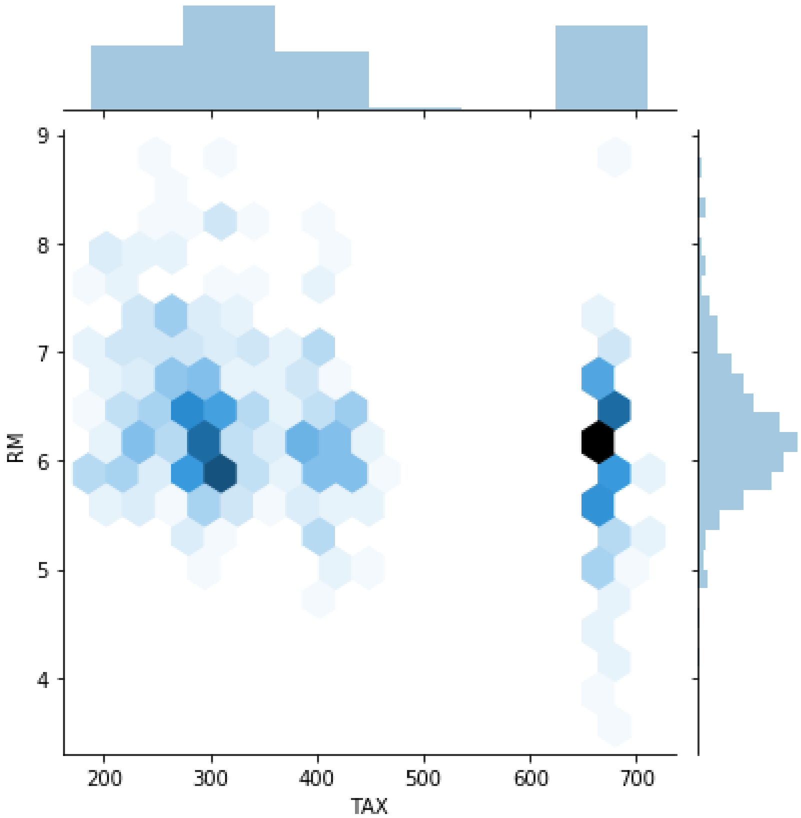
Out[14]: <seaborn.axisgrid.JointGrid at 0x177a256dcf8>



Another possibility is to aggregate data points over 2D areas and estimate the [probability desnsity function (https://en.wikipedia.org/wiki/Probability_density_function)](https://en.wikipedia.org/wiki/Probability_density_function). Its a 2D generalization of a histogram. We can either use a rectangular grid, or even a hexagonal one.

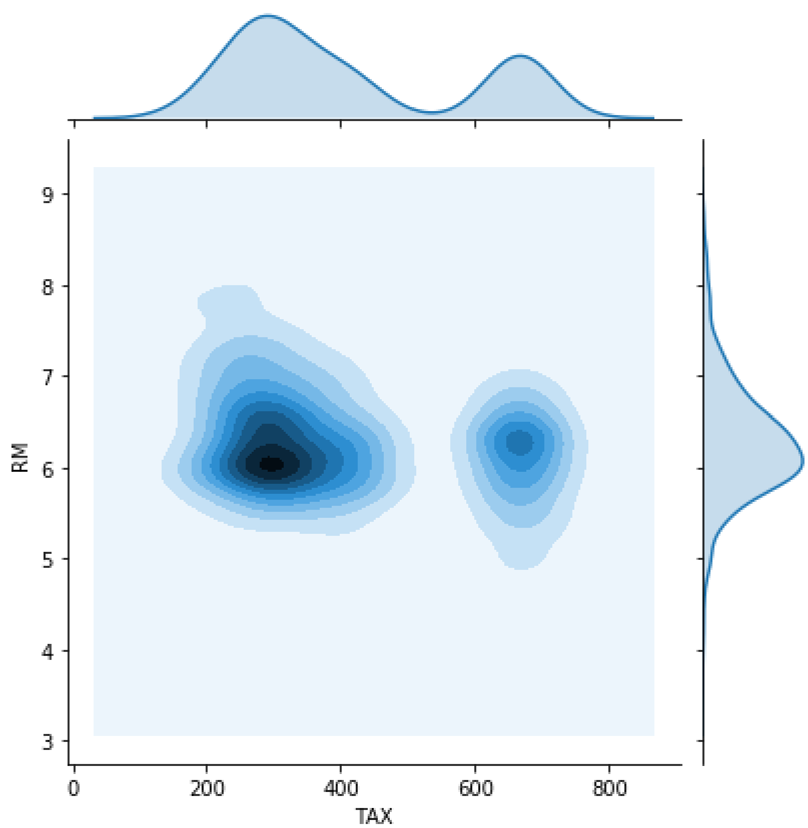
In [15]:
```
sns.jointplot(df['TAX'], df['RM'], kind='hex')
```

Out[15]: <seaborn.axisgrid.JointGrid at 0x177a2664358>



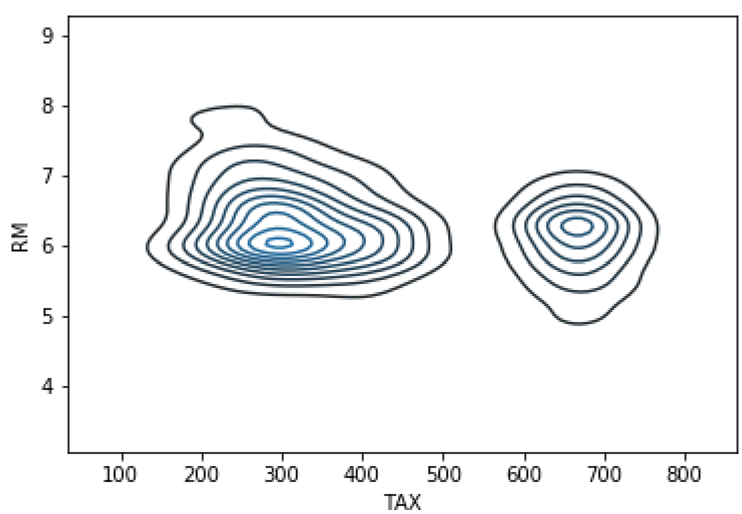
In [16]:
```
sns.jointplot(df['TAX'], df['RM'], kind='kde')
```

Out[16]: <seaborn.axisgrid.JointGrid at 0x177a27eed68>



In [17]:
```
sns.kdeplot(df['TAX'], df['RM'])
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x177a237b278>



What you'll see is you have access to so many visualizations. A great way to explore them is through the gallery: [https://seaborn.pydata.org/examples/index.html (https://seaborn.pydata.org/examples/index.html)](https://seaborn.pydata.org/examples/index.html)

How to implement Random Forest

First, we need to get a train and test dataset going...

In [18]:
```
from sklearn.model_selection import train_test_split

X = df
```

In [19]:
```
y = data.target
```

In [20]:
```
print(X.shape, y.shape)
```

```
(506, 13) (506,)
```

In [21]:
```
X_train, X_test, Y_train, Y_test = train_test_split (X, y, test_size = 0.25, r
andom_state = 0)
print(X_train.shape, Y_train.shape)
```

```
(379, 13) (379,)
```

The 'ravel' command flattens an array: "ravel(): when you have y.shape == (10, 1), using y.ravel().shape == (10,). In words... it flattens an array."

https://stackoverflow.com/questions/34165731/a-column-vector-y-was-passed-when-a-1d-array-was-expected (https://stackoverflow.com/questions/34165731/a-column-vector-y-was-passed-when-a-1d-array-was-expected)

In [22]:
```
reg = ske.RandomForestRegressor(n_estimators = 1000, random_state = 0)
```

In [23]:
```
Y_train = np.ravel(Y_train)  #this is redundant here because the original data
set was already flat
print(Y_train.shape)
```

```
(379,)
```

In [24]:
```
reg.fit(X_train, Y_train)
```

Out[24]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=1000, n_jobs=None,
           oob_score=False, random_state=0, verbose=0, warm_start=False)
```

In [25]:
```
Y_pred = reg.predict(X_test)
```

How do we evaluate this model? Previously, we've worked with labels for classifications but now instead of a DISCRETE target, we've got a continuous target. For example, the confusion matrix doesn't make sense and the code will error out below:

In [26]:
```
from sklearn.metrics import confusion_matrix

confusion_matrix(Y_test, Y_pred)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-26-237cb7848c80> in <module>
      1 from sklearn.metrics import confusion_matrix
      2 
----> 3 confusion_matrix(Y_test, Y_pred)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.py
 in confusion_matrix(y_true, y_pred, labels, sample_weight)
    251 
    252     """
--> 253     y_type, y_true, y_pred = _check_targets(y_true, y_pred)
    254     if y_type not in ("binary", "multiclass"):
    255         raise ValueError("%s is not supported" % y_type)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.py
 in _check_targets(y_true, y_pred)
     86     # No metrics support "multiclass-multioutput" format
     87     if (y_type not in ["binary", "multiclass", "multilabel-indicator"
]):
---> 88         raise ValueError("{0} is not supported".format(y_type))
     89 
     90     if y_type in ["binary", "multiclass"]:

ValueError: continuous is not supported
```

Check out this [documentation (https://scikit-learn.org/stable/modules/model_evaluation.html)](https://scikit-learn.org/stable/modules/model_evaluation.html) and see if you can find some ways to evaluate this model.

In [27]:
```
from sklearn.metrics import explained_variance_score, mean_absolute_error, mea
n_squared_error, r2_score
```

The importance of our features can be found in reg.feature*importances*. We sort them by decreasing order of importance:

In [28]:
```
explained_variance_score(Y_test, Y_pred)
```

Out[28]:
```
0.8038932325577687
```

In [29]:
```
mean_absolute_error(Y_test, Y_pred)
```
Out[29]:
```
2.5419283464567104
```

In [30]:
```
mean_squared_error(Y_test, Y_pred)
```
Out[30]:
```
16.427632250630026
```

In [31]:
```
r2_score(Y_test, Y_pred)
```
Out[31]:
```
0.7989249666895867
```

In [32]:
```
reg.feature_importances_
```
Out[32]:
```
array([0.03802909, 0.00096075, 0.00795783, 0.00118918, 0.0158759 ,
       0.39465879, 0.01264835, 0.04119215, 0.00404612, 0.01747215,
       0.02039368, 0.0102544 , 0.43532161])
```

We can compute how much each feature contributes to decreasing the weighted impurity within a tree. This is a fast calculation, but one should be cautious because it can be a biased approach. It has a tendency to inflate the importance of continuous features or high-cardinality categorical variables (a lot of very uncommon or unique variables).

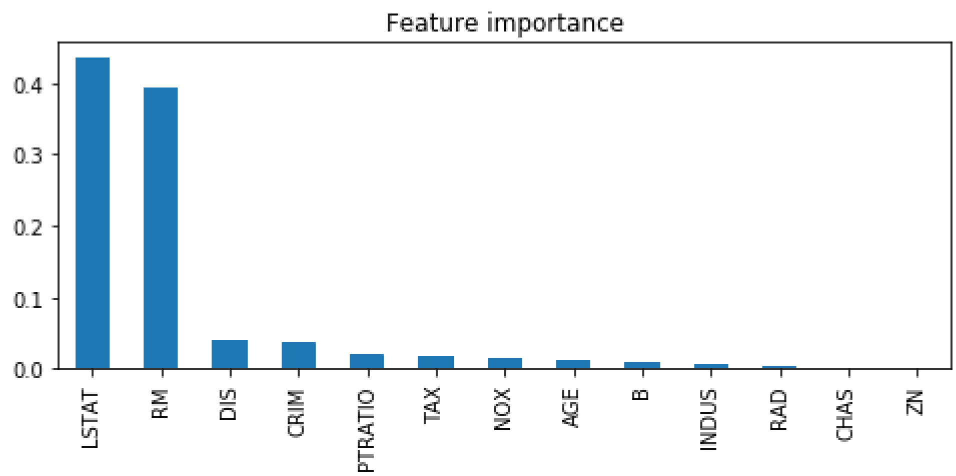
In [38]:
```
fet_ind = np.argsort(reg.feature_importances_)[::-1]
```

In [39]:
```
fet_imp = reg.feature_importances_[np.argsort(reg.feature_importances_)][::-1]
```

In [40]:
```
data['feature_names'][fet_ind]
```
Out[40]:
```
array(['LSTAT', 'RM', 'DIS', 'CRIM', 'PTRATIO', 'TAX', 'NOX', 'AGE', 'B',
       'INDUS', 'RAD', 'CHAS', 'ZN'], dtype='<U7')
```

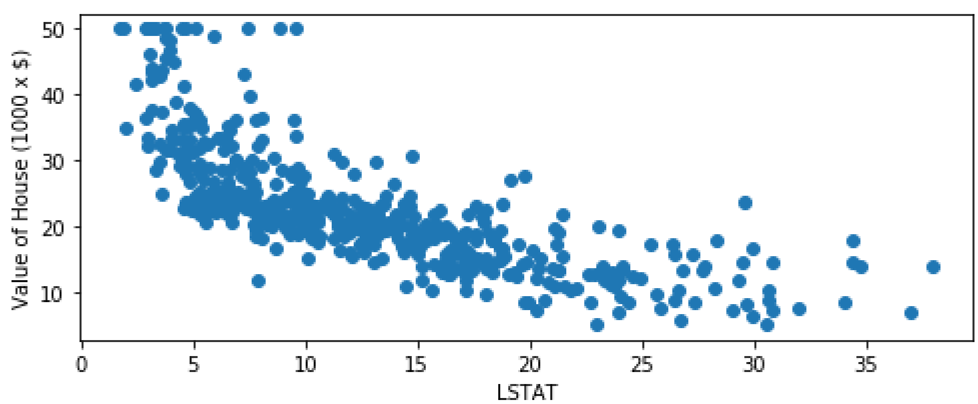
In [44]:
```
fig, ax = plt.subplots(1, 1, figsize=(8,3))
labels = data['feature_names'][fet_ind]
pd.Series(fet_imp, index = labels).plot('bar', ax=ax)
ax.set_title('Feature importance')
```

Out[44]:
```
Text(0.5, 1.0, 'Feature importance')
```



In [46]:
```
fig, ax = plt.subplots(1, 1, figsize=(8,3))
ax.scatter(X['LSTAT'],y)
ax.set_xlabel('LSTAT')
ax.set_ylabel('Value of House (1000 x $)')
```

Out[46]:
```
Text(0, 0.5, 'Value of House (1000 x $)')
```



In [48]:
```
from sklearn import tree

reg.estimators_[0]
```

Out[48]:
```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features='auto',
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=209652396, splitter='best')
```

In [49]:
```
tree.export_graphviz(reg.estimators_[0], 'tree.dot')
```

You'll need to open tree.dot file in a text editor, e.g., notepad. Select all the code and paste in here: http://www.webgraphviz.com/ (http://www.webgraphviz.com/). Scroll right and the tree should show up.

More practice - optional but recommended because its interesting and doesn't take too long

This is another good tutorial (https://towardsdatascience.com/random-forest-in-python-24d0893d51c0) on random forest: . You can perform this tutorial on your own and expand it for your choose your adventure, though you should be sure to demonstrate knowledge of this topic vs. copying and executing the tutorial.

In [52]:
```
# alternative tree structure
reg.estimators_[5]

tree.export_graphviz(reg.estimators_[5], 'tree2.dot', feature_names = data['fe
ature_names'], rounded = True, precision=1)
```

In [ ]:
```
```