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
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
```

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

Out[2]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
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	
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	
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	
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	
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	

In [3]:
```
df.shape
```

Out[3]:
```
(506, 13)
```

In [4]:
```
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 (attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots ove
r 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 cent
res
        - 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 b
lacks 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 Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hed
onic
prices and the demand for clean air', J. Environ. Economics & Manage
ment,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression di
agnostics
...', Wiley, 1980.   N.B. Various transformations are used in the ta
ble on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning p
apers that address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Inf
luential Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Lea
rning. In Proceedings on the Tenth International Conference of Machi
ne Learning, 236-243, University of Massachusetts, Amherst. Morgan K
aufmann.
```

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

In [5]:
```
pd.isnull(df).any()
```

Out[5]:
```
CRIM       False
ZN         False
INDUS      False
CHAS       False
NOX        False
RM         False
AGE        False
DIS        False
RAD        False
TAX        False
PTRATIO    False
B          False
LSTAT      False
dtype: bool
```

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 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 [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	5
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	

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]:

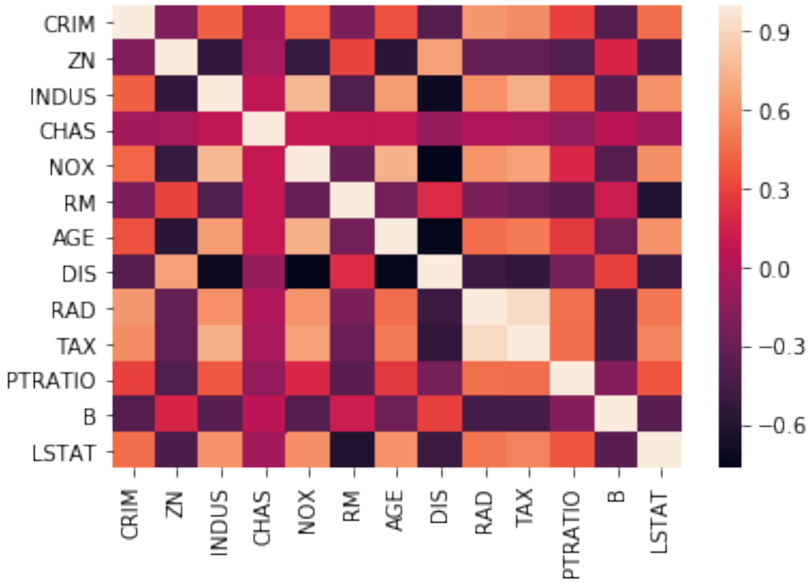
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DI
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.37967
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.66440
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.70802
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.09917
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.76923
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.20524
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.74788
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.00000
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.49458
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.53443
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.23247
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.29151
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.49699

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.

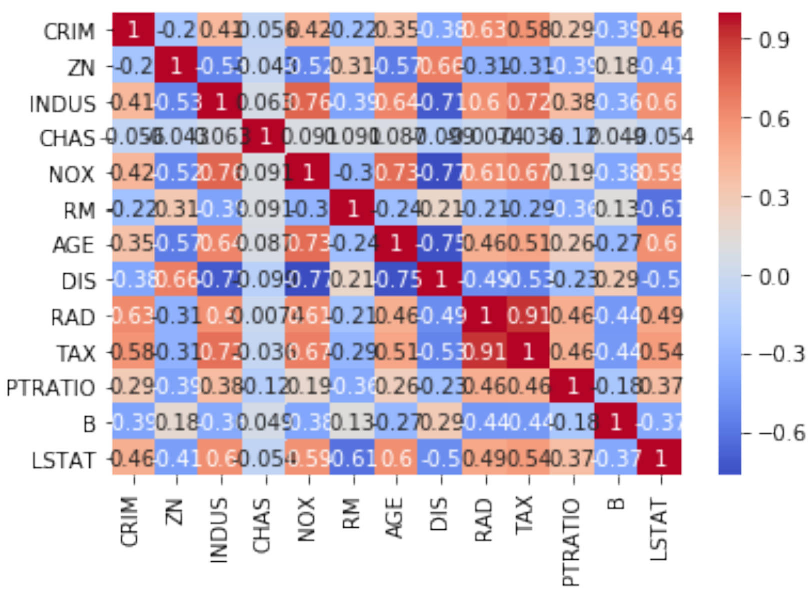
In [9]:
```
import seaborn as sns
sns.heatmap(corr)
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x1a19e07278>



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

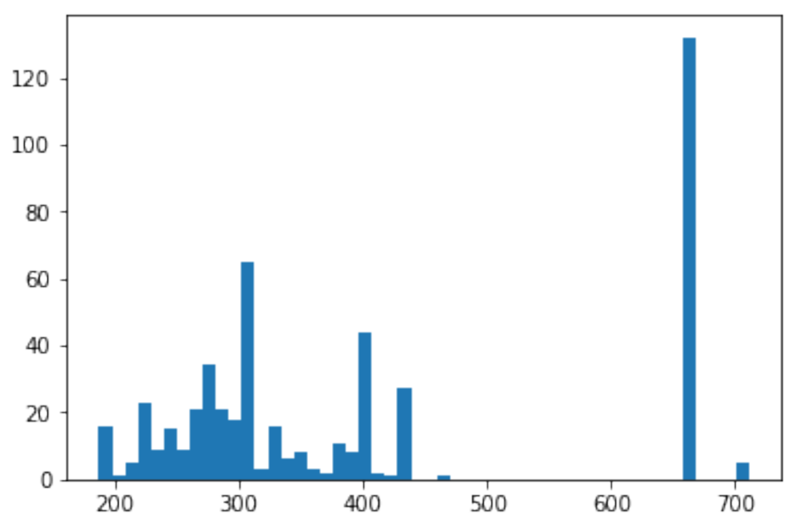
In [10]:
```
sns.heatmap(corr, annot = True, cmap = 'coolwarm')
plt.savefig('heatmap.png', tight_layout = True)
```



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!

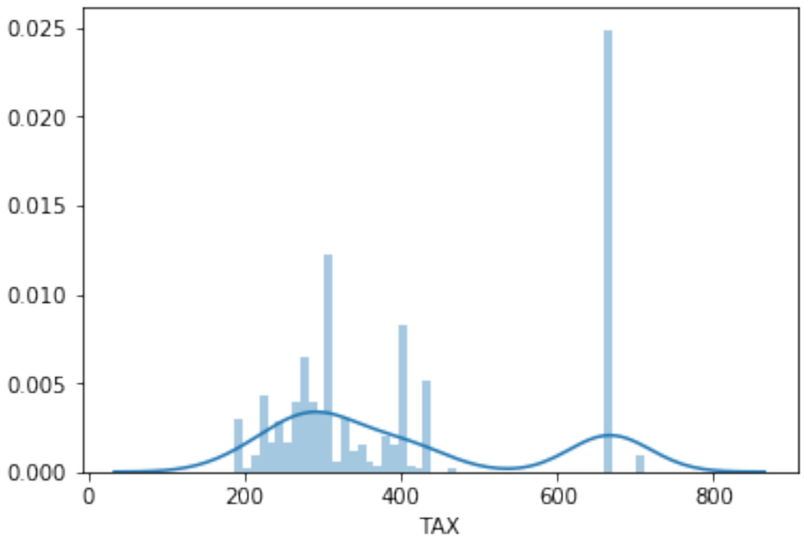
In [11]:
```
attr = df['TAX']
plt.hist(attr, bins = 50)
```

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)
```

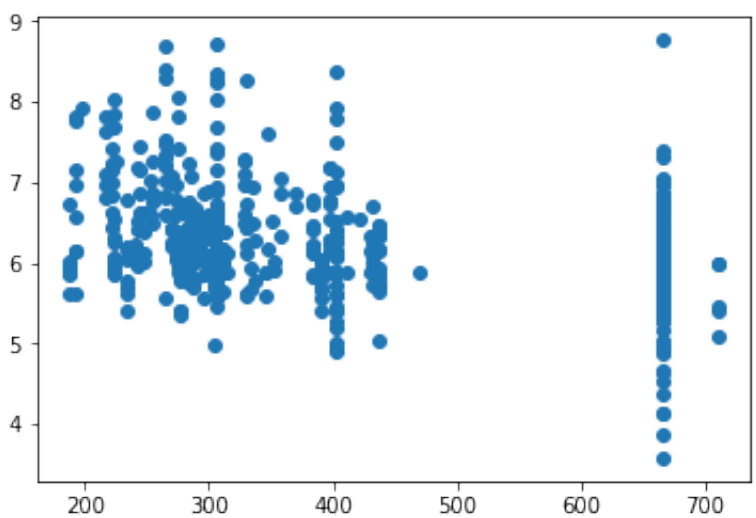
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1a525ba8>



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

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

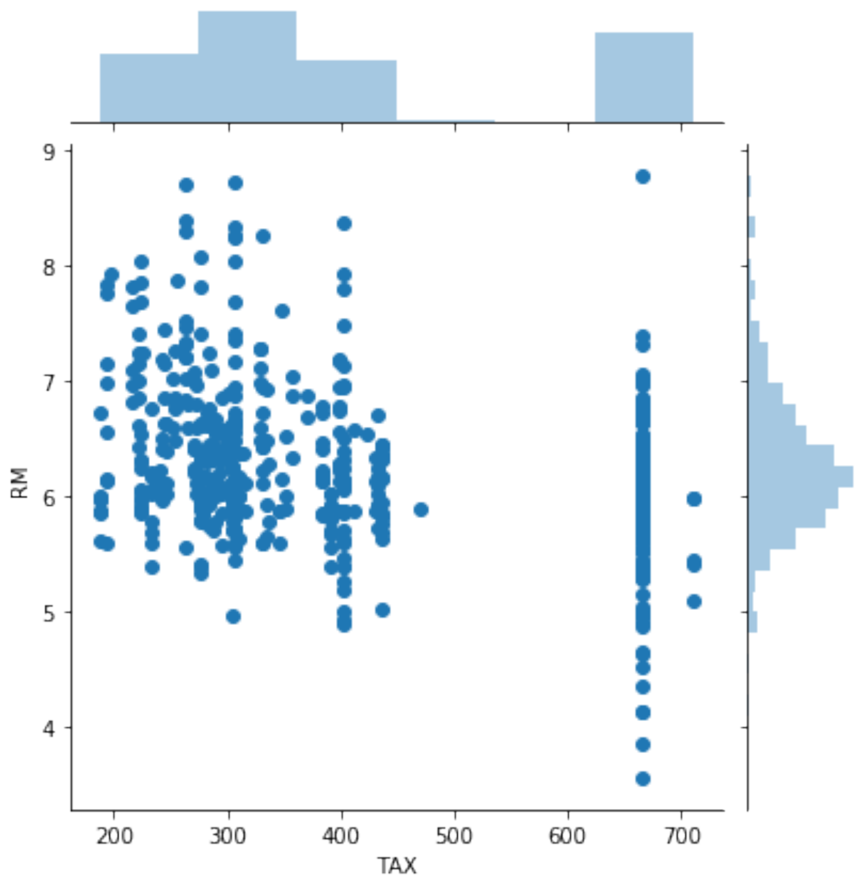
Out[13]: <matplotlib.collections.PathCollection at 0x1a1a280940>



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.

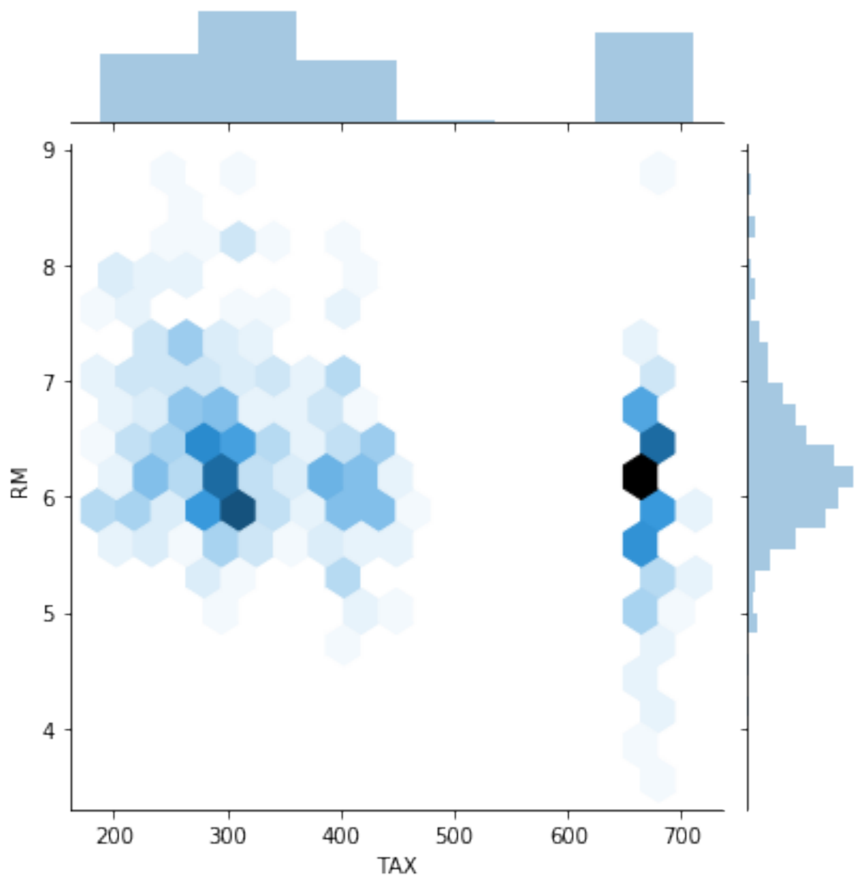
In [14]:
```
sns.jointplot(df['TAX'], df['RM'], kind = 'scatter')
```

Out[14]: <seaborn.axisgrid.JointGrid at 0x1a1a27b160>



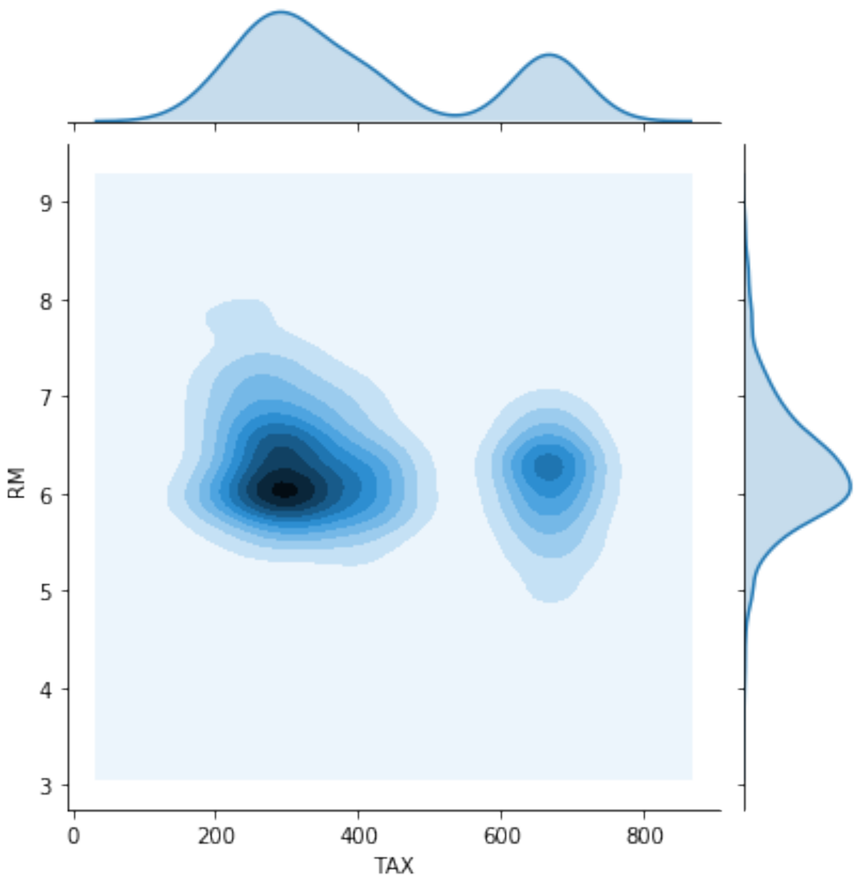
In [15]:
```
sns.jointplot(df['TAX'], df['RM'], kind = 'hex')
```

Out[15]: <seaborn.axisgrid.JointGrid at 0x1062705c0>



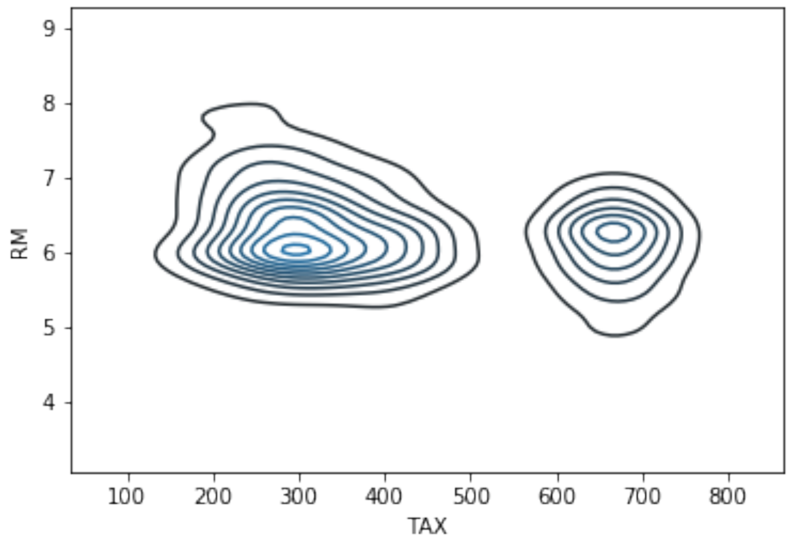
In [16]:
```
sns.jointplot(df['TAX'], df['RM'], kind = 'kde')
```

Out[16]: <seaborn.axisgrid.JointGrid at 0x1a1a773fd0>



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

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1a93fc50>



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
y = data.target
```

In [19]:
```
print(x.shape, y.shape)
```

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

In [20]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size =
0.25, random_state = 0)
```

In [21]:
```
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)](https://stackoverflow.com/questions/34165731/a-column-vector-y-was-passed-when-a-1d-array-was-expected)

In [22]:
```
y_train = np.ravel(y_train)
```

In [23]:
```
reg = ske.RandomForestRegressor(n_estimators= 1000, random_state= 0)
```

In [24]:
```
reg.fit(x_train, y_train)
```

Out[24]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=Non
e,
                      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=N
one,
                      oob_score=False, random_state=0, verbose=0, warm_start=Fa
lse)
```

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 [25]:
```
y_pred = reg.predict(x_test)
```

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

### ERROR confusion_matrix(y_test, y_pred)
```

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_er
ror, mean_squared_error, r2_score
# max_error
```

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]:
```
# max_error(y_test, y_pred)
```

In [30]:
```
print(mean_absolute_error(y_test, y_pred))
print(mean_squared_error(y_test, y_pred))
print(r2_score(y_test, y_pred, multioutput = 'variance_weighted'))
```

```
2.5419283464567104
16.427632250630026
0.7989249666895868
```

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 [31]:
```
reg.feature_importances_
```

Out[31]:
```
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])
```

In [32]:
```
fet_ind = np.argsort(reg.feature_importances_)
```

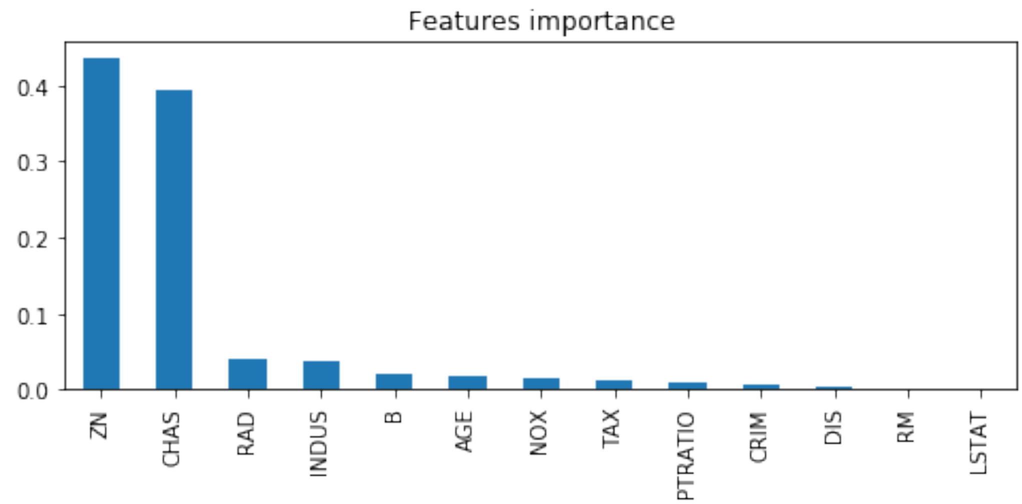
In [33]:
```
fet_imp = reg.feature_importances_[np.argsort(reg.feature_importances_
)][::-1]
```

In [34]:
```
data['feature_names'][fet_ind]
```

Out[34]:
```
array(['ZN', 'CHAS', 'RAD', 'INDUS', 'B', 'AGE', 'NOX', 'TAX', 'PTRA
TIO',
       'CRIM', 'DIS', 'RM', 'LSTAT'], dtype='<U7')
```

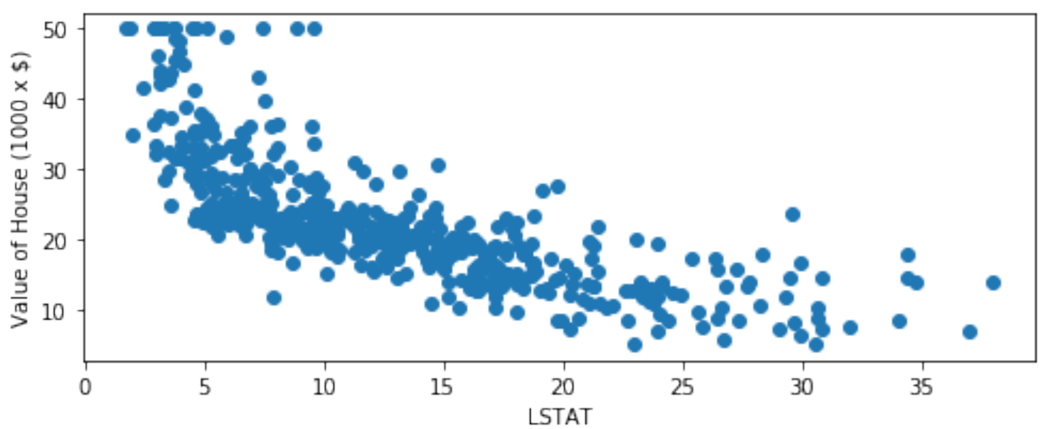
In [35]:
```
fig, ax = plt.subplots(1, 1, figsize = (8, 3))
labels = data['feature_names'][fet_ind]
pd.Series(fet_imp, index= labels).plot('bar', ax=ax)
ax.set_title('Features importance')
```

Out[35]: Text(0.5, 1.0, 'Features importance')



In [36]:
```
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[36]: Text(0, 0.5, 'Value of House (1000 x $)')



In [37]:
```
from sklearn import tree

reg.estimators_[0]
```

Out[37]:
```
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 [38]:
```
tree.export_graphviz(reg.estimators_[0],'tree.dot', feature_names= dat
a['feature_names'], rounded =True, precision = 1)
```

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 [39]:
```
import pandas as pd

features = pd.read_csv('temps.csv')
features.head()
```

Out[39]:

	year	month	day	week	temp_2	temp_1	average	actual	forecast_noaa	forecast_acc	fo
0	2016	1	1	Fri	45	45	45.6	45	43	50	
1	2016	1	2	Sat	44	45	45.7	44	41	50	
2	2016	1	3	Sun	45	44	45.8	41	43	46	
3	2016	1	4	Mon	44	41	45.9	40	44	48	
4	2016	1	5	Tues	41	40	46.0	44	46	46	

In [40]:
```
print('The shape of our features is:', features.shape)
```

```
The shape of our features is: (348, 12)
```

In [41]:
```python
# Descriptive statistics for each column
features.describe()
```

Out[41]:

	year	month	day	temp_2	temp_1	average	actual	foreca
count	348.0	348.000000	348.000000	348.000000	348.000000	348.000000	348.000000	34
mean	2016.0	6.477011	15.514368	62.652299	62.701149	59.760632	62.543103	5
std	0.0	3.498380	8.772982	12.165398	12.120542	10.527306	11.794146	1
min	2016.0	1.000000	1.000000	35.000000	35.000000	45.100000	35.000000	4
25%	2016.0	3.000000	8.000000	54.000000	54.000000	49.975000	54.000000	4
50%	2016.0	6.000000	15.000000	62.500000	62.500000	58.200000	62.500000	5
75%	2016.0	10.000000	23.000000	71.000000	71.000000	69.025000	71.000000	6
max	2016.0	12.000000	31.000000	117.000000	117.000000	77.400000	92.000000	7

In [42]:
```python
# One-hot encode the data using pandas get_dummies
features = pd.get_dummies(features)
# Display the first 5 rows of the last 12 columns
features.iloc[:,5:].head(5)
```

Out[42]:

	average	actual	forecast_noaa	forecast_acc	forecast_under	friend	week_Fri	week_Mon
0	45.6	45	43	50	44	29	1	0
1	45.7	44	41	50	44	61	0	0
2	45.8	41	43	46	47	56	0	0
3	45.9	40	44	48	46	53	0	1
4	46.0	44	46	46	46	41	0	0

In [43]:
```python
# Use numpy to convert to arrays
import numpy as np
# Labels are the values we want to predict
labels = np.array(features['actual'])
# Remove the labels from the features
# axis 1 refers to the columns
features= features.drop('actual', axis = 1)
# Saving feature names for later use
feature_list = list(features.columns)
# Convert to numpy array
features = np.array(features)
```

In [44]:
```
# Using Skicit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets
train_features, test_features, train_labels, test_labels = train_test_
split(features, labels, test_size = 0.25, random_state = 42)
```

In [45]:
```
print('Training Features Shape:', train_features.shape)
print('Training Labels Shape:', train_labels.shape)
print('Testing Features Shape:', test_features.shape)
print('Testing Labels Shape:', test_labels.shape)
```

```
Training Features Shape: (261, 17)
Training Labels Shape: (261,)
Testing Features Shape: (87, 17)
Testing Labels Shape: (87,)
```

In [46]:
```
# The baseline predictions are the historical averages
baseline_preds = test_features[:, feature_list.index('average')]
# Baseline errors, and display average baseline error
baseline_errors = abs(baseline_preds - test_labels)
print('Average baseline error: ', round(np.mean(baseline_errors), 2))
```

```
Average baseline error:  5.06
```

In [47]:
```
# Import the model we are using
from sklearn.ensemble import RandomForestRegressor
# Instantiate model with 1000 decision trees
rf = RandomForestRegressor(n_estimators = 1000, random_state = 42)
# Train the model on training data
rf.fit(train_features, train_labels);
```

In [48]:
```
# Use the forest's predict method on the test data
predictions = rf.predict(test_features)
# Calculate the absolute errors
errors = abs(predictions - test_labels)
# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
```

```
Mean Absolute Error: 3.87 degrees.
```

In [49]:
```
# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / test_labels)
# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
Accuracy: 93.93 %.
```

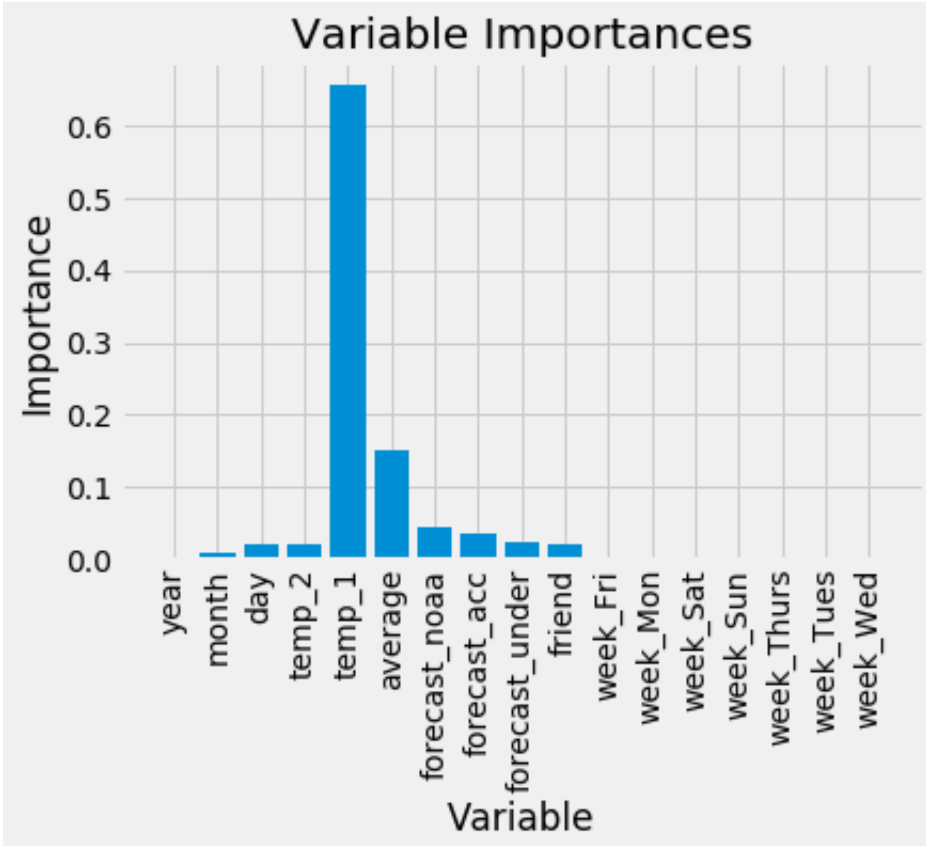
In [50]:
```
# Get numerical feature importances
importances = list(rf.feature_importances_)
# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 2)) for feature, im
portance in zip(feature_list, importances)]
# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x: x[1]
, reverse = True)
# Print out the feature and importances
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in fea
ture_importances];
```

```
Variable: temp_1               Importance: 0.66
Variable: average              Importance: 0.15
Variable: forecast_noaa        Importance: 0.05
Variable: forecast_acc         Importance: 0.03
Variable: day                  Importance: 0.02
Variable: temp_2               Importance: 0.02
Variable: forecast_under       Importance: 0.02
Variable: friend               Importance: 0.02
Variable: month                Importance: 0.01
Variable: year                 Importance: 0.0
Variable: week_Fri             Importance: 0.0
Variable: week_Mon             Importance: 0.0
Variable: week_Sat             Importance: 0.0
Variable: week_Sun             Importance: 0.0
Variable: week_Thurs           Importance: 0.0
Variable: week_Tues            Importance: 0.0
Variable: week_Wed             Importance: 0.0
```

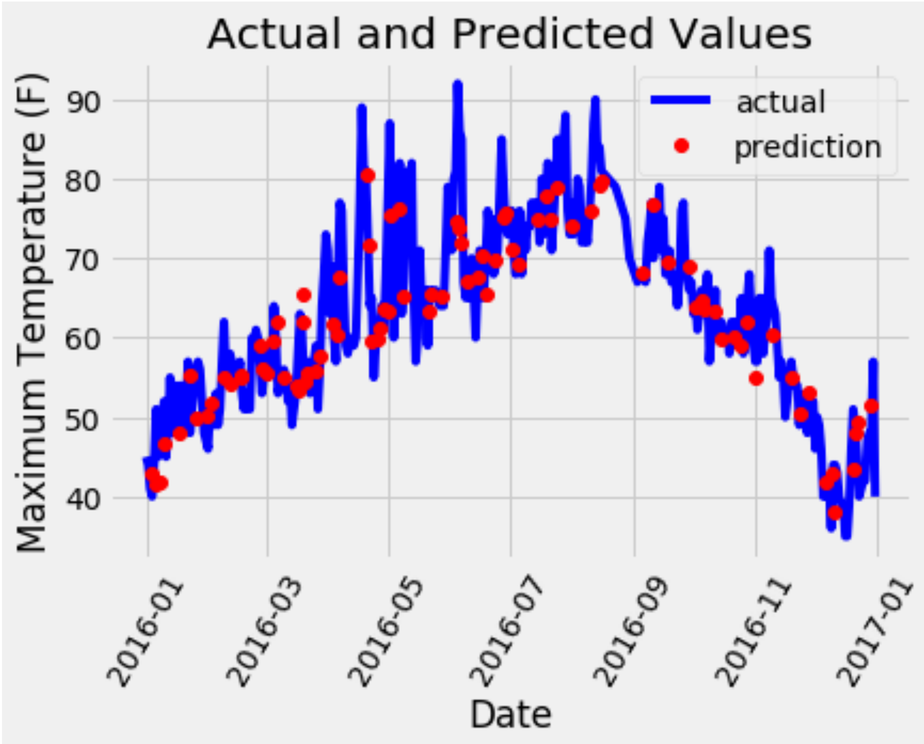
In [51]:
```
# New random forest with only the two most important variables
rf_most_important = RandomForestRegressor(n_estimators= 1000, random_state=42)
# Extract the two most important features
important_indices = [feature_list.index('temp_1'), feature_list.index('average')]
train_important = train_features[:, important_indices]
test_important = test_features[:, important_indices]
# Train the random forest
rf_most_important.fit(train_important, train_labels)
# Make predictions and determine the error
predictions = rf_most_important.predict(test_important)
errors = abs(predictions - test_labels)
# Display the performance metrics
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
mape = np.mean(100 * (errors / test_labels))
accuracy = 100 - mape
print('Accuracy:', round(accuracy, 2), '%.')
```

```
Mean Absolute Error: 3.92 degrees.
Accuracy: 93.76 %.
```

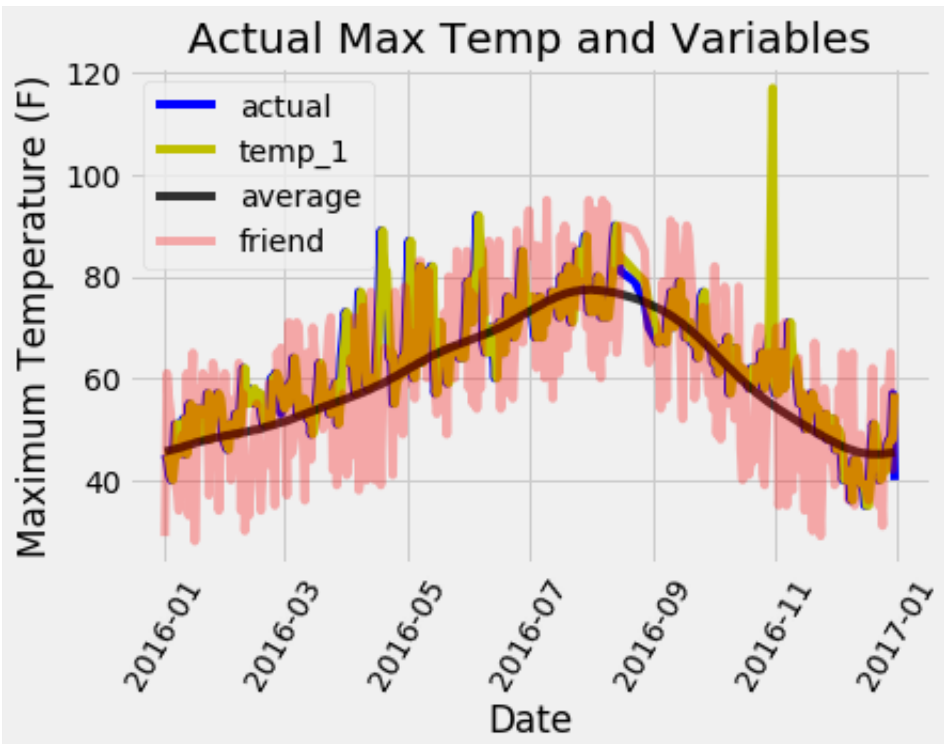
In [52]:
```
# Import matplotlib for plotting and use magic command for Jupyter Notebooks
import matplotlib.pyplot as plt
%matplotlib inline
# Set the style
plt.style.use('fivethirtyeight')
# list of x locations for plotting
x_values = list(range(len(importances)))
# Make a bar chart
plt.bar(x_values, importances, orientation = 'vertical')
# Tick labels for x axis
plt.xticks(x_values, feature_list, rotation='vertical')
# Axis labels and title
plt.ylabel('Importance'); plt.xlabel('Variable'); plt.title('Variable Importances');
```


Variable Importances
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Importance
year
month
day
temp_2
temp_1
average
forecast_noaa
forecast_acc
forecast_under
friend
week_Fri
week_Mon
week_Sat
week_Sun
week_Thurs
week_Tues
week_Wed
Variable

In [53]:
```
# Use datetime for creating date objects for plotting
import datetime
# Dates of training values
months = features[:, feature_list.index('month')]
days = features[:, feature_list.index('day')]
years = features[:, feature_list.index('year')]
# List and then convert to datetime object
dates = [str(int(year)) + '-' + str(int(month)) + '-' + str(int(day))
for year, month, day in zip(years, months, days)]
dates = [datetime.datetime.strptime(date, '%Y-%m-%d') for date in date
s]
# Dataframe with true values and dates
true_data = pd.DataFrame(data = {'date': dates, 'actual': labels})
# Dates of predictions
months = test_features[:, feature_list.index('month')]
days = test_features[:, feature_list.index('day')]
years = test_features[:, feature_list.index('year')]
# Column of dates
test_dates = [str(int(year)) + '-' + str(int(month)) + '-' + str(int(d
ay)) for year, month, day in zip(years, months, days)]
# Convert to datetime objects
test_dates = [datetime.datetime.strptime(date, '%Y-%m-%d') for date in
test_dates]
# Dataframe with predictions and dates
predictions_data = pd.DataFrame(data = {'date': test_dates, 'predictio
n': predictions})
# Plot the actual values
plt.plot(true_data['date'], true_data['actual'], 'b-', label = 'actual
')
# Plot the predicted values
plt.plot(predictions_data['date'], predictions_data['prediction'], 'ro
', label = 'prediction')
plt.xticks(rotation = '60');
plt.legend()
# Graph labels
plt.xlabel('Date'); plt.ylabel('Maximum Temperature (F)'); plt.title('
Actual and Predicted Values');
```


Actual and Predicted Values
Maximum Temperature (F)
90
80
70
60
50
40
actual
prediction
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01
Date

In [54]:
```
# Make the data accessible for plotting
true_data['temp_1'] = features[:, feature_list.index('temp_1')]
true_data['average'] = features[:, feature_list.index('average')]
true_data['friend'] = features[:, feature_list.index('friend')]
# Plot all the data as lines
plt.plot(true_data['date'], true_data['actual'], 'b-', label  = 'actua
l', alpha = 1.0)
plt.plot(true_data['date'], true_data['temp_1'], 'y-', label  = 'temp_
1', alpha = 1.0)
plt.plot(true_data['date'], true_data['average'], 'k-', label = 'avera
ge', alpha = 0.8)
plt.plot(true_data['date'], true_data['friend'], 'r-', label = 'friend
', alpha = 0.3)
# Formatting plot
plt.legend(); plt.xticks(rotation = '60');
# Lables and title
plt.xlabel('Date'); plt.ylabel('Maximum Temperature (F)'); plt.title('
Actual Max Temp and Variables');
```



Original code are posted on github. ([https://github.com/joyleeisu/ABE516X-Random-Forest.git (https://github.com/joyleeisu/ABE516X-Random-Forest.git)](https://github.com/joyleeisu/ABE516X-Random-Forest.git))