boston_housing

May 10, 2016

1 Boston Housing Prices:

The objective of this workbook is to generate an optimal model based on a statistical analysis with the tools available to estimate the best selling price for the client's home. Additional information on the Boston Housing dataset can be found here. The source code developed for this project can be found in the ipython notebook: boston_housing.ipynb

```
In [1]: %matplotlib inline
```

```
In [2]: """Load the Boston dataset and examine its target (label) distribution."""
        # Load libraries
        import os
        import numpy as np
        import pylab as pl
        import matplotlib.pyplot as pl

        from sklearn import datasets
        from sklearn.tree import DecisionTreeRegressor
        from sklearn.metrics import mean_squared_error, mean_absolute_error, make_scorer
        from sklearn.cross_validation import train_test_split
        from sklearn.grid_search import GridSearchCV
        from sklearn.linear_model import LinearRegression

        import seaborn as sns
        sns.set(style="ticks", color_codes=True)
        sns.set_context('notebook')
        import pandas as pd
```

1.0.1 Load Data

```
In [11]: def load_data():
             """Load the Boston dataset."""
             boston = datasets.load_boston()
             #print boston.keys()
             #print boston.DESCR
             return boston

         boston = load_data()
```

1.0.2 DataFrame

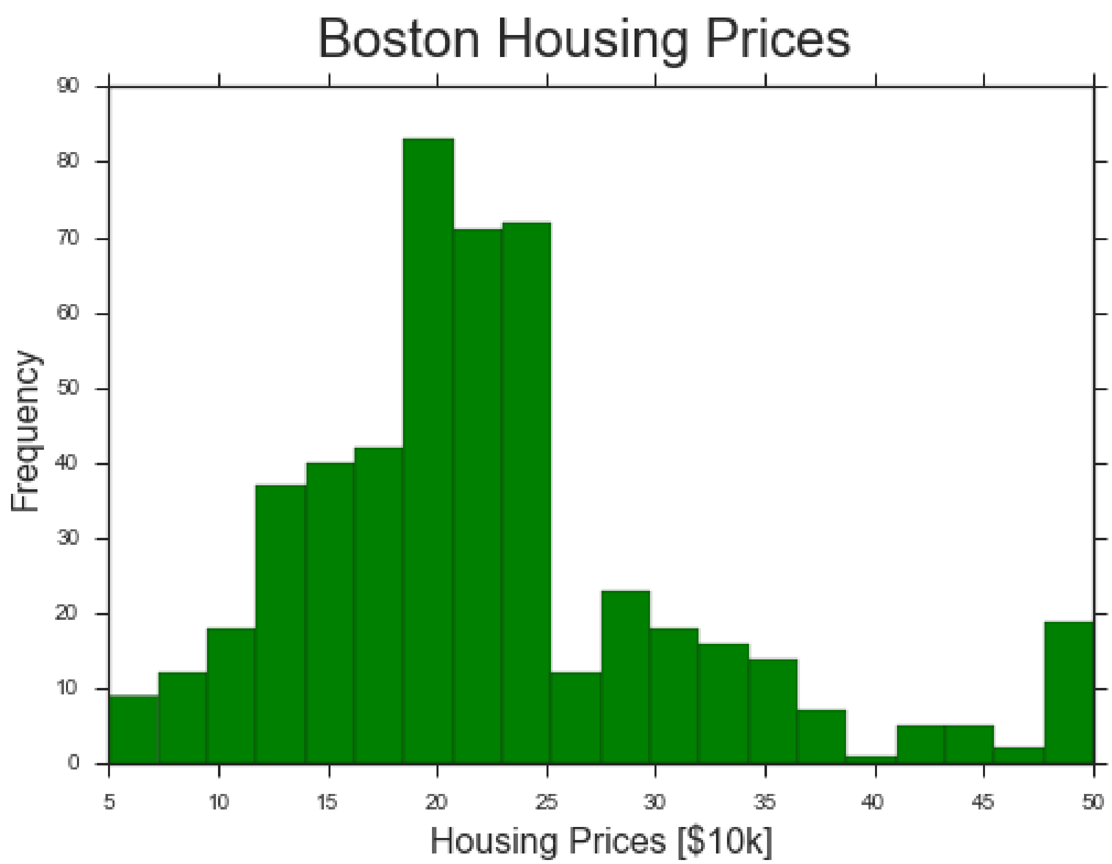
```
In [12]: def dataframe(city_data):
             housing_prices = city_data.target
             housing_features = city_data.data
```

```
    X,y = housing_features, housing_prices
    df_data = pd.DataFrame(housing_features, columns = boston.feature_names)
    df_target = pd.DataFrame(housing_prices, columns =['MEDV'])
    df_boston = pd.concat([df_data, df_target,], axis = 1)
    df = df_boston.corr()
    corr_target = df.ix[-1][:-1]
    predict = corr_target.sort(ascending=False)
    df_sort = corr_target.sort_values(ascending=False)
    print(df_sort)
    print(df)
    return df_boston
```

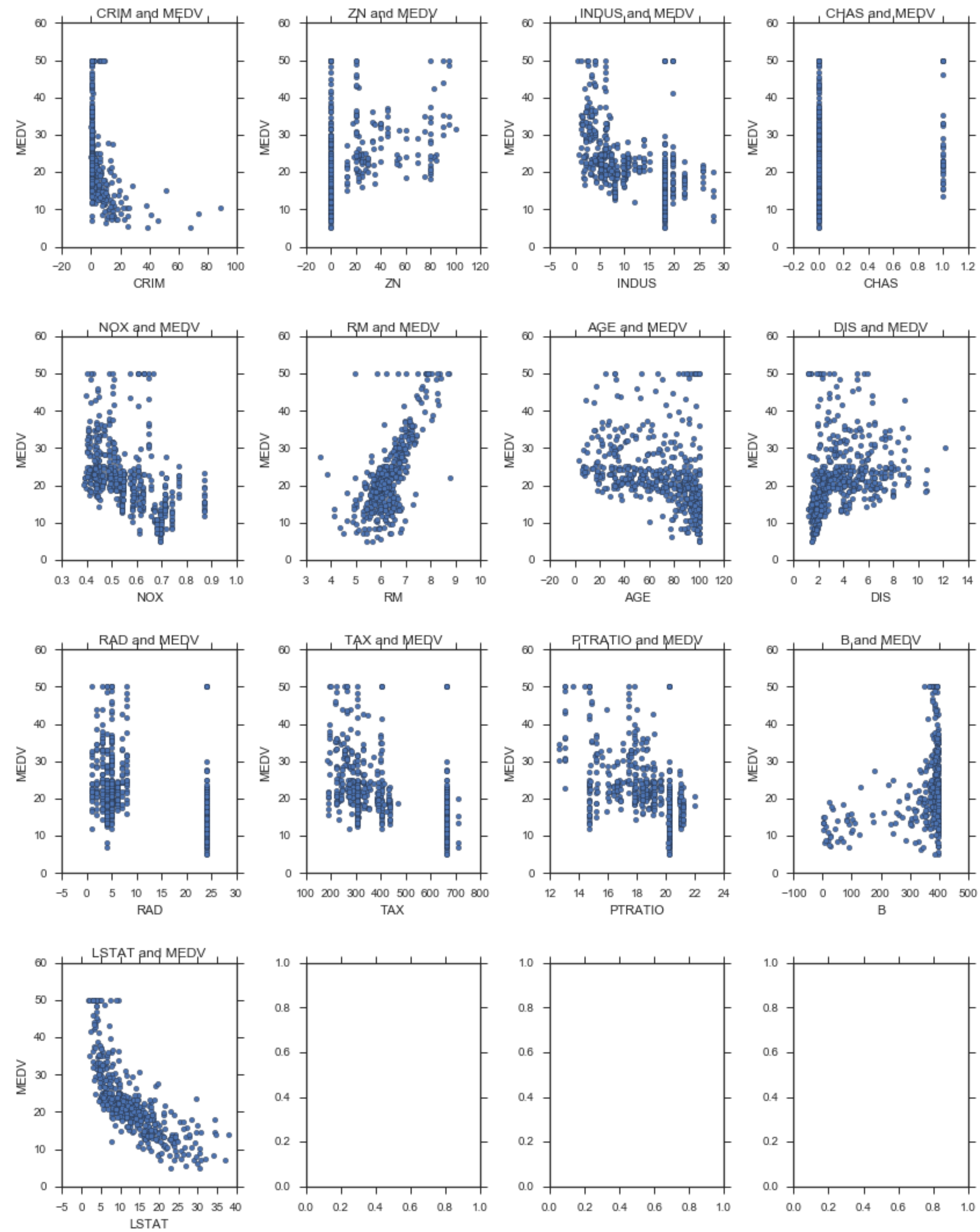
1.0.3 Statistical Analysis and Data Exploration

```
In [14]: def histogram(city_data):
             housing_prices = city_data.target
             housing_features = city_data.data
             X,y = housing_features, housing_prices
             pl.hist(y, bins =20, color = 'green')
             pl.suptitle('Boston Housing Prices', fontsize = 24)
             pl.xlabel('Housing Prices [$10k]', fontsize = 16)
             pl.ylabel('Frequency', fontsize = 16)
             pl.show()

         def scatter_plots(city_data):
             pl.figure()
             fig,axes = pl.subplots(4, 4, figsize=(14,18))
             fig.subplots_adjust(wspace=.4, hspace=.4)
             img_index = 0
             for i in range(boston.feature_names.size):
                 row, col = i // 4, i % 4
                 axes[row][col].scatter(boston.data[:,i],boston.target)
                 axes[row][col].set_title(boston.feature_names[i] + ' and MEDV')
                 axes[row][col].set_xlabel(boston.feature_names[i])
                 axes[row][col].set_ylabel('MEDV')
             pl.show()

         def explore_city_data(city_data):
             """Calculate the Boston housing statistics."""
             # Get the labels and features from the housing data
             housing_prices = city_data.target
             housing_features = city_data.data
             # Size of data (number of houses)?
             number_houses = housing_features.shape[0]
             print "Number of houses:", number_houses
             # Number of features?
             number_features = housing_features.shape[1]
             print "Number of features:", number_features
             # Minimum price?
             min_price = np.min(housing_prices)
             print "Minimum Housing Price: ${:,.2f}".format(min_price)
             # Maximum price?
             max_price = np.max(housing_prices)
             print "Maximum Housing Price: ${:,.2f}".format(max_price)
```

```
    # Calculate mean price?
    mean_price = np.mean(housing_prices)
    print "Mean Housing Price: ${:,.2f}".format(mean_price)
    # Calculate median price?
    median_price = np.median(housing_prices)
    print "Median Housing Price: ${:,.2f}".format(median_price)
    # Calculate standard deviation?
    std_price = np.std(housing_prices)
    print "Standard Deviation: ${:,.2f}".format(std_price)

if __name__ == "__main__":
    #Histogram
    histogram(city_data)
    scatter_plots(city_data)
    explore_city_data(city_data)
```




```
<matplotlib.figure.Figure at 0x116632f90>
```




```
Number of houses: 506
Number of features: 13
Minimum Housing Price: $5.00
Maximum Housing Price: $50.00
Mean Housing Price: $22.53
Median Housing Price: $21.20
```

```
Standard Deviation: $9.19
```

1.0.4 Evaluating Model Performance

```
In [45]: def split_data(city_data):
             # Get the features and labels from the Boston housing data
             X, y = city_data.data, boston.target
             X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,random_state=None
             print "X_training:", X_train.shape
             print "X_test:", X_test.shape
             return X_train, y_train, X_test, y_test
```

1.0.5 Performance Metric

```
In [29]: def performance_metric(label, prediction):
             """Calculates and returns the appropriate error performance metric."""
             #mae = mean_absolute_error(label, prediction)
             mse = mean_squared_error(label, prediction)
             return mse
             pass
```

1.0.6 Learning Curve

```
In [33]: depth = 5
         def learning_curve(depth, X_train, y_train, X_test, y_test):
             """Calculate the performance of the model after a set of training data."""

             # We will vary the training set size so that we have 50 different sizes
             sizes = np.round(np.linspace(1, len(X_train), 50))
             train_err = np.zeros(len(sizes))
             test_err = np.zeros(len(sizes))

             print "Decision Tree with Max Depth: "
             print (depth)

             for i, s in enumerate(sizes):

                 # Create and fit the decision tree regressor model
                 regressor = DecisionTreeRegressor(max_depth=depth)
                 regressor.fit(X_train[:s], y_train[:s])

                 # Find the performance on the training and testing set
                 train_err[i] = performance_metric(y_train[:s], regressor.predict(X_train[:s]))
                 test_err[i] = performance_metric(y_test, regressor.predict(X_test))


             # Plot learning curve graph
             learning_curve_graph(sizes, train_err, test_err)


         def learning_curve_graph(sizes, train_err, test_err):
             """Plot training and test error as a function of the training size."""
             pl.figure()
             pl.figure(figsize=(8,6))
             pl.title('Decision Trees: Performance vs Training Size', fontsize = 20)
```

```
        pl.plot(sizes, test_err, lw=2, label = 'test error')
        pl.plot(sizes, train_err, lw=2, label = 'training error')
        pl.legend()
        pl.xlabel('Training Size', fontsize = 14)
        pl.ylabel('Error', fontsize =14)
        pl.show()
```

In [34]:
```
learning_curve(depth, X_train, y_train, X_test, y_test)
```

```
Decision Tree with Max Depth:
5

/Users/tracesmith/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:17: DeprecationWarning: usi
/Users/tracesmith/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:20: DeprecationWarning: usi

<matplotlib.figure.Figure at 0x115fc0ed0>
```

1.0.7 Model Complexity

In [35]:
```
def model_complexity(X_train, y_train, X_test, y_test):
    """Calculate the performance of the model as model complexity increases."""

    print "Model Complexity: "
```

```
    # We will vary the depth of decision trees from 2 to 25
    max_depth = np.arange(1, 25)
    train_err = np.zeros(len(max_depth))
    test_err = np.zeros(len(max_depth))

    for i, d in enumerate(max_depth):
        # Setup a Decision Tree Regressor so that it learns a tree with depth d
        regressor = DecisionTreeRegressor(max_depth=d)

        # Fit the learner to the training data
        regressor.fit(X_train, y_train)

        # Find the performance on the training set
        train_err[i] = performance_metric(y_train, regressor.predict(X_train))

        # Find the performance on the testing set
        test_err[i] = performance_metric(y_test, regressor.predict(X_test))

    # Plot the model complexity graph
    model_complexity_graph(max_depth, train_err, test_err)


def model_complexity_graph(max_depth, train_err, test_err):
    """Plot training and test error as a function of the depth of the decision tree learn."""
    pl.figure()
    pl.title('Decision Trees: Performance vs Max Depth', fontsize = 20)
    pl.plot(max_depth, test_err, lw=2, label = 'test error')
    pl.plot(max_depth, train_err, lw=2, label = 'training error')
    pl.legend()
    pl.xlabel('Max Depth',fontsize =14)
    pl.ylabel('Error', fontsize =14)
    pl.show()
```

1.0.8 Fit Model

```
In [36]: def fit_predict_model(city_data):

    # Get the features and labels from the Boston housing data

    X, y = city_data.data, city_data.target

    # Setup a Decision Tree Regressor

    regressor = DecisionTreeRegressor()

    parameters = {'max_depth':(1,2,3,4,5,6,7,8,9,10)}

    mse_scoring = make_scorer(mean_squared_error, greater_is_better=False)

    #using grid search to fine tune the Decision Tree Regressor and
    #obtain the parameters that generate the best training performance.

    reg = GridSearchCV(regressor, parameters, scoring = mse_scoring)
```

```
    reg.fit(X,y)

    # Fit the learner to the training data to obtain the best parameter set
    print "Final Model: "
    print (reg.fit(X, y))

    # Using the model to predict the output of a particular sample
    x = [11.95, 0.00, 18.100, 0, 0.6590, 5.6090, 90.00, 1.385, 24, 680.0, 20.20, 332.09, 12.13]
    x = np.array(x)
    x = x.reshape(1, -1)
    y = reg.predict(x)

    print "Best Parameters: ", reg.best_params_
    print "Best Estimator:", reg.best_estimator_
    print "Grid Score:", reg.grid_scores_

    print "House: " + str(x)
    print "Predicted: " + str(y)

    #DataFrame of Client_Features
    #x = [11.95, 0.00, 18.100, 0, 0.6590, 5.6090, 90.00, 1.385, 24, 680.0, 20.20, 332.09, 12.1
    #pd.DataFrame(zip(boston.feature_names, x), columns = ['Features', 'Client_Features'])
```

1.0.9 Main

In [ ]:
```
def main():

    # Load data
    city_data = load_data()

    #DataFrame
    dataframe(city_data)

    # Explore the data
    explore_city_data(city_data)

    # Training/Test dataset split
    X_train, y_train, X_test, y_test = split_data(city_data)

    # Learning Curve Graphs
    max_depths = [1,2,3,4,5,6,7,8,9,10]
    for max_depth in max_depths:
        learning_curve(max_depth, X_train, y_train, X_test, y_test)

    # Model Complexity Graph
    model_complexity(X_train, y_train, X_test, y_test)

    #Tune and predict Model
    fit_predict_model(city_data)


if __name__ == "__main__":
    main()
```

```
RM         0.695360
ZN         0.360445
B          0.333461
DIS        0.249929
CHAS       0.175260
AGE       -0.376955
RAD       -0.381626
CRIM      -0.385832
NOX       -0.427321
TAX       -0.468536
INDUS     -0.483725
PTRATIO   -0.507787
LSTAT     -0.737663
Name: MEDV, dtype: float64
             CRIM        ZN     INDUS      CHAS       NOX        RM       AGE  \
CRIM     1.000000 -0.199458  0.404471 -0.055295  0.417521 -0.219940  0.350784
ZN      -0.199458  1.000000 -0.533828 -0.042697 -0.516604  0.311991 -0.569537
INDUS    0.404471 -0.533828  1.000000  0.062938  0.763651 -0.391676  0.644779
CHAS    -0.055295 -0.042697  0.062938  1.000000  0.091203  0.091251  0.086518
NOX      0.417521 -0.516604  0.763651  0.091203  1.000000 -0.302188  0.731470
RM      -0.219940  0.311991 -0.391676  0.091251 -0.302188  1.000000 -0.240265
AGE      0.350784 -0.569537  0.644779  0.086518  0.731470 -0.240265  1.000000
DIS     -0.377904  0.664408 -0.708027 -0.099176 -0.769230  0.205246 -0.747881
RAD      0.622029 -0.311948  0.595129 -0.007368  0.611441 -0.209847  0.456022
TAX      0.579564 -0.314563  0.720760 -0.035587  0.668023 -0.292048  0.506456
PTRATIO  0.288250 -0.391679  0.383248 -0.121515  0.188933 -0.355501  0.261515
B       -0.377365  0.175520 -0.356977  0.048788 -0.380051  0.128069 -0.273534
LSTAT    0.452220 -0.412995  0.603800 -0.053929  0.590879 -0.613808  0.602339
MEDV    -0.385832  0.360445 -0.483725  0.175260 -0.427321  0.695360 -0.376955

              DIS       RAD       TAX   PTRATIO         B     LSTAT      MEDV
CRIM    -0.377904  0.622029  0.579564  0.288250 -0.377365  0.452220 -0.385832
ZN       0.664408 -0.311948 -0.314563 -0.391679  0.175520 -0.412995  0.360445
INDUS   -0.708027  0.595129  0.720760  0.383248 -0.356977  0.603800 -0.483725
CHAS    -0.099176 -0.007368 -0.035587 -0.121515  0.048788 -0.053929  0.175260
NOX     -0.769230  0.611441  0.668023  0.188933 -0.380051  0.590879 -0.427321
RM       0.205246 -0.209847 -0.292048 -0.355501  0.128069 -0.613808  0.695360
AGE     -0.747881  0.456022  0.506456  0.261515 -0.273534  0.602339 -0.376955
DIS      1.000000 -0.494588 -0.534432 -0.232471  0.291512 -0.496996  0.249929
RAD     -0.494588  1.000000  0.910228  0.464741 -0.444413  0.488676 -0.381626
TAX     -0.534432  0.910228  1.000000  0.460853 -0.441808  0.543993 -0.468536
PTRATIO -0.232471  0.464741  0.460853  1.000000 -0.177383  0.374044 -0.507787
B        0.291512 -0.444413 -0.441808 -0.177383  1.000000 -0.366087  0.333461
LSTAT   -0.496996  0.488676  0.543993  0.374044 -0.366087  1.000000 -0.737663
MEDV     0.249929 -0.381626 -0.468536 -0.507787  0.333461 -0.737663  1.000000
Number of houses: 506
Number of features: 13
Minimum Housing Price: $5.00
Maximum Housing Price: $50.00
Mean Housing Price: $22.53
Median Housing Price: $21.20
Standard Deviation: $9.19
X_training: (354, 13)
X_test: (152, 13)
```

```
Decision Tree with Max Depth:
1

/Users/tracesmith/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:10: FutureWarning: sort is d
/Users/tracesmith/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:17: DeprecationWarning: usin
/Users/tracesmith/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:20: DeprecationWarning: usin

<matplotlib.figure.Figure at 0x11d47edd0>
```


```
Decision Tree with Max Depth:
2

<matplotlib.figure.Figure at 0x11c373ed0>
```

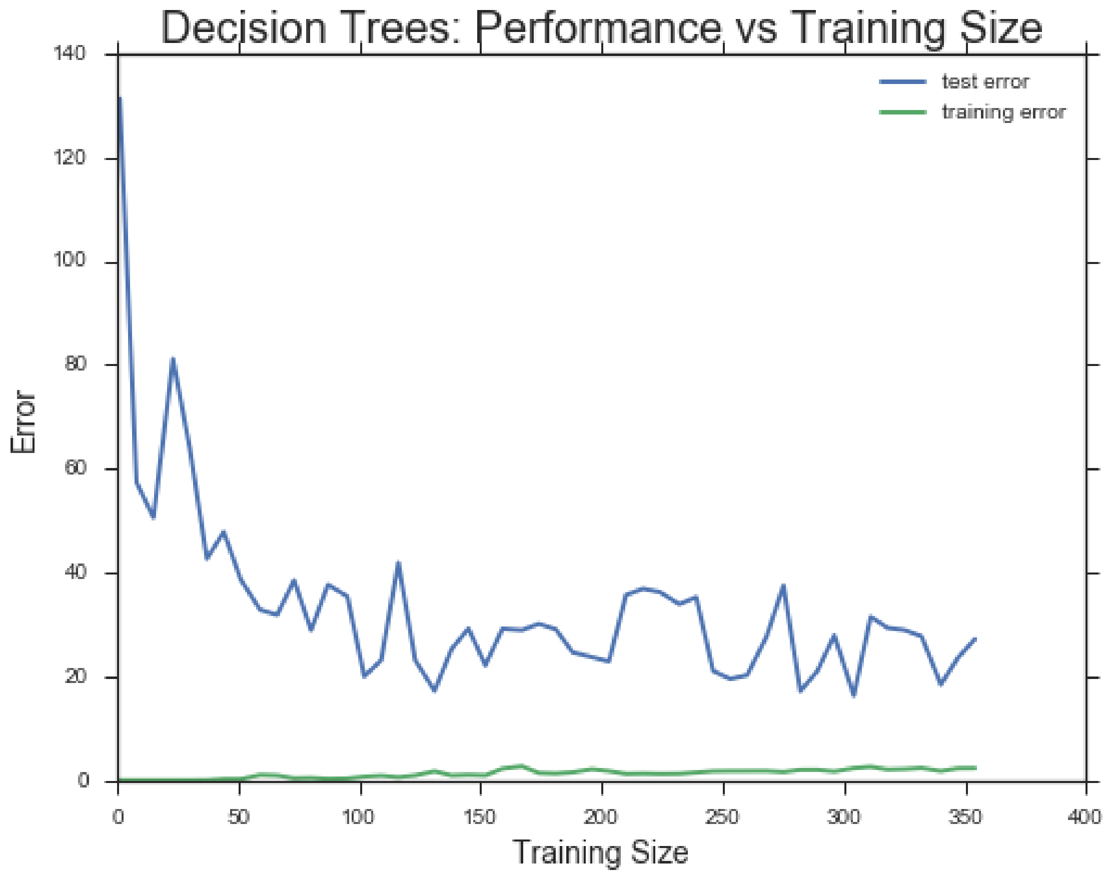

```
Decision Tree with Max Depth:
3

<matplotlib.figure.Figure at 0x116050f90>
```


```
Decision Tree with Max Depth:
4

<matplotlib.figure.Figure at 0x11c39fed0>
```


```
Decision Tree with Max Depth:
5

<matplotlib.figure.Figure at 0x11ce0fe90>
```


```
Decision Tree with Max Depth:
6

<matplotlib.figure.Figure at 0x11d277c90>
```


```
Decision Tree with Max Depth:
7

<matplotlib.figure.Figure at 0x11cc62f50>
```




```
Decision Tree with Max Depth:
8

<matplotlib.figure.Figure at 0x11ca49b90>
```


```
Decision Tree with Max Depth:
9

<matplotlib.figure.Figure at 0x11cf0be10>
```


```
Decision Tree with Max Depth:
10

<matplotlib.figure.Figure at 0x11b641f90>
```

Model Complexity:

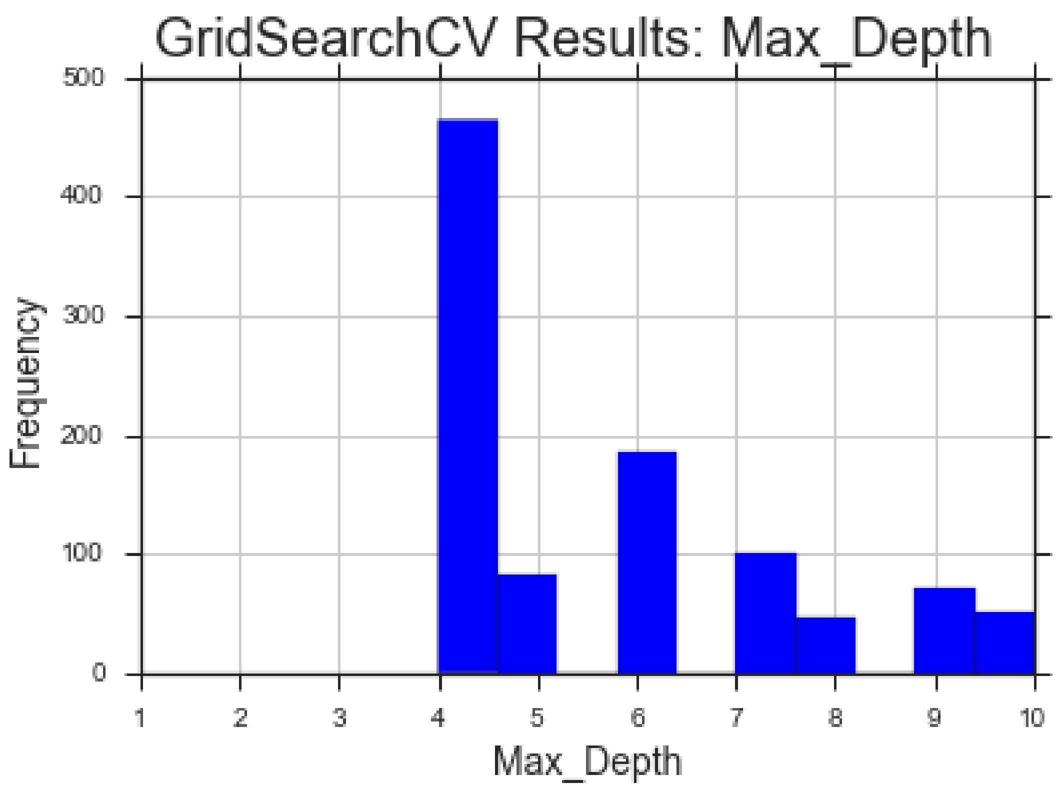
In [47]:
```
def iterate_fit_predict(city_data):
    """Find and tune the optimal model. Make a prediction on housing data."""

    # Get the features and labels from the Boston housing data
    X, y = city_data.data, city_data.target

    # Setup a Decision Tree Regressor
    regressor = DecisionTreeRegressor()

    mse_scoring = make_scorer(mean_squared_error, greater_is_better=False)

    parameters = {'max_depth':(1,2,3,4,5,6,7,8,9,10)}

    reg = GridSearchCV(regressor, parameters, scoring = mse_scoring, cv=3)

    # Fit the learner to the training data to obtain the best parameter set
    reg.fit(X, y)

    # Use the model to predict the output of a particular sample
    x = [11.95, 0.00, 18.100, 0, 0.6590, 5.6090, 90.00, 1.385, 24, 680.0, 20.20, 332.09, 12.13
    x = np.array(x)
    x = x.reshape(1, -1)
    y = reg.predict(x)

    return (reg.best_params_['max_depth'], y[0])
```

Iteration: Fit and Predict Model (GridSearchCV Results)

```
In [48]: Grid_Search_Results = []

         for i in range(1000):
             Grid_Search_Results.append(iterate_fit_predict(city_data))

         Grid_Search= np.asarray(Grid_Search_Results)
         Grid_Search_Results_Depth = np.asarray(Grid_Search.T[0], dtype=int)
         Grid_Search_Results_Price = np.asarray(Grid_Search.T[1], dtype=float)
```

Histogram: Maximum Depth (iterations = 10000)

```
In [49]: pl.hist(Grid_Search_Results_Depth, bins = 10, color = 'blue')
         pl.suptitle("GridSearchCV Results: Max_Depth", fontsize = 20)
         pl.xlabel("Max_Depth", fontsize = 14)
         pl.ylabel("Frequency", fontsize = 14)
         pl.xlim(1,10)
         pl.grid()
         pl.show()
```



Histogram: Predicted Housing Prices (iteraitons = 1000)

```
In [50]: pl.hist(Grid_Search_Results_Price, bins = 8, color = 'red')
         pl.suptitle("GridSearchCV Results: Housing Price", fontsize = 20)
         pl.xlabel("Predicted Housing Price", fontsize = 14)
         pl.ylabel("Frequency", fontsize = 14)
```

```
pl.xlim(18, 23.0)
pl.ylim(0, 500)
pl.grid()
pl.show()
```