Machine Learning Engineer Nanodegree

Model Evaluation & Validation

Project: Predicting Boston Housing Prices

Welcome to the first project of the Machine Learning Engineer Nanodegree! In this notebook, some template code has already been provided for you, and you will need to implement additional functionality to successfully complete this project. You will not need to modify the included code beyond what is requested. Sections that begin with **'Implementation'** in the header indicate that the following block of code will require additional functionality which you must provide. Instructions will be provided for each section and the specifics of the implementation are marked in the code block with a 'TODO' statement. Please be sure to read the instructions carefully!

In addition to implementing code, there will be questions that you must answer which relate to the project and your implementation. Each section where you will answer a question is preceded by a **'Question X'** header. Carefully read each question and provide thorough answers in the following text boxes that begin with **'Answer:'**. Your project submission will be evaluated based on your answers to each of the questions and the implementation you provide.

> **Note:** Code and Markdown cells can be executed using the **Shift + Enter** keyboard shortcut. In addition, Markdown cells can be edited by typically double-clicking the cell to enter edit mode.

Getting Started

In this project, you will evaluate the performance and predictive power of a model that has been trained and tested on data collected from homes in suburbs of Boston, Massachusetts. A model trained on this data that is seen as a *good fit* could then be used to make certain predictions about a home — in particular, its monetary value. This model would prove to be invaluable for someone like a real estate agent who could make use of such information on a daily basis.

The dataset for this project originates from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Housing). The Boston housing data was collected in 1978 and each of the 506 entries represent aggregated data about 14 features for homes from various suburbs in Boston, Massachusetts. For the purposes of this project, the following preprocessing steps have been made to the dataset:

- 16 data points have an `'MEDV'` value of 50.0. These data points likely contain **missing or censored values** and have been removed.
- 1 data point has an `'RM'` value of 8.78. This data point can be considered an **outlier** and has been removed.
- The features `'RM'`, `'LSTAT'`, `'PTRATIO'`, and `'MEDV'` are essential. The remaining **non-relevant features** have been excluded.
- The feature `'MEDV'` has been **multiplicatively scaled** to account for 35 years of market inflation.

Run the code cell below to load the Boston housing dataset, along with a few of the necessary Python libraries required for this project. You will know the dataset loaded successfully if the size of the dataset is reported.

In [1]:

```
%load_ext autoreload
%autoreload 2
```

In [2]:

```
# Import libraries necessary for this project
import numpy as np
import pandas as pd
#from sklearn.cross_validation import ShuffleSplit
# Using ShuffleSplit from sklearn.model_selection as it'll be depreciated cross_validation will be depreciated.
from sklearn.model_selection import ShuffleSplit

# Import supplementary visualizations code visuals.py
import visuals as vs

# Pretty display for notebooks
%matplotlib inline

# Load the Boston housing dataset
data = pd.read_csv('housing.csv')
prices = data['MEDV']
features = data.drop('MEDV', axis = 1)

# Success
print("Boston housing dataset has {} data points with {} variables each.".format(*data.shape))
```

```
Populating the interactive namespace from numpy and matplotlib
Boston housing dataset has 489 data points with 4 variables each.
```

Data Exploration

In this first section of this project, you will make a cursory investigation about the Boston housing data and provide your observations. Familiarizing yourself with the data through an explorative process is a fundamental practice to help you better understand and justify your results.

Since the main goal of this project is to construct a working model which has the capability of predicting the value of houses, we will need to separate the dataset into **features** and the **target variable**. The **features**, `'RM'` , `'LSTAT'` , and `'PTRATIO'` , give us quantitative information about each data point. The **target variable**, `'MEDV'` , will be the variable we seek to predict. These are stored in `features` and `prices` , respectively.

Implementation: Calculate Statistics

For your very first coding implementation, you will calculate descriptive statistics about the Boston housing prices. Since `numpy` has already been imported for you, use this library to perform the necessary calculations. These statistics will be extremely important later on to analyze various prediction results from the constructed model.

In the code cell below, you will need to implement the following:

- Calculate the minimum, maximum, mean, median, and standard deviation of `'MEDV'` , which is stored in `prices` .
  - Store each calculation in their respective variable.

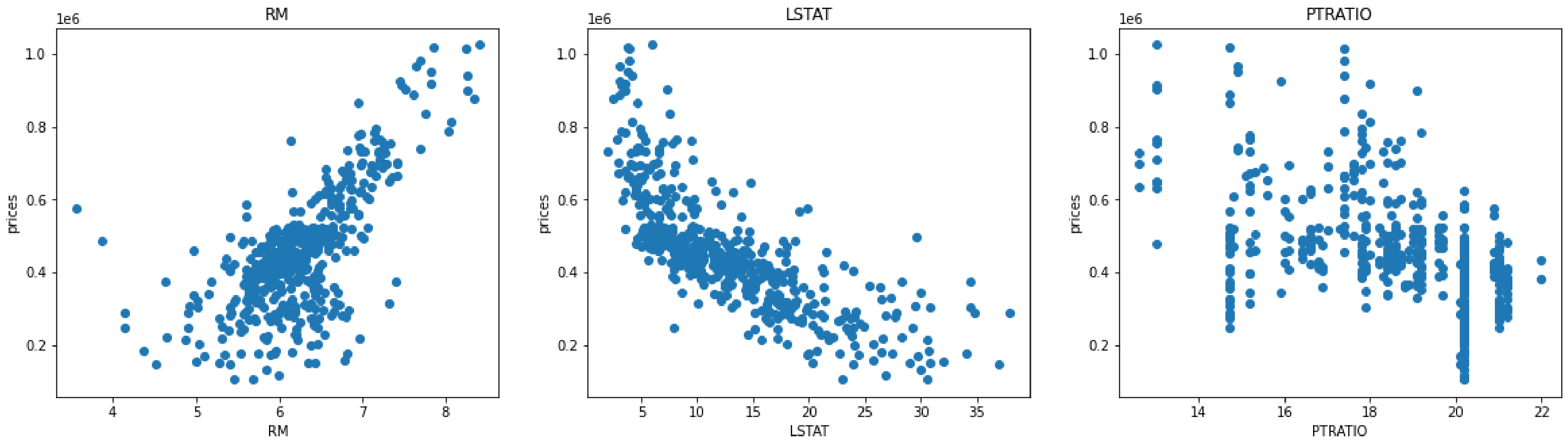
In [3]:

```
# TODO: Minimum price of the data
minimum_price = np.min(prices)

# TODO: Maximum price of the data
maximum_price = np.max(prices)

# TODO: Mean price of the data
mean_price = np.mean(prices)

# TODO: Median price of the data
median_price = np.median(prices)

# TODO: Standard deviation of prices of the data
std_price = np.std(prices)

# Show the calculated statistics
print("Statistics for Boston housing dataset:\n")
print("Minimum price: ${:,.2f}".format(minimum_price))
print("Maximum price: ${:,.2f}".format(maximum_price))
print("Mean price: ${:,.2f}".format(mean_price))
print("Median price ${:,.2f}".format(median_price))
print("Standard deviation of prices: ${:,.2f}".format(std_price))
```

```
Statistics for Boston housing dataset:

Minimum price: $105,000.00
Maximum price: $1,024,800.00
Mean price: $454,342.94
Median price $438,900.00
Standard deviation of prices: $165,171.13
```

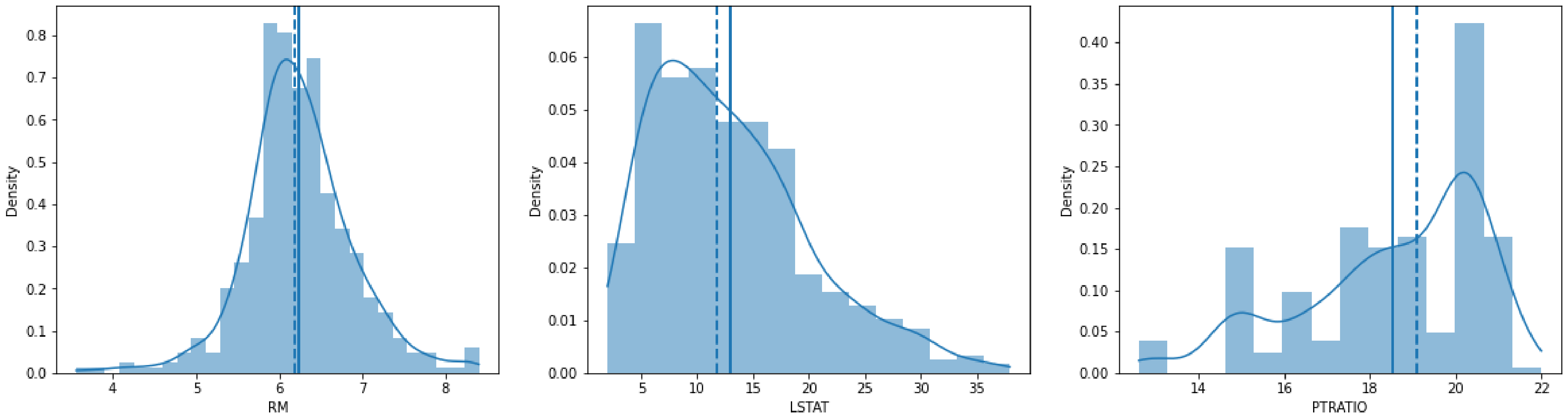
In [4]:

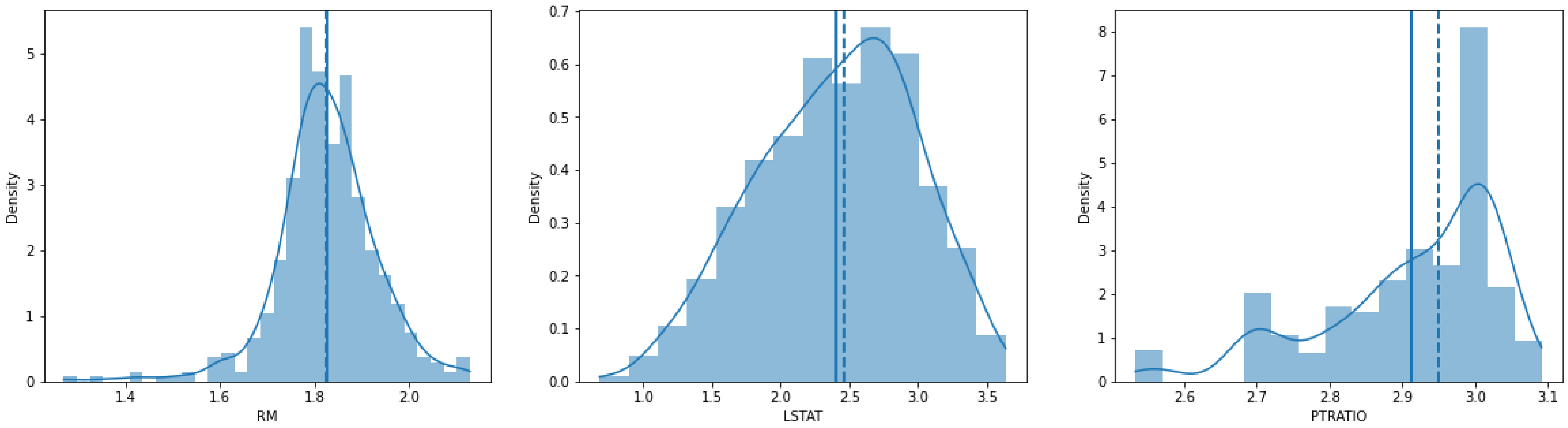
```
# Thanks to my reviewer for below nice code
import matplotlib.pyplot as plt
plt.figure(figsize=(20,5))
for i, col in enumerate(features.columns):
    plt.subplot(1, 3, i+1)
    plt.plot(features[col], prices, 'o')
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel('prices')
```



In [5]:

```python
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(20, 5))

# original data
for i, col in enumerate(features.columns):
    plt.subplot(131 + i)
    sns.histplot(data[col], kde=True, stat="density", linewidth=0)
    plt.axvline(data[col].mean(), linestyle='solid', linewidth=2)
    plt.axvline(data[col].median(), linestyle='dashed', linewidth=2)

# plot the log transformed data
plt.figure(figsize=(20, 5))
for i, col in enumerate(features.columns):
    plt.subplot(131 + i)
    sns.histplot(np.log(data[col]), kde=True, stat="density", linewidth=0)
    plt.axvline(np.log(data[col]).mean(), linestyle='solid', linewidth=2)
    plt.axvline(np.log(data[col]).median(), linestyle='dashed', linewidth=2)
```





Question 1 - Feature Observation

As a reminder, we are using three features from the Boston housing dataset: `'RM'`, `'LSTAT'`, and `'PTRATIO'`. For each data point (neighborhood):

- `'RM'` is the average number of rooms among homes in the neighborhood.
- `'LSTAT'` is the percentage of homeowners in the neighborhood considered "lower class" (working poor).
- `'PTRATIO'` is the ratio of students to teachers in primary and secondary schools in the neighborhood.

** Using your intuition, for each of the three features above, do you think that an increase in the value of that feature would lead to an **increase** in the value of `'MEDV'` or a **decrease** in the value of `'MEDV'`? Justify your answer for each.**

Hint: This problem can phrased using examples like below.

- Would you expect a home that has an `'RM'` value(number of rooms) of 6 be worth more or less than a home that has an `'RM'` value of 7?

- Would you expect a neighborhood that has an `'LSTAT'` value(percent of lower class workers) of 15 have home prices be worth more or less than a neighborhood that has an `'LSTAT'` value of 20?
- Would you expect a neighborhood that has an `'PTRATIO'` value(ratio of students to teachers) of 10 have home prices be worth more or less than a neighborhood that has an `'PTRATIO'` value of 15?

*_Answer:_ * Generally speaking, a bigger number of rooms means larger house, thus, higher value of 'MEDV'. Lower percentage of "poor" neighbors implies higher home price in the neighborhood, higher value of 'MEDV'. The smaller ratio of students to teachers means the schools have enough money to hire more teachers and that usually happen in rich neighborhood and thus higher 'MEDV'.

Developing a Model

In this second section of the project, you will develop the tools and techniques necessary for a model to make a prediction. Being able to make accurate evaluations of each model's performance through the use of these tools and techniques helps to greatly reinforce the confidence in your predictions.

Implementation: Define a Performance Metric

It is difficult to measure the quality of a given model without quantifying its performance over training and testing. This is typically done using some type of performance metric, whether it is through calculating some type of error, the goodness of fit, or some other useful measurement. For this project, you will be calculating the *coefficient of determination* (http://stattrek.com/statistics/dictionary.aspx?definition=coefficient_of_determination), R^2, to quantify your model's performance. The coefficient of determination for a model is a useful statistic in regression analysis, as it often describes how "good" that model is at making predictions.

The values for R^2 range from 0 to 1, which captures the percentage of squared correlation between the predicted and actual values of the **target variable**. A model with an R^2 of 0 is no better than a model that always predicts the *mean* of the target variable, whereas a model with an R^2 of 1 perfectly predicts the target variable. Any value between 0 and 1 indicates what percentage of the target variable, using this model, can be explained by the **features**. *A model can be given a negative R^2 as well, which indicates that the model is **arbitrarily worse** than one that always predicts the mean of the target variable.*

For the `performance_metric` function in the code cell below, you will need to implement the following:

- Use `r2_score` from `sklearn.metrics` to perform a performance calculation between `y_true` and `y_predict`.
- Assign the performance score to the `score` variable.

In [6]:

```python
# TODO: Import 'r2_score'
from sklearn.metrics import r2_score

def performance_metric(y_true, y_predict):
    """ Calculates and returns the performance score between
        true and predicted values based on the metric chosen. """

    # TODO: Calculate the performance score between 'y_true' and 'y_predict'
    score = r2_score(y_true, y_predict)

    # Return the score
    return score
```

Question 2 - Goodness of Fit

Assume that a dataset contains five data points and a model made the following predictions for the target variable:

True Value	Prediction
3.0	2.5
-0.5	0.0
2.0	2.1
7.0	7.8
4.2	5.3

Run the code cell below to use the `performance_metric` function and calculate this model's coefficient of determination.

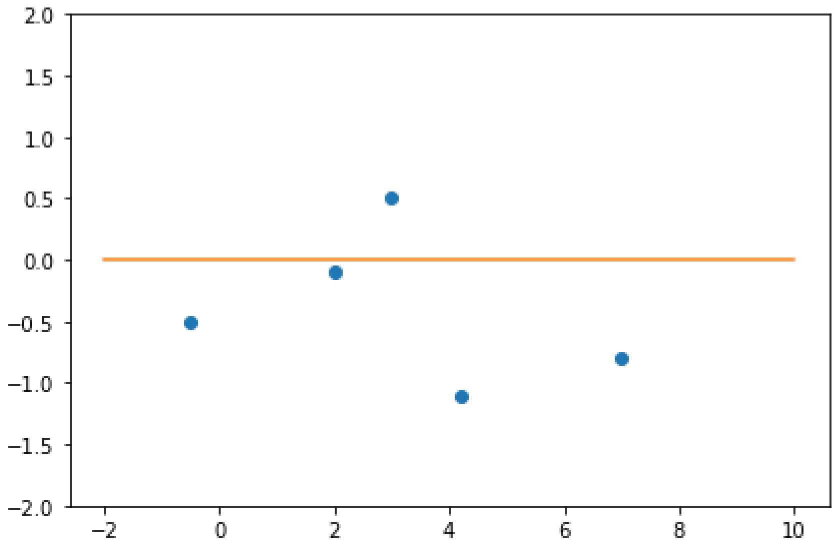
In [7]:

```python
# Calculate the performance of this model
score = performance_metric([3, -0.5, 2, 7, 4.2], [2.5, 0.0, 2.1, 7.8, 5.3])
print("Model has a coefficient of determination, R^2, of {:.3f}.".format(score))
```

```
Model has a coefficient of determination, R^2, of 0.923.
```

In [8]:

```python
# Check the residual via plot
import matplotlib.pyplot as plt
trueValues = [3, -0.5, 2, 7, 4.2]
predictions = [2.5, 0.0, 2.1, 7.8, 5.3]
residuals = [i - j for i, j in zip(trueValues, predictions)]
plt.plot(trueValues, residuals, 'o',[-2,10], [0,0], '-')
plt.ylim(-2, 2)
plt.tight_layout()
```



- Would you consider this model to have successfully captured the variation of the target variable?
- Why or why not?

** Hint: ** The R2 score is the proportion of the variance in the dependent variable that is predictable from the independent variable. In other words:

- R2 score of 0 means that the dependent variable cannot be predicted from the independent variable.
- R2 score of 1 means the dependent variable can be predicted from the independent variable.
- R2 score between 0 and 1 indicates the extent to which the dependent variable is predictable. An
- R2 score of 0.40 means that 40 percent of the variance in Y is predictable from X.

Answer: Yes, in my opinion the model explained 92% of variance, and that's a good sign. It's quite close to the perfect 100%.

Implementation: Shuffle and Split Data

Your next implementation requires that you take the Boston housing dataset and split the data into training and testing subsets. Typically, the data is also shuffled into a random order when creating the training and testing subsets to remove any bias in the ordering of the dataset.

For the code cell below, you will need to implement the following:

- Use `train_test_split` from `sklearn.cross_validation` to shuffle and split the `features` and `prices` data into training and testing sets.
  - Split the data into 80% training and 20% testing.
  - Set the `random_state` for `train_test_split` to a value of your choice. This ensures results are consistent.
- Assign the train and testing splits to `X_train` , `X_test` , `y_train` , and `y_test` .

In [9]:

```python
# TODO: Import 'train_test_split'
from sklearn.model_selection import train_test_split

# TODO: Shuffle and split the data into training and testing subsets
X_train, X_test, y_train, y_test = train_test_split(features, prices, test_size=0.2,
                                                    random_state=0)

# Success
print("Training and testing split was successful.")
```

```
Training and testing split was successful.
```

Question 3 - Training and Testing

- What is the benefit to splitting a dataset into some ratio of training and testing subsets for a learning algorithm?

Hint: Think about how overfitting or underfitting is contingent upon how splits on data is done.

*Answer: * If we don't split the data into training and testing, we won't have a baseline of how good/bad our model performs, we also have no idea when to stop the training of the model as again we have no idea how good/bad it's performing. In another way, we have no idea whether the model is just right, underfit or overfit.

Analyzing Model Performance

In this third section of the project, you'll take a look at several models' learning and testing performances on various subsets of training data. Additionally, you'll investigate one particular algorithm with an increasing `'max_depth'` parameter on the full training set to observe how model complexity affects performance. Graphing your model's performance based on varying criteria can be beneficial in the analysis process, such as visualizing behavior that may not have been apparent from the results alone.

Learning Curves

The following code cell produces four graphs for a decision tree model with different maximum depths. Each graph visualizes the learning curves of the model for both training and testing as the size of the training set is increased. Note that the shaded region of a learning curve denotes the uncertainty of that curve (measured as the standard deviation). The model is scored on both the training and testing sets using R^2, the coefficient of determination.

Run the code cell below and use these graphs to answer the following question.

NOTE by Victor

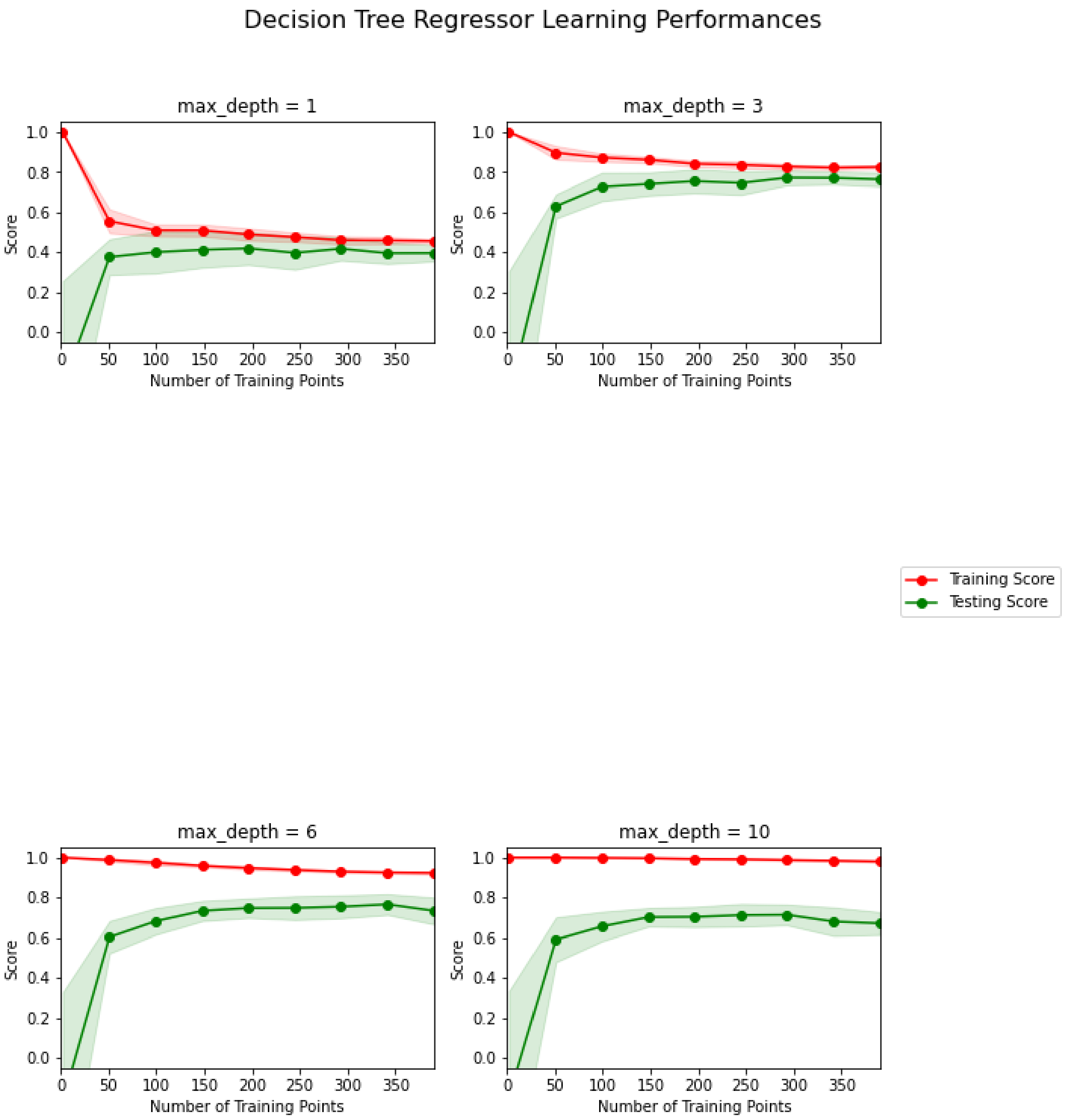
To avoid below error, `train_sizes` in function `ModelLearning(X, y)` value starts from 2, instead of 1 previously `UndefinedMetricWarning: R^2 score is not well-defined with less than two samples.`

```
train_sizes = np.rint(np.linspace(2, X.shape[0]*0.8 - 1, 9)).astype(int)
```

In [10]:

```
# Produce learning curves for varying training set sizes and maximum depths
vs.ModelLearning(features, prices)
```

```
D:\GoogleDrive\Study\Machine Learning Engineer Nanodegree - Udacity\Projects\MLND-Projects\projects\boston_housing\visu
als.py:69: UserWarning: Matplotlib is currently using module://ipykernel.pylab.backend_inline, which is a non-GUI backe
nd, so cannot show the figure.
  fig.show()
```



Question 4 - Learning the Data

- Choose one of the graphs above and state the maximum depth for the model.

- What happens to the score of the training curve as more training points are added? What about the testing curve?
- Would having more training points benefit the model?

Hint: Are the learning curves converging to particular scores? Generally speaking, the more data you have, the better. But if your training and testing curves are converging with a score above your benchmark threshold, would this be necessary? Think about the pros and cons of adding more training points based on if the training and testing curves are converging.

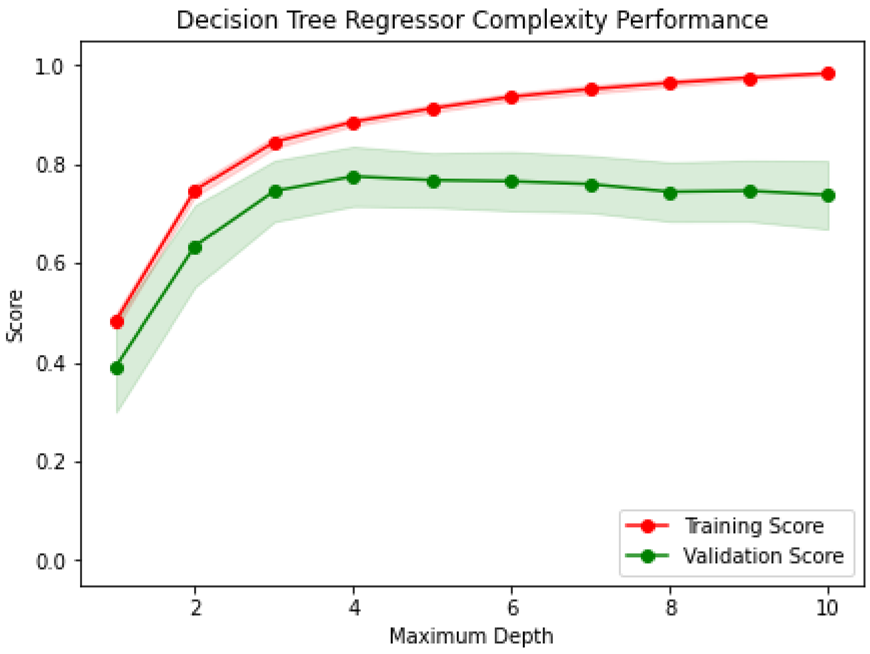
**Answer:* * The fourth model has maximum depth of 10. When more data points fitted in, the training score descreased very little. Testing score keeps increasing until it converged after 50 training points. There is very little increament thereafter and there is a huge gap between training and testing score, which implies the model is overfitting. For all the models, testing score started to converge from 50 training points, not much improvement until almost 400 training points. So I don't think it'll benefit the model with more training points.

Complexity Curves

The following code cell produces a graph for a decision tree model that has been trained and validated on the training data using different maximum depths. The graph produces two complexity curves — one for training and one for validation. Similar to the **learning curves**, the shaded regions of both the complexity curves denote the uncertainty in those curves, and the model is scored on both the training and validation sets using the `performance_metric` function.

** Run the code cell below and use this graph to answer the following two questions Q5 and Q6. **

```
In [11]:  vs.ModelComplexity(X_train, y_train)
```



Question 5 - Bias-Variance Tradeoff

- When the model is trained with a maximum depth of 1, does the model suffer from high bias or from high variance?
- How about when the model is trained with a maximum depth of 10? What visual cues in the graph justify your conclusions?

Hint: High bias is a sign of underfitting(model is not complex enough to pick up the nuances in the data) and high variance is a sign of overfitting(model is by-hearting the data and cannot generalize well). Think about which model(depth 1 or 10) aligns with which part of the tradeoff.

**Answer:* * When the maximum depth is 1, the model suffers from high bias because it's too generalized and can not provide accurate predications. Both the training and testing scores are just around 50% which are low.

When the max depth is 10, it suffers from high variance as the training score is almost 100% but validation score is around 70%. It cannot generalize to predict well on 'unseen' data.

Question 6 - Best-Guess Optimal Model

- Which maximum depth do you think results in a model that best generalizes to unseen data?
- What intuition lead you to this answer?

** Hint: ** Look at the graph above Question 5 and see where the validation scores lie for the various depths that have been assigned to the model. Does it get better with increased depth? At what point do we get our best validation score without overcomplicating our model? And remember, Occams Razor states "Among competing hypotheses, the one with the fewest assumptions should be selected."

**Answer:* * Depth of either 4 or 5 should be good. Based on the graph from "ModelComplexity", the any depth below them will produce lower validation score, which is underfitting. Any depth above 5 will have better training score, but almost no help on validation score. It will only make the model more complicated, which is overfitting.

Evaluating Model Performance

In this final section of the project, you will construct a model and make a prediction on the client's feature set using an optimized model from `fit_model`.

_

Question 7 - Grid Search

- What is the grid search technique?
- How it can be applied to optimize a learning algorithm?

** Hint: ** When explaining the Grid Search technique, be sure to touch upon why it is used, what the 'grid' entails and what the end goal of this method is. To solidify your answer, you can also give an example of a parameter in a model that can be optimized using this approach.

**Answer:* * Grid search is a technique to use different parameters to train model(s) and test against the testing dataset to find the best model.

Instead of trying each and every combination of parameter manually, we can put the different parameters in the grid search, it helps find the best combination of parameters fast. For example, in the previous section we can also use grid search to try different max depth to find the optimized value.

Question 8 - Cross-Validation

- What is the k-fold cross-validation training technique?
- What benefit does this technique provide for grid search when optimizing a model?

Hint: When explaining the k-fold cross validation technique, be sure to touch upon what 'k' is, how the dataset is split into different parts for training and testing and the number of times it is run based on the 'k' value.

When thinking about how k-fold cross validation helps grid search, think about the main drawbacks of grid search which are hinged upon **using a particular subset of data for training or testing** and how k-fold cv could help alleviate that. You can refer to the docs (http://scikit-learn.org/stable/modules/cross_validation.html#cross-validation) for your answer.

**Answer:* * K-Fold cross-validation split the dataset into K portions/folds. At each training and validation, one fold will be reserved for validation while the other K-1 folds are used for training. This process will be repeated for K times. In this way, each and every fold of datasets will be used for both training and validation. The final estimation will be averaged out from the K results, this will provide a better estimation of the model performance.

The K-Fold provides a more 'accurate' estimation of the model. If we test the model against only one validation dataset, how well the model performs really depends on how 'lucky' it is. It can be a coincidence that the model performs better than average on the specific validation dataset despite we randomly chose the data points to form the dataset. Using K-Fold will address this problem as each and every data point will have a chance to test againt the model. The estimation of the model will be more proven.

Implementation: Fitting a Model

Your final implementation requires that you bring everything together and train a model using the **decision tree algorithm**. To ensure that you are producing an optimized model, you will train the model using the grid search technique to optimize the `'max_depth'` parameter for the decision tree. The `'max_depth'` parameter can be thought of as how many questions the decision tree algorithm is allowed to ask about the data before making a prediction. Decision trees are part of a class of algorithms called *supervised learning algorithms*.

In addition, you will find your implementation is using `ShuffleSplit()` for an alternative form of cross-validation (see the `'cv_sets'` variable). While it is not the K-Fold cross-validation technique you describe in **Question 8**, this type of cross-validation technique is just as useful!. The `ShuffleSplit()` implementation below will create 10 (`'n_splits'`) shuffled sets, and for each shuffle, 20% (`'test_size'`) of the data will be used as the *validation set*. While you're working on your implementation, think about the contrasts and similarities it has to the K-fold cross-validation technique.

Please note that ShuffleSplit has different parameters in scikit-learn versions 0.17 and 0.18. For the `fit_model` function in the code cell below, you will need to implement the following:

- Use `DecisionTreeRegressor` (http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html) from `sklearn.tree` to create a decision tree regressor object.
    - Assign this object to the `'regressor'` variable.
- Create a dictionary for `'max_depth'` with the values from 1 to 10, and assign this to the `'params'` variable.
- Use `make_scorer` (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.make_scorer.html) from `sklearn.metrics` to create a scoring function object.
    - Pass the `performance_metric` function as a parameter to the object.
    - Assign this scoring function to the `'scoring_fnc'` variable.
- Use `GridSearchCV` (http://scikit-learn.org/0.17/modules/generated/sklearn.grid_search.GridSearchCV.html) from `sklearn.grid_search` to create a grid search object.
    - Pass the variables `'regressor'` , `'params'` , `'scoring_fnc'` , and `'cv_sets'` as parameters to the object.
    - Assign the `GridSearchCV` object to the `'grid'` variable.

In [12]:

```python
# TODO: Import 'make_scorer', 'DecisionTreeRegressor', and 'GridSearchCV'
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import make_scorer
#from sklearn.grid_search import GridSearchCV
# skleanr.grid_search will be deprecaited, use sklearn.model_selection
from sklearn.model_selection import GridSearchCV

def fit_model(X, y):
    """ Performs grid search over the 'max_depth' parameter for a
        decision tree regressor trained on the input data [X, y]. """

    # Create cross-validation sets from the training data
    # sklearn version 0.18: ShuffleSplit(n_splits=10, test_size=0.1, train_size=None, random_state=None)
    # sklearn versiin 0.17: ShuffleSplit(n, n_iter=10, test_size=0.1, train_size=None, random_state=None)

    # Below code is based on version 0.19
    rs = ShuffleSplit(n_splits=10, test_size = 0.20, random_state = 0)
    cv_sets = rs.split(X)

    # TODO: Create a decision tree regressor object
    regressor = DecisionTreeRegressor(random_state=0)

    # TODO: Create a dictionary for the parameter 'max_depth' with a range from 1 to 10
    params = {'max_depth':range(1,11)}

    # TODO: Transform 'performance_metric' into a scoring function using 'make_scorer'
    scoring_fnc = make_scorer(performance_metric)

    # TODO: Create the grid search cv object --> GridSearchCV()
    # Make sure to include the right parameters in the object:
    # (estimator, param_grid, scoring, cv) which have values 'regressor', 'params', 'scoring_fnc', and 'cv_sets' res

    # Must specify the cv in the sklearn verison 0.19, cv can be an int, cross-validation generator, or iterable,
    # so cv=cv_sets is also correct for below code.
    grid = GridSearchCV(regressor, params, scoring=scoring_fnc, cv=rs)

    # Fit the grid search object to the data to compute the optimal model
    grid = grid.fit(X, y)

    # Return the optimal model after fitting the data
    return grid.best_estimator_
```

Making Predictions

Once a model has been trained on a given set of data, it can now be used to make predictions on new sets of input data. In the case of a *decision tree regressor*, the model has learned *what the best questions to ask about the input data are*, and can respond with a prediction for the **target variable**. You can use these predictions to gain information about data where the value of the target variable is unknown — such as data the model was not trained on.

Question 9 - Optimal Model

- What maximum depth does the optimal model have? How does this result compare to your guess in **Question 6**?

Run the code block below to fit the decision tree regressor to the training data and produce an optimal model.

In [13]:

```python
# Fit the training data to the model using grid search
reg = fit_model(X_train, y_train)

# Produce the value for 'max_depth'
print("Parameter 'max_depth' is {} for the optimal model.".format(reg.get_params()['max_depth']))
```

```
Parameter 'max_depth' is 4 for the optimal model.
```

** Hint: ** The answer comes from the output of the code snipped above.

**Answer:* * I tailies with my guess in Question 6, the optimal model has max_depth of 4.

If the ShuffleSplit is not used in the GridSearchCV, the returned result will be 5.

Question 10 - Predicting Selling Prices

Imagine that you were a real estate agent in the Boston area looking to use this model to help price homes owned by your clients that they wish to sell. You have collected the following information from three of your clients:

Feature	Client 1	Client 2	Client 3
Total number of rooms in home	5 rooms	4 rooms	8 rooms
Neighborhood poverty level (as %)	17%	32%	3%

Feature	Client 1	Client 2	Client 3
Student-teacher ratio of nearby schools	15-to-1	22-to-1	12-to-1

- What price would you recommend each client sell his/her home at?
- Do these prices seem reasonable given the values for the respective features?

Hint: Use the statistics you calculated in the **Data Exploration** section to help justify your response. Of the three clients, client 3 has has the biggest house, in the best public school neighborhood with the lowest poverty level; while client 2 has the smallest house, in a neighborhood with a relatively high poverty rate and not the best public schools.

Run the code block below to have your optimized model make predictions for each client's home.

In [14]:

```python
# Produce a matrix for client data
client_data = [[5, 17, 15], # Client 1
               [4, 32, 22], # Client 2
               [8, 3, 12]]  # Client 3

# Show predictions
for i, price in enumerate(reg.predict(client_data)):
    print("Predicted selling price for Client {}'s home: ${:,.2f}".format(i+1, price))
```

```
Predicted selling price for Client 1's home: $391,183.33
Predicted selling price for Client 2's home: $189,123.53
Predicted selling price for Client 3's home: $942,666.67
```

In [15]:

```python
print("Accuracy on the testing dataset for the best model is", reg.score(X_test, y_test))
data.describe()
```

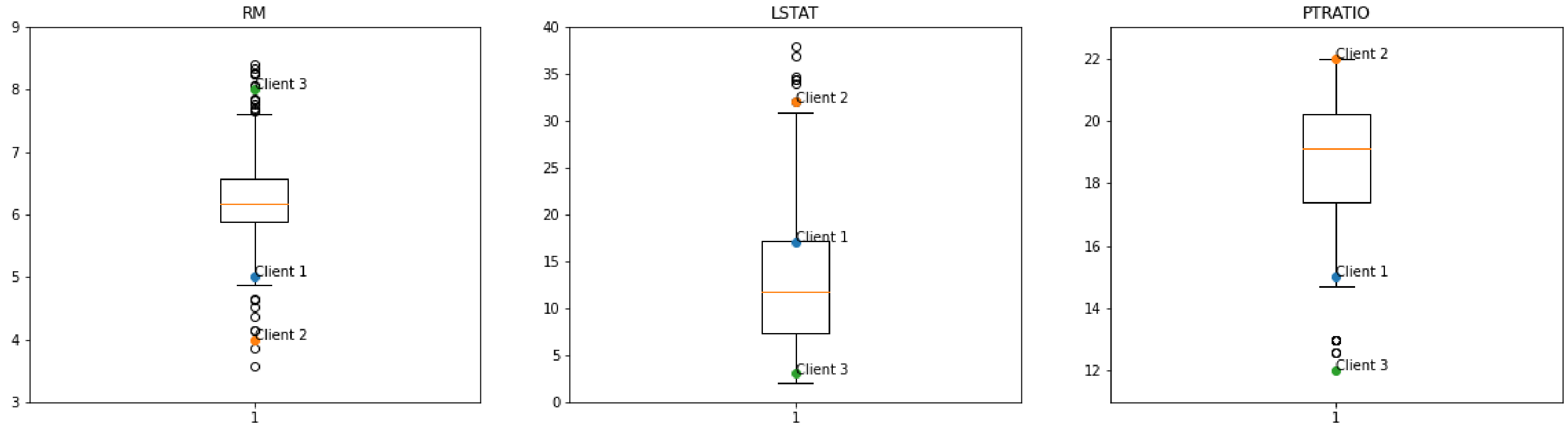
```
Accuracy on the testing dataset for the best model is 0.7731335857130197
```

Out[15]:

	RM	LSTAT	PTRATIO	MEDV
count	489.000000	489.000000	489.000000	4.890000e+02
mean	6.240288	12.939632	18.516564	4.543429e+05
std	0.643650	7.081990	2.111268	1.653403e+05
min	3.561000	1.980000	12.600000	1.050000e+05
25%	5.880000	7.370000	17.400000	3.507000e+05
50%	6.185000	11.690000	19.100000	4.389000e+05
75%	6.575000	17.120000	20.200000	5.187000e+05
max	8.398000	37.970000	22.000000	1.024800e+06

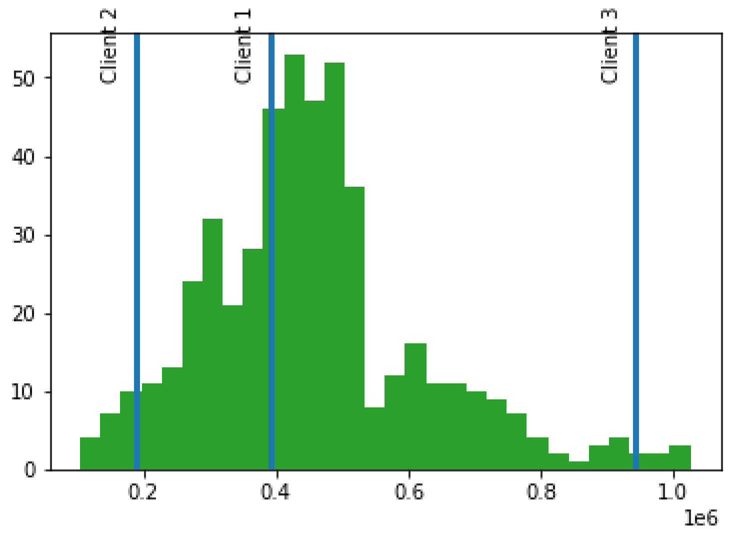
In [16]:

```python
# Thanks to my reviewer for below code
import matplotlib.pyplot as plt
plt.figure(figsize=(20, 5))
y_ax = [[3,9],[0,40],[11,23]]
for i, col in enumerate(features.columns):
    plt.subplot(1, 3, i+1)
    plt.boxplot(data[col])
    plt.title(col)
    for j in range(3):
        plt.plot(1, client_data[j][i], marker="o")
        plt.annotate('Client '+str(j+1), xy=(1,client_data[j][i]))
        plt.ylim(y_ax[i])
```



In [17]:

```
import matplotlib.pyplot as plt
for i,price in enumerate(reg.predict(client_data)):
    plt.hist(prices, bins = 30)
    plt.axvline(price, lw = 3)
    plt.text(price-50000, 50, 'Client '+str(i+1), rotation=90)
```



**Answer: ** The data.describe() output above table of statistics. Regardless of the other affecting factors in real work, I would say all the predications are reasonale, maybe client 2 needs to lower down the price a little.

The room number of client 1 is below 25% quantile, neighbourhood poverty level is around 75% quantitle. These two will drive the predicted price close to 25% of the price, which is $350K. The ratio of student to teach is below 25$350K.

Client 2's room number and neighborhood poverty level are in the worst 25% quantile. Student to teacher ratio is maximum. So in my opinion, the price should be slightly closer to the minimum house price of $105K.

Client 3 has a remarkable profile, all the features are in the best 25% quantile and close to the best feature value, the $942K looks a reasonable value.

Sensitivity

An optimal model is not necessarily a robust model. Sometimes, a model is either too complex or too simple to sufficiently generalize to new data. Sometimes, a model could use a learning algorithm that is not appropriate for the structure of the data given. Other times, the data itself could be too noisy or contain too few samples to allow a model to adequately capture the target variable — i.e., the model is underfitted.

Run the code cell below to run the `fit_model` function ten times with different training and testing sets to see how the prediction for a specific client changes with respect to the data it's trained on.

In [18]:

```
vs.PredictTrials(features, prices, fit_model, client_data)
```

```
Trial 1: $391,183.33
Trial 2: $424,935.00
Trial 3: $415,800.00
Trial 4: $420,622.22
Trial 5: $418,377.27
Trial 6: $411,931.58
Trial 7: $399,663.16
Trial 8: $407,232.00
Trial 9: $351,577.61
Trial 10: $413,700.00

Range in prices: $73,357.39
```

Question 11 - Applicability

- In a few sentences, discuss whether the constructed model should or should not be used in a real-world setting.

Hint: Take a look at the range in prices as calculated in the code snippet above. Some questions to answering:

- How relevant today is data that was collected from 1978? How important is inflation?
- Are the features present in the data sufficient to describe a home? Do you think factors like quality of apppliances in the home, square feet of the plot area, presence of pool or not etc should factor in?
- Is the model robust enough to make consistent predictions?
- Would data collected in an urban city like Boston be applicable in a rural city?
- Is it fair to judge the price of an individual home based on the characteristics of the entire neighborhood?

*_Answer:_ * The model shouldn't be used for a real-world setting.

Despite we can use other methods like ensembling to avoid the problem in the sensitivity section to make the model more 'robost', the main issue is the data didn't cath the impact from time series. The data is from year 1978, that time we don't havce Google, Facebook, maybe we just start to have Microsoft, needlessly to say, we never consider the inflation due to time.

In addition, the features only have data of room numbers but not area, no data on the facilities of the apartment, condition of the house, whether the house is nicely renovated, etcs and etcs. All in all, it's fun to explore, to have a reference from the model, but cannot rely much on the model, at least from the data we've got in this project.

> **Note**: Once you have completed all of the code implementations and successfully answered each question above, you may finalize your work by exporting the iPython Notebook as an HTML document. You can do this by using the menu above and navigating to **File -> Download as -> HTML (.html)**. Include the finished document along with this notebook as your submission.