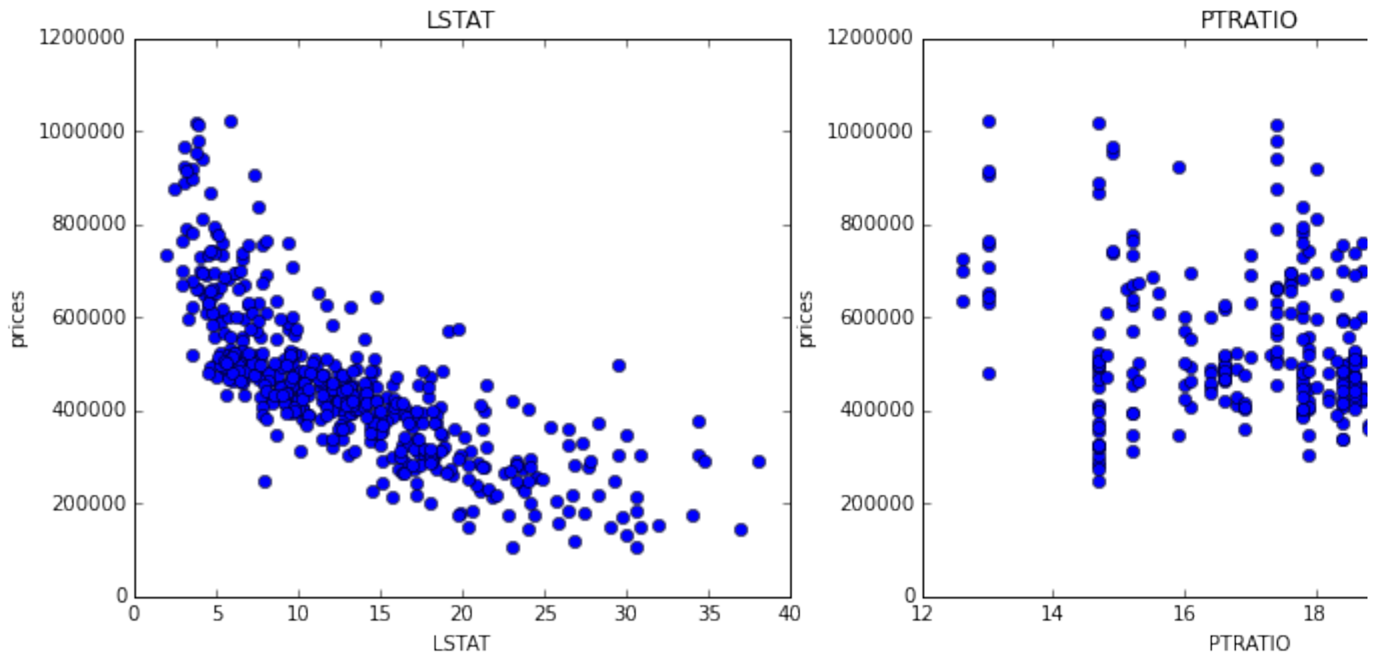
Data Exploration

All requested statistics for the Boston Housing dataset are accurately calculated. Student correctly leverages NumPy functionality to obtain these results.
Good job utilizing the power of Numpy!! Always important to get a basic understanding of our dataset before diving in. As we now know that a "dumb" classifier that only predicts the mean would predict $454,342.94 for all houses.
Student correctly justifies how each feature correlates with an increase or decrease in the target variable.
Nice observations for the features in this dataset. As we can confirm these ideas by plotting each feature vs MEDV housing prices.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(20, 5))

for i, col in enumerate(features.columns):

    plt.subplot(1, 3, i)

    plt.plot(data[col], prices, 'o')

    plt.title(col)

    plt.xlabel(col)

    plt.ylabel('prices')
```



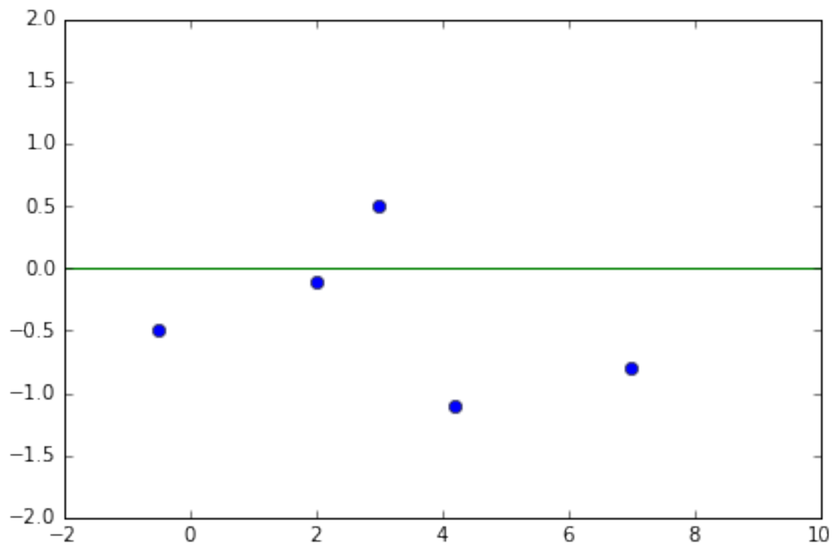
Developing a Model

Student correctly identifies whether the hypothetical model successfully captures the variation of the target variable based on the model's R^2 score.
The performance metric is correctly implemented in code.
Based on your submitted comment of

"I will certainly do some more reading on how to interpret the results of my models for future projects."

Another interesting thing to check out for regression problems would be a residual plot. A residual plot is a graph that shows the residuals on the vertical axis and the independent variable on the horizontal axis. If the points in a residual plot are randomly dispersed around the horizontal axis, a linear regression model is appropriate for the data; otherwise, a non-linear model is more appropriate.

```
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()
```



Student provides a valid reason for why a dataset is split into training and testing subsets for a model. Training and testing split is correctly implemented in code.
Great reasons! We need a way to determine how well our model is doing! As we can get a good estimate of our generalization accuracy on this testing dataset. Since our main goal is to accurately predict on new unseen data. Also that we can try and protect against overfitting with this independent dataset.

If you would like to learn some more ideas in why we need to split our data and what to avoid, such as data leakage, check out these lectures

- https://classroom.udacity.com/courses/ud730/lessons/6370362152/concepts/63798118300923
- https://classroom.udacity.com/courses/ud730/lessons/6370362152/concepts/63798118310923

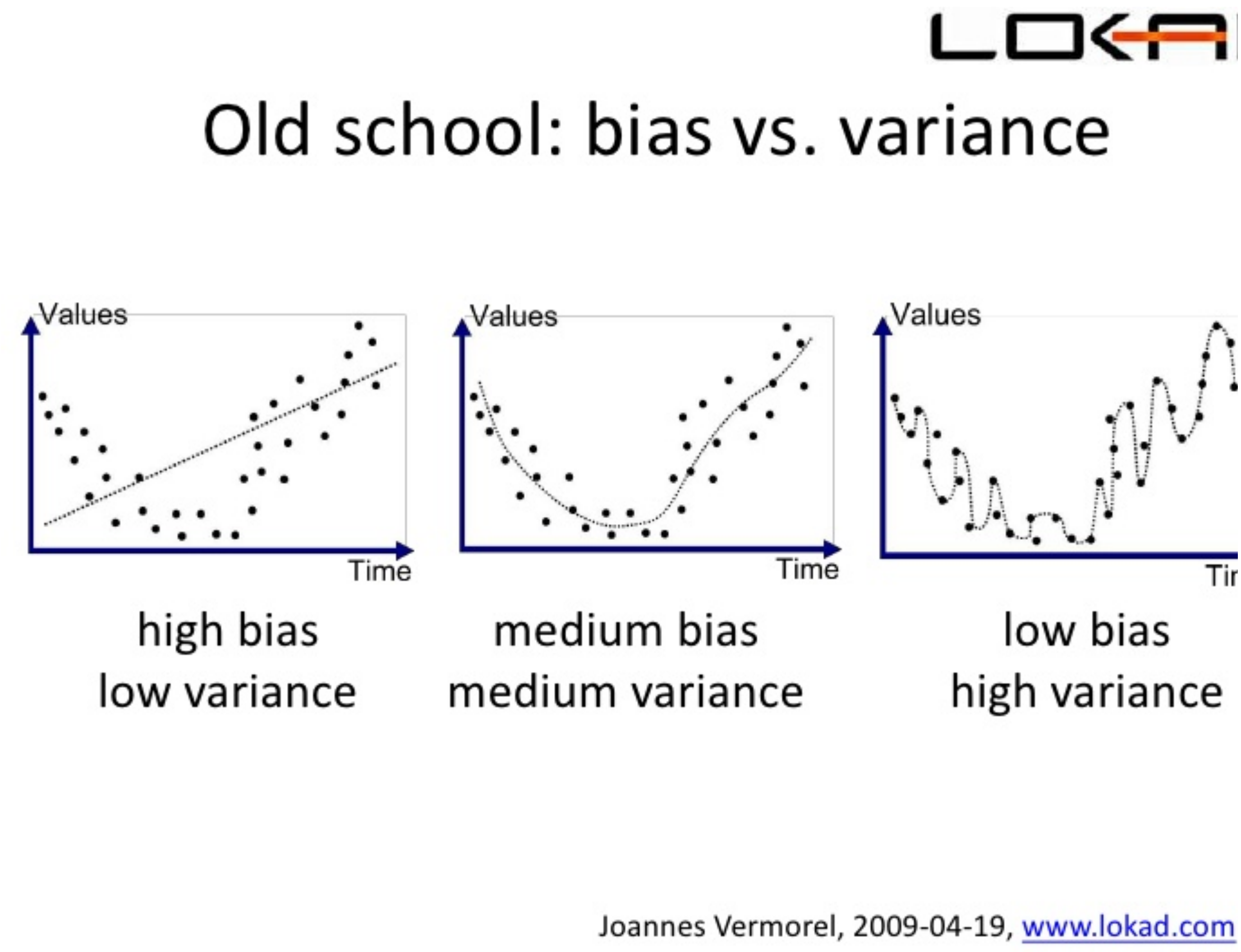
Analyzing Model Performance

Student correctly identifies the trend of both the training and testing curves from the graph as more training points are added. Discussion is made as to whether additional training points would benefit the model.
Would recommend also mentioning that in the initial phases, the training score decreases and testing score increases. Which makes sense, since with little amounts of the data we simply memorize the training data(no generalization), then when we receive more and more data points we can't simply memorize the training data and we start to generalize better(higher testing accuracy).

"In all graphs (at all max depths) the accuracy of the model flattens out beyond 200 data points. This indicates that more data may not actually benefit the model"
Correct! As in the beginning it is beneficial, but at the end if we look at the testing curve here, we can clearly see that it has converged to its optimal score, so more data is not necessary.

Also note that in practice collecting more data can often be time consuming and/or expensive, so when we can avoid having to collect more data the better. Therefore sometimes receiving very minor increases in performance is not beneficial, which is why plotting these curves can be very critical at times.
Student correctly identifies whether the model at a max depth of 1 and a max depth of 10 suffer from either high bias or high variance, with justification using the complexity curves graph.
Nice justification here! As the low training score is what truly depicts high bias. You clearly understand the bias/variance tradeoff.

- As a max_depth of 1 suffers from high bias, visually this is due to the low training score(also note that it has low variance since the scores are close together). As this model is not complex enough to learn the structure of the data
- And a max_depth of 10 suffers from high variance, since we see a large gap between the training and validation scores, as we are basically just memorizing our training data and will not generalize well to new unseen data

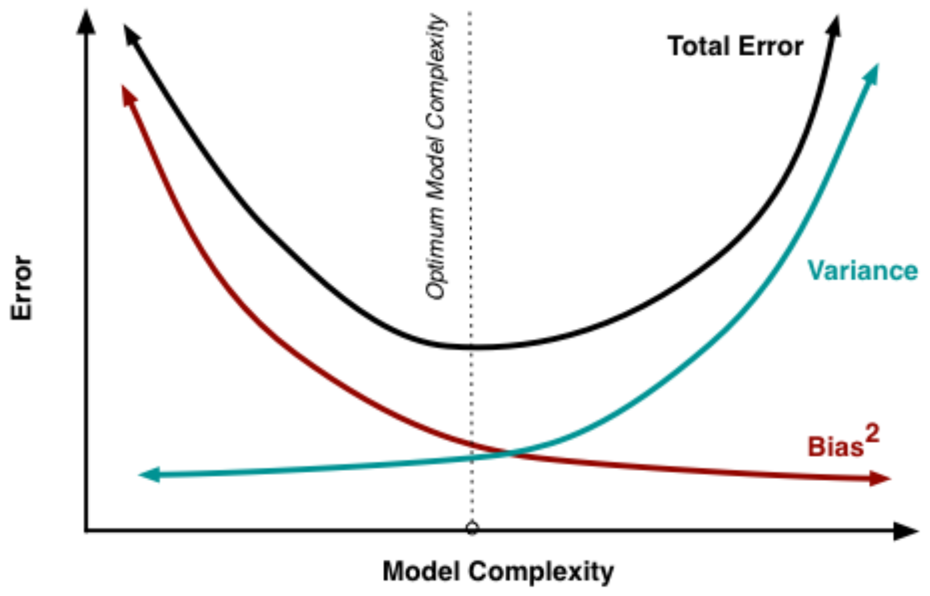


Student picks a best-guess optimal model with reasonable justification using the model complexity graph.

"A max-depth of 4 seems to the most likely to best generalise unseen data. Beyond a max-depth of 4, the training and testing results start to diverge."

I would choose the same! As we are looking for the highest validation score(which is what gridSearch searches for). And we are also looking for a good bias / variance tradeoff(with close training and validation scores).

Check out this visual, it refers to error, but same can be applied to accuracy(just flipped)



Evaluating Model Performance

Student correctly describes the grid search technique and how it can be applied to a learning algorithm.
Nice update here! As you can see that we are using max depth, a decision tree and r square score in this project.

Let's understand what the grid search does with the example we have. In our example we are passing 10 different values [1,2,..9,10] for 'max_depth' to the grid search, meaning, we are asking to run the decision tree regression for each value of 'max_depth'. (1) First fit the decision tree regression model with max_depth = 1, scoring function (r2_score) and train/test data set (which is actually 10 sets of train/test data that was produced using the ShuffleSplit method). Then we do the same for a max depth of 2 and so on. And at the end we are returned the highest scoring max depth for the validation set.

Can also note that since this is an "exhaustive search", one limitation of GridSearch is that it can be very computationally expensive when dealing with a large number of different hyperparameters and much bigger datasets. Therefore there are two other techniques that we could explore to validate our hyperparameters

- RandomizedSearchCV which can sample a given number of candidates from a parameter space with a specified distribution. Which performs surprisingly well!
- Or a train / validation / test split, and we can validate our model on the validation set. Often used with much bigger datasets

Student correctly describes the k-fold cross-validation technique and discusses the benefits of its application when used with grid search when optimizing a model.

"k-fold provides added benefits when used with grid search as it provides more test points, and more validation of the selected parameters."

This actually isn't the main reason, one of them, but not the main. This is an extremely important concept in machine learning, as this allows for multiple testing datasets and is not just reliant on the particular subset of partitioned data. For example, if we use single validation set and perform grid search then it is the chance that we just select the best parameters for that specific validation set. But using k-fold we perform grid search on various validation set so we select best parameter for generalize case. Thus cross-validation better estimates the volatility by giving you the average error rate and will better represent generalization error.

If you would like a full run example, run this code based on the iris data set in your python shell or something and examine the print statements, as this is a great example

```
import numpy as np

from sklearn import cross_validation

from sklearn import datasets

from sklearn import svm



iris = datasets.load_iris()



# Split the iris data into train/test data sets with 30% reserved for test
ing

X_train, X_test, y_train, y_test = cross_validation.train_test_split(iris.
data, iris.target, test_size=0.3, random_state=0)



# Build an SVC model for predicting iris classifications using training da
ta

clf = svm.SVC(kernel='linear', C=1, probability=True).fit(X_train, y_train
)



# Now measure its performance with the test data with single subset

print clf.score(X_test, y_test)



# We give cross_val_score a model, the entire data set and its "real" valu
es, and the number of folds:
```

```
scores = cross_validation.cross_val_score(clf, iris.data, iris.target, cv=
5)


# Print the accuracy for each fold:

print scores


# And the mean accuracy of all 5 folds:

print scores.mean()
```

Student correctly implements the `fit_model` function in code.
In your

```
scoring_fnc = make_scorer(r2_score)
```

You really should pass in `performance_metric` instead of `r2_score`, just to let you know that we can use custom functions.

```
scoring_fnc = make_scorer(performance_metric)
```

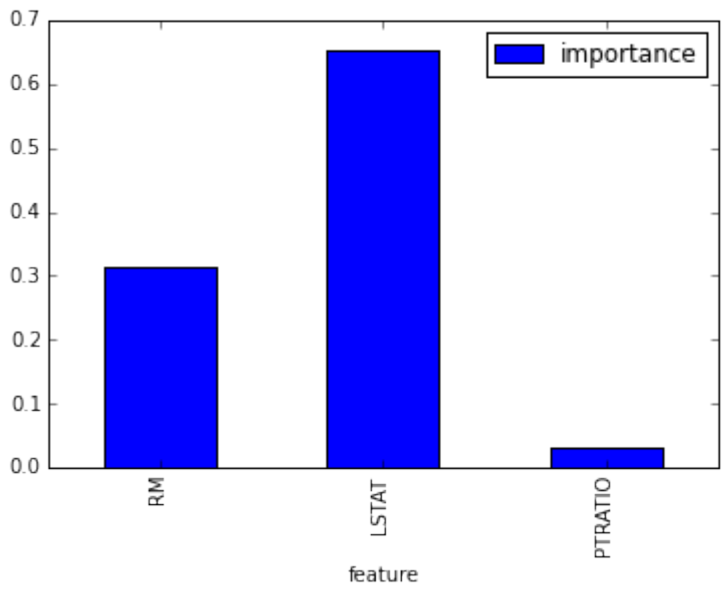
Student reports the optimal model and compares this model to the one they chose earlier.
Congrats! Can note that GridSearch searches for the highest validation score on the different data splits in this ShuffleSplit.
Student reports the predicted selling price for the three clients listed in the provided table. Discussion is made for each of the three predictions as to whether these prices are reasonable given the data and the earlier calculated descriptive statistics.
Excellent justification for these predictions by comparing them to the descriptive stats of the dataset and features. Definitely have not gone off on a tangent with your revised answer. This is exactly what I would do. Just remember to keep in mind the testing error here.

```
reg.score(X_test, y_test)
```

Another cool thing with tree based method is that we can use feature_importances to determine the most important features for the predictions. Check this out(which one of these features contributes to the model the most?)

```
%matplotlib inline
```

```
pd.DataFrame(zip(X_train.columns, reg.feature_importances_), columns=['fea
ture', 'importance']).set_index('feature').plot(kind='bar')
```



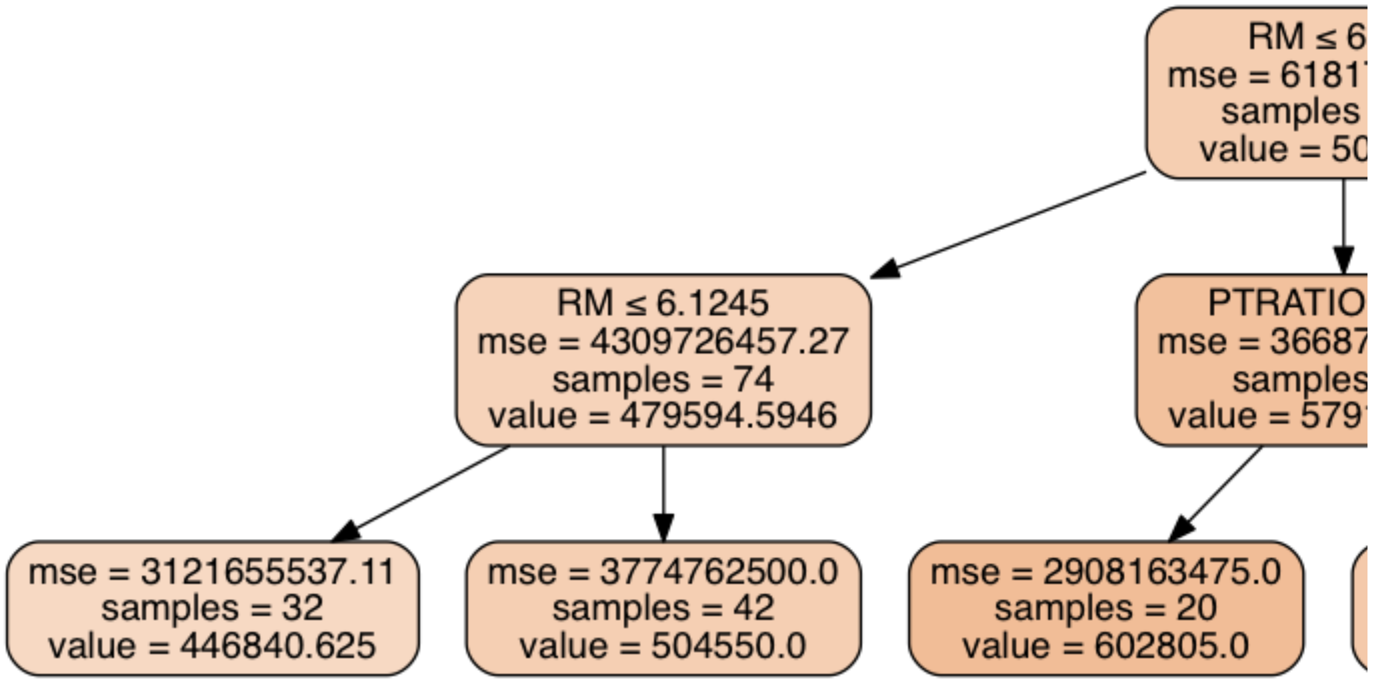
You will see this more in the next project!

Student thoroughly discusses whether the model should or should not be used in a real-world setting.

One of the biggest advantages when using a decision tree as a classifier in the interpretability of the model. Therefore we can actually visualize this exact tree with the use of export_graphviz

```
from IPython.display import Image

from sklearn.externals.six import StringIO

import pydot

from sklearn import tree


clf = DecisionTreeRegressor(max_depth=4)

clf = clf.fit(X_train, y_train)

dot_data = StringIO()

tree.export_graphviz(clf, out_file=dot_data,

    feature_names=X_train.columns,

    class_names="PRICES",

    filled=True, rounded=True,

    special_characters=True)
```

```
graph = pydot.graph_from_dot_data(dot_data.getvalue())

Image(graph.create_png())
```


RM ≤ 6
mse = 6181
samples
value = 50
RM ≤ 6.1245
mse = 4309726457.27
samples = 74
value = 479594.5946
PTRATIO
mse = 3668
samples
value = 579
mse = 3121655537.11
samples = 32
value = 446840.625
mse = 3774762500.0
samples = 42
value = 504550.0
mse = 2908163475.0
samples = 20
value = 602805.0