Cassandra and Machine Learning (Boston Housing) Project

Class: SQL and NoSQL

Mikhail Terentev

Contents

[Summary 1](#_Toc62691834)

[Dataset and Tools 1](#_Toc62691835)

[I. Dataset 1](#_Toc62691836)

[II. Database 1](#_Toc62691837)

[III. Programming Language 1](#_Toc62691838)

[IV. Libraries and Drivers for Python 1](#_Toc62691839)

[V. Tools 1](#_Toc62691840)

[Procedure 2](#_Toc62691841)

[I. Set-Up 2](#_Toc62691842)

[II. Clone data to Git repository 3](#_Toc62691843)

[III. Activate Cassandra Shell 3](#_Toc62691844)

[IV. Upload Data into Cassandra Database 4](#_Toc62691845)

[1) Create a keyspace called BOSTON 4](#_Toc62691846)

[2) Create a table called HOUSING 4](#_Toc62691847)

[3) Copy data from housing.csv to HOUSING table in Cassandra 5](#_Toc62691848)

[4) Test Connection to database HOUSING 6](#_Toc62691849)

[5) Data Processing 7](#_Toc62691850)

[Data Observations 7](#_Toc62691851)

[1) Analyzing Model Performance 8](#_Toc62691852)

[2) Machine Learning Curves 8](#_Toc62691853)

[3) Complexity Curves 9](#_Toc62691854)

[4) Predicting Selling Prices 10](#_Toc62691855)

# Summary

This final project shows basic machine learning concepts on data collected for housing prices in Boston, MA. We will predict the selling price of a new home. This project is designed to get acquainted to working with Cassandra’s datasets in Python and applying basic machine learning techniques using NumPy and Scikit-Learn. Working on that project I was able to use different tools to achieve one goal and was able to go through the whole process: from setting up an environment and creating a database to predicting prices by choosing the prediction model.

# Dataset and Tools

## Dataset

In this project we are using the *CSV* file with house prices and statistics *housing.csv.*

Source: <https://github.com/sushantdhumak/Predicting-Boston-Housing-Prices>

## Database

**The Apache Cassandra** database is the right choice when you need scalability and high availability without compromising performance.

## Programming Language

**Python v. 3.7** is an interpreted, object-oriented, high-level programming language with dynamic semantics.

## Libraries and Drivers for Python

1. **NumPy** is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, etc.
2. **Pandas** is Data Analysis Library for Python.
3. **Cassandra-driver** isDataStax Python Driver**.**
4. **Scikit-learn** is simple and efficient tools for predictive data analysis.

## Tools

1. **VirtualBox v. 6.1** is Oracle VM VirtualBox is a free and open-source hosted hypervisor for x86 virtualization, developed by Oracle Corporation.
2. **Jupyter Notebok v. 6.0.3** exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages.
3. **PuTTY v. 0.73** is free SSH and telnet client for Windows.
4. **Git** is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

# Procedure

## Set-Up

1. Install Git on VM
2. Install and start Cassandra service on VM
3. Start Cassandra client - PuTTy
4. Launch Jupyter Notebook

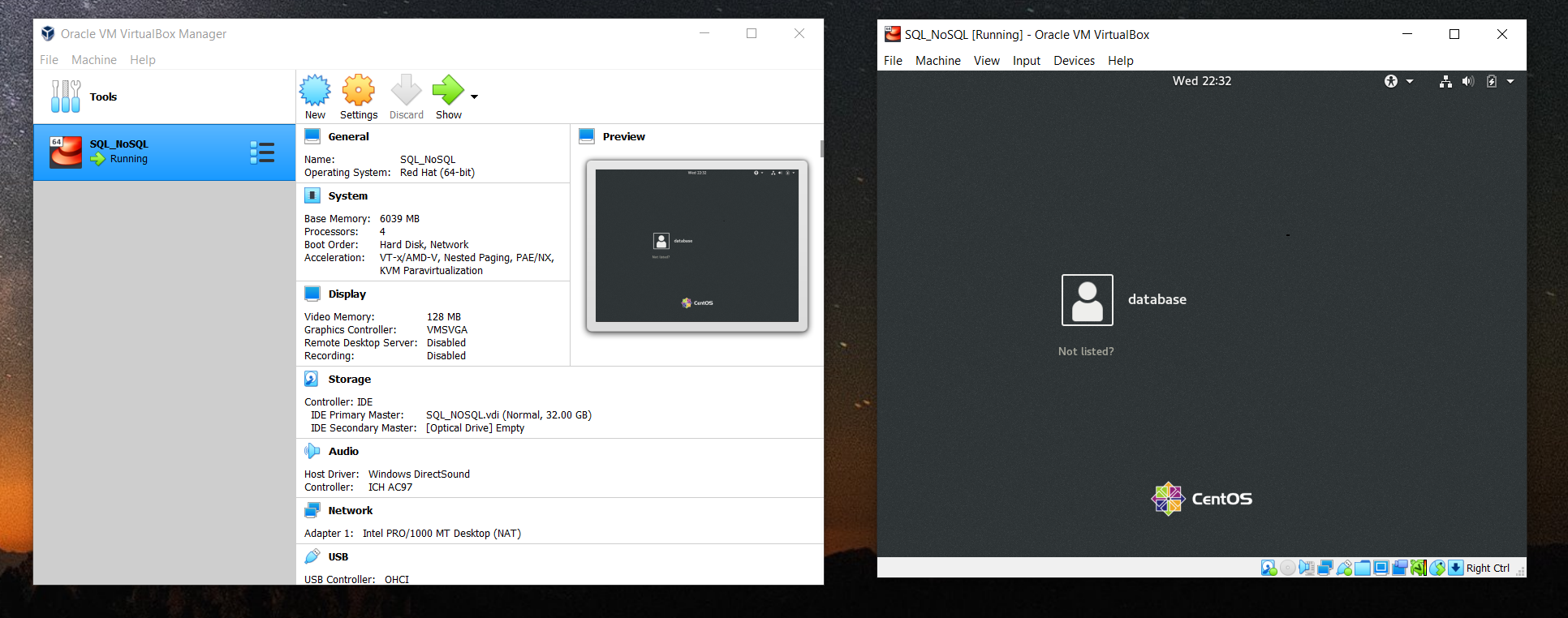


Figure 1 Start the VM

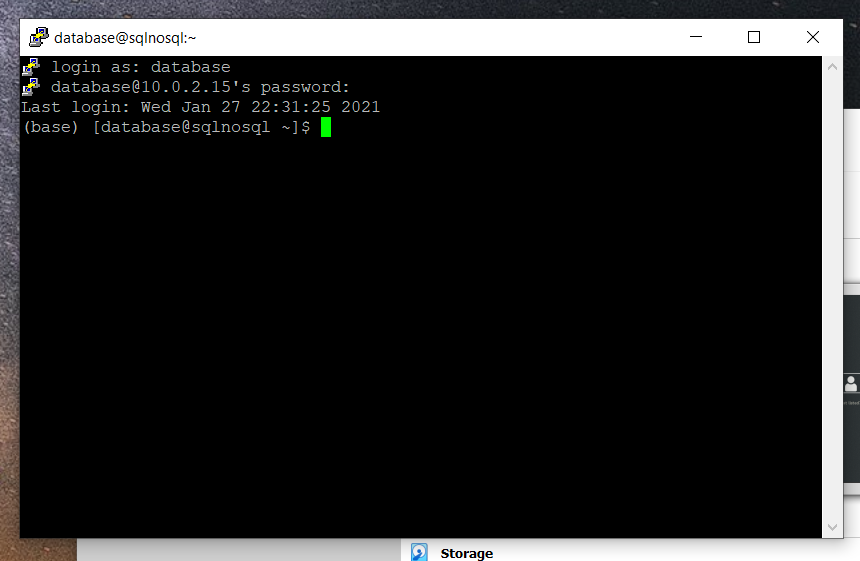


Figure 2 Connect to VM

## Clone data to Git repository

Original project located in GitHub: Boston Housing Price Prediction

https://github.com/sushantdhumak/Predicting-Boston-Housing-Prices

Git clone this project, run in terminal:

*git clone https://github.com/sushantdhumak/Predicting-Boston-Housing-Prices.git*

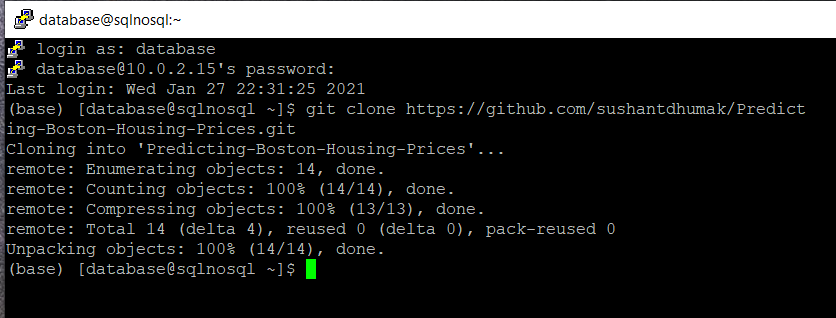


Figure 3 Clone the original project to Git repository

## Activate Cassandra Shell

To connect to Cassandra type following in the terminal:

*cqlsh*

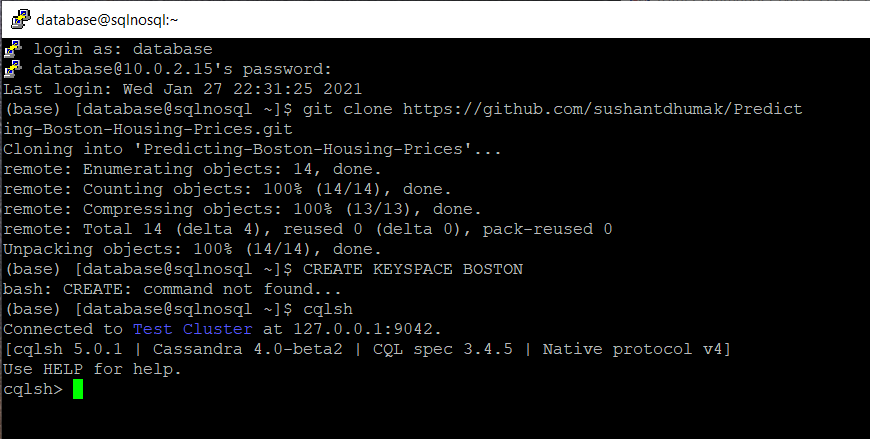


Figure 4 PuTTY client connect to Cassandra on VM

## Upload Data into Cassandra Database

### Create a keyspace called BOSTON

In cqlsh, create a keyspace called BOSTON:

*CREATE KEYSPACE BOSTON WITH replication = {'class': 'SimpleStrategy', 'replication\_factor' : 3};*

Check keyspace:

*desc keyspaces;*

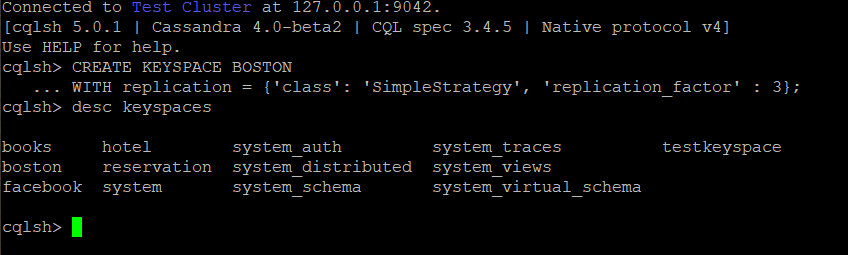


Figure 5 Creating a keyspace called BOSTON

### Create a table called HOUSING

Create a table called HOUSING, which has 4 columns RM, LSTAT, PTRATIO, MEDV, all of them float type:

*CREATE TABLE BOSTON.HOUSING (*

*RM float,*

*LSTAT float,*

*PTRATIO float,*

*MEDV float,*

*PRIMARY KEY ((RM), LSTAT, PTRATIO, MEDV)*

*) WITH comment = 'Predicting Boston Housing Prices';*

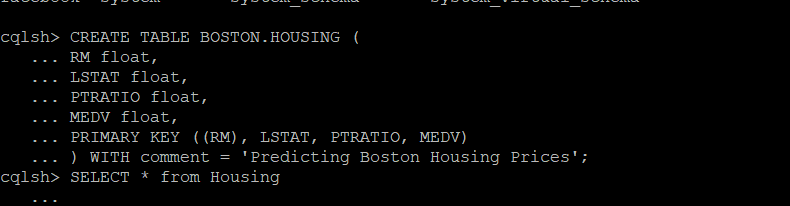


Figure 6 Creating a table called HOUSING

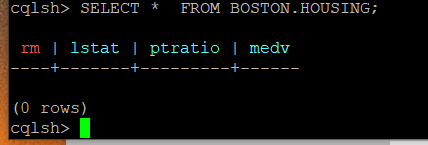


Figure 7 Check a table called HOUSING

### Copy data from housing.csv to HOUSING table in Cassandra

Copy data from housing.csv to housing table in Cassandra in the PuTTy:

*COPY BOSTON.HOUSING FROM '/home/database/Predicting-Boston-Housing-Prices/housing.csv' WITH DELIMITER=',' AND HEADER=TRUE;*

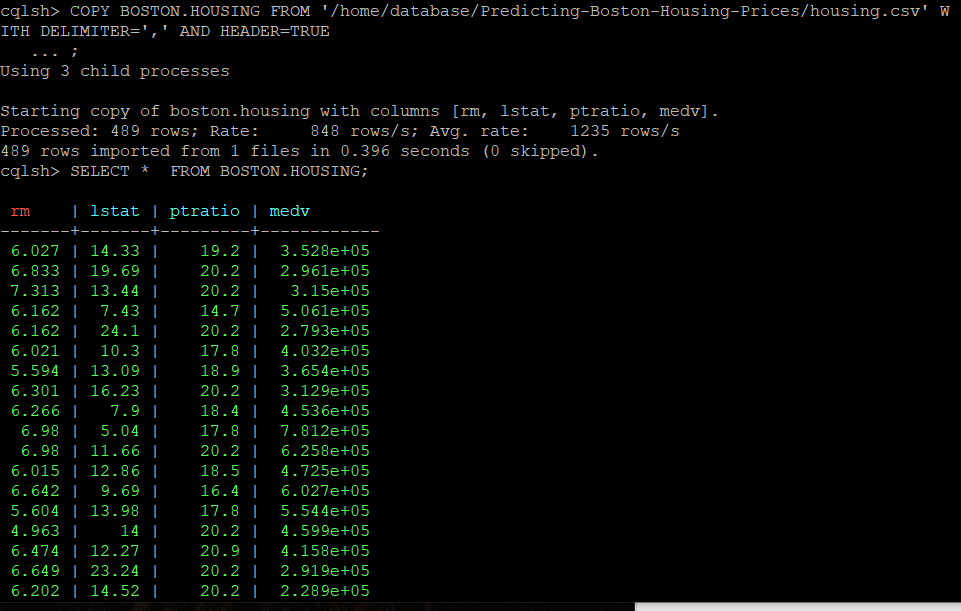


Figure 8 Copy data from housing.csv to housing table in Cassandra

Verify data copy to the table:

*SELECT \* FROM BOSTON.HOUSING;*

Now we have the housing data in BOSTON.HOUSING table in Cassandra on VM.

### Test Connection to database HOUSING

For check connection to database HOUSING we need run in the Jupyter Notebok code below (please see file ***Boston\_Project.ipynb***):

*from cassandra.cluster import Cluster*

*from cassandra.auth import PlainTextAuthProvider*

*import pandas as pd*

*def pandas\_factory(colnames, rows):*

*return pd.DataFrame(rows, columns=colnames)*

*cluster = Cluster(*

*contact\_points=['localhost], port=34,*

*auth\_provider = PlainTextAuthProvider(username='database', password='123456')*

*)*

*session = cluster.connect()*

*session.set\_keyspace('boston')*

*session.row\_factory = pandas\_factory*

*session.default\_fetch\_size = 10000000 #needed for large queries, otherwise driver will do pagination. Default is 50000.*

*rows = session.execute("""select \* from housing""")*

*df = rows.\_current\_rows*

*print (df.head())*

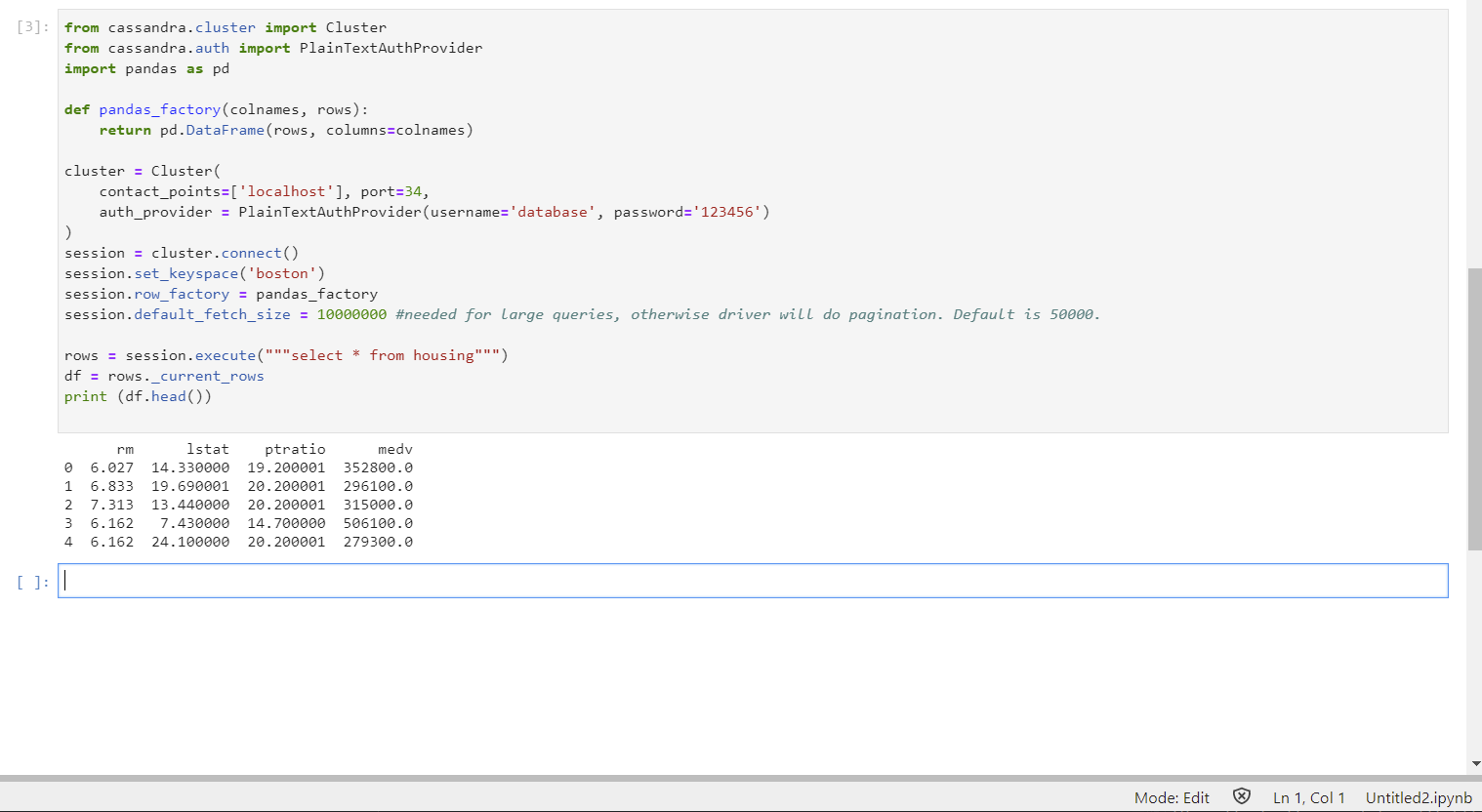


Figure 9 Print data from Database

rm lstat ptratio medv

0 6.027 14.330000 19.200001 352800.0

1 6.833 19.690001 20.200001 296100.0

2 7.313 13.440000 20.200001 315000.0

3 6.162 7.430000 14.700000 506100.0

4 6.162 24.100000 20.200001 279300.0

Figure 10 Result of data frame. It shows first 5 rows of the table

Now we know the dataset loaded successfully if the size of the dataset.

### Data Processing

For Data Observation we have to run ***Predicting Boston Housing Prices.ipynb*** without any modification in the code.

# Data Observations

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

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

Scatter plots for each feature vs price:

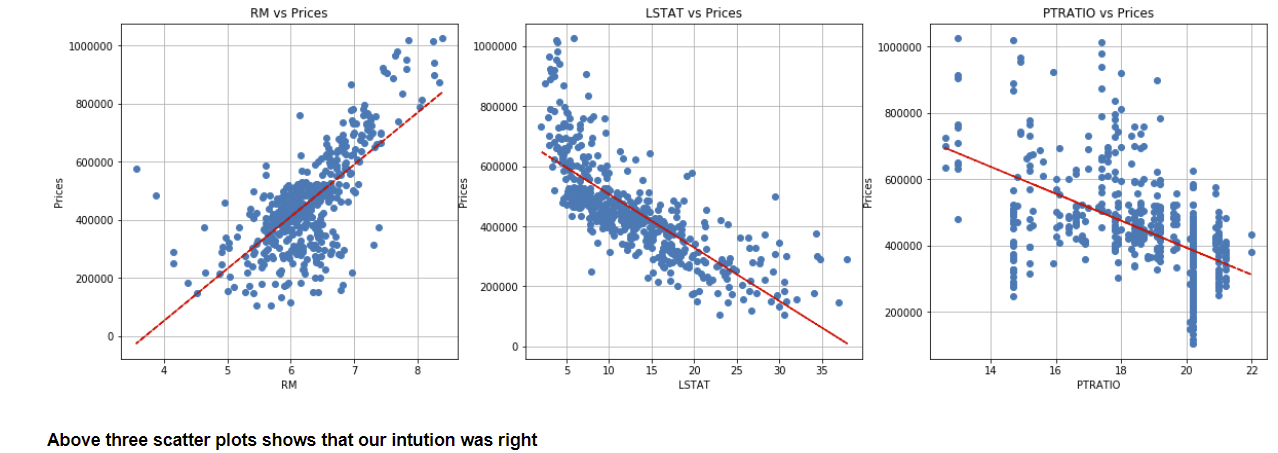


Figure 11 Scatter plots for features vs price

### Analyzing Model Performance

Let’s take a look at several models' learning and testing performances on various subsets of training data. Additionally, we will investigate one particular algorithm with an ncreasing 'max\_depth' parameter on the full training set to observe how model complexity affects performance. Graphing our 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.

### Machine Learning Curves

Let’s take a look at 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 R2, the coefficient of determination.

**Note:** The section uses helper functions i.e. ‘visuals’ module, supplied with this project.

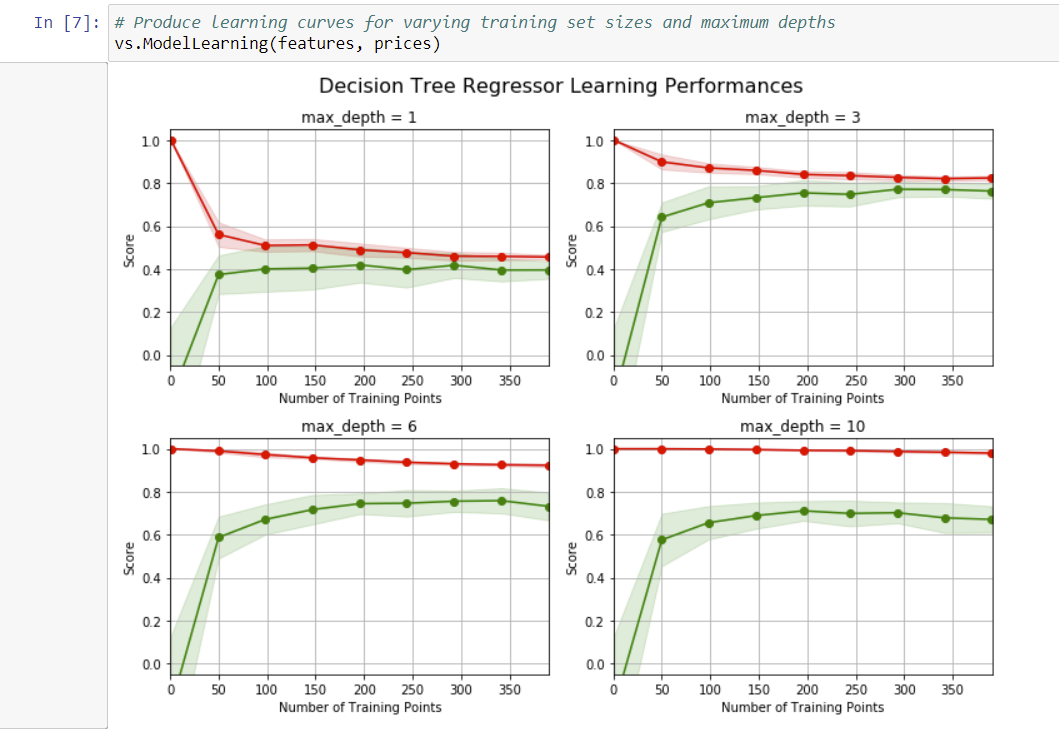


Figure 12 Four graphs for a decision tree model with different maximum depths

#### **Considering learning curve of model with Maximum Depth = 3**

1. For given depth value, as more training points are added to the model the score of training curve slowly starts decreasing
2. There is a instant increase in testing score from 0 to 0.7 for first 50 data points and then it is gradually increasing for further addition of data points.
3. The training and testing curve seems to be converging to a score of 0.8

Therefore, having more training point will surely not benefits the model.

### Complexity Curves

Let’s take a look at 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.

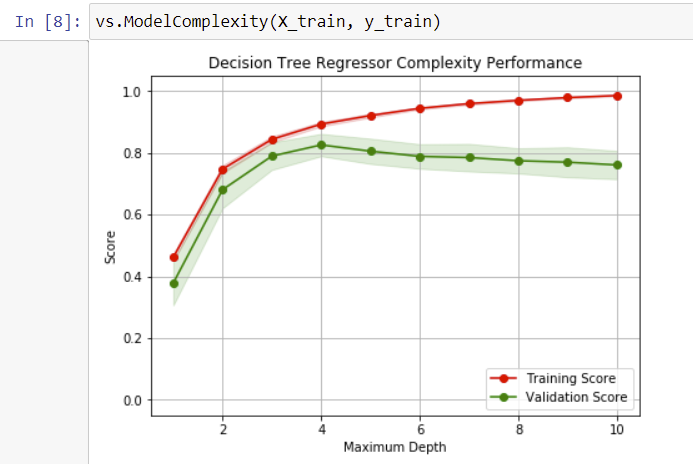


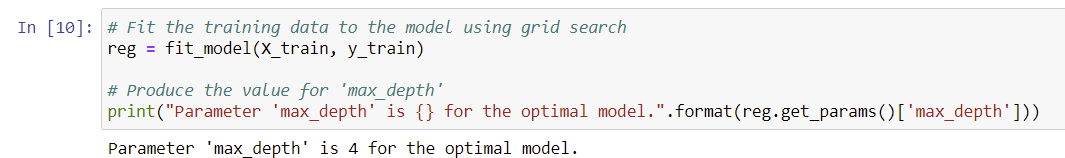
Figure 13 Decision tree regressor complexity performance

#### **Bias-Variance Tradeoff**

1. **Model trained with max\_depth of 1:** The model suffers from high bias at this depth. Looking at the graph, we can see that both training and validation scores are low. The model is over-simplified or underfitted and is not capturing the underlying relationships present in the data for both training and validation datasets.
2. **Model trained with max\_depth of 10:** The model suffers from high variance at this depth, and is overfitted to the training data. In the graph, we can see that the training score at this depth is almost equal to 1.0, while the validation score is lower, at around 0.75. The curves also seem to be diverging away from each other at this point.

#### **Optimal Model**

Let’s run next part of code:



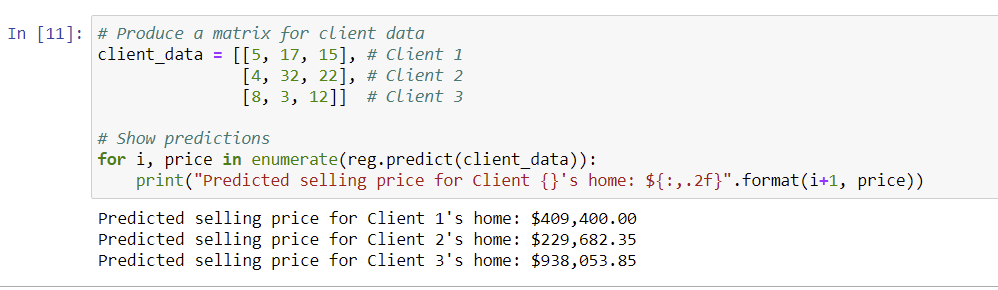
As we can see the optimal model has a Max depth = 4, which is exactly what we guessed from the model complexity curve.

### Predicting Selling Prices

Imagine that we are a real estate agent in the Boston area looking to use this model to help price homes owned by our clients that they wish to sell. We have collected the following information from three of our 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% |
| Student-teacher ratio of nearby schools | 15-to-1 | 22-to-1 | 12-to-1 |

Let’s see what prices our model will predict for these clients to sell their home, and also whether it is reasonable for the respective features:



Using the best fit model, we able to predict house prices depends on features:

Predicted selling price for Client 1's home: $409,400.00

Predicted selling price for Client 2's home: $229,682.35

Predicted selling price for Client 3's home: $938,053.85