Getting familiarized with Making predictions process using Machine Learning

Learning Objectives

At the end of this session you will be able to:

- Understand machine learning based making prediction process
- Apply correlation technique for feature selection purpose
- Build machine learning predictor for Boston Housing
- Learn how to evaluate the predictor

Introduction

Making predictions using Machine Learning isn't just about grabbing the data and feeding it to algorithms. The algorithm might spit out some prediction but that's not what you are aiming for. The difference between good data science professionals and naive data science aspirants is that the former set follows this process religiously. The process is as follows: 1. Understand the problem: Before getting the data, we need to understand the problem we are trying to solve. If you know the domain, think of which factors could play an epic role in solving the problem. If you don't know the domain, read about it. 2. Hypothesis Generation: This is quite important, yet it is often forgotten. In simple words, hypothesis generation refers to creating a set of features which could influence the target variable given a confidence interval (taken as 95% all the time). We can do this before looking at the data to avoid biased thoughts. This step often helps in creating new features. 3. Get Data: Now, we download the data and look at it. Determine which features are available and which aren't, how many features we generated in hypothesis generation hit the mark, and which ones could be created. Answering these questions will set us on the right track. 4. Data Exploration: We can't determine everything by just looking at the data. We need to dig deeper. This step helps us understand the nature of variables (missing, zero variance feature) so that they can be treated properly. It involves creating charts, graphs (univariate and bivariate analysis), and cross-tables to understand the behavior of features. 5. *Data Preprocessing: *Here, we impute missing values and clean string variables (remove space, irregular tabs, data time format) and anything that shouldn't be there. This step is usually followed along with the data exploration stage. 6. Feature Engineering: Now, we create and add new features to the data set. Most of the ideas for these features come during the hypothesis generation stage. 7. Model Training: Using a suitable algorithm, we train the model on the given data set. 8. Model Evaluation: Once the model is trained, we evaluate the model's performance using a suitable error metric. Here, we also look for variable importance, i.e., which variables have proved to be significant in determining the target variable. And, accordingly we can shortlist the best variables and train the model again. 9. Model Testing: Finally, we test the model on the unseen data (test data) set.

We'll follow this process in the project to arrive at our final predictions. Let's get started.

1.Understand the problem

This lab aims at predicting house prices (residential) in Boston, USA. I believe this problem statement is quite self-explanatory and doesn't need more explanation. Hence, we move to the next step.

2. Hypothesis Generation

Well, this is going to be interesting. What factors can you think of right now which can influence house prices ? As you read this, I want you to write down your factors as well, then we can match them with the data set. Defining a hypothesis has two parts: Null Hypothesis (Ho) and Alternate Hypothesis(Ha). They can be understood as:

Ho - There exists no impact of a particular feature on the dependent variable. Ha - There exists a direct impact of a particular feature on the dependent variable.
Based on a decision criterion (say, 5% significance level), we always 'reject' or 'fail to reject' the null hypothesis in statistical parlance. Practically, while model building we look for probability (p) values. If p value < 0.05, we reject the null hypothesis. If $p > 0.05$, we fail to reject the null hypothesis. Some factors which I can think of that directly influence house prices are the following:
Per capita crime rate by town
Proportion of residential land zoned for lots over 25,000 sq. ft
Proportion of non-retail business acres per town
Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
Nitric oxide concentration (parts per 10 million)
Average number of rooms per dwelling
Proportion of owner-occupied units built prior to 1940
Weighted distances to five Boston employment centers
Index of accessibility to radial highways
Full-value property tax rate per $10,000
Pupil-teacher ratio by town
$1000(Bk — 0.63)^2$, where Bk is the proportion of [people of African American descent] by town
LSTAT: Percentage of lower status of the population
Median value of owner-occupied homes in $1000s
…keep thinking. I am sure you can come up with many more apart from these.

3. Get Data

You can download the data from this https://www.kaggle.com/altavish/boston-housing-dataset and load it in your python IDE. Also, check the competition page where all the details about the data and variables are given. The data set consists of 13 explanatory variables. Yes, it's going to be one heck of a data exploration ride. But, we'll learn how to deal with so many variables. The target variable is MEDV. As you can see the data set comprises numeric, categorical, and ordinal variables.

4. Data Exploration

Data Exploration is the key to getting insights from data. Practitioners say a good data exploration strategy can solve even complicated problems in a few hours. A good data exploration strategy comprises the following:

1. Univariate Analysis - It is used to visualize one variable in one plot. Examples: histogram, density plot, etc.
2. Bivariate Analysis - It is used to visualize two variables (x and y axis) in one plot. Examples: bar chart, line chart, area chart, etc.
3. Multivariate Analysis - As the name suggests, it is used to visualize more than two variables at once. Examples: stacked bar chart, dodged bar chart, etc.

4. Cross Tables -They are used to compare the behavior of two categorical variables (used in pivot tables as well).

Let's load the necessary libraries and data and start coding.

```
In [ ]: #Loading libraries
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        %matplotlib inline
        plt.rcParams['figure.figsize'] = (10.0, 8.0)
        import seaborn as sns
        from scipy import stats
        from scipy.stats import norm
```

```
In [ ]: #loading data
        data = pd.read_csv("HousingData.csv")
```

After we read the data, we can look at the data using:

```
In [3]: data.head()
```

```
Out[3]:
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	NaN	36.2

The description of all the features is given below:

CRIM: Per capita crime rate by town
ZN: Proportion of residential land zoned for lots over 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 oxide 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 centers
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 [people of African American descent] by town
LSTAT: Percentage of lower status of the population
MEDV: Median value of owner-occupied homes in $1000s

The prices of the house indicated by the variable `MEDV` is our ***target variable*** and the remaining are the ***feature variables*** based on which we will predict the value of a house.

```
print ('The train data has {0} rows and {1} columns'.format(data.shape[0],data.shape[1]))
print ('---------------------------')
```

```
The train data has 506 rows and 14 columns
---------------------------
```

Alternatively, you can also check the data set information using the info() command.

5.Data Preprocessing

After loading the data, it's a good practice to see if there are any missing values in the data. We count the number of missing values for each feature using isnull()

```
data.columns[data.isnull().any()]
```

```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'AGE', 'LSTAT'], dtype='object')
```

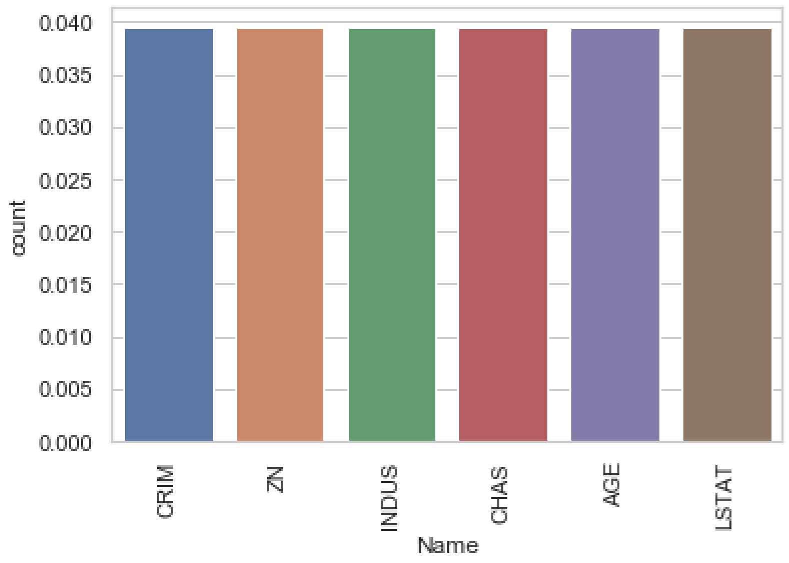
Out of 14 features, 6 features have missing values. Let's check the percentage of missing values in these columns.

```
#missing value counts in each of these columns
miss = data.isnull().sum()/len(data)
miss = miss[miss > 0]
miss.sort_values(inplace=True)
miss
```

```
CRIM     0.039526
ZN       0.039526
INDUS    0.039526
CHAS     0.039526
AGE      0.039526
LSTAT    0.039526
dtype: float64
```

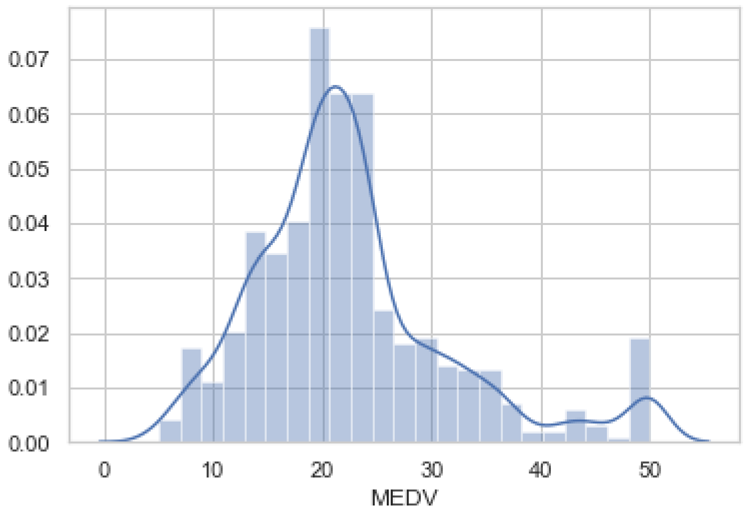
We can infer that the all variables has 3.9% missing values. Let's look at a pretty picture explaining these missing values using a bar plot.

```python
#visualising missing values
miss = miss.to_frame()
miss.columns = ['count']
miss.index.names = ['Name']
miss['Name'] = miss.index

#plot the missing value count
sns.set(style="whitegrid", color_codes=True)
sns.barplot(x = 'Name', y = 'count', data=miss)
plt.xticks(rotation = 90)
plt.show()
```



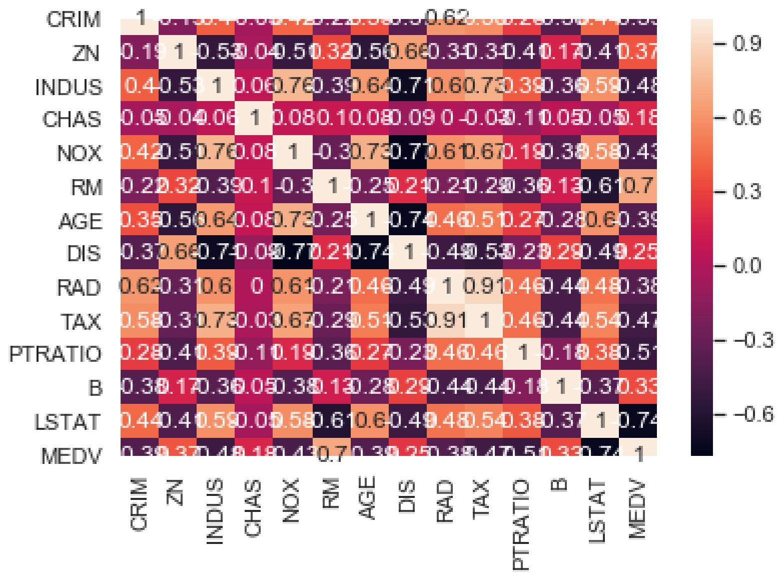
Let's proceed and check the distribution of the target variable.

```python
#Target Variable
sns.distplot(data['MEDV'])
```



We see that the values of `MEDV` are distributed normally with few outliers.
Next, we create a correlation matrix that measures the linear relationships between the variables. The correlation matrix can be formed by using the `corr` function from the pandas dataframe library. We will use the `heatmap` function from the seaborn library to plot the correlation matrix.

```
correlation_matrix = data.corr().round(2)
# annot = True to print the values inside the square
sns.heatmap(data=correlation_matrix, annot=True)
```



The correlation coefficient ranges from -1 to 1. If the value is close to 1, it means that there is a strong positive correlation between the two variables. When it is close to -1, the variables have a strong negative correlation.

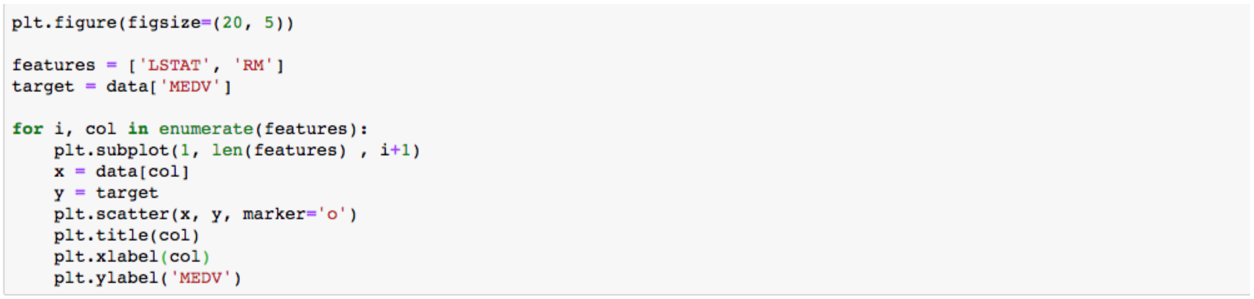
Observations:

To fit a linear regression model, we select those features which have a high correlation with our target variable MEDV. By looking at the correlation matrix we can see that RM has a strong positive correlation with MEDV (0.7) where as LSTAT has a high negative correlation with MEDV(-0.74).

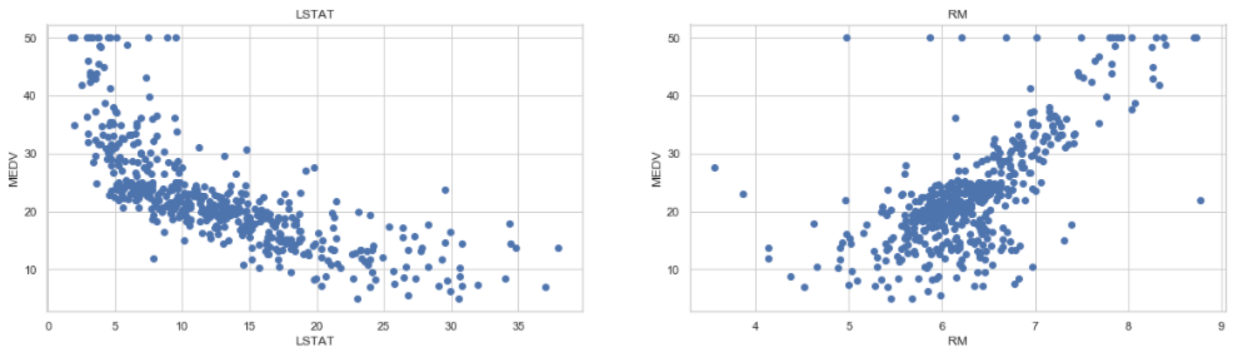
6. Feature Engineering

An important point in selecting features for a linear regression model is to check for multi-co-linearity. The features RAD, TAX have a correlation of 0.91. These feature pairs are strongly correlated to each other. We should not select both these features together for training the model. Check this for an explanation. Same goes for the features DIS and AGE which have a correlation of -0.75.

Based on the above observations we will select RM and LSTAT as our features. Using a scatter plot let's see how these features vary with MEDV.



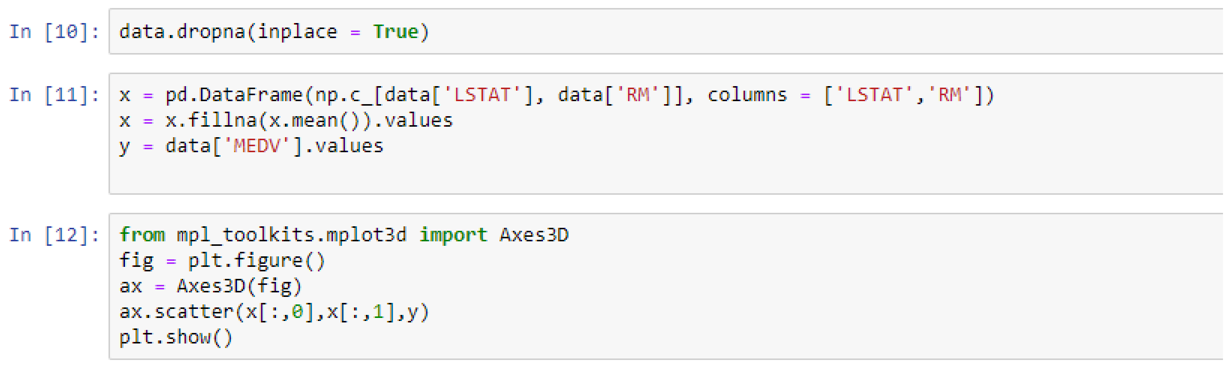
```
plt.figure(figsize=(20, 5))

features = ['LSTAT', 'RM']
target = data['MEDV']

for i, col in enumerate(features):
    plt.subplot(1, len(features) , i+1)
    x = data[col]
    y = target
    plt.scatter(x, y, marker='o')
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel('MEDV')
```



Observations:

The prices increase as the value of RM increases linearly. There are few outliers and the data seems to be capped at 50.

The prices tend to decrease with an increase in LSTAT. Though it doesn't look to be following exactly a linear line.

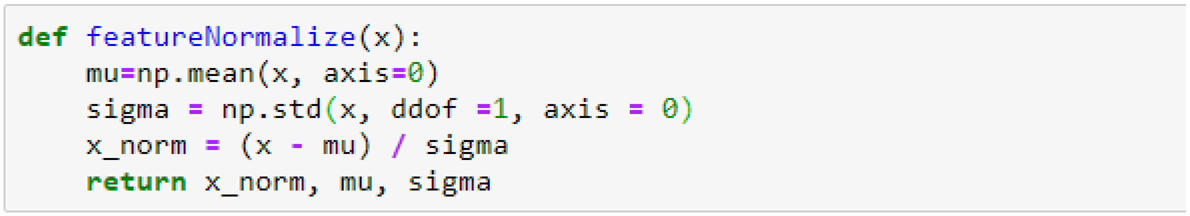


```
In [10]: data.dropna(inplace = True)

In [11]: x = pd.DataFrame(np.c_[data['LSTAT'], data['RM']], columns = ['LSTAT','RM'])
         x = x.fillna(x.mean()).values
         y = data['MEDV'].values

In [12]: from mpl_toolkits.mplot3d import Axes3D
         fig = plt.figure()
         ax = Axes3D(fig)
         ax.scatter(x[:,0],x[:,1],y)
         plt.show()
```

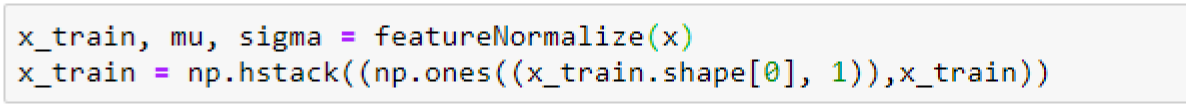
Feature Normalization

If you look at the values, note that house sizes are about 1000 times the number of bedrooms. When features differ by orders of magnitude, first performing feature scaling can make gradient descent converge much more quickly. To normalize the dataset the following steps is used:
Subtract the mean value of each feature from the dataset.
After subtracting the mean, additionally scale (divide) the feature values by their respective "standard deviations."

The standard deviation is a way of measuring how much variation there is in the range of values of a particular feature (most data points will lie within ±2 standard deviations of the mean); this is an alternative to taking the range of values (max-min). When normalizing the features, the values used for normalization should be kept for later use. After learning the parameters from the model, we often want to predict the prices of houses we have not seen before. Given a new x value (living room area and number of bedrooms), we must first normalize x using the mean and standard deviation that we had previously computed from the training set. Here the code snapshot that is used for the normalization purpose.



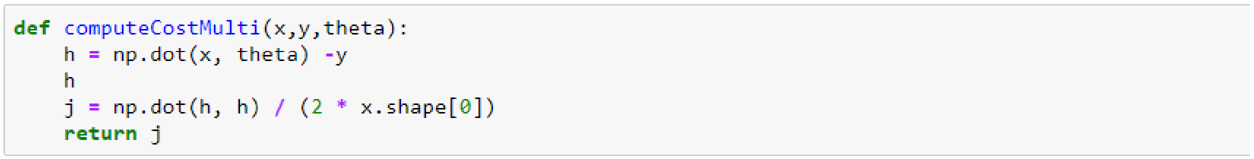
```python
def featureNormalize(x):
    mu=np.mean(x, axis=0)
    sigma = np.std(x, ddof =1, axis = 0)
    x_norm = (x - mu) / sigma
    return x_norm, mu, sigma
```



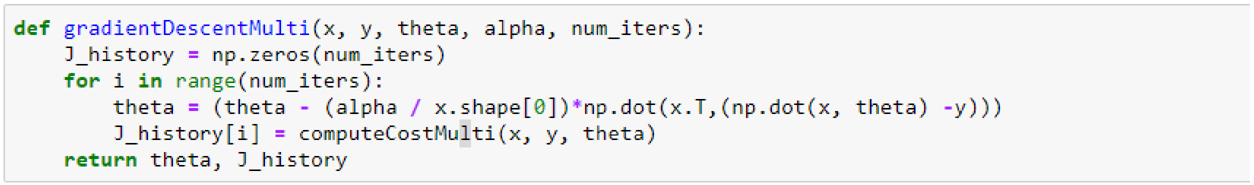
```python
x_train, mu, sigma = featureNormalize(x)
x_train = np.hstack((np.ones((x_train.shape[0], 1)),x_train))
```

Model Training using Gradient descent

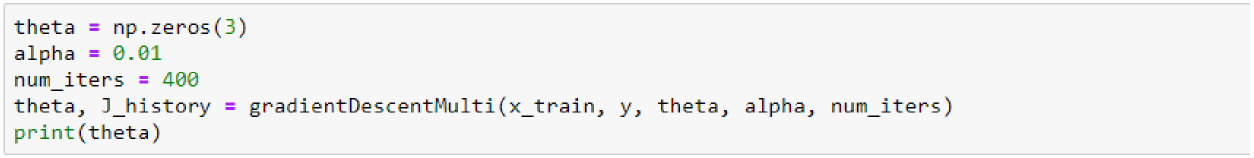
Here is the code snap shot for implementing the gradient descent optimization techniques according to the formula discussed during the lecture session.



```python
def computeCostMulti(x,y,theta):
    h = np.dot(x, theta) -y
    h
    j = np.dot(h, h) / (2 * x.shape[0])
    return j
```



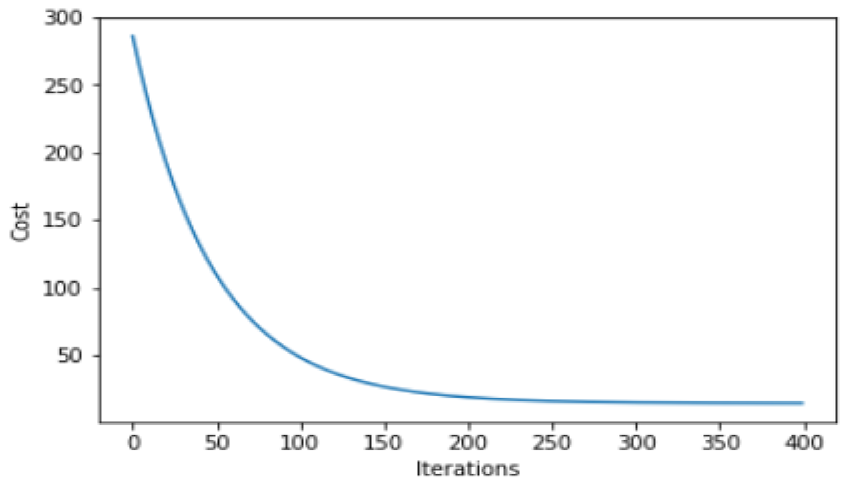
```python
def gradientDescentMulti(x, y, theta, alpha, num_iters):
    J_history = np.zeros(num_iters)
    for i in range(num_iters):
        theta = (theta - (alpha / x.shape[0])*np.dot(x.T,(np.dot(x, theta) -y)))
        J_history[i] = computeCostMulti(x, y, theta)
    return theta, J_history
```



```python
theta = np.zeros(3)
alpha = 0.01
num_iters = 400
theta, J_history = gradientDescentMulti(x_train, y, theta, alpha, num_iters)
print(theta)
```

One of the mechanisms to make sure that gradient descent is working properly or not is to look at the value of the cost function and check that it is decreasing with each iteration. After the gradient descent is properly implemented, the value of the cost function must decease and should converge to a steady value by the end of the algorithm. The final values for the model parameters will be used to make prediction based on the new observations. The following code snap shoot and graph demonstrate the correct behavior of the cost function.

```python
plt.figure()
plt.plot(np.arange(num_iters),J_history)
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.show()
```



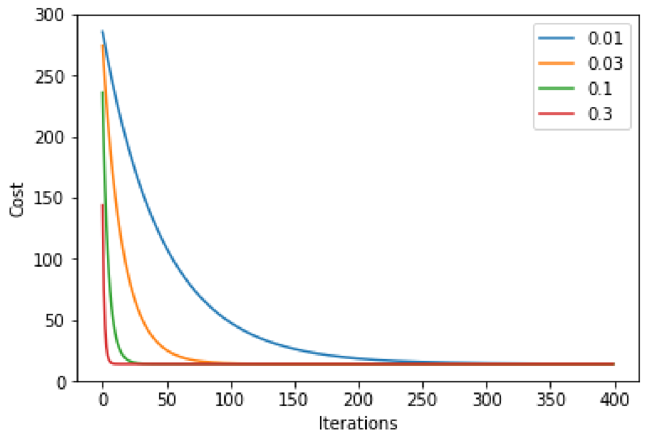
Testing the model

Using the final value of the model parameters predict MEDV with LSTAT 9.14 and RM 6.42. The values should be normalized.

Analyzing the Impact of Learning Rate

In this part you will apply different values of learning rates for the dataset and find a learning rate that converges quickly. Here is the code snap shoot and the graph of cost function against the number of iterations for different values of learning rates.

```
# different learning rates
alphas = [0.01, 0.03, 0.1, 0.3]
plt.figure()
for alpha in alphas:
    theta = np.zeros(3)
    num_iters = 400
    theta, J_history = gradientDescentMulti(x_train, y, theta, alpha, num_iters)
    plt.plot(np.arange(num_iters),J_history, label = str(alpha))
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.legend()
plt.show()
```



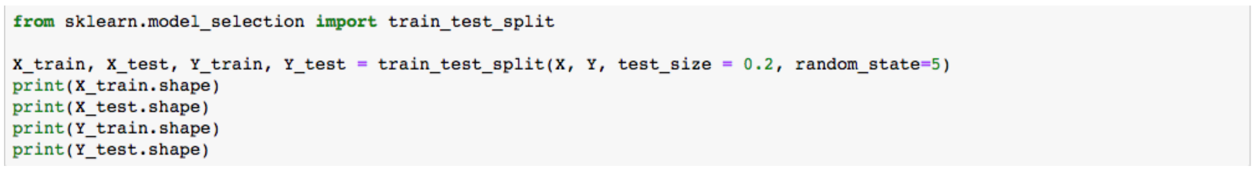
7. Model Training

We concatenate the LSTAT and RM columns using np.c_ provided by the numpy library.

```
X = pd.DataFrame(np.c_[data['LSTAT'], data['RM']], columns = ['LSTAT','RM'])
Y = data['MEDV']
```

Splitting the data into training and testing sets

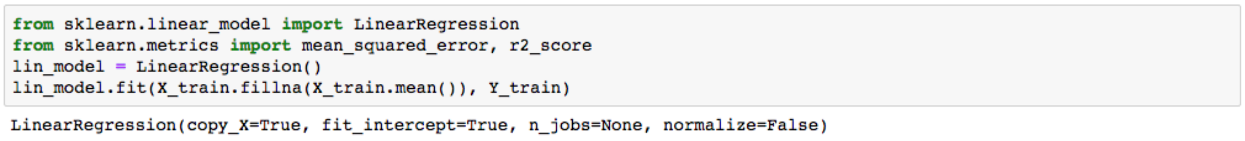
Next, we split the data into training and testing sets. We train the model with 80% of the samples and test with the remaining 20%. We do this to assess the model’s performance on unseen data. To split the data we use train_test_split function provided by scikit-learn library. We finally print the sizes of our training and test set to verify if the splitting has occurred properly.



```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=5)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

Training and testing the model

We use scikit-learn’s LinearRegression to train our model on both the training and test sets.

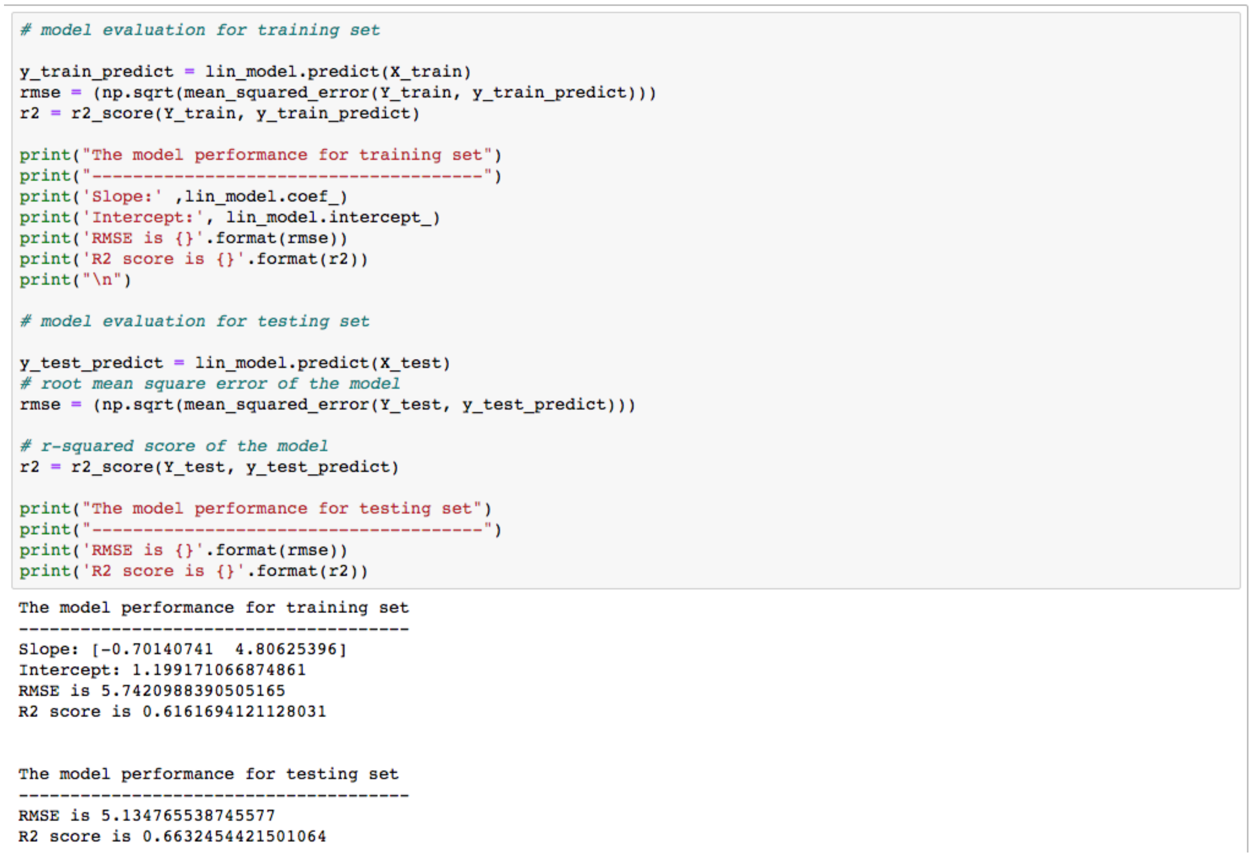


```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
lin_model = LinearRegression()
lin_model.fit(X_train.fillna(X_train.mean()), Y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Model evaluation

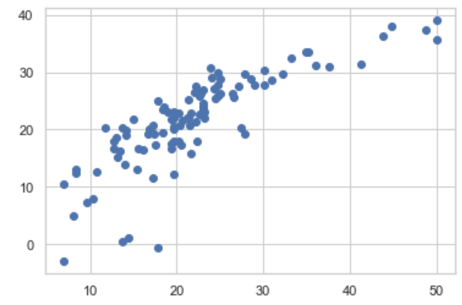
We will evaluate our model using RMSE and R2-score.



```
# model evaluation for training set

y_train_predict = lin_model.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))
r2 = r2_score(Y_train, y_train_predict)

print("The model performance for training set")
print("--------------------------------------")
print('Slope:' ,lin_model.coef_)
print('Intercept:', lin_model.intercept_)
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")

# model evaluation for testing set

y_test_predict = lin_model.predict(X_test)
# root mean square error of the model
rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))

# r-squared score of the model
r2 = r2_score(Y_test, y_test_predict)

print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

```
The model performance for training set
--------------------------------------
Slope: [-0.70140741  4.80625396]
Intercept: 1.199171066874861
RMSE is 5.7420988390505165
R2 score is 0.6161694121128031


The model performance for testing set
--------------------------------------
RMSE is 5.134765538745577
R2 score is 0.6632454421501064
```

```
# plotting the y_test vs y_pred
# ideally should have been a straight line
plt.scatter(Y_test, y_test_predict)
plt.show()
```



Assignment

This question will use the Boston housing dataset once again. Again, create a test set consisting of 1/2 of the data using the rest for training.

1. Build and evaluate the model by using additional one feature which has high feature next to RM and LSTV
2. Fit a polynomial regression model to the training data.
3. Predict the labels for the corresponding test data.
4. Evaluate and generate the model parameters.
5. Out of these predictors used in this assignment, which would you choose as a final model for the boston housing?