```
# coding: utf-8

# ## Predicting Continuous Target Variables With Regresssion Analysis

# - Read the dataset from the url into a data frame.
# - Display the first few rows of the data frame to make sure the data was read properly

# In[1]:

import pandas as pd
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data', header=None, sep='\s+')
df.columns = ['CRIM', 'ZN', 'INDUS', 'CHAS',
              'NOX', 'RM', 'AGE', 'DIS', 'RAD',
              'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
df.head()

# - Visualize the important characteristics of the dataset before attempting to build a regression model (Exploratory Data Analysis - EDA - is a recommended first step prior to the training of a machine learning model. We will create a scatterplot matrix that allows us to visualize the pair-wise correlations between different features in the dataset in one place.

# In[2]:

import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style='whitegrid', context='notebook')
cols = ['LSTAT', 'INDUS', 'NOX', 'RM', 'MEDV']
```

```
sns.pairplot(df[cols], size=2.5)

plt.show()
```

- Create a correlation matrix to quantify the linear relationships between features.

In[3]:

```
import numpy as np

cor_matrix = np.corrcoef(df[cols].values.T) # note we transpose to get the data by columns. COlumns become rows.

sns.set(font_scale=1.5)

cor_heat_map = sns.heatmap(cor_matrix,

                cbar=True,

                annot=True,

                square=True,

                fmt='.2f',

                annot_kws={'size':15},

                yticklabels=cols,

                xticklabels=cols)

plt.show()
```

- Separate the independent and dependent variables into two variables X and y and also standardize the data

In[4]:

```
X = df[['RM']].values # note how we pass a list of columns (here just a single-item list) to access data in df
```

```
y = df[['MEDV']].values # you either have to define this list of columns separately or inline like here, where you get the [[ ...]]

#print(df[['MEDV']].head())

from sklearn.preprocessing import StandardScaler

sc_x = StandardScaler()

sc_y = StandardScaler()

X_std = sc_x.fit_transform(X)

y_std = sc_y.fit_transform(y)


# let's see how sklearn does the scaling

print('original data\n', y[0:4])

print('scaled data using sklearn StandardScaler\n', y_std[0:5])


#let's do the same computation manually

print('average of original data', np.average(y[:]))

print('STDV of original data', np.std(y[:]))

z_scores = (y[:] - np.average(y[:])) / np.std(y[:])

print('Manually computed Z scores:\n', z_scores[0:5] )



# - Build the linear regressionmodel


# In[5]:


from sklearn.linear_model import LinearRegression

slr = LinearRegression()

# first, let's fit the un-standardized data

slr.fit(X,y)

print('Slope: %.3f' % slr.coef_[0])
```

print('Intercept: %.3f' % slr.intercept_)

now, let's try with standardized data

slr_std = LinearRegression()

slr_std.fit(X_std,y_std)

print('Slope: %.3f' % slr_std.coef_[0])

print('Intercept: %.3f' % slr_std.intercept_)

- let's make a prediction using both models (the model using standardized data and the model using un-standardized data)

In[6]:

predict the price of a 5 bedroom house

import numpy as np

num_rooms = [5.0]

num_rooms_std = sc_x.transform(np.array(num_rooms).reshape(len(num_rooms),1)) # note transform expects a 2D array

print('standardized rooms: %.3f', num_rooms_std)

predicted_price_std = slr_std.predict(num_rooms_std)

print('Predicted Price std: %.3f' % predicted_price_std)

print("Predicted Price in $1000's using standardized data: %.3f" % sc_y.inverse_transform(predicted_price_std))

Now let's predict using the model that uses un-standardized data

predicted_price_non_std = slr.predict(np.array(num_rooms).reshape(len(num_rooms),1))

print("Predicted Price in $1000's Using un-standardized data: %.3f" % predicted_price_non_std)

- Visualize how well the linear regression line fits the data

```
#     - plot a scatterplot of the training data
#     - add the regression line

# In[7]:

plt.scatter(X_std,y_std, c='blue')
plt.plot(X_std,slr_std.predict(X_std), color='red')
plt.xlabel('Average number of rooms [RM] (standardized)')
plt.ylabel('Price in $1000\'s [MEDV] (standardized)')
plt.show()


# - now visualize the original, un-standardized training data and regression line

# In[8]:

plt.scatter(X,y, color='blue')
plt.plot(X, slr.predict(X), color='red')
plt.xlabel('Average Number of Rooms [RM]')
plt.ylabel('Price in $1000\'s [MEDV]')
plt.show()


# - evaluate the performance of the regresion model on training data

# In[9]:

from sklearn.metrics import mean_squared_error, r2_score
# first let's focus on original un-standardized data and model
```

```
MSE1 = mean_squared_error(y_true= y, y_pred= slr.predict(X))

r2_1 = r2_score(y_true= y, y_pred= slr.predict(X))

print('MSE for regression model using un-standrdized features: %.3f' % MSE1)

print('RMSE:', np.sqrt(MSE1))

print('r2:', r2_1)

print('==============================')

#now let's do the same thing for standardized data and regression model

MSE_2 = mean_squared_error(y_true= y_std, y_pred=slr_std.predict(X_std) )

print('MSE for standardized data:', MSE_2)

y_true_converted_to_originl = sc_y.inverse_transform(y_std)

y_pred_converted_to_original = sc_y.inverse_transform(slr_std.predict(X_std))

MSE2 = mean_squared_error(y_true= y_true_converted_to_originl, y_pred=
y_pred_converted_to_original)


r2_2 = r2_score(y_true= y_std, y_pred= slr_std.predict(X_std))

r2_2_using_data_converted_to_original_scale = r2_score(y_true= y_true_converted_to_originl,
y_pred= y_pred_converted_to_original)

print('MSE using standardized features converted to original scale: %.3f' % MSE2)

print('RMSE:', np.sqrt(MSE2))

print('r2 using standardized data and model:',r2_2)

print('r2 using data converted to original scale', r2_2_using_data_converted_to_original_scale)
```