Dylan Cozloff
AST05
CS 422 1001

Diamond Price OLS / SGD

Installs:
pip install pandas
pip install scikit-learn
pip install matplotlib
pip install seaborn

Dataset:
https://www.kaggle.com/datasets/shivam2503/diamonds

I used a diamond price dataset (53.9k entries) which is a collection of diamond dimensions and carats, with their prices. All features: carat, depth-percentage, width-percentage, length, width, depth, cut, color, and clarity. But, I only manually extracted six continuous features (carat, depth-percentage, width-percentage, length, width, depth) from this dataset. The continuous output was the diamond's price. I loaded the dataset using the pandas library, in which I use the function read_csv().

Training/Test Data Splits: I used the sklearn.model_selection function train_test_split() to use a random 20% of the dataset for testing, and the remaining for training.

OLS Data Preperation: I appended an x0 column of 1's to be the first row of the dataset. This allows the OLS matrix operations to generate a bias (intercept) term in the set of parameters.

OLS Model Training: I first converted the pandas training/test & input/output datasets to numpy for matrix operations. I then transposed the training input using np.matrix.transpose(), and multiplied it to the original using np.matmul(), I then got the inverse of the product using np.linalg.inv(). I then calculated the product of the prior operations with the training input transposed and the training output transposed to get the optimal parameters.

```
OLS Linear Equation:
price = 20976.563736020486*x0 + 10683.18425965889*carat + -204.09955335264056*depth% + -104.26736871735861*width% + -1286.7995742830
012*Length + 37.63252068422567*Width + 53.3948642392736*Depth
```

OLS Model Predictions: I used np.matmul() to get the product of the OLS parameters and the training and test inputs individually to get an array of training and testing output predictions respectively.

OLS Model Evaluation: I used the sklearn.metrics function mean_squared_error() to pass the training & test data with its real-valued predictions to get the mean squared error: **Training MSE** = 2240338, **Test MSE** = 2242178. I also used the sklearn.metrics function mean_absolute_error() to get the mean absolute error: **Training MAE** = 891, **Test MAE** = 888. I also used the sklearn.metrics function r2_score() to get the coefficient of determination: **Training R2** = 0.8593, **Test R2** = 0.8590.

Dylan Cozloff
AST05
CS 422 1001

SGD Model Training: I first created a pipeline (sklearn.pipeline) with make_pipeline(), StandardScaler(), and SGDRegressor() to scale and organize my data. In SGDRegressor I passed the default parameters max_iter=1000 to clarify the number of passes throught the training set, and tol=1e-3 for the stopping criterion. This is a gradient descent model that dynamically changes the learning rate as we reach an optimal solution.

SGD Model Predictions: Using the SGDRegressor model from before and the predict() function with the training and test inputs individually to get an array of training and testing output predictions respectively.

SGD Model Evaluation: I used the sklearn.metrics function mean_squared_error() to pass the training & test data with its real-valued predictions to get the mean squared error: **Training MSE** = 2242415, **Test MSE** = 2245159.

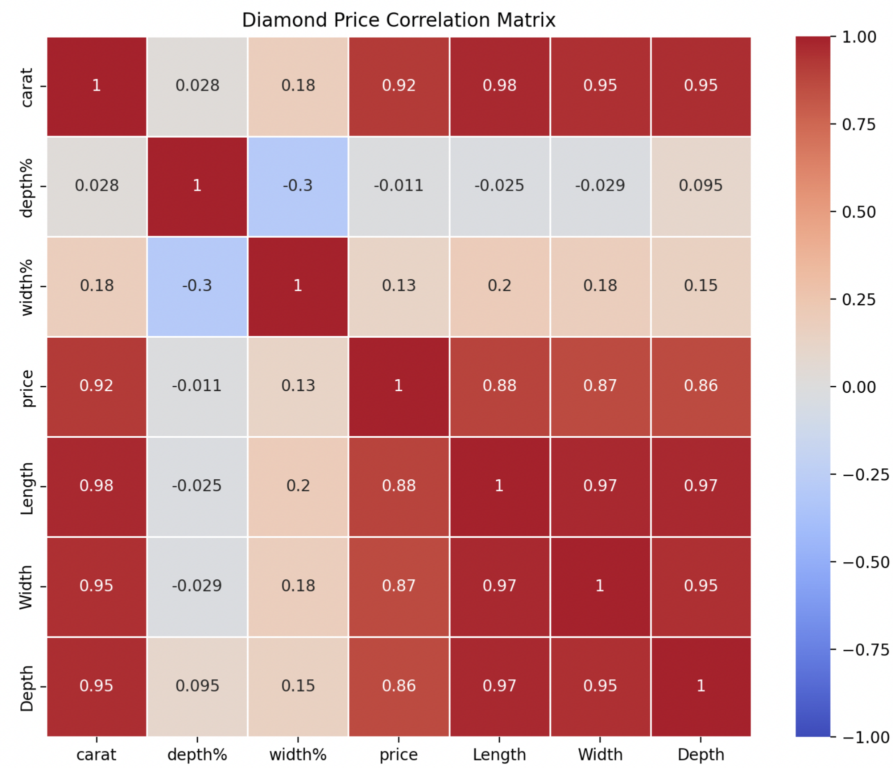
I also used the sklearn.metrics function mean_absolute_error() to get the mean absolute error: **Training MAE** = 892, **Test MAE** = 889.

I also used the sklearn.metrics function r2_score() to get the coefficient of determination: **Training R2** = 0.8591, **Test R2** = 0.8588.

NEXT PAGE

Dylan Cozloff
AST05
CS 422 1001

Diamond Price Correlation Heat Map:



I used the corr() function to evaluate the dataset, and matplotlib.pyplot / seaborn to visualize the correlation matrix. As you can see there is definitely a correlation between diamond price and length, width, height, and carat. You can also see that width% or fatter diamonds sold for more.