CS-C3240 - Machine Learning D Project 2024 - Stage 2

Diamond Price prediction using Polynomial Regression and Multilayer Perceptron models

1. Introduction

In the domain of gemstone appraisal, accurate pricing is crucial for both buyers and sellers, as it directly impacts financial decisions in the luxury goods market. With a view to providing more reliable price estimates, the project aims to use Machine Learning to predict the price of diamonds based on their measurements by leveraging data on key attributes like carat, volume, cut, clarity, and color. With potential applications in jewelry retail auction houses, and insurance companies, the model can help professionals make informed pricing decisions and assist the process of diamond valuation [1]. In this supervised learning project, we will use polynomial regression with degree 2 with mean squared error. As an alternative, multilayer perception (MLP) with maximum 8 layers was applied with mean squared error.

Section 1 of this report is an introduction to the topic, while section 2 discusses how to formulate the above-mentioned problem to make it suitable for applying different machine learning problems. Section 3 discusses the details of the dataset, data processing, appliance of polynomial regression, and MLP. Section 4 compares the results of different methods and evaluates them, and section 5 summarizes the report and discusses further possible improvements.

2. Problem Formulation

The dataset is retrieved from Kaggle, originally taken from Diamond Price Prediction [2]. Each data point/entry (or row) represents a diamond with 10 main features. The meaning and datatype of each feature are stated in Table 1. The data types are continuous (for numeric features carat, depth, table, length, width, height, and price) and categorical (for features cut, color, and clarity).

Table 1. Features' explanation and datatype

Feature	Explanation	Datatype
carat	The diamond's physical weight measured in metric carats	float
cut	The quality of the cut (Fair, Good, Very Good, Premium, Ideal)	object
color	The diamond's color, ranges from J (worst) to D (best)	object
clarity	The measurement of how clear the diamond is (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best))	object
depth	The diamond's height (in mm) measured from the culet (bottom tip) to the table (flat, top surface). Total depth percentage z / mean(x, y)	float
table	The width of top surface of the diamond relative to the widest point	float
x	The diamond's length in mm	float
y	The diamond's width in mm	float
z	The diamond's height in mm	float
price (label)	The price of the diamond in US dollars.	integer

3. Methods

3.1 Dataset

As mentioned above, the dataset was taken from Diamond Price Prediction, with 53940 entries and 10 features. There is missing data from length, width and height fields, representing dimensionless and 2-D diamonds. I filtered out the faulty data points with feature values "x", "y", "z" = 0. To reduce redundancy in "x", "y", "z", we create a new feature called "volume" to characterize the size of the diamond, which follows the formula (assuming the diamond is an ellipsoid):

$$V = \frac{4\pi}{3} \cdot x \cdot y \cdot z$$

Linear Regression model is used as the visualization shows a linear relationship between the features and the labels. From the visualization, we can see a linear relationship between the price and volume, and the price and carat. Therefore, for **feature selection**, "volume" and "carat" are chosen as features for the training of the model since they show a linear proportional relationship with the "price" (Figure 1). Other features such as "cut" and "table" show no clear correlation with the "price".

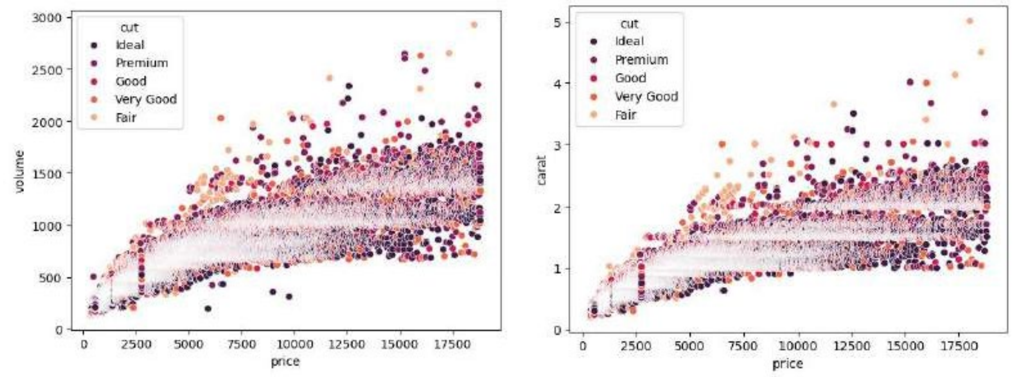


Figure 1. Price-volume and price-carat correlation of diamonds.

Next, we have to do an extra step. According to the Figure 2 showing the relation between "volume" and "price", "volume" has some dimensional outliers in our dataset that need to be eliminated. We dropped the data points with volume equal or greater than 3000 mm^3.

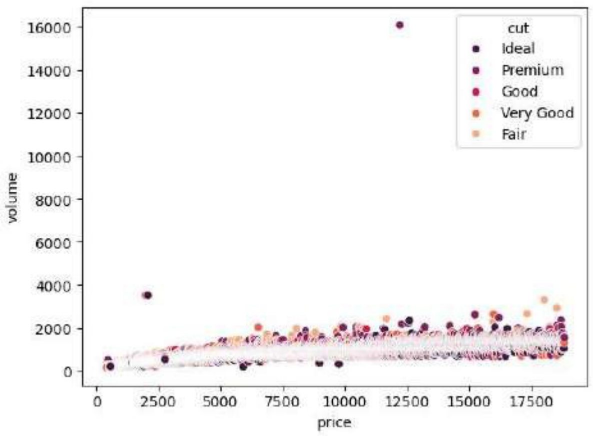


Figure 2. Price-volume correlation of diamonds before dropping the outliers.

3.2 Polynomial regression model

Polynomial regression model is used as there is no simple linear relationship between the feature values and label value. Therefore, in order to capture the non-linear relationship between carat, volume and price more accurately, we use polynomial regression.

For this method, mean squared error (MSE) was used since it is applied in polynomial regression problems, which involve a combination of polynomial features with linear mapping. By using MSE, we can ensure that larger errors are penalized more heavily, which is useful in this context where even small inaccuracies in price prediction can be significant.

MLP is used since it can learn complex, non-linear relationships between the input features and target variable. It can learn the non-linear interactions pattern automatically through its hidden layers, allowing approximation of intricate functions that might exist between the features and the price, improving predictive performance.

For this method, mean squared error (MSE) was used because it directly quantifies how well the model's predictions align with the true prices. By minimizing MSE, the model learns to reduce the prediction errors, which is crucial in price forecasting where precision is key.

To **construct the training and validation set**, we will use a single split into training, validation and testing set, with 60% of the data for training, 30% for validating and 10% for testing. The ratio is chosen because this dataset is balanced, so no k–fold cross validation is needed. 30% is reserved for validating to ensure no overfitting occurs during the training, and 10% testing will be used to calculate the final model accuracy.

4. Results

4.1 Polynomial regression model results

We ran 2 polynomial regression models, one with only carat chosen as the feature value, the other one with both carat and volume chosen as the feature value. By using MSE to calculate training and validation errors, polynomial regressions with the different sets of features chosen were compared.

Feature chosen	carat	carat and volume
Training error	2228881.4264446446	2166557.8259721794
Validation error	2229900.192454173	2158954.170942284

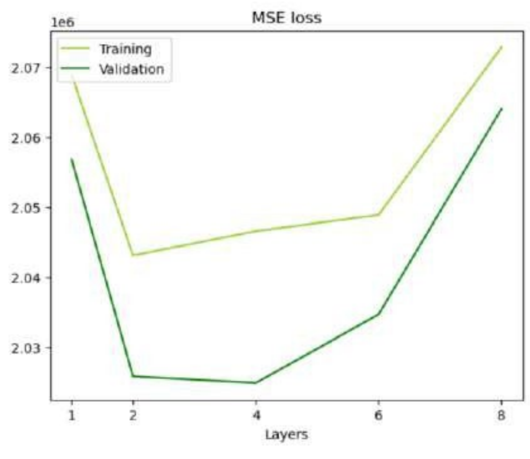
Considering these training and validation errors, the better option is using both carat and volume as input features since they output smaller training and validation errors.

4.2 MLP model results

We ran 5 MLP models with the number of layers being 1, 2, 4, 6, 8 to find the optimal number of layers that yield the smallest error. By using MSE to calculate training and validation errors, MLP with different numbers of layers were compared.

Layers	1	2	4	6	8
Training error	2068894	2043110	2046573	2048870	2072813
Validation error	2056802	2025843	2024896	2034672	2064018

Considering these training and validation errors and the figure below, the best option is 4 layers since it outputs the smallest set of training and validation errors. The model yields 0.86 R^2 correlation.



4.3 Comparing different models using test sets

Comparing the validation errors of polynomial regression with degree 2 for both carat and volume dependency (2158954) and validation error of MLP with a number of layers 4 (2024896) shows that on the given validation set, MLP is performing better. Therefore, MLP network with 4 layers was chosen as the final model.

However, to understand the accuracy of the final model, it is needed to evaluate its performance with the test set. The test set consists of elements that are completely new to the model and gives a valuable metric to measure the model's success. In this project, the test set is 10% of the whole dataset. The test set accuracy was calculated by using mean squared error, the same as before and it is 2033726.1682279143 for the final chosen model, MLP with 4 layers.

5. Conclusions

It was expected to have better predictions by MLP, since unlike polynomial regression, which requires manual feature engineering to capture non-linear interactions, an MLP can automatically learn these patterns through its hidden layers. However, the difference between the validation errors of the best MLP model with the better polynomial regression model is not a huge number. Although MLP is a better option, it is more computationally expensive compared to polynomial regression, therefore; this is not a viable option when the data volume is larger.

The training error was close to the validation error, which implies low chance of overfitting. However, the errors are still a huge number, so for improvement in the future, we consider experimenting with another model, e.g. a deeper network with a more efficient non-linearity.

6. References

[1] *The Art and Science of Jewelry Appraisals: A Comprehensive Guide. Link:* https://harrisjewelersnm.com/blogs/rio-rancho-and-albuquerque-jewelers/the-art-and-science-of-jewelry-appraisals-a-comprehensive-guide

[2] *Diamond Price Prediction. Link: https://www.kaggle.com/code/karnikakapoor/diamond-price-prediction/notebook*

Appendix

October 9, 2024

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

[2]:
```
data = pd.read_csv("diamonds.csv")
data.head()
data.shape
```

[2]:
```
(53940, 11)
```

[3]:
```
#The first column seems to be just index
data = data.drop(["Unnamed: 0"], axis=1)
data.describe()
```

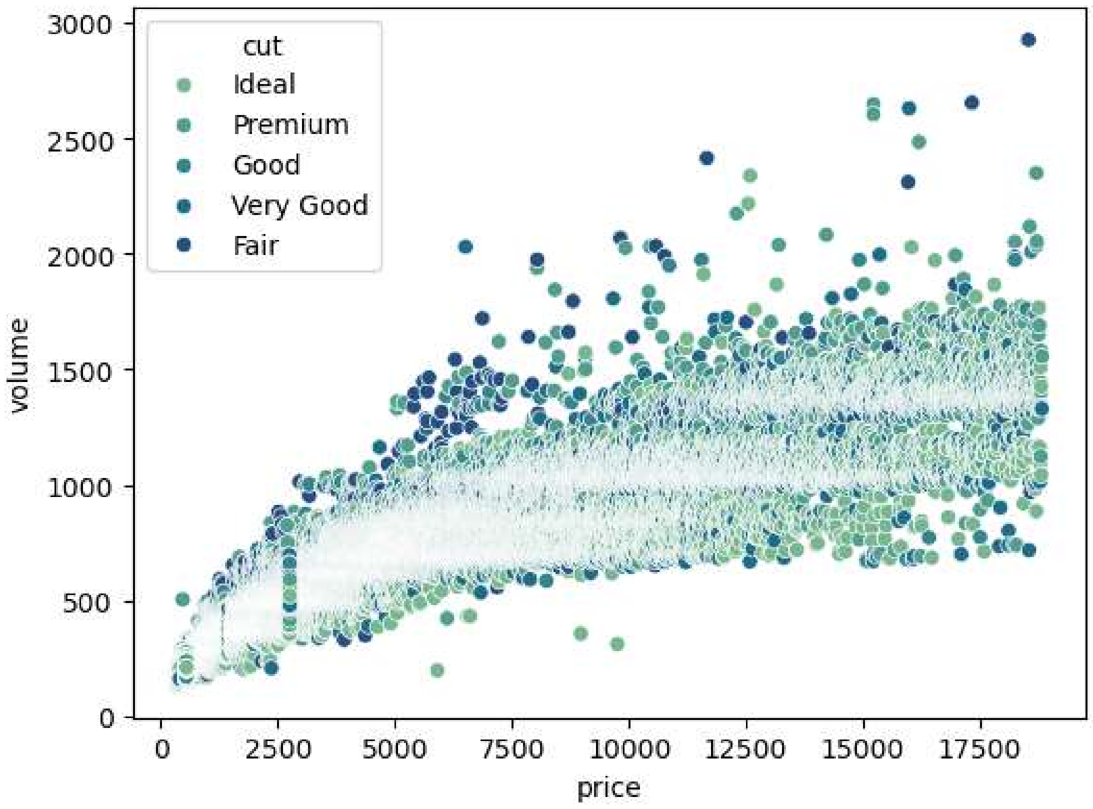
[3]:
```
              carat         depth         table         price             x  \
count  53940.000000  53940.000000  53940.000000  53940.000000  53940.000000
mean       0.797940     61.749405     57.457184   3932.799722      5.731157
std        0.474011      1.432621      2.234491   3989.439738      1.121761
min        0.200000     43.000000     43.000000    326.000000      0.000000
25%        0.400000     61.000000     56.000000    950.000000      4.710000
50%        0.700000     61.800000     57.000000   2401.000000      5.700000
75%        1.040000     62.500000     59.000000   5324.250000      6.540000
max        5.010000     79.000000     95.000000  18823.000000     10.740000

                  y             z
count  53940.000000  53940.000000
mean       5.734526      3.538734
std        1.142135      0.705699
min        0.000000      0.000000
```

```
25%        4.720000      2.910000
50%        5.710000      3.530000
75%        6.540000      4.040000
max       58.900000     31.800000
```

[4]:
```
#Dropping dimentionless diamonds
data = data.drop(data[data["x"]==0].index)
data = data.drop(data[data["y"]==0].index)
data = data.drop(data[data["z"]==0].index)
```

[5]:
```
data["volume"] = 4*np.pi*data["x"]*data["y"]*data["z"]/3
data = data.drop(data[data["volume"]>=3000].index)
sns.scatterplot(data=data, x="price", y="volume", hue="cut", palette="crest")
```

[5]:
```
<Axes: xlabel='price', ylabel='volume'>
```



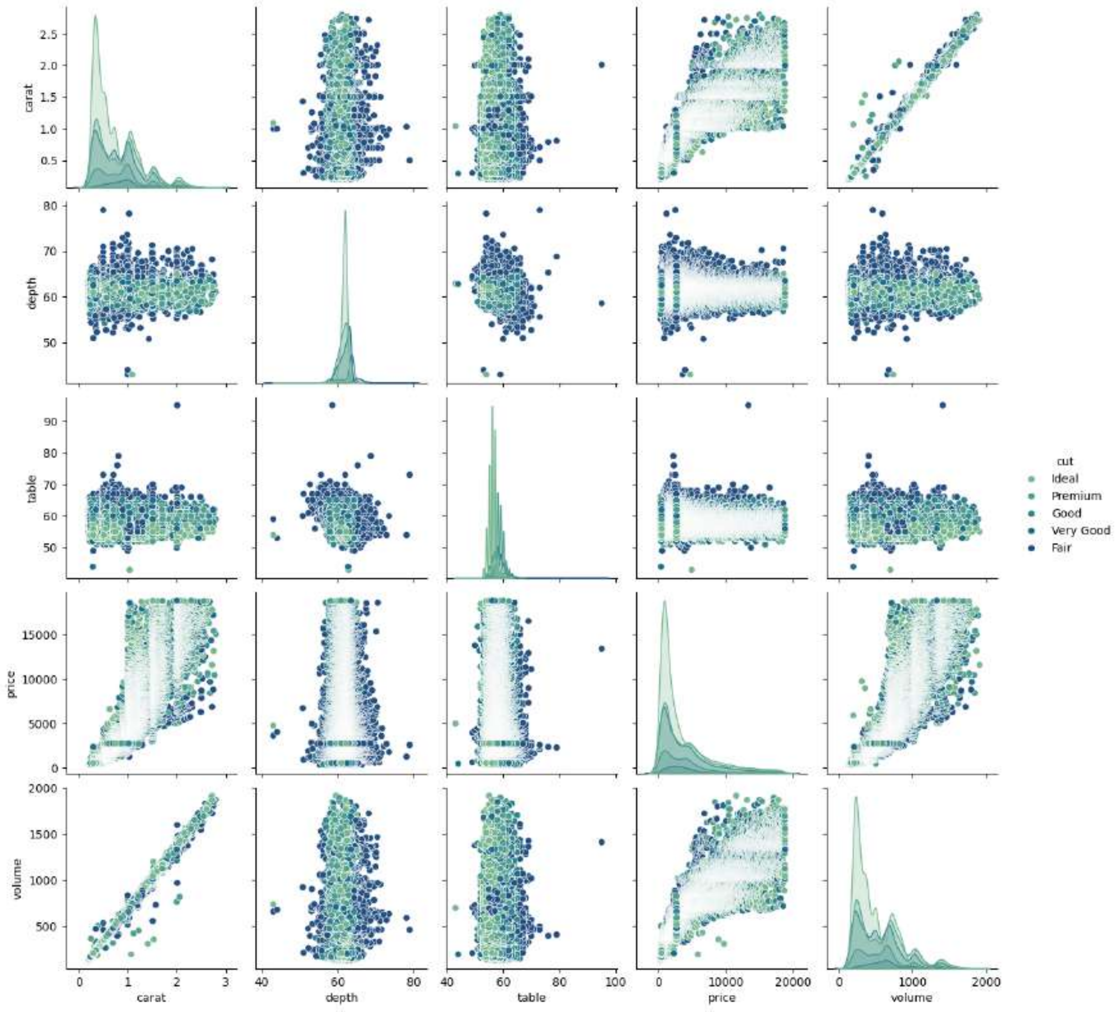
[6]:
```
data = data.drop(data[data["volume"]>=3000].index)
data = data.drop(data[data["carat"]>=3].index)
data = data.drop(["x", "y", "z"], axis=1)
data.shape
```

```
[6]: (53877, 8)
```

```
[7]: data.describe()
```

```
[7]:
              carat         depth         table         price        volume
count  53877.000000  53877.000000  53877.000000  53877.000000  53877.000000
mean       0.795843     61.748913     57.456054   3922.937357    542.500277
std        0.468913      1.430678      2.233336   3976.773245    317.122365
min        0.200000     43.000000     43.000000    326.000000    132.818093
25%        0.400000     61.000000     56.000000    949.000000    272.942022
50%        0.700000     61.800000     57.000000   2399.000000    480.814891
75%        1.040000     62.500000     59.000000   5313.000000    715.271532
max        2.800000     79.000000     95.000000  18823.000000   1911.284003
```

```
[8]: ax = sns.pairplot(data, hue="cut", palette="crest")
```



1 Polynomial regression

1.1 Univariate polynomial regression: carat-dependent

[9]:
```
X = data['carat'].to_numpy().reshape(-1, 1)
y = data['price'].to_numpy()
```

Split the data into 3 sets: training, validation, and testing.

[10]:
```
# Split the test set
X, X_test, y, y_test = train_test_split(X, y, test_size = 0.1, random_state=2)
# Split the training-validation set
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size = 0.333,␣
↪random_state=2)
```

Define the model and fit the data.

[11]:
```
# Create the polynominal transformation
poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(X_train)

# Create and fit the transformed training data into the model
lin_regr = LinearRegression(fit_intercept = False)
lin_regr.fit(X_train_poly, y_train)
y_train_pred = lin_regr.predict(X_train_poly)

# Transform and fit test data
X_val_poly = poly.transform(X_val)
y_val_pred = lin_regr.predict(X_val_poly)
```

Error evaluation for training and validation set.

[12]:
```
tr_error = mean_squared_error(y_train, y_train_pred)
val_error = mean_squared_error(y_val, y_val_pred)
tr_r2 = r2_score(y_train, y_train_pred)
val_r2 = r2_score(y_val, y_val_pred)

print('The training MSE error is: ', tr_error)    # print the training error
print('The training R^2 error is: ', tr_r2)
print('The validation MSE error is: ', val_error)    # print the testing error
print('The validation R^2 error is: ', val_r2)
```

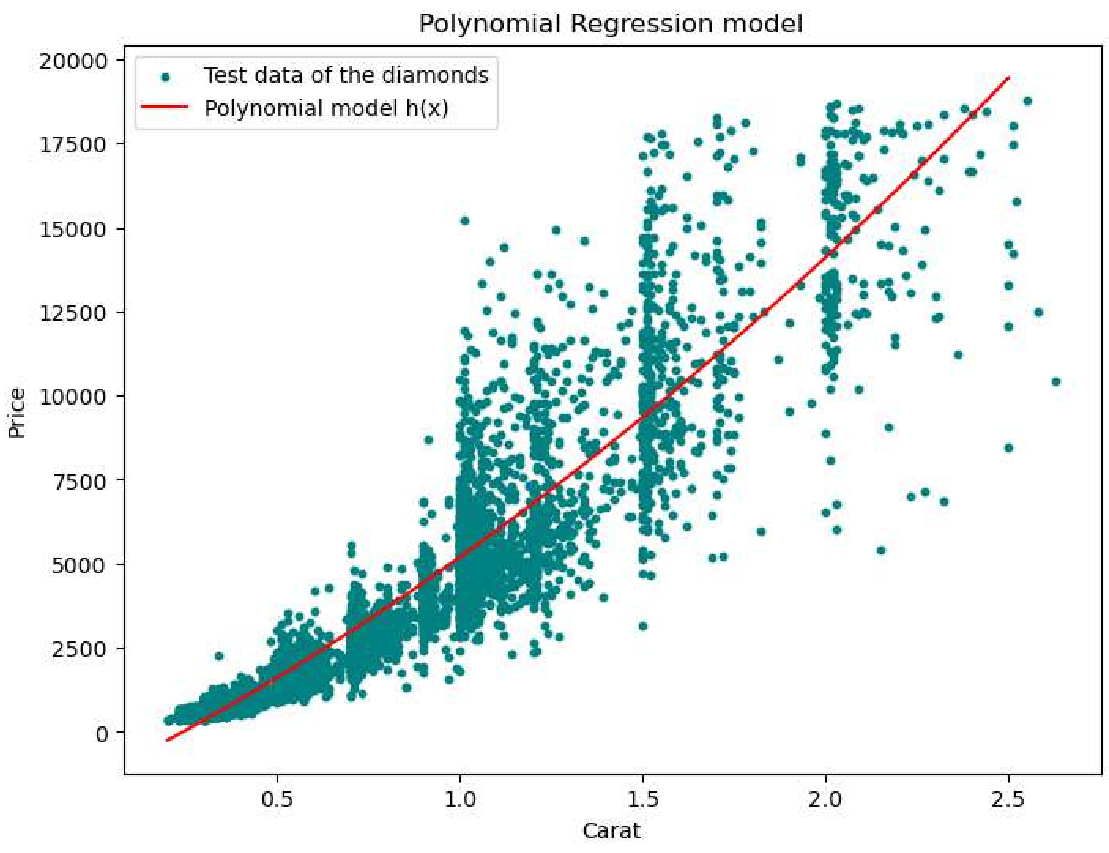
```
The training MSE error is:  2228881.4264446446
The training R^2 error is:  0.8584784711509137
The validation MSE error is:  2229900.192454173
The validation R^2 error is:  0.8613163858500533
```

Final error: error evaluation for testing set.

[13]:
```
# Transform and fit test data
X_test_poly = poly.transform(X_test)
y_test_pred = lin_regr.predict(X_test_poly)

# Calculate the errors
ts_error = mean_squared_error(y_test, y_test_pred)
ts_r2 = r2_score(y_test, y_test_pred)

print('The testing MSE error is: ', ts_error)    # print the testing error
print('The testing R^2 error is: ', ts_r2)
```

```
The testing MSE error is:  2220439.52702327
The testing R^2 error is:  0.8559137214466572
```

[14]:
```
## visualize the model you have learnt, you are supposed to see the datapoints␣
 ↪and the fitted h(x), a straignt line

plt.figure(figsize=(8, 6))    # create a new figure with size 8*6

plt.scatter(X_test, y_test, color="teal", s=10, label="Test data of the␣
 ↪diamonds")
X_fit = np.linspace(0.2, 2.5, 100)    # generate samples
plt.plot(X_fit, lin_regr.predict(poly.transform(X_fit.reshape(-1, 1))),'r',␣
 ↪label="Polynomial model h(x)")    # plot the polynomial regression model

plt.xlabel('Carat') # define label for the horizontal axis
plt.ylabel('Price') # define label for the vertical axis

plt.title('Polynomial Regression model') # define the title of the plot
plt.legend(loc='best') # define the location of the legend

plt.show()    # show the plot
```



1.2 Multivariate polynomial regression: carat-dependent & volume-dependent

[15]:
```
# Include both carat and volume to the data
X = data[['carat','volume']].to_numpy()
y = data['price'].to_numpy()
```

Split the data into 3 sets: training, validation, and testing.

[16]:
```
# Split the test set
X, X_test, y, y_test = train_test_split(X, y, test_size = 0.1, random_state=2)
# Split the training-validation set
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size = 0.333,␣
 ↪random_state=2)
```

Define the model and fit the data.

[17]:
```
# Create the polynominal transformation
poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(X_train)

# Create and fit the transformed training data into the model
```

```python
lin_regr = LinearRegression(fit_intercept = False)
lin_regr.fit(X_train_poly, y_train)
y_train_pred = lin_regr.predict(X_train_poly)

# Transform and fit test data
X_val_poly = poly.transform(X_val)
y_val_pred = lin_regr.predict(X_val_poly)
```

Error evaluation for **training** and **validation** set.

[18]:
```python
tr_error = mean_squared_error(y_train, y_train_pred)
val_error = mean_squared_error(y_val, y_val_pred)
tr_r2 = r2_score(y_train, y_train_pred)
val_r2 = r2_score(y_val, y_val_pred)

print('The training MSE error is: ', tr_error)    # print the training error
print('The training R^2 error is: ', tr_r2)
print('The validation MSE error is: ', val_error)    # print the testing error
print('The validation R^2 error is: ', val_r2)
```

```
The training MSE error is:  2166557.8259721794
The training R^2 error is:  0.8624356718873889
The validation MSE error is:  2158954.170942284
The validation R^2 error is:  0.8657287136780536
```

Final error: error evaluation for **testing** set.

[19]:
```python
# Transform and fit test data
X_test_poly = poly.transform(X_test)
y_test_pred = lin_regr.predict(X_test_poly)

# Calculate the errors
ts_error = mean_squared_error(y_test, y_test_pred)
ts_r2 = r2_score(y_test, y_test_pred)

print('The testing MSE error is: ', ts_error)    # print the testing error
print('The testing R^2 error is: ', ts_r2)
```

```
The testing MSE error is:  2131017.2121359585
The testing R^2 error is:  0.8617164142986489
```

2 Multilayer perceptron (MLP)

[20]:
```python
X = data[['carat','volume']].to_numpy()
y = data['price'].to_numpy()
```

Split the data into 3 sets: training, validation, and testing.

```
[21]: # Split the test set
X, X_test, y, y_test = train_test_split(X, y, test_size = 0.1, random_state=2)
# Split the training-validation set
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size = 0.333,␣
 ↪random_state=2)
```

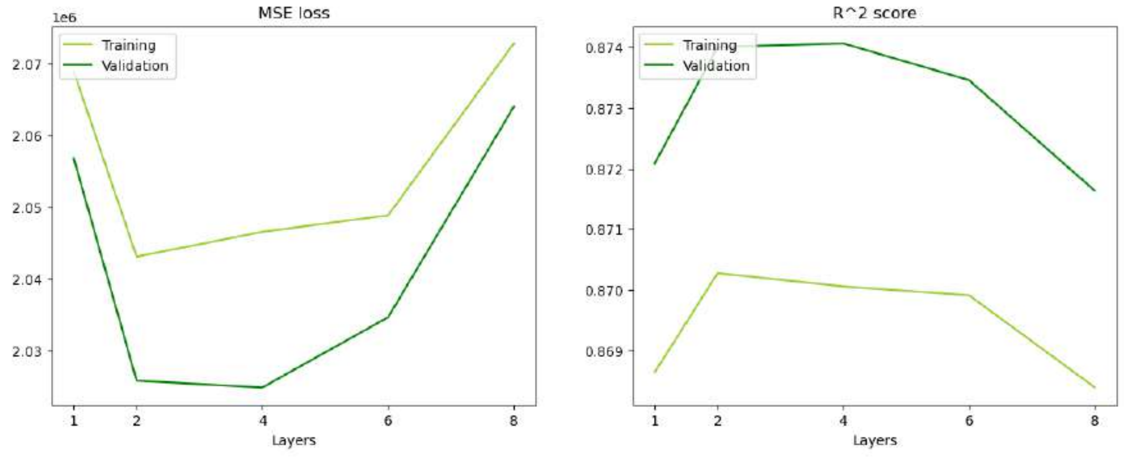
Define the model and fit the data.

```
[22]: ## define a list of values for the number of hidden layers
num_layers = [1,2,4,6,8]    # number of hidden layers
num_neurons = 15  # number of neurons in each layer

# we will use this variable to store the resulting training errors␣
 ↪corresponding to different hidden-layer numbers
mlp_tr_errors = []
mlp_val_errors = []
mlp_tr_r2 = []
mlp_val_r2 = []

for i, num in enumerate(num_layers):
    hidden_layer_sizes = tuple([num_neurons]*num) # size (num of neurons) of␣
 ↪each layer stacked in a tuple

    # YOUR CODE HERE
    mlp_regr = MLPRegressor(hidden_layer_sizes, max_iter = 1000, random_state =␣
 ↪42)
    mlp_regr.fit(X_train, y_train)

    ## Evaluate the trained MLP on both training set and validation set
    y_pred_train = mlp_regr.predict(X_train)    # predict on the training set
    tr_error = mean_squared_error(y_train, y_pred_train)    # calculate the␣
 ↪training error
    tr_r2 = r2_score(y_train, y_pred_train)

    y_pred_val = mlp_regr.predict(X_val) # predict values for the validation␣
 ↪data
    val_error = mean_squared_error(y_val, y_pred_val) # calculate the␣
 ↪validation error
    val_r2 = r2_score(y_val, y_pred_val)

    mlp_tr_errors.append(tr_error)
    mlp_val_errors.append(val_error)
    mlp_tr_r2.append(tr_r2)
    mlp_val_r2.append(val_r2)

print('The training MSE error is: ', mlp_tr_errors)    # print the training␣
 ↪error
```

```
print('The training R^2 error is: ', mlp_tr_r2)
print('The validation MSE error is: ', mlp_val_errors)    # print the testing␣
 ↪error
print('The validation R^2 error is: ', mlp_val_r2)
```

```
The training MSE error is:  [2068894.3885410798, 2043110.395315478,
2046573.7186627768, 2048870.6494591692, 2072813.2483786286]
The training R^2 error is:  [0.8686367550019597, 0.8702738946442161,
0.8700539929441108, 0.8699081506601252, 0.8683879292775515]
The validation MSE error is:  [2056802.6687181217, 2025843.0628625345,
2024896.0705675427, 2034672.0752730106, 2064018.6584391766]
The validation R^2 error is:  [0.8720817960120667, 0.8740072588858042,
0.8740661549361683, 0.8734581583679412, 0.8716330138030821]
```

[23]:
```
# Visualize data
fig, axes = plt.subplots(1, 2, figsize=(14,5)) # create a figure with two axes␣
 ↪(1 row,2 columns) on it

axes[0].plot(num_layers, mlp_tr_errors, label = 'Training', color='yellowgreen')
axes[0].plot(num_layers, mlp_val_errors, label = 'Validation', color='green')
axes[0].set_xticks(num_layers)
axes[0].legend(loc = 'upper left')
axes[0].set_xlabel("Layers")
axes[0].set_title("MSE loss")

axes[1].plot(num_layers, mlp_tr_r2, label = 'Training', color='yellowgreen')
axes[1].plot(num_layers, mlp_val_r2, label = 'Validation', color='green')
axes[1].set_xticks(num_layers)
axes[1].legend(loc = 'upper left')
axes[1].set_xlabel("Layers")
axes[1].set_title("R^2 score")

plt.show()
```



We will use the network with **4 layers**. Now, we apply the model to the testing set.

[24]:
```
# Model with 4 layer
num = 4
hidden_layer_sizes = tuple([num_neurons]*num) # size (num of neurons) of each␣
 ↪layer stacked in a tuple

# Refit the model
mlp_regr = MLPRegressor(hidden_layer_sizes, max_iter = 1000, random_state = 42)
mlp_regr.fit(X_train, y_train)

# Fit the testing data
y_pred_test = mlp_regr.predict(X_test) # predict values for the validation data
test_error = mean_squared_error(y_test, y_pred_test) # calculate the validation␣
 ↪error
test_r2 = r2_score(y_test, y_pred_test)

# Print the error
print('The training MSE error is: ', test_error)    # print the testing error
print('The training R^2 error is: ', test_r2)
```

```
The training MSE error is:  2033726.1682279143
The training R^2 error is:  0.8680297159142407
```