Diamond Values

June 29, 2024

1 Introduction

The objective of this project is to create a website that determines the price of a diamond based on its characteristics: carat, cut, color, clarity, price, depth, table, length (x), width (y), and depth (z). However, in situations where it is necessary to quickly estimate the value of a diamond, it is not feasible to consider all these characteristics. Therefore, a study of the dataset is required to identify the minimum characteristics necessary for an accurate price estimation of a diamond.

To carry out this study, we will use the CRISP-DM (Cross-Industry Standard Process for Data Mining) project model. CRISP-DM has six stages of project planning: business understanding, data understanding, data processing, modeling, evaluation, and implementation. All these processes will be followed during the study of the Diamonds dataset.

2 Step 1: Business Understanding

The first step of CRISP-DM is business understanding; we need to understand exactly what the client needs us to do. To achieve this, we will use two strategies to solve the problem. The first is creating an ERD (Entity-Relationship Diagram), and the second is creating an agile BDD (Behavior-Driven Development) process.

1) To get a clearer view of the database, we will start by creating an Entity-Relationship Diagram like the one shown below.

<img src="DER.png" alt="descrição_da_imagem">

2) We will use BDD to create scenarios for our project, as shown below:

Scenario 1: Estimate a price for the diamond

AS a user,

I want to find out the value of a diamond,

SO THAT I am not deceived when selling my diamond.

3 Step 2: Data Understanding

With the business understanding already established, we will now move on to the second stage of CRISP-DM: Data Understanding. For this process, we will use the “Diamonds” dataset, obtained from the Kaggle platform. This dataset is in CSV format and contains 10 columns and 53,940 rows.

3.1 Database Features

- `Carat`: The carat weight of the diamond.
- `Cut`: The type of cut of the diamond.
- `Color`: The color of the diamond.
- `Clarity`: The purity/clarity of the diamond.
- `Price`: The price of the diamond.
- `Depth`: The total depth percentage of the diamond.
- `Table`: The width of the diamond's top relative to its widest point.
- `x`: The length of the diamond.
- `y`: The width of the diamond.
- `z`: The depth of the diamond.

4 Step 3: Data Preparation

Next, we will address process 3 of CRISP-DM: data preparation. At this stage, we will import some libraries in Python and investigate the existence of incorrect or missing values in the dataset. If we find any undesirable or missing values, we will perform the necessary treatment to ensure they do not negatively influence the results of the project's research.

[1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import math
import numpy as np
from sklearn.impute import KNNImputer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import r2_score
```

[2]:
```
# Change the database path
path = r"DataBases\Diamonds_values_faltantes.csv"
diamonds = pd.read_csv(fr"{path}")

diamonds
```

[2]:
```
       carat      cut color clarity  depth  table   price     x     y     z
0       0.23    Ideal     E     SI2   61.5   55.0   326.0  3.95  3.98  2.43
1       0.21  Premium     E     NaN   61.2   61.0   326.0  3.89  3.84  2.31
2       0.23     Good     E     VS1   56.9   65.0   327.0  4.05  4.07  2.31
3       0.29  Premium     I     VS2   62.4   58.0   334.0   NaN  4.23  2.63
4       0.31     Good     J     SI2   63.3   58.0   335.0  4.34  4.35  2.75
…        …       …    …      …      …      …      …     …     …     …
53935   0.72    Ideal     D     SI1   62.5   57.0  2757.0  5.75  5.76  3.50
53936   0.72     Good     D     SI1   63.1   55.0  2757.0  5.69  5.75  3.61
53937   0.70      NaN     D     SI1   62.8   60.0  2757.0  5.66  5.68  3.56
53938   0.86  Premium     H     SI2   61.0   58.0  2757.0  6.15  6.12  3.74
```

```
53939    0.75     Ideal     D      SI2    62.2    55.0  2757.0  5.83  5.87  3.64

[53940 rows x 10 columns]
```

Below is the amount of missing values per column

```
[3]: counter = {}
     for x in range(diamonds.shape[1]):
         column_name = diamonds.columns[x]
         counter[column_name] = diamonds.shape[0] - len(diamonds[column_name].
      ↪dropna())

     counter_df = pd.DataFrame(list(counter.items()), columns=['Column', 'Quantity␣
      ↪of NaN'])
     counter_df
```

```
[3]:     Column  Quantity of NaN
     0     carat             1649
     1       cut             1556
     2     color             1540
     3   clarity             1476
     4     depth             1421
     5     table             1369
     6     price             1340
     7         x             1308
     8         y             1253
     9         z             1257
```

4.1 Processing the database using the algorithm K-NN (K-Nearest Neighbors)

Placing measurements equal to 0 of length, width and/or depth of a diamond as NaN

```
[4]: for x in range(diamonds.shape[0]):
         for y in range(7, diamonds.shape[1]):
             if diamonds.iloc[x, y] == 0: diamonds.iloc[x, y] = np.nan
             elif diamonds.iloc[x, y] >= 30: diamonds.iloc[x, y] = np.nan
     diamonds
```

```
[4]:        carat      cut  color clarity  depth  table   price     x     y     z
     0       0.23    Ideal      E     SI2   61.5   55.0   326.0  3.95  3.98  2.43
     1       0.21  Premium      E     NaN   61.2   61.0   326.0  3.89  3.84  2.31
     2       0.23     Good      E     VS1   56.9   65.0   327.0  4.05  4.07  2.31
     3       0.29  Premium      I     VS2   62.4   58.0   334.0   NaN  4.23  2.63
     4       0.31     Good      J     SI2   63.3   58.0   335.0  4.34  4.35  2.75
     …        …       …     …      …     …      …      …     …     …     …
     53935   0.72    Ideal      D     SI1   62.5   57.0  2757.0  5.75  5.76  3.50
     53936   0.72     Good      D     SI1   63.1   55.0  2757.0  5.69  5.75  3.61
```

```
53937    0.70       NaN      D      SI1    62.8    60.0  2757.0  5.66  5.68  3.56
53938    0.86   Premium      H      SI2    61.0    58.0  2757.0  6.15  6.12  3.74
53939    0.75     Ideal      D      SI2    62.2    55.0  2757.0  5.83  5.87  3.64

[53940 rows x 10 columns]
```

To perform the calculation of the distance of the diamond, in which we want to find out the price, we will use the calculation of the euclidean distance:

$$d(A, B) = \sqrt{\sum_{i=1}^{n}(A_i - B_i)^2}$$

- A is the axis of the point that we want to predict the value.
- B is the axis of an already defined point.

[5]:
```
'''KNN for categorical values'''
encoder = OrdinalEncoder()
diamonds_encoder = encoder.fit_transform(diamonds)

knn_imputer = KNNImputer(n_neighbors = round(math.log(diamonds.shape[0])),␣
 ↪metric = "nan_euclidean")
test = knn_imputer.fit_transform(diamonds_encoder)


diamonds_imputer = pd.DataFrame(test, columns = diamonds.columns)
diamonds_imputer = encoder.inverse_transform(diamonds_imputer)
diamonds = pd.DataFrame(diamonds_imputer.tolist(), columns = diamonds.columns)

diamonds
```

[5]:
```
        carat       cut  color clarity  depth   table   price     x     y     z
0        0.23     Ideal      E     SI2   61.5    55.0   326.0  3.95  3.98  2.43
1        0.21   Premium      E     VS1   61.2    61.0   326.0  3.89  3.84  2.31
2        0.23      Good      E     VS1   56.9    65.0   327.0  4.05  4.07  2.31
3        0.29   Premium      I     VS2   62.4    58.0   334.0  4.22  4.23  2.63
4        0.31      Good      J     SI2   63.3    58.0   335.0  4.34  4.35  2.75
…        …       …   …      …     …    …      …      …    …    …     …
53935    0.72     Ideal      D     SI1   62.5    57.0  2757.0  5.75  5.76  3.50
53936    0.72      Good      D     SI1   63.1    55.0  2757.0  5.69  5.75  3.61
53937    0.70     Ideal      D     SI1   62.8    60.0  2757.0  5.66  5.68  3.56
53938    0.86   Premium      H     SI2   61.0    58.0  2757.0  6.15  6.12  3.74
53939    0.75     Ideal      D     SI2   62.2    55.0  2757.0  5.83  5.87  3.64

[53940 rows x 10 columns]
```

Saving the database already clean and without missing values

```
[6]: path = r"DataBases\Diamonds_limpa.csv"
     try:
         pd.read_csv(f"{path}")
         print(f"This dataframe already exists in the directory: {path}")
     except FileNotFoundError:
         diamonds.to_csv(fr"{path}", index = False)
         print(f'''Clean database added to directory:
               {path}
               successfully!!''')
```

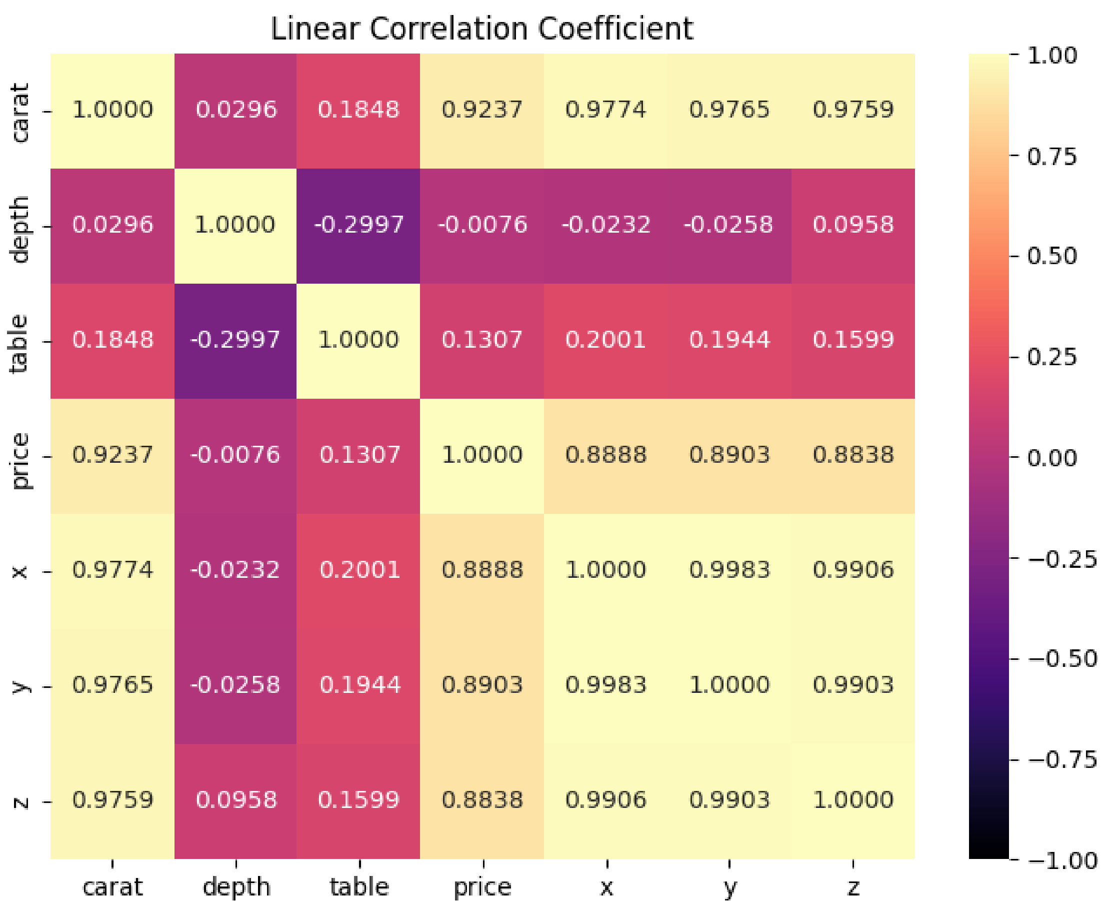
```
This dataframe already exists in the directory: DataBases\Diamonds_limpa.csv
```

Finally, we try to save the database without any missing or incorrect value in the "Databases" folder. If we succeed, this indicates that the database was not previously saved. Otherwise, the database was already saved.

5 Step 4: Modeling

For the fourth stage of CRISP-DM, we will be focusing on the central part of the project, which consists in the in-depth study of the database. At this stage, our main objective is to identify the main factors that influence the price of a diamond.

```
[7]: plt.figure(figsize = (8, 6))
     sns.heatmap((diamonds[["carat", "depth", "table", "price", "x", "y", "z"]]).
      ↪corr(), vmin = -1, vmax = 1, annot = True, cmap = 'magma', fmt = ".4f")
     plt.title("Linear Correlation Coefficient")
     plt.show()
```



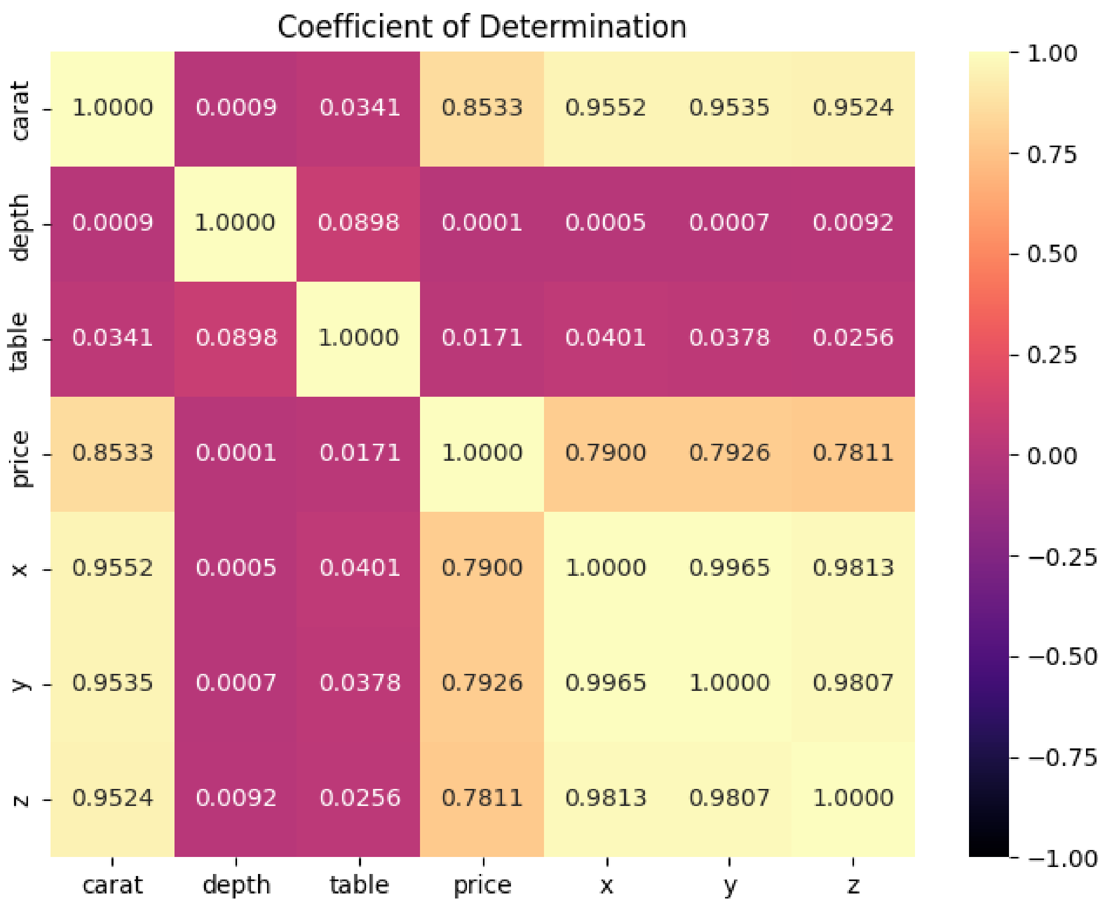
	carat	depth	table	price	x	y	z
carat	1.0000	0.0296	0.1848	0.9237	0.9774	0.9765	0.9759
depth	0.0296	1.0000	-0.2997	-0.0076	-0.0232	-0.0258	0.0958
table	0.1848	-0.2997	1.0000	0.1307	0.2001	0.1944	0.1599
price	0.9237	-0.0076	0.1307	1.0000	0.8888	0.8903	0.8838
x	0.9774	-0.0232	0.2001	0.8888	1.0000	0.9983	0.9906
y	0.9765	-0.0258	0.1944	0.8903	0.9983	1.0000	0.9903
z	0.9759	0.0958	0.1599	0.8838	0.9906	0.9903	1.0000

Analysis of the heatmap above based on price(price): - We can conclude that the price (price) does not have a good correlation with the total percentage of diamond (depth) and also does not have a high correlation with the table, being an inverse proportional correlation of -0.0086 with the depth, and a proportional relation of 0.13 with the table. - We can also conclude that the price has a good linear correlation with the carat (carat) of 0.92, x (length) of 0.89, y (width) of 0.89 and z (depth) of 0.88.

Based on this heatmap analysis, we can conclude that the higher the carat (carat), x(length), y(width) and z(depth), the higher the price (price) of the diamond.

Please, there may be some cases of having a diamond with a very high carat but with a low price, just as there may be diamonds with a low carat but with a high price. This can also happen with x(length), y(width) and z(depth), because of that we ask the following question, how much carat(carat), x(length), y(width) and z(depth) can determine the value of the diamond? To answer this, we need to take the Determination Coefficient.

[8]:
```
plt.figure(figsize = (8, 6))
sns.heatmap((diamonds[["carat", "depth", "table", "price", "x", "y", "z"]]).
  ↪corr()**2, vmin = -1, vmax = 1, annot = True, cmap = 'magma', fmt = ".4f")
plt.title("Coefficient of Determination")
plt.show()
```



	carat	depth	table	price	x	y	z
carat	1.0000	0.0009	0.0341	0.8533	0.9552	0.9535	0.9524
depth	0.0009	1.0000	0.0898	0.0001	0.0005	0.0007	0.0092
table	0.0341	0.0898	1.0000	0.0171	0.0401	0.0378	0.0256
price	0.8533	0.0001	0.0171	1.0000	0.7900	0.7926	0.7811
x	0.9552	0.0005	0.0401	0.7900	1.0000	0.9965	0.9813
y	0.9535	0.0007	0.0378	0.7926	0.9965	1.0000	0.9807
z	0.9524	0.0092	0.0256	0.7811	0.9813	0.9807	1.0000

Analysis of the heatmap above based on price(price):

When we analyze the heatmap above, we can see that we can define the price of diamond with greater reliability using the variable carat (carat) with reliability of 85%, this means that although we can say that the higher the carat of the diamond, the higher its price, unfortunately this rule is only valid for 85% of the data.

For x(length), y(width) and z(depth), this reliability is only 79% for length and width, and 78% for depth, which is not a strong determination, and therefore may be disregarded if the categorical variables, Be able to accurately set the price of the diamond.

Below we are performing the separation process of the diamonds database. So that the machine learn process is more effective.

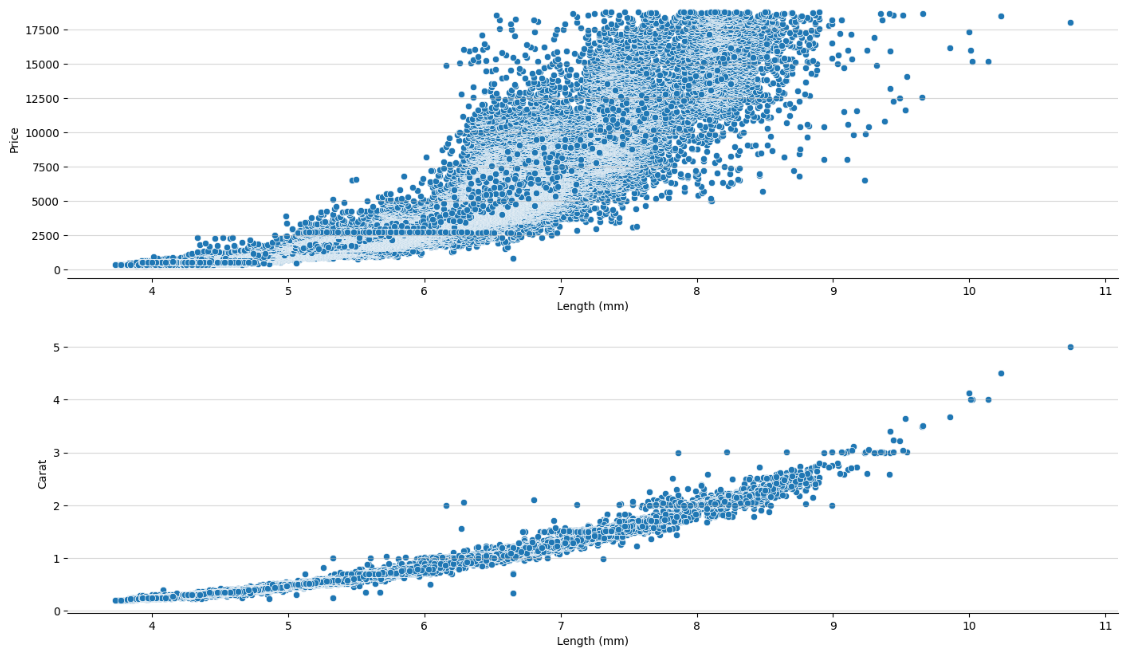
- Cut has 5 types of classification Ideal, Premium, Good, Very Good and Fair
- Codor has 7 types classification E, I, J, H, F, G and D
- Clarity has 8 types of classification SI2, SI1, VS1, VS2, VVS2, VVS1, I1 and IF

5.1 Analysis of the price ratio of the numerical columns

Important Information: - 1 Carat equals 200mg - 1 Point equals 0.01 carats

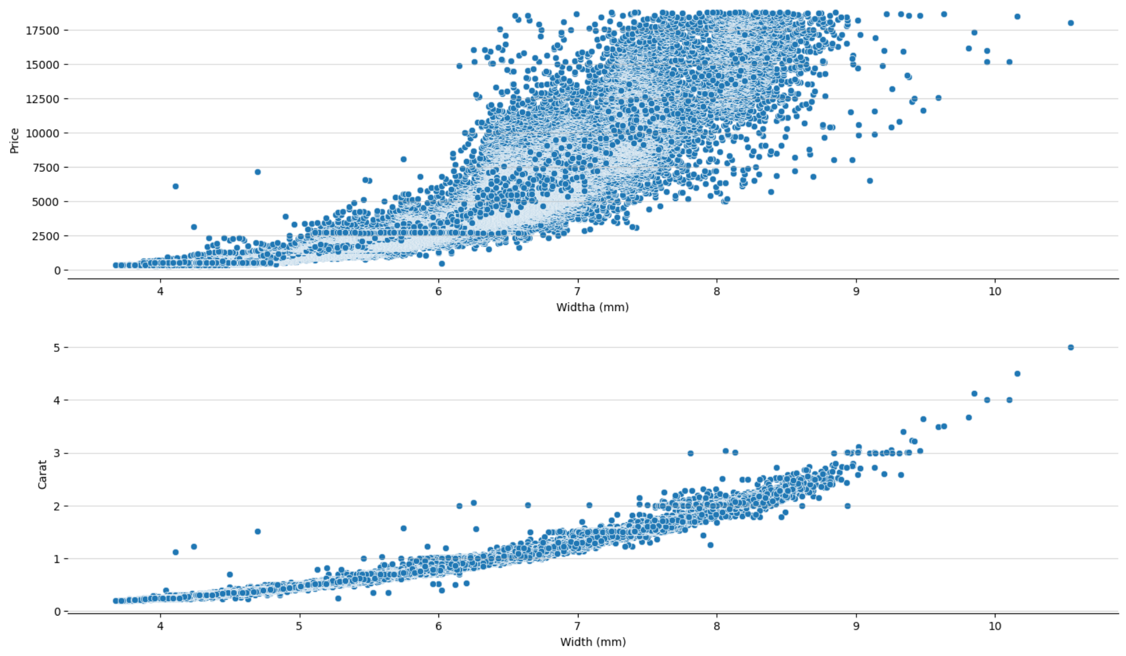
The chart below compares the ratio of the length of a diamond to the carat and price

[9]:
```
plt.figure(figsize=(17, 10))

plt.subplot(2, 1, 1)
sns.scatterplot(data=diamonds, x =  "x", y = "price")
plt.xlabel("Length (mm)")
plt.ylabel("Price")
plt.gca().spines["right"].set_visible(False)
plt.gca().spines["top"].set_visible(False)
plt.gca().spines["left"].set_visible(False)
plt.gca().spines["left"].set_visible(False)
plt.grid(axis = "y", alpha = 0.5)

plt.subplot(2, 1, 2)
sns.scatterplot(data=diamonds, x = "x", y = "carat")
plt.xlabel("Length (mm)")
plt.ylabel("Carat")
plt.gca().spines["right"].set_visible(False)
plt.gca().spines["top"].set_visible(False)
plt.gca().spines["left"].set_visible(False)
plt.grid(axis = "y", alpha = 0.5)

plt.show()
```



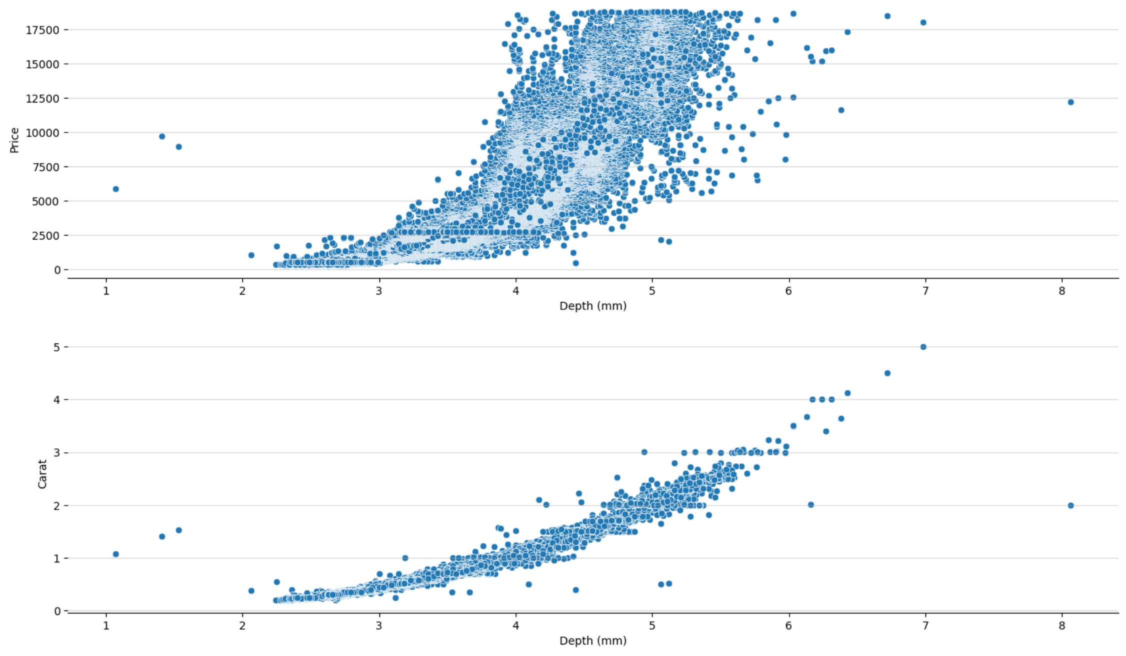
The chart below compares the ratio of diamond width to carat and price

[10]:
```python
plt.figure(figsize=(17, 10))

plt.subplot(2, 1, 1)
sns.scatterplot(diamonds, x = "y", y = "price")
plt.xlabel("Widtha (mm)")
plt.ylabel("Price")
plt.gca().spines["right"].set_visible(False)
plt.gca().spines["top"].set_visible(False)
plt.gca().spines["left"].set_visible(False)
plt.grid(axis = "y", alpha = 0.5)

plt.subplot(2, 1, 2)
sns.scatterplot(diamonds, x = "y", y = "carat")

plt.xlabel("Width (mm)")
plt.ylabel("Carat")
plt.gca().spines["right"].set_visible(False)
plt.gca().spines["top"].set_visible(False)
plt.gca().spines["left"].set_visible(False)
plt.grid(axis = "y", alpha = 0.5)

plt.show()
```



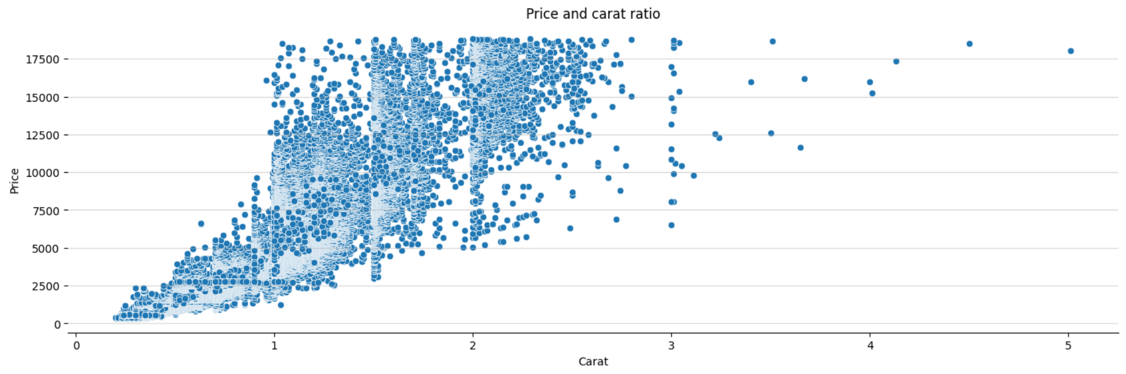
The chart below compares the ratio of diamond depth to carat and price

[11]:
```
plt.figure(figsize=(17, 10))

plt.subplot(2, 1, 1)
sns.scatterplot(diamonds, x = "z", y = "price")
plt.xlabel("Depth (mm)")
plt.ylabel("Price")
plt.gca().spines["right"].set_visible(False)
plt.gca().spines["top"].set_visible(False)
plt.gca().spines["left"].set_visible(False)
plt.grid(axis = "y", alpha = 0.5)

plt.subplot(2, 1, 2)
sns.scatterplot(diamonds, x = "z", y = "carat")
plt.xlabel("Depth (mm)")
plt.ylabel("Carat")
plt.gca().spines["right"].set_visible(False)
plt.gca().spines["top"].set_visible(False)
plt.gca().spines["left"].set_visible(False)
plt.grid(axis = "y", alpha = 0.5)

plt.show()
```



The chart below compares the ratio of the carat of a diamond to the price

```
[12]: plt.figure(figsize=(17, 5))
      sns.scatterplot(diamonds, x = "carat", y = "price")
      plt.xlabel("Carat")
      plt.ylabel("Price")
      plt.title("Price and carat ratio")
      plt.gca().spines["right"].set_visible(False)
      plt.gca().spines["top"].set_visible(False)
      plt.gca().spines["left"].set_visible(False)
      plt.grid(axis = "y", alpha = 0.5)

      plt.show()
```



Based on the graphs presented, it is evident that the length, width and depth of a diamond have a more reliable relationship with its weight in carats than with its price. Therefore, when determining the value of a diamond with the minimum of measurements required, we can rely on the carat data provided. Physical dimensions such as length, width and depth give a more accurate indication of the weight of the diamond than its monetary value.

However, it is important to note that this does not mean that we cannot use the measurements of length, width and depth to estimate the value of a diamond. On the contrary, the more information we have, the more accurate the estimate of the price of the diamond. However, if we have to choose the minimum of information to estimate the value of a diamond, we can say that the carat is sufficient for this evaluation.

There are 3 ways to estimate the price of diamond for the program user:

1) Request the diamond mass to the customer, and thus perform the calculation:

$$\text{Carat} = \frac{\text{Mass (mg)}}{200}$$

2) When the user provides the diamond points:

$$Carat = \frac{\text{points (pt)}}{100}$$

3) For the second way to estimate the carat of diamond, 4 things are required: Length (mm), Width (mm), Depth (mm) and density ($\frac{mm}{mm^3}$). With this we will use the calculation of the density of an object, to thus calculate the mass of the diamond:

$$Density = \frac{Mass}{Volume} \rightarrow Mass = Density \times Volume$$

However we have a problem, we do not have the volume of the diamond, however for this, we will calculate the volume of an object, being:

$$Volume = Length \times Width \times Depth$$

Replacing in the formula then, will be:

$$Mass = Length \times Width \times Depth \times density$$

Now we have to find the carat of the diamond, for this, we will use the way 1 to estimate the calculation of the diamond:

$$Carat = \frac{Mass(mg)}{200}$$

Getting into the general formula:

$$Carat = \frac{Density \times Volume}{200}$$

OR

$$Carat = \frac{Length \times Width \times Depth \times Density}{200}$$

5.2 Price relation with the categorical columns

```
[13]: diamonds.describe()
```

```
[13]:               carat         depth         table         price             x  \
count  53940.000000  53940.000000  53940.000000  53940.000000  53940.000000
mean       0.797388     61.751368     57.439803   3931.331201      5.730953
std        0.473151      1.406329      2.201157   3977.347590      1.117914
min        0.200000     43.000000     43.000000    326.000000      3.730000
25%        0.400000     61.100000     56.000000    949.000000      4.710000
50%        0.700000     61.800000     57.000000   2405.000000      5.700000
75%        1.040000     62.500000     59.000000   5358.000000      6.540000
max        5.010000     79.000000     95.000000  18823.000000     10.740000

                  y            z
count  53940.000000  53940.00000
mean       5.733120      3.53925
std        1.109959      0.69108
min        3.680000      1.07000
25%        4.720000      2.91000
50%        5.710000      3.53000
75%        6.540000      4.03000
```

```
max        10.540000      8.06000
```

[14]:

```
description = diamonds.describe()
price = [f"until ${description.iloc[4, 3]}",
    f"until ${description.iloc[5, 3]}",
    f"until ${description.iloc[6, 3]}",
    f"greater than ${description.iloc[6, 3]}"]

carat = [f"until ${description.iloc[4, 0]}",
    f"until ${description.iloc[5, 0]}",
    f"until ${description.iloc[6, 0]}",
    f"greater than ${description.iloc[6, 0]}"]


def agrupamento(diamonds, coluna, index_coluna: list):
    if coluna == "price":
        coluna_aux = 3
    else:
        coluna_aux = 0

    description = diamonds.describe()
    cut = pd.DataFrame({"Fair": [0.0 for x in range(4)],
                        "Good": [0.0 for x in range(4)],
                        "Very Good": [0.0 for x in range(4)],
                        "Premium": [0.0 for x in range(4)],
                        "Ideal": [0.0 for x in range(4)]},
                        index = index_coluna)

    color = pd.DataFrame({"J": [0.0 for x in range(4)],
                        "D": [0.0 for x in range(4)],
                        "I": [0.0 for x in range(4)],
                        "E": [0.0 for x in range(4)],
                        "F": [0.0 for x in range(4)],
                        "H": [0.0 for x in range(4)],
                        "G": [0.0 for x in range(4)]},
                        index = index_coluna)

    clarity = pd.DataFrame({"I1": [0.0 for x in range(4)],
                            "IF": [0.0 for x in range(4)],
                            "VVS1": [0.0 for x in range(4)],
                            "VVS2": [0.0 for x in range(4)],
                            "VS1": [0.0 for x in range(4)],
                            "VS2": [0.0 for x in range(4)],
                            "SI2": [0.0 for x in range(4)],
                            "SI1": [0.0 for x in range(4)]},
                            index = index_coluna)
    for intervalo in ["25%", "50%", "75%", "max"]:
        if intervalo == "25%":
            diamonds_aux = diamonds[diamonds[coluna] <= diamonds.
↪describe()[coluna][intervalo]].reset_index()

        elif intervalo == "50%":
            diamonds_aux = diamonds[diamonds[coluna] > diamonds.
↪describe()[coluna]["25%"]].reset_index()
            diamonds_aux = diamonds_aux[diamonds_aux[coluna] <= diamonds.
↪describe()[coluna][intervalo]].reset_index()

        elif intervalo == "75%":
            diamonds_aux = diamonds[diamonds[coluna] > diamonds.
↪describe()[coluna]["50%"]].reset_index()
            diamonds_aux = diamonds_aux[diamonds_aux[coluna] <= diamonds.
↪describe()[coluna][intervalo]].reset_index()

        else:
            diamonds_aux = diamonds[diamonds[coluna] > diamonds.
↪describe()[coluna]["75%"]].reset_index()

        describe = diamonds.describe()[coluna][intervalo]

        for x in range(diamonds_aux.shape[0]):
            for y in range(cut.shape[1]):
                if diamonds_aux.loc[x, "cut"] == cut.columns[y]:
                    try:
                        cut.loc[f"until ${describe}", cut.columns[y]] += 1.0
                    except KeyError:
                        cut.loc[f"greater than ${description.iloc[6,␣
↪coluna_aux]}", cut.columns[y]] += 1.0
                    break

            for y in range(color.shape[1]):
                if diamonds_aux.loc[x, "color"] == color.columns[y]:
                    try:
                        color.loc[f"until ${describe}", color.columns[y]] += 1.0
                    except KeyError:
                        color.loc[f"greater than ${description.iloc[6,␣
↪coluna_aux]}", color.columns[y]] += 1.0
                    break

            for y in range(clarity.shape[1]):
                if diamonds_aux.loc[x, "clarity"] == clarity.columns[y]:
                    try:
                        clarity.loc[f"until ${describe}", clarity.columns[y]]␣
↪+= 1.0
                    except (KeyError, KeyboardInterrupt):
                        clarity.loc[f"greater than ${description.iloc[6,␣
↪coluna_aux]}", clarity.columns[y]] += 1.0
                    break

    soma_cut = [sum(cut.iloc[:, x]) for x in range(cut.shape[1])]
    soma_color = [sum(color.iloc[:, x]) for x in range(color.shape[1])]
    soma_clarity = [sum(clarity.iloc[:, x]) for x in range(clarity.shape[1])]

    for x in range(4):
        for y in range(cut.shape[1]):
            cut.iloc[x, y] = round(cut.iloc[x, y] / soma_cut[y], 4).
↪astype(float)
        for y in range(color.shape[1]):
            color.iloc[x, y] = round(color.iloc[x, y] / soma_color[y], 4).
↪astype(float)
        for y in range(clarity.shape[1]):
            clarity.iloc[x, y] = round(clarity.iloc[x, y] / soma_clarity[y], 4).
↪astype(float)

    if "carat" == coluna:
        cut.index = [f"until {description.iloc[4, 0]}",
                    f"until {description.iloc[5, 0]}",
                    f"until {description.iloc[6, 0]}",
                    f"greater than {description.iloc[6, 0]}"]

        color.index = [f"until {description.iloc[4, 0]}",
                    f"until {description.iloc[5, 0]}",
                    f"until {description.iloc[6, 0]}",
                    f"greater than {description.iloc[6, 0]}"]

        clarity.index = [f"until {description.iloc[4, 0]}",
                    f"until {description.iloc[5, 0]}",
                    f"until {description.iloc[6, 0]}",
                    f"greater than {description.iloc[6, 0]}"]


    return cut, color, clarity
```

[15]:
```
cut, color, clarity = agrupamento(diamonds, "price", price)
cut_carat, color_carat, clarity_carat = agrupamento(diamonds, "carat", carat)
```

The above command creates six tables that display, in percentages, the amount of diamonds with certain characteristics within specific value ranges. In addition, three other similar tables are generated, but instead of grouping the data by price, they are grouped by weight in carats (carat).

[16]:
```
cut
```

[16]:
```
                        Fair    Good  Very Good  Premium   Ideal
until $949.0          0.0532  0.2163     0.2579   0.2142  0.2886
until $2405.0         0.2854  0.2196     0.2114   0.2191  0.2926
until $5358.0         0.4208  0.3346     0.2786   0.2549  0.2027
greater than $5358.0  0.2405  0.2295     0.2520   0.3119  0.2161
```

[17]:
```
cut_carat
```

[17]:
```
                     Fair    Good  Very Good  Premium   Ideal
until 0.4          0.0423  0.1925     0.2452   0.2360  0.3300
until 0.7          0.2168  0.2339     0.2260   0.1895  0.2713
until 1.04         0.4554  0.3646     0.2951   0.2496  0.1850
greater than 1.04  0.2854  0.2089     0.2337   0.3249  0.2137
```

By analyzing the above charts, we can identify which cuts tend to have higher carat weights and prices, and which cuts tend to have lower carat weights and prices. We observed that the cut influences the weight in carats more than the price. However, the cut can help us in determining the range of values in which the diamond fits. Once the carat is set, it becomes clearer to determine a price range for the diamond, thus allowing a more accurate estimate of its value.

[18]:
```
color
```

[18]:
```
                           J       D       I       E       F       H       G
until $949.0          0.1540  0.2800  0.2155  0.2863  0.2416  0.2386  0.2551
until $2405.0         0.1852  0.3030  0.1631  0.3109  0.2762  0.1709  0.2538
until $5358.0         0.2754  0.2505  0.2695  0.2495  0.2643  0.2761  0.2041
greater than $5358.0  0.3854  0.1665  0.3518  0.1532  0.2179  0.3144  0.2870
```

[19]:
```
color_carat
```

[19]:
```
                        J       D       I       E       F       H       G
until 0.4          0.1183  0.3228  0.1941  0.3316  0.2673  0.2340  0.2730
until 0.7          0.1292  0.3038  0.1411  0.2992  0.2663  0.1594  0.2374
until 1.04         0.2175  0.2523  0.2204  0.2536  0.2902  0.2367  0.2328
greater than 1.04  0.5349  0.1211  0.4444  0.1156  0.1761  0.3699  0.2568
```

Unlike cut (cut) charts, we can notice a clearer separation in the value ranges when analyzing the colors of diamonds. This allows us to observe more precisely which colors have a higher tendency to be high carat and which tend to be lower carat. We also identify which diamond colors are associated with higher prices and which tend to have lower values. As with the cut, color can be used to estimate the price of the diamond because it gives a clearer indication of price and carat trends.

[20]:
```
clarity
```

[20]:
```
                            I1      IF    VVS1    VVS2     VS1     VS2     SI2  \
until $949.0            0.0738  0.3494  0.3860  0.3531  0.2853  0.2813  0.1161
until $2405.0           0.2496  0.4077  0.3726  0.3001  0.2698  0.2558  0.1698
until $5358.0           0.4298  0.0938  0.1201  0.1308  0.1761  0.1776  0.4333
greater than $5358.0    0.2468  0.1491  0.1213  0.2161  0.2688  0.2853  0.2809

                           SI1
until $949.0            0.2196
until $2405.0           0.2189
until $5358.0           0.3190
greater than $5358.0    0.2425
```

[21]:
```
clarity_carat
```

[21]:
```
                         I1      IF    VVS1    VVS2     VS1     VS2     SI2  \
until 0.4            0.0255  0.5999  0.5525  0.4317  0.3080  0.2954  0.0870
until 0.7            0.1191  0.2132  0.2659  0.2786  0.2729  0.2538  0.1553
until 1.04           0.3021  0.0903  0.0993  0.1480  0.2062  0.2247  0.3540
greater than 1.04    0.5532  0.0967  0.0823  0.1418  0.2128  0.2261  0.4037

                        SI1
until 0.4            0.1815
until 0.7            0.2415
until 1.04           0.3157
greater than 1.04    0.2613
```

As we saw in cut(cut) and color(color), clarity(clarity) is also a good feature to be able to find out the price(price) of the diamond, since just like the other characteristics, it has a greater precision when setting a value for carat(carat) than for the price of diamond. We also identify which diamond clearances are associated with higher prices and which tend to have lower values. As with the cut, color can be used to estimate the price of the diamond because it gives a clearer indication of price and carat trends.

However, we can state that the categorical columns of the database are essential to estimate the value of the diamond. They provide crucial information that allows an estimate of the price of the jewel, helping to determine the value of the diamond. Therefore, these columns should be considered mandatory variables for the user when performing this analysis.

6 Step 5: evaluation

In the penultimate step of CRISP-DM, it is crucial to evaluate the performance of the adopted prediction model. In this context, we will use the scikit-learn library to employ the coefficient of determination (R 2). This coefficient helps us to evaluate the accuracy of the model, both to replace missing values in the database and to estimate the value of diamonds provided by users.

[22]:
```
# Transforming categorical variables into numerical
encoder = OrdinalEncoder()
diamonds_encoder = encoder.fit_transform(diamonds.drop(columns=['price']))
```

```
# Putting these changes in the database
X = pd.DataFrame(diamonds_encoder.tolist(), columns = list(diamonds.columns).
 ↪remove("price"))
y = diamonds['price']

# Divide data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Create and train the KNN model  # K value based on the number of observations␣
 ↪log
knn = KNeighborsRegressor(n_neighbors = int(round(math.log(diamonds.shape[0]),␣
 ↪0)))
knn.fit(X_train, y_train)

# Make predictions in the test set
y_pred = knn.predict(X_test)

# evaluate the model
r2 = r2_score(y_test, y_pred)
print(f'R² (determination coefficient): {r2 * 100:.2f}%')
```

```
R² (determination coefficient): 90.98%
```

Based on the above program, we can conclude that the reliability of the KNN algorithm is 90.98%. This means that when predicting the price of a user-supplied diamond, the program has an accuracy of 90.98%.

7 Step 6: implementation

Finally, the implementation is the last stage of CRISP-DM. At this stage, we put into practice the project studied. Now that we know the level of reliability of the algorithm and the minimum variables that are important for estimating the price of the diamond, we can implement our study in the final project. This means that we can use all the knowledge and model developed to predict the price of a diamond effectively and accurately. So the final step is to carry out the program that predict the value of the diamond.