Data Visualization with Diamond Data

Zachary Tarell

April 17, 2020

Diamond Data Set - Description

Prices of >50,000 round cut diamonds

Description:

A dataset containing the prices and other attributes of almost 54,000 diamonds. The variables are as follows:

Format:

A data frame with 53940 rows and 10 variables:

price

```
price in US dollars ($326–$18,823)
```

carat

```
weight of the diamond (0.2–5.01)
```

cut

```
quality of the cut (Fair, Good, Very Good, Premium, Ideal)
```

color

```
diamond colour, from J (worst) to D (best)
```

clarity

```
a measurement of how clear the diamond is (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1,
IF (best))
```

x

```
length in mm (0–10.74)
```

y

```
width in mm (0–58.9)
```

z

```
depth in mm (0–31.8)
```

Click here for links to data and description:

Link (https://www.kaggle.com/shivam2503/diamonds) for data

Link (https://vincentarelbundock.github.io/Rdatasets/doc/ggplot2/diamonds.html) for description

In [108]:
```
import numpy as np
import pandas as pd
import seaborn as sb
from sklearn import datasets
df = pd.read_csv('diamonds.csv', usecols=['carat', 'cut', 'color', 'clarity', 'pri
ce', 'x', 'y', 'z'])
print('\nI cut the DEPTH and TABLES columns out from the beginning as a way of Dat
a Cleaning.')
print('For my algorithms, there was not much use for them.')
print('So, now down to 8 variables as shown below.\n')
print(df.head())
# print the dimensions of the data
print('\nDimensions of data frame:', df.shape)
```

```
I cut the DEPTH and TABLES columns out from the beginning as a way of Data Clean
ing.
For my algorithms, there was not much use for them.
So, now down to 8 variables as shown below.

   carat      cut color clarity  price     x     y     z
0   0.23    Ideal     E     SI2    326  3.95  3.98  2.43
1   0.21  Premium     E     SI1    326  3.89  3.84  2.31
2   0.23     Good     E     VS1    327  4.05  4.07  2.31
3   0.29  Premium     I     VS2    334  4.20  4.23  2.63
4   0.31     Good     J     SI2    335  4.34  4.35  2.75

Dimensions of data frame: (53940, 8)
```

Data Cleaning

In [109]:
```
# check for NAs
print('\nAfter only reading in 8 variables instead of 10 - I check and delete any
NAs for rest of Data Cleaning (there were none).', "\n")
print(df.isnull().sum())
# delete rows with NAs
df = df.dropna()
# show total of NA rows after deletion
print('\nTotal number of NAs after deleting them = ', df.isnull().sum().sum())
# print the new dimensions
print('\nDimensions of data frame:', df.shape)
```

```
After only reading in 8 variables instead of 10 - I check and delete any NAs for
rest of Data Cleaning (there were none).

carat      0
cut        0
color      0
clarity    0
price      0
x          0
y          0
z          0
dtype: int64

Total number of NAs after deleting them =  0

Dimensions of data frame: (53940, 8)
```

Data Exploration

In [110]:
```
print('\nExploring data with the head function:\n')
df.head()
```

```
Exploring data with the head function:
```

Out[110]:

	carat	cut	color	clarity	price	x	y	z
0	0.23	Ideal	E	SI2	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	335	4.34	4.35	2.75

In [111]:
```
print('\nDescribe the diamonds Length(x), Width(y), and Depth(z) in millimeters:\n
')
df.loc[:, ['x', 'y', 'z']].describe()
```

```
Describe the diamonds Length(x), Width(y), and Depth(z) in millimeters:
```

Out[111]:

	x	y	z
count	53940.000000	53940.000000	53940.000000
mean	5.731157	5.734526	3.538734
std	1.121761	1.142135	0.705699
min	0.000000	0.000000	0.000000
25%	4.710000	4.720000	2.910000
50%	5.700000	5.710000	3.530000
75%	6.540000	6.540000	4.040000
max	10.740000	58.900000	31.800000

In [112]:
```
print('\nShow all the data types of the diamonds:\n')
df.dtypes
```

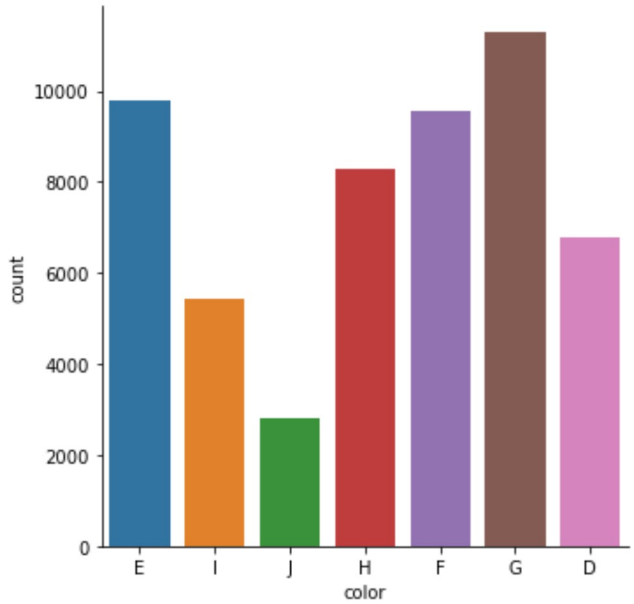
```
Show all the data types of the diamonds:
```

Out[112]:
```
carat       float64
cut          object
color        object
clarity      object
price         int64
x           float64
y           float64
z           float64
dtype: object
```

Graphs

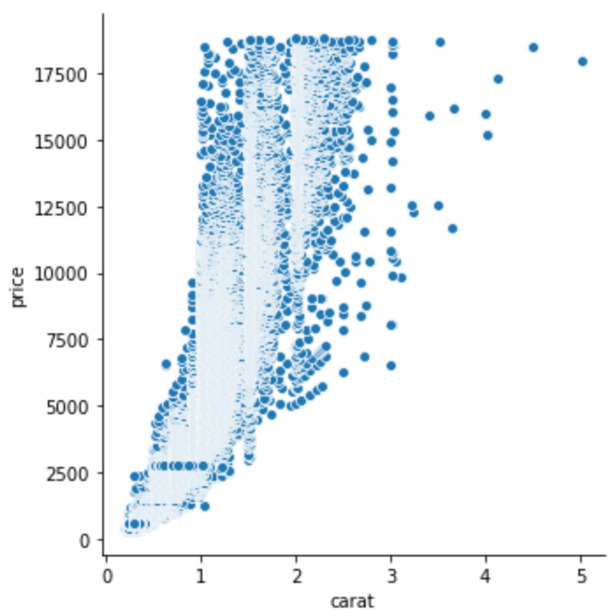
In [113]:
```
sb.catplot("color", kind='count', data=df)
```

Out[113]: <seaborn.axisgrid.FacetGrid at 0x17cc2bb0>



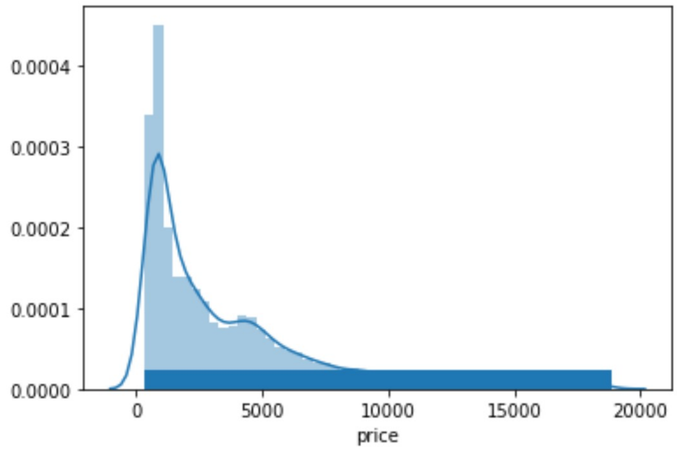
In [114]:
```
sb.relplot(x='carat', y='price', data=df)
```

Out[114]: <seaborn.axisgrid.FacetGrid at 0x160991d8>



In [115]:
```
sb.distplot(df['price'], kde=True, rug=True)
```

Out[115]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a743b50>
```



In [116]:
```
df.cut = df.cut.astype('category').cat.codes
df.color = df.color.astype('category').cat.codes
df.clarity = df.clarity.astype('category').cat.codes
df.dtypes
```

Out[116]:
```
carat      float64
cut           int8
color         int8
clarity       int8
price        int64
x          float64
y          float64
z          float64
dtype: object
```

Linear Regression

In [198]:
```
# train test split
from sklearn.model_selection import train_test_split
X = df.iloc[:, 0:6]
y = df.iloc[:, 7]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_st
ate=1234)

print('train size:', X_train.shape)
print('test size:', X_test.shape)
```

```
train size: (43152, 6)
test size: (10788, 6)
```

In [199]:
```
# train the algorithm
from sklearn.linear_model import LinearRegression

linreg = LinearRegression()
linreg.fit(X_train, y_train)
```

Out[199]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [200]:
```
# retrieve coefficients
print('intercept:', linreg.intercept_)
print('coefficients:', linreg.coef_)
```

```
intercept: 0.395048744830511 56
coefficients: [ 2.62957474e-01 -1.16650482e-02 -6.16243427e-04 -5.88806332e-04
 -6.12501432e-06  5.22006602e-01]
```

In [201]:
```
# make predictions

y_pred = linreg.predict(X_test)
```

In [202]:
```
# evaluation
from sklearn.metrics import mean_squared_error, r2_score
print('mse=', mean_squared_error(y_test, y_pred))
print('correlation=', r2_score(y_test, y_pred))
```

```
mse= 0.012726771763953109
correlation= 0.9738198594250213
```

Train/Test Split for Classification Algorithms

In [225]:
```
# train test split
from sklearn.model_selection import train_test_split
X = df.loc[:, ['x', 'y', 'z']]
y = df.loc[:, ['cut']]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_st
ate=1234)

print('train size:', X_train.shape)
print('test size:', X_test.shape)
```

```
train size: (43152, 3)
test size: (10788, 3)
```

Decision Tree

In [226]:
```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
clf.score(X_train, y_train)
```

Out[226]: 0.9065860215053764

In [227]:
```
# make predictions
pred = clf.predict(X_test)
```

In [228]:
```
# Evaluate
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_scor
e
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred, average='weighted'))
print('recall score: ', recall_score(y_test, pred, average='weighted'))
print('f1 score: ', f1_score(y_test, pred, average='weighted'))
```

```
accuracy score:  0.5613644790507972
precision score:  0.5584763812891341
recall score:  0.5613644790507972
f1 score:  0.5580859869305906
```

In [229]:
```
# confusion matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, pred)
```

Out[229]:
```
array([[ 233,   64,    9,   25,    8],
       [  36,  581,   60,   53,  260],
       [   8,   77, 2851,  813,  530],
       [  20,   91, 1111, 1325,  241],
       [   7,  286,  757,  276, 1066]], dtype=int64)
```

In [230]:
```
# Classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, pred, labels=np.unique(pred)))
```

```
              precision    recall  f1-score   support

           0       0.77      0.69      0.72       339
           1       0.53      0.59      0.56       990
           2       0.60      0.67      0.63      4279
           3       0.53      0.48      0.50      2788
           4       0.51      0.45      0.47      2392

    accuracy                           0.56     10788
   macro avg       0.59      0.57      0.58     10788
weighted avg       0.56      0.56      0.56     10788
```

Logistical Regression

In [231]:
```
from sklearn.linear_model import LogisticRegression
# train a logistic regression model using solver lbfgs
clf = LogisticRegression(solver= 'lbfgs', max_iter=1500)
clf.fit(X_train, y_train.values.ravel())
clf.score(X_train, y_train)
```

Out[231]: 0.4977057842046719

In [232]:
```
# make predictions
pred = clf.predict(X_test)
# Evaluate
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_scor
e
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred, average='weighted'))
print('recall score: ', recall_score(y_test, pred, average='weighted'))
print('f1 score: ', f1_score(y_test, pred, average='weighted', labels=np.unique(pr
ed)))
```

```
accuracy score:  0.5029662588060808
precision score:  0.4500014690974514
recall score:  0.5029662588060808
f1 score:  0.47512763647376727

c:\users\ztare\appdata\local\programs\python\python38-32\lib\site-packages\sklea
rn\metrics\_classification.py:1272: UndefinedMetricWarning: Precision is ill-def
ined and being set to 0.0 in labels with no predicted samples. Use `zero_divisio
n` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [233]:
```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, pred)
```

Out[233]:
```
array([[ 135,    0,   92,   80,   32],
       [   5,    0,  714,  144,  127],
       [   2,    0, 3520,  667,   90],
       [   1,    0, 1140, 1610,   37],
       [   0,    0, 2037,  194,  161]], dtype=int64)
```

In [234]:
```
# Classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, pred, labels=np.unique(pred)))
```

```
              precision    recall  f1-score   support

           0       0.94      0.40      0.56       339
           2       0.47      0.82      0.60      4279
           3       0.60      0.58      0.59      2788
           4       0.36      0.07      0.11      2392

   micro avg       0.50      0.55      0.53      9798
   macro avg       0.59      0.47      0.46      9798
weighted avg       0.50      0.55      0.48      9798
```

Results Analysis

In Linear Regression, I get mse: 0.012726771763953109 and correlation: 0.9738198594250213. These are very good numbers and if you look at the relplot where I compared price to carat, it can be shown that this model is very good for linear regression. Almost all of the data is directly correlated with the higher the carat, size, length, width, depth, etc., the higher the price and better the diamond, overall.

As for the 2 classification algorithms (I chose DT and LogReg) the data had to be manipulated a little bit more to show good results. I manipulated the data by choosing the length, depth, and width of the diamond and compared it directly to the cut, or my target value. I was getting memory overload errors so I decided to compare these 2 algorithms by cutting down on the size of the models. Decision Tree had a higher score of 0.9065860215053764 to the LogReg score of 0.4977057842046719. In fact, it wasn't even close.
Actually neither of the 2 did very well with this data and they both averaged between 50-60% scores. This data was very large and I feel I didn't know enough about hwo to manipulate it to get better results. I had to up my iterations to 1500 for LogReg and there was an f1-score warning that I jsut let pass because the 2nd row (1 index) on the array results was all zero's, f1-score was dividing by zero and popped a warning to let me know. I think if the memory overload errors and being to use price as a target, LogReg would've outperformed DT because of the size of the data, but I was unable to make that work.

Personal Impressions

As for comparison of the R models with the Python, my Linear Regression outperformed with Python as opposed to R. The correlation was better by 2% and the mse was slightly smaller as well. However, the Decision and Classification algorithms in R way outperformed the ones in Python. R was not giving me any problems with the size of my data. R took a little longer to run but was way more powerful in the sense that it didn't have any allocation of memory errors.

Overall, I like Python better because it is easier to classify types and categorically visualize them for me. R is definetely better, in my opinion, for large data sets but is harder to make visualizations. R is easier to make simple math equations because it is written in a way that is very upfront and you don't need to allocate variables as often. But, if it's true that data is only as good as the people reading it, then Python is the best becasue it can do more with the data in terms of outsiders being able to comprehend the results. And as for me only, I like both of them and plan using them for a long time to come.