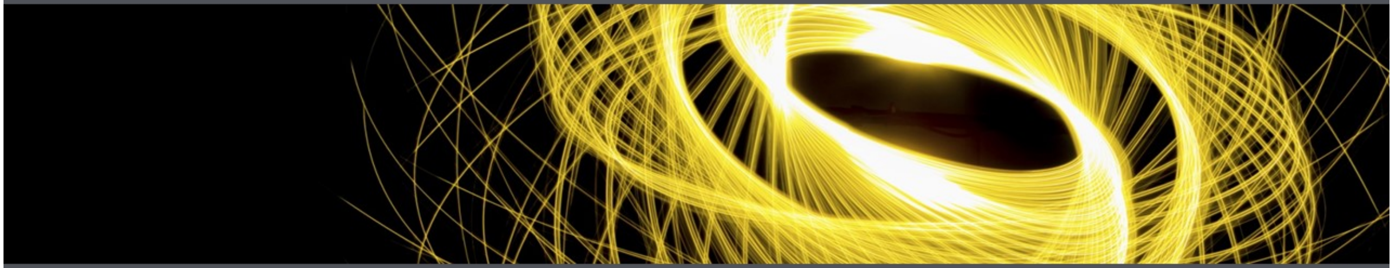
Department of Computer Science

University of Reading

CSMAD21 – Applied Data Science with Python



Data Visualisation 1

University of Reading

Lecture Objectives

- Identify and understand the different types of data
- Implement data discovery and analysis developing graphs via:
  - Pandas
  - Matplotlib
  - Seaborn

Outline

Categorical and Numerical Variables

- **Categorical variables** are variables whose values are one of a finite set of possible values. In pandas these are represented by the Categorical class. We might generate numerical values for each of the unique values in the category.
  - An example of this would be the cut variable in the diamonds data set:
  - Categories:
    - Ideal, Premium, Good, Very Good and Fair

In [1]:

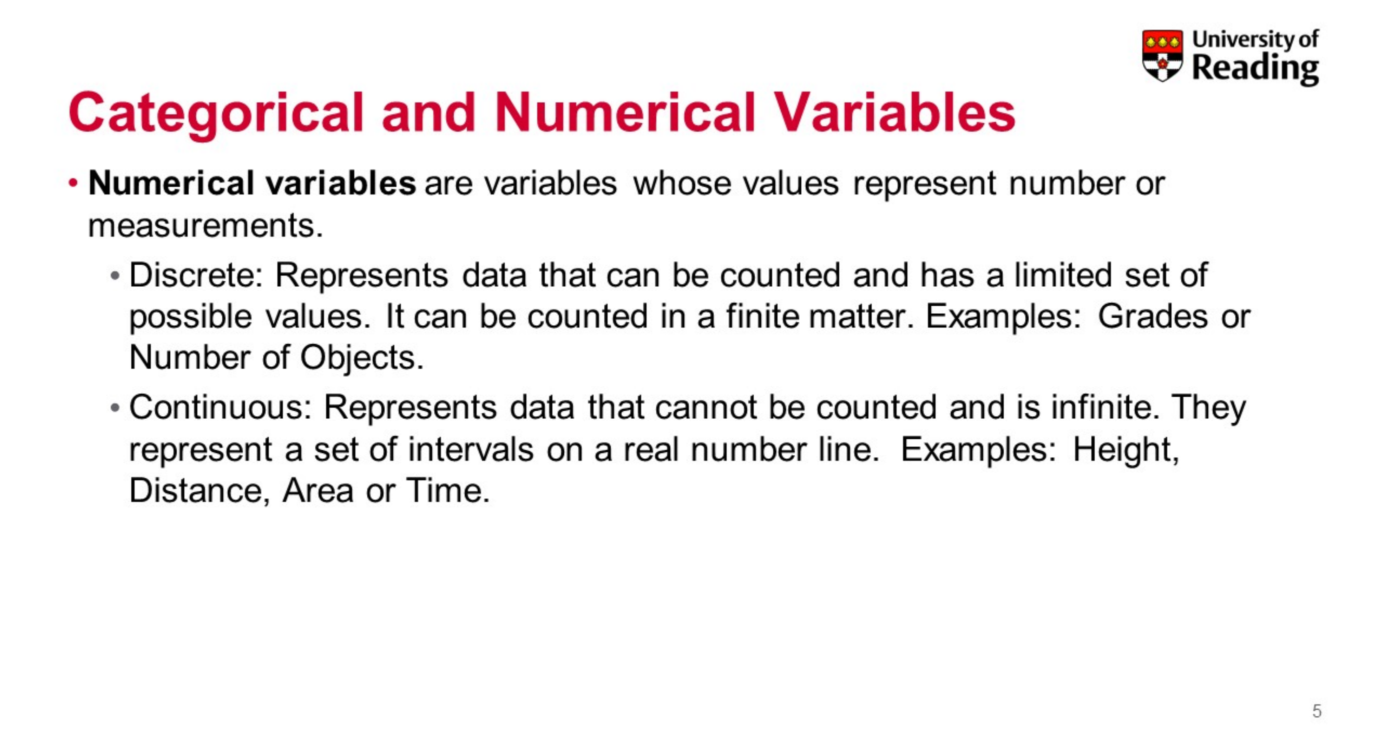
```
## Importing Libraries
import pandas as pd
##Loading Data Set
diamonds = pd.read_csv("Datasets/diamonds.csv")
```

In [2]:

```python
diamonds['cut'].unique()
```

Out[2]:

```
array(['Ideal', 'Premium', 'Good', 'Very Good', 'Fair'], dtype=object)
```



University of Reading

Categorical and Numerical Variables

- **Numerical variables** are variables whose values represent number or measurements.
  - Discrete: Represents data that can be counted and has a limited set of possible values. It can be counted in a finite matter. Examples: Grades or Number of Objects.
  - Continuous: Represents data that cannot be counted and is infinite. They represent a set of intervals on a real number line. Examples: Height, Distance, Area or Time.

5

In [3]:

```python
diamonds.head()
##Numerical - Discrete values
diamonds[['price']].head()
```

Out[3]:

	price
0	326
1	326
2	327
3	334
4	335

In [4]:

```
##Numerical - Continous
diamonds[['depth']].head()
```

Out[4]:

	depth
0	61.5
1	59.8
2	56.9
3	62.4
4	63.3

Simple Plots with Pandas

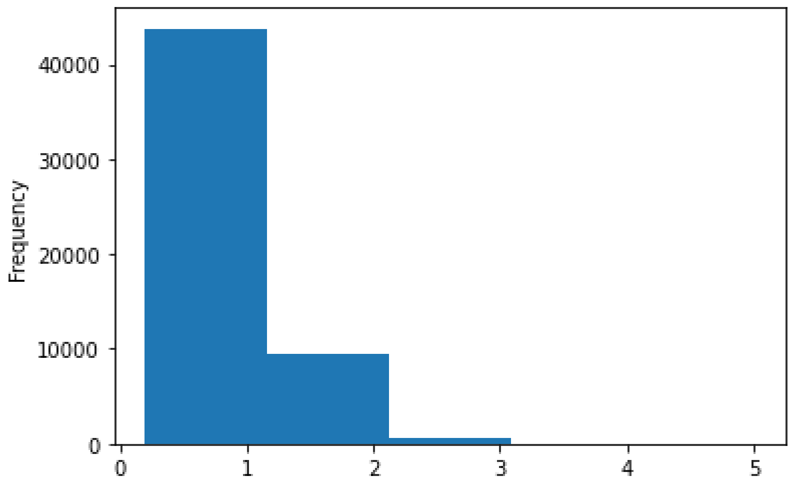
- Pandas has some built-in functionality for quickly generating simple plots from Data Frames. We can plot a histogram, showing the frequency of values in a given column in the data frame:

In [5]:

```
diamonds['carat'].plot.hist(bins=5)
```

Out[5]:

```
<AxesSubplot:ylabel='Frequency'>
```



Simple Plots with Pandas

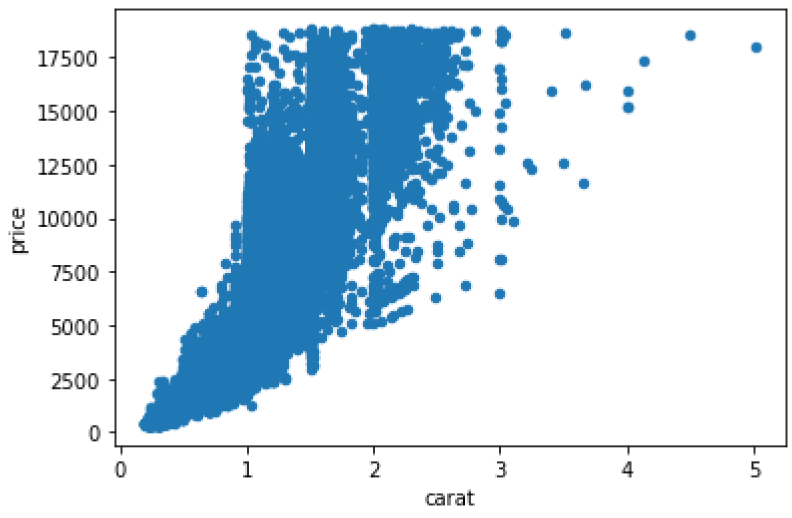
- We can also generate scatter plots to visualise how two variables are related to one another. Here we specify which column of the data frame we would like on each axis.

In [6]:

```
diamonds.plot.scatter(x='carat',y='price')
```

Out[6]:

```
<AxesSubplot:xlabel='carat', ylabel='price'>
```



Simple Plots with Pandas

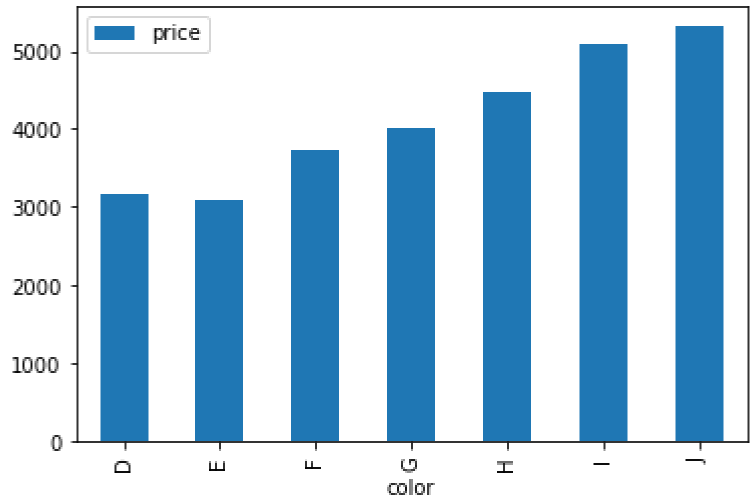
- Given values for a number of different categories we can generate a bar plot. This works well with grouped data.

In [7]:

```python
#diamonds.head()
diamonds_cut = diamonds.groupby('color')
diamonds_cut[['price']].mean().head(10).plot.bar()
```

Out[7]:

```
<AxesSubplot:xlabel='color'>
```



Simple Plots with Pandas

- We can plot information from timeseries.
- The internet traffic time series data has obvious spikes corresponding to day and night. If we are more interested in trends over longer time periods we can smooth the data out by taking a rolling window over 24 hours.

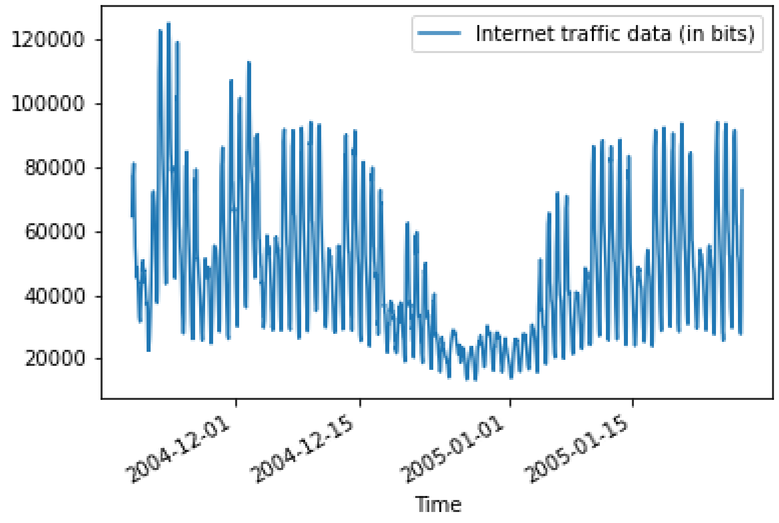
- Data by P. Cortez, M. Rio, M. Rocha and P. Sousa., sourced from Hyndman, R.J. Time Series Data Library, https://github.com/FinYang/tsdl.

In [8]:

```
##Timeseries
internet = pd.read_csv("Datasets/internet.csv", index_col=0,parse_dates=True)
internet.plot()
```

Out[8]:

```
<AxesSubplot:xlabel='Time'>
```

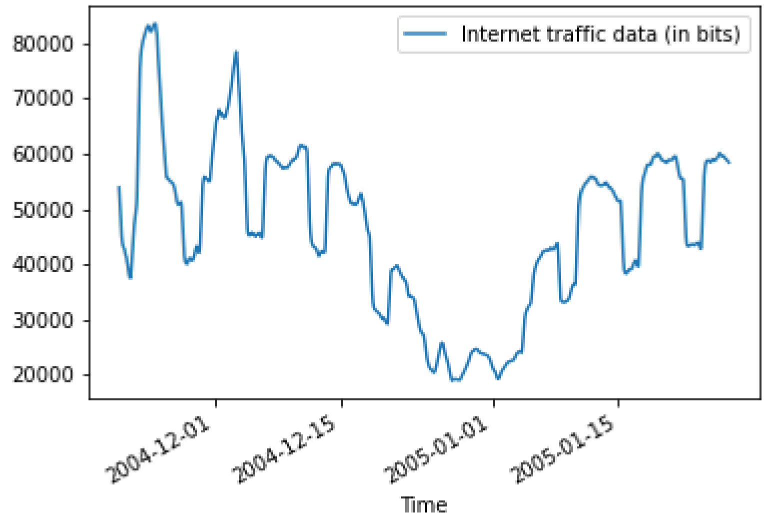


In [9]:

```
##Timeseries 2
internet.rolling(24).mean().plot()
```

Out[9]:

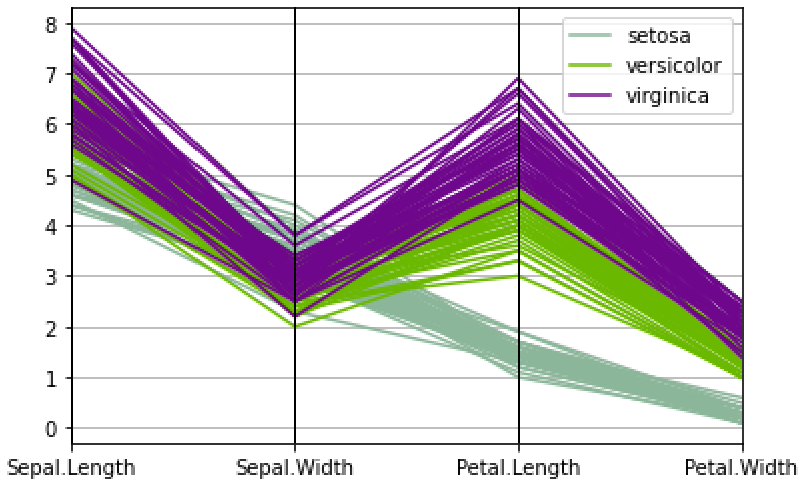
```
<AxesSubplot:xlabel='Time'>
```



Simple Plots with Pandas

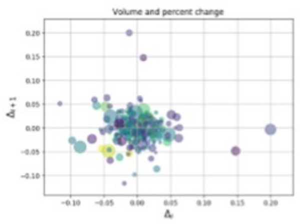
- Parallel Plots
  - One way of visualising data with more than two dimensions is to use a parallel plot, where each variable is shown as an axis side by side, and data points are drawn as lines across the axes.

```
In [10]:
from pandas.plotting import parallel_coordinates
iris = pd.read_csv("Datasets/iris.csv")
parallel_coordinates(iris,'Species')
Out[10]:

<AxesSubplot:>
```



Matplotlib

- Matplotlib is a plotting library for Python. It is included in Anaconda 3, and works from Jupyter notebooks.
  - Low level plotting.
  - Several other Python plotting libraries are based on matplotlib .
  - Plots can be highly customised, but this requires fairly large amounts of code.
- API documentation can be found at https://matplotlib.org/



Matplotlib

- When using matplotlib inside Jupyter, you should import it after first setting an option to display plots inline in the notebook:

```
%matplotlib inline
import matplotlib.pyplot as plt
```

Matplotlib

- The simplest syntax for drawing matplotlib plots is to use various functions to generate plots, and then modifying the resulting plot.
- There is a method to generate a new plot that gives more direct control over the plot and allows you to pass the plot as a parameter to a function:

```
fig, ax = plt.subplots()
```

- This allows you to use methods like ax.set_xticklabels and ax.set_xlim to modify aspects of the plot.

In [11]:

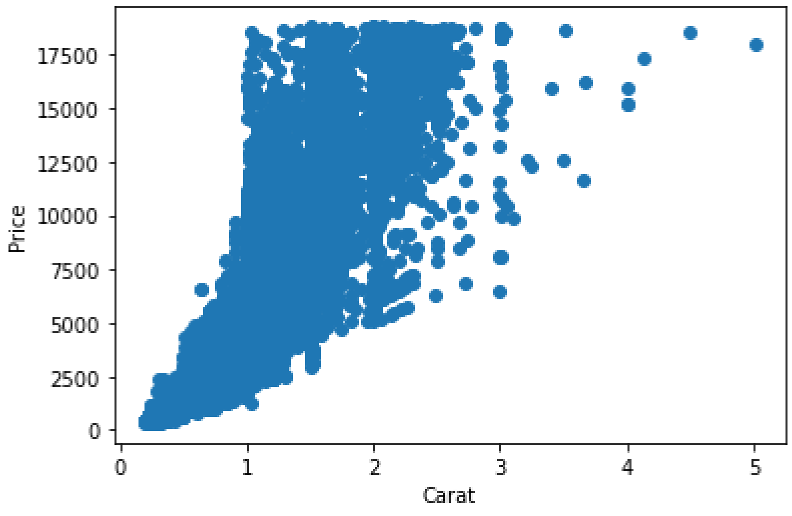
```
## Importing Libraries
#%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd

##Loading Data Set
diamonds = pd.read_csv("Datasets/diamonds.csv")
```

In [12]:

```
##Creating the Plot
plt.scatter(diamonds['carat'], diamonds['price'])
plt.xlabel('Carat')
plt.ylabel('Price')
```

Out[12]:

```
Text(0, 0.5, 'Price')
```



Matplotlib

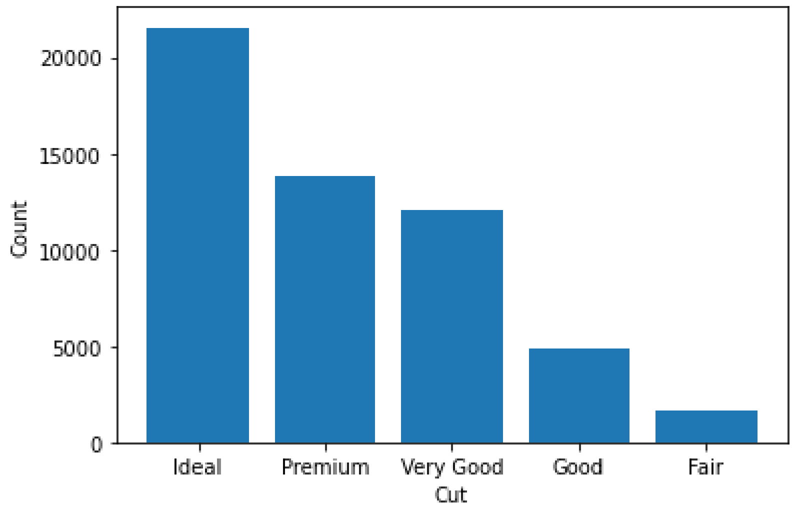
- Bar Plots:
  - To visualise categorical data the most straightforward approach is to use a bar plot, with one bar per category.
  - To generate a bar plot in matplotlib we can use the bar function and specify the categories, and heights for each category.
- Another approach is to use shapes instead of bars, and have the area of the shape represent the value. To show the same information as a bar plot in a different way we can use a Pie chart. This makes it easier to plot data where there is a large difference between the values, as the area of the partition is used to represent the value.

In [13]:

```
##Bar Plot
cc = diamonds['cut'].value_counts()
plt.bar(cc.index,cc)
plt.xlabel('Cut')
plt.ylabel('Count')
display
```

Out[13]:

```
<function IPython.core.display.display(*objs, include=None, exclude=None, me
tadata=None, transient=None, display_id=None, **kwargs)>
```

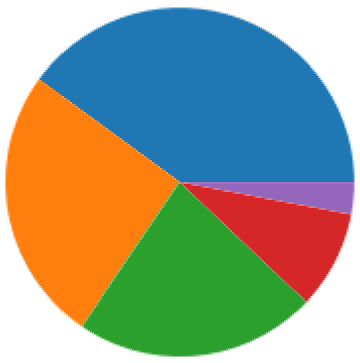


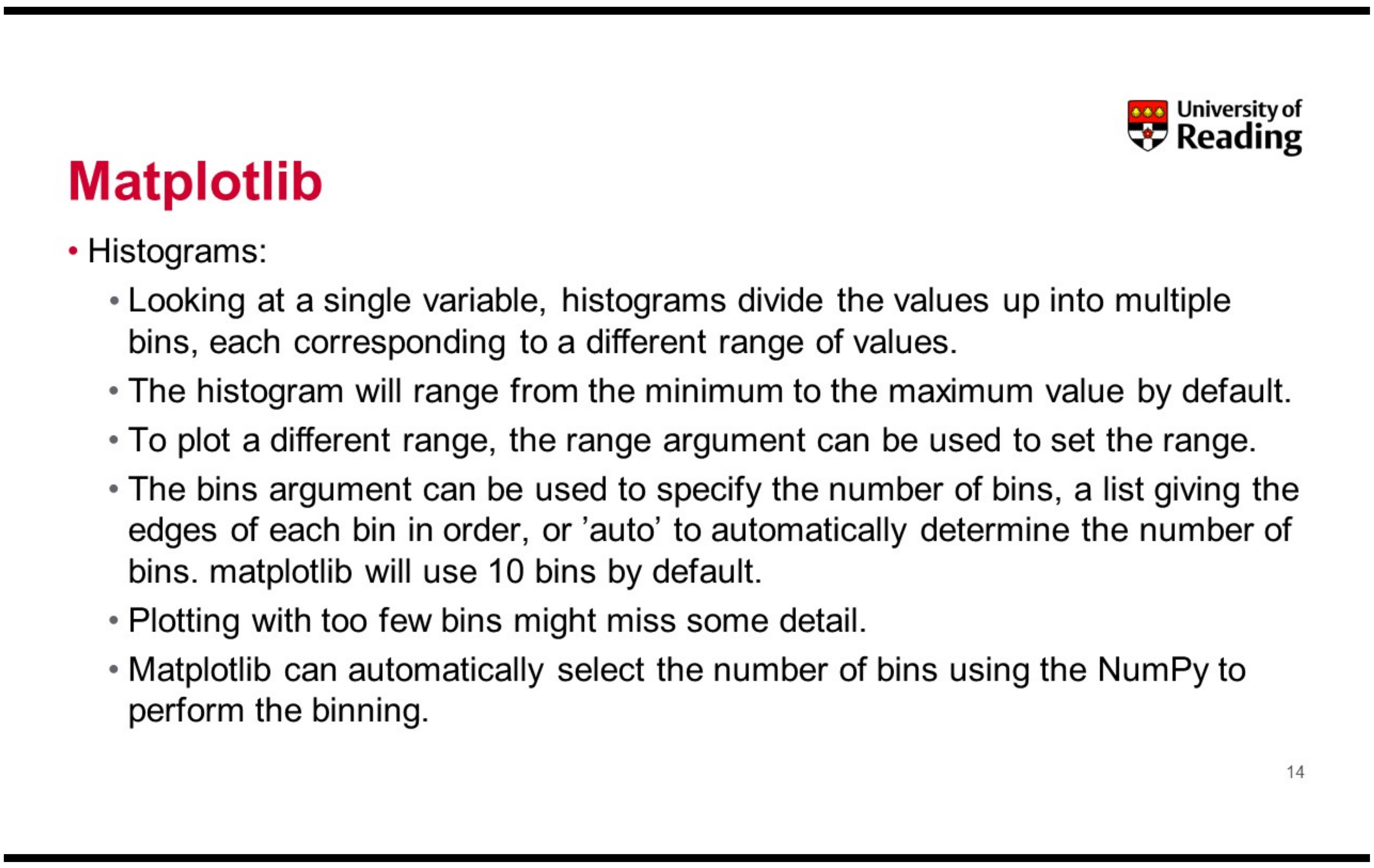
In [14]:

```
##Pie Chart
plt.pie(cc.values)
display
```

Out[14]:

```
<function IPython.core.display.display(*objs, include=None, exclude=None, me
tadata=None, transient=None, display_id=None, **kwargs)>
```





Matplotlib

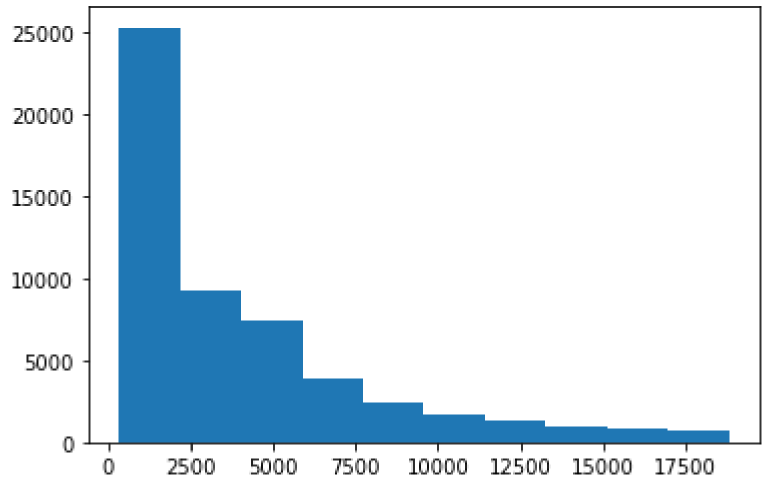
- Histograms:
  - Looking at a single variable, histograms divide the values up into multiple bins, each corresponding to a different range of values.
  - The histogram will range from the minimum to the maximum value by default.
  - To plot a different range, the range argument can be used to set the range.
  - The bins argument can be used to specify the number of bins, a list giving the edges of each bin in order, or 'auto' to automatically determine the number of bins. matplotlib will use 10 bins by default.
  - Plotting with too few bins might miss some detail.
  - Matplotlib can automatically select the number of bins using the NumPy to perform the binning.

In [15]:

```
plt.hist(diamonds['price'])
display
```

Out[15]:

```
<function IPython.core.display.display(*objs, include=None, exclude=None, me
tadata=None, transient=None, display_id=None, **kwargs)>
```

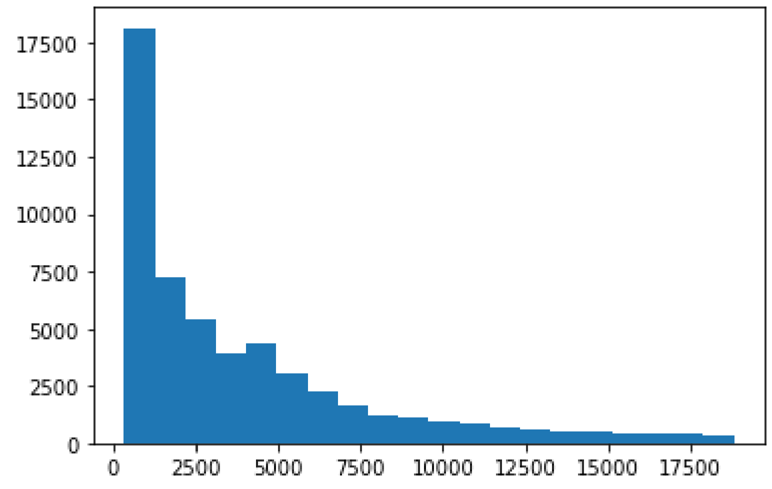


In [16]:

```
plt.hist(diamonds['price'], bins = 20)
display
```

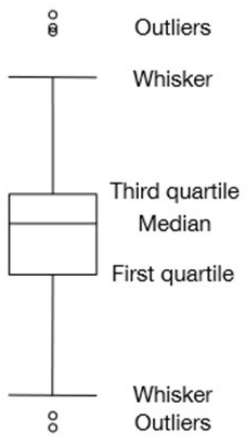
Out[16]:

```
<function IPython.core.display.display(*objs, include=None, exclude=None, me
tadata=None, transient=None, display_id=None, **kwargs)>
```



Matplotlib

- Box and whisker plots:
  - Box and whisker plots allow us to see the distribution of the data in a way that allows us to compare distributions between different sets of values.
    - The whiskers by default show the largest value occurring within 1.5 times the interquartile range (first to third quartiles).
    - Any values outside the whiskers are considered outliers and plotted as points.
- They show **quantiles** of the data, the **median**, and any **outliers**.

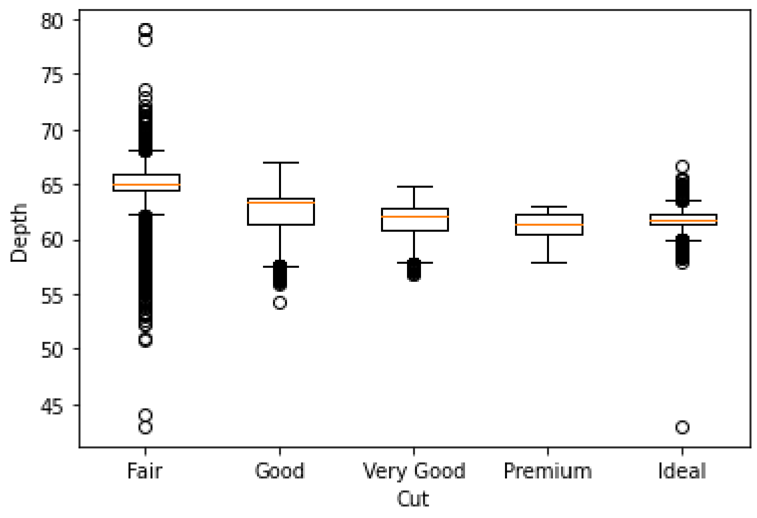


In [17]:

```
##Looking at the distribution of depth values for each diamond cut:

cuts=[]
cutlabel=['Fair','Good','Very Good','Premium','Ideal']
for c in cutlabel:
    cuts.append(diamonds.loc[diamonds.cut==c,'depth'])

plt.xlabel('Cut')
plt.ylabel('Depth')
plt.boxplot(cuts,labels=cutlabel)
display
```

Out[17]:

```
<function IPython.core.display.display(*objs, include=None, exclude=None, me
tadata=None, transient=None, display_id=None, **kwargs)>
```



Matplotlib

- Density Estimates
  - Histograms show a coarse estimate of the density of a variable. The density is the underlying probability density function of a variable.
  - Various approaches can be used to estimate a smoothed density for a continuous variable.

Kernel density estimates

Kernel density estimates place a kernel function centred at each data point to generate a smooth density estimate:

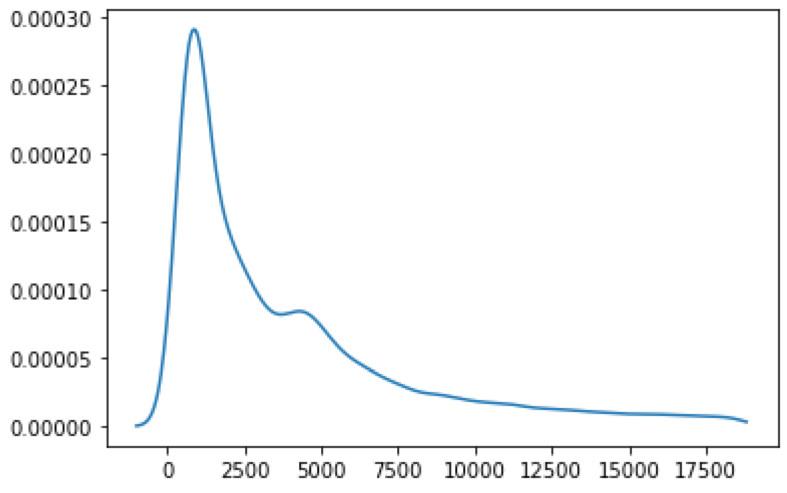
$$\hat{g}(x) = \sum_{i=1}^{N} \frac{1}{N} K(x - x_i)$$

Often K is a Gaussian density, with a variance (or bandwidth) selected automatically from the data.

```
In [18]:
```

```python
import scipy.stats
import numpy as np
dens = scipy.stats.gaussian_kde(diamonds['price'])
xs = np.linspace(-1000,max(diamonds['price']),1000)
plt.plot(xs,dens(xs))
```

```
Out[18]:

[<matplotlib.lines.Line2D at 0x1fe196ffa90>]
```



Note that the density estimate is non-zero below a price of zero. The density estimate is non-zero outside the range of the data.

Matplotlib

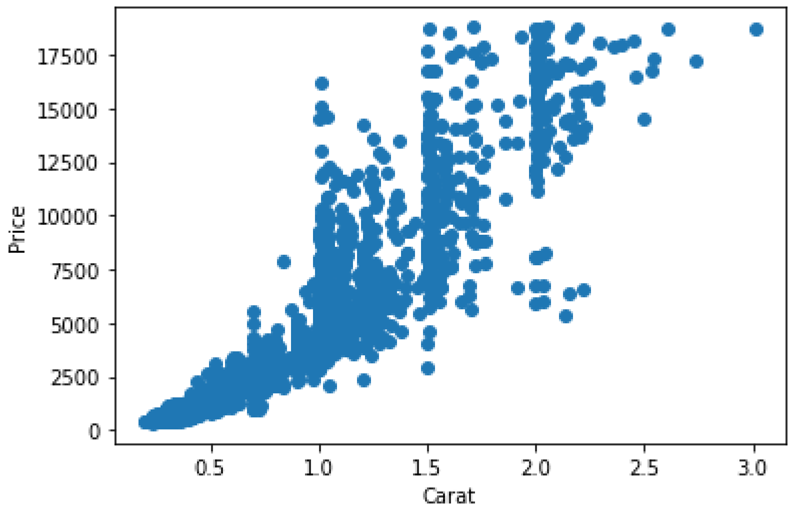
- Scatter Plots:
  - To visualise how two variables are related we can generate a scatter plot. This allows us to quickly spot relationships between the two variables.
  - In matplotlib the colour and size of each point can also be specified. This allows three or four variables to be plotted at once.

In [19]:

```
diamonds_tmp = diamonds.sample(frac=0.05)
plt.scatter(diamonds_tmp['carat'],diamonds_tmp['price'])
plt.xlabel("Carat")
plt.ylabel("Price")
```

Out[19]:

```
Text(0, 0.5, 'Price')
```



In [20]:

```
diamonds.head()
```

Out[20]:

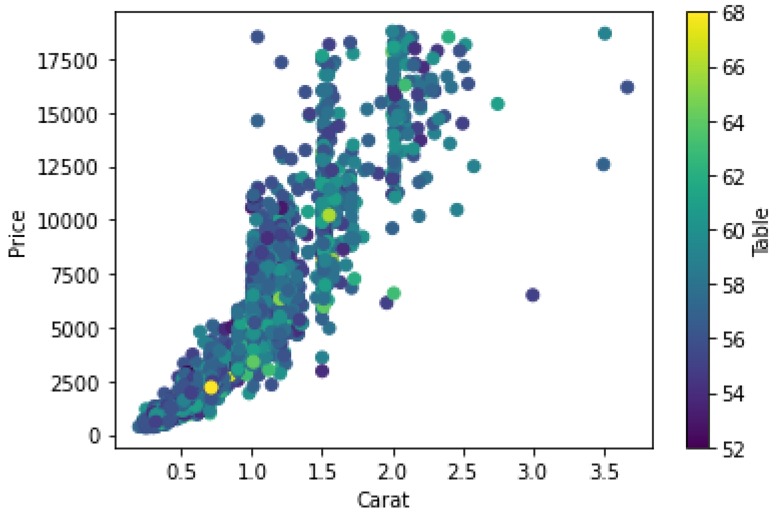
	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

In [21]:

```
diamonds_tmp = diamonds.sample(frac=0.05)
plt.scatter(diamonds_tmp['carat'],diamonds_tmp['price'], c=diamonds_tmp['table'])
cbar = plt.colorbar()
plt.xlabel("Carat")
plt.ylabel("Price")
cbar.ax.set_ylabel('Table', rotation=90)
```

Out[21]:

```
Text(0, 0.5, 'Table')
```



Seaborn

- Seaborn is a library built on matplotlib that takes away some of the extra work required to produce nicely labelled plots in Python. Seaborn is provided with a pandas Data Frame and a specification which columns to use for each aspect of the plot.

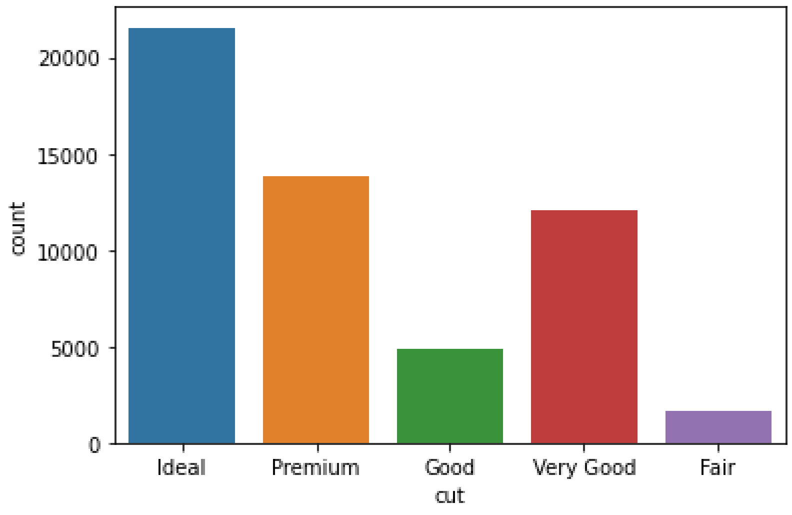
Bar plot:

In [22]:

```
sns.countplot(x="cut", data=diamonds)
```

Out[22]:

```
<AxesSubplot:xlabel='cut', ylabel='count'>
```

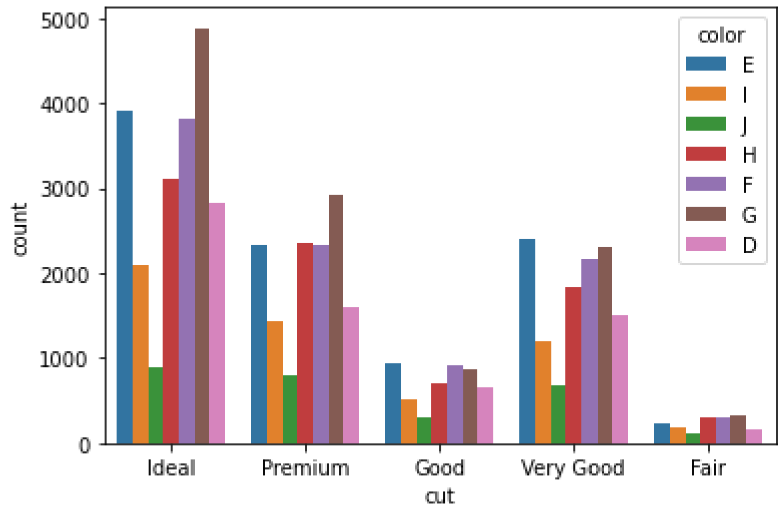


```
In [23]:
```

```
##Grouping more variables
sns.countplot(x="cut", hue = 'color', data=diamonds)
```

```
Out[23]:
```

```
<AxesSubplot:xlabel='cut', ylabel='count'>
```

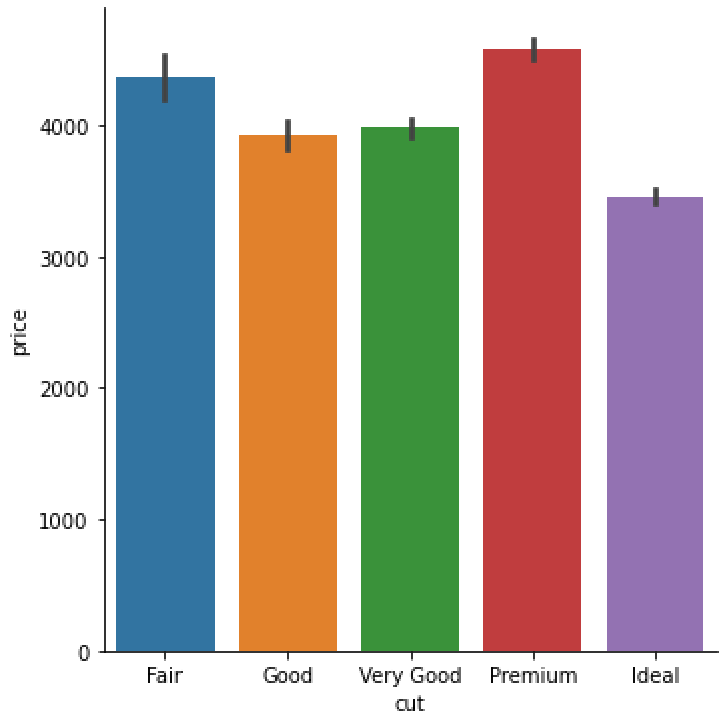


In [24]:

```
sns.catplot(x='cut', y='price', data=diamonds, order=cutlabel, kind="bar")
```

Out[24]:

```
<seaborn.axisgrid.FacetGrid at 0x1fe19720490>
```

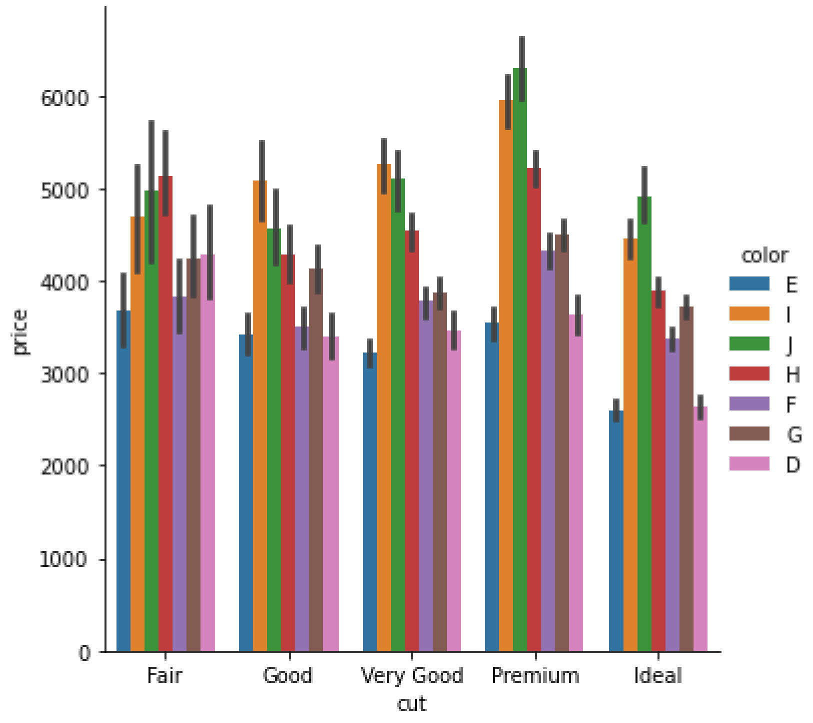


In [25]:

```
sns.catplot(x='cut', y='price', hue='color', data=diamonds, order=cutlabel, kind="bar")
```

Out[25]:

```
<seaborn.axisgrid.FacetGrid at 0x1fe1979c550>
```



Histograms

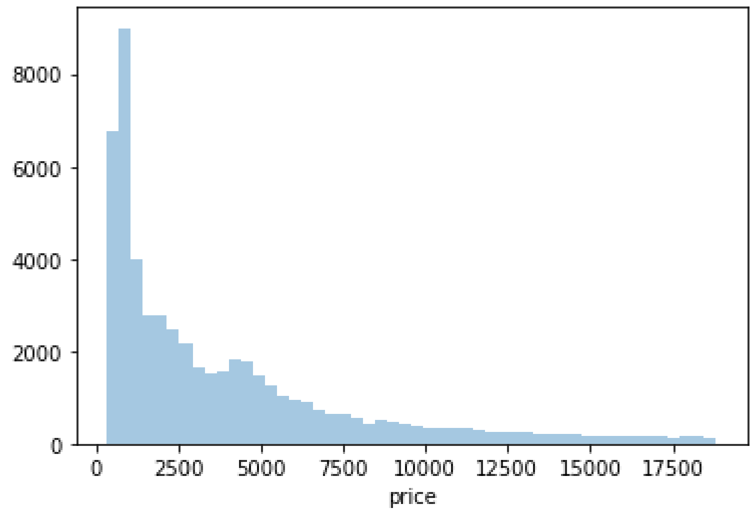
In [26]:

```
sns.distplot(diamonds['price'], kde=False)
```

```
C:\Users\Miguel\anaconda3\lib\site-packages\seaborn\distributions.py:2557: F
utureWarning: `distplot` is a deprecated function and will be removed in a f
uture version. Please adapt your code to use either `displot` (a figure-leve
l function with similar flexibility) or `histplot` (an axes-level function f
or histograms).
  warnings.warn(msg, FutureWarning)
```

Out[26]:

```
<AxesSubplot:xlabel='price'>
```

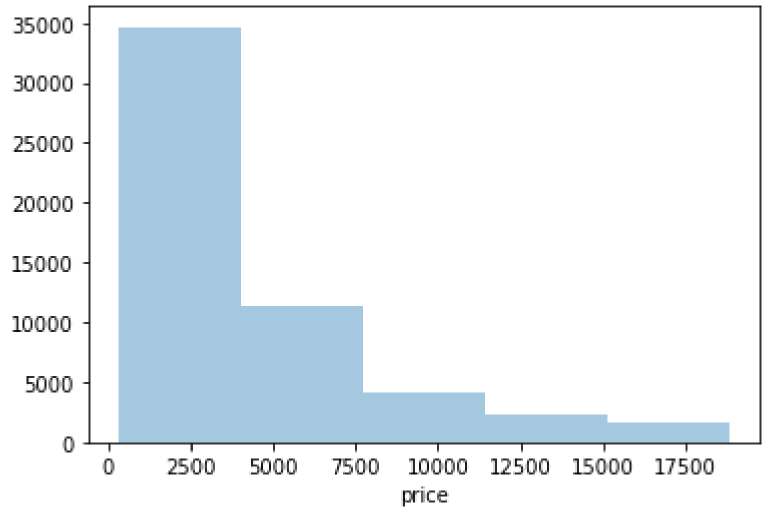


In [27]:

```
sns.distplot(diamonds['price'], kde=False, bins=5)
```

Out[27]:

```
<AxesSubplot:xlabel='price'>
```



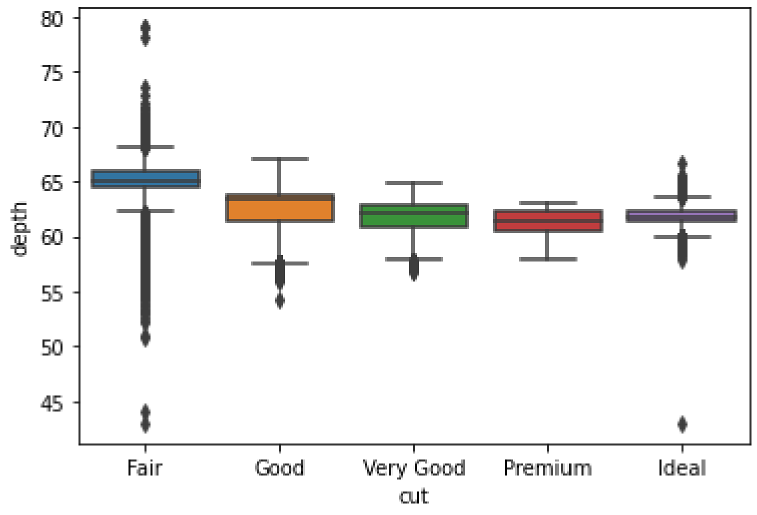
Boxplots

In [28]:

```
sns.boxplot(x='cut',y='depth', order=cutlabel, data=diamonds)
```

Out[28]:

```
<AxesSubplot:xlabel='cut', ylabel='depth'>
```



Density Plots

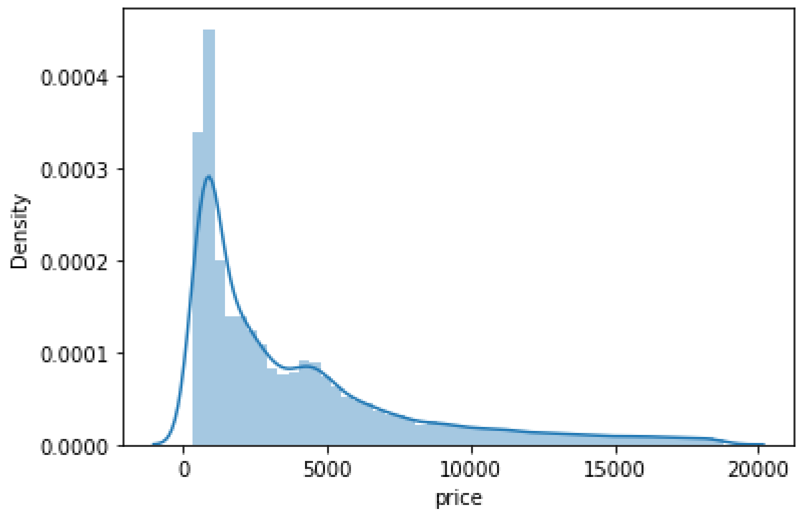
In [29]:

```
sns.distplot(diamonds['price'])
```

```
C:\Users\Miguel\anaconda3\lib\site-packages\seaborn\distributions.py:2557: F
utureWarning: `distplot` is a deprecated function and will be removed in a f
uture version. Please adapt your code to use either `displot` (a figure-leve
l function with similar flexibility) or `histplot` (an axes-level function f
or histograms).
  warnings.warn(msg, FutureWarning)
```

Out[29]:

```
<AxesSubplot:xlabel='price', ylabel='Density'>
```



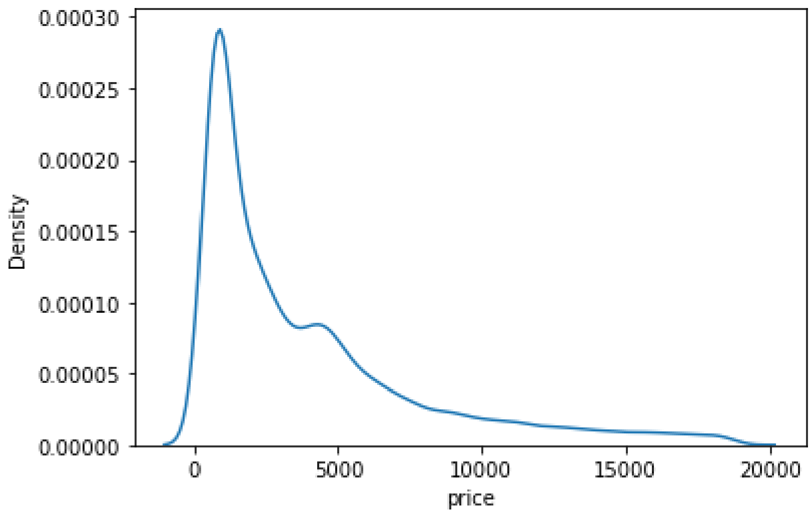
In [30]:

```
sns.distplot(diamonds['price'],hist=False)
```

```
C:\Users\Miguel\anaconda3\lib\site-packages\seaborn\distributions.py:2557: F
utureWarning: `distplot` is a deprecated function and will be removed in a f
uture version. Please adapt your code to use either `displot` (a figure-leve
l function with similar flexibility) or `kdeplot` (an axes-level function fo
r kernel density plots).
  warnings.warn(msg, FutureWarning)
```

Out[30]:

```
<AxesSubplot:xlabel='price', ylabel='Density'>
```



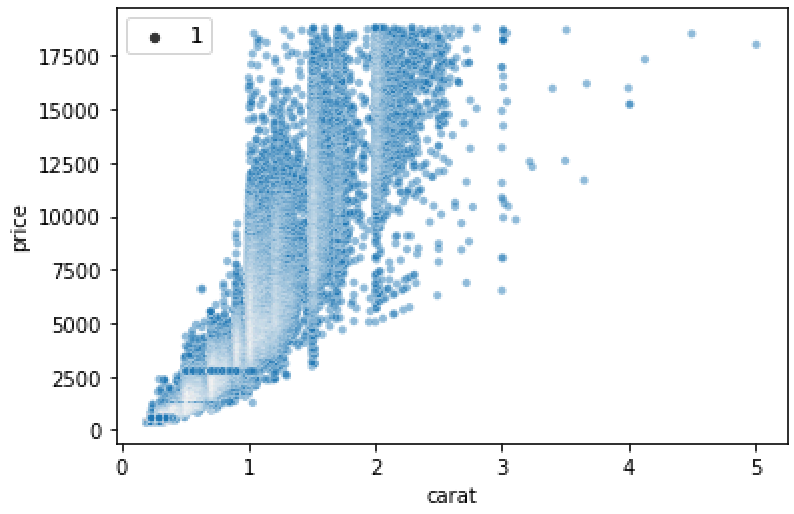
Scatter Plot

In [31]:

```
##Simple 2 variables scatter plot
sns.scatterplot( x='carat', y='price', data=diamonds,alpha=0.5, size=1)
```

Out[31]:

```
<AxesSubplot:xlabel='carat', ylabel='price'>
```

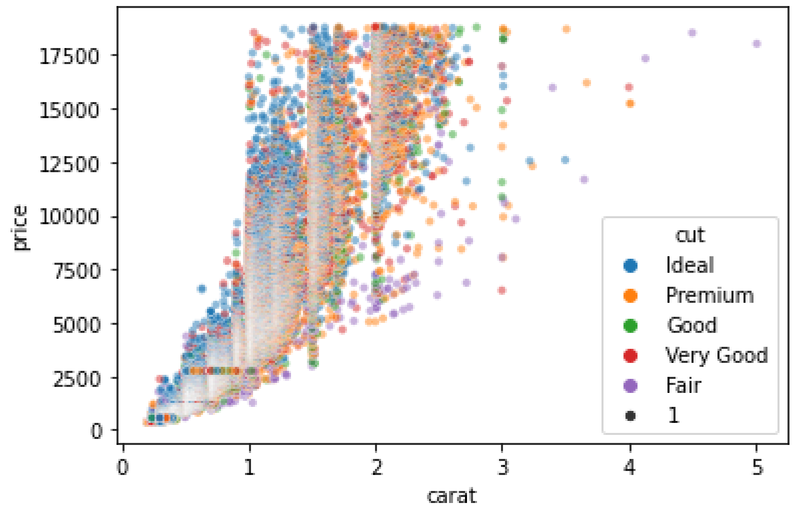


In [32]:

```
##Simple 3 variables scatter plot
sns.scatterplot( x='carat', y='price', hue='cut', data=diamonds,alpha=0.5, size=1)
```

Out[32]:

```
<AxesSubplot:xlabel='carat', ylabel='price'>
```



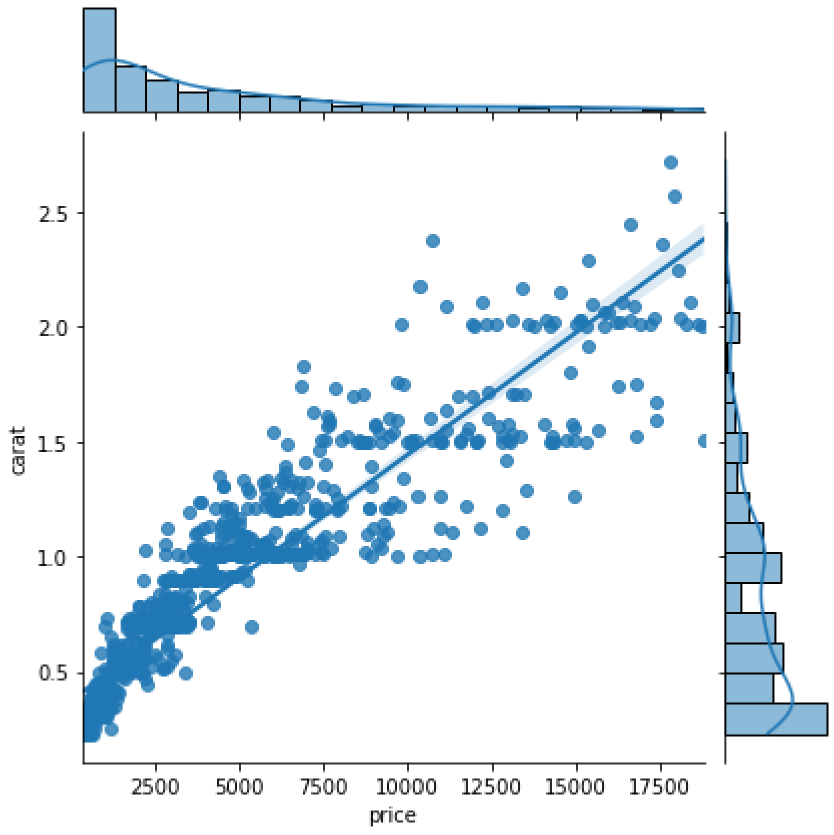
Joint Plots

In [33]:

```
##Diamonds
sns.jointplot(x='price',y='carat',data=diamonds.sample(1000), kind='reg')
```

Out[33]:

```
<seaborn.axisgrid.JointGrid at 0x1fe1afeab80>
```

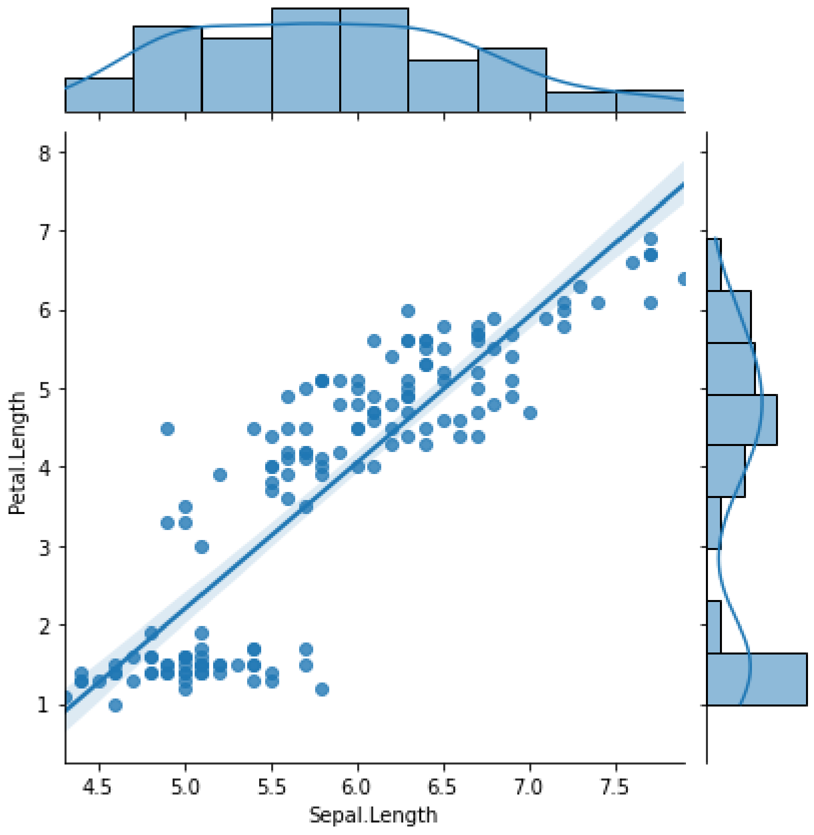


In [34]:

```
##Iris
sns.jointplot(x='Sepal.Length',y='Petal.Length',data=iris, kind='reg')
```

Out[34]:

```
<seaborn.axisgrid.JointGrid at 0x1fe1b2c4f10>
```



Pair plots

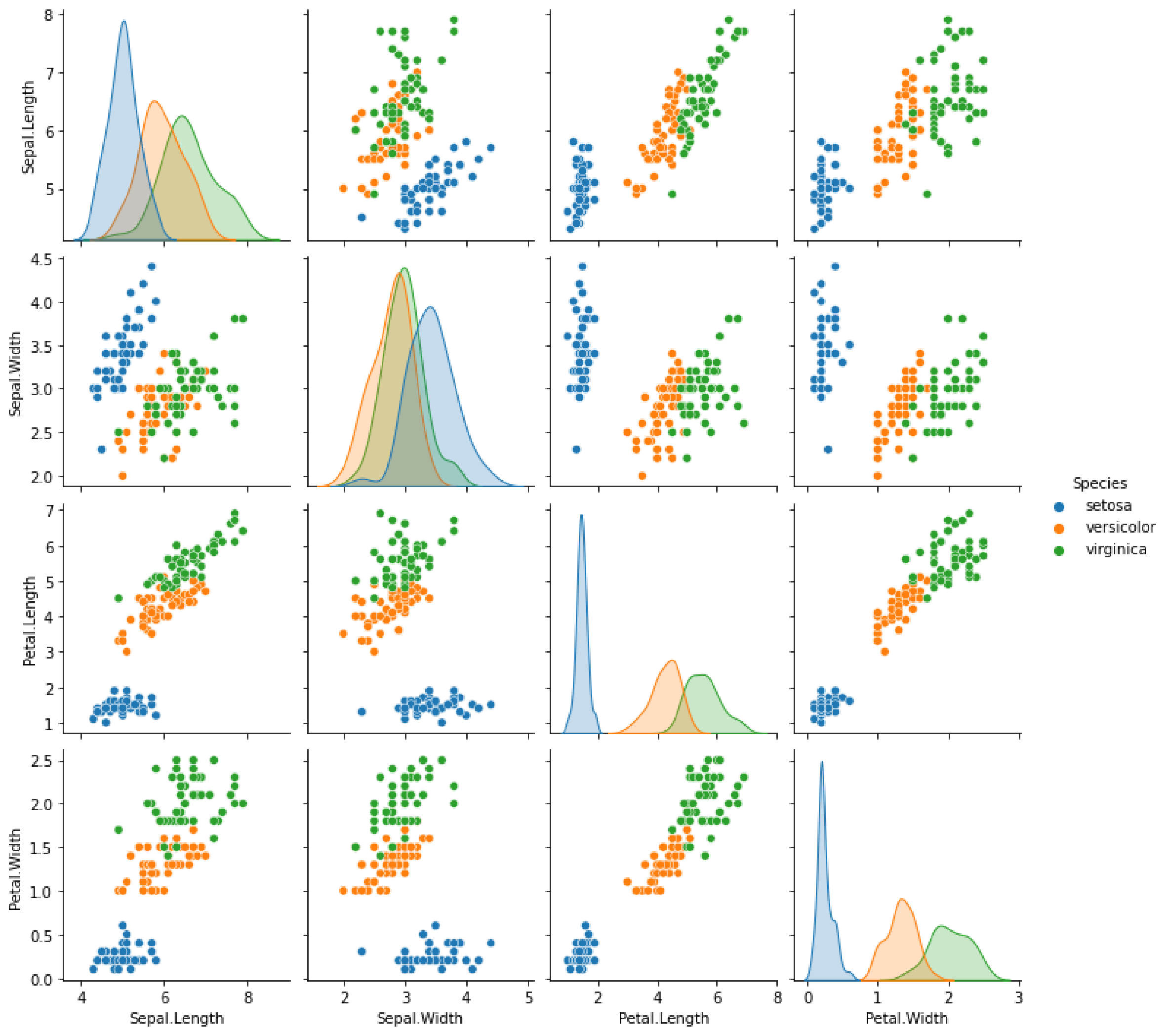
To visualise relationships between multiple variables we can use pair plots:

In [35]:

```
##Iris Dataset
sns.pairplot(iris, hue="Species")
```

Out[35]:

```
<seaborn.axisgrid.PairGrid at 0x1fe1972d8e0>
```



Summary

- Making informative visualisations is one of the most important tasks in data analysis.
- Pandas provides basic plotting capabilities.
- matplotlib is important to know because it forms the basis of most plotting in Python.
- Different plot types for categorical and numerical, multivariate data.
- Seaborn provides higher level plotting features.

Questions