Visualising Distributions of Data ¶

In this section, we will see how to:

- Visualise univariate distributions
- Visualise bivariate distributions

We will also start using the `seaborn` library for data visualisation. Seaborn is a python library built on top of `matplotlib`. It creates much more attractive plots than `matplotlib`, and is often more concise than `matplotlib` when you want to customize your plots, add colors, grids etc.

Let's start with univariate distributions.

Visualising Univariate Distributions

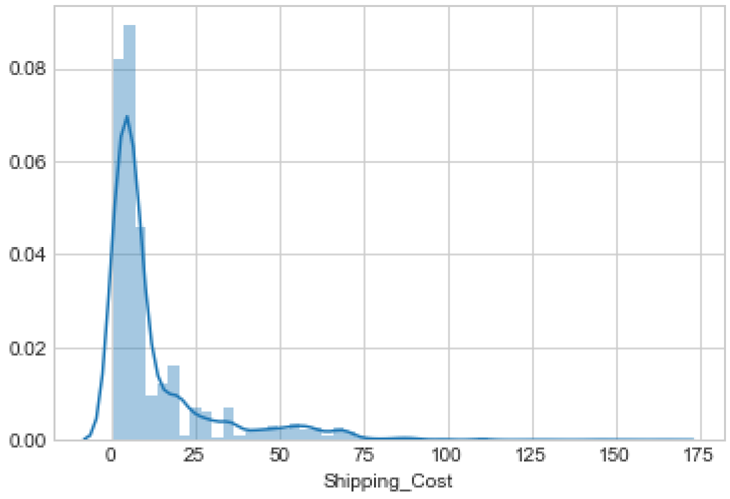
We have already visualised univariate distributions before using boxplots, histograms etc. Let's now do that using seaborn. We'll use the sales data for the upcoming few exercises.

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

# the commonly used alias for seaborn is sns
import seaborn as sns

# set a seaborn style of your taste
sns.set_style("whitegrid")

# data
df = pd.read_csv("./global_sales_data/market_fact.csv")
```

Histograms and Density Plots

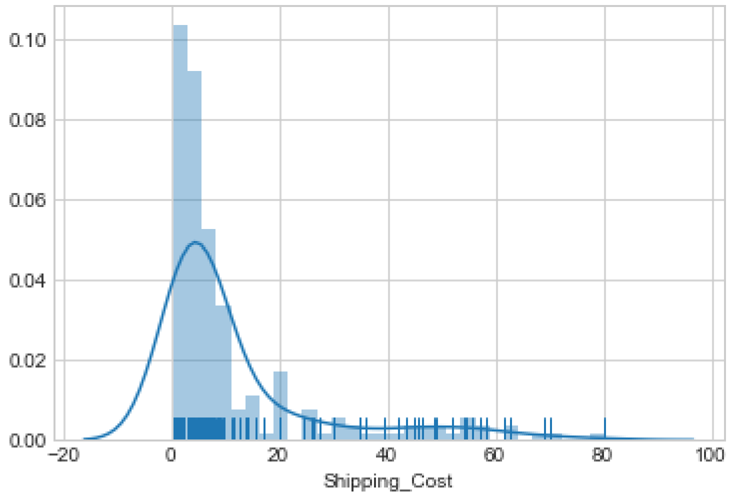
Histograms and density plots show the frequency of a numeric variable along the y-axis, and the value along the x-axis. The `sns.distplot()` function plots a density curve. Notice that this is aesthetically better than vanilla `matplotlib`.

In [2]:
```
# simple density plot
sns.distplot(df['Shipping_Cost'])
plt.show()
```



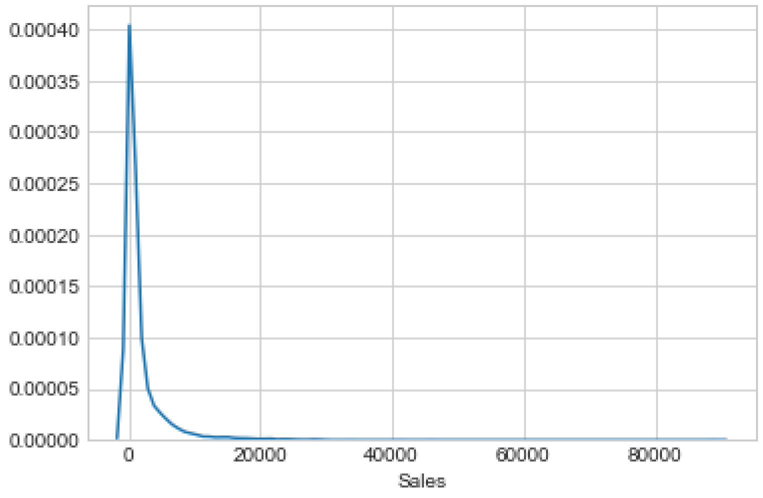
You can also plot what is known as the **rug plot** which plots the actual data points as small vertical bars. The rug plot is simply specified as an argument of the `distplot()`.

In [3]:
```
# rug = True
# plotting only a few points since rug takes a long while
sns.distplot(df['Shipping_Cost'][:200], rug=True)
plt.show()
```



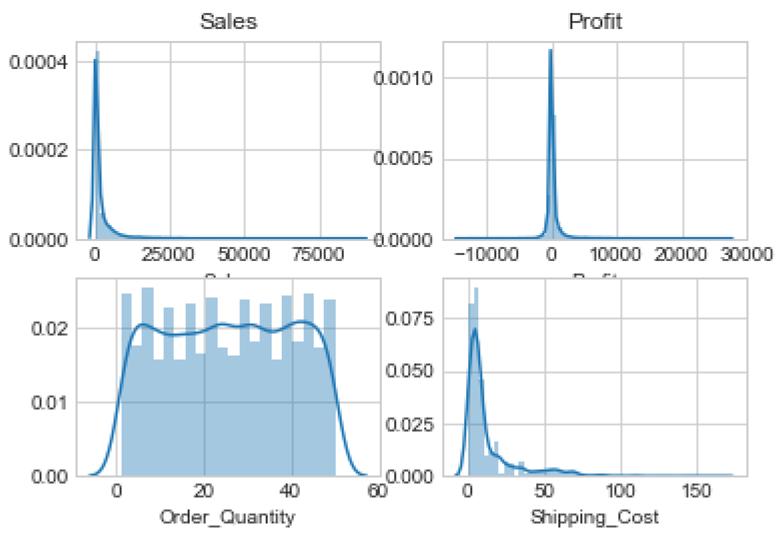
Simple density plot (without the histogram bars) can be created by specifying `hist=False`.

```
In [4]: sns.distplot(df['Sales'], hist=False)
        plt.show()
```



Since seaborn uses matplotlib behind the scenes, the usual matplotlib functions work well with seaborn. For example, you can use subplots to plot multiple univariate distributions.

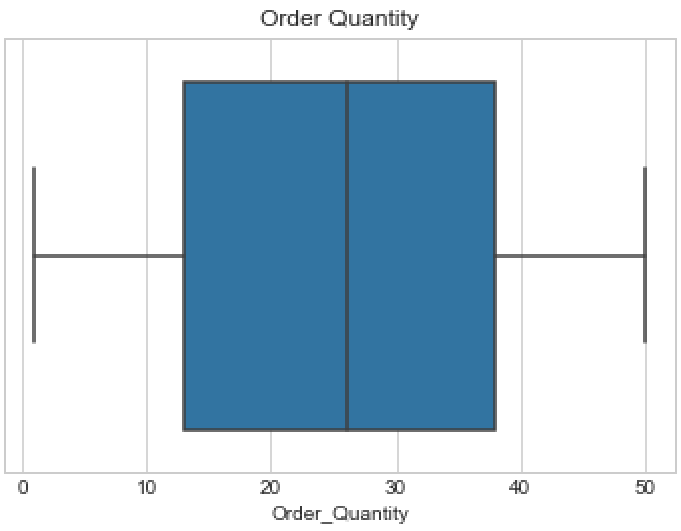
In [5]:
```
# subplots

# subplot 1
plt.subplot(2, 2, 1)
plt.title('Sales')
sns.distplot(df['Sales'])

# subplot 2
plt.subplot(2, 2, 2)
plt.title('Profit')
sns.distplot(df['Profit'])

# subplot 3
plt.subplot(2, 2, 3)
# plt.title('Order Quantity')
sns.distplot(df['Order_Quantity'])

# subplot 4
plt.subplot(2, 2, 4)
# plt.title('Shipping Cost')
sns.distplot(df['Shipping_Cost'])

plt.show()
```



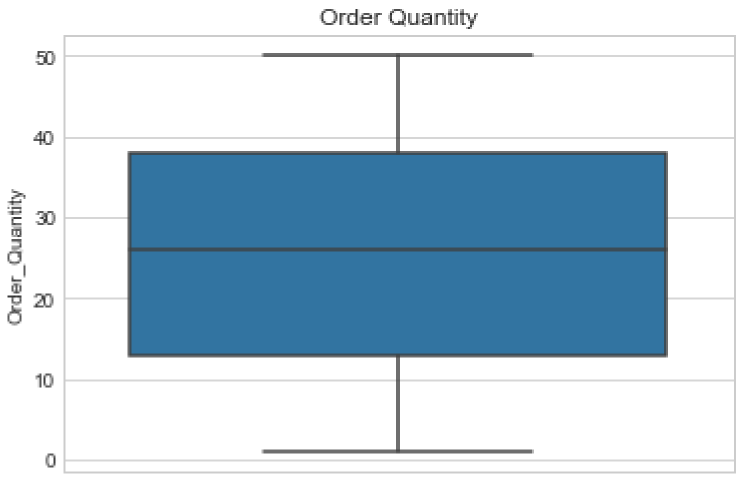
Boxplots

Boxplots are a great way to visualise univariate data because they represent statistics such as the 25th percentile, 50th percentile, etc.

In [6]:
```
# boxplot
sns.boxplot(df['Order_Quantity'])
plt.title('Order Quantity')

plt.show()
```



In [7]:
```
# to plot the values on the vertical axis, specify y=variable
sns.boxplot(y=df['Order_Quantity'])
plt.title('Order Quantity')

plt.show()
```

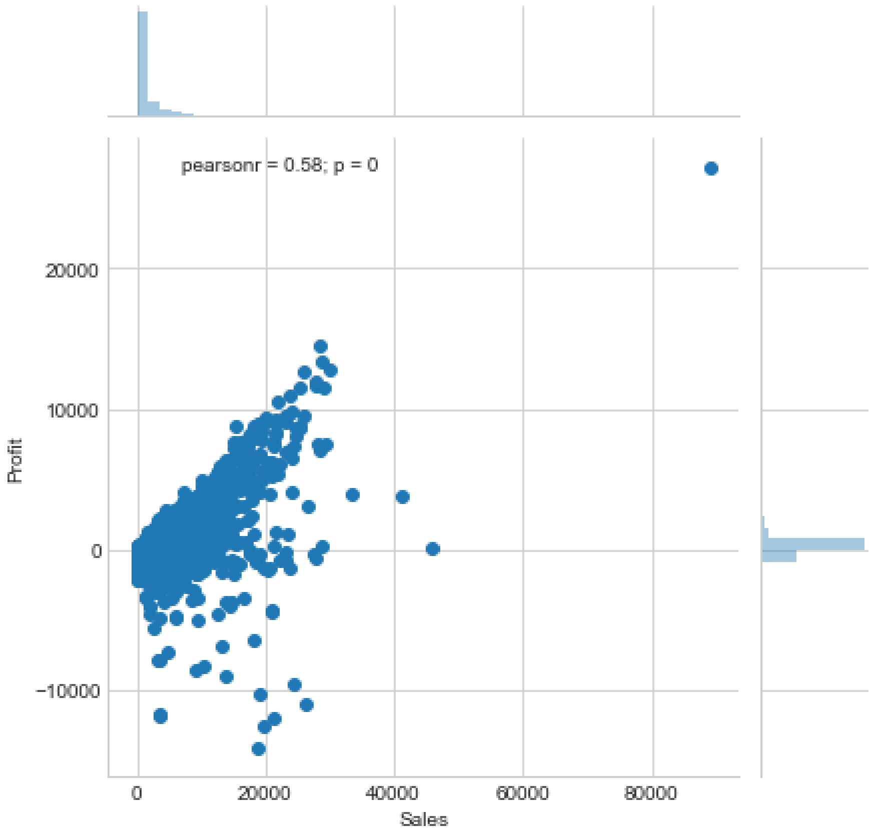


Visualising Bivariate Distributions

Bivariate distributions are simply two univariate distributions plotted on x and y axes respectively. They help you observe the relationship between the two variables.

They are also called joint distributions and are created using `sns.jointplot()`.

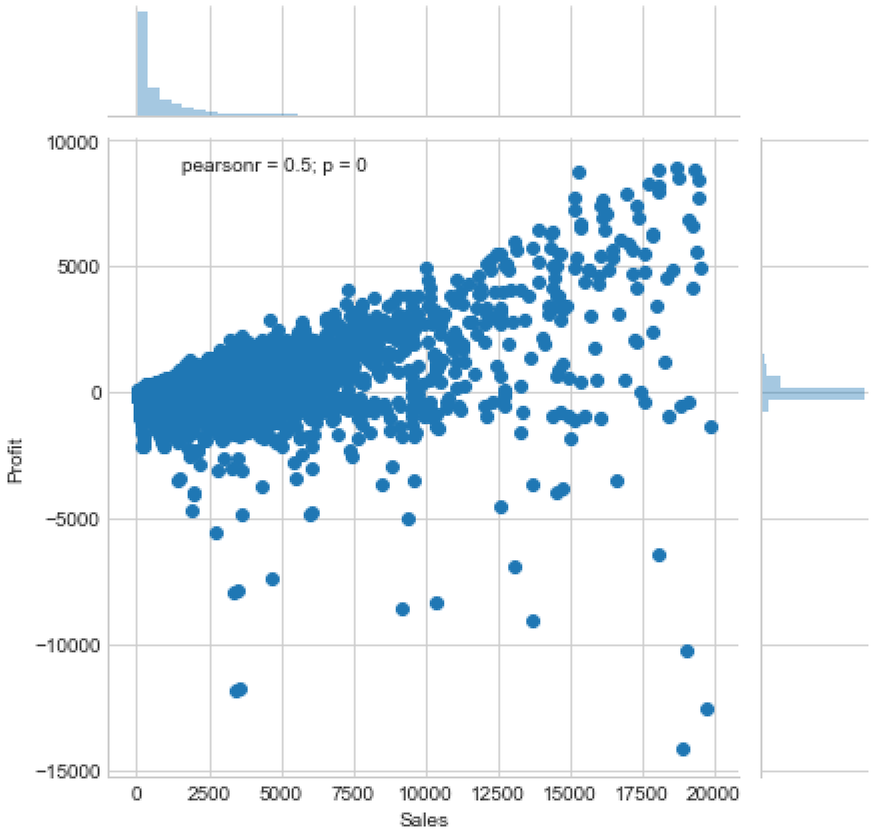
In [8]:
```
# joint plots of Profit and Sales

sns.jointplot('Sales', 'Profit', df)
plt.show()

# same as sns.jointplot(df['Sales'], df['Profit'])
```



Notice that both the distributions are heavily skewed and all the points seem to be concentrated in one region. That is because of some extreme values of Profits and Sales which matplotlib is trying to accomodate in the limited space of the plot.

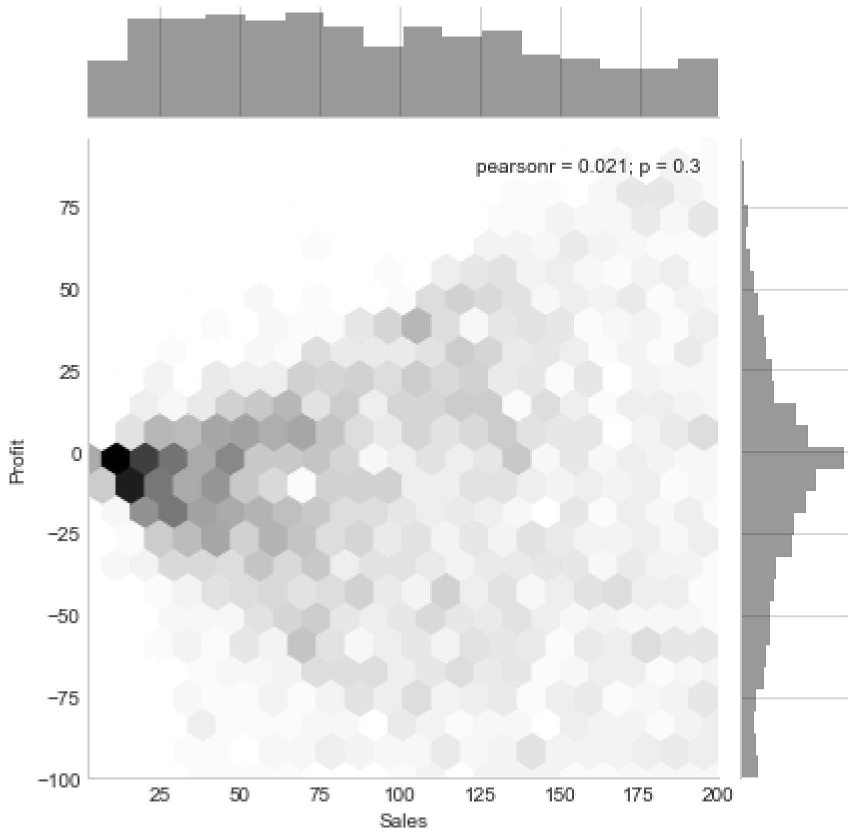
Let's remove that point and plot again.

In [9]:
```
# remove points having extreme values
df = df[(df.Profit < 10000) & (df.Sales < 20000)]

sns.jointplot('Sales', 'Profit', df)
plt.show()
```



You can adjust the arguments of the jointplot() to make the plot more readable. For e.g. specifying `kind=hex` will create a 'hexbin plot'.

In [10]:
```
# plotting low Sales value orders
# hex plot
df = pd.read_csv("./global_sales_data/market_fact.csv")
df = df[(df.Profit < 100) & (df.Profit > -100) & (df.Sales < 200)]
sns.jointplot('Sales', 'Profit', df, kind="hex", color="k")
plt.show()
```



The bottom-right region of the plot represents orders where the Sales is high but the Profit is low, i.e. even when the store is getting highb revenue, the orders are still making losses. These are the kind of orders a business would want to avoid.

We'll see how to drill further down in the next section. For now, let's move to plotting pairwise relationships.

Plotting Pairwise Relationships

You'll find it helpful to plot pairwise relationships between multiple numeric variables. For e.g., here we have taken the prices of some popular cryptocurrencies such as bitcoin, litecoin, ethereum, monero, neo, quantum and ripple.

Now, the crypto enthusiasts would know that the prices of these currencies vary with each other. If bitcoin goes up, the others will likely follow suit, etc.

Now, say you want to trade in some currencies. Given a set of cryptocurrencies, how will you decide when and which one to buy/sell? It will be helpful to analyse past data and identify some trends in these currencies.

In [11]:
```
# reading cryptocurrency files
btc = pd.read_csv("crypto_data/bitcoin_price.csv")
ether = pd.read_csv("crypto_data/ethereum_price.csv")
ltc = pd.read_csv("crypto_data/litecoin_price.csv")
monero = pd.read_csv("crypto_data/monero_price.csv")
neo = pd.read_csv("crypto_data/neo_price.csv")
quantum = pd.read_csv("crypto_data/qtum_price.csv")
ripple = pd.read_csv("crypto_data/ripple_price.csv")

# putting a suffix with column names so that joins are easy
btc.columns = btc.columns.map(lambda x: str(x) + '_btc')
ether.columns = ether.columns.map(lambda x: str(x) + '_et')
ltc.columns = ltc.columns.map(lambda x: str(x) + '_ltc')
monero.columns = monero.columns.map(lambda x: str(x) + '_mon')
neo.columns = neo.columns.map(lambda x: str(x) + '_neo')
quantum.columns = quantum.columns.map(lambda x: str(x) + '_qt')
ripple.columns = ripple.columns.map(lambda x: str(x) + '_rip')

btc.head()
```

Out[11]:

	Date_btc	Open_btc	High_btc	Low_btc	Close_btc	Volume_btc	Market Cap_btc
0	Nov 07, 2017	7023.10	7253.32	7023.10	7144.38	2,326,340,000	117,056,000,000
1	Nov 06, 2017	7403.22	7445.77	7007.31	7022.76	3,111,900,000	123,379,000,000
2	Nov 05, 2017	7404.52	7617.48	7333.19	7407.41	2,380,410,000	123,388,000,000
3	Nov 04, 2017	7164.48	7492.86	7031.28	7379.95	2,483,800,000	119,376,000,000
4	Nov 03, 2017	7087.53	7461.29	7002.94	7207.76	3,369,860,000	118,084,000,000

In [12]:
```
# merging all the files by date
m1 = pd.merge(btc, ether, how="inner", left_on="Date_btc", right_on="Date_et")
m2 = pd.merge(m1, ltc, how="inner", left_on="Date_btc", right_on="Date_ltc")
m3 = pd.merge(m2, monero, how="inner", left_on="Date_btc", right_on="Date_mon"
)
m4 = pd.merge(m3, neo, how="inner", left_on="Date_btc", right_on="Date_neo")
m5 = pd.merge(m4, quantum, how="inner", left_on="Date_btc", right_on="Date_qt"
)
crypto = pd.merge(m5, ripple, how="inner", left_on="Date_btc", right_on="Date_
rip")

crypto.head()
```

Out[12]:

	Date_btc	Open_btc	High_btc	Low_btc	Close_btc	Volume_btc	Market Cap_btc	Da
0	Nov 07, 2017	7023.10	7253.32	7023.10	7144.38	2,326,340,000	117,056,000,000	No 20
1	Nov 06, 2017	7403.22	7445.77	7007.31	7022.76	3,111,900,000	123,379,000,000	No 20
2	Nov 05, 2017	7404.52	7617.48	7333.19	7407.41	2,380,410,000	123,388,000,000	No 20
3	Nov 04, 2017	7164.48	7492.86	7031.28	7379.95	2,483,800,000	119,376,000,000	No 20
4	Nov 03, 2017	7087.53	7461.29	7002.94	7207.76	3,369,860,000	118,084,000,000	No 20

5 rows × 49 columns

In [13]:
```
# Subsetting only the closing prices column for plotting
curr = crypto[["Close_btc", "Close_et", 'Close_ltc', "Close_mon", "Close_neo",
"Close_qt"]]
curr.head()
```

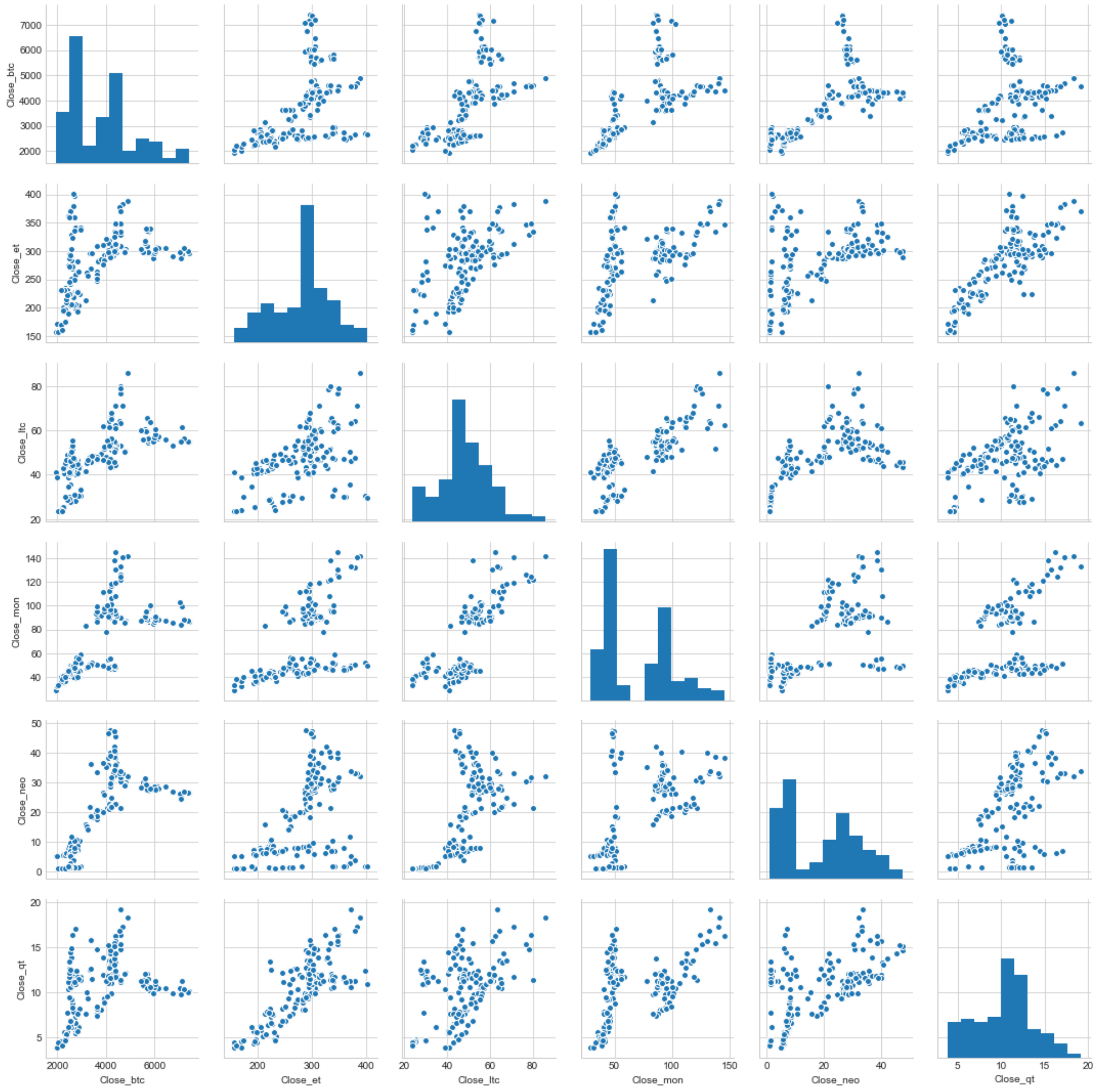
Out[13]:

	Close_btc	Close_et	Close_ltc	Close_mon	Close_neo	Close_qt
0	7144.38	294.66	61.30	99.76	26.23	11.21
1	7022.76	298.89	55.17	102.92	26.32	10.44
2	7407.41	296.26	54.75	86.35	26.38	10.13
3	7379.95	300.47	55.04	87.30	26.49	10.05
4	7207.76	305.71	56.18	87.99	26.82	10.38

Pairwise Scatter Plots

Now, since we have multiple numeric variables, `sns.pairplot()` is a good choice to plot all of them in one figure.

```
In [14]: # pairplot
         sns.pairplot(curr)
         plt.show()
```



In [15]:
```
# You can also observe the correlation between the currencies
# using df.corr()
cor = curr.corr()
round(cor, 3)
```

Out[15]:

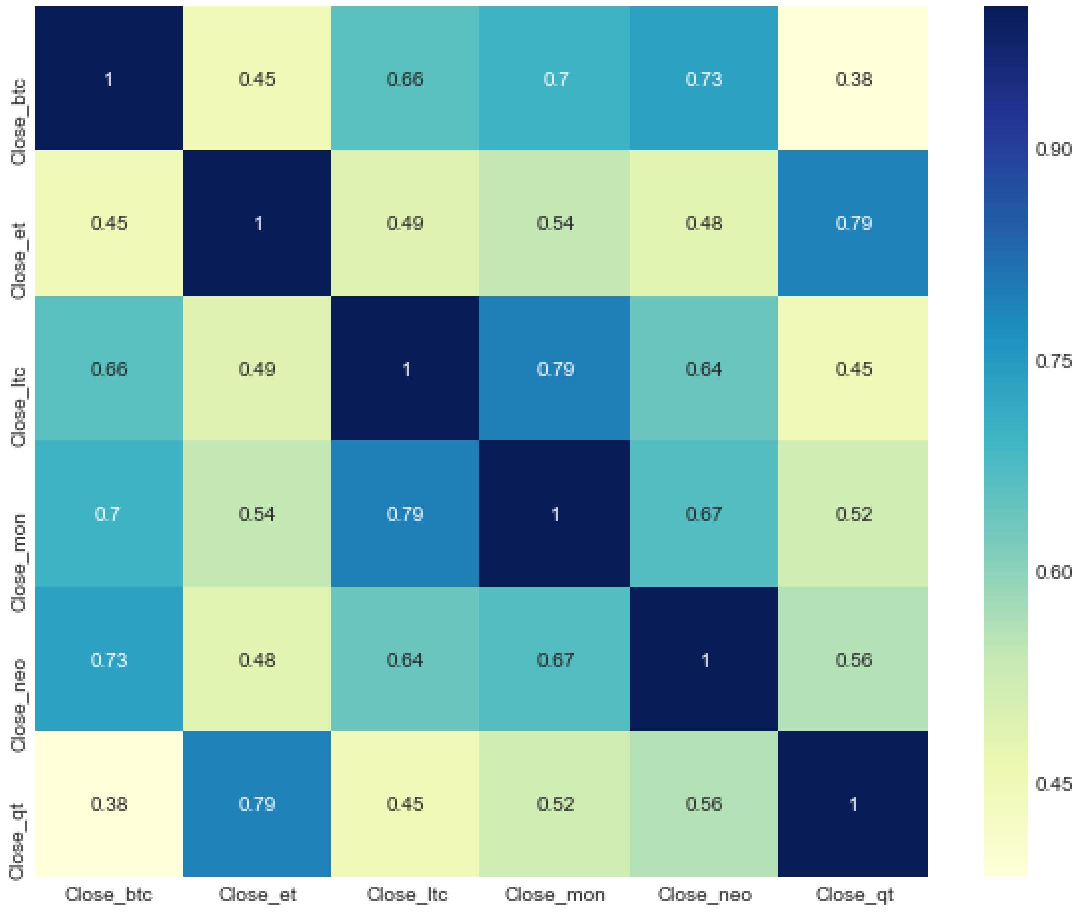
	Close_btc	Close_et	Close_ltc	Close_mon	Close_neo	Close_qt
Close_btc	1.000	0.449	0.658	0.697	0.735	0.382
Close_et	0.449	1.000	0.490	0.539	0.482	0.791
Close_ltc	0.658	0.490	1.000	0.793	0.641	0.448
Close_mon	0.697	0.539	0.793	1.000	0.669	0.518
Close_neo	0.735	0.482	0.641	0.669	1.000	0.557
Close_qt	0.382	0.791	0.448	0.518	0.557	1.000

The dataframe above is a **correlation matrix** of cryptocurrencies. Try finding some important relationships between currencies. Notice that quantum and ethereum are highly correlated (0.79).

Heatmaps

It will be helpful to visualise the correlation matrix itself using `sns.heatmap()`.

In [21]:
```
# figure size
plt.figure(figsize=(10,8))

# heatmap
sns.heatmap(cor, cmap="YlGnBu", annot=True)
plt.show()
```



The orange boxes show the most correlated currencies. Specifically, **ethereum-quantum (0.79)** and **monero-ltc (0.79)** are the most correlated pairs. Also, **neo-btc (0.73)** are quite highly correlated.

Please note that this data is from a specific time period only.

Thus, from a risk-minimisation point of view, you should not invest in these pairs of cryptocurrencies, since if one goes down, the other is likely to go down as well (but yes, if one goes up, you'll become filthy rich).

In the next section, we will explore how to plot categorical variables.