Basic Statistics and Statistical Graphs

February 5, 2024

1 Applied Data Science 1

1.0.1 Module Leader: Dr. William Cooper

1.1 Basic Statistics

We will explore here some basic statistics and operations on pandas dataframes.

[152]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns  # note this new package! used a lot in online examples
import pandas as pd
```

Let's read those financial files as before, but this time combine them into one dataframe.

[153]:
```
df_tesco = pd.read_csv('Data/TSCO_ann.csv', index_col='year')
df_bp = pd.read_csv('Data/BP_ann.csv', index_col='year')
df_barclays = pd.read_csv('Data/BCS_ann.csv', index_col='year')
df_vodaphone = pd.read_csv('Data/VOD_ann.csv', index_col='year')
companies = ('Tesco', 'BP', 'Barclays', 'Vodaphone')

for i, df in enumerate((df_tesco, df_bp, df_barclays, df_vodaphone)):
    df.rename(columns={col: f'{companies[i]} {col}' for col in df.columns},␣
 ↪inplace=True)

df = df_tesco.join([df_bp, df_barclays, df_vodaphone])
df.head()
```

[153]:
```
      Tesco price  Tesco ann_return   BP price  BP ann_return  Barclays price  \
year
1989    24.494144         22.941372  41.180229      19.174906        1.571551
1990    30.810261         17.814613  49.884350     -11.157063        1.973807
1991    36.818260          5.895034  44.617970      -6.082900        2.217046
1992    39.053959          5.149691  41.984802     -15.618524        2.199452
1993    41.117802        -12.174125  35.913830      46.029975        2.234173

      Barclays ann_return  Vodaphone price  Vodaphone ann_return
year
```

```
1989             25.596115         4.003933            33.478386
1990             12.323343         5.596050           -21.962895
1991             -0.793578         4.492602            17.645562
1992              1.578620         5.359591            10.330250
1993             53.486592         5.942858            48.265578
```

Let's look at each of these columns and see some basic information about them.

Mean:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Median:

$$\text{median} = \begin{cases} x_{\frac{n+1}{2}}, & \text{if } n \text{ is odd} \\ \frac{x_{\frac{n}{2}} + x_{\frac{n}{2}+1}}{2}, & \text{if } n \text{ is even} \end{cases}$$

Standard deviation:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

[154]:

```python
# mean and median:
print(df.mean(), end='\n\n')
print(df.median(), end='\n\n')

# standard deviation
print(df.std(), end='\n\n')

# sum by column = not that meaningful
print(df.sum(axis=0), end='\n\n')

# sum by row but only just prices
df_prices = df[[col for col in df.columns if 'price' in col]].copy()
df_prices['Total'] = df_prices.sum(axis=1)
print(df_prices.Total)
```

```
Tesco price             164.413870
Tesco ann_return          7.561509
BP price                201.571405
BP ann_return             6.714966
Barclays price           10.985586
Barclays ann_return      15.326749
Vodaphone price          64.510986
Vodaphone ann_return     10.546701
dtype: float64

Tesco price             160.541336
Tesco ann_return          7.674430
```

```
BP price                214.922073
BP ann_return             7.406347
Barclays price            9.691092
Barclays ann_return       5.469468
Vodaphone price          49.015934
Vodaphone ann_return     11.465884
dtype: float64

Tesco price              98.825000
Tesco ann_return         19.519435
BP price                110.286764
BP ann_return            20.749894
Barclays price            7.700208
Barclays ann_return      46.756708
Vodaphone price          51.260722
Vodaphone ann_return     29.045485
dtype: float64

Tesco price             5425.657696
Tesco ann_return         249.529807
BP price                6651.856367
BP ann_return            221.593885
Barclays price           362.524330
Barclays ann_return      505.782704
Vodaphone price         2128.862528
Vodaphone ann_return     348.041119
dtype: float64

year
1989     71.249857
1990     88.264468
1991     88.145878
1992     88.597804
1993     85.208663
1994    106.370790
1995    118.452896
1996    153.759720
1997    207.804806
1998    279.071566
1999    326.308958
2000    395.713158
2001    442.709822
2002    407.228013
2003    295.033504
2004    386.299917
2005    487.707494
2006    567.866259
2007    609.931957
```

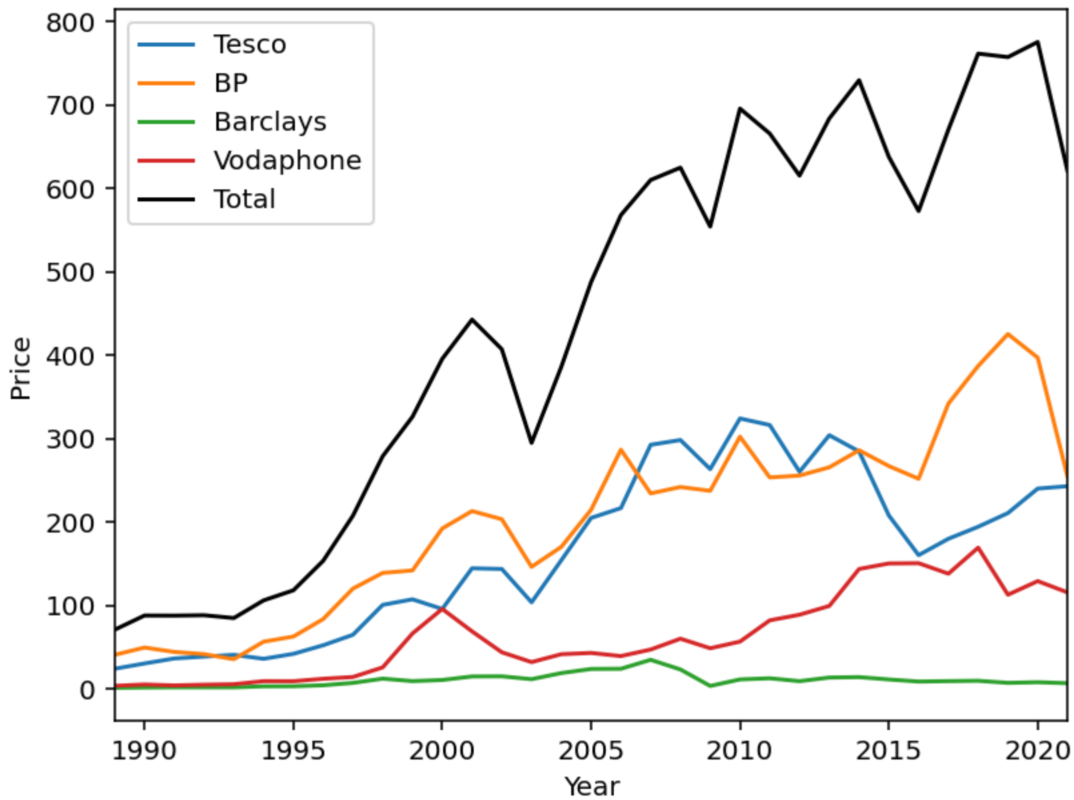
```
2008    624.731457
2009    554.122681
2010    695.423678
2011    665.523784
2012    615.146909
2013    683.612111
2014    729.407784
2015    637.596990
2016    572.671862
2017    670.291280
2018    761.287522
2019    757.100665
2020    775.208532
2021    621.050136
Name: Total, dtype: float64
```

[155]:

```
def plot_prices_series(df_prices):
    """
    Plots the prices of stocks and total prices by year
    """
    plt.figure(dpi=144)

    for i, col in enumerate(df_prices.columns):
        if col == 'Total':
            break
        df_prices[col].plot(label=companies[i])

    plt.plot(df_prices.Total, color='k', label='Total')

    plt.xlim(df_prices.index.min(), df_prices.index.max())
    plt.xlabel('Year')
    plt.ylabel('Price')
    plt.legend()
    plt.show()
    return
```

[156]:

```
plot_prices_series(df_prices)
```



How does each stock relate to each other directly, though?

Covariance:

$$\mathrm{Cov}(X,Y)=\frac{1}{n}\sum_{i=1}^{n}(X_i-\bar{X})(Y_i-\bar{Y})$$

[157]:
```
df_prices.cov()
```

[157]:

	Tesco price	BP price	Barclays price	Vodaphone price	\
Tesco price	9766.380656	8677.650198	443.508580	3022.483101	
BP price	8677.650198	12163.170225	345.370058	4699.801428	
Barclays price	443.508580	345.370058	59.293199	69.118230	
Vodaphone price	3022.483101	4699.801428	69.118230	2627.661654	
Total	21910.022536	25885.991910	917.290068	10419.064413	

	Total
Tesco price	21910.022536
BP price	25885.991910
Barclays price	917.290068
Vodaphone price	10419.064413
Total	59132.368926

Covariance has large positive values if the values of x and y tend to move together. Large negative values if they tend to move in the opposite direction.

Problem: the covariance can be large because stocks move together a lot or because the variances are high.

Solution: normalise by dividing by the standard deviations. That gives the correlation coefficient, ρ.

$$\rho_{X,Y} = \frac{\mathrm{Cov}(X,Y)}{\sigma_X \sigma_Y}$$

or:

$$\rho_{X,Y} = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}}$$

ρ has values between -1 and +1. - +1: perfect correlation. Values are fixed multiples. - -1: perfect anticorrelation - 0: no correlation. Values are independent.

[158]:
```
df_prices.corr()
```

[158]:
```
                 Tesco price  BP price  Barclays price  Vodaphone price  \
Tesco price         1.000000  0.796181        0.582818         0.596640
BP price            0.796181  1.000000        0.406686         0.831326
Barclays price      0.582818  0.406686        1.000000         0.175108
Vodaphone price     0.596640  0.831326        0.175108         1.000000
Total               0.911724  0.965225        0.489882         0.835856

                    Total
Tesco price      0.911724
BP price         0.965225
Barclays price   0.489882
Vodaphone price  0.835856
Total            1.000000
```

Note the values on the diagonal will always be 1. This is the *Pearson's* correlation coefficient. It measures **linear** correlation. It is not good measuring non-linear correlation.

Kendall's ranked correlation coefficient instead measures ordinal association. - values are sorted in increasing (or decreasing order) - if by moving from the ith value to the next and values of both columns are increasing this is counted as *concordant*. - *discordant* if decreasing.

$$\tau = \frac{2}{n(n-1)} \sum_{i<j} \mathrm{sign}(X_i - X_j)\mathrm{sign}(Y_i - Y_j)$$

[159]:
```
df_prices.corr(method='kendall')
```

[159]:
```
                 Tesco price  BP price  Barclays price  Vodaphone price  \
Tesco price         1.000000  0.643939        0.454545         0.503788
```

```
BP price          0.643939  1.000000        0.340909         0.708333
Barclays price    0.454545  0.340909        1.000000         0.261364
Vodaphone price   0.503788  0.708333        0.261364         1.000000
Total             0.715909  0.890152        0.367424         0.727273

                    Total
Tesco price      0.715909
BP price         0.890152
Barclays price   0.367424
Vodaphone price  0.727273
Total            1.000000
```

A good way of quickly looking at some basic stats is with the *describe* function.

[160]:
```python
df_prices.describe()
```

[160]:

	Tesco price	BP price	Barclays price	Vodaphone price	Total
count	33.000000	33.000000	33.000000	33.000000	33.000000
mean	164.413870	201.571405	10.985586	64.510986	441.481846
std	98.825000	110.286764	7.700208	51.260722	243.171480
min	24.494144	35.913830	1.571551	4.003933	71.249857
25%	65.155853	120.503860	4.718605	14.598728	207.804806
50%	160.541336	214.922073	9.691092	49.015934	487.707494
75%	243.157288	265.760223	13.961011	99.705025	637.596990
max	324.262756	425.389618	35.202843	169.534927	775.208532

2 Question 1

Read the *UK_cities.txt* file (hint: sep='\s+') and check some basic statistics. Compare the population mean and median. Compare the Pearson's and Kendall correlations (you may need to set *numeric_only=True*). Sum the population, grouped by their nation/region (df.groupby), and create a pie chart.

[ ]:
```python
df_cities = pd.read_csv(, index_col='City')
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:
```python
def plot_region_pie(dfgrp_region):
    """
    Plots sum of regional cities
```

```
    """
    return
```

[ ]:
```
dfgrp_region = df_cities.
plot_region_pie(dfgrp_region)
```

2.1 End Question 1

Let’s now look at some other ways of visualising these basic statistics.

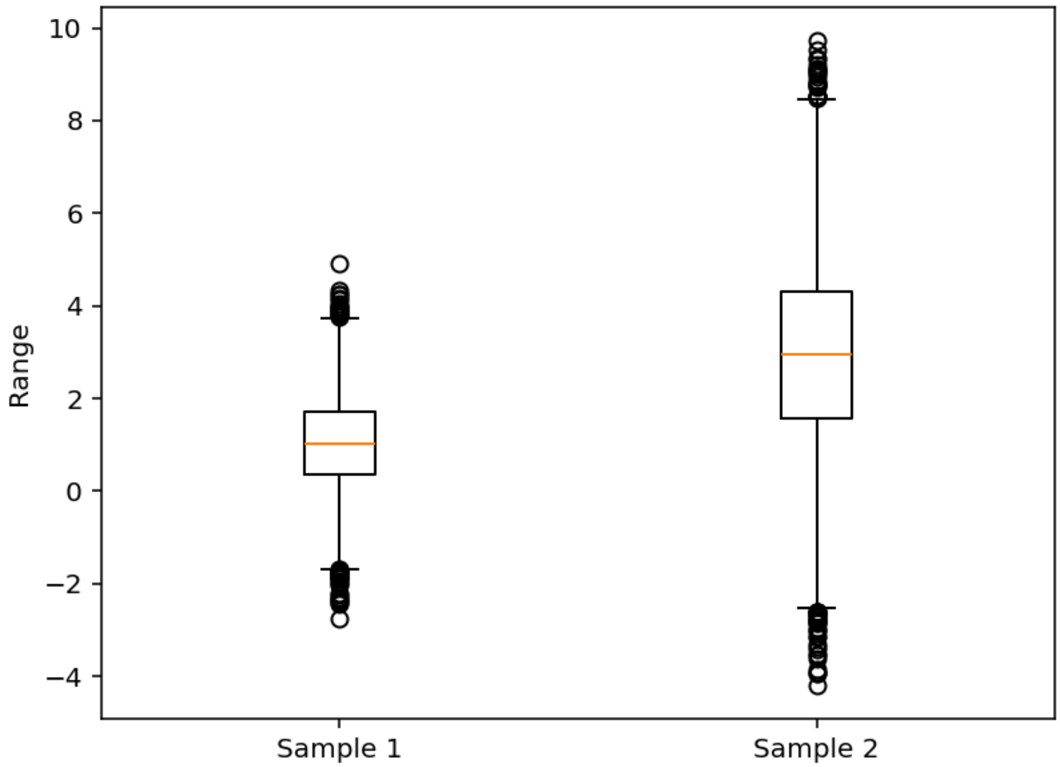
2.2 Statistical Graphs

Starting with some random data, what are the later rows in *describe* showing us?

[168]:
```
sample1 = np.random.normal(1.0, 1.0, 10000)
sample2 = np.random.normal(3.0, 2.0, 10000)
```

[169]:
```
def plot_sample_box(*samples):
    """
    Creates box plot of random samples
    """
    plt.figure(dpi=144)

    plt.boxplot(samples, labels=[f'Sample {i+1}' for i in range(len(samples))])

    plt.ylabel('Range')
    plt.show()
    return
```

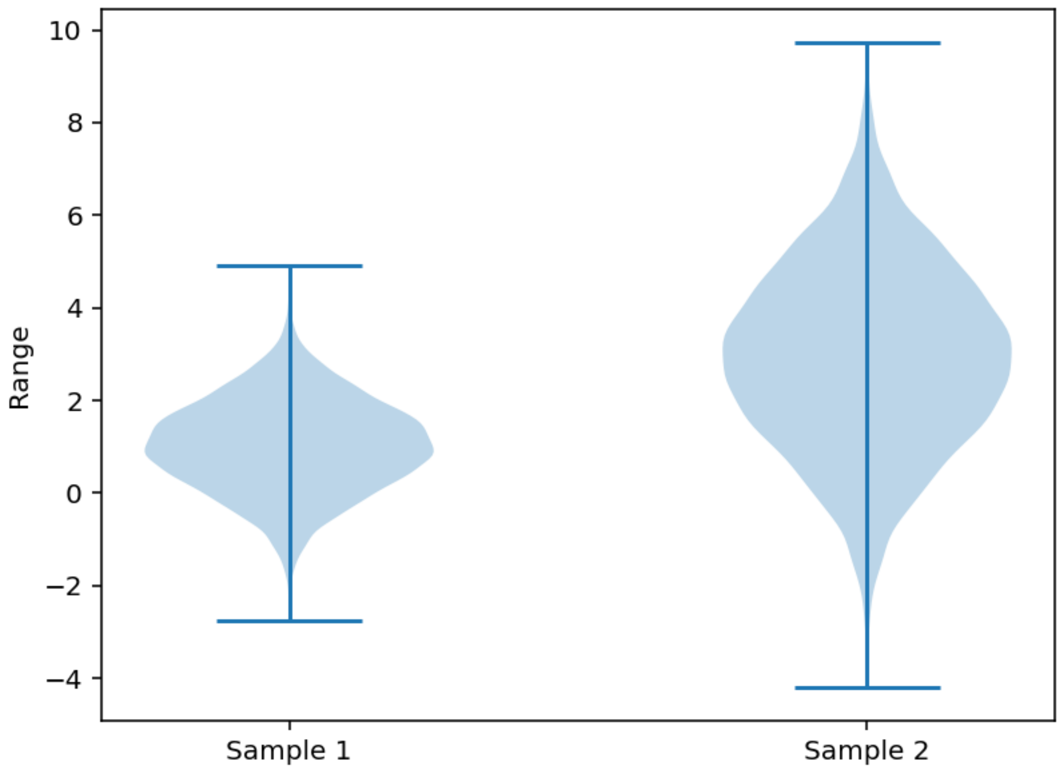
[170]:
```
plot_sample_box(sample1, sample2)
```



The orange line is the median, the box around it is the interquartile range (25% to 75%). By default the whiskers are 1.5X the interquartile range. Black circles beyond this are outliers. In data science, you will want to customise this selection and **justify** why you are selecting/rejecting data.

[171]:
```
def plot_sample_violin(*samples):
    """
    Creates violin plot of random samples
    """
    plt.figure(dpi=144)

    plt.violinplot(samples)

    plt.xticks(np.arange(1, len(samples) + 1), labels=[f'Sample {i+1}' for i in␣
 ↪range(len(samples))])
    plt.ylabel('Range')
    plt.show()
    return
```

[172]:
```
plot_sample_violin(sample1, sample2)
```



Violin plots show all of the data (this can be customised) but also shows a more meaningful representation of where the data points lie.

3 Question 2

Create a boxplot and violin plot for the annual returns of the previous financial data (Tesco, BP, Barclays, Vodaphone).

[ ]:
```
df_returns =
df_returns.head()
```

[ ]:
```
def plot_returns_box(df_returns):
    """
    Plot the annual returns as a boxplot
    """
    return
```

[ ]:
```
def plot_returns_violin(df_returns):
    """
    Plot the annual returns as a violin plot
    """
```

```
    return
```

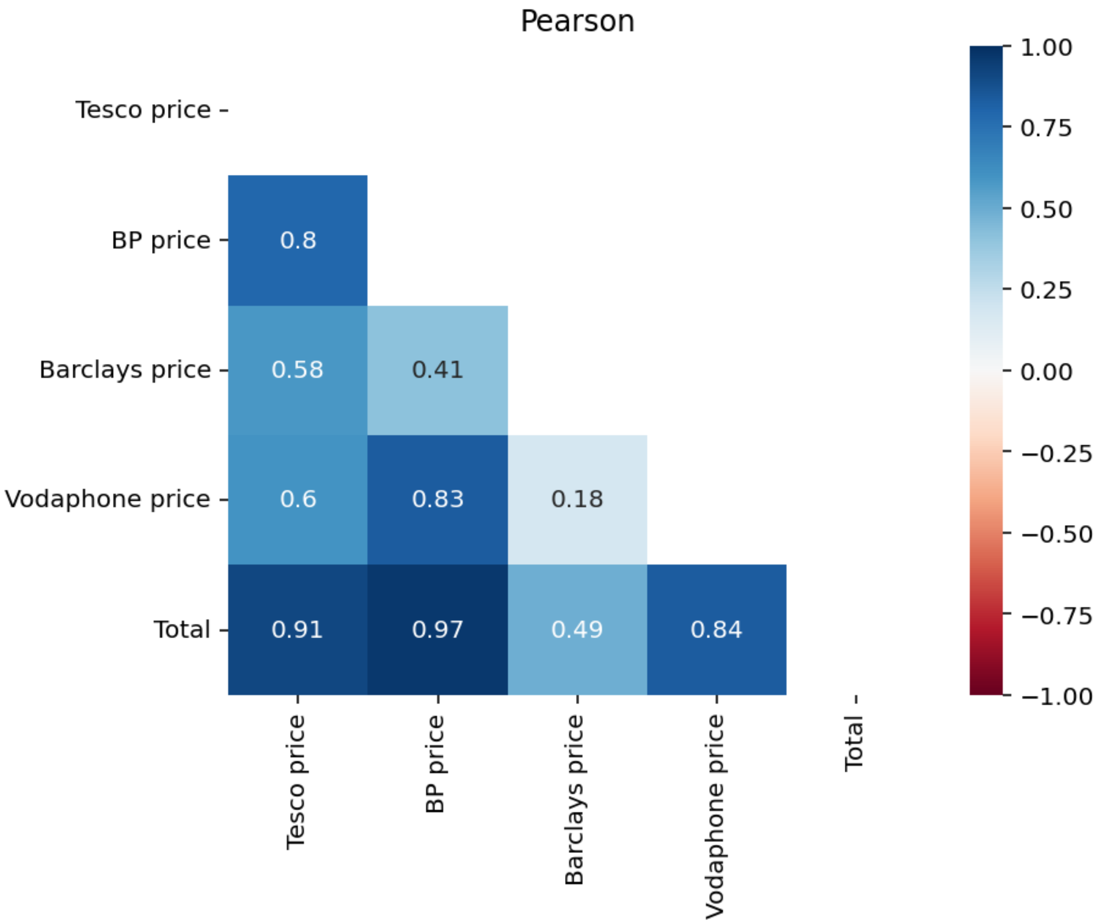
[ ]:
```
plot_returns_box(df_returns)
plot_returns_violin(df_returns)
```

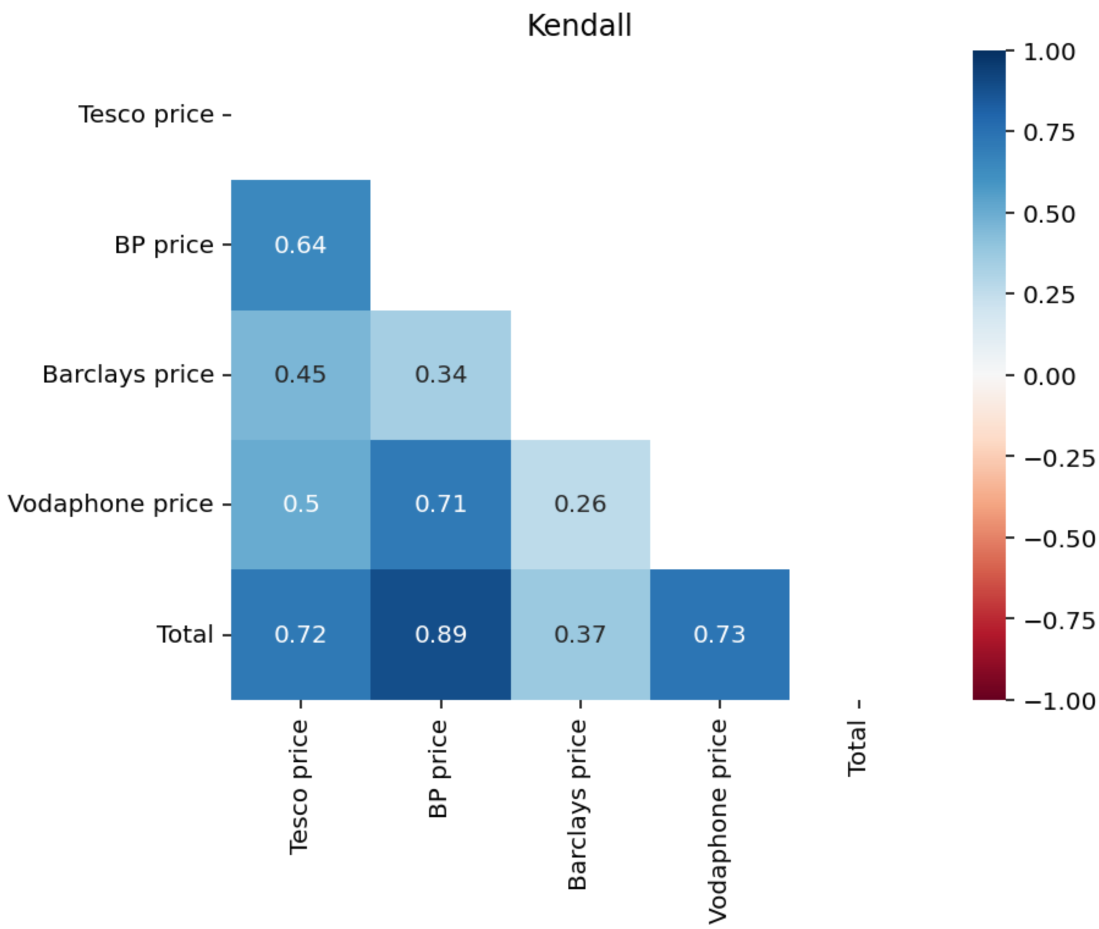
3.1 End Question 2

What about our correlations, how can we visualise those?

[177]:
```
def plot_price_correlation(df_prices, method):
    """
    Plots correlation of prices with different methods
    """
    fig, ax = plt.subplots(dpi=144)

    mask = np.triu(np.ones_like(df_prices.corr()))
    sns.heatmap(df_prices.corr(method=method), ax=ax, vmin=-1, vmax=1,␣
 ↪cmap='RdBu', annot=True, mask=mask)
    plt.title(method.capitalize())
    plt.show()
    return
```

[178]:
```
plot_price_correlation(df_prices, 'pearson')
plot_price_correlation(df_prices, 'kendall')
```


Pearson
Tesco price
BP price
Barclays price
Vodaphone price
Total
0.8
0.58
0.41
0.6
0.83
0.18
0.91
0.97
0.49
0.84
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00



4 Question 3

Create both types of correlations as heatmaps for the annual returns financial data (including a total), grouped by decade, between 1990 and 2019.

[ ]:
```
def plot_returns_correlation(dfgrp_returns, method):
    """
    Plots correlation of returns with different methods
    """
    return
```

[ ]:
```
df_returns_cut =
dfgrp_returns_decade =

for decade, dfgrp_returns in dfgrp_returns_decade:
    plot_returns_correlation(dfgrp_returns, 'pearson')
```

```
plot_returns_correlation(dfgrp_returns, 'kendall')
```

4.1 End Question 3