Introduction to Python
Financial Time Series

Contents

4 Financial Time Series

- Time series are ubiquitous in finance.
- `pandas` is the main library in Python to deal with time series.

4.1 Financial Data

Financial data

- For the time being we work with locally stored data files.
- These are in `.csv`-files (comma-separated values), where the data entries in each row are separated by commas.
- Some initialisation:

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('seaborn')
plt.rcParams['font.family'] = 'serif'
```

Data import

- `pandas` provides a numer of different functions and `DataFrame` methods for importing and exporting data.
- Here we use `pd.read_csv()`.
- The file that we load contains end-of-day data for different financial instruments retrieved from Thomson Reuters.

[2]:
```
# If using colab, then uncomment the line below and comment the line after that
#filename = 'https://raw.githubusercontent.com/packham/Python_CFDS/main/data/
↪tr_eikon_eod_data.csv'
filename = './data/tr_eikon_eod_data.csv' # path and filename
f = open(filename, 'r') # this will give an error when using colab; just ignore it
f.readlines()[:5]  # show first five lines
```

[2]:
```
['Date,AAPL.O,MSFT.O,INTC.O,AMZN.O,GS.N,SPY,.SPX,.VIX,EUR=,XAU=,GDX,GLD\n',
 '2010-01-01,,,,,,,,,1.4323,1096.35,,\n',
 '2010-01-04,30.57282657,30.95,20.88,133.9,173.08,113.33,1132.99,20.04,1.4411,11
20.0,47.71,109.8\n',
 '2010-01-05,30.625683660000004,30.96,20.87,134.69,176.14,113.63,1136.52,19.35,1
.4368,1118.65,48.17,109.7\n',
 '2010-01-06,30.138541290000003,30.77,20.8,132.25,174.26,113.71,1137.14,19.16,1.
```

```
4412,1138.5,49.34,111.51\n']
```

Data import

[3]:
```python
data = pd.read_csv(filename,  # import csv-data into DataFrame
                   index_col=0, # take first column as index
                   parse_dates=True)  # index values are datetime
```

[4]:
```python
data.info()  # information about the DataFrame object
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2216 entries, 2010-01-01 to 2018-06-29
Data columns (total 12 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   AAPL.O  2138 non-null   float64
 1   MSFT.O  2138 non-null   float64
 2   INTC.O  2138 non-null   float64
 3   AMZN.O  2138 non-null   float64
 4   GS.N    2138 non-null   float64
 5   SPY     2138 non-null   float64
 6   .SPX    2138 non-null   float64
 7   .VIX    2138 non-null   float64
 8   EUR=    2216 non-null   float64
 9   XAU=    2211 non-null   float64
 10  GDX     2138 non-null   float64
 11  GLD     2138 non-null   float64
dtypes: float64(12)
memory usage: 225.1 KB
```

Data import

[5]:
```python
data.head()
```

[5]:
```
               AAPL.O  MSFT.O  INTC.O  AMZN.O    GS.N     SPY     .SPX   .VIX  \
Date
2010-01-01        NaN     NaN     NaN     NaN     NaN     NaN      NaN    NaN
2010-01-04  30.572827  30.950   20.88  133.90  173.08  113.33  1132.99  20.04
2010-01-05  30.625684  30.960   20.87  134.69  176.14  113.63  1136.52  19.35
2010-01-06  30.138541  30.770   20.80  132.25  174.26  113.71  1137.14  19.16
2010-01-07  30.082827  30.452   20.60  130.00  177.67  114.19  1141.69  19.06

              EUR=     XAU=    GDX     GLD
Date
2010-01-01  1.4323  1096.35    NaN     NaN
2010-01-04  1.4411  1120.00  47.71  109.80
2010-01-05  1.4368  1118.65  48.17  109.70
2010-01-06  1.4412  1138.50  49.34  111.51
2010-01-07  1.4318  1131.90  49.10  110.82
```

Data import

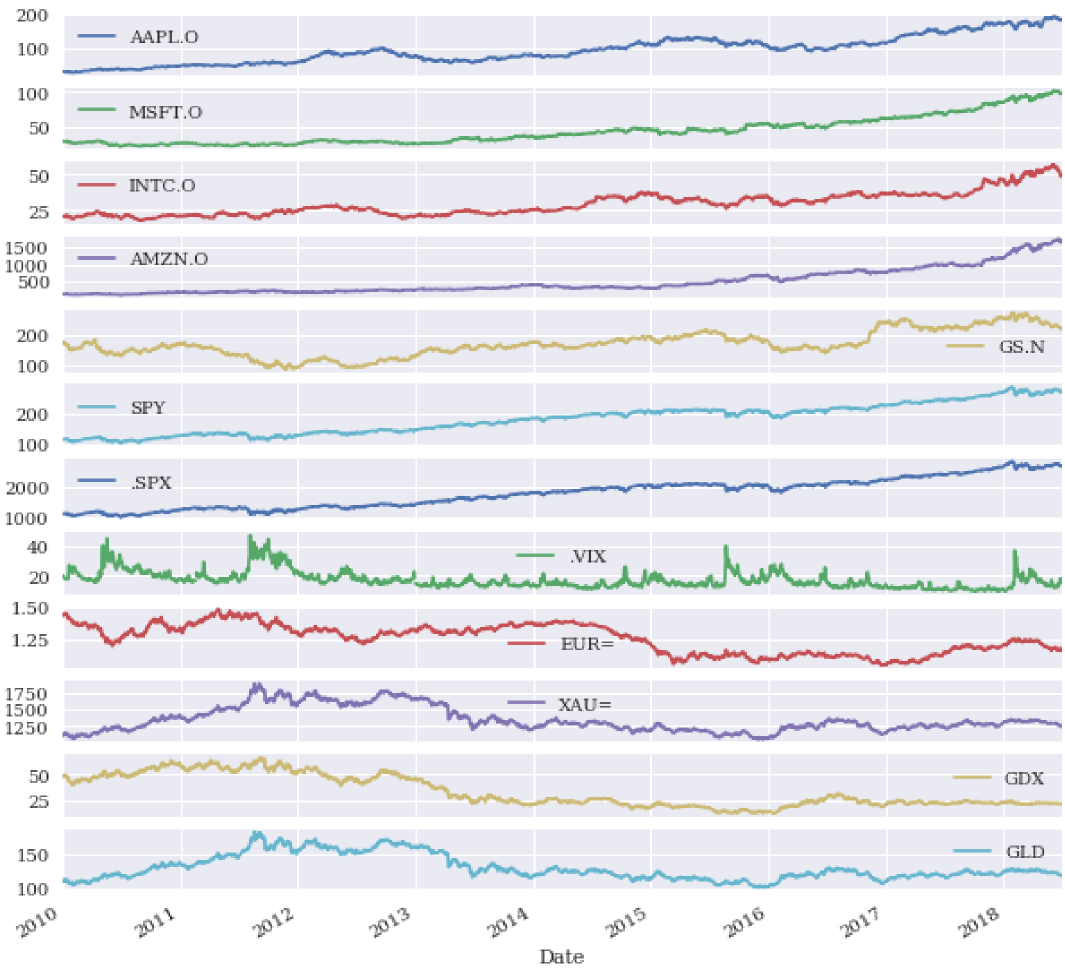
[6]:
```python
data.tail()
```

[6]:
```
            AAPL.O  MSFT.O  INTC.O  AMZN.O    GS.N     SPY     .SPX   .VIX  \
Date
```

```
2018-06-25  182.17   98.39   50.71  1663.15  221.54  271.00  2717.07  17.33
2018-06-26  184.43   99.08   49.67  1691.09  221.58  271.60  2723.06  15.92
2018-06-27  184.16   97.54   48.76  1660.51  220.18  269.35  2699.63  17.91
2018-06-28  185.50   98.63   49.25  1701.45  223.42  270.89  2716.31  16.85
2018-06-29  185.11   98.61   49.71  1699.80  220.57  271.28  2718.37  16.09

              EUR=     XAU=    GDX     GLD
Date
2018-06-25  1.1702  1265.00  22.01  119.89
2018-06-26  1.1645  1258.64  21.95  119.26
2018-06-27  1.1552  1251.62  21.81  118.58
2018-06-28  1.1567  1247.88  21.93  118.22
2018-06-29  1.1683  1252.25  22.31  118.65
```

Data import

[7]:
```
data.plot(figsize=(10, 10), subplots=True);
```



Data import

- The identifiers used by Thomson Reuters are so-called RIC's.
- The financial instruments in the data set are:

[8]:
```
instruments = ['Apple Stock', 'Microsoft Stock',
               'Intel Stock', 'Amazon Stock', 'Goldman Sachs Stock',
               'SPDR S&P 500 ETF Trust', 'S&P 500 Index',
               'VIX Volatility Index', 'EUR/USD Exchange Rate',
               'Gold Price', 'VanEck Vectors Gold Miners ETF',
               'SPDR Gold Trust']
```

Data import

[9]:
```
for ric, name in zip(data.columns, instruments):
    print('{:8s} | {}'.format(ric, name))
```

```
AAPL.O   | Apple Stock
MSFT.O   | Microsoft Stock
INTC.O   | Intel Stock
AMZN.O   | Amazon Stock
GS.N     | Goldman Sachs Stock
SPY      | SPDR S&P 500 ETF Trust
.SPX     | S&P 500 Index
.VIX     | VIX Volatility Index
EUR=     | EUR/USD Exchange Rate
XAU=     | Gold Price
GDX      | VanEck Vectors Gold Miners ETF
GLD      | SPDR Gold Trust
```

Summary statistics

[10]:
```
data.describe().round(2)
```

[10]:

	AAPL.O	MSFT.O	INTC.O	AMZN.O	GS.N	SPY	.SPX	.VIX	\
count	2138.00	2138.00	2138.00	2138.00	2138.00	2138.00	2138.00	2138.00	
mean	93.46	44.56	29.36	480.46	170.22	180.32	1802.71	17.03	
std	40.55	19.53	8.17	372.31	42.48	48.19	483.34	5.88	
min	27.44	23.01	17.66	108.61	87.70	102.20	1022.58	9.14	
25%	60.29	28.57	22.51	213.60	146.61	133.99	1338.57	13.07	
50%	90.55	39.66	27.33	322.06	164.43	186.32	1863.08	15.58	
75%	117.24	54.37	34.71	698.85	192.13	210.99	2108.94	19.07	
max	193.98	102.49	57.08	1750.08	273.38	286.58	2872.87	48.00	

	EUR=	XAU=	GDX	GLD
count	2216.00	2211.00	2138.00	2138.00
mean	1.25	1349.01	33.57	130.09
std	0.11	188.75	15.17	18.78
min	1.04	1051.36	12.47	100.50
25%	1.13	1221.53	22.14	117.40
50%	1.27	1292.61	25.62	124.00
75%	1.35	1428.24	48.34	139.00
max	1.48	1898.99	66.63	184.59

Summary statistics

- The aggregate()-function allows to customise the statistics viewed:

[11]:
```
data.aggregate([min,
                np.mean,
                np.std,
```

```
                np.median,
                max]
).round(2)
```

[11]:

```
         AAPL.O  MSFT.O  INTC.O   AMZN.O    GS.N     SPY     .SPX   .VIX  EUR=  \
min       27.44   23.01   17.66   108.61   87.70  102.20  1022.58   9.14  1.04
mean      93.46   44.56   29.36   480.46  170.22  180.32  1802.71  17.03  1.25
std       40.55   19.53    8.17   372.31   42.48   48.19   483.34   5.88  0.11
median    90.55   39.66   27.33   322.06  164.43  186.32  1863.08  15.58  1.27
max      193.98  102.49   57.08  1750.08  273.38  286.58  2872.87  48.00  1.48

           XAU=    GDX     GLD
min     1051.36  12.47  100.50
mean    1349.01  33.57  130.09
std      188.75  15.17   18.78
median  1292.61  25.62  124.00
max     1898.99  66.63  184.59
```

Returns

- When working with financial data we typically (=always - you must have good reasons to deviate from this) work with performance data, i.e., **returns**.
- Reasoning:
  - Historical data are mainly used to make forecasts one or several time periods forward.
  - The daily average stock price over the last eight years is meaningless to make a forecast for tomorrow's stock price.
  - However, the daily returns are possible scenarios for the next time period(s).
- The function `pct_change()` calculates discrete returns:

$$r_t^{\mathrm{d}} = \frac{S_t - S_{t-1}}{S_{t-1}},$$

  where S_t denotes the stock price at time t.

Returns

[12]:

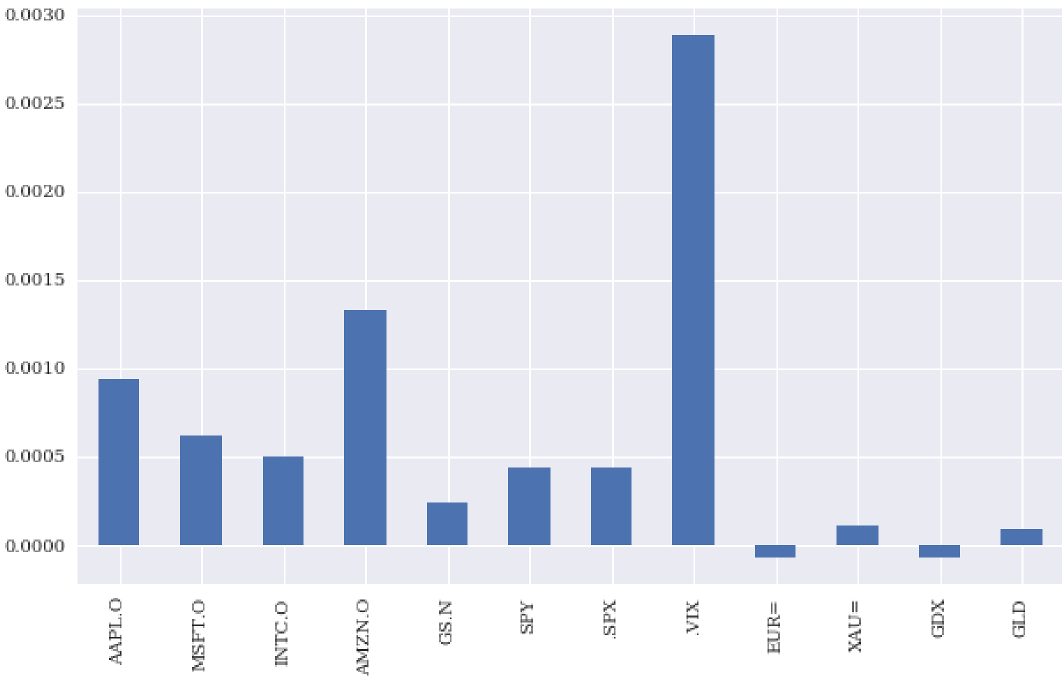
```
data.pct_change().round(3).head()
```

[12]:

```
            AAPL.O  MSFT.O  INTC.O  AMZN.O   GS.N    SPY   .SPX   .VIX   EUR=  \
Date
2010-01-01     NaN     NaN     NaN     NaN    NaN    NaN    NaN    NaN    NaN
2010-01-04     NaN     NaN     NaN     NaN    NaN    NaN    NaN    NaN  0.006
2010-01-05   0.002   0.000  -0.000   0.006  0.018  0.003  0.003 -0.034 -0.003
2010-01-06  -0.016  -0.006  -0.003  -0.018 -0.011  0.001  0.001 -0.010  0.003
2010-01-07  -0.002  -0.010  -0.010  -0.017  0.020  0.004  0.004 -0.005 -0.007

             XAU=    GDX    GLD
Date
2010-01-01    NaN    NaN    NaN
2010-01-04  0.022    NaN    NaN
2010-01-05 -0.001  0.010 -0.001
2010-01-06  0.018  0.024  0.016
2010-01-07 -0.006 -0.005 -0.006
```

Returns

[13]:
```
data.pct_change().mean().plot(kind='bar', figsize=(10, 6));
```



Returns

- In finance, **log-returns**, also called **continuous returns**, are often preferred over discrete returns: $r_t^c = \ln\left(\frac{S_t}{S_{t-1}}\right)$.
- The main reason is that log-return are additive over time.
- For example, the log-return from $t-1$ to $t+1$ is the sum of the single-period log-returns:

$$r_{t-1,t+1}^c = \ln\left(\frac{S_{t+1}}{S_t}\right) + \ln\left(\frac{S_t}{S_{t-1}}\right) = \ln\left(\frac{S_{t+1}}{S_t} \cdot \frac{S_t}{S_{t-1}}\right) = \ln\left(\frac{S_{t+1}}{S_{t-1}}\right).$$

- Note: If the sampling (time) interval is small (e.g. one day or one week), then the difference between discrete returns and log-returns is negligible.

Returns

[14]:
```
rets = np.log(data / data.shift(1))  # calculates log-returns in a vectorised way
```

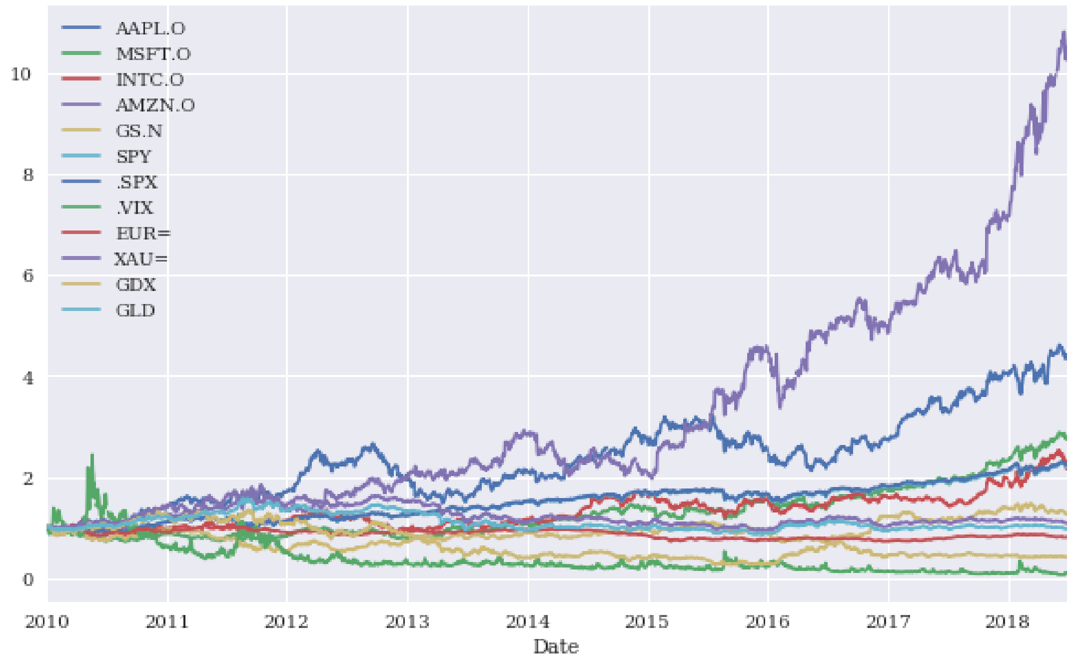
[15]:
```
rets.head().round(3)
```

[15]:
```
            AAPL.O  MSFT.O  INTC.O  AMZN.O   GS.N    SPY   .SPX   .VIX   EUR=  \
Date
2010-01-01     NaN     NaN     NaN     NaN    NaN    NaN    NaN    NaN    NaN
2010-01-04     NaN     NaN     NaN     NaN    NaN    NaN    NaN    NaN  0.006
2010-01-05   0.002   0.000  -0.000   0.006  0.018  0.003  0.003 -0.035 -0.003
2010-01-06  -0.016  -0.006  -0.003  -0.018 -0.011  0.001  0.001 -0.010  0.003
2010-01-07  -0.002  -0.010  -0.010  -0.017  0.019  0.004  0.004 -0.005 -0.007

             XAU=    GDX    GLD
Date
```

```
2010-01-01     NaN     NaN     NaN
2010-01-04   0.021     NaN     NaN
2010-01-05  -0.001   0.010  -0.001
2010-01-06   0.018   0.024   0.016
2010-01-07  -0.006  -0.005  -0.006
```

Returns

[16]:
```
rets.cumsum().apply(np.exp).plot(figsize=(10, 6));  # recover price paths from␣
↪log-returns
```



4.2 Correlation analysis and linear regression

- To further illustrate how to work with financial time series we consider the S&P 500 stock index and the VIX volatility index.
- Empirical stylised fact: As the S&P 500 rises, the VIX falls, and vice versa.
- Note: This is about **correlation** not **causation**.

Correlation analysis

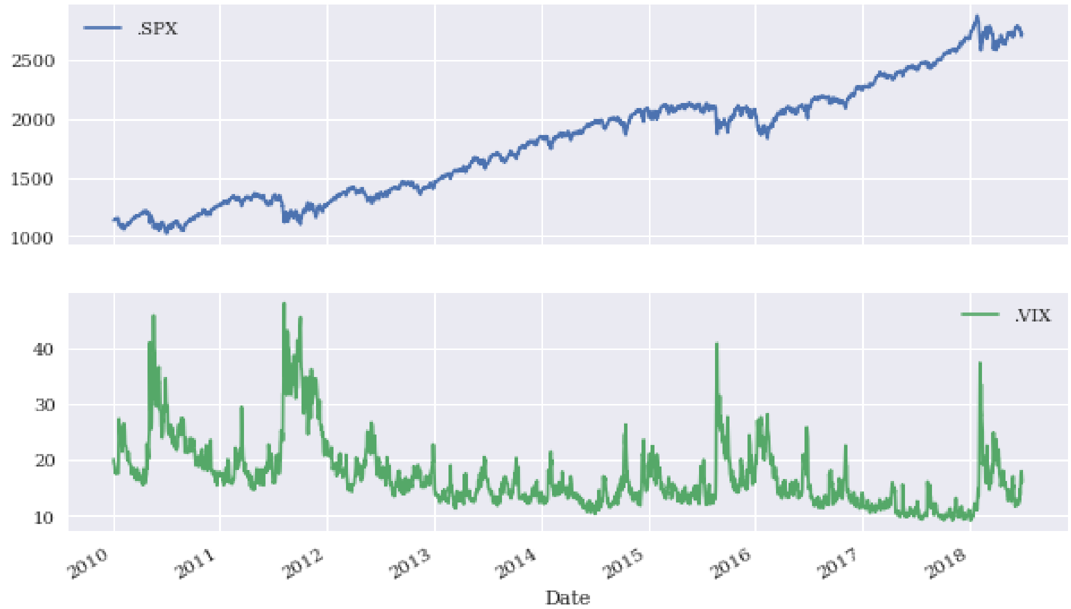
[17]:
```
# EOD data from Thomson Reuters Eikon Data API

# If using colab, then uncomment the line below and comment the line after that
#raw = pd.read_csv('https://raw.githubusercontent.com/packham/Python_CFDS/main/data/
↪tr_eikon_eod_data.csv', index_col=0, parse_dates=True)
raw = pd.read_csv('./data/tr_eikon_eod_data.csv', index_col=0, parse_dates=True)
data = raw[['.SPX', '.VIX']].dropna()
data.tail()
```

[17]:
```
               .SPX   .VIX
Date
2018-06-25  2717.07  17.33
```

```
2018-06-26  2723.06  15.92
2018-06-27  2699.63  17.91
2018-06-28  2716.31  16.85
2018-06-29  2718.37  16.09
```

Correlation analysis

[18]:
```
data.plot(subplots=True, figsize=(10, 6));
```



Correlation analysis

- Transform both data series into log-returns:

[19]:
```
rets = np.log(data / data.shift(1))
rets.head()
```

[19]:
```
                .SPX       .VIX
Date
2010-01-04       NaN        NaN
2010-01-05  0.003111 -0.035038
2010-01-06  0.000545 -0.009868
2010-01-07  0.003993 -0.005233
2010-01-08  0.002878 -0.050024
```

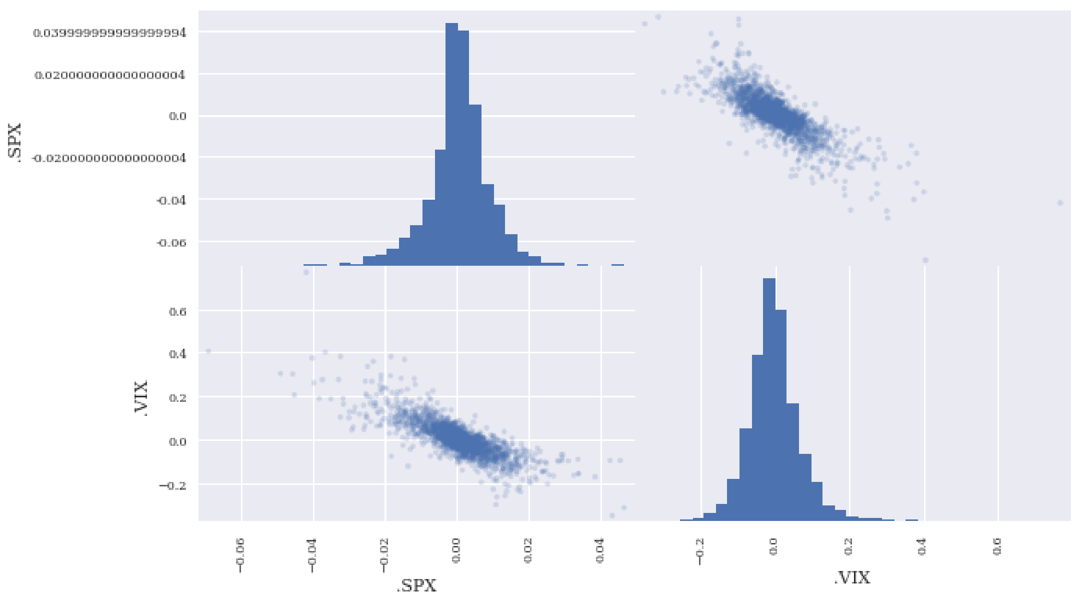
[20]:
```
rets.dropna(inplace=True) # drop NaN (not-a-number) entries
```

Correlation analysis

[21]:
```
rets.plot(subplots=True, figsize=(10, 6));
```

Correlation analysis

[22]:
```
pd.plotting.scatter_matrix(rets,
                           alpha=0.2,
                           diagonal='hist',
                           hist_kwds={'bins': 35},
                           figsize=(10, 6));
```



Correlation analysis

[23]:
```
rets.corr()
```

[23]:
```
          .SPX      .VIX
.SPX  1.000000 -0.804382
.VIX -0.804382  1.000000
```

OLS regression

- **Linear regression** captures the linear relationship between two variables.
- For two variables x, y, we postulate a linear relationship:

$$y = \alpha + \beta x + \varepsilon, \quad \alpha, \beta \in \mathbb{R}.$$

- Here, α is the **intercept**, β is the **slope (coefficient)** and ε is the **error term**.
- Given data sample of joint observations $(x_1, y_1), \ldots, (x_n, y_n)$, we set

$$y_i = \hat{\alpha} + \hat{\beta} x_i + \hat{\varepsilon}_i,$$

  where $\hat{\alpha}$ and $\hat{\beta}$ are estimates of α, β and $\hat{\varepsilon}_1, \ldots, \hat{\varepsilon}_n$ are the so-called **residuals**.
- The **ordinary least squares (OLS)** estimator $\hat{\alpha}, \hat{\beta}$ corresponds to those values of α, β that minimise the sum of squared residuals:

$$\min_{\alpha,\beta} \sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n} (y_i - \alpha - \beta x_i)^2.$$

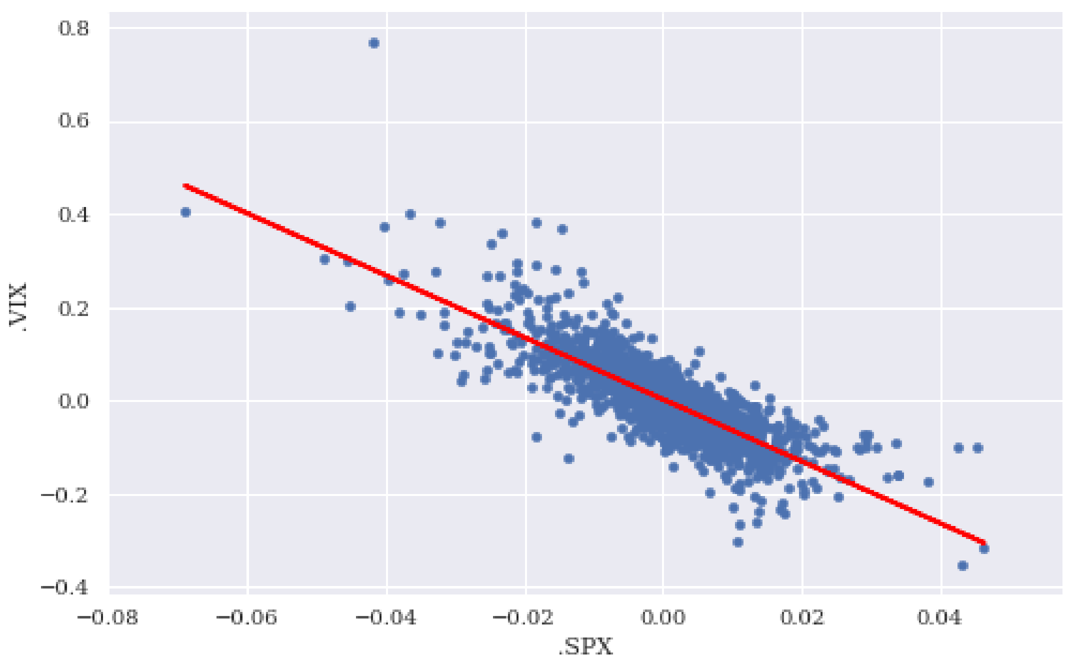
OLS regressions

- Simplest form of OLS regression:

[24]:
```
reg = np.polyfit(rets['.SPX'], rets['.VIX'], deg=1)  # fit a linear equation (a␣
↪polynomial of degree 1)
reg.view() # the fitted paramters
```

[24]:
```
array([-6.65160028e+00,  2.62132142e-03])
```

[25]:
```
ax = rets.plot(kind='scatter', x='.SPX', y='.VIX', figsize=(8, 5))
ax.plot(rets['.SPX'], np.polyval(reg, rets['.SPX']), 'r', lw=2);
```



OLS regression

- To do a more refined OLS regression with a proper analysis, use the package `statsmodels`.

[26]:
```
import statsmodels.api as sm

Y=rets['.VIX']
X=rets['.SPX']
X = sm.add_constant(X)
```

[27]:
```
model = sm.OLS(Y,X)
results = model.fit()
```

[28]:
```
results.params
```

[28]:
```
const     0.002621
.SPX     -6.651600
dtype: float64
```

[29]:
```
results.predict()[0:10]
```

[29]:
```
array([-0.01807052, -0.0010063 , -0.0239404 , -0.01651898, -0.00898726,
        0.06531557, -0.05252965, -0.01349928,  0.07500527, -0.08000615])
```

OLS regression

[30]:
```
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   .VIX   R-squared:                       0.647
Model:                            OLS   Adj. R-squared:                  0.647
Method:                 Least Squares   F-statistic:                     3914.
Date:                Tue, 06 Dec 2022   Prob (F-statistic):               0.00
Time:                        14:44:29   Log-Likelihood:                 3550.1
No. Observations:                2137   AIC:                            -7096.
Df Residuals:                    2135   BIC:                            -7085.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0026      0.001      2.633      0.009       0.001       0.005
.SPX          -6.6516      0.106    -62.559      0.000      -6.860      -6.443
==============================================================================
Omnibus:                      518.582   Durbin-Watson:                   2.094
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             6789.425
Skew:                           0.766   Prob(JB):                         0.00
Kurtosis:                      11.597   Cond. No.                         107.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

OLS regression: Interpretation of output and forecasting

- The column `coef` lists the coefficients of the regression: the coefficient in the row labelled `const` corresponds to $\hat{\alpha}$ $(= 0.0026)$ and the coefficient in the row `.SPX` denotes $\hat{\beta}$ $(= -6.6515)$.
- The estimated model in the example is thus:

$$.\text{VIX} = 0.0026 - 6.6516.\text{SPX}.$$

- The best forecast of the VIX return when observing an S&P return of 2% is therefore $0.0026 - 6.6516 \cdot 0.02 = -0.130432 = -13.0432\%$.

OLS regression: Validation (R^2)

- To **validate** the model, i.e., to determine, if the model in itself and the explanatory variable(s) make sense, we look R^2 and various p-values (or confidence intervals or t-statistics).
- R^2 measures the fraction of variance in the dependent variable Y that is captured by the regression line; $1 - R^2$ is the fraction of Y-variance that remaines in the residuals ε_i^2, $i = 1, \ldots, n$.
- In the output above R^2 is given as 0.647. In other words, 64.7% of the variance in VIX returns are "explained" by SPX returns.
- A high R^2 (and this one is high) is necessary for making forecasts.

OLS regression: Validation (confidence interval)

- An important hypothesis to test in any regression model is whether the explanatory variable(s) have an effect on the independent variable.
- This can be translated into testing whether $\beta \neq 0$. ($\beta = 0$ is the same as saying that the X variable can be removed from the model.)
- Formally, we test the null hypothesis $H_0 : \beta = 0$ against the alternative hypothesis $H_1 : \beta \neq 0$.
- There are several statistics to come to the same conclusion: confidence intervals, t-statistics and p-values.
- The **confidence interval** is an interval around the estimate $\hat{\beta}$ that we are confident contains the true parameter β. A typial **confidence level** is 95%.
- If the 95% confidence interval does **not** contain 0, then we say β is **statistically significant** at the 5% (=1-95%) level, and we conclude that $\beta \neq 0$.

OLS regression: Validation (t-statistic)

- The t-statistic corresponds to the **number of standard deviations** that the estimated coefficient $\hat{\beta}$ is away from 0 (the mean under H_0).
- For a normal distribution, we have the following rules of thumb:
  - 66% of observations lie within one standard deviation of the mean
  - 95% of observations lie within two standard deviations of the mean
  - 99.7% of observations lie within three standard deviations of the mean

- If the sample size is large enough, then the t-statistic is approximately normally distributed, and if it is large (in absolute terms), then this is an indication against $\beta = 0$.
- In the example above, the t-statistics is -62.559, i.e., $\hat{\beta}$ is approx. 63 standard deviations away from zero, which is practically impossible.

OLS regression: Validation (p-value)

- The p-value expresses the probability of observing a coefficient estimate as extreme (away from zero) as $\hat{\beta}$ under H_0, i.e., when $\beta = 0$.

- In other words, it measures the probability of observing a t-statistic as extreme as the one observed if $\beta = 0$.

- If the p-value (column `P>|t|`) is smaller than the desired level of significance (typically 5%), then the H_0 can be rejected and we conclude that $\beta \neq 0$.

- In the example above, the p-value is given as 0.000, i.e., it is so small, that we can conclude the estimated coefficient $\hat{\beta}$ is so extreme (= away from zero) that is virtually impossible to obtain such an estimated if $\beta = 0$.
- Finally, the F-test tests the hypotheses $H_0 : R^2 = 0$ versus $H_1 : R^2 \neq 0$. In a multiple regression with k independent variables, this is equivalent to $H_0 : \beta_1 = \cdots = \beta_k = 0$.
- In the example above, the p-value of the F-test is 0, so we conclude that the model overall has explanatory power.